TUGAS AKHIR - KM184801

PENGEMBANGAN MODUL REUSE PADA SISTEM PENATAAN MUATAN KAPAL PETI KEMAS BERBASIS CBR UNTUK RUTE MULTIPORT

NUR LAILI ASTI PRAMESTI
06111640000058

Dosen Pembimbing
Dr. Imam Mukhlash, S.Si., M.T.
Dr.-Ing Setyo Nugroho

Departemen Matematika
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR – KM184801

PENGEMBANGAN MODUL *REUSE* PADA SISTEM PENATAAN MUATAN KAPAL PETI KEMAS BERBASIS *CBR* UNTUK RUTE *MULTIPORT*

NUR LAILI ASTI PRAMESTI
NRP 061116 4000 058

Dosen Pembimbing :
Dr. Imam Mukhlash, S.Si., M.T.
Dr.-Ing. Setyo Nugroho

DEPARTEMEN MATEMATIKA
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember
Surabaya 2020

FINAL PROJECT – KM184801

DEVELOPMENT REUSE MODULE ON STOWAGE PLANNING SYSTEM USING CBR FOR MULTIPORT ROUTE

NUR LAILI ASTI PRAMESTI
NRP 061116 4000 058

Supervisors :
Dr.Imam Mukhlash, S.Si., M.T.
Dr.-Ing Setyo Nugroho

DEPARTEMENT OF MATHEMATICS
Faculty of Sains and Analitical Data
Institut Teknologi Sepuluh Nopember
Surabaya 2020

LEMBAR PENGESAHAN

PENGEMBANGAN MODUL *REUSE* PADA SISTEM PENATAAN MUATAN KAPAL PETI KEMAS BERBASIS *CBR* UNTUK RUTE *MULTIPORT*

DEVELOPMENT REUSE MODULE ON STOWAGE PLANNING SYSTEM USING CBR FOR MULTIPORT ROUTE

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
Untuk memperoleh gelar Sarjana Matematika
Pada bidang studi Ilmu Komputer
Program Studi S-1 Departemen Matematika
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember Surabaya

Oleh:
Nur Laili Asti Pramesti
NRP. 06111640000058

Menyetujui,

Dosen Pembimbing II

Dr.-Ing. Setyo Nugroho
NIP. 19651020 199601 1 001

Dosen Pembimbing I

Dr. Imam Mukhlash, S.Si., MT.
NIP. 19700831 199403 1 003

Mengetahui,
Kepala Departemen Matematika
FSAD ITS

Subchan, Ph.D
NIP. 19710513 199702 1 001

Surabaya, Agustus 2020

PENGEMBANGAN MODUL *REUSE* PADA SISTEM PENATAAN MUATAN KAPAL PETI KEMAS BERBASIS *CBR* UNTUK RUTE *MULTIPORT*

Nama : **Nur Laili Asti Pramesti**
NRP : **061116 4000 0058**
Departemen : **Matematika**
Dosen Pembimbing : **1. Dr. Imam Mukhlash, S.Si., M.T.**
2. Dr.-Ing. Setyo Nugroho

ABSTRAK

Stowage planning merupakan kegiatan penataan peti kemas pada kapal kontainer. Penataan peti kemas yang baik dan cepat memerlukan suatu perangkat lunak yang dapat membantu kinerja *planner* terutama untuk kasus *multiport.* Metode *Case Based Reasoning (CBR)* dapat digunakan dalam mengimplementasi perangkat lunak supaya menghasilkan penataan peti kemas berdasarkan penataan yang sudah ada sebelumnya. Metode *CBR* yang dibahas dalam tugas akhir ini difokuskan pada tahap *reuse* dengan merancang Algoritma *Similarity,* Algoritma Adaptasi *Container Loading List* serta Algoritma Evaluasi *Restow.* Rangkaian algoritma tersebut digunakan untuk menghasilkan penataan peti kemas di setiap pelabuhan dan menampilkan jumlah restow dalam satu voyage. Hasil dari penelitian ini juga dapat dilakukan analisa untuk beban kerja *planner* dimana semakin kecil populasi *casebase*, *swap manual* yang dilakukan rata-rata lebih dari 70% dari total peti kemas pada kapal.

Kata Kunci: *Case-Based Reasoning, Container, Stowage Planning, Container Ships, Multiport, Restow, Stowage Planning Satisfy*

DEVELOPMENT REUSE MODULE ON STOWAGE PLANNING SYSTEM CBR FORM MULTIPORT ROUTE

Name : **Nur Laili Asti Pramesti**
NRP : **061116 4000 0058**
Department : **Mathematics**
Supervisors : **1. Dr.Imam Mukhlash, S.Si., M.T.**
2. Dr.-Ing Setyo Nugroho

ABSTRACT

Stowage planning is an activity of containers arrangement on container vessels. A good and fast container setup requires a piece of software that can help planner's performance especially for multiport cases. The Case Based Reasoning method (CBR) can be used to implement software in order to produce a container setup based on pre-existing setups. The Case Based Reasoning method discussed in this final task is focused on the reuse phase by designing the Similarity algorithm, Container Loading List adaptation algorithms as well as the Restow evaluation algorithm. The set of algorithms is used to automatically produce a container setup at each port in one voyage and display the number of Restow and many containers that experience loading and unloading in one voyage so as to assist the performance of the planner in every port. The results of this study can also be analyzed for the workload of planners where the smaller the casebase population, manual swaps performed on average more than 70% of the total containers on the ship.

***Keywords**: Case-Based Reasoning, Container, Stowage Planning, Container Ships, Multiport, Restow, Stowage Planning Satisfy*

KATA PENGANTAR

Segala puji syukur penulis panjatkan kehadirat Allah SWT, karena dengan ridho-Nya penulis dapat menyelesaikan Penelitian Tugas Akhir yang berjudul

PENGEMBANGAN MODUL *REUSE* PADA SISTEM PENATAAN MUATAN KAPAL PETI KEMAS BERBASIS *CBR* UNTUK RUTE *MULTIPORT*

merupakan salah satu persyaratan akademis dalam menyelesaikan Program Sarjana Departemen Matematika, Fakultas Sains dan Analitika Data, Institut Teknologi Sepuluh Nopember Surabaya.

Tugas Akhir ini dapat diselesaikan dengan baik berkat kerja sama, bantuan, dan dukungan dari banyak pihak. Sehubungan dengan hal tersebut, penulis ingin mengucapkan terimakasih kepada:

1. Kedua orang tua, kakak, adik dan saudara penulis yang senantiasa memberikan dukungan dan doa.
2. Bapak Robert Gunawan, selaku pemilik PT Mahaghora yang telah memberikan saya beasiswa selama berkuliah.
3. Bapak Dr. Imam Mukhlash, S.Si, MT. selaku Kepala Departemen Matematika ITS sekaligus Dosen Pembimbing yang telah memberikan bimbingan dan motivasi.
4. Bapak Dr.-Ing. Setyo Nugroho selaku Dosen Pembimbing yang telah memberikan bimbingan dan motivasi.

5. Bapak Subchan, Ph.D selaku Ketua Program Studi S1 Departemen Matematika ITS
6. Ibu Dr. Dwi Ratna Sulistyaningrum, MT selaku Sekretaris Departemen Matematika ITS.
6. Ibu Dr. Valeriana Lukitosari, S.Si, MT selaku dosen wali yang telah memberika dukungan pengarahan selama perkuliahan.
7. Seluruh jajaran dosen dan staf di Departemen Matematika ITS.
8. Team *iStow* yang telah memberikan bantuan dan sebagai teman diskusi dalam menyelesaikan Tugas Akhir ini.
9. Keluarga Beasiswa Mahaghora, Ko Licco, Kak Gabby, Mbak Silvy, Salsa, Juan, Okto, Eggi, Kak Dimas, Harum, Angel, Joshua, Ferdian, Elisa yang telah memberikan semangat selama pengerjaan.
10. Sahabat dan kerabat dekat, Thalia, Septi, Edwina, Soma, Safira N.L, Jurisa, Ratri dan Dhafa yang selalu memberi semangat dalam pengerjaan,
11. Lemniscate (STI-51) dan teman-teman penulis yang telah memberikan semangat.
12. Semua pihak yang belum disebutkan yang telah membantu dalam penyusunan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih jauh dari kesempurnaan. Oleh karena itu, penulis mengharapkan kritik dan saran dari pembaca. Akhir kata, semoga Tugas Akhir ini dapat bermanfaat bagi semua pihak yang berkepentingan.

Surabaya, Agustus 2020

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

Pada bab ini dijelaskan hal-hal yang melatarbelakangi permasalahan yang dibahas dalam tugas akhir ini, rumusan masalah, batasan masalah, tujuan penelitian dan manfaat penelitian.

1.1 Latar Belakang Masalah

Indonesia adalah negara kepulauan dimana 2/3 wilayah Indonesia merupakan lautan. Didukung pula dengan berkembangnya pengolahan industri saat ini di berbagai wilayah guna meningkatkan strategi ekonomi regional, menjadikan potensi permintaan jasa pengiriman dengan menggunakan transportasi laut dalam bentuk kontainer banyak diminati oleh pengusaha industri. Saat ini, lebih dari 70% perusahaan industri memilih untuk melakukan pengiriman barang dengan menggunakan transportasi laut dalam jumlah kontainer yang besar [1]. Hal ini membuat pelabuhan menjadi sangat penting dalam mempengaruhi kuantitas dan ukuran kapal yang dapat bersandar di dermaga. Pelabuhan harus menangani kapal yang lebih besar dan membawa jumlah peti kemas yang lebih banyak dalam waktu singkat [2]. Setiap pelabuhan yang masuk dalam daftar perjalanan kapal akan melayani proses penurunan peti kemas atau penambahan peti kemas untuk dikirim ke pelabuhan selanjutnya. *Stowage planning* untuk kapal kontainer yang meninggalkan pelabuhan masih dilakukan secara manual oleh manusia atau biasa disebut *stowage planner*. *Stowage planner* bekerja dibawah kendala waktu yang singkat dan jumlah

konfigurasi kontainer yang terbatas sebagai pertimbangan [3].

Sebelum kedatangan kapal di pelabuhan, bagian perencanaan akan menerima daftar bongkar muat peti kemas dan kondisi kapal yang akan datang di pelabuhan tersebut. Namun, dalam banyak kasus, *stowage planning* masih berubah-ubah bahkan 2 jam sebelum proses *loading* selesai. Hal ini dikarenakan adanya perubahan pada daftar bongkar muat peti kemas (CLL), mulai dari penambahan peti kemas yang dimuat hingga adanya pembatalan bongkar muat peti kemas. Situasi ini menyebabkan *stowage planning* rentan akan *human error* [4]. Adanya *human error* menyebabkan biaya operasional pelabuhan lebih besar dan waktu yang dibutuhkan untuk kapal beroperasi juga lebih lama karena adanya proses *restow* yaitu proses penataan ulang posisi peti kemas di kapal untuk efisiensi ruang kapal. Untuk kasus *multiport*, penataan peti kemas ke kapal tidak hanya mempertimbangkan ukuran dan berat peti kemas. Namun juga lokasi pelabuhan tujuan peti kemas dikirim. Penataan peti kemas harus mempertimbangkan rute perjalanan kapal untuk meminimalisir pergerakan *crane* saat proses bongkar muat di setiap pelabuhan karena pergerakan *crane* dapat mempengaruhi total biaya yang harus ditanggung di setiap pelabuhan.

Dengan demikina, dibutuhkan perangkat lunak yang membantu *planner* agar *stowage planning* dapat dengan cepat dibuat meski *CLL* sering berubah-ubah. Perangkat lunak *stowage planning* yang ada saat ini telah menyediakan informasi mengenai kapal yang akan datang ke pelabuhan, seperti keadaan kargo yang mempengaruhi

stabilitas kapal, kekuatan kapal, trim, pemeriksaan visibilitas, *crane split*, maupun kargo berbahaya. Apabila kapal tidak dapat memberikan informasi secara lengkap, maka kapal tidak dapat dilayani di pelabuhan yang dituju yang artinya tidak akan ada proses pemindahan peti kemas [5].

Case-Based Reasoning (CBR) adalah metode pendekatan penyelesaian masalah menggunakan pengalaman penyelesaian masalah masa lalu yang memiliki kemiripan tertinggi. Sehingga seseorang dapat menganalisa satu atau beberapa penyelesaian dari masalah sebelumnya yang relevan untuk mencari solusi permasalahan baru [6]. Metode ini memungkinkan *planner* menghasilkan *stowage planning* yang lebih cepat dengan kualitas yang sama. Peningkatan kualitas dapat dilihat dari durasi waktu yang dilakukan *planner* lebih cepat dan bersamaan dengan hasil CLL yang lebih akurat [5]. Metode *Case-Based Reasoning* memiliki empat tahapan. Tahapan pertama yaitu *Retrieve* dimana pada tahap ini mulai memilah dan memilih masalah di masa lalu yang telah memiliki penyelesaian dan memiliki kemiripan tertinggi dengan masalah yang diselesaikan saat ini. Masalah masa lalu dan penyelesaiannya ini akan digunakan untuk tahap kedua, yaitu tahap *Reuse*. Tahap *Reuse* yaitu mengadaptasi daftar bongkar muat peti kemas saat ini agar menyerupai masalah yang dipilih di tahap sebelumnya. Kemudian, tahap ketiga yaitu tahap *Revise* dimana dilakukan peninjauan kembali hasil adaptasi yang telah dilakukan di tahap *Reuse* supaya menghasilkan solusi untuk masalah yang baru. Tahap *Revise* tidak harus dilakukan, mengikuti kondisi dari hasil tahap *Reuse*. Selanjutnya, tahap terakhir

yaitu *Retain* yang berguna untuk menyimpan masalah terbaru beserta penyelesaiannya ke dalam *Database* supaya dapat digunakan untuk menyelesaikan masalah-masalah berikutnya [7].

Pada penelitian sebelumnya, telah dibuat prototipe modul sistem *stowage planning* yang dapat berintegrasi dengan perangkat lunak *iStow*. Penelitian ini dilakukan oleh Nurdin Akbar Herwanto di tahun 2011 dengan menggunakan Metode Sistem Perencanaan Stowage Berdasarkan Kasus (*Casestow)* yang merupakan implementasi tahap *retrieval* dari metode *Case-Based Reasoning* [8]. Penelitian selanjutnya dilakukan oleh Agung Pratama pada tahun 2019 yang menghasilkan perangkat lunak yang dapat memudahkan penataan peti kemas untuk *loading* dengan menggunakan Metode *Case-Based Reasoning* pada tahap Reuse [9]. Namun, pada penelitian tersebut, perangkat lunak hanya dapat digunakan untuk proses *loading* dengan transportasi kontainer ke satu pelabuhan saja. Untuk penelitian lainnya yang berkaitan dengan *multiport* telah dilakukan di tahun 2013 hingga 2015 oleh beberapa peneliti dengan menggunakan metode MIP (*Mixed Integer Programming*) [1][2][11]. Penelitian *multiport* ini menghasilkan penataan peti kemas yang optimal hingga untuk 5.600 TEU kontainer dengan 7 rute pelayaran. Dari penelitian yang sudah ada sebelumnya, penulis tertarik untuk mengembangkan penelitian-penelitian tersebut hingga menghasilkan perangkat lunak yang dapat digunakan untuk memudahkan sistem penataan peti kemas pada rute *multiport*. Supaya perangkat lunak dapat meminimalisasi *restow*, memaksimalkan jumlah muatan dan memenuhi semua persyaratan keselamatan

serta persyaratan "subyektif" perusahaan pelayaran dan kapal dalam menghasilkan tatanan peti kemas pada rute *multiport* maka penulis menggunakan Metode *Case Based Reasoning* dalam pengerjaan ini.

Berdasarkan penelitian sebelumnya, penulis tertarik untuk mengembangkan penelitian dengan menerapkan tahap *Reuse* pada sistem penataan peti kemas rute *multiport* dengan Metode *Case Based Reasoning.* Sehingga penulis mengangkat sebuah penelitian Tugas Akhir dengan judul "*Pengembangan Modul Reuse pada Sistem Penataan Muatan Kapal Peti Kemas Berbasis CBR untuk Rute Multiport"*

1.2 Rumusan Masalah

Berdasarkan latar belakang diatas, permasalahan yang ingin diselesaikan dalam tugas akhir ini adalah

1. Bagaimana merancang algoritma untuk sistem penataan peti kemas pada rute *multiport* berbasis *Case Based Reasoning* pada tahap *Reuse.*
2. Bagaimana mengimplementasikan algoritma yang telah dirancang untuk sistem penataan peti kemas pada rute *multiport* berbasis *Case Based Reasoning* pada tahap *Reuse.*
3. Bagaimana pengaruh modul *Reuse* terhadap beban kinerja *planner* dalam melakukan penataan peti kemas *multiport.*

1.3 Batasan Masalah

Batasan masalah dalam tugas akhir ini adalah sebagai berikut:

1. Obyek uji algoritma adalah kapal KM Kendhaga Nusantara 1
2. Peti kemas yang digunakan berukuran 20ft atau 40ft
3. Muatan atau isi di dalam peti kemas adalah muatan tidak berbahaya
4. Jenis kontainer yang digunakan adalah *dry container standard*
5. *Container Loading List* (CLL) yaitu daftar kontainer yang akan dimuat, meliputi parameter pelabuhan asal dan tujuan, ukuran, berat
6. Kapal diasumsikan dalam keadaan stabil sehingga syarat keselamatan kapal dapat terpenuhi
7. Program ini dirancang khusus untuk mengimplementasi rangkaian algoritma pada tahap *Reuse* dengan kasus penataan peti kemas *multiport*.

1.4 Tujuan

Adapun tujuan dalam proposal tugas akhir ini adalah:

1. Merancang algoritma untuk sistem penataan peti kemas pada rute *multiport* berbasis *Case Based Reasoning* pada tahap *Reuse.*
2. Mengimplementasikan algoritma yang telah dirancang untuk menghasilkan penataan peti kemas dalam satu *voyage* dan mengetahui jumlah *restow* serta jumlah peti kemas yang mengalami bongkar muat di setiap pelabuhan.
3. Mengetahui dampak penggunaan modul *Reuse* terhadap beban kinerja *planner* dalam melakukan penataan peti kemas *multiport* dengan parameter

jumlah swap manual dalam satu voyage yang dipengaruhi oleh jumlah populasi casebase

1.5 Manfaat

Adapun manfaat pada proposal tugas akhir adalah sebagai berikut:

1. Membantu *stowage planner* dalam merencanakan penataan peti kemas lebih cepat terutama untuk rute pelayaran *multiport*
2. Membantu mengurangi biaya operasional pada proses *loading* dan *unloading* di setiap pelabuhan tujuan
3. Membantu mengembangkan perangkat lunak yang bermanfaat dalam dunia pelayaran

BAB II
TINJAUAN PUSTAKA

Pada bab ini akan dipaparkan penelitian sebelumnya yang berkaitan dengan penelitian Tugas Akhir ini dan teori-teori yang digunakan untuk mendukung pengerjaan penelitian Tugas Akhir ini.

2.1. Penelitian Terdahulu

Pada penelitian Tugas Akhir ini, penulis merujuk pada penelitian-penelitian yang telah dilakukan sebelumnya di bidang penataan peti kemas dan transportasi kontainer *multiport*. Pada tahun 2005, Setyo Nugroho melakukan penelitian yang berjudul "*CASESTOW: Recycling of past stowage plans"* yang membahas tentang *stowage planning* pada kapal kontainer menggunakan Metode *Case-Based Reasoning* dengan cara mengambil masalah tentang *stowage planning* yang memiliki penyelesaian sebelumnya dan memiliki kemiripan tertinggi untuk digunakan sebagai referensi. Metode *Case-Based Reasoning* dibagi menjadi 4 tahap, yaitu tahap *Retrieve*, tahap *Reuse*, tahap *Revise*, dan yang terakhir tahap *Retain* [5].

Kemudian di tahun 2011, Nurdin Akbar Harwanto melakukan penelitian yang berjudul "*Pengembangan Prototipe Modul Sistem Perencanaan Stowage Berdasarkan Kasus untuk Penataan Semi-Otomatis Peti Kemas pada Kapal*" yang menghasilkan output berupa prototipe untuk *stowage planning* yang terintegrasi dengan aplikasi *iStow*. Penelitian ini mengimplementasikan Metode *Case-Based Reasoning* terutama pada tahap

Retrieve. Sedangkan, untuk tiga tahap lainnya pada Metode *Case Based-Reasoning* masih dilakukan secara manual pada prototipe tersebut [8].

Pada tahun 2013, Daniela Ambrosino, Massimo Paolucci dan Anna Sciomachen melakukan penelitian yang berjudul "*Experimental evaluation of mixed integer programming models for the multi-port master bay plan problem*" yang membahas tentang urutan bongkar dan muat di setiap pelabuhan dari rute untuk pengiriman ke pelabuhan tujuan berturut-turut dengan memperkenalkan dua model *integer programming* (MIP) yang bertujuan untuk menganani aspek-aspek praktis dan operatif dari masalah penataan peti kemas. Hal ini juga meneliti beberapa komputasi yang efisien dan melaporkan eksperimen komputasi yang dapat digunakan secara luas dan dapat dilakukan untuk kasus nyata [11].

Kemudian pada tahun 2015, Daniela Ambrosino, Massimo Paolucci dan Anna Sciomachen mengembangkan penelitiannya dengan judul "*Computational evaluation of a MIP model for multi-port stowage planning problems*". Penelitian ini menyajikan sebuah model *integer programming new mixed* yang mampu mengelola skenario realistis dan mendapatkan rencana penataan peti kemas hingga 18.000 TEUs dengan kontainer yang dimuat diatas kapal di setiap rute pelabuhan terdiri dari kontainer standar, *reefer* dan *open top* atau terbuka yang atas [2]. Lalu, mengembangkan lagi penelitiannya dengan judul "*A MIP heuristic for multi port stowage planning*" yang membahas tentang efektivitas *stowage planning* di setiap rute pelabuhan menggunakan *Heuristic Mixed Integer Programming* yang merupakan pengembangan metode

pada penelitian sebelumnya. MIP heuristik ini diusulkan untuk menemukan solusi penyimpanan yang baik dalam waktu singkat terutama untuk kapal yang sangat besar dengan kapasitas 18.000 TEUs [1].

Pada tahun 2019, penelitian tentang penataan kontainer dilakukan oleh Agung Pratama dengan judul "*Penerapan Tahap Reuse pada Sistem Penataan Peti Kemas dengan Metode Case-Based Reasoning*". Penelitian ini menghasilkan output berupa perangkat lunak yang dapat mempermudah pengguna dalam membuat *stowage planning*. Metode yang digunakan dalam penelitian ini adalah Metode *Case-Based Reasoning* terutama pada tahap *Reuse* yang berguna unuk mengadaptasi *Container Loading List* agar menyerupai kasus yang telah diambil pada tahap sebelumnya, yaitu tahap *Retrieve*.

2.2. Landasan Teori

Pada bagian ini akan dijelaskan tentang teori-teori yang digunakan untuk mendukung pengerjaan penelitian Tugas Akhir ini.

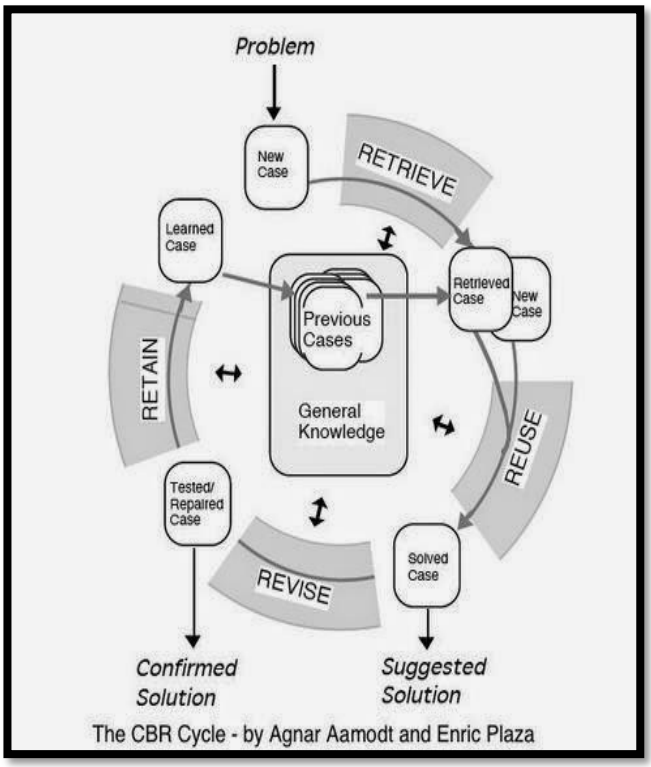
2.2.1. Metode Case-Based Reasoning

Metode *Case-Based Reasoning* merupakan salah satu metodologi yang digunakan untuk mencari penyelesaian masalah dengan cara memanfaatkan pengalaman untuk masalah sebelumnya yang hampir sama dengan masalah yang sedang dibahas. Umumnya, metode ini disajikan suatu masalah, masalah ini bisa berasal dari sistem atau program. Kemudian, dengan menggunakan metode ini, mulai mencari masalah lalu yang telah diselesaikan dan memiliki spesifikasi masalah yang sama dengan masalah baru yang akan diselesaikan. Apabila tidak

ada masalah yang sama, maka akan dicari beberapa masalah yang hamper serupa dengan masalah yang baru [12]. Metode *Case-Based Reasoning* memiliki empat tahapan, yaitu *Retrieve, Reuse, Revise, Retain*. Tahap *Retrieve* yaitu mengambil kembali kasus yang memiliki tingkat kemiripan tertinggi yang telah diselesaikan sebelumnya, dan memberikan saran solusi untuk tahap *Reuse* [7]. Masalah-masalah yang tidak identik dengan masalah yang baru akan membuat masalah yang baru mengalami adaptasi dan ini terjadi di tahap *Reuse.* Pada tahap *Reuse* dilakukan adaptasi *Container Loading List* menyerupai kasus yang telah diambil pada tahap *Retrieve*. *Revise* yaitu meninjau kembali hasil adaptasi *Container Loading List* yang telah di dapat pada tahap *Reuse* agar menjadi solusi untuk permasalahan yang baru, kemudian masuk ke tahap *Revise* dimana ini merupakan tahapan yang kondisional, tahapan yang boleh dilakukan maupun tidak dilakukan (sesuai kebutuhan). Yang terakhir adalah tahap *Retain* yaitu menyimpan masalah terbaru yang telah diselesaikan ke dalam *Database* untuk digunakan menyelesaikan permasalahan-permasalahan berikutnya [7]. Struktur sistem *Case Based Reasoning* mencakup mekanisme penalaran dan aspek eksternal, meliputi spesifikasi masukan atau masalah dari suatu permasalahan, solusi yang diharapkan sebagai luaran, dan masalah-masalah sebelumnya yang tersimpan sebagai referensi pada mekanisme penalaran [6].

Pada Metode *Case-Based Reasoning*, semakin banyak data yang tersimpan dalam *database*, maka solusi yang dihasilkan untuk masalah baru semakin baik dibandingkan jika data yang tersimpan di *database* sedikit,

maka hasil atau solusi yang didapatkan untuk menyelesaikan masalah baru juga kurang baik. Diagram alir Metode *Case-Based Reasoning* dapat diilustrasikan seperti pada



Gambar 2. 1 Diagram Alir Metode Case-Based Reasoning [7]

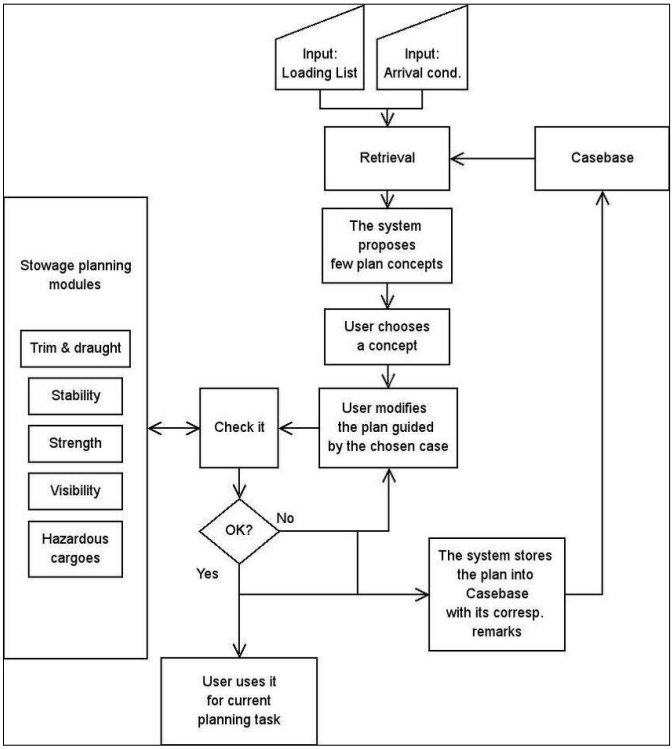
2.2.2. Casestow

Casestow merupakan implementasi dari Metode *Case-Based Reasoning* yang digunakanan pada *stowage planning*, sehinggga proses *stowage planning* menjadi semi-otomatis. Metode ini tentu saja dapat meringankan beban dari *planner*. Ketika *stowage planner* mendapatkan masalah baru, maka ia akan mulai memanfaatkan pengalaman untuk mengatasi masalah *stowage planning* sebelumnya untuk mencari solusi dari masalah *stowage planning* yang baru. Intinya adalah dapat mengatur pengalaman untuk melakukan perencanaan, sehingga proses pengambilan dan penggunaan kembali suatu kasus dapat dijalankan. Proses seperti ini juga terdapat pada proses *Casestow*.

Input Casestow terdiri dari dua elemen, yaitu Kondisi Kapal Setelah Proses Bongkar Muat (KKSPB) dan *Container Loading List* (CLL) yaitu daftar peti kemas yang akan dimuat. Permasalahan *stowage planning* dapat diuraikan sebagai berikut:

a. Masalah :
   Kondisi Kapal Setelah Proses Bongkar Muat (KKSPB), *Container Loading List* (CLL).
b. Solusi :
   Stowage planning baru (aktual).
c. Penjelasan:
   Kualitas *stowage planning* dengan melihat dari parameter-parameter seperti stabilitas kapal, kekuatan kapal dan lain sebagainya

Casestow dapat di ilustrasikan dengan diagram alir seperti pada Gambar 2.2.



Gambar 2. 2 Diagram Alir Proses *Casestow.*

Rincian penjelasan Gambar 2.2 diatas adalah sebagai berikut:

a. Dalam proses *casestow*, ada dua input yang digunakan yaitu *Container Loading List (CLL)* dan kondisi kapal ketika datang ke pelabuhan

b. Terjadi proses *retrieve* dimana sistem menampilkan beberapa *casebase* yang memiliki kemiripan tertinggi dengan kondisi input
c. Lalu *stowage planner* memilih *casebase* mana yang akan dipakai sebagai acuan untuk kasus baru, pemilihan ini dilakukan secara manual jadi pengaruh juga dengan seberapa pengalaman *stowage planner* yang memilih
d. Setelah itu, *stowage planner* atau *user* mulai memodifikasi *stowage planning* untuk kasus baru dengan acuan dari *casebase* yang telah dipilih
e. *Stowage planning* yang sudah jadi mulai dicek sesuai dengan modul yang telah ada di *iStow*, seperti: stabilitas kapal, *trim* dan *draft*, muatan berbahaya dan lainnya
f. Apabila semua hasil pengecekan baik, maka *stowage planning* dapat digunakan. Namun jika stabilitas, kekuatan atau modul lainnya ada yang kurang baik, maka kembali ke tahap *retrieve* guna memilih *casebase* lain yang dapat digunakan sebagai acuan sampai pengecekan *stowage planning* yang baru menghasilkan hasil yang baik.

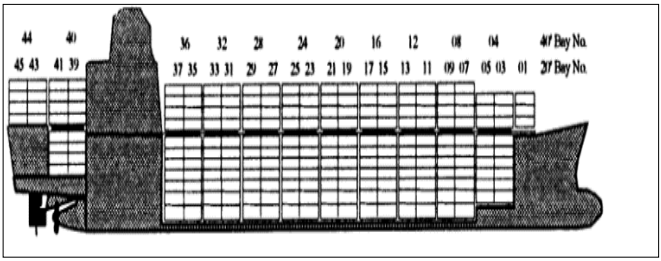
Perlu diketahui, bahwa hasil *stowage planning* yang kurang baik akan digunakan *user* atau *stowage planner* untuk dijadikan pengalaman agar tidak mengulangi pengembangan *stowage planning* yang sama dan peringatan juga agar lebih berhati-hati. Hal ini dilakukan supaya *stowage planning* dapat dikerjakan dengan lebih cepat dan lebih berkualitas [9].

Dr.-Ing. Setyo Nugroho telah mematenkan metode *casestow* pada tahun 2004 di Kantor Paten dan Merk

Jerman (*Deutches Patent- und Markenamt*), dan pada tahun 2005 Eropa (*The European Patent Office*) telah mengakuinya [13].

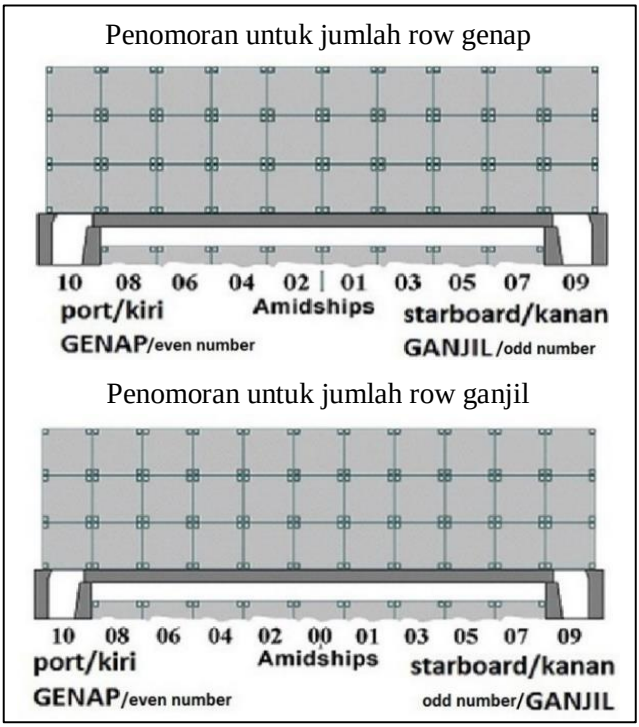
2.2.3. Konsep koordinat *bay-row-tier*

Konsep koordinat *bay-row-tier* digunakan agar mempermudah visualisasi proses penataan peti kemas. Konsep *bay-row-tier* mengikuti sistem koordinat numerik yang berkaitan dengan panjang, lebar dan tinggi. *Bay* merupakan sistem koordinat numerik penataan peti kemas untuk arah melintang, pada peti kemas berukuran 20ft dimulai dengan angka 01 dan berlanjut dengan penambahan 2 angka pada nomor kontainer dibelakangnya, sedangkan untuk peti kemas berukuran 40ft dimulai dengan angka 02 dan berlanjut dengan penambahan 4 angka pada nomor kontainer dibelakangnya. Seperti pada contoh gambar dibawah ini



Gambar 2. 3 Konsep Koordinat Numerik *Bay*.

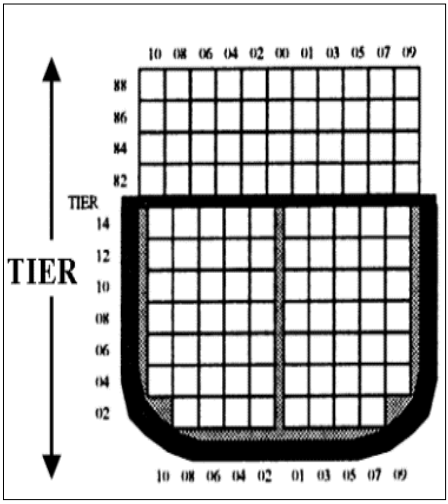
Row merupakan sistem koordinat numerik penataan peti kemas untuk baris memanjang. Ada perbedaan untuk jumlah *row* genap dan jumlah *row* yang ganjil. Perbedaan ini dapat dilihat dari gambar di bawah ini



Gambar 2. 4 Konsep Koordinat Numerik *Row*.

Tier merupakan sistem koordinat numerik penataan peti kemas untuk lapisan vertical atau tumpukan. Kontainer diberi nomor dengan nomor genap, mulai dari lapisan terbawah. Biasanya dimulai dengan angka 02 kemudian bertambah 2 angka untuk lapisan atasnya secara berurutan.

Untuk peti kemas yang terletak di *deck* kapal, diberi nomor genap juga tapi dimulai dengan nomor 80 atau 82 dan akan terus bertambah 2 angka untuk lapisan yang lebih tinggi. Contoh penomoran untuk konsep *tier* dapat dilihat pada gambar dibawah ini



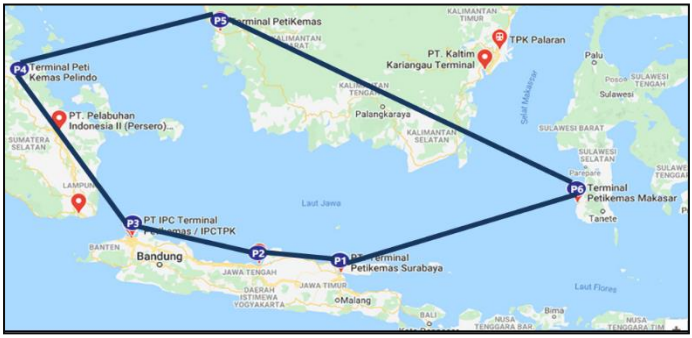
Gambar 2. 5 Konsep Koordinat Numerik *Tier*.

2.2.4. Konsep Multiport

Penelitian mengenai *stowage planning* biasanya menyangkut tentang pengiriman kontainer dengan jasa kapal yang melalui jalur pelayaran *singleport* atau *multiport*. Permasalahan dalam penataan kontainer dengan jalur *singleport* hanya mempertimbangkan cara membuat *stowage planning* untuk penyimpanan kontainer di kapal yang akan dikirim dari pelabuhan saat ini ke satu pelabuhan berbeda lainnya yang merupakan tujuan pelabuhan pengiriman kontainer. Berbeda dengan permasalahan

multiport, pembuatan *stowage planning* harus dilakukan untuk setiap pelabuhan dalam satu rute kapal. Seluruh pelabuhan dalam rute kapal harus diselidiki untuk menghasilkan *stowage planning* di setiap pelabuhan.

Kapal mengunjungi beberapa pelabuhan selama pelayarannya, sementara kontainer akan selalu dimuat atau dibongkar di setiap pelabuhan dalam satu rute kapal. *Stowage planning* yang dibuat untuk setiap pelabuhan menggabungkan *stowage planning* yang dibuat pada pelabuhan sebelumnya dengan informasi kontainer yang akan dibongkar dan dimuat di pelabuhan tersebut, sehingga *stowage planning* pada setiap pelabuhan saling berkaitan. Dalam penataan kontainer dengan jalur pelayaran *multiport*, daftar kontainer yang akan dimuat tidak hanya menggunakan parameter berat, ukuran, dan jenis kontainer, tapi juga harus memuat parameter pelabuhan asal dan pelabuhan tujuan pengiriman container



Gambar 2. 6 Rute kapal dengan 6 pelabuhan

Gambar di atas menunjukkan salah satu contoh rute pengiriman kontainer dengan jalur pelayaran *multiport*,

dimana satu rute perjalanan kapal memuat enam pelabuhan yang harus dilalui dengan rincian sebagai berikut :

P1 : Terminal Peti Kemas wilayah Surabaya
P2 : Terminal Peti Kemas wilayah Semarang
P3 : Terminal Peti Kemas wilayah Jakarta
P4 : Terminal Peti Kemas wilayah Jambi
P5 : Terminal Peti Kemas wilayah Pontianak
P6 : Terminal Peti Kemas wilayah Makasar

Kapal akan berangkat dari Surabaya ke Semarang dengan memuat kontainer yang akan dikirimkan ke pelabuhan di Semarang, Jakarta, Jambi, Pontianak dan Makasar. Saat kapal tiba di Semarang, terjadi proses penurunan kontainer yang harus dikirim ke Semarang dan proses muat kontainer ke kapal yang harus dikirim dari Semarang ke kota berikutnya, yaitu Jakarta, Jambi, Pontianak, Makasar termasuk juga Surabaya. Namun, untuk kontainer yang dikirim ke Surabaya, kapal harus menyelesaikan perjalananya dalam satu rute terlebih dahulu sampai di pelabuhan terakhir yaitu di Makasar, kemudian kapal kembali ke Surabaya. Pola ini juga berlaku di semua pelabuhan dalam satu rute kapal, akan selalu ada proses bongkar atau muat kontainer yang terjadi pada setiap pelabuhan yang dikunjungi. Rute kapal dibuat berdasarkan daftar kontainer yang akan dimuat di semua pelabuhan terutama berhubungan dengan parameter asal dan tujuan kontainer dikirim. Rute kapal yang dibuat diatas berdasarkan ringkasan daftar kontainer yang akan dimuat dibawah ini

Tabel 2. 1 *Container Loading List* dengan rute Surabaya – Makasar

Kode Track	Jumlah Kontainer *Loading*	Asal	Tujuan
101	5 TEUs	Surabaya	Jambi
	5 TEUs	Semarang	Jambi
	6 TEUs	Makasar	Semarang
	3 TEUs	Pontianak	Jakarta
	11 TEUs	Surabaya	Pontianak
	7 TEUs	Jakarta	Makasar
	2 TEUs	Pontianak	Surabaya
	3 TEUs	Jambi	Surabaya
	16 TEUs	Jambi	Jakarta
	14 TEUs	Semarang	Pontianak
	7 TEUs	Jakarta	Jambi
	8 TEUs	Jakarta	Pontianak
	3 TEUs	Makasar	Surabaya
	12 TEUs	Jambi	Semarang
	3 TEUs	Pontianak	Semarang
	7 TEUs	Makasar	Jakarta
	5 TEUs	Semarang	Makasar

Daftar kontainer yang akan dimuat diatas dapat diringkas dengan membuat *voyage*. Manfaat dari membuat *voyage* ini adalah memudahkan *planner* mengetahui berapa kontainer yang harus dimuat dan dibongkar di setiap pelabuhan dan berapa kontainer yang berada di kapal saat kapal berlayar ke pelabuhan selanjutnya. Selain itu, *planner* dapat mengetahui pelabuhan tujuan mana saja

yang harus dilalui kapal sehingga dapat dibentuk sebuah rute perjalanan kapal.

Tabel 2. 2 *Voyage* 1 dari *Container Loading List* Surabaya – Makasar

Voyage 1

TO / FROM	P1	P2	P3	P4	P5	P6
P1	0	0	0	5	11	0
P2	0	0	0	5	14	5
P3	0	0	0	7	8	7
P4	3	12	16	0	0	0
P5	2	3	3	0	0	0
P6	3	6	7	0	0	0

Dari tabel *voyage 1* dapat diketahui bahwa kapal mengangkut total kontainer yang dimuat oleh kapal selama satu rute perjalanan mencapai 117 TEUs dengan beberapa rincian proses seperti yang terjadi pada pelabuhan P4 atau di wilayah Jambi, terjadi proses penurunan kontainer sebanyak 17 TEUs, yaitu 5 TEUs dari Surabaya (P1), 5 TEUs dari Semarang (P2) dan 7 TEUs dari Jakarta (P3). Setelah dilakukan penurunan kontainer, sisa kontainer yang ada di kapal sebanyak 45 TEUs dengan rincian 11 TEUs dari Surabaya (P1) yang harus dikirim ke Pontianak (P5), 19 TEUs dari Semarang (P2) yang harus dikirim ke Pontianak (P5) dan Makasar (P6), serta 15 TEUs dari Jakarta (P3) yang harus dikirim ke Pontianak (P5) dan Makasar (P6) juga. Kemudian terjadi prosen *loading* kontainer dari Jambi (P4) sebanyak 31 TEUs untuk dikirim ke Surabaya (P1) sebanyak 3 TEUs, Semarang (P2)

sebanyak 12 TEUs dan Jakarta (P3) sebanyak 16 TEUs. Sehingga total kontainer yang dibawa oleh kapal saat berlayar dari kota Jambi (P4) ke Pontianak (P5) adalah sebanyak 76 TEUs. Jumlah kontainer yang dibawa oleh kapal dari pelabuhan awal ke pelabuhan setelahnya ini diringkas dalam sebuah tabel *on board* seperti contoh dibawah ini

Tabel 2. 3 *On Board* dari *Voyage* 1

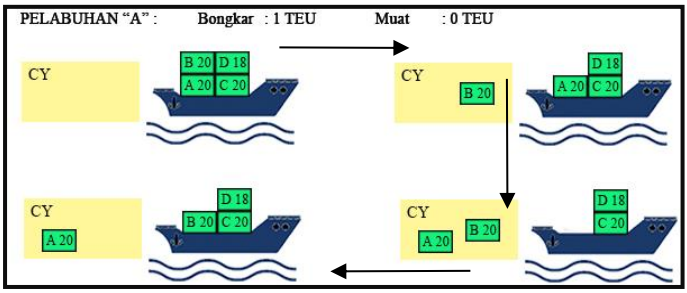
On Board

TO / FROM	P1	P2	P3	P4	P5	P6
P1	-	16	-	-	-	-
P2	-	-	40	-	-	-
P3	-	-	-	62	-	-
P4	-	-	-	-	76	-
P5	-	-	-	-	-	51
P6	55	-	-	-	-	-

2.2.5. Konsep Restow

Restow adalah proses dimana terjadi penurunan kontainer dari kapal dan dimuat kembali ke kapal yang sama dalam satu lokasi pelabuhan atau terminal. Hal ini bisa terjadi salah satunya karena ada kontainer yang terletak di dalam kapal bagian bawah yang harus dipindahkan di pelabuhan tersebut. Namun, karena terhalang oleh kontainer diatasnya, maka dilakukan penurunan kontainer atas terlebih dahulu, lalu kontainer dibawahnya dapat diambil dan dipindahkan ke pelabuhan. Apabila tujuan pelabuhan kontainer yang terletak di atas ini berbeda, maka kontainer harus diletakkan kembali ke kapal

untuk dikirim ke pelabuhan tujuan. Seperti dalam ilustrasi dibawah ini



Gambar 2. 7 Ilustrasi Proses Restow

Ketika kapal membawa 4 kontainer ke pelabuhan A dan harus menurunkan 1 kontainer saja dimana posisi kontainer ini berada dibawah kontainer tujuan pelabuhan B, maka kontainer untuk pelabuhan B harus diturunkan terlebih dahulu, untuk kemudian menurukan kontainer untuk pelabuhan A. Setelah itu, kontainer pelabuhan B harus dimuat lagi pada kapal yang sama untuk dikirim ke pelabuhan B. Proses ini lah yang dinamakan dengan *restow. Restow* cukup merugikan untuk pengguna jasa maupun pihak pelabuhan, karena selain memakan waktu yang lebih lama dalam penyelesaian bongkar muat di pelabuhan tersebut, *restow* juga dapat memakan biaya yang lebih besar, karena adanya pergerakan yang lebih banyak saat terjadi proses bongkar muat. Apabila dalam ilustrasi pada gambar 2.9, posisi kontainer untuk pelabuhan A dan kontainer untuk pelabuhan B ditukar, maka hanya perlu satu kali pergerakan saja, yaitu hanya menurunkan kontainer untuk pelabuhan A saja. Sehingga, waktu yang dibutuhkan untuk menyelesaikan proses bongkar kontainer

lebih cepat dan dengan pergerakan alat yang lebih sedikit maka biaya yang dikeluarkan juga lebih kecil. *Restow* tidak bisa dihindari dalam penataan *stowage* planning terutama jika posisi kontainer yang menyebabkan *restow* ini mempengaruhi kestabilan kapal. Namun, *restow* masih dapat diminimalisir dengan penataan *stowage planning* yang baik.

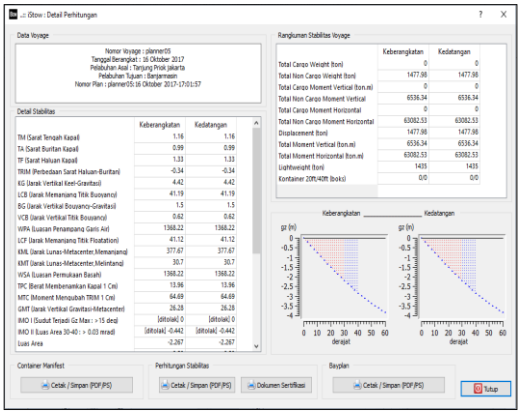
2.2.6. iStow

iStow adalah perangkat lunak *stowage planning* kapal kontainer untuk mendukung kegiatan operasional lapangan. Arsitektur dari iStow disesuaikan dengan bisnis proses dari masing-masing perusahaan yang unik. iStow memungkinkan pembuatan *stowage planning* secara kolaboratif melalui jaringan dengan tingkat keamanan yang tinggi. Selain itu, arsitektur iStow juga disiapkan untuk dapat berintegrasi dengan sistem-sistem yang lain, misalnya sistem *booking*, sistem depo/*yard planning*, dan lain-lain.

Perhitungan kondisi kapal otomatis akan dilakukan setiap ada perubahan pada *stowage planning* dan dokumen-dokumen penunjang untuk penerbitan Surat Ijin Berlayar (SIB), seperti *manifest*, *bayplan*, *stowage planning*, perhitungan stabilitas dapat diproduksi secara otomatis. Perangkat lunak ini dapat berjalan dalam sistem operasi: Linux, Windows, Solaris, Macintosh.

Gambar 2. 8 Tampilan stowage planning pada iStow

Pada perangkat lunak iStow ini sudah terdapat beberapa modul penunjang, diantaranya yaitu Modul Stabilitas Kapal seperti pada Gambar 2.11. Saat ini iStow telah diimplementasikan pada beberapa kapal, diantaranya KM Kendhaga Nusantara 1, KM Sinar Jambi, KM Sinar Demak, KM Sinar Bintan dan KM Sinar Padang [6].



Gambar 2. 9 Modul Stabilitas pada iStow

Selain itu, iStow memiliki fitur IMO *requirements check.* Nomor IMO ini memegang peran penting dalam melacak keberadaan kapal dan mempermudah dalam mengidentifikasi kapal, sehingga apabila terjadi pencurian kapal, ciri-ciri kapal dapat dilakukan pengecekan melalui registrasi nomor IMO kapal tersebut.

BAB III
METODE PENELITIAN

Pada bab ini dijelaskan mengenai langkah-langkah yang digunakan dalam menerapkan tahap *Reuse* pada sistem penataan peti kemas dengan transportasi kontainer *multiport* menggunakan Metode *Case-Based Reasoning*. Adapun tahapan penelitian yang akan dilakukan adalah sebagai berikut:

a. Studi Literatur

Langkah pertama yang dilakukan adalah studi literature dengan referensi dari buku, jurnal serta sumber materi lainnya yang mendukung penyelesaian masalah *Stowage Planning,* Metode *Case-Based Reasoning* dan pelayaran *multiport.*

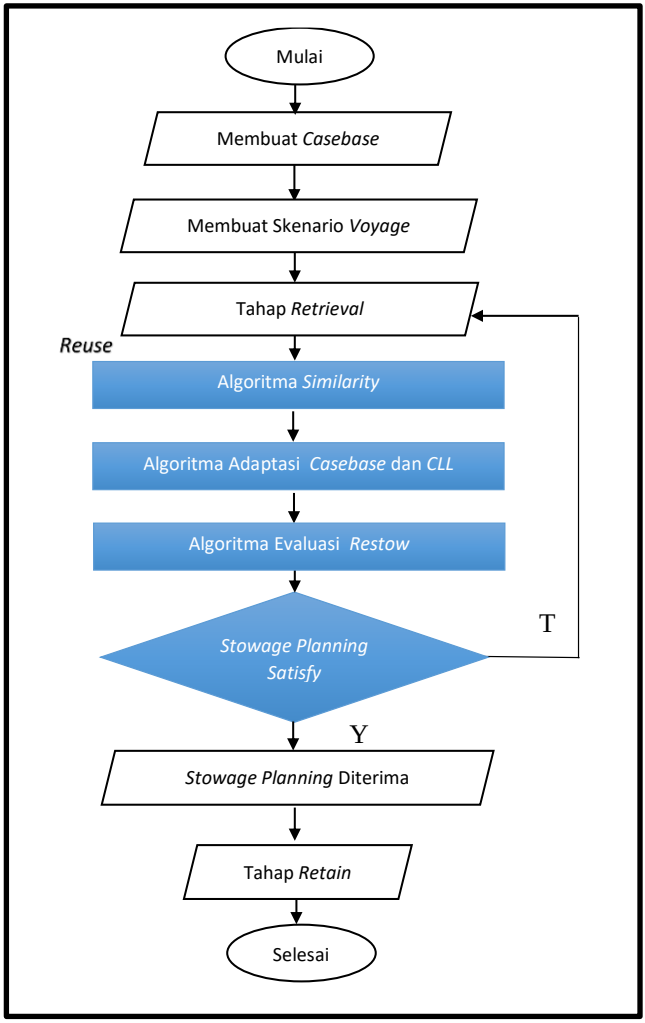
b. Pengumpulan Data

Pada langkah ini mulai dilakukan pengumpulan data berupa masalah penataan peti kemas dengan jalur transportasi *multiport (casebase)* yang memiliki kemiripan tertinggi dengan masalah yang dibahas. Data ini digunakan untuk mengadaptasi *Container Loading List* untuk masalah yang diangkat di Tugas Akhir ini.

c. Analisis dan Perancangan Algoritma

Pada langkah ini dilakukan analisis data untuk digunakan di tahap *Reuse* sebagai salah satu tahap dalam Metode *Case-Based Reasoning* pada sistem penataan peti kemas kemudian mulai merancang algoritma dari hasil analisis data tersebut, seperti Algoritma *Similarity,* Algoritma Adaptasi *Casebase* dan *CLL*, Algoritma Evaluasi *Restow.* Alur dari

rancangan algoritma ini dapat dilihat dari diagram alir berikut:



Gambar 3. 1 Diagram Alir Rancangan Penelitian

Penjelasan diagram alir pada gambar 3.1 diatas adalah sebagai berikut:

1. Pada langkah awal ini dibuat *casebase* sebanyak-banyaknya untuk database program yang akan dibuat. *Casebase* berisi penataan peti kemas yang pernah digunakan pada kasus *stowage planning* sebelumnya. Semakin banyak *casebase* dibuat maka semakin mudah *planner* membuat *stowage planning* baru secara otomatis karena kasus baru telah tersedia di *casebase*. *Casebase* berisi kode rute, *Container Loading List, stowage planning* dan komentar. Komentar bisa berupa numerik seperti kondisi stabilitas dan trim kapal atau berupa komentar verbal seperti permintaan khusus dari beberapa perusahaan terkait letak peti kemas yang akan dikirim pada saat di kapal.
2. Langkah kedua adalah membuat skenario *voyage*. Hal ini dibutuhkan untuk mengetahui pelabuhan mana saja tempat kapal akan transit selama satu rute perjalanan dan berapa kontainer yang harus *loading* dan *unloading* di masing-masing pelabuhan. Dengan melakukan tahap ini, dapat diketahui juga jumlah total kontainer yang diangkut oleh kapal saat sampai atau meninggalkan pelabuhan transit selama satu rute perjalanan.
3. Langkah ketiga yaitu tahap *retrieval* dimana menampilkan beberapa *casebase* yang hampir sama dengan *Container Loading List* yang baru untuk digunakan di langkah selanjutnya. Kemiripan *casebase* tidak hanya dilihat dari satu pelabuhan saja, tapi juga untuk satu rute perjalanan.

4. **Algoritma Similarity** digunakan untuk melakukan penataan peti kemas pada *Container Loading List* baru yang memiliki golongan berat dan pelabuhan tujuan yang sama dengan peti kemas pada *casebase*. *Planner* dapat memilih penataan peti kemas secara otomatis atau secara manual.
5. **Algoritma Adaptasi *Container Loading List*** digunakan untuk mengadaptasi *Container Loading List* pada kasus baru agar membentuk *stowage planning* yang utuh dimana semua peti kemas pada *Container Loading List* baru telah dilakukan penataan.
6. **Algoritma Evaluasi *Restow*** bertujuan untuk mengetahui jumlah peti kemas yang terkena *restow* di setiap pelabuhan dan total keseluruhan *restow* yang terjadi selama satu rute perjalanan.
7. Setelah dipilih satu *stowage planning* dari langkah sebelumnya, dilakukan pengecekan ulang untuk standard kepuasan melalui kolom komentar, jika *stowage planning* yang terpilih sudah memenuhi syarat seperti stabilitas dan permintaan khusus dari pelanggan atau perusahaan yang mengirim peti kemasnya untuk lokasi peti kemas di kapal, maka *stowage planning* dapat dijalankan dan diterima. Namun, jika belum memenuhi kepuasan dari pelanggan, *stowage planning* harus dilakukan perbaikan dengan cara menukar lokasi peti kemas sesuai dengan komentar yang diminta.
8. *Stowage planning* yang telah diterima, masuk ke dalam tahap *retain* yaitu penyimpanan solusi *stowage planning* yang ada untuk membantu penyelesaian masalah *stowage planning* yang baru nantinya.

Sehingga, setiap *stowage planning* yang diterima akan ditambahkan ke dalam *casebase*.

9. Tatanan peti kemas yang telah menghasilkan trim dan GM yang sesuai dengan syarat dianggap layak untuk berlayar. Hal ini karena dapat dipastikan bahwa kapal dengan tatanan peti kemas tersebut dalam keadaan stabil, jadi aman untuk membawa muatan sampai ke pelabuhan selanjutnya.

d. Implementasi Algoritma

Pada langkah ini dilakukan pengimplementasian algoritma yang telah dirancang pada langkah sebelumnya untuk menjadi *sourcecode* perangkat lunak yang dapat digunakan secara umum. Sehingga, perangkat lunak dapat digunakan dengan mudah oleh siapa saja. Metode *Case-Based Reasoning* pada tahap *Reuse* menunjang keberhasilan langkah ini.

e. Uji Coba dan Validasi

Pada langkah ini akan dilakukan pengujian terhadap *sourcecode* hasil implementasi dari algoritma tahap *Reuse* pada sistem penataan peti kemas yang bertujuan untuk mengetahui apakah *output* dari implementasi algoritma sesuai dengan yang diharapkan. Hal ini juga dilakukan untuk melakukan validasi program dengan cara memasukkan data yang sudah pernah digunakan di kehidupan nyata ke dalam program. Jika output dari algoritma yang telah dibuat menghasilkan penataan peti kemas yang sama dengan output pada masalah di kehidupan nyata yang digunakan, maka sistem dianggap valid. Namun, jika tidak sesuai dengan output pada masalah di kehidupan

nyata, maka perlu adanya revisi algoritma yang telah dibuat.

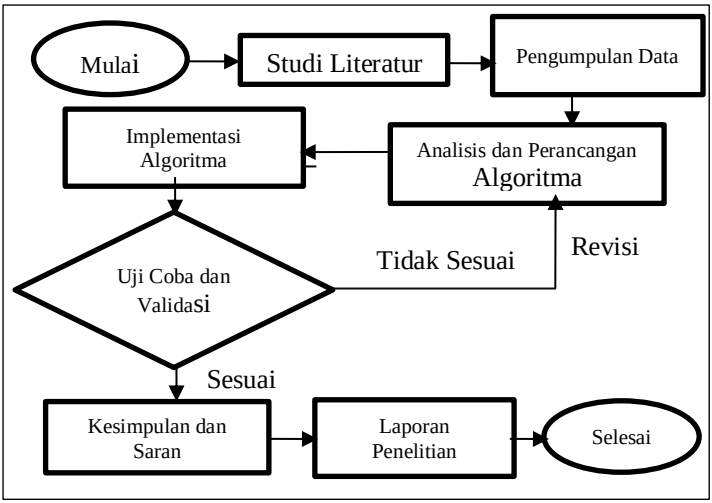
f. Kesimpulan dan Saran

Dilakukan penarikan kesimpulan dari uji coba di langkah sebelumnya dan penulisan saran untuk pengembangan penelitian selanjutnya. Output yang sesuai dengan yang diharapkan telah dihasilkan dan akan disimpan untuk membantu penyelesaian masalah baru lainnya.

g. Pembuatan Laporan Tugas Akhir

Langkah terakhir dala metode penelitian ini adalah penulisan Tugas Akhir dengan hasil penelitian atau *output* berupa perangkat lunak yang dihasilkan dengan menggunakan Metode *Case-Based Reasoning*.

Langkah – langkah yang telah dijelaskan diatas, dapat diilustrasikan dalam diagram alir dibawah ini



Gambar 3. 2 Diagram Alir Metodologi Penelitian

BAB IV
PENGEMBANGAN PERANGKAT LUNAK

Pada bab ini akan dibahas dan diuraikan terkait analisis, perancangan serta implementasi sistem pada penelitian Tugas Akhir ini.

4.1. Analisis Sistem

Penelitian ini menggunakan KM Kendhaga Nusantara 1 yang merupakan kapal kontainer dengan muatan maksimal 100 TEUs dan rute pelayaran multiport. Untuk mengembangkan perangkat lunak yang mampu melakukan penataan peti kemas pada *Container Loading List* untuk rute pelayaran multiport secara otomatis di setiap pelabuhan berdasarkan *Casebase,* diperlukan sistem yang dapat mengolah data pada *Container Loading List* dan *Casebase.* Karena penelitian tugas akhir ini menggunakan metode *Case Based Reasoning* yang difokuskan pada tahap *reuse,* maka *Container Loading List* dan *casebase* diasumsikan telah didapat dari tahap *retrieve*, sehingga pengguna dapat langsung memilih penataan peti kemas secara otomatis maupun manual tanpa harus menata peti kemas satu per satu ke dalam kapal. Penataan peti kemas dilakukan di setiap pelabuhan dengan *Container Laoding List* dan *casebase* yang berbeda beda di setiap pelabuhan menyesuaikan dengan pelabuhan keberangkatan kapal. Penataan pada pelabuhan pertama mempengaruhi penataan di semua pelabuhan dalam satu rute pelayaran.

Ketika pengguna memilih menata secara otomatis maka akan didapatkan hasil tatanan peti kemas yang menyerupai *casebase*. Setelah itu pengguna dapat melakukan perubahan pada letak beberapa peti kemas

apabila hasil tatanan tidak memenuhi standar penataan peti kemas seperti adanya penataan peti kemas 20*ft* yangberada diatas peti kemas 40*ft*. Sehingga didapatkan hasil tatanan peti kemas yang sesuai dengan standar penataan peti kemas pada kapal dan memenuhi syarat *subyektif* dari *planner*. Namun, jika pengguna memilih menata secara manual, maka peti kemas akan ditata secara terurut dari peti kemas yang memiliki beban terberat terletak di bagian bawah hingga beban teringan berada di bagian atas kapal tanpa memperhatikan pelabuhan tujuan masing-masing peti kemas dikirim. Pengguna dapat mengaturnya secara manual dengan cara menukar letak peti kemas. Penataan manual ini dilakukan pada masing-masing pelabuhan.

Selanjutnya, akan dijelaskan mengenai kebutuhan fungsional, kebutuhan non-fungsional, *use case diagram, activity diagram,* dan diagram blok dari tahap *resue* pada metode *Case Based Reasoning* untuk penataan peti kemas pada rute pelayaran *multiport.*

4.1.1. Analisis Kebutuhan Fungsional

Dalam pengembangan perangkat lunak ini, sistem diharapkan mampu melakukan hal sebagai berikut:

1. Pengembangan perangkat lunak ini khusus untuk KM Kendhaga Nusantara 1
2. Sistem dapat melakukan penataan peti kemas sesuai dengan *Container Loading List* di setiap pelabuhan secara otomatis yang diadaptasi dari *Casebase* yang dipilih.
3. Sistem dapat menampilkan visualisasi hasil tatanan
4. Sistem dapat memberikan informasi letak peti kemas di kapal

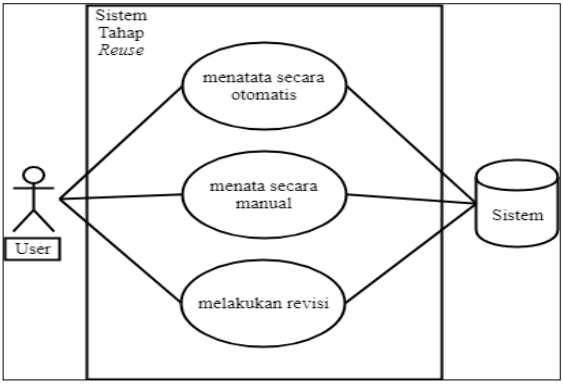
5. Sistem dapat menghitung jumlah peti kemas yang mengalami *restow* di setiap pelabuhan
6. Sistem dapat melakukan penataan peti kemas secara manual sehingga pengguna diberi keleluasaan untuk mengatur tatanan peti kemas
7. Sistem dapat membebaskan pengguna mengubah letak peti kemas pada hasil penataan peti kemas yang otomatis apabila ada peti kemas yang letaknya tidak sesuai dengan syarat subyektif dari pengirim peti kemas

4.1.2. Analisis Kebutuhan Non-Fungsional

Perangkat lunak ini dibangun menggunakan bahasa pemrograman JAvA dengan NetBeans IDE 12 dan menggunakan laptop dengan prosesor Intel Core i3 dan *Random Access Memory 2 GigaByte.*

4.1.3. Use Case Diagram

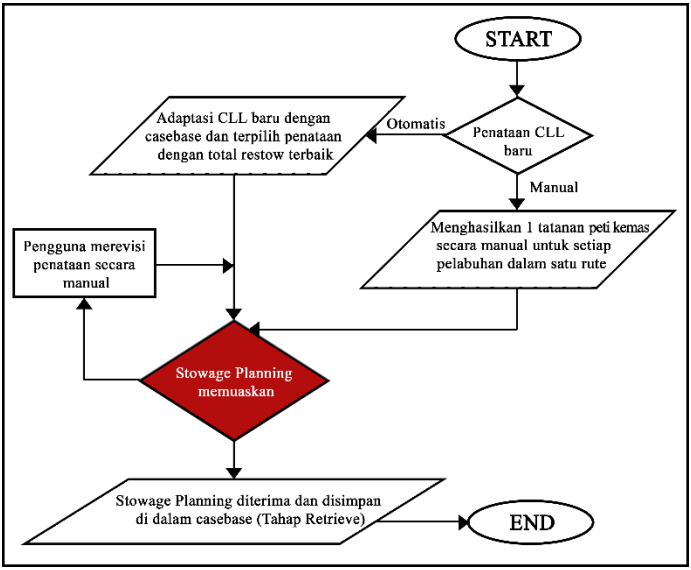
Use Case Diagram adalah diagram yang menggambarkan secara ringkas siapa yang menggunakan sistem ini dan apa saja yang dapat dilakukannya. *Use Case Diagram* pada sistem ini dapat dilihat dari gambar 4.1 dibawah ini



Gambar 4. 1 Use Case Diagram Sistem

4.1.4. Activity Diagram

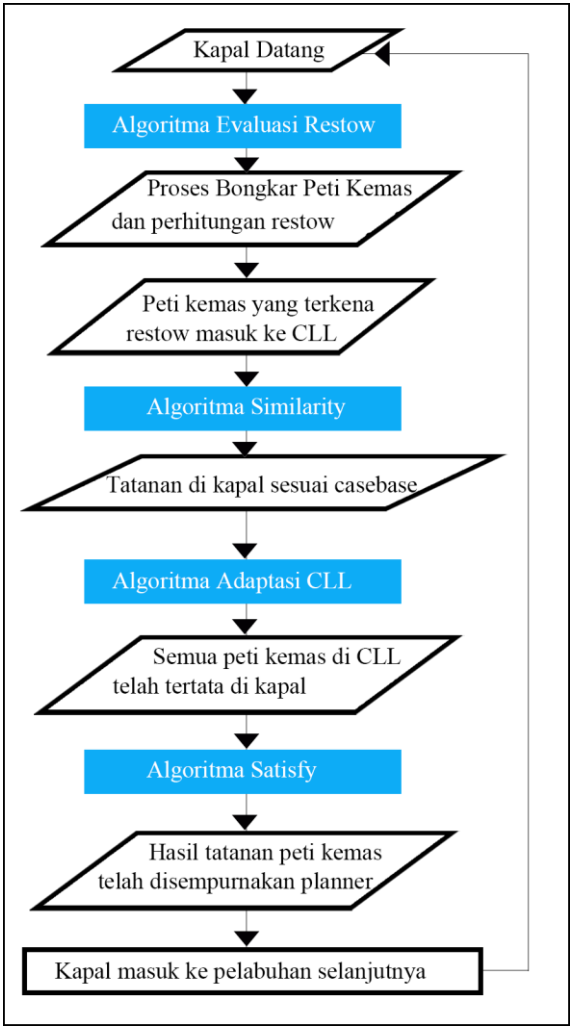
Activity Diagram dapat memperlihatkan urutan aktifitas proses pada sistem sehingga dapat membantu memahami proses sistem secara keseluruhan.



Gambar 4. 2 *Activity Diagram* pada Sistem

4.1.5. Diagram Alir Algoritma Tahap Reuse

Adapun interpretasi proses dari sistem yang dirancang dapat diperhatikan pada diagram di bawah ini



Gambar 4. 3 Diagram Alir Algoritma Tahap Reuse.

4.1.5.1. Algoritma Similarity

Algoritma adaptasi *Similarity* ini bertujuan untuk mengadaptasi *Container Loading List* di setiap pelabuhan keberangkatan agar menjadi tatanan yang mirip dengan *casebase* terutama untuk peti kemas yang memiliki golongan dan pelabuhan tujuan yang sama. Berikut ini merupakan algoritma dari fungsi *similarity*.

Input: Container Loading List yang belum tertata, *casebase* hasil tahap retrieve untuk setiap pelabuhan dan kondisi kapal ketika sampai di pelabuhan.

Langkah 1. Untuk seluruh lokasi peti kemas pada *casebase*, lakukan langkah 2-5

Langkah 2. Mengecek golongan berat dan pelabuhan tujuan seluruh peti kemas pada *casebase*

Langkah 3. Untuk seluruh lokasi peti kemas pada *Container Loading List*, lakukan langkah 4-5

Langkah 4. Mencari peti kemas pada *Container Loading List* yang memiliki golongan berat dan pelabuhan tujuan yang sama dengan *casebase*.

Langkah 5. Menempatkan peti kemas pada Langkah 4 ke dalam sistem proses sesuai dengan letak peti kemas pada Langkah 2 selama lokasi peti kemas pada kapal masih kosong.

Output: Tataan peti kemas *Container Loading List* yang bersesuaian dengan tatanan peti kemas pada *casebase* berdasarkan golongan dan pelabuhan tujuan peti kemas.

4.1.5.2. Algoritma Adaptasi *Container Loading List*.

Algoritma adaptasi CLL ini berfungsi untuk menyempurnakan hasil dari algoritma similarity. Umumnya, *casebase* dan CLL tidak pernah 100% mirip, sehingga ada golongan peti kemas pada CLL yang jumlahnya lebih banyak atau lebih sedikit dari pada golongan pada *casebase*. Sehingga hasil tatanan dari algoritma similarity tidak sempurna dengan adanya peti kemas yang melayang ataupun peti kemas pada CLL yang belum tertata. Untuk itu, perlu dilakukan algoritma *fill-blank* dan algoritma penambahan sisa peti kemas.

4.1.5.2.1. Algoritma Fill-Blank

Algoritma *fill-blank* ini untuk menyelesaikan permasalahan ketika jumlah peti kemas pada *Container Loading List* untuk golongan dan pelabuhan tujuan yang sama dengan *casebase* lebih sedikit, sehingga terjadi beberapa peti kemas yang melayang. Algoritma *fill-blank* ini akan mengubah lokasi peti kemas yang melayang ke bawah lokasi awal. Berikut ini merupakan algoritma dari fungsi *fill-blank*.

Input: tatanan peti kemas *Container Loading List* yang bersesuaian dengan tatanan peti kemas pada *casebase* berdasarkan golongan dan pelabuhan tujuan peti kemas.

Langkah 1. Untuk seluruh lokasi peti kemas dengan pada sistem proses, lakukan langkah 2-3

Langkah 2. Mencari lokasi peti kemas yang melayang karena ada lokasi peti kemas yang kosong di bawahnya. Pencarian ini dilakukan dari lokasi peti kemas paling atas.

Langkah 3. Mencari lokasi peti kemas yang kosong untuk ditempati peti kemas yang melayang.

Langkah 4. Untuk peti kemas berukuran 20*ft*, lakukan langkah 5 dan untuk peti kemas berukuran 40*ft*, lakukan langkah 6--7

Langkah 5. Pindahkan peti kemas yang melayang pada Langkah 2 ke lokasi peti kemas yang kosong di bawahnya seperti pada Langkah 3

Langkah 6. Karena peti kemas 40*ft* menempati 2 lokasi, maka dilakukan pengecekan bahwa kedua lokasi peti kemas pada Langkah 2 melayang keduanya.

Langkah 7. Memindahkan kedua lokasi peti kemas pada Langkah 6 ke dua lokasi peti kemas yang kosong pada Langkah 3.

Output: Tidak ada susunan peti kemas berukuran 20*ft* yang melayang, namun masih ada susunan peti kemas berukuran 40*ft* yang melayang di salah satu lokasinya apabila salah satu dari dua lokasi peti kemas yang kosong dibawah peti kemas yang melayang telah terisi peti kemas lainnya atau salah satunya tidak kosong.

4.1.5.2.2. Algoritma Penambahan Sisa Peti Kemas

Algoritma penambahan sisa peti kemas ini menyelesaikan permasalahan ketika jumlah peti kemas yang berada di *Container Loading List* lebih banyak dibandingkan pada *casebase*, sehingga terdapat beberapa peti kemas yang belum masuk ke dalam tatanan. Berikut

ini merupakan algoritma dari fungsi penambahan sisa peti kemas

Input: Tatanan peti kemas dari *Container Loading List* yang output dari algoritma *fill-blank* dan peti kemas pada *Container Loading List* yang belum tertata.

Langkah 1. Untuk seluruh peti kemas berukuran 20*ft* pada *Container Loading List,* lakukan Langkah 2, sedangkan untuk peti kemas berukuran 40*ft* pada *Container Loading List,* lakukan Langkah 3-4

Langkah 2. Sisipkan peti kemas berukuran 20*ft* yang belum tertata ke dalam sistem proses sesuai dengan urutan golongan, sehingga peti kemas dengan muatan yang lebih berat akan ditata terlebih dahulu.

Langkah 3. Mencari dua lokasi kosong pada tatanan peti kemas sesuai aturan penataan peti kemas

Langkah 4. Meletakkan peti kemas berukuran 40*ft* ke setiap dua lokasi kosong pada Langkah 3 sesuai dengan urutan golongan, sehingga peti kemas dengan muatan yang lebih berat akan ditata terlebih dahulu

Output: peti kemas pada *Container Loading List* telah tertata semua dan tidak ada susunan peti kemas yang melayang.

4.1.5.3. Algoritma Satisfy

Algoritma *satisfy* berfungsi untuk memindahkan lokasi peti kemas hasil dari tatanan otomatis apabila

terdapat beberapa lokasi peti kemas pada tatanan yang tidak sesuai dengan syarat penataan peti kemas dimana terdapat peti kemas 20 kaki yang berada di atas peti kemas 40 kaki atau penataan peti kemas yang tidak memenuhi syarat "*subyektif"* planner

Input: Hasil tatanan peti kemas yang belum memenuhi syarat penataan peti kemas atau syarat "*subyektif*" dari *planner.*

Langkah 1. Apabila hasil tatanan peti kemas belum memenuhi syarat penataan peti kemas, lakukan Langkah 2-4, namun jika hasil tatanan peti kemas belum memenuhi syarat "*subyektif*" dari pemiliki peti kemas, lakukan langkah 5-6.

Langkah 2. Pindahkan peti kemas 20 kaki yang berada diatas peti kemas 40 kaki

Langkah 3. Mencari lokasi kosong yang tidak berada diatas peti kemas 40 kaki

Langkah 4. Memindahkan peti kemas pada Langkah 2 ke lokasi kosong pada Langkah 3.

Langkah 5. Mencari lokasi peti kemas yang tidak sesuai dengan syarat "*subyektif*" pemilik peti kemas

Langkah 6. Menukar lokasi peti kemas pada Langkah 5 dengan lokasi yang sesuai dengan syarat "*subyektif*" dari *planner*. Langkah ini bisa menyebabkan pertukaran antar dua peti kemas atau bisa juga pertukaran peti kemas pada Langkah 5 dengan lokasi yang kosong.

Output: Hasil tatanan peti kemas yang memenuhi syarat penataan peti kemas dan syarat *"subyektif"* dari *planner*.

4.1.5.4. Algoritma Evaluasi Restow

Algoritma Evaluasi *Restow* berfungsi untuk mengetahui banyak peti kemas yang mengalami *restow* di setiap pelabuhan tempat peti kemas dibongkar atau diturunkan. Berikut ini merupakan algoritma dari fungsi evaluasi *restow*.

Input: Hasil tatanan peti kemas dari pelabuhan sebelumnya yang belum dilakukan bongkar peti kemas di pelabuhan tujuan

Langkah 1.	Untuk seluruh peti kemas hasil dari proses sebelumnya, lakukan langkah 2
Langkah 2.	Mencari peti kemas dengan pelabuhan tujuan adalah pelabuhan yang sedang diproses
Langkah 3.	Menghapus peti kemas yang memiliki pelabuhan tujuan sama dengan pelabuhan yang sedang diproses
Langkah 4.	Mencari peti kemas pada tatanan yang memiliki lokasi kosong di bawah peti kemas yang tersedia. Untuk peti kemas berukuran 20*ft*, lakukan langkah 5-6 dan untuk peti kemas berukuran 40*ft*, lakukan langkah 7-8.
Langkah 5.	Apabila terdapat peti kemas berukuran 20*ft* yang lokasi dibawahnya kosong, maka peti kemas tersebut terkena *restow* dan nilai *restow* bertambahsatu.

Langkah 6. Untuk peti kemas yang terkena *restow* akan dimasukkan ke *Container Loading List* pada pelabuhan yang sedang diproses

Langkah 7. Apabila terdapat peti kemas berukuran 40*ft* dimana menempati dua lokasi pada tatanan peti kemas yang memiliki lokasi yang kosong di salah satu atau kedua lokasi di bawahnya, maka peti kemas tersebut terkena *restow* dan nilai *restow* bertambah satu.

Langkah 8. Untuk peti kemas yang terkena *restow* akan dimasukkan ke *Container Loading List* pada pelabuhan yang sedang diproses

Output: Tatanan peti kemas yang memiliki beberapa lokasi kosong sesuai dengan peti kemas yang mengalami bongkar peti kemas pada pelabuhan tujuan dan penambahan peti kemas pada *Container Loading List* di pelabuhan yang sedang diproses.

4.2. Perancangan Sistem

Pada sub-bab ini, akan diuraikan tentang perancangan sistem yang meliputi perancangan data, perancangan fungsi dalam sistem dan perancangan *interface*.

4.2.1. Perancangan Data

Data yang digunakan dalam program ini terdiri dari data *input* dan data *output.*

1. Data Input

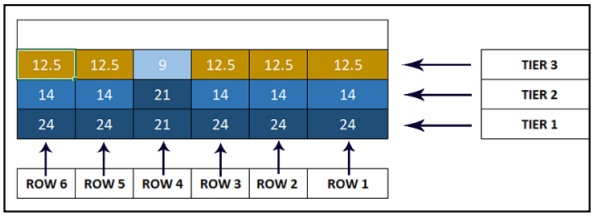
   Data yang digunakan sebagai masukan dalam program dinamakan dengan data input. Data input ini akan diproses dalam sistem supaya menghasilkan output. Selama data input masuk ke dalam sistem, data akan diproses dengan melalui tahap-tahap tertentu supaya output atau hasil proses sesuai dengan yang diinginkan. Dalam kasus ini, *casebase* dari setiap port digunakan sebagai data input dimana *casebase* yang berupa tatanan peti kemas dengan kemiripan tertinggi ini merupakan solusi dari tahap *Retrieve*. Kemiripan *casebase* ini dilihat dari jumlah peti kemas per golongan dari *Container Loading List* pada *casebase* dengan *Container Loading List* yang baru.

2. Data Output

   Data yang dihasilkan dari proses sistem dengan melalui tahap tahap tertentu pada sistem disebut dengan data keluaran atau data output. Dalam kasus ini, hasil tatanan peti kemas di setiap port dalam satu rute dari *Container Loading List* baru yang mirip dengan *casebase* merupakan data output. Hasil tatanan peti kemas dari *Container Loading List* ini telah memenuhi syarat penataan peti kemas pada kapal dimana tidak ada peti kemas yang melayang atau peti kemas berukuran 20*ft* yang berada di atas peti kemas 40*ft*.

4.2.2. Perancangan Struktur Data

KM Kendhaga Nusantara 1 memiliki kapasitas 100 *TEUs (Twenty-foot Equivalent Units)*, dengan konfigurasi 4 *bay* dengan 6 *row* + 3 *tier*, 2 *bay* dengan 6 *row* + 2 *tier* dan 1 *bay* dengan 2 *row* + 2 *tier*. Gambar 4.4 merupakan ilustrasi dari salah satu *bay* yang memiliki 6 *row* + 3 *tier*, dimana warna *background* pada kotak mewakili warna golongan peti kemas, *warna font* pada tulisan merupakan warna pelabuhan tujuan peti kemas dan angka pada tulisan merupakan berat pada setiap peti kemas.



Gambar 4. 4 Ilustrasi Bay dengan 6 Row + 3 Tier.

Pembagian golongan berdasarkan berat dan ukuran peti kemas dibagi menjadi enam golongan. Golongan tersebut dapat dilihat pada tabel dibawah ini.

Tabel 4. 1 Pembagian Golongan Peti Kemas Berdasarkan Berat.

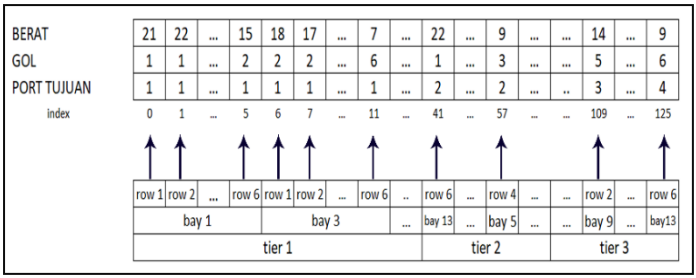
golongan	klasifikasi	keterangan
1	20ft-berat	<10 ton
2	20ft-sedang	10-20 ton
3	20ft-ringan	>20 ton
4	40ft-berat	<10 ton
5	40ft-sedang	10-20 ton
6	40ft-ringan	>20 ton

Sedangkan untuk pembagian pelabuhan tujuan disesuaikan dengan jumlah pelabuhan yang dilalui kapal dalam satu rute perjalanan. Untuk kasus ini, terdapat empat pelabuhan yang dilalui oleh KM. Kendhaga Nusantara 1. Daftar pelabuhan yang dilalui tersebut telah diurutkan sesuai dengan rute perjalanan dalam tabel dibawah ini.

Tabel 4. 2 Pembagian Golongan Peti Kemas Berdasarkan Pelabuhan Tujuan

Golongan	**Pelabuhan Tujuan**
1	Makasar
2	Surabaya
3	Jakarta
4	Belawan

Karena masing-masing peti kemas memiliki 3 atribut yaitu berat (ton), golongan berdasarkan ukuran dan berat peti kemas, dan pelabuhan tujuan peti kemas dikirim, maka satu rangkaian tatanan peti kemas pada KM. Kendhaga Nusantara 1 untuk masing-masing pelabuhan disimpan ke dalam *array of triplet* dua dimensi dengan panjang 126. Dimana dimensi pertama untuk mengakses indeks (lokasi) dan dimensi kedua untuk mengakses berat, golongan berdasarkan berat atau pelabuhan tujuan peti kemas. llustrasi bentuk struktur data dalam satu rangkaian tatanan peti kemas pada KM Kendhaga Nusantara 1 di masing-masing pelabuhan dapat dilihat pada gambar dibawah ini.



Gambar 4. 5 Ilustrasi Struktur Data.

indeks ke-0 hingga 5 (kelipatan 6) merupakan *bay* yang terdiri dari beberapa *row* , dan indek ke-0 hingga 41 (kelipatan 42) merupakan *tier* yang terdiri dari beberapa *bay*.

4.2.3. Perancangan Fungsi Similarity

Fungsi *Similarity* bertujuan untuk mengadaptasi *Container Loading List* agar menjadi tatanan peti kemas yang mirip dengan *casebase*. *Casebase* yang digunakan merupakan solusi yang didapatkan dari tahap *retrieve* pada Metode *Case Based Reasoning*. Setiap pelabuhan memiliki *Container Loading List* dan *casebase* yang berbeda namun saling berkaitan, karena tatanan di pelabuhan pertama mempengaruhi tatanan di pelabuhan berikutnya. Sehingga, dalam satu *casebase* terdapat empat *Container Loading List* dan empat tatanan peti kemas yang berisi *Container Loading List* dan tatanan peti kemas di masing-masing pelabuhan.

Fungsi *similarity* memiliki dua input, yaitu *Container Loading List* dan *casebase* dengan data yang berbeda-beda di setiap pelabuhan namun berkaitan. Kedua data tersebut tersedia dalam bentuk *array of triplet* dua

dimensi dengan panjang 126, sehingga tidak perlu dilakukan proses input secara satu per satu. Untuk menjalankan proses ini diperlukan sebuah tempat penyimpanan, sehingga digunakan *array of triplet* dua dimensi dengan panjang 126 sebagai tempat proses dan penyimpanan hasil.

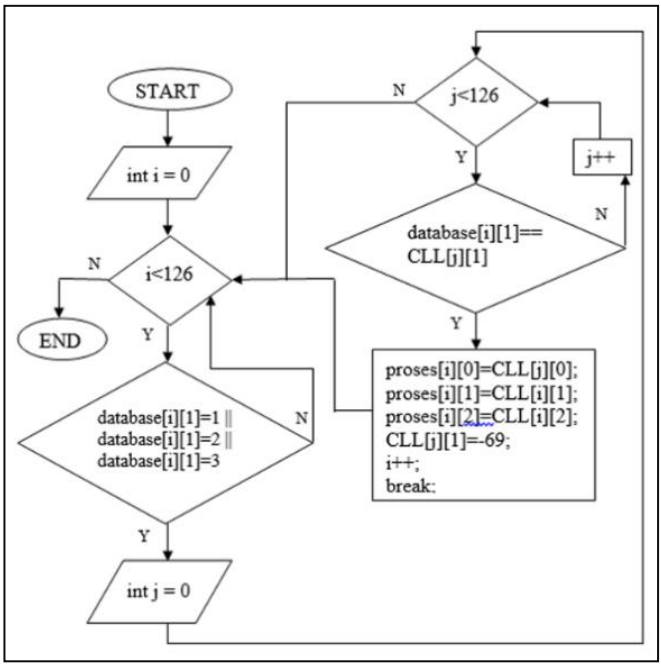
Output fungsi ini adalah didapatkan hasil tatanan peti kemas dari *Container Loading List* yang mirip dengan *casebase* untuk golongan berat peti kemas dan pelabuhan tujuan peti kemas. Hasil tatanan ini disimpan dalam *array* proses. Namun, fungsi *similarity* masih memiliki kekurangan yang dapat dilihat dari *output* yang dihasilkan. Masih adanya peti kemas yang melayang dalam hasil tatanan dan jumlah peti kemas yang telah tertata tidak sesuai dengan jumlah peti kemas pada *Container Loading List* merupakan kekurangan fungsi ini yang perlu diperbaiki dengan melalui tahap-tahap pada sistem berikutnya.

Pada fungsi similarity ini terdapat dua bagian, yaitu bagian untuk penataan peti kemas berukuran 20*ft* dan bagian untuk penataan peti kemas yang berukuran 40*ft*.

4.2.4.1. Perancangan Modul Kesamaan Peti Kemas 20*ft*

Penataan peti kemas berukuran 20*ft* ini memiliki 2 *pointer* untuk melakukan pencarian, yaitu *pointer i* dan *j*. *Pointer i* digunakan untuk melakukan pencarian peti kemas pada *array casebase* dan *pointer j* digunakan untuk melakukan pencarian peti kemas pada *array Container Loading List*. *Pointer i* mengakses golongan peti kemas yang berukuran 20*ft* dan menyimpan data pelabuhan tujuan

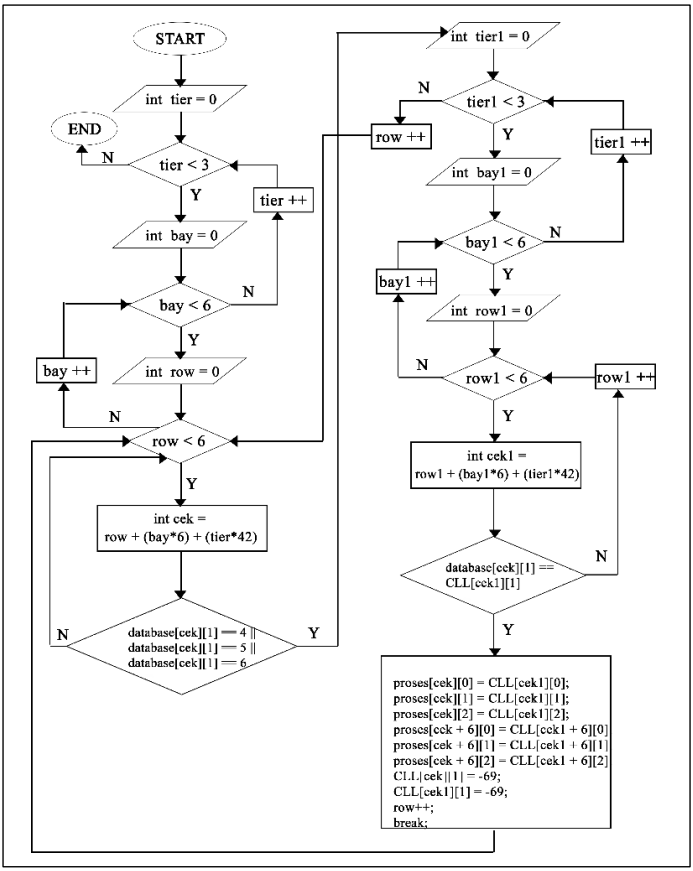
peti kemas pada *casebase* serta menyimpan lokasi dari peti kemas yang tertata di *casebase*. *Pointer j* akan mengakses golongan peti kemas berukuran 20*ft* dan menyimpan data pelabuhan tujuan peti kemas yang tersedia di *Container Loading List* baru. Selain itu, *pointer j* juga mencari peti kemas pada *Container Loading List* yang memiliki golongan dan pelabuhan tujuan yang sama dengan peti kemas yang terdapat di *casebase* untuk dilakukan penataan peti kemas pada *array* proses dengan lokasi yang sama dengan lokasi peti kemas pada *casebase*. Gambar berikut merupakan diagram alir untuk peti kemas berukuran 20*ft*.



Gambar 4. 6 Diagram Alir *Similarity* Peti Kemas 20*ft*

4.2.4.2. Perancangan Modul Kesamaan Peti Kemas 40*ft*

Penataan peti kemas berukuran 40*ft* ini memiliki tahapan yang hampir sama dengan cara penataan pada peti kemas berukuran 20*ft*. Namun karena peti kemas berukuran 40*ft* memiliki panjang dua kali dari peti kemas 20*ft*, maka lokasi penataannya menempati dua *bay* sekaligus. Peti kemas 40*ft* memiliki perlakukan khusus dimana peti kemas ukuran ini tidak dapat berada di bawah peti kemas 20*ft*. Perlakukan khusus inilah yang menyebabkan adanya perbedaan penataan untuk peti kemas ukuran 40*ft* pada *array* proses. Diagram alir proses penempatan peti kemas berukuran 40*ft* pada fungsi *similarity* dapat dilihat pada gambar di bawah ini.



Gambar 4.7 Diagram Alir *Similarity* Peti Kemas 40*ft*

4.2.4. Perancangan Fungsi Adaptasi *Container Loading List*

Fungsi Adaptasi *Container Loading List* ini digunakan untuk melengkapi hasil tatanan pada tahap sebelumnya supaya semua peti kemas pada *Container Loading List* dapat tertata secara keseluruhan dan tidak ada

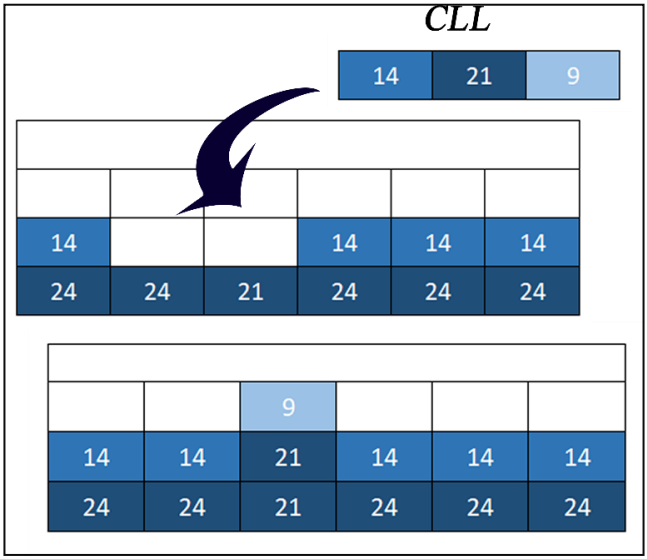
peti kemas yang melayang pada tatanan peti kemas di *array* proses.

Input pada fungsi ini adalah *array* proses dan *array Container Loading List*. *Array* proses digunakan untuk mencari lokasi yang masih kosong sedangkan *array Container Loading List* digunakan untuk mencari peti kemas yang masih belum tertata pada *array* proses. Terdapat dua kondisi yang tersedia sebelum dilakukan fungsi Adaptasi *Container Loading List*, yaitu adanya peti kemas pada *array* proses yang melayang dan adanya peti kemas pada *Container Loading List* yang belum tertata di *array* proses.

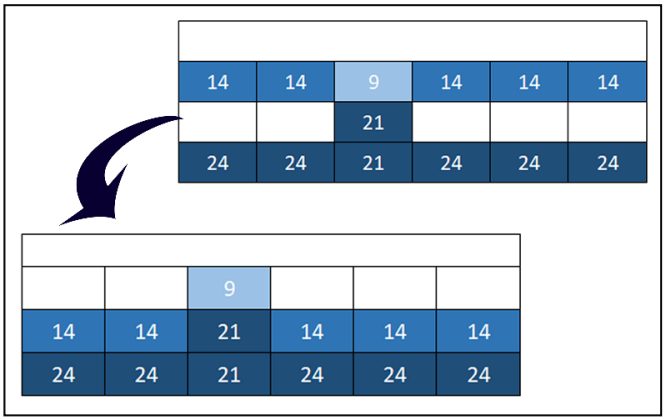
Untuk peti kemas yang melayang pada tatanan di *array* proses, akan dilakukan fungsi *fill-blank*. Kondisi peti kemas melayang ini dikarenakan jumlah golongan berat tertentu pada *Container Loading List* lebih sedikit dibandingkkan jumlah golongan berat yang sama pada *casebase*. Terdapat perbedaan proses pada peti kemas yang berukuran *20ft* dengan peti kemas yang berukuran *40ft*.

Untuk peti kemas pada *Container Loading List* yang belum tertata pada *array* proses, akan dilakukan fungsi penambahan sisa peti kemas. Kondisi ini terjadi karena jumlah golongan berat tertentu pada *Container Loading List* lebih banyak dibandingkkan jumlah golongan berat yang sama pada *casebase*. Masing-masing fungsi memiliki perlakukan yang berbeda untuk peti kemas berukuran 20*ft* dan peti kemas berukuran 40*ft*. Setelah melalui tahapan-tahapan ini, *output* dari fungsi adapatasi *Container Loading List* adalah tidak adanya peti kemas yang melayang pada *array* proses dan semua peti kemas

pada *array Container Loading List* telah tertata secara keseluruhan di *array* proses.



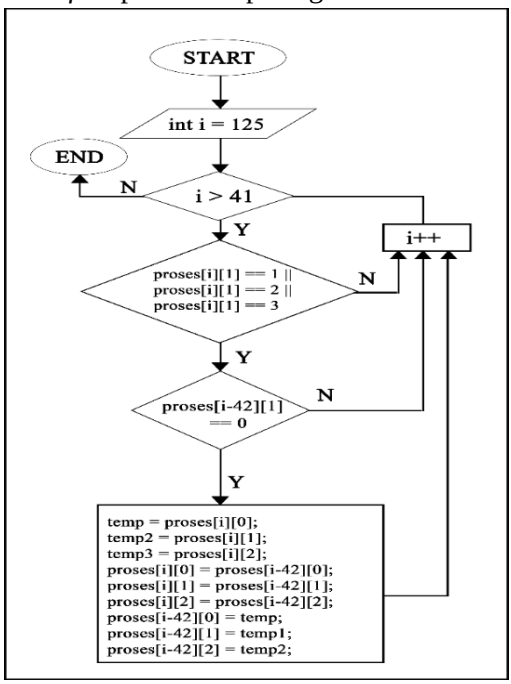
Gambar 4.8 Ilustrasi Fungsi Penambahan Sisa Peti Kemas



Gambar 4.9 Ilustrasi Fungsi Fill-Blank.

4.2.5.1. Perancangan Fungsi *Fill-Blank* Peti Kemas20*ft*

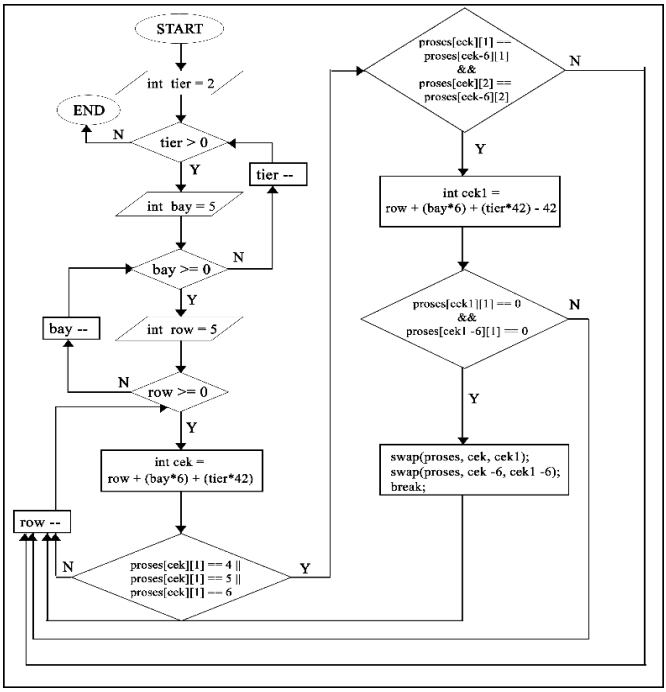
Pada penataan peti kemas berukuran 20*ft* ini, digunakan satu *pointer* saja, yaitu *pointer i*. *Pointer i* ini berfungsi untuk mencari peti kemas pada *array* proses yang memiliki lokasi kosong dibawahnya. Pencarian dilakukan dari peti kemas yang memiliki lokasi paling atas. Apabila ditemukan peti kemas yang melayang atau memiliki lokasi kosong dibawahnya, maka akan dilakukan penukaran lokasi supaya peti kemas pada *pointer i* berpindah ke lokasi kosong yang berada di bawahnya. Diagram alir dari fungsi *fill blank* untuk peti kemas berukuran 20*ft* dapat dilihat pada gambar dibawah ini



Gambar 4.10 Diagram Alir *Fill-Blank* Peti Kemas 20 *ft*

4.2.5.2. Perancangan Fungsi *Fill-Blank* Peti Kemas 40*ft*

Sama halnya dengan tahapan fungsi *fill-blank* peti kemas 20*ft*, fungsi *fill-blank* untuk peti kemas 40*ft* ini juga mencari peti kemas yang memiliki lokasi kosong dibawahnya dengan pencarian dilakukan dari peti kemas yang memiliki lokasi paling atas atau dalam *array* proses terletak di bagian paling belakang. Namun, karena peti kemas berukuran 40*ft* menempati 2 lokasi sekaligus, maka dilakukan pengecekan lokasi kosong di bawah kedua lokasi peti kemas tersebut. Apabila hanya salah satu yang memiliki lokasi kosong dibawah peti kemas tersebut, maka tidak dapat dilakukan pemindahan lokasi peti kemas. Jadi, syarat utama untuk penukaran atau pemindahan lokasi peti kemas supaya tidak melayang adalah kedua lokasi dibawah lokasi peti kemas berada pada *array* proses haruslah kosong keduanya. Diagram alir pada fungsi ini dapat dilihat pada gambar dibawah ini

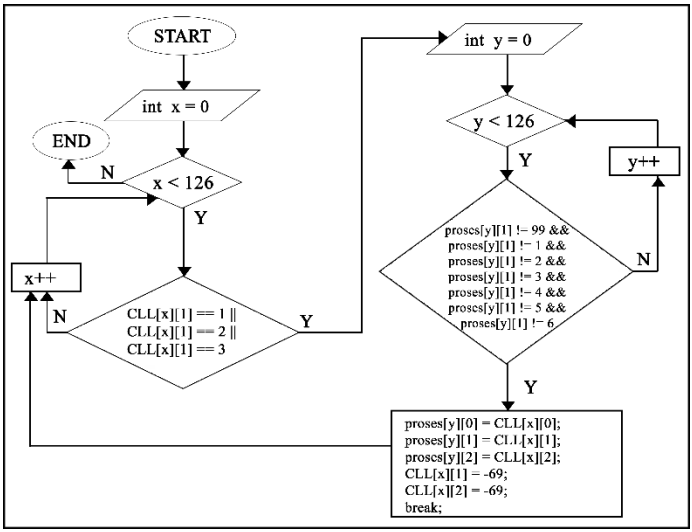


Gambar 4. 11 Fungsi *Fill Blank* Kemas 4*0ft*.

4.2.5.3. Perancangan Fungsi Penambahan Sisa Peti Kemas 20*ft*

Pada fungsi ini digunakan dua *pointer*, yaitu *pointer x* dan *pointer y*. *Pointer x* digunakan untuk mencari peti kemas berukuran 20*ft* yang terdapat pada *Container Loading List. Pointer y* digunakan untuk mencari lokasi yang masih kosong pada *array* proses. Pencarian lokasi kosong ini dilakukan mulai dari *indeks* terkecil pada *array* proses. Peti kemas pada *pointer x* akan mengisi lokasi kosong yang ditunjuk oleh *pointer j*. Penambahan peti kemas ini dilakukan mulai dari golongan peti kemas

terberat ke terkecil. Gambar dibawah ini menunjukkan diagram alir untuk proses penambahan sisa peti kemas 20*ft*.

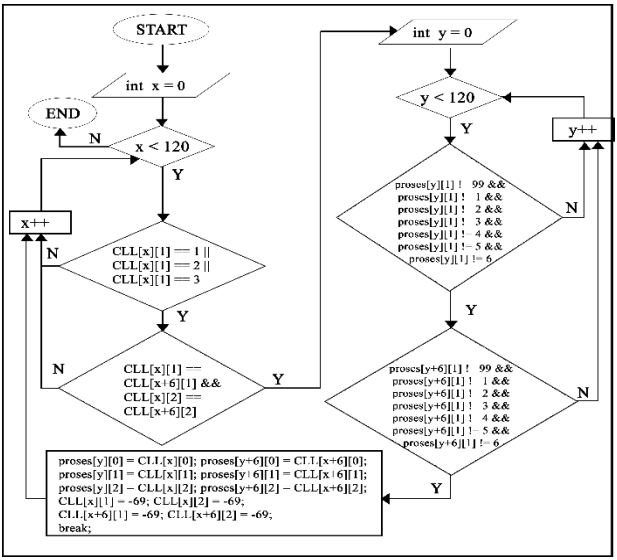


Gambar 4.12 Fungsi Penambahan Sisa Peti Kemas 20*ft*.

4.2.5.4. Perancangan Fungsi Penambahan Sisa Peti Kemas 40*ft*

Penambahan sisa peti kemas 40*ft* juga menggunakan dua *pointer*, dimana *pointer x* berfungsi untuk mencari peti kemas yang berukuran 40*ft* pada *Container Loading List*, sedangkan *pointer y* mencari 2 lokasi kosong pada *array* proses yang bersesuain untuk dapat ditempati oleh peti kemas pada *pointer x*. Pencairan lokasi kosong ini dilakukan dari *array* proses dengan *indeks* terkecil. Lokasi 2 *bay* yang bersesuain ini harus sama sama kosong untuk dapat terisi peti kemas baru. Penambahan peti kemas berukuran 40*ft* ini dilakukan dari

golongan peti kemas yang terberat terlabih dulu. Gambar dibawah ini menunjukkan diagram alir untuk proses penambahan sisa peti kemas 40*ft*.



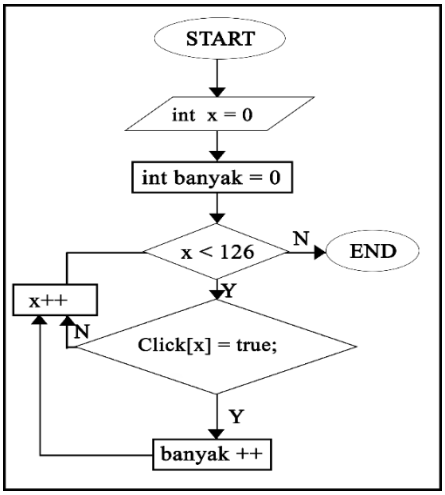
Gambar 4.13 Fungsi Penambahan Sisa Peti Kemas 40*ft*.

4.2.5. Perancangan Fungsi *Satisfy*

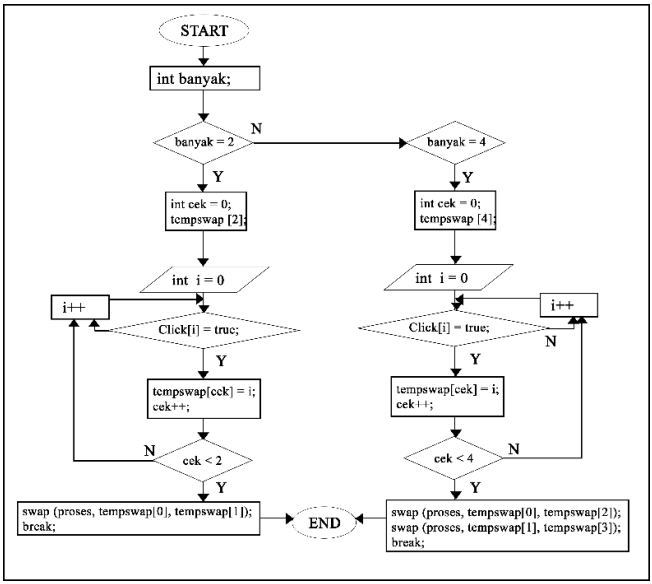
Fungsi *Satisfy* ini digunakan untuk menghasilkan tatanan peti kemas pada *array* proses yang memenuhi syarat penataan peti kemas dan syarat "*subyektif*" yang dilakukan oleh *planner*. Pada fungsi ini, hal paling utama yang dilakukan adalah pertukaran peti kemas. Pada pertukaran peti kemas ini dibutuhkan satu *pointer* untuk mencari lokasi peti kemas yang akan ditukar dan *array* baru yang digunakan sebagai tempat penyimpanan sementara peti kemas yang terpilih supaya dapat dilakukan penukaran data lokasi peti kemas. Terdapat tiga kondisi

pertukaran peti kemas, yaitu pertukaran antar dua peti kemas berukuran 20*ft*, pertukaran antar dua peti kemas berukuran 40*ft* dan pertukaran antar dua peti kemas berukuran 20*ft* dengan satu peti kemas berukuran 40*ft*.

Untuk dua peti kemas berukuran 20*ft*, *pointer* mencari dua lokasi peti kemas yang akan ditukar. Setiap *pointer* menemukan satu peti kemas yang akan dilakukan penukaran, peti kemas pada *pointer* akan disimpan di *array* penyimpanan sementara dan terletak di *indeks* paling awal, kemudian untuk peti kemas yang ditemukan *pointer* selanjutnya akan masuk ke *array* penyimapan sementara dengan mengisi *indeks* berikutnya. Kemudian kedua *indeks* ini akan menukar data lokasi peti kemas pada *array* proses. Hal ini juga dilakukan untuk menukar peti kemas yang berukuran 40*ft* dengan peti kemas yang berukuran sama atau dengan dua peti kemas berukuran 20*ft*. Namun, perbedaannya hanya terletak pada jumlah peti kemas yang harus terpilih dan panjang *array* pada penyimpanan sementara. Untuk penukaran peti kemas berukuran 40*ft* perlu dilakukan pemilihan empat lokasi yang bersesuaian. Apabila peti kemas yang ditunjuk *pointer* tidak memiliki dua lokasi yang bersesuai untuk peti kemas 40*ft*, maka proses penukaran tidak dapat dilakukan. Gambar dibawah ini menjelaskan diagram alir dari fungsi *satisfy* dan fungsi penukaran peti kemas yang terjadi selama tahapan ini.



Gambar 4. 14 Proses Pertama Fungsi Satisfy.



Gambar 4. 15 Proses Swapping Peti Kemas

4.2.6. Perancangan Fungsi Evaluasi *Restow*

Fungsi evaluasi *restow* digunakan untuk mendapatkan jumlah peti kemas yang terkena *restow* di pelabuhan tujuan. *Input* pada fungsi ini adalah *array* proses yang merupakan *output* dari penataan pelabuhan sebelumnya. Sebelum menghitung nilai *restow,* dilakukan pencarian data pelabuhan tujuan pada peti kemas di *array* proses yang menyimpan angka yang mewakili pelabuhan pada proses ini. Peti kemas yang memiliki data pelabuhan tujuan yang sama dengan pelabuhan yang sedang proses, akan dihapus dari *array* proses. *Output* yang dihasilkan dari fungsi ini adalah jumlah peti kemas yang terkena *restow* dan memindahkan peti kemas yang mengalami *restow* ke *Container Loading List* di pelabuhan tersebut.
Fungsi ini hanya menggunakan satu *pointer* yang berfungsi untuk mencari peti kemas yang melayang pada *array* proses setelah dilakukan pembongkaran peti kemas di pelabuhan. *Pointer* akan mengecek lokasi dibawah peti kemas yang ditunjuk *pointer* untuk menentukan apakah peti kemas melayang atau tidak. Pencarian ini dilakukan mulai dari *indeks array* proses paling belakang. Jika terdapat lokasi kosong di bawah peti kemas yang ditunjuk *pointer,* maka nilai *restow* akan bertambah satu. Untuk peti kemas berukuran 40*ft* yang menempati dua lokasi pada *array* proses, nilai *restow* bertambah satu jika terdapat lokasi kosong dibawah salah satu atau kedua lokasi peti kemas berukuran 40*ft.*

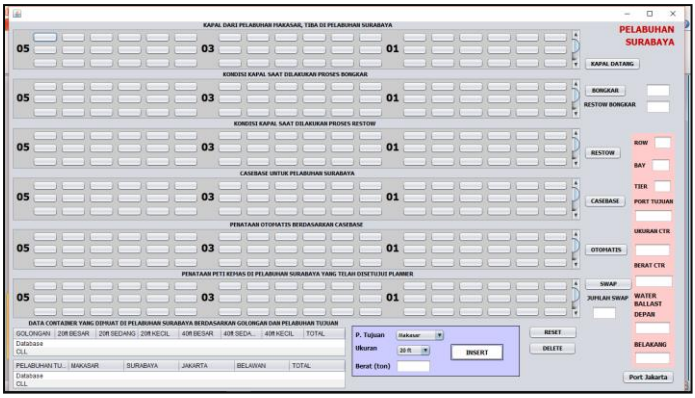
4.2.7. Perancangan *Interface*

Desain *interface* sistem dibutuhkan untuk mempermudah pengguna dalam mengoperasikan perangkat lunak yang dibangun

4.2.8.1. Desain *Interface* Pelabuhan dengan Penataan Otomatis

Pada Gambar 4.16 menujukkan desain *interface* untuk pelabuhan pertama untuk penataan otomatis dari perangkat lunak yang dibangun. Dari *interface* tersebut terdapat:

1. Ilustrasi tampilan tataan peti kemas dari *Casebase*, *Container Loading List*, kedatangan kapal, dan proses bongkar serta *restow* yang terbentuk dari *Toggle Button*.
2. Tombol 'Kapal Datang', 'Bongkar', 'Restow', 'Casebase', 'Otomatis', 'Swap', 'Insert', 'Delete' dan 'Reset' yang terbentuk dari *Button.*
3. Tombol 'Port Selanjutnya' yang terbentuk dari *Button* menunjukkan pelabuhan yang akan dituju setelah pelabuhan awal diproses
4. Informasi letak dan rincian data peti kemas yang terdiri dari '*Row*', '*Bay*', '*Tier*', 'Ukuran', 'Berat' dan 'Port Tujuan' yang terbentuk dari *Text Filed*.
5. Tabel 'Data Container', 'Data Pelabuhan Tujuan' yang terbentuk dari *Table*.

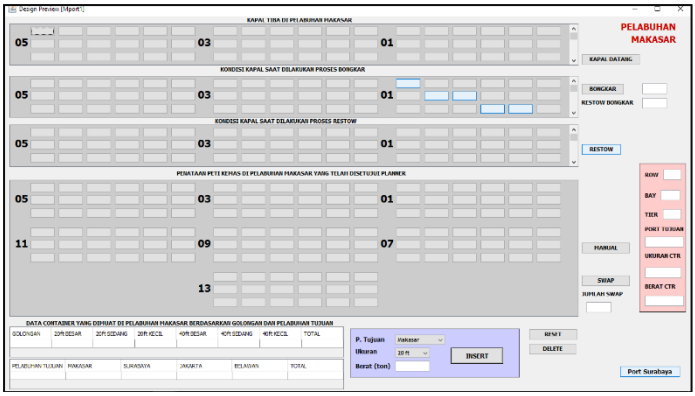


Gambar 4. 16 Perancangan *Interface* Pelabuhan dengan Penataan Otomatis

4.2.8.2. Desain *Interface* Penataan Manual

. Pada Gambar 4.17 menujukkan desain *interface* penataan peti kemas secara manual dari perangkat lunak yang dibangun. Dari *interface* tersebut terdapat:

1. Ilustrasi tampilan tataan peti kemas dari *Container Loading List*, kedatangan kapal, dan proses bongkar serta *restow* yang terbentuk dari *Toggle Button*.
2. Tombol 'Manual', 'Swap', 'Insert', 'Delete' dan 'Reset' yang terbentuk dari *Button.*
3. Tombol 'Port Selanjutnya' yang terbentuk dari *Button* menunjukkan pelabuhan yang akan dituju setelah pelabuhan awal diproses
4. Informasi letak dan rincian data peti kemas yang terdiri dari '*Row*', '*Bay*', '*Tier*', 'Ukuran', 'Berat' dan 'Port Tujuan' yang terbentuk dari *Text Filed*.
5. Tabel 'Data Container', 'Data Pelabuhan Tujuan' yang terbentuk dari *Table*.

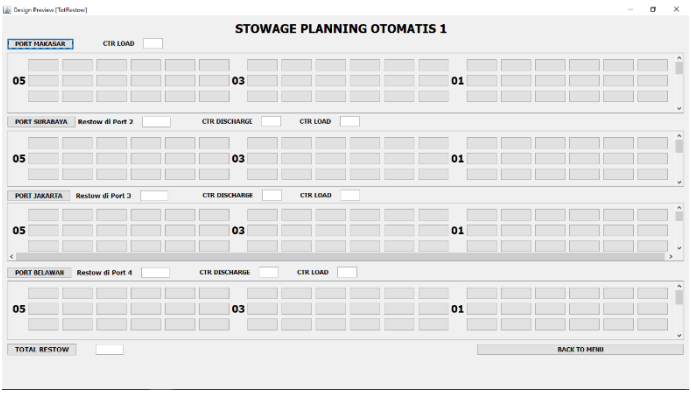


Gambar 4. 17 Perancangan *Interface* Penataan Manual

4.2.8.3. Desain Interface Output

. Pada Gambar 4.18 menujukkan desain *interface output* tatanan peti kemas untuk semua pelabuhan dari perangkat lunak yang dibangun. Dari *interface* tersebut terdapat:

1. Ilustrasi tampilan empat tatanan peti kemas di masing-masing pelabuhan yang terbentuk dari *Toggle Button*.
2. Tombol 'Port Makasar', 'Port Surabaya', 'Port Jakarta', 'Port belawan' dan 'Total Restow' yang terbentuk dari *Button*.
4. Informasi jumlah *restow* di masing-masing pelabuhan yang terdiri dari '*Restow di port 2*', '*Restow di port 3'*, '*Restow di port 4*' dan '*Total Restow'* yang terbentuk dari *Text Field*.
5. Informasi jumlah peti kemas yang terbongkar dan termuat di setiap pelabuhan yang terbentuk dari *Text Field.*



Gambar 4.*18* Perancangan *Interface Output*

4.3. Implementasi Sistem

Dari hasil perancangan sistem akan diimplementasikan pada program untuk membangun perangkat lunak

4.3.1. Implementasi Struktur Data

Pada sistem ini digunaka struktur data berupa *array of triplet* dua dimensi dengan panjang 126. Dimensi pertama berfungsi untuk menyimpan lokasi peti kemas, sedangkan dimensi kedua berfungsi untuk memyimpan data peti kemas, seperti berat peti kemas, golongan berdasarkan berat peti kemas dan pelabuhan tujuan peti kemas dikirim. Berikut merupakan *source code* untuk *array of triplet* dua dimensi dengan panjang 126.

```
double [][] database = new double[126][3];
```

atau dapat dituliskan dengan database[i][j], dimana i merupakan indeks untuk mengakses lokasi peti kemas, dimulai dari indeks ke-0 hingga ke indeks ke-125, sedangakan digunakan untuk mengakses data peti kemas,

untuk j =0 adalah data berat peti kemas, j=1 untuk data golongan berdasarkan berat peti kemas dan j=2 untuk menyimpan data pelabuhan tujuan peti kemas.

4.3.2. Implementasi Fungsi Similarity

Fungsi *similarity* ini bertujuan untuk mengadapatasi peti kemas pada *Container Loading List* yang memiliki golongan berdasarkan berat peti kemas dan pelabuhan tujuan sama dengan peti kemas pada *casebase* untuk menempati lokasi yang sama pada tatana peti kemas. *Casebase* yang digunakan merupakan solusi yang didapatkan pada tahap *Retrieve* Metode *Case Based Reasoning*.

4.3.2.1. Implementasi Fungsi *Similarity* Peti Kemas Ukuran 20*ft*

Fungsi *similarity* untuk peti kemas berukuran 20*ft* ini menghasilkan *output* penataan peti kemas berukuran 20*ft* dari *Container Loading List* dengan lokasi yang sama dengan peti kemas dari *casebase* untuk golongan berat dan pelabuhan tujuan yang sama. Hasil penataan ini disimpan di *array* proses. Berikut ini *source code* dari implementasi fungsi *similarity* peti kemas ukuran 20*ft*.

```
Public void adaptasi20kaki(double[][] proses){
  for(int i =0; i <= 125; i++){
    if (database[i][1] == 1 || database[i][1] == 2
    || database[i][1] == 3) {
      for (int j = 0; j <= 125; j++) {
        if (database[i][1] ==  CLL[j][1] &&
        database[i][2] == CLL[j][2] ) {
          if(proses[i][1] == 0 || proses[i][1] ==
          69){
             proses[i][0] = CLL[j][0];
             proses[i][1] = CLL[j][1];
             proses[i][2] = CLL[j][2];
```

```
                    CLL[j][1] = -69;
                    CLL[j][2] = -69;
                    break;
                    }
                }
            }
        }
    }
```

4.3.2.2. Implementasi Fungsi *Similarity* Peti Kemas Ukuran 40*ft*

Fungsi *similarity* untuk peti kemas berukuran 40*ft* ini menghasilkan *output* penataan peti kemas berukuran 40*ft* dari *Container Loading List* dengan lokasi yang sama dengan peti kemas dari *casebase* untuk golongan berat dan pelabuhan tujuan yang sama. Hasil penataan ini disimpan di *array* proses. Berikut ini *source code* dari implementasi fungsi *similarity* peti kemas ukuran 40*ft*.

```
public void adaptasi40kaki(double[][] proses) {
for (int tier = 0; tier <= 2; tier++) {
  for (int bay = 0; bay <= 5; bay++) {
    for (int row = 0; row <= 5; row++) {
      if (database[row + (bay * 6) + (tier * 42)][1]
      == 4 || database[row + (bay * 6) + (tier *
      42)][1] == 5 || database[row + (bay * 6) +
      (tier * 42)][1] == 6) {
      here:
        for (int tier1 = 0; tier1 <= 2; tier1++) {
          for (int bay1 = 0; bay1 <= 5; bay1++) {
            for (int row1 = 0; row1 <= 5; row1++) {
              if (database[row + (bay * 6) + (tier *
              42)][1] == CLL[row1 + (bay1 * 6) +
              (tier1 * 42)][1] && database[row +
              (bay * 6) + (tier * 42)][2] ==
              CLL[row1 + (bay1 * 6) + (tier1 *
              42)][2]) {
                 if(proses[row + (bay * 6) + (tier
                 * 42)][1] == 0 && proses[row +
                 (bay * 6) + (tier * 42)+6][1] == 0
                 || proses[row + (bay * 6) + (tier
```

```
                * 42)][1] == 69 && proses[row +
                (bay * 6) + (tier * 42)+6][1] == 69){
                    proses[row + (bay * 6) + (tier *
                    42)][0] = CLL[row1 + (bay1 * 6) +
                    (tier1 * 42)][0];
                    proses[row + (bay * 6) + (tier *
                    42)][1] = CLL[row1 + (bay1 * 6) +
                    (tier1 * 42)][1];
                    proses[row + (bay * 6) + (tier *
                    42)][2] = CLL[row1 + (bay1 * 6) +
                    (tier1 * 42)][2];
                    proses[row + (bay * 6) + (tier *
                    42) + 6][0] = CLL[row1 + (bay1 *
                    6) + (tier1 * 42) + 6][0];
                    proses[row + (bay * 6) + (tier *
                    42) + 6][1] = CLL[row1 + (bay1 *
                    6) + (tier1 * 42) + 6][1];
                    proses[row + (bay * 6) + (tier *
                    42) + 6][2] = CLL[row1 + (bay1 *
                    6) + (tier1 * 42) + 6][2];
                    CLL[row1 + (bay1 * 6) + (tier1 *
                    42)][1] = -69;
                    CLL[row1 + (bay1 * 6) + (tier1 *
                    42)][2] = -69;
                    CLL[row1 + (bay1 * 6) + (tier1 *
                    42) + 6][1] = -69;
                    CLL[row1 + (bay1 * 6) + (tier1 *
                    42) + 6][2] = -69;
                    break here;
                    }
                }
            }
        }
    }
}
```

4.3.3. Implementasi Fungsi Adaptasi *Container Loading List*

Fungsi Adaptasi *Container Loading List* ini digunakan untuk melengkapi hasil tatanan pada tahap sebelumnya supaya semua peti kemas pada *Container*

Loading List dapat tertata secara keseluruhan dan tidak ada peti kemas yang melayang pada tatanan peti kemas di *array* proses.

4.3.4.1. Implementasi *Fill-Blank* Peti Kemas 20*ft*

Fungsi ini menghasilkan *output* dimana tidak ada peti kemas berukuran 20*ft* yang melayang di *array* proses. Berikut *source code* dari implementasi fungsi *fill-blank* peti kemas 20*ft*.

```
public void fillblank20kaki(double[][] array) {
 for (int i=125; i>=84; i--){
   if (array[i][1] == 1 || array[i][1] == 2 ||
   array[i][1] == 3){
    if(array[i - 42][1] != 1 && array[i - 42][1] != 2 &&
    array[i - 42][1] != 3 && array[i - 42][1] != 4 &&
    array[i - 42][1] != 5 && array[i - 42][1] != 6 &&
    array[i - 42][1] != 99){
       swap(array, i, i-42);
       }
    else if(array[i-84][1] != 1 && array[i-84][1] != 2
    && array[i-84][1] != 3 && array[i-84][1] != 4 &&
    array[i-84][1] != 5 && array[i-84][1] != 6 &&
    array[i-84][1] != 99){
       swap(array, i, i-84);
       }
    }
 }
 for (int i=83; i>=42; i--){
   if (array[i][1] == 1 || array[i][1] == 2 ||
   array[i][1] == 3){
     if(array[i - 42][1] != 1 && array[i - 42][1] != 2
     && array[i - 42][1] != 3 && array[i - 42][1] != 4
     && array[i - 42][1] != 5 && array[i - 42][1] != 6
     && array[i - 42][1] != 99){
       swap(array, i, i-42);
       }
    }
 }
```

Pada fungsi *Fill-Blank* peti kemas 20*ft* ini terdapat satu fungsi yang dipanggil, yaitu `swap(double[][] proses, int i, int j)`. Fungsi ini bertujuan untuk menukar letak peti kemas pada suatu *array of triplet*. Fungsi ini memiliki tiga parameter, yaitu *array of triplet* dua dimensi, `i` dengan tipe data *integer* dan `j` dengan tipe data *integer*. Dimana `i` dan `j` merupakan letak (indeks) dari peti kemas tersebut. Fungsi ini memiliki *output* penukaran letak peti kemas antara indeks ke-`i` dan indeks ke-`j` dari suatu *array of triplet* dua dimensi. Berikut ini merupakan *source code* dari fungsi `swap(double[][] proses, int i, int j)`.

```
public void swap(double[][] proses, int i, int j) {
  for (int k = 0; k < 3; k++) {
    double temp = proses[i][k];
    proses[i][k] = proses[j][k];
    proses[j][k] = temp;
  }
}
```

4.3.4.2. Implementasi *Fill-Blank* Peti Kemas 40*ft*

Fungsi ini menghasilkan *output* dimana terdapat perpindahan lokasi peti kemas berukuran 40*ft* yang melayang di *array* proses apabila kedua lokasi dibawah peti kemas kosong. Sehingga, fungsi ini tidak memindahkan seluruh peti kemas 40*ft* yang melayang apabila syarat lokasi dibawah peti kemas tidak terpenuhi. Berikut *source code* dari implementasi fungsi *fill-blank* peti kemas 40*ft*.

```
public void fillblank40kaki(double[][] array){
  for (int tier = 2; tier >=1; tier--) {
    for (int bay = 5; bay >= 0; bay--) {
      for (int row = 5; row >= 0; row--) {
        int cek = row +(bay*6) +(tier*42);
        if (array[cek][1] == 4 && array[cek + 6][1]
```

```
          == 4 ){
            if(array[cek][2] == array[cek + 6][2] ){
              if(array[cek - 42][1] != 1 &&
              array[cek + 6 - 42][1] != 1 &&
              array[cek - 42][1] != 2 && array[cek +
              6 - 42][1] != 2 && array[cek - 42][1]
              != 3 && array[cek + 6 - 42][1] != 3 &&
              array[cek - 42][1] != 4 && array[cek +
              6 - 42][1] != 4 && array[cek - 42][1]
              != 5 && array[cek + 6 - 42][1] != 5 &&
              array[cek - 42][1] != 6 && array[cek +
              6 - 42][1] != 6 && array[cek - 42][1]
              != 99 && array[cek + 6 - 42][1] !=
              99){
                 swap(array, cek, cek - 42 );
                 swap(array, cek+ 6 , cek + 6 - 42);
                 }
            }
         }
         else if (array[cek][1] == 5 && array[cek +
         6][1] == 5 ){
            if(array[cek][2] == array[cek + 6][2] ){
               if(array[cek - 42][1] != 1 &&
               array[cek + 6 - 42][1] != 1 &&
               array[cek - 42][1] != 2 && array[cek
               + 6 - 42][1] != 2 && array[cek -
               42][1] != 3 && array[cek + 6 - 42][1]
               != 3 && array[cek - 42][1] != 4 &&
               array[cek + 6 - 42][1] != 4 &&
               array[cek - 42][1] != 5 && array[cek
               + 6 - 42][1] != 5 && array[cek -
               42][1] != 6 && array[cek + 6 - 42][1]
               != 6 && array[cek - 42][1] != 99 &&
               array[cek + 6 - 42][1] != 99){
                 swap(array, cek, cek - 42 );
                 swap(array, cek+ 6 , cek + 6 - 42);
                 }
            }
         }
         else if (array[cek][1] == 6 && array[cek +
         6][1] == 6 ){
            if(array[cek][2] == array[cek + 6][2] ){
               if(array[cek - 42][1] != 1 &&
               array[cek + 6 - 42][1] != 1 &&
               array[cek - 42][1] != 2 && array[cek
               + 6 - 42][1] != 2 && array[cek -
               42][1] != 3 && array[cek + 6 - 42][1]
               != 3 && array[cek - 42][1] != 4 &&
               array[cek + 6 - 42][1] != 4 &&
```

```
                array[cek - 42][1] != 5 && array[cek
                + 6 - 42][1] != 5 && array[cek -
                42][1] != 6 && array[cek + 6 -
                42][1] != 6 && array[cek - 42][1] !=
                99 && array[cek + 6 - 42][1] != 99){
                 swap(array, cek, cek - 42 );
                 swap(array, cek+ 6 , cek + 6 - 42);
                 }
               }
            }
         }
      }
   }

}
```

Pada fungsi *Fill-Blank* peti kemas 40*ft* ini terdapat satu fungsi yang dipanggil, yaitu `swap(double[][] proses, int i, int j)`. Fungsi ini bertujuan untuk menukar letak peti kemas pada suatu *array of triplet*. Fungsi ini memiliki tiga parameter, yaitu *array of triplet* dua dimensi, `i` dengan tipe data *integer* dan `j` dengan tipe data *integer*. Dimana `i` dan `j` merupakan letak (indeks) dari peti kemas tersebut. Fungsi ini memiliki *output* penukaran letak peti kemas antara indeks ke-`i` dan indeks ke-`j` dari suatu *array of triplet* dua dimensi. Berikut ini merupakan *source code* dari fungsi `swap(double[][] proses, int i, int j)`.

```
public void swap(double[][] proses, int i, int j) {
  for (int k = 0; k < 3; k++) {
    double temp = proses[i][k];
    proses[i][k] = proses[j][k];
    proses[j][k] = temp;
  }
}
```

4.3.4.3. Implementasi Fungsi Penambahan Sisa Peti Kemas 20*ft*

Fungsi ini menghasilkan *output* dimana seluruh peti kemas berukuran 20*ft* pada *Container Loading List* tertata pada *array* proses dan mengisi lokasi yang kosong pada *array* proses, sehingga tidak ada peti kemas yang melayang pada *array* proses. Karena penambahan sisa peti kemas dilakukan dari indeks *array* proses terkecil, maka peti kemas yang tertata diurutkan dari golongan yang terberat. Berikut *source code* untuk implemnetasi fungsi penambahan sisa peti kemas 20*ft*.

```
public void tambahsisa20kaki(double[][] array){
  for(int x = 0; x <= 125; x++){
    if (CLL [x][1] == 1 ){
      for (int y = 0; y <= 125; y++){
        if (array[y][1] != 99 && array[y][1] != 1
        && array[y][1] != 2 && array[y][1] != 3 &&
        array[y][1] != 4 && array[y][1] != 5 &&
        array[y][1] != 6){
          double temp = array[y][0];
          array[y][0] = CLL[x][0];
          array[y][1] = CLL[x][1];
          array[y][2] = CLL[x][2];
          CLL[x][0] = temp;
          CLL[x][1] = 69;
          }
        }
     }
  }
  for(int x = 0; x <= 125; x++){
    if (CLL [x][1] == 2 ){
      for (int y = 0; y <= 125; y++){
        if (array[y][1] != 99 && array[y][1] != 1
        && array[y][1] != 2 && array[y][1] != 3 &&
        array[y][1] != 4 && array[y][1] != 5 &&
        array[y][1] != 6){
          double temp = array[y][0];
          array[y][0] = CLL[x][0];
          array[y][1] = CLL[x][1];
          array[y][2] = CLL[x][2];
          CLL[x][0] = temp;
```

```
            CLL[x][1] = 69;
            }
          }
        }
    }
  for(int x = 0; x <= 125; x++){
    if (CLL [x][1] == 3 ){
      for (int y = 0; y <= 125; y++){
        if (array[y][1] != 99 && array[y][1] != 1 &&
        array[y][1] != 2 && array[y][1] != 3 &&
        array[y][1] != 4 && array[y][1] != 5 &&
        array[y][1] != 6){
          double temp = array[y][0];
          array[y][0] = CLL[x][0];
          array[y][1] = CLL[x][1];
          array[y][2] = CLL[x][2];
          CLL[x][0] = temp;
          CLL[x][1] = 69;
          }
        }
    }
  }

}
```

Fungsi ini bekerja dengan mencari peti kemas berukuran 20*ft* pada *Container Loading List* yang masih belum tertata dalam *array* proses. Setelah menemukan peti kemas yang belum tertata, mulai dilakukan pencarian indeks *array* proses yang masih kosong atau tidak terisi. Kemudian peti kemas 20*ft* pada *Container Loading List* dipindahkan ke indeks yang kosong pada *array* proses. Proses ini terus berulang dari indeks terkecil pada *array Container Loading List* hingga indeks terakhir. Peti kemas 20*ft* pada *Container Loading List* belum tentu dapat tertata semua pada *array* proses tergantung dari jumlah indeks kosong yang tersisa.

4.3.4.4. Implementasi Fungsi Penambahan Sisa Peti Kemas 40*ft*

Fungsi ini menghasilkan *output* dimana seluruh peti kemas berukuran 40*ft* pada *Container Loading List* tertata pada *array* proses dan mengisi lokasi yang kosong pada *array* proses. Namun, untuk penambahan peti kemas 40*ft* yang menempati dua lokasi dengan *bay* yang bersebelahan harus memenuhi syarat dimana dua lokasi kosong untuk pengisian peti kemas 40*ft* harus terletak pada dua *bay* dengan *row* dan *tier* yang sama. Karena penambahan sisa peti kemas dilakukan dari indeks *array* proses terkecil, maka peti kemas yang tertata diurutkan dari golongan yang terberat. Berikut *source code* untuk implemnetasi fungsi penambahan sisa peti kemas 40*ft*.

```
public void tambahsisa40kaki(double[][] array){
  for(int x = 0; x <= 119; x++){
    if (CLL [x][1] == 6 && CLL [x+6][1] == 6 ){
      for (int y = 119; y >= 0; y--){
        if (array[y][1] != 99 && array[y][1] != 1
        && array[y][1] != 2 && array[y][1] != 3 &&
        array[y][1] != 4 && array[y][1] != 5 &&
        array[y][1] != 6){
          if (array[y+6][1] != 99 && array[y+6][1]
          != 1 && array[y+6][1] != 2 &&
          array[y+6][1] != 3 && array[y+6][1] != 4
          && array[y+6][1] != 5 && array[y+6][1] !=
          6){
             double temp = array[y][0];
             double temp1 = array[y+6][0];
             array[y][0] = CLL[x][0];
             array[y][1] = CLL[x][1];
             array[y][2] = CLL[x][2];
             array[y+6][0] = CLL[x+6][0];
             array[y+6][1] = CLL[x+6][1];
             array[y+6][2] = CLL[x+6][2];
             CLL[x][0] = temp;
             CLL[x][1] = 69;
             CLL[x+6][0] = temp1;
             CLL[x+6][1] = 69;
            }
```

```
        }
      }
    }
  }
  for(int x = 0; x <= 119; x++){
    if (CLL [x][1] == 5 && CLL [x+6][1] == 5 ){
      for (int y = 119; y >= 0; y--){
        if (array[y][1] != 99 && array[y][1] != 1
        && array[y][1] != 2 && array[y][1] != 3 &&
        array[y][1] != 4 && array[y][1] != 5 &&
        array[y][1] != 6){
          if (array[y+6][1] != 99 && array[y+6][1]
          != 1 && array[y+6][1] != 2 &&
          array[y+6][1] != 3 && array[y+6][1] != 4
          && array[y+6][1] != 5 && array[y+6][1] !=
          6){
             double temp = array[y][0];
             double temp1 = array[y+6][0];
             array[y][0] = CLL[x][0];
             array[y][1] = CLL[x][1];
             array[y][2] = CLL[x][2];
             array[y+6][0] = CLL[x+6][0];
             array[y+6][1] = CLL[x+6][1];
             array[y+6][2] = CLL[x+6][2];
             CLL[x][0] = temp;
             CLL[x][1] = 69;
             CLL[x+6][0] = temp1;
             CLL[x+6][1] = 69;
           }
        }
      }
    }
  }
  for(int x = 0; x <= 119; x++){
    if (CLL [x][1] == 4 && CLL [x+6][1] == 4 ){
      for (int y = 119; y >= 0; y--){
        if (array[y][1] != 99 && array[y][1] != 1
        && array[y][1] != 2 && array[y][1] != 3 &&
        array[y][1] != 4 && array[y][1] != 5 &&
        array[y][1] != 6){
          if (array[y+6][1] != 99 && array[y+6][1]
          != 1 && array[y+6][1] != 2 &&
          array[y+6][1] != 3 && array[y+6][1] != 4
          && array[y+6][1] != 5 && array[y+6][1] !=
          6){
             double temp = array[y][0];
             double temp1 = array[y+6][0];
             array[y][0] = CLL[x][0];
             array[y][1] = CLL[x][1];
```

```
                array[y][2] = CLL[x][2];
                array[y+6][0] = CLL[x+6][0];
                array[y+6][1] = CLL[x+6][1];
                array[y+6][2] = CLL[x+6][2];
                CLL[x][0] = temp;
                CLL[x][1] = 69;
                CLL[x+6][0] = temp1;
                CLL[x+6][1] = 69;
               }
          }
        }
     }
  }
  pecah40kaki(array);

}
```

Fungsi ini bekerja dengan mencari peti kemas berukuran 40*ft* pada *Container Loading List* yang masih belum tertata dalam *array* proses. Setelah menemukan peti kemas yang belum tertata, mulai dilakukan pencarian dua indeks *array* proses yang masih kosong atau tidak terisi. Kemudian peti kemas 40*ft* pada *Container Loading List* dipindahkan ke indeks yang kosong pada *array* proses. Peletakkan peti kemas 40*ft* ini harus pada dua indeks yang bersesuaian, jadi tidak bisa sembarang indeks kosong yang tersedia pada *array* proses dapat terisi. Proses ini terus berulang dari indeks terkecil pada *array Container Loading List* hingga indeks terakhir. Peti kemas 40*ft* pada *Container Loading List* belum tentu dapat tertata semua pada *array* proses tergantung dari jumlah indeks kosong yang tersisa dan lokasi indeks kosong yang bersesuaian.

4.3.4. Implementasi Fungsi *Satisfy*

Fungsi ini menghasilkan *output* hasil tatanan peti kemas yang memenuhi syarat penataan peti kemas dan

syarat *subyektif* yang dilakukan *planner*. Fungsi ini melakukan penukaran antar dua peti kemas atau antara peti kemas dengan lokasi yang kosong pada tatanan sesuai dengan kondisi syarat yang ingin dipenuhi. Selain itu, dilakukan juga perhitungan *restow* muat atau jumlah peti kemas yang terkena *restow* akibat dari pertukaran lokasi peti kemas. Berikut *implementasi source code* untuk fungsi *satisfy* dimana dilakukannya proses *swap* peti kemas.

```
public void swapaction(double[][] proses, boolean[]
click){
  int banyak = 0;
  for (int i = 0; i <= 125; i++) {
    if (click[i] == true) { banyak++; }
  }
  if (banyak == 2) {
   int cek = 0;
   int hitungswap = 2;
   int[] tempswap = new int[2];
   for (int i = 0; i < click.length; i++) {
     if (click[i] == true) {
       tempswap[cek] = i;
       cek++; }
   }
   if(proses[tempswap[0]][1] != 4 &&
   proses[tempswap[0]][1] != 5 &&
   proses[tempswap[0]][1] != 6 &&
   proses[tempswap[0]][1] != 99){
     if(proses[tempswap[1]][1] != 4 &&
     proses[tempswap[1]][1] != 5 &&
     proses[tempswap[1]][1] != 6 &&
     proses[tempswap[1]][1] != 99){
      //proses restow muat
      if(tempswap[0] <= 41){
        if(proses[tempswap[0] + 42][1] == 1 ||
        proses[tempswap[0] + 42][1] == 2 ||
        proses[tempswap[0] + 42][1] == 3 ||
        proses[tempswap[0] + 42][1] == 4 ||
        proses[tempswap[0] + 42][1] == 5 ||
        proses[tempswap[0] + 42][1] == 6){ rm++ ; }
        if(proses[tempswap[0] + 84][1] == 1 ||
        proses[tempswap[0] + 84][1] == 2 ||
        proses[tempswap[0] + 84][1] == 3 ||
        proses[tempswap[0] + 84][1] == 4 ||
```

```
          proses[tempswap[0] + 84][1] == 5 ||
          proses[tempswap[0] + 84][1] == 6){ rm++ ; }
       }
      else if(tempswap[0] > 41 && tempswap[0] <=83){
        if(proses[tempswap[0] + 42][1] == 1 ||
        proses[tempswap[0] + 42][1] == 2 ||
        proses[tempswap[0] + 42][1] == 3 ||
        proses[tempswap[0] + 42][1] == 4 ||
        proses[tempswap[0] + 42][1] == 5 ||
        proses[tempswap[0] + 42][1] == 6){ rm++ ; }
      }
      if(tempswap[1] <= 41){
        if(proses[tempswap[1] + 42][1] == 1 ||
        proses[tempswap[1] + 42][1] == 2 ||
        proses[tempswap[1] + 42][1] == 3 ||
        proses[tempswap[1] + 42][1] == 4 ||
        proses[tempswap[1] + 42][1] == 5 ||
        proses[tempswap[1] + 42][1] == 6){ rm++ ; }
        if(proses[tempswap[1] + 84][1] == 1 ||
        proses[tempswap[1] + 84][1] == 2 ||
        proses[tempswap[1] + 84][1] == 3 ||
        proses[tempswap[1] + 84][1] == 4 ||
        proses[tempswap[1] + 84][1] == 5 ||
        proses[tempswap[1] + 84][1] == 6){ rm++ ; }
      }
      else if(tempswap[1] > 41 && tempswap[1] <= 83){
        if(proses[tempswap[1] + 42][1] == 1 ||
        proses[tempswap[1] + 42][1] == 2 ||
        proses[tempswap[1] + 42][1] == 3 ||
        proses[tempswap[1] + 42][1] == 4 ||
        proses[tempswap[1] + 42][1] == 5 ||
        proses[tempswap[1] + 42][1] == 6){ rm++ ; }
      }
      // proses mindah
      for (int i = 0; i < 3; i++) {
        double temp = proses[tempswap[0]][i];
        proses[tempswap[0]][i] =
        proses[tempswap[1]][i];
        proses[tempswap[1]][i] = temp;
        }
      }
   swapbutton20ft();
   totalswap++;
    }
    else { JOptionPane.showMessageDialog(this,
    "kontainer tidak dapat ditukar");
    }
}
```

```
else if (banyak == 4){
  int cek = 0;
  int hitungswap = 4;
  int[] tempswap = new int[4];
  for (int i = 0; i < click.length; i++) {
    if (click[i] == true) {
      tempswap[cek] = i;
      cek++;
    }
  }
  if(tempswap[1] == tempswap[0] + 6 && tempswap[3]
  == tempswap[2] + 6){
    //cek pasangan
    if (proses[tempswap[0]][1] == 4 ||
    proses[tempswap[0]][1] == 5 ||
    proses[tempswap[0]][1] == 6 &&
    proses[tempswap[0]][1] ==
    proses[tempswap[1]][1] &&
    proses[tempswap[0]][2] ==
    proses[tempswap[1]][2]){
      if(proses[tempswap[2]][1] == 4 ||
      proses[tempswap[2]][1] == 5 ||
      proses[tempswap[2]][1] == 6  &&
      proses[tempswap[2]][1] ==
      proses[tempswap[3]][1] &&
      proses[tempswap[2]][2] ==
      proses[tempswap[3]][2]){
        //proses restow muat
        if(tempswap[0] <= 41){
           if(proses[tempswap[0] + 42][1] == 1 ||
           proses[tempswap[0] + 42][1] == 2 ||
           proses[tempswap[0] + 42][1] == 3 ||
           proses[tempswap[0] + 42][1] == 4 ||
           proses[tempswap[0] + 42][1] == 5 ||
           proses[tempswap[0] + 42][1] == 6){
           rm++ ; }
           if(proses[tempswap[0] + 84][1] == 1 ||
           proses[tempswap[0] + 84][1] == 2 ||
           proses[tempswap[0] + 84][1] == 3 || \
           proses[tempswap[0] + 84][1] == 4 ||
           proses[tempswap[0] + 84][1] == 5 ||
           proses[tempswap[0] + 84][1] == 6){
           rm++ ; }
        }
        else if(tempswap[0] > 41 && tempswap[0] <=
        83){
           if(proses[tempswap[0] + 42][1] == 1 ||
           proses[tempswap[0] + 42][1] == 2 ||
           proses[tempswap[0] + 42][1] == 3 ||
```

```
            proses[tempswap[0] + 42][1] == 4 ||
            proses[tempswap[0] + 42][1] == 5 ||
            proses[tempswap[0] + 42][1] == 6){
             rm++ ; }
          }
          if(tempswap[1] <= 41){
             if(proses[tempswap[1] + 42][1] == 1 ||
             proses[tempswap[1] + 42][1] == 2 ||
             proses[tempswap[1] + 42][1] == 3 ||
             proses[tempswap[1] + 42][1] == 4 ||
             proses[tempswap[1] + 42][1] == 5 ||
             proses[tempswap[1] + 42][1] == 6){
             rm++ ; }
             if(proses[tempswap[1] + 84][1] == 1 ||
             proses[tempswap[1] + 84][1] == 2 ||
             proses[tempswap[1] + 84][1] == 3 ||
             proses[tempswap[1] + 84][1] == 4 ||
             proses[tempswap[1] + 84][1] == 5 ||
             proses[tempswap[1] + 84][1] == 6){
             rm++ ; }
          }
          else if(tempswap[1] > 41 && tempswap[1] <=
          83){
             if(proses[tempswap[1] + 42][1] == 1 ||
             proses[tempswap[1] + 42][1] == 2 ||
             proses[tempswap[1] + 42][1] == 3 ||
             proses[tempswap[1] + 42][1] == 4 ||
             proses[tempswap[1] + 42][1] == 5 ||
             proses[tempswap[1] + 42][1] == 6){
             rm++ ; }
          }
          if(tempswap[2] <= 41){
             if(proses[tempswap[2] + 42][1] == 1 ||
             proses[tempswap[2] + 42][1] == 2 ||
             proses[tempswap[2] + 42][1] == 3 ||
             proses[tempswap[2] + 42][1] == 4 ||
             proses[tempswap[2] + 42][1] == 5 ||
             proses[tempswap[2] + 42][1] == 6){
             rm++ ;}
             if(proses[tempswap[2] + 84][1] == 1 ||
             proses[tempswap[2] + 84][1] == 2 ||
             proses[tempswap[2] + 84][1] == 3 ||
             proses[tempswap[2] + 84][1] == 4 ||
             proses[tempswap[2] + 84][1] == 5 ||
             proses[tempswap[2] + 84][1] == 6){
             rm++ ;
                 }
             }
          else if(tempswap[2] > 41 && tempswap[2] <=
```

```
 83){
    if(proses[tempswap[2] + 42][1] == 1 ||
    proses[tempswap[2] + 42][1] == 2 ||
    proses[tempswap[2] + 42][1] == 3 ||
    proses[tempswap[2] + 42][1] == 4 ||
    proses[tempswap[2] + 42][1] == 5 ||
    proses[tempswap[2] + 42][1] == 6){
    rm++ ; }
}
if(tempswap[3] <= 41){
    if(proses[tempswap[3] + 42][1] == 1 ||
    proses[tempswap[3] + 42][1] == 2 ||
    proses[tempswap[3] + 42][1] == 3 ||
    proses[tempswap[3] + 42][1] == 4 ||
    proses[tempswap[3] + 42][1] == 5 ||
    proses[tempswap[3] + 42][1] == 6){
    rm++ ; }
    if(proses[tempswap[3] + 84][1] == 1 ||
    proses[tempswap[3] + 84][1] == 2 ||
    proses[tempswap[3] + 84][1] == 3 ||
    proses[tempswap[3] + 84][1] == 4 ||
    proses[tempswap[3] + 84][1] == 5 ||
    proses[tempswap[3] + 84][1] == 6){
    rm++ ; }
}
else if(tempswap[3] > 41 && tempswap[3] <=
83){
    if(proses[tempswap[3] + 42][1] == 1 ||
    proses[tempswap[3] + 42][1] == 2 ||
    proses[tempswap[3] + 42][1] == 3 ||
    proses[tempswap[3] + 42][1] == 4 ||
    proses[tempswap[3] + 42][1] == 5 ||
    proses[tempswap[3] + 42][1] == 6){
    rm++ ; }
}
//jika keduanya 40ft
for (int i = 0; i < 3; i++) {
  double temp = proses[tempswap[0]][i];
  double temp1 = proses[tempswap[1]][i];
  proses[tempswap[0]][i] =
  proses[tempswap[2]][i];
  proses[tempswap[2]][i] = temp;
  proses[tempswap[1]][i] =
  proses[tempswap[3]][i];
  proses[tempswap[3]][i] = temp1; }
swapbutton40ft();
totalswap++;
}
else if (proses[tempswap[2]][1] == 1 ||
```

```
        proses[tempswap[2]][1] == 2 ||
        proses[tempswap[2]][1] == 3 ||
        proses[tempswap[2]][1] == 69 ||
        proses[tempswap[2]][1] == 0){
          if(proses[tempswap[3]][1] == 1 ||
          proses[tempswap[3]][1] == 2 ||
          proses[tempswap[3]][1] == 3 ||
          proses[tempswap[3]][1] == 69 ||
          proses[tempswap[3]][1] == 0){
            //proses restow muat
            if(tempswap[0] <= 41){
               if(proses[tempswap[0] + 42][1] == 1
               || proses[tempswap[0] + 42][1] == 2
               || proses[tempswap[0] + 42][1] == 3
               || proses[tempswap[0] + 42][1] == 4
               || proses[tempswap[0] + 42][1] == 5
               || proses[tempswap[0] + 42][1] == 6)
               { rm++ ; }
               if(proses[tempswap[0] + 84][1] == 1
               || proses[tempswap[0] + 84][1] == 2
               || proses[tempswap[0] + 84][1] == 3
               || proses[tempswap[0] + 84][1] == 4
               || proses[tempswap[0] + 84][1] == 5
               || proses[tempswap[0] + 84][1] == 6)
               { rm++ ; }
          }
          else if(tempswap[0] > 41 && tempswap[0]
          <= 83){
            if(proses[tempswap[0] + 42][1] == 1 ||
            proses[tempswap[0] + 42][1] == 2 ||
            proses[tempswap[0] + 42][1] == 3 ||
            proses[tempswap[0] + 42][1] == 4 ||
            proses[tempswap[0] + 42][1] == 5 ||
            proses[tempswap[0] + 42][1] == 6){
            rm++ ; }
         }
         if(tempswap[1] <= 41){
            if(proses[tempswap[1] + 42][1] == 1 ||
            proses[tempswap[1] + 42][1] == 2 ||
            proses[tempswap[1] + 42][1] == 3 ||
            proses[tempswap[1] + 42][1] == 4 ||
            proses[tempswap[1] + 42][1] == 5 ||
            proses[tempswap[1] + 42][1] == 6){
            rm++ ; }
            if(proses[tempswap[1] + 84][1] == 1 ||
            proses[tempswap[1] + 84][1] == 2 ||
            proses[tempswap[1] + 84][1] == 3 ||
            proses[tempswap[1] + 84][1] == 4 ||
            proses[tempswap[1] + 84][1] == 5 ||
```

```
            proses[tempswap[1] + 84][1] == 6){
            rm++ ; }
        }
        else if(tempswap[1] > 41 && tempswap[1] <=
        83){
            if(proses[tempswap[1] + 42][1] == 1 ||
            proses[tempswap[1] + 42][1] == 2 ||
            proses[tempswap[1] + 42][1] == 3 ||
            proses[tempswap[1] + 42][1] == 4 ||
            proses[tempswap[1] + 42][1] == 5 ||
            proses[tempswap[1] + 42][1] == 6){
            rm++ ; }
        }
        if(tempswap[2] <= 41){
            if(proses[tempswap[2] + 42][1] == 1 ||
            proses[tempswap[2] + 42][1] == 2 ||
            proses[tempswap[2] + 42][1] == 3 ||
            proses[tempswap[2] + 42][1] == 4 ||
            proses[tempswap[2] + 42][1] == 5 ||
            proses[tempswap[2] + 42][1] == 6){
            rm++ ; }
            if(proses[tempswap[2] + 84][1] == 1 ||
            proses[tempswap[2] + 84][1] == 2 ||
            proses[tempswap[2] + 84][1] == 3 ||
            proses[tempswap[2] + 84][1] == 4 ||
            proses[tempswap[2] + 84][1] == 5 ||
            proses[tempswap[2] + 84][1] == 6){
            rm++ ; }
        }
        else if(tempswap[2] > 41 && tempswap[2] <=
        83){
            if(proses[tempswap[2] + 42][1] == 1 ||
            proses[tempswap[2] + 42][1] == 2 ||
            proses[tempswap[2] + 42][1] == 3 ||
            proses[tempswap[2] + 42][1] == 4 ||
            proses[tempswap[2] + 42][1] == 5 ||
            proses[tempswap[2] + 42][1] == 6){
            rm++ ; }
        }
        if(tempswap[3] <= 41){
            if(proses[tempswap[3] + 42][1] == 1 ||
            proses[tempswap[3] + 42][1] == 2 ||
            proses[tempswap[3] + 42][1] == 3 ||
            proses[tempswap[3] + 42][1] == 4 ||
            proses[tempswap[3] + 42][1] == 5 ||
            proses[tempswap[3] + 42][1] == 6){
            rm++ ; }
            if(proses[tempswap[3] + 84][1] == 1 ||
            proses[tempswap[3] + 84][1] == 2 ||
```

```
            proses[tempswap[3] + 84][1] == 3 ||
            proses[tempswap[3] + 84][1] == 4 ||
            proses[tempswap[3] + 84][1] == 5 ||
            proses[tempswap[3] + 84][1] == 6){
            rm++ ; }
        }
        else if(tempswap[3] > 41 && tempswap[3] <=
        83){
            if(proses[tempswap[3] + 42][1] == 1 ||
            proses[tempswap[3] + 42][1] == 2 ||
            proses[tempswap[3] + 42][1] == 3 ||
            proses[tempswap[3] + 42][1] == 4 ||
            proses[tempswap[3] + 42][1] == 5 ||
            proses[tempswap[3] + 42][1] == 6){
            rm++ ; }
        }
        //jika swap dengan 20ft dan kosong
        for (int i = 0; i < 3; i++) {
          double temp = proses[tempswap[0]][i];
          double temp1 = proses[tempswap[1]][i];
          proses[tempswap[0]][i] =
          proses[tempswap[2]][i];
          proses[tempswap[2]][i] = temp;
          proses[tempswap[1]][i] =
          proses[tempswap[3]][i];
          proses[tempswap[3]][i] = temp1; }
        swapbutton40ft();
        totalswap++;  }
        else { JOptionPane.showMessageDialog(this,
        "pilih kontainer 20ft"); }
    }
}
else if (proses[tempswap[0]][1] == 1 ||
proses[tempswap[0]][1] == 2 ||
proses[tempswap[0]][1] == 3 ||
proses[tempswap[0]][1] == 69 ||
proses[tempswap[0]][1] == 0 ){
  if(proses[tempswap[1]][1] == 1 ||
    proses[tempswap[1]][1] == 2 ||
    proses[tempswap[1]][1] == 3 ||
    proses[tempswap[1]][1] == 69 ||
    proses[tempswap[1]][1] == 0){
      if(proses[tempswap[2]][1] == 4 ||
      proses[tempswap[2]][1] == 5 ||
      proses[tempswap[2]][1] == 6 &&
      proses[tempswap[2]][1] ==
      proses[tempswap[3]][1] &&
      proses[tempswap[2]][2] ==
      proses[tempswap[3]][2]){
```

```
      //proses restow muat
      if(tempswap[0] <= 41){
         if(proses[tempswap[0] + 42][1] == 1 ||
         proses[tempswap[0] + 42][1] == 2 ||
         proses[tempswap[0] + 42][1] == 3 ||
         proses[tempswap[0] + 42][1] == 4 ||
         proses[tempswap[0] + 42][1] == 5 ||
         proses[tempswap[0] + 42][1] == 6){ rm++ ; }
         if(proses[tempswap[0] + 84][1] == 1 ||
         proses[tempswap[0] + 84][1] == 2 ||
         proses[tempswap[0] + 84][1] == 3 ||
         proses[tempswap[0] + 84][1] == 4 ||
         proses[tempswap[0] + 84][1] == 5 ||
         proses[tempswap[0] + 84][1] == 6){ rm++ ; }
      }
      else if(tempswap[0] > 41 && tempswap[0] <=
      83){
         if(proses[tempswap[0] + 42][1] == 1 ||
         proses[tempswap[0] + 42][1] == 2 ||
         proses[tempswap[0] + 42][1] == 3 ||
         proses[tempswap[0] + 42][1] == 4 ||
         proses[tempswap[0] + 42][1] == 5 ||
         proses[tempswap[0] + 42][1] == 6){ rm++ ; }
      }
      if(tempswap[1] <= 41){
         if(proses[tempswap[1] + 42][1] == 1 ||
         proses[tempswap[1] + 42][1] == 2 ||
         proses[tempswap[1] + 42][1] == 3 ||
         proses[tempswap[1] + 42][1] == 4 ||
         proses[tempswap[1] + 42][1] == 5 ||
         proses[tempswap[1] + 42][1] == 6){ rm++ ; }
         if(proses[tempswap[1] + 84][1] == 1 ||
         proses[tempswap[1] + 84][1] == 2 ||
         proses[tempswap[1] + 84][1] == 3 ||
         proses[tempswap[1] + 84][1] == 4 ||
         proses[tempswap[1] + 84][1] == 5 ||
         proses[tempswap[1] + 84][1] == 6){ rm++ ; }
      }
      else if(tempswap[1] > 41 && tempswap[1] <=
      83){
         if(proses[tempswap[1] + 42][1] == 1 ||
         proses[tempswap[1] + 42][1] == 2 ||
         proses[tempswap[1] + 42][1] == 3 ||
         proses[tempswap[1] + 42][1] == 4 ||
         proses[tempswap[1] + 42][1] == 5 ||
         proses[tempswap[1] + 42][1] == 6){ rm++ ; }
      }
      if(tempswap[2] <= 41){
         if(proses[tempswap[2] + 42][1] == 1 ||
```

```
        proses[tempswap[2] + 42][1] == 2 ||
        proses[tempswap[2] + 42][1] == 3 ||
        proses[tempswap[2] + 42][1] == 4 ||
        proses[tempswap[2] + 42][1] == 5 ||
        proses[tempswap[2] + 42][1] == 6){ rm++ ; }
        if(proses[tempswap[2] + 84][1] == 1 ||
        proses[tempswap[2] + 84][1] == 2 ||
        proses[tempswap[2] + 84][1] == 3 ||
        proses[tempswap[2] + 84][1] == 4 ||
        proses[tempswap[2] + 84][1] == 5 ||
        proses[tempswap[2] + 84][1] == 6){ rm++ ; }
    }
    else if(tempswap[2] > 41 && tempswap[2] <= 83){
        if(proses[tempswap[2] + 42][1] == 1 ||
        proses[tempswap[2] + 42][1] == 2 ||
        proses[tempswap[2] + 42][1] == 3 ||
        proses[tempswap[2] + 42][1] == 4 ||
        proses[tempswap[2] + 42][1] == 5 ||
        proses[tempswap[2] + 42][1] == 6){ rm++ ; }
    }
    if(tempswap[3] <= 41){
        if(proses[tempswap[3] + 42][1] == 1 ||
        proses[tempswap[3] + 42][1] == 2 ||
        proses[tempswap[3] + 42][1] == 3 ||
        proses[tempswap[3] + 42][1] == 4 ||
        proses[tempswap[3] + 42][1] == 5 ||
        proses[tempswap[3] + 42][1] == 6){ rm++ ; }
        if(proses[tempswap[3] + 84][1] == 1 ||
        proses[tempswap[3] + 84][1] == 2 ||
        proses[tempswap[3] + 84][1] == 3 ||
        proses[tempswap[3] + 84][1] == 4 ||
        proses[tempswap[3] + 84][1] == 5 ||
        proses[tempswap[3] + 84][1] == 6){ rm++ ; }
    }
    else if(tempswap[3] > 41 && tempswap[3] <= 83){
        if(proses[tempswap[3] + 42][1] == 1 ||
        proses[tempswap[3] + 42][1] == 2 ||
        proses[tempswap[3] + 42][1] == 3 ||
        proses[tempswap[3] + 42][1] == 4 ||
        proses[tempswap[3] + 42][1] == 5 ||
        proses[tempswap[3] + 42][1] == 6){ rm++ ; }
    }
    //untuk swap 20ft dengan 40ft
    for (int i = 0; i < 3; i++) {
      double temp = proses[tempswap[0]][i];
      double temp1 = proses[tempswap[1]][i];
      proses[tempswap[0]][i] =
      proses[tempswap[2]][i];
      proses[tempswap[2]][i] = temp;
```

```
        proses[tempswap[1]][i] =
        proses[tempswap[3]][i];
        proses[tempswap[3]][i] = temp1; }
      swapbutton40ft();
      totalswap++; }
    else { JOptionPane.showMessageDialog(this,
    "pilih pasangan kontainer 40ft"); }
  } else { JOptionPane.showMessageDialog(this,
   "pilih kontainer 20ft"); }
 }
} else if (tempswap[2] == tempswap[0] + 6 &&
  tempswap[3] == tempswap[1] + 6){
  JOptionPane.showMessageDialog(this, "lokasi
  kontainer yang akan ditukar terlalu dekat");
} else {
  JOptionPane.showMessageDialog(this, "pemilihan
  pasangan kotainer tidak sesuai"); }
} else {
  JOptionPane.showMessageDialog(this, "harus 2
  button yang di klik untuk menukar kontainer 20ft
  dan 4 button yang di klik untuk menukar kontainer
  40ft"); }

}
```

Terdapat dua jenis proses swap, yaitu penukaran peti kemas 20*ft* dan penukaran peti kemas 40*ft*. Untuk penukaran sesama peti kemas 20*ft*, maka dilakukan pemilihan hanya dua *button*, kemudian terjadi pertukaran antar dua *button* tersebut. Namun, untuk penukaran peti kemas 40*ft*, maka harus dipilih empat *button* dengan lokasi yang bersesuaian. Peti kemas 40*ft* dapat ditukar dengan sesame peti kemas 40*ft* atau dengan dua peti kemas berukuran 20*ft*. Penukaran terjadi pada empat indeks, yaitu indeks 0 dengan 2 dan indeks 1 dengan indeks 3.

4.3.5. Implementasi Fungsi Evaluasi *Restow*

Fungsi evaluasi *restow* menghasilkan *output* berupa jumlah peti kemas yang terkena *restow*. Sebelum melakukan *restow*, dilakukan pembongkaran peti kemas

pada pelabuhan tujuan. Berikut *source code* untuk membongkar peti kemas di salah satu pelabuhan tujuan

```
public void bongkar(double[][] proses3){
  for(int j=0; j<=125; j++){
    if(proses2[j][2] == 1){
       proses2[j][1] = 69;
       proses2[j][2] = 69;
    }
  }
}
```

setelah dilakukan pembongkaran peti kemas, maka dilakukan pengecekan peti kemas yang melayang akibat pembongkaran ini. Kemudian peti kemas yang terkena *restow* ditambahkan ke dalam *Container Loading List* pelabuhan yang sedang diproses. Berikut *implementasi source code* untuk fungsi evaluasi *restow.*

```
public int restow(){
  int a = 0;
  for(int r=125; r>=84; r--){
    if(prosesDatang[r][1] == 1 ||
    prosesDatang[r][1] == 2 || prosesDatang[r][1]
    == 3){
      if(prosesDatang[r-42][1] == 69 &&
      prosesDatang[r-42][2] == 69 ){
         a++; }
      else if(prosesDatang[r-84][1] == 69 &&
      prosesDatang[r-84][2] == 69){
         a++; }
     }
  }
  for(int r=125; r>=90; r--){
    if (prosesDatang[r][1] == 4 && prosesDatang[r-
    6][1] == 4 && prosesDatang[r][2] ==
    prosesDatang[r-6][2]){
      if(prosesDatang[r-42][1] == 69 &&
      prosesDatang[r-42][2] == 69 ||
      prosesDatang[r-42-6][1] == 69 &&
      prosesDatang[r-42-6][2] == 69){
        a++; }
```

```
        else if(prosesDatang[r-84][1] == 69 &&
        prosesDatang[r-84][2] == 69 ||
        prosesDatang[r-84-6][1] == 69 &&
        prosesDatang[r-84-6][2] == 69){
          a++; }
    }
    else if (prosesDatang[r][1] == 5 &&
    prosesDatang[r-6][1] == 5 && prosesDatang[r][2]
    == prosesDatang[r-6][2]){
        if(prosesDatang[r-42][1] == 69 && \
        prosesDatang[r-42][2] == 69 ||
        prosesDatang[r-42-6][1] == 69 &&
        prosesDatang[r-42-6][2] == 69){
          a++; }
        else if(prosesDatang[r-84][1] == 69 &&
        prosesDatang[r-84][2] == 69 ||
        prosesDatang[r-84-6][1] == 69 &&
        prosesDatang[r-84-6][2] == 69){
          a++; }
    }
    else if (prosesDatang[r][1] == 6 &&
    prosesDatang[r-6][1] == 6 && prosesDatang[r][2]
    == prosesDatang[r-6][2]){
        if(prosesDatang[r-42][1] == 69 &&
        prosesDatang[r-42][2] == 69 ||
        prosesDatang[r-42-6][1] == 69 &&
        prosesDatang[r-42-6][2] == 69){
          a++; }
        else if(prosesDatang[r-84][1] == 69 &&
        prosesDatang[r-84][2] == 69 ||
        prosesDatang[r-84-6][1] == 69 &&
        prosesDatang[r-84-6][2] == 69){
          a++; }
    }
  }
  for(int r=83; r>=42;r--){
     if(prosesDatang[r][1] == 1 ||
     prosesDatang[r][1] == 2 || prosesDatang[r][1]
     == 3){
        if(prosesDatang[r-42][1] == 69 &&
        prosesDatang[r-42][2] == 69 ){
          a++; }
     }
  }
  for(int r=83;r>=48;r--){
     if (prosesDatang[r][1] == 4 && prosesDatang[r-
     6][1] == 4 && prosesDatang[r][2] ==
     prosesDatang[r-6][2]){
        if(prosesDatang[r-42][1] == 69 &&
```

```
            prosesDatang[r-42][2] == 69 ||
            prosesDatang[r-42-6][1] == 69 &&
            prosesDatang[r-42-6][2] == 69){
               a++; }
         }
         else if (prosesDatang[r][1] == 5 &&
         prosesDatang[r-6][1] == 5 &&
         prosesDatang[r][2] == prosesDatang[r-6][2]){
            if(prosesDatang[r-42][1] == 69 &&
            prosesDatang[r-42][2] == 69 ||
            prosesDatang[r-42-6][1] == 69 &&
            prosesDatang[r-42-6][2] == 69){
               a++; }
         }
         else if (prosesDatang[r][1] == 6 &&
         prosesDatang[r-6][1] == 6 &&
         prosesDatang[r][2] == prosesDatang[r-6][2]){
            if(prosesDatang[r-42][1] == 69 &&
            prosesDatang[r-42][2] == 69 ||
            prosesDatang[r-42-6][1] == 69 &&
            prosesDatang[r-42-6][2] == 69){
               a++; }
       }
   }
   return a;

}
```

Untuk menambahkan peti kemas berukuran 20*ft* yang terkena *restow* ke dalam *Container Loading List* pelabuhan yang sedang diproses, digunakan fungsi ubah*CLL20ft*. Berikut *source code* dari fungsi ubah*CLL20ft*

```
public void ubahCLL20ft(double[][] proses){
  for(int i=125; i>=84; i--){
    if(proses[i][1] == 1 || proses[i][1] == 2 ||
    proses[i][1] == 3){
      if(proses[i-42][1] == 69  && proses[i-42][2]
      == 69){
        for(int j=125; j>=0; j--){
          if (CLL[j][1] != 99 && CLL[j][1] != 1 &&
          CLL[j][1] != 2 && CLL[j][1] != 3 &&
          CLL[j][1] != 4 && CLL[j][1] != 5 &&
          CLL[j][1] != 6){
             double temp = proses[i][0];
```

```
            double temp1 = proses[i][1];
            double temp2 = proses[i][2];
            CLL[j][0] = temp;
            CLL[j][1] = temp1;
            CLL[j][2] = temp2;
            proses[i][1] = 69;
            proses[i][2] = 69;
            proses[i][0] = 0; }
        }
    }
    else if (proses[i-84][1] == 69  && proses[i-
    84][2] == 69){
      for(int j=125; j>=0; j--){
        if (CLL[j][1] != 99 && CLL[j][1] != 1 &&
        CLL[j][1] != 2 && CLL[j][1] != 3 &&
        CLL[j][1] != 4 && CLL[j][1] != 5 &&
        CLL[j][1] != 6){
           double temp = proses[i][0];
           double temp1 = proses[i][1];
           double temp2 = proses[i][2];
           CLL[j][0] = temp;
           CLL[j][1] = temp1;
           CLL[j][2] = temp2;
           proses[i][1] = 69;
           proses[i][2] = 69;
           proses[i][0] = 0;
           }
         }
       }
     }
   }
for (int i=83; i>=42; i--){
  if(proses[i][1] == 1 || proses[i][1] == 2 ||
  proses[i][1] == 3){
    if(proses[i-42][1] == 69  && proses[i-42][2]
    == 69){
      for(int j=125; j>=0; j--){
        if (CLL[j][1] != 99 && CLL[j][1] != 1 &&
        CLL[j][1] != 2 && CLL[j][1] != 3 &&
        CLL[j][1] != 4 && CLL[j][1] != 5 &&
        CLL[j][1] != 6){
           double temp = proses[i][0];
           double temp1 = proses[i][1];
           double temp2 = proses[i][2];
           CLL[j][0] = temp;
           CLL[j][1] = temp1;
           CLL[j][2] = temp2;
           proses[i][1] = 69;
           proses[i][2] = 69;
```

```
                    proses[i][0] = 0;
                  }
                }
              }
            }
          }
        }
```

Sedangkan untuk menambahkan peti kemas berukuran 40*ft* yang terkena *restow* ke dalam *Container Loading List* pelabuhan yang sedang diproses, digunakan fungsi ubah*CLL40ft.* Berikut *source code* dari fungsi ubah*CLL40ft*

```
public void ubahkeCLL40ft(double[][] proses3){
  for(int r=125; r>=90; r--){
    if (proses[r][1] == 4 && proses[r-6][1] == 4 &&
    proses[r][2] == proses[r-6][2]){
      if(proses[r-42][1] == 69 && proses[r-42][2]
      == 69 || proses[r-42-6][1] == 69 && proses[r-
      42-6][2] == 69){
        for(int j=125; j>=6; j--){
          if (CLL[j][1] != 99 && CLL[j][1] != 1 &&
          CLL[j][1] != 2 && CLL[j][1] != 3 &&
          CLL[j][1] != 4 && CLL[j][1] != 5 &&
          CLL[j][1] != 6){
           if (CLL[j-6][1] != 99 && CLL[j-6][1] !=1
           && CLL[j-6][1] != 2 && CLL[j-6][1] != 3
           && CLL[j-6][1] != 4 && CLL[j-6][1] != 5
           && CLL[j-6][1] != 6){
              CLL[j-6][0] = proses[r-6][0]*2;
              CLL[j-6][1] = proses[r-6][1];
              CLL[j-6][2] = proses[r-6][2];
              CLL[j][0] = 0;
              CLL[j][1] = proses[r][1];
              CLL[j][2] = proses[r][2];
              proses[r][1] = 69;
              proses[r][2] = 69;
              proses[r][0] = 0;
              proses[r-6][1] = 69;
              proses[r-6][2] = 69;
              proses[r-6][0] = 0;
           }
          }
```

```
      }
    }
    else if (proses[r-84][1] == 69 && proses[r-
    84][2] == 69 || proses[r-84-6][1] == 69 &&
    proses[r-84-6][2] == 69){
      for(int j=125; j>=6; j--){
        if (CLL[j][1] != 99 && CLL[j][1] != 1 &&
        CLL[j][1] != 2 && CLL[j][1] != 3 &&
        CLL[j][1] != 4 && CLL[j][1] != 5 &&
        CLL[j][1] != 6){
          if (CLL[j-6][1] != 99 && CLL[j-6][1] != 1
          && CLL[j-6][1] != 2 && CLL[j-6][1] != 3 &&
          CLL[j-6][1] != 4 && CLL[j-6][1] != 5 &&
          CLL[j-6][1] != 6){
             CLL[j-6][0] = proses[r-6][0]*2;
             CLL[j-6][1] = proses[r-6][1];
             CLL[j-6][2] = proses[r-6][2];
             CLL[j][0] = 0;
             CLL[j][1] = proses[r][1];
             CLL[j][2] = proses[r][2];
             proses[r][1] = 69;
             proses[r][2] = 69;
             proses[r][0] = 0;
             proses[r-6][1] = 69;
             proses[r-6][2] = 69;
             proses[r-6][0] = 0;
           }
         }
       }
     }
   }
   else if (proses[r][1] == 5 && proses[r-6][1] == 5
   && proses[r][2] == proses[r-6][2]){
     if(proses[r-42][1] == 69 && proses[r-42][2] ==
     69 || proses[r-42-6][1] == 69 && proses[r-42-
     6][2] == 69){
        for(int j=125; j>=6; j--){
          if (CLL[j][1] != 99 && CLL[j][1] != 1 &&
          CLL[j][1] != 2 && CLL[j][1] != 3 &&
          CLL[j][1] != 4 && CLL[j][1] != 5 &&
          CLL[j][1] != 6){
            if (CLL[j-6][1] != 99 && CLL[j-6][1] !=1
            && CLL[j-6][1] != 2 && CLL[j-6][1] != 3
            && CLL[j-6][1] != 4 && CLL[j-6][1] != 5
            && CLL[j-6][1] != 6){
               CLL[j-6][0] = proses[r-6][0]*2;
               CLL[j-6][1] = proses[r-6][1];
               CLL[j-6][2] = proses[r-6][2];
               CLL[j][0] = 0;
```

```
                CLL[j][1] = proses[r][1];
                CLL[j][2] = proses[r][2];
                proses[r][1] = 69;
                proses[r][2] = 69;
                proses[r][0] = 0;
                proses[r-6][1] = 69;
                proses[r-6][2] = 69;
                proses[r-6][0] = 0;
               }
             }
           }
         }
         else if (proses[r-84][1] == 69 &&
         proses[r-84][2] == 69 || proses[r-84-
         6][1] == 69 && proses[r-84-6][2] == 69){
           for(int j=125; j>=6; j--){
             if (CLL[j][1] != 99 && CLL[j][1] != 1
             && CLL[j][1] != 2 && CLL[j][1] != 3
             && CLL[j][1] != 4 && CLL[j][1] != 5
             && CLL[j][1] != 6){
             if (CLL[j-6][1] != 99 && CLL[j-6][1]
             != 1 && CLL[j-6][1] != 2 && CLL[j-
             6][1] != 3 && CLL[j-6][1] != 4 &&
             CLL[j-6][1] != 5 && CLL[j-6][1] !=
             6){
                CLL[j-6][0] = proses[r-6][0]*2;
                CLL[j-6][1] = proses[r-6][1];
                CLL[j-6][2] = proses[r-6][2];
                CLL[j][0] = 0;
                CLL[j][1] = proses[r][1];
                CLL[j][2] = proses[r][2];
                proses[r][1] = 69;
                proses[r][2] = 69;
                proses[r][0] = 0;
                proses[r-6][1] = 69;
                proses[r-6][2] = 69;
                proses[r-6][0] = 0;
                }
               }
              }
             }
            }
            else if (proses[r][1] == 6 && proses[r-
            6][1] == 6 && proses[r][2] == proses[r-
            6][2]){
               if(proses[r-42][1] == 69 && proses[r-
               42][2] == 69 || proses[r-42-6][1] ==
               69 && proses[r-42-6][2] == 69){
                  for(int j=125; j>=6; j--){
```

```
            if (CLL[j][1] != 99 && CLL[j][1]
            != 1 && CLL[j][1] != 2 &&
            CLL[j][1] != 3 && CLL[j][1] != 4
            && CLL[j][1] != 5 && CLL[j][1] !=
            6){
              if (CLL[j-6][1] != 99 && CLL[j-
              6][1] != 1 && CLL[j-6][1] != 2
              && CLL[j-6][1] != 3 && CLL[j-
              6][1] != 4 && CLL[j-6][1] != 5
              && CLL[j-6][1] != 6){
                CLL[j-6][0] = proses[r-6][0]*2;
                CLL[j-6][1] = proses[r-6][1];
                CLL[j-6][2] = proses[r-6][2];
                CLL[j][0] = 0;
                CLL[j][1] = proses[r][1];
                CLL[j][2] = proses[r][2];
                proses[r][1] = 69;
                proses[r][2] = 69;
                proses[r][0] = 0;
                proses[r-6][1] = 69;
                proses[r-6][2] = 69;
                proses[r-6][0] = 0;
                }
            }
          }
        }
      else if (proses[r-84][1] == 69 &&
      proses[r-84][2] == 69 || proses[r-84-
      6][1] == 69 && proses[r-84-6][2] ==
      69){
        for(int j=125; j>=6; j--){
          if (CLL[j][1] != 99 && CLL[j][1] !=1
          && CLL[j][1] != 2 && CLL[j][1] != 3
          && CLL[j][1] != 4 && CLL[j][1] != 5
          && CLL[j][1] != 6){
            if (CLL[j-6][1] != 99 && CLL[j-
            6][1] != 1 && CLL[j-6][1] != 2 &&
            CLL[j-6][1] != 3 && CLL[j-6][1] !=
            4 && CLL[j-6][1] != 5 && CLL[j-
            6][1] != 6){
              CLL[j-6][0] = proses[r-6][0]*2;
              CLL[j-6][1] = proses[r-6][1];
              CLL[j-6][2] = proses[r-6][2];
              CLL[j][0] = 0;
              CLL[j][1] = proses[r][1];
              CLL[j][2] = proses[r][2];
              proses[r][1] = 69;
              proses[r][2] = 69;
              proses[r][0] = 0;
```

```
                    proses[r-6][1] = 69;
                    proses[r-6][2] = 69;
                    proses[r-6][0] = 0;
                  }
                }
              }
            }
          }
        }
for (int r=83; r>=48; r--){
  if (proses[r][1] == 4 && proses[r-6][1] == 4 &&
  proses[r][2] == proses[r-6][2]){
    if(proses[r-42][1] == 69 && proses[r-42][2] ==
    69 || proses[r-42-6][1] == 69 && proses[r-42-
    6][2] == 69){
       for(int j=125; j>=6; j--){
         if (CLL[j][1] != 99 && CLL[j][1] != 1 &&
         CLL[j][1] != 2 && CLL[j][1] != 3 &&
         CLL[j][1] != 4 && CLL[j][1] != 5 &&
         CLL[j][1] != 6){
           if (CLL[j-6][1] != 99 && CLL[j-6][1] !=1
           && CLL[j-6][1] != 2 && CLL[j-6][1] != 3
           && CLL[j-6][1] != 4 && CLL[j-6][1] != 5
           && CLL[j-6][1] != 6){
              CLL[j-6][0] = proses[r-6][0]*2;
              CLL[j-6][1] = proses[r-6][1];
              CLL[j-6][2] = proses[r-6][2];
              CLL[j][0] = 0;
              CLL[j][1] = proses[r][1];
              CLL[j][2] = proses[r][2];
              proses[r][1] = 69;
              proses[r][2] = 69;
              proses[r][0] = 0;
              proses[r-6][1] = 69;
              proses[r-6][2] = 69;
              proses[r-6][0] = 0;
              }
           }
         }
       }
  else if (proses[r][1] == 5 && proses[r-6][1] == 5
  && proses[r][2] == proses[r-6][2]){
    if(proses[r-42][1] == 69 && proses[r-42][2] ==
    69 || proses[r-42-6][1] == 69 && proses[r-42-
    6][2] == 69){
      for(int j=125; j>=6; j--){
        if (CLL[j][1] != 99 && CLL[j][1] != 1 &&
        CLL[j][1] != 2 && CLL[j][1] != 3 &&
```

```
        CLL[j][1] != 4 && CLL[j][1] != 5 &&
        CLL[j][1] != 6){
          if (CLL[j-6][1] != 99 && CLL[j-6][1] != 1
          && CLL[j-6][1] != 2 && CLL[j-6][1] != 3 &&
          CLL[j-6][1] != 4 && CLL[j-6][1] != 5 &&
          CLL[j-6][1] != 6){
             CLL[j-6][0] = proses[r-6][0]*2;
             CLL[j-6][1] = proses[r-6][1];
             CLL[j-6][2] = proses[r-6][2];
             CLL[j][0] = 0;
             CLL[j][1] = proses[r][1];
             CLL[j][2] = proses[r][2];
             proses[r][1] = 69;
             proses[r][2] = 69;
             proses[r][0] = 0;
             proses[r-6][1] = 69;
             proses[r-6][2] = 69;
             proses[r-6][0] = 0;
             }
          }
        }
      }
    }
  else if (proses[r][1] == 6 && proses[r-6][1] == 6
  && proses[r][2] == proses[r-6][2]){
    if(proses[r-42][1] == 69 && proses[r-42][2] ==
    69 || proses[r-42-6][1] == 69 && proses[r-42-
    6][2] == 69){
      for(int j=125; j>=6; j--){
        if (CLL[j][1] != 99 && CLL[j][1] != 1 &&
        CLL[][1] != 2 && CLL[j][1] != 3 &&
        CLL[j][1] != 4 && CLL[j][1] != 5 &&
        CLL[j][1] != 6){
          if (CLL[j-6][1] != 99 && CLL[j-6][1] != 1
          && CLL[j-6][1] != 2 && CLL[j-6][1] != 3
          && CLL[j-6][1] != 4 && CLL[j-6][1] != 5
          && CLL[j-6][1] != 6){
             CLL[j-6][0] = proses[r-6][0]*2;
             CLL[j-6][1] = proses[r-6][1];
             CLL[j-6][2] = proses[r-6][2];
             CLL[j][0] = 0;
             CLL[j][1] = proses[r][1];
             CLL[j][2] = proses[r][2];
             proses[r][1] = 69;
             proses[r][2] = 69;
             proses[r][0] = 0;
             proses[r-6][1] = 69;
             proses[r-6][2] = 69;
             proses[r-6][0] = 0;
```

```
                                    }
                                }
                            }
                        }
                    }
                }
}
```

BAB V
UJI COBA DAN PEMBAHASAN

Pada bab ini akan dibahas terkait dengan pengujian program dan hasil dari uji coba. Pengujian dilakukan dengan menggunakan *casebase* dan *Container Loading List* yang telah didapat dari tahap *Retrieve*.

5.1. Data Uji Coba

Terdapat dua data uji coba yang digunakan dalam penelitian ini. Data uji coba pertama digunakan untuk mengetahui pengaruh populasi casebase dengan jumlah swap manual yang perlu dilakukan. Sedangkan data uji coba kedua digunakan untuk mengetahui pengaruh restow dan swap program dengan jumlah isi peti kemas di dalam case yang digunakan.

- **Data Uji Coba Pertama**

Data uji coba pertama ini digunakan untuk mengetahui pengaruh jumlah populasi casebase terhadap jumlah swap manual dalam pembuatan *stowage planning* pada *query*.

Tabel 5. 1 Voyage Casebase 1

Voyage CLL 1	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	0	0	82	82
P2	40	42	50	92
P3	53	39	47	86
P4	57	29	55	84

Tabel 5. 2 Voyage Casebase 2

Voyage CLL 2	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	39	45	29	74
P2	45	29	48	77
P3	44	33	51	84
P4	50	34	38	72

Tabel 5. 3 Voyage Casebase 3

Voyage CLL 3	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	34	38	56	94
P2	38	56	30	86
P3	43	43	49	92
P4	43	49	39	88

Tabel 5. 4 Voyage Casebase 4

Voyage CLL 4	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	44	44	55	99
P2	44	55	32	87
P3	40	47	47	94
P4	64	30	65	95

➢ **Data Uji Coba Kedua**

Data uji coba kedua ini digunakan untuk mengetahui pengaruh jumlah *restow* dan *swap program* pada *query* atau CLL baru terhadap jumlah isi peti kemas yang termuat di dalam case.

- Casebase

Tabel 5. 5 Voyage untuk Case 1 #0

CASE 1 #0		TO			
		P1	P2	P3	P4
FROM	P1	0	58	20	22
	P2	0	0	41	17
	P3	29	10	0	22
	P4	30	31	0	0

	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	0	0	100	100
P2	58	42	58	100
P3	61	39	61	100
P4	61	39	61	100

Tabel 5. 6 Voyage untuk Case 1 #1

CASE 1 #1		TO			
		P1	P2	P3	P4
FROM	P1	0	40	20	22
	P2	0	0	33	17
	P3	19	10	0	18
	P4	20	35	0	0

	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	0	0	82	82
P2	40	42	50	92
P3	53	39	47	86
P4	57	29	55	84

Tabel 5. 7 Voyage untuk Case 1 #2

CASE 1 #2		TO			
		P1	P2	P3	P4
FROM	P1	0	0	20	22
	P2	0	0	13	17
	P3	19	10	0	18
	P4	20	15	0	0

	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	0	0	42	42
P2	0	42	30	72
P3	33	39	47	86
P4	57	29	35	64

Tabel 5. 8 Voyage untuk Case 1 #3

CASE 1 #3		TO			
		P1	P2	P3	P4
FROM	P1	0	0	10	22
	P2	0	0	13	5
	P3	5	10	0	0
	P4	10	15	0	0

	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	0	0	32	32
P2	0	32	18	50
P3	23	27	15	42
P4	27	15	25	40

Tabel 5. 9 Voyage untuk Case 1 #4

CASE 1 #4		TO			
		P1	P2	P3	P4
FROM	P1	0	0	5	5
	P2	0	0	3	2
	P3	4	5	0	0
	P4	3	7	0	0

	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	0	0	10	10
P2	0	10	5	15
P3	8	7	9	16
P4	7	9	10	19

- Container Loading List

Tabel 5. 10 Voyage untuk CLL 1

Voyage CLL1		TO				SUM
		P1	P2	P3	P4	
FROM	P1	0	35	29	22	226
	P2	0	0	29	17	
	P3	31	10	0	0	
	P4	20	33	0	0	

	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	0	0	86	86
P2	35	51	46	97
P3	58	39	41	80
P4	39	41	53	94

Tabel 5. 11 Voyage untuk CLL 2

Voyage CLL2		TO				SUM
		P1	P2	P3	P4	
FROM	P1	0	35	30	26	211
	P2	0	0	21	17	
	P3	28	10	0	0	
	P4	20	24	0	0	

	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	0	0	91	91
P2	35	56	38	94
P3	51	43	38	81
P4	43	38	44	82

Tabel 5. 12 Voyage untuk CLL 3

Voyage CLL3		TO				SUM
		P1	P2	P3	P4	
FROM	P1	0	35	24	22	233
	P2	0	0	28	23	
	P3	24	10	0	18	
	P4	20	29	0	0	

	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	0	0	81	81
P2	35	46	51	97
P3	52	45	52	97
P4	63	34	49	83

Tabel 5. 13 Voyage untuk CLL 4

Voyage CLL4		TO				SUM
		P1	P2	P3	P4	
FROM	P1	0	35	30	32	223
	P2	0	0	15	17	
	P3	27	10	0	0	
	P4	30	27	0	0	

	Ctr discharged	Ctrs prior to loading	Ctrs to be loaded	ON BOARD
P1	0	0	97	97
P2	35	62	32	94
P3	45	49	37	86
P4	49	37	57	94

5.2. Proses Uji Coba

Pada bagian ini akan dilakukan proses uji coba terhapad perangkat lunak yang dibangun. Terdapat dua uji coba, yaitu uji coba penataan secara otomatis dan penataan secara manual

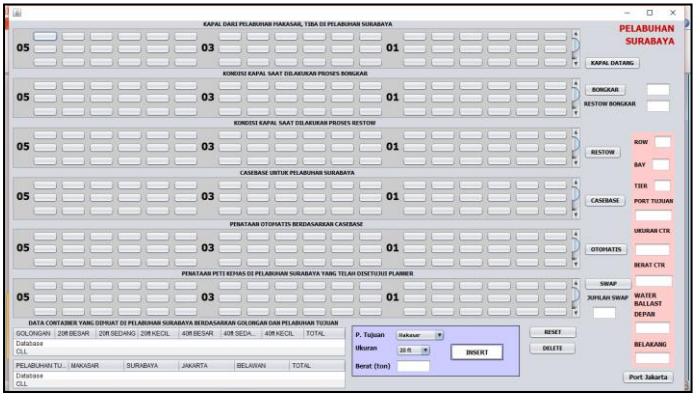
5.2.1. Tampilan Awal *Interface* Penataan Otomatis

Pada tampilan awal *interface* terdapat beberapa *tools* yang dibutuhkan untuk mempermudah pengguna dalam pengoperasian perangkat lunak. *Tools* tersebut diantaranya:

1. Ilustrasi tampilan tatanan peti kemas dari *casebase Container Loading List*, kedatangan kapal, proses bongkar dan proses *restow.*
2. Tombol 'Casebase', untuk menampilkan ilustrasi hasil penataan peti kemas pada *casebase*
3. Tombol 'Otomatis' untuk menampilkan ilustrasi hasil penataan peti kemas pada *Container Loading List* baru berdasarkan *casebase* dan pentaan yang dapat diubah-ubah sesuai kemauan *planner.*
4. Tombol 'Kapal Datang' untuk menampilkan ilustrasi penataan peti kemas dari pelabuhan sebelumnya

5. Tombol ‘Bongkar’ untuk menghapus peti kemas yang memiliki data pelabuhan tujuan sama dengan pelabuhan yang akan diproses
6. Informasi jumlah peti kemas yang dibongkar di pelabuhan tersebut dapat dilihat pada *text field* ‘bongkar’
7. Informasi jumlah peti kemas yang terkena *restow* pada *text field ‘restow’*
8. Tombol ‘*Restow*’ untuk menghapus peti kemas yang terkena *restow* pada ilustrasi penataan peti kemas saat kapal datang
9. Tombol ‘Reset’ untuk menghapus ilutrasi hasil penataan *Container Loading List* baru dan menghapus informasi pada *text field* yang berisi ‘*row*’, ‘*bay’, ‘tier’,* ‘berat ctr’, ‘ukuran ctr’ dan ‘port tujuan’
10. Tombol ‘Swap’ untuk melakukan penukaran lokasi dua atau empat peti kemas yang dipilih pada ilustrasi hasil tatanan *Container Loading List* baru
11. Informasi terkait banyak *planner* melakukan swap dapat dilihat pada *text field* ‘jumlah swap’.
12. Informasi lokasi peti kemas terdiri dari ‘*row*’, ‘*bay*’, ‘t*ier*’ dan data peti kemas yang terdiri dari ‘ukuran ctr’, ‘berat ctr’ dan ‘port tujuan’dapat diakses dengan memilih peti kemas pada ilustrasi penataan.
13. Tabel ‘Data Container’ dan ‘Data Pelabuhan Tujuan’ untuk menampilkan rincian peti kemas pada *casebase* dan *Container Loading List*

14. Tombol ‘Delete’ digunakan untuk menghapus peti kemas yang telah dipilih pada ilustrasi penataan peti kemas *Container Loading List.*
15. Tombol ‘Insert’ untuk menambahkan peti kemas yang tidak termuat pada *Container Loading List* dan ditata dalam penataan peti kemas *Container Loading List.*
16. Tombol ‘Port Selanjutnya’ untuk mengirim penataan pada *Container Loading List* ke pelabuhan selanjutnya dan mulai melakukan proses bongkar muat peti kemas di pelabuhan selanjutnya.

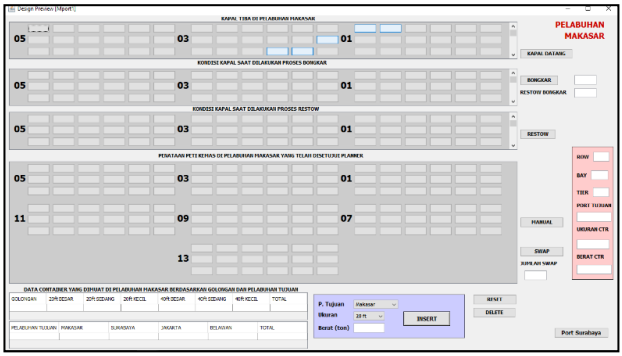


Gambar 5. 1 Tampilan Awal Interface Penataan Otomatis

5.2.2. Tampilan *Interface* Penataan Manual

Tampilan untuk penataan manual tidak jauh berbeda dengan penataan otomatis. Hanya saja pada penataan manual tidak terdapat tampilan ilustrasi *casebase* karena memang tidak dibutuhkan. *Tools* tambahan yang

tidak terdapat pada *interface* lainnya adalah tombol ‘Manual’ yang berfungsi untuk menampilkan ilustrasi penataan peti kemas *Container Loading List*



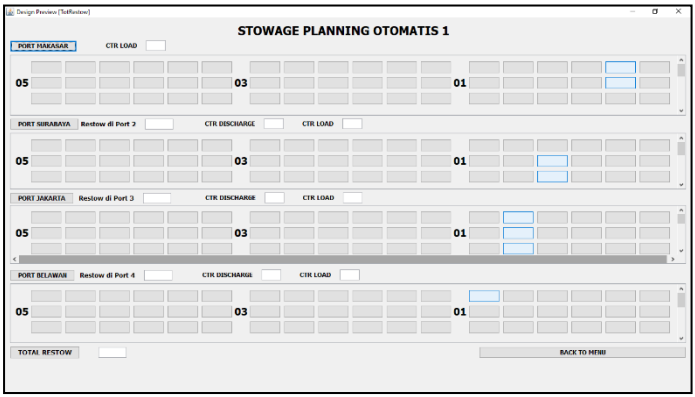
Gambar 5. 2 Tampilan interface untuk penataan manual

5.2.3. Tampilan Interface Output

Tampilan *interface output* berfungsi untuk menampilkan seluruh penataan dalam satu rute perjalanan dan menghitung total *restow* dalam satu rute perjalanan. Berikut *tools* yang terdapat pada *interface output* :

1. Ilustrasi tampilan penataan peti kemas pada empat pelabuhan yang dilalui dalam satu rute perjalanan
2. Tombol ‘Port Makasar’, ‘Port Surabaya’, ‘Port Jakarta’ dan ‘Port Belawan’ untuk menampilkan ilustrasi penataan peti kemas pada masing-masing pelabuhan
3. Informasi jumlah peti kemas yang terkena *restow* di masing-masing pelabuhan ditampilkan pada ‘Restow di Port 2’, ‘Restow di Port 3’ dan ‘Restow di Port 4’

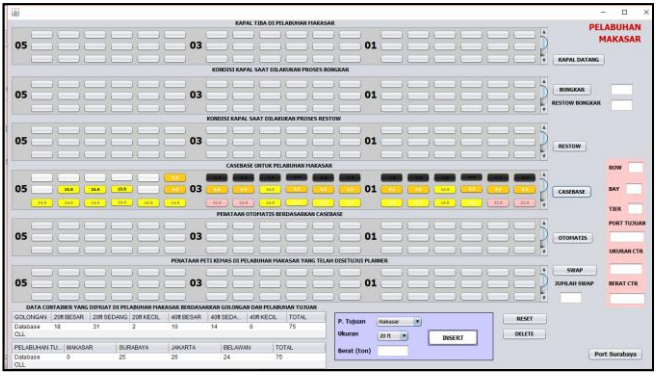
4. Tombol ‘Total Restow’ untuk menampilkan jumlah seluruh peti kemas yang terkena *restow* dalam satu rute perjalanan



Gambar 5. 3 Tampilan interface output

5.2.4. Uji Coba Tombol *Casebase*

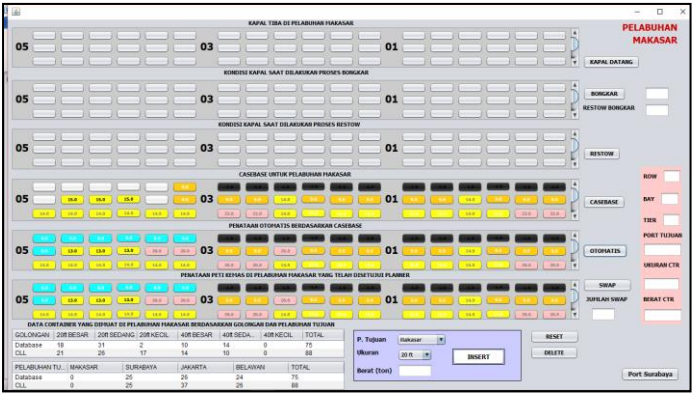
Tombol ‘Casebase’ berfungsi untuk menampilkan ilustrasi tatanan peti kemas pada *casebase* dan melakukan perhitungan kestabilan tatanan peti kemas serta menampilkan hasil perhitungan kestabilan dalam tabel Data Stabilitas



Gambar 5. 4 Uji coba tombol ‘Casebase’

5.2.5. Uji Coba Tombol Otomatis

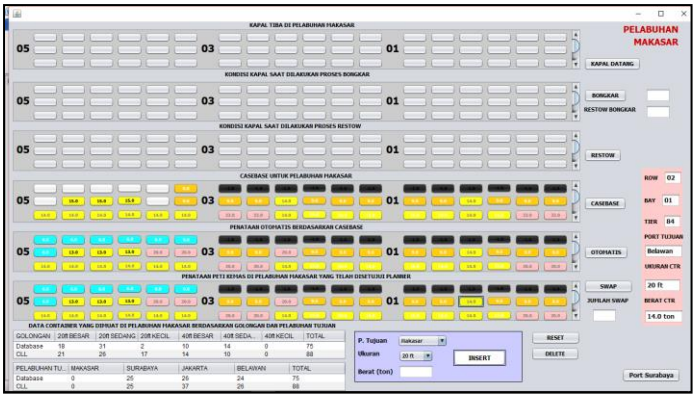
Tombol ‘Otomatis’ berfungsi untuk menampilkan ilustrasi tatanan peti kemas pada *Container Loading List* yang telah melalui proses, melakukan perhitungan kestabilan tatanan peti kemas serta menampilkan hasil perhitungan kestabilan dalam tabel Data Stabilitas



Gambar 5. 5 Uji coba tombol ‘Otomatis’

5.2.6. Uji Coba *Toogle Button* Peti Kemas

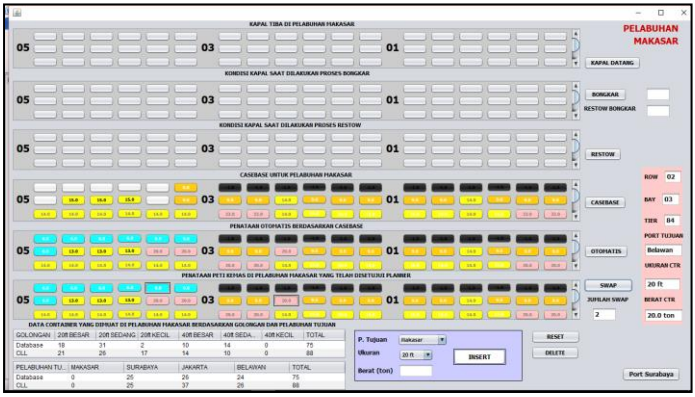
Apabila *toogle button* pada peti kemas dipilih, akan muncul lokasi peti kemas pada kapal dan rincian data peti kemas, seperti berat peti kemas, golongan berdasarkan berat dan pelabuhan tujuan peti kemas dikirim.



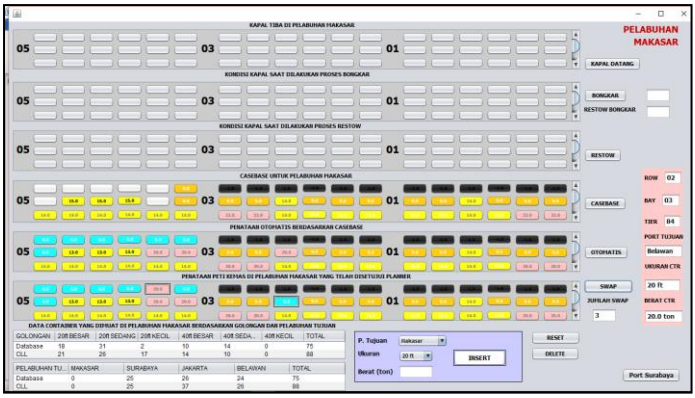
Gambar 5. 6 Uji coba toogle button peti kemas

5.2.7. Uji Coba Tombol Swap

Tombol ‘Swap’ berfungsi untuk menukar peti kemas yang telah dipilih. Untuk penukaran sesama peti kemas 20*ft* harus memilih dua peti kemas, namun untuk penukaran peti kemas berukuran 40*ft* harus memilih empat peti kemas.



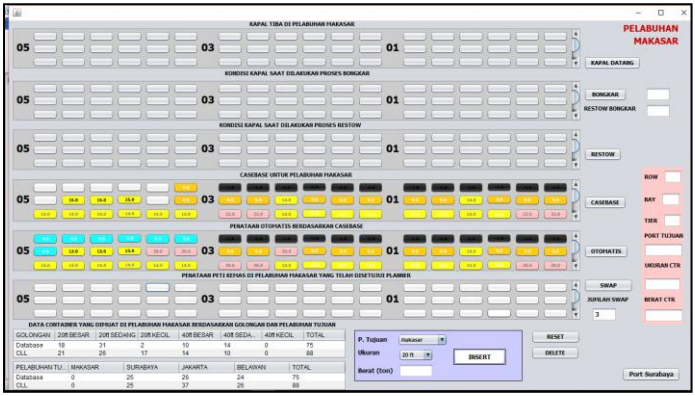
Gambar 5. 7 Uji coba tombol ‘swap’ – kondisi peti kemas sebelum ditukar



Gambar 5. 8 Uji coba tombol ‘swap’ – kondisi peti kemas setelah dilakukan penukaran

5.2.8. Uji Coba Tombol Reset

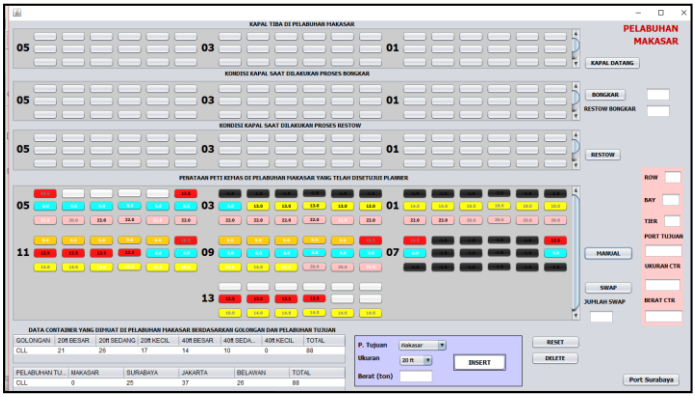
Tombol ‘Reset’ digunakan untuk menghapus ilustrasi penataan peti kemas dari *Container Loading List* dan data *water ballast* serta informasi peti kemas seperti ‘*row’, ‘bay’, ‘tier’, ‘port* tujuan’, ‘ukuran ctr’ dan ‘berat ctr’.



Gambar 5. 9 Uji coba tombol ‘reset’

5.2.9. Uji Coba Tombol Manual

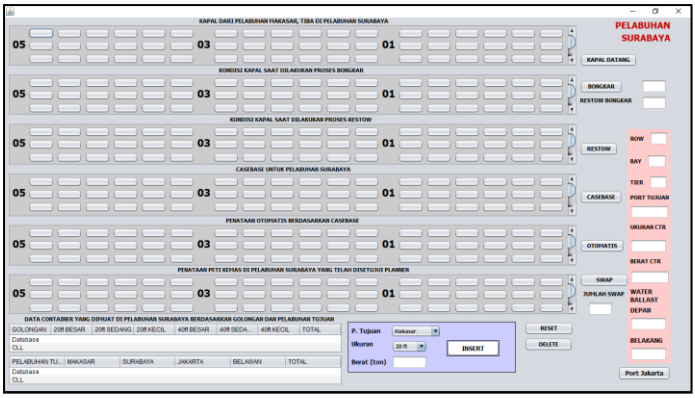
Tombol ‘Manual’ berfungsi untuk menampilkan ilustrasi tatanan peti kemas pada *Container Loading List* yang ditata sesuai dengan berat peti kemas, melakukan perhitungan kestabilan tatanan peti kemas serta menampilkan hasil perhitungan kestabilan dalam tabel Data Stabilitas



Gambar 5. 10 Uji coba tombol ‘Manual’

5.2.10. Uji Coba Tombol Port Pelabuhan

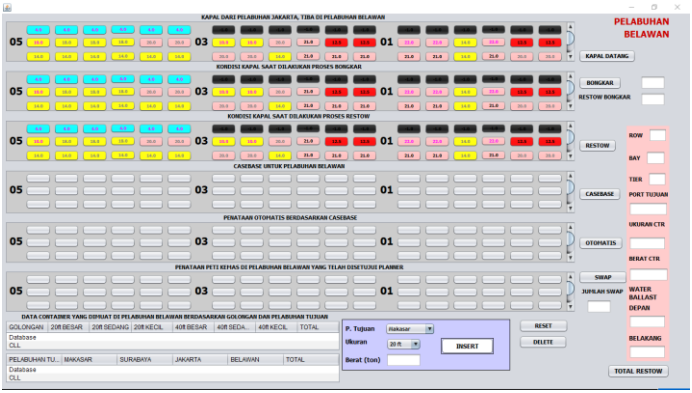
Tombol ‘Port Surabaya’, ‘Port Jakarta’ atau ‘Port Belawan’ yang berada di masing-masing pelabuhan berfungsi untuk mengirim hasil penataan peti kemas ke pelabuhan selanjutnya sesuai dengan nama pelabuhan yang ada pada tombol. Sebagai contoh pada gambar dibawah ini, setelah tombol ‘Port Surabaya’ dipilih, maka *user* masuk ke pelabuhan Surabaya



Gambar 5. 11 Uji coba tombol 'Port Pelabuhan'

5.2.11. Uji Coba Tombol Kapal Datang

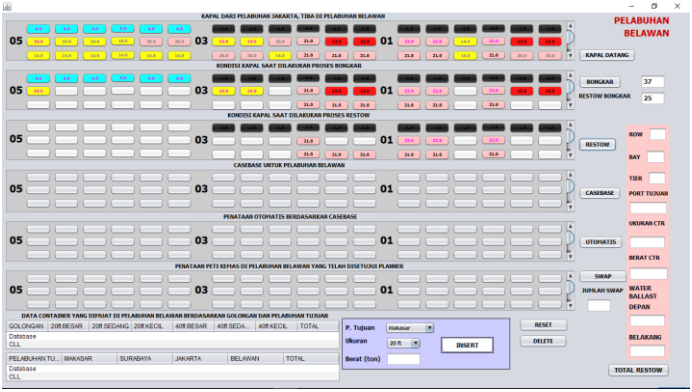
Tombol 'Kapal Datang' berfungsi untuk menampilkan ilustrasi penataan peti kemas dari pelabuhan sebelumnya



Gambar 5. 12 Uji coba tombol 'Kapal Datang'

5.2.12. Uji Coba Tombol Bongkar

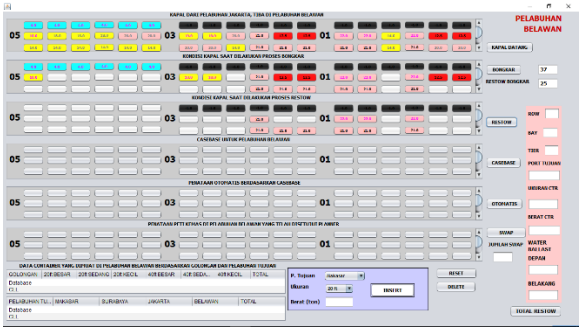
Tombol ‘Bongkar’ berfungsi untuk menghapus peti kemas yang memiliki data pelabuhan tujuan sesuai dengan pelabuhan yang ditampilkan atau yang sedang diakses dan menampilkan jumlah peti kemas yang mengalami *restow.*



Gambar 5. 13 Uji coba tombol ‘Bongkar’

5.2.13. Uji Coba Tombol Restow

Tombol ‘Restow berfungsi untuk menghapus peti kemas yang terkena *restow* dan memasukkannya ke dalam *Container Loading List* di Pelabuhan yang akan diproses.

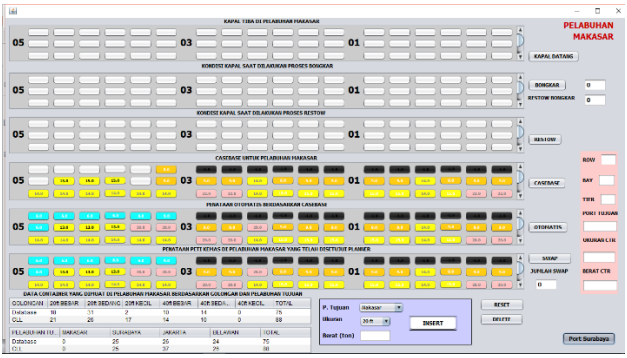


Gambar 5. 14 Uji coba tombol ‘Restow’

5.2.14. Uji Coba Proses Setiap Pelabuhan

Berikut ini ditampilkan proses penataan peti kemas secara otomatis untuk masing-masing pelabuhan tanpa dilakukan pertukaran peti kemas di semua pelabuhan.

- Pelabuhan Makasar

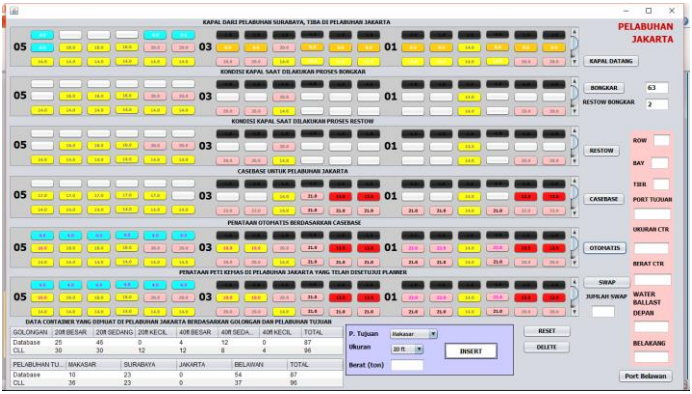


Gambar 5. 15 Penataan Otomatis di Pelabuhan Makasar

- Pelabuhan Surabaya

Gambar 5. 16 Penataan Otomatis di Pelabuhan Surabaya

- Pelabuhan Jakarta



Gambar 5. 17 Penataan Otomatis di Pelabuhan Jakarta

- Pelabuhan Belawan

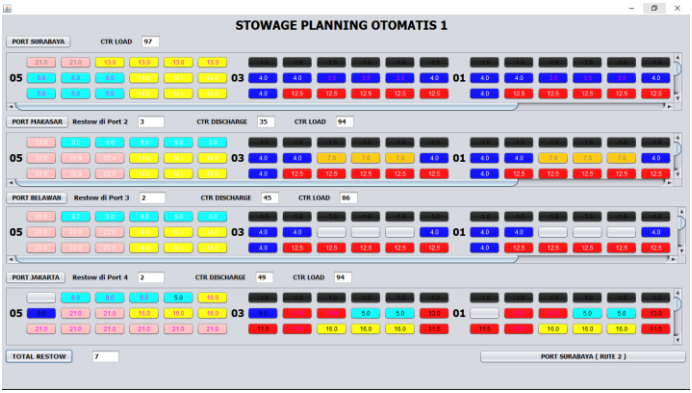
Gambar 5. 18 Penataan Otomatis di Pelabuhan Belawan

5.2.15. Uji Coba *Output*

Pada tampilan *output*, ditampilkan seluruh penataan dalam satu rute perjalanan dan menghitung total *restow* dalam satu rute perjalanan. Sehingga, data *output*

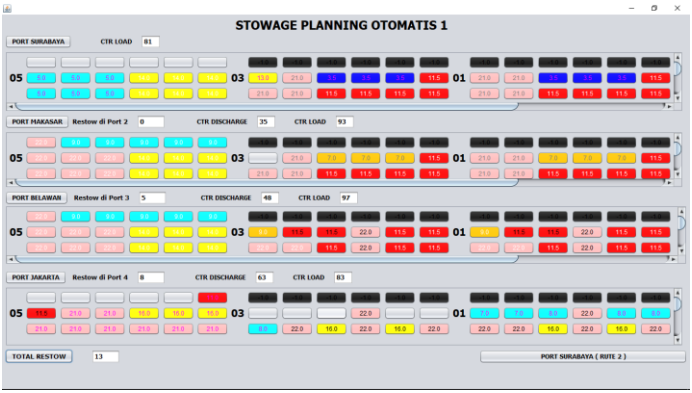
ini yang digunakan untuk penataan peti kemas pada keempat pelabuhan dalam satu rute perjalanan. Berikut beberapa contoh output untuk penataan otomatis

- Penataan Otomatis CB 1 – CLL 4



Gambar 5. 19 Output penataan peti kemas pada penataan otomatis CLL 4 berdasarkan casebase 1

- Penataan Otomatis CB 2 – CLL 3



Gambar 5. 20 Output penataan peti kemas pada penataan otomatis CLL 3 berdasarkan casebase 2

5.3. Pembahasan Hasil Uji Coba

Proses pengujian telah dilakukan untuk 4 CLL yang disesuaikan dengan 5 macam casebase dengan populasi casebase 1 paling banyak diantara yang lain dan casebase 5 memiliki populasi terkecil diantara yang lain. Dihasilkan data restow, swap serta jumlah muatan yang berbeda-beda.

- Hasil Uji Coba Pertama

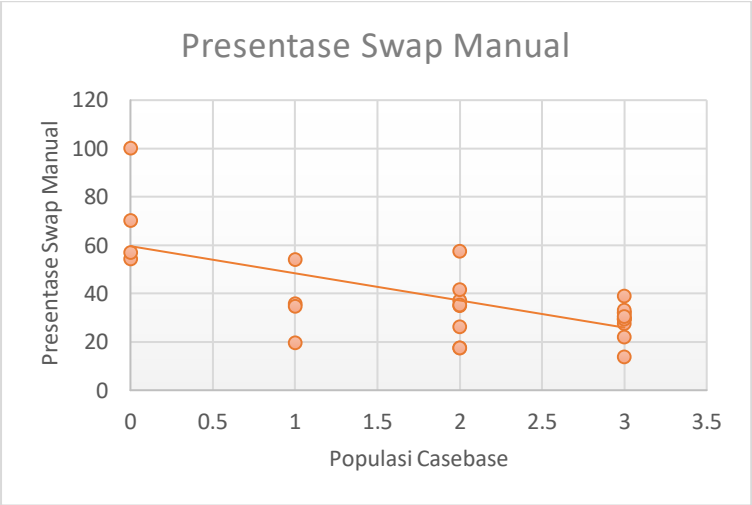
Pada uji coba pertama ini digunakan data uji coba pertama untuk mengetahui pengaruh jumlah populasi casebase terhadap swap manual yang perlu dilakukan oleh *planner* dalam membuat *stowage planning* untuk *query* atau CLL baru

Tabel 5. 14 Data Swap Manual vs Populasi Casebase

No	NO voyage case	No Voyage Query	JUMLAH SWAP MANUAL				Presentase Swap Manual (Jumlah Swap Manual / Query(CLL) Sekarang x 100%)				
			Sby-Mks	Mks-Blw	Blw-Jkt	Jkt-Sbt	Sby-Mks	Mks-Blw	Blw-Jkt	Jkt-Sbt	Rata-rata
1	0	1	82	50	47	59	100	54,35	54,7	70,2	69,81
2	1	2	40	15	30	25	54,05	19,48	35,7	34,7	35,99
3	1	3	54	15	34	23	57,45	17,44	37	26,1	33,37
4	2	3	39	15	32	31	41,49	17,44	34,8	35,2	
5	1	4	29	19	30	29	29,29	21,84	31,9	30,5	28,99
6	2	4	29	24	30	37	29,29	27,59	31,9	38,9	
7	3	4	29	12	31	29	29,29	13,79	33	30,5	

Tabel 5.14 merupakan data hasil uji pengaruh dari populasi casebase terhadap swap manual yang terjadi dalam membuat *stowage planning* pada *query* baru. Swap manual didefinisikan sebagai penataan peti kemas yang perlu dilakukan secara manual karena tidak ada jenis/golongan peti kemas yang sama pada

case. Swap manual dihitung dari jumlah peti kemas yang tersisa di CLL setelah dilakukan penataan peti kemas yang memiliki jenis yang sama dengan case.



Gambar 5. 21 Grafik Swap Manual vs Populasi Casebase

Pada Gambar 5.21, dapat dilihat bahwa semakin besar atau semakin kaya casebase, maka presentase swap manual semakin sedikit dikarenakan populasi pada casebase semakin banyak, sehingga dalam pembuatan *stowage planning* baru, *planner* dapat memilih lebih banyak case yang digunakan sebagai contoh dalam pembuatan *stowage planning* baru. Sedangkan, jika *casebase* sedikit maka swap manual semakin besar dikarenakan case yang ada pada casebase terbatas dan belum tentu cocok dengan *query* yang baru.

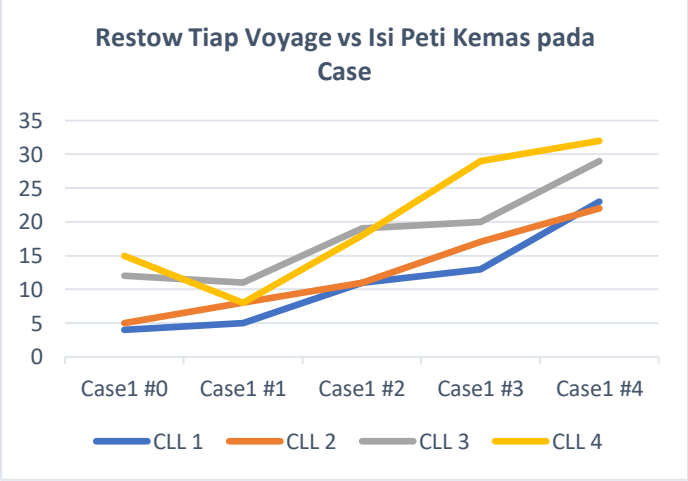
Dengan besarnya populasi casebase, maka kinerja planner semakin ringan, karena planner dapat memilih case yang paling mirip dengan *query* sehingga meminimalisir swap manual yang perlu dilakukan.

➢ Hasil Uji Coba Kedua

Pada uji coba kedua ini digunakan data uji coba kedua untuk mengetahui pengaruh jumlah *restow* dan jumlah swap program terhadap jumlah isi peti kemas pada case yang digunakan.

- Data Peti Kemas yang terkena restow di setiap port

Restow peti kemas terjadi karena adanya proses bongkar di pelabuhan dan pada program dihitung berdasarkan unit bukan TEUS.



Gambar 5. 22 Grafik Restow tiap Voyage vs Isi Peti Kemas pada Case

Grafik pada Gambar 5.22 diperoleh dari data pada Tabel 5.15 dibawah ini

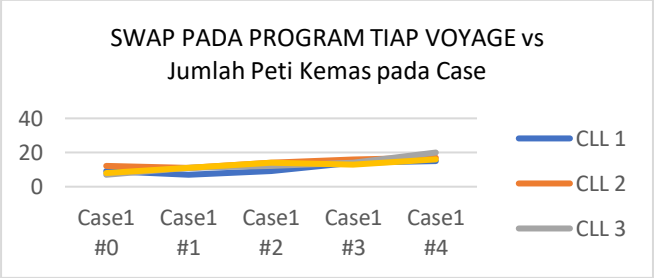
Tabel 5. 15 Data Peti Kemas Restow

No	Nomer CLL	KODE CASE BASE	JUMLAH RESTOWS PER RUAS				Jml Restow / voy
			Sby - Mks	Mks-Blw	Blw -Jkt	Jkt-Sbt	
1	1	0	-	-	3	1	4
2	2	0	-	2	-	3	5
3	3	0	-	-	9	3	12
4	4	0	-	3	3	9	15
5	1	1	-	1	-	4	5
6	2	1	-	4	3	1	8
7	3	1	-	-	7	4	11
8	4	1	-	-	5	3	8
9	1	2	-	6	3	2	11
10	2	2	-	5	6	-	11
11	3	2	-	4	6	9	19
12	4	2	-	7	9	2	18
13	1	3	-	5	3	5	13
14	2	3	-	9	4	4	17
15	3	3	-	2	5	13	20
16	4	3	-	16	13	-	29
17	1	4	-	6	6	11	23
18	2	4	-	12	3	7	22
19	3	4	-	1	10	18	29
20	4	4	-	15	11	6	32

Pada Gambar 5.22 dapat dilihat bahwa Case1 #0 yang memiliki jumlah peti kemas terbanyak pada case dibandingkan Case1 #1 hinggan Case1 #4 dapat mempengaruhi jumlah *restow* dalam satu *voyage* rata-rata 9 peti kemas untuk empat *query* yang dilakukan pengujian. Sedangkan, untuk Case1 #4 yang memiliki jumlah peti kemas paling sedikit diantara yang lain mempengaruhi jumlah *restow* dalam satu *voyage* rata-rata 27 peti kemas. Sehingga, dapat disimpulkan bahwa semakin banyak jumlah peti kemas pada case, semakin kecil *restow* yang terjadi dalam satu *voyage.* Begitupun sebaliknya, semakin sedikit jumlah peti kemas pada case, maka contoh yang diberikan untuk membuat *stowage planning* pada *query* semakin sedikit sehingga *restow* yang terjadi dalam satu *voyage* semakin besar.

- Data Peti Kemas Swap Program

  Swap program didefinisikan sebagai proses pertukaran atau pemindahan peti kemas pada hasil tatanan peti kemas di program oleh *planner* atau *user* karena hasil tatanan yang belum memenuhi syarat penataan peti kemas. Hal ini dilakukan supaya penataan memenuhi syarat sehingga dapat melanjutkan penataan untuk pelabuhan selanjutnya.



Gambar 5. 23 Grafik Swap Program vs Jumlah Peti Kemas pada Case

Tabel 5. 16 Data Swap Program

No	Nomer CLL	KODE CASEBASE	JUMLAH SWAP PER RUAS SETELAH DIKENAI ALGORITMA				Jml swap/voyage
			Sby-Mks	Mks-Blw	Blw-Jkt	Jkt-Sbt	
1	1	0	3	2	2	2	9
2	2	0	4	2	3	3	12
3	3	0	1	2	1	3	7
4	4	0	4	2	-	2	8
5	1	1	3	2	2	-	7
6	2	1	4	3	2	2	11
7	3	1	4	1	3	3	11
8	4	1	4	3	3	1	11
9	1	2	6	1	2	-	9
10	2	2	7	5	-	2	14
11	3	2	5	2	3	2	12
12	4	2	8	4	2	-	14
13	1	3	9	1	4	-	14
14	2	3	8	2	3	3	16
15	3	3	8	2	2	2	14
16	4	3	9	2	2	-	13
17	1	4	11	2	2	-	15
18	2	4	13	1	2	1	17
19	3	4	13	3	2	2	20
20	4	4	15	-	1	-	16

Pada Gambar 5.23 dapat dilihat bahwa Case1 #0 yang memiliki jumlah peti kemas terbanyak pada case dibandingkan Case1 #1 hinggan Case1 #4 dapat mempengaruhi jumlah swap program dalam satu *voyage* rata-rata 9 peti kemas untuk empat *query* yang dilakukan pengujian. Sedangkan, untuk Case1 #4 yang memiliki jumlah peti kemas paling sedikit diantara yang lain mempengaruhi jumlah swap program dalam satu *voyage* rata-rata 17 peti kemas. Sehingga, dapat disimpulkan bahwa semakin banyak jumlah peti kemas pada case, semakin kecil swap program yang terjadi dalam satu *voyage*. Begitupun sebaliknya, semakin sedikit jumlah peti kemas pada case, maka contoh yang diberikan untuk membuat *stowage planning* pada *query* semakin sedikit sehingga swap program yang terjadi dalam satu *voyage* semakin besar.

- **Analisis Hasil Uji**

Setelah dilakukan uji coba pada menggunakan data uji pertama dan data uji kedua maka hasil dari tabel dan grafik diatas dapat dianalisa bahwa:

1. Rangkaian algoritma pada tahap *Reuse* menghasilkan penataan peti kemas yang telah memenuhi syarat penataan peti kemas pada kapal dimana tidak ada peti kemas yang melayang atau peti kemas 20*ft* yang berada diatas peti kemas 40*ft*.
2. Rangkaian algoritma pada tahap *Reuse* dapat melakukan bongkar muat peti kemas di setiap pelabuhan. Namun, tidak semua peti kemas pada CLL dapat termuat karena bergantung dengan case yang digunakan.
3. Berdasarkan grafik pada Gambar 5.21, terlihat bahwa semakin besar populasi pada casebase, maka jumlah swap manual atau beban kerja *planner* untuk menata peti

kemas secara manual semakin ringan karena contoh case yang digunakan banyak, sehingga dapat menentukan case mana yang paling mirip dengan *query* yang digunakan.

4. Semakin kecil populasi pada casebase, maka jumlah swap manual atau beban kinerja *planner* untuk menata peti kemas secara manual semakin berat karena contoh case yang digunakan sedikit dan belum tentu mirip dengan *query* yang digunakan.
5. Perbedaan jumlah peti kemas pada *case* mempengaruhi jumlah *restow* dan jumlah proses swap di setiap pelabuhan.
6. Berdasarkan grafik pada Gambar 5.22, rangkaian algoritma pada tahap *Reuse* ini menghasilkan jumlah *restow* yang kecil jika casebase yang digunakan memiliki jumlah peti kemas yang besar, sebaliknya jika casebase yang digunakan memiliki jumlah peti kemas yang sedikit, maka jumlah *restow* dalam satu *voyage* semakin besar.
7. Berdasarkan grafik pada Gambar 5.23, rangkaian algoritma yang digunakan dapat menghasilkan tatanan peti kemas yang teratur atau mendekati syarat penataan peti kemas jika *case* yang digunakan memiliki jumlah peti kemas yang besar, sehingga *planner* hanya perlu sedikit merevisi hasil tatanan untuk menjadi sempurna.
8. Jika *case* yang digunakan memiliki jumlah peti kemas yang sedikit maka rangkaian algoritma akan menghasilkan penataan peti kemas yang tidak rapi dan *planner* perlu banyak merevisi hasil tatanan peti kemas supaya memenuhi syarat penataan peti kemas dan ini membuat beban kinerja *planner* besar.

9. Case1 #4 yang memiliki jumlah peti kemas paling sedikit membuat hasil tatanan yang buruk karena menghasil jumlah *restow* yang banyak dalam satu *voyage* dan jumlah swap manual serta swap program yang banyak pula dalam satu *voyage*, sehingga beban kinerja *planner* berat.
10. Kesalahan dalam memilih lokasi penukaran peti kemas dapat mempengaruhi jumlah *restow* di setiap Pelabuhan.
11. Pengguna dapat melihat data peti kemas baru yang dibawa di setiap pelabuhan pada 'Tabel Data Kontainer' dan 'Tabel Pelabuhan Tujuan'
12. Pengguna dapat menghapus hasil tataan peti kemas dan melakukan proses swap program dari awal setalah rangkaian algoritma membentuk tatanan peti kemas jika ada proses yang ingin diulang.
13. Perangkat lunak dijalankan menggunakan *notebook* dengan prosesor Intel Core i3 dan *Random Access Memory* (RAM) 4GB.

BAB VI
PENUTUP

Pada bab ini, berisi kesimpulan yang diperoleh dari pembahasan pada bab sebelumnya serta saran untuk pengembangan penelitian ini.

6.1 Kesimpulan

Berdasarkan hasil uji coba program dan pembahasan pada bagian sebelumnya, data disimpulkan bahwa

1. Algoritma tahap *Resue* pada Metode *Case Based Reasoning* dapat diimplementasikan pada rancangan perangkat lunak untuk penataan peti kemas dengan rute *multiport.*
2. Perangkat lunak memudahkan *planner* dalam membuat penataan peti kemas untuk *Container Loading List* baru lebih cepat. Namun, hasil tatanan peti kemas yang baik dan buruk bergantung dari *casebase* yang digunakan.
3. Rangakaian algoritma yang diimplementasikan pada perangkat lunak sangat bergantung dengan banyaknya populasi *casebase* dan jumlah peti kemas pada *case*, semakin banyak jumlahnya, semakin baik hasil tatanan peti kemas yang dibuat.
4. Perangkat lunak memudahkan *planner* untuk mengetahui jumlah *restow* di setiap pelabuhan dan total *restow* dalam satu rute perjalanan.
5. Adanya *casebase* dapat mempengaruhi jumlah peti kemas yang terkena *restow*. Semakin besar jumlah peti kemas pada *case*, semakin kecil *restow* dalam satu *voyage*.

6. Adanya *casebase* dapat meringankan kinerja *planner* dalam membuat *stowage plan*, semakin banyak jumlah populasi c*asebase*, semakin ringan kinerja *planner*. Hal ini dilihat dari jumlah *swap* manual yang harus dilakukan *planner*. Pada hasil uji, jika populasi *casebase* tidak ada atau 0, maka rata-rata presentase *swap* manual diatas 70% dari jumlah peti kemas di kapal dan presentase ini semakin turun seiring dengan meningkatnya populasi *casebase*.

6.2 Saran

Pada penelitian ini terdapat beberapa saran yang diambil dari hasil penelitian. Berikut merupakan saran untuk pengembangan penelitian ini

1. Perlu dikembangkan algoritma baru dnegan metode lain agar ditemukan hasil penataan yang lebih baik dan sesuai dengan kebutuhan pengguna
2. Perlu dikembangkan untuk optimasi *trim* dan optimasi GM secara otomatis supaya dapat dilihat juga syarat kestabilan kapal sudah terpenuhi atau belum.
3. Perlu adanya pengembangan lebih lanjut dari rancangan perangkat lunak ini agar dapat diterapkan di kapal kontainer lainnya
4. Perlu adanya pengembangan penelitian lebih lanjut, seperti ketika bentuk peti kemas yang berbeda-beda dan ketika muatan kapal tidak hanya dalam bentuk padat.

DAFTAR PUSTAKA

[1] Ambrosino, D., Massimo, P., & Sciomachen, A. (2015). A MIP heuristic for multi port stowage planning. *Transportation Research Procedia, 10,* 725-734. doi:10.1016/j.trpro.2015.09.026.

[2] Ambrosino, D., Massimo, P., & Sciomachen, A. (2017). Computational evaluation of a MIP model for multi-port stowage planning problems. *Soft Comput, 21*, 1753-1763. doi:10.1007/s00500-015-1879-y

[3] Wilson, I., & Roach, P. (2000). Container stowage planning: a methodology for generating computerised solutions. *Journal of the Operational Reasearch Society, 51:11,* 1248-1255. doi:10.1057/palgrave.jors.2601022

[4] Nugroho, S. (2004). Case-based Stowage Planning for Container Ship. *International Logistic Congress* (pp. 609-618). Dokuz Eylul University, Izmir, Turkey: Dokuz Eylul Publication.

[5] Nugroho, S. (2005). CASESTOW: Recycling of Past Stowage Plans. *1st International Conference on Operations and Supply Chain Management.* Bali.

[6] Mulyana, S., & Hartati, S. (2009, Mei 23). Tinjauan Singkat Perkembangan Case-Based Reasoning. *Seminar Nasional Informatika 2009 (semnasIF 2009).* UPN "Veteran" Yogyakarta.

[7] Riesback, C., & Schank, R. (1989). Inside Case-Based Reasoning. Lawrence Erlbaum Associates, Inc.

[8] Harwanto, N. A. (2011). Pengembangan Prototipe Modul Sistem Perencanaan Stowage Berdasarkan Kasus untuk Penataan Semi-Otomatis Peti Kemas pada Kapal. *Tugas Akhir*.

[9] Pratama, A. (2019). Design and Application of Reuse Stage Algorithm of Case-Based Reasoning Method on Container Stowage Planning. In *Journal of Physics: Conference Series* (Vol. 1490, No. 1,p.012014). IOP Publishing

[10] Wilson, I., Roach, P., & Ware, J. (2001). Container stowage pre-planning: using search to generate solutions, a case study. *Knowledge Based System*, 137-145.

[11] Ambrosino, D., Paolucci, M., & Sciomachen, A. (2013). Experimental evaluation of mixed integer programming models for the multi-port master bay plan problem. *Flex Serf Manuf J*, 263-284. doi:10.1007/s10696-013-9185-4

[12] Main, J., Dillon, T., & Shiu, S. (2001). A Tutorial on Case Based Reasoning : Soft Computing in Case Based Reasoning (Eds). *Sprenger-Verlag*, 1-28.

[13] Lewis, E. (n.d.). Principle of Naval Architecture. *Second Revision Volume II. The Society of Naval Architecture and Marine Engineers*. 601 Pavonia Avenue Jersey City, NJ.

[14] Jahar, R. (2018). Studi Pengaruh Kondisi Inisial Terhadap Operasi Peluncuran Struktur Jacket.

Thesis. Departemen Teknik Kelautan, Institit Teknologi Sepuluh Nopember, Surabaya

[15] Sumaryanto. (2013). *Konsep Dasar Kapal SMK Kelas X Semester 1*. Kementerian Pendidikan & Kebudayaan Republik Indonesia.

[16] Jun, L., Yu, Z., Jie, M., & Sanyou, J. (2018). Multi-Port Stowage Planning for Inland Container Liner Shipping Considering Weight Uncertainties. *IEEE Access*, 66468-66480.

LAMPIRAN

Lampiran A

Tabel Terminologi atau Tabel Istilah

NO	Istilah	Deskripsi / Keterangan
1	Case Based Reasoning	Metode yang digunakan untuk mencari penyelesaian masalah dengan cara memanfaatkan pengalaman untuk masalah sebelumnya yang hampir sama dengan masalah yang sedang dibahas.
2	Casebase	Istilah ini khusus untuk database menggunakan metode Case Based Reasoning. Casebase adalah kumpulan dari case yang sudah pernah terjadi sebelumnya dan disimpan dalam database.
3	Case	Solusi yang telah dihasilkan dari masalah yang sudah ada sebelumnya. Berisi masalah dan solusi yang dihasilkan
4	Container Loading List (CLL)	Daftar Peti Kemas yang akan dimuat di pelabuhan, mengandung parameter asal dan pelabuhan tujuan peti kemas dikirim, ukuran peti kemas, berat peti kemas dan kode pengirim peti kemas.

5	Multiport	Rute pelayaran kapal ke banyak pelabuhan.
6	Populasi Casebase	Jumlah case yang tedapat pada casebase.
7	Query	Masalah baru yang akan dibahas atau diselesaikan menggunakan metode Case Based Reasoning. Berisi dua masalah, yaitu CLL dan kondisi kedatangan kapal.
8	Restow	Proses dimana terjadi penurunan kontainer dari kapal dan dimuat kembali ke kapal yang sama dalam satu lokasi pelabuhan atau terminal.
9	Reuse	Salah satu tahap pada metode Case Based Reasoning yang bertujuan menyelesaikan masalah baru dengan mengadaptasi solusi dari masalah sebelumnya yang memiliki kemiripan tertinggi dengan masalah baru.
10	Swap Manual	Penataan peti kemas yang perlu dilakukan secara manual karena tidak ada jenis/golongan peti kemas yang sama pada case. Swap manual dihitung dari jumlah peti kemas

		yang tersisa di CLL setelah dilakukan penataan peti kemas yang memiliki jenis yang sama dengan case.
11	Swap Program	Proses pertukaran atau pemindahan peti kemas pada hasil tatanan peti kemas di program oleh *planner* atau *user* karena hasil tatanan yang belum memenuhi syarat penataan peti kemas.
12	Voyage	Perjalanan kapal dalam satu putaran dihitung dari pelabuhan awal (P1) sampai pelabuhan paling akhir (Pn).

Lampiran B

Contoh Daftar Peti Kemas pada Casebase dan CLL

- Casebase 1

Kode pengirim	Asal Pelabuhan	Tujuan Pelabuhan	Berat	Ukuran(ft)	Jumlah	TEUs
1SB	Surabaya	Belawan	12	20	6	6
2SB	Surabaya	Belawan	20	20	4	4
1SB	Surabaya	Belawan	27	40	5	10
2SJ	Surabaya	Jakarta	28	40	8	16
1SJ	Surabaya	Jakarta	14	20	6	6
3MB	Makasar	Belawan	18	40	6	12
3MB	Makasar	Belawan	8	20	5	5
4MB	Makasar	Belawan	14	20	2	2
4MB	Makasar	Belawan	17	40	4	8
4MB	Makasar	Belawan	26	40	3	6
3MJ	Makasar	Jakarta	9	20	10	10
3MJ	Makasar	Jakarta	21	20	7	7
5BS	Belawan	Surabaya	20	20	7	7
6BS	Belawan	Surabaya	13	20	4	4
5BS	Belawan	Surabaya	23	40	4	8
7BM	Belawan	Makasar	14	20	5	5
6BM	Belawan	Makasar	8	20	5	5
6BJ	Belawan	Jakarta	24	40	8	16
7BJ	Belawan	Jakarta	13	20	2	2
8JS	Jakarta	Surabaya	23	40	3	6
8JS	Jakarta	Surabaya	16	20	6	6

9JS	Jakarta	Surabaya	17	40	4	8
8JM	Jakarta	Makasar	21	20	9	9
9JM	Jakarta	Makasar	12	20	6	6
1SM	Surabaya	Makasar	9	40	7	14
1SM	Surabaya	Makasar	5	20	8	8
2SM	Surabaya	Makasar	14	40	4	8
2SM	Surabaya	Makasar	23	20	5	5
1SM	Surabaya	Makasar	21	20	5	5
3SM	Surabaya	Makasar	24	40	6	12
3SM	Surabaya	Makasar	15	20	6	6
3MB	Makasar	Belawan	7	40	4	8
5BJ	Belawan	Jakarta	8	40	2	4
7BS	Belawan	Surabaya	22	20	10	10
9JS	Jakarta	Surabaya	5	40	3	6
9JS	Jakarta	Surabaya	5	20	4	4
8JM	Jakarta	Makasar	8	20	10	10
8JM	Jakarta	Makasar	27	40	3	6

- Container Loading List 1

Kode pengirim	Asal Pelabuhan	Tujuan Pelabuhan	Berat	Ukuran(ft)	Jumlah	TEUs
1SM	Surabaya	Makasar	7	40	3	6
1SM	Surabaya	Makasar	5	20	6	6
2SM	Surabaya	Makasar	17	40	4	8
2SM	Surabaya	Makasar	22	20	10	10
3SM	Surabaya	Makasar	13	20	5	5

1SB	Surabaya	Belawan	15	20	5	5
2SB	Surabaya	Belawan	20	20	4	4
1SB	Surabaya	Belawan	25	40	5	10
1SB	Surabaya	Belawan	21	20	10	10
2SJ	Surabaya	Jakarta	25	40	5	10
1SJ	Surabaya	Jakarta	14	20	12	12
3MB	Makasar	Belawan	5	20	5	5
4MB	Makasar	Belawan	17	20	2	2
4MB	Makasar	Belawan	14	40	4	8
4MB	Makasar	Belawan	23	40	3	6
3MB	Makasar	Belawan	7	40	4	8
3MJ	Makasar	Jakarta	9	20	8	8
3MJ	Makasar	Jakarta	22	20	9	9
5BS	Belawan	Surabaya	13	20	11	11
5BS	Belawan	Surabaya	23	40	5	10
7BS	Belawan	Surabaya	22	20	10	10
7BM	Belawan	Makasar	17	20	4	4
6BM	Belawan	Makasar	7	20	6	6
8JS	Jakarta	Surabaya	26	40	3	6
8JS	Jakarta	Surabaya	16	20	4	4
9JS	Jakarta	Surabaya	22	20	4	4
9JS	Jakarta	Surabaya	9	40	3	6
8JM	Jakarta	Makasar	21	20	10	10
9JM	Jakarta	Makasar	16	20	5	5
8JM	Jakarta	Makasar	8	20	10	10
8JM	Jakarta	Makasar	22	40	4	8

Lampiran C

Contoh salah satu source code casebase dan CLL

- Casebase 5 di Pelabuhan Makasar

```
        database[18][0] = -1;
        database[19][0] = -1;
        database[20][0] = -1;
        database[21][0] = -1;
        database[22][0] = -1;
        database[23][0] = -1;
        database[61][0] = -1;
        database[62][0] = -1;
        database[63][0] = -1;
        database[64][0] = -1;
        database[84][0] = -1;
        database[85][0] = -1;
        database[86][0] = -1;
        database[87][0] = -1;
        database[88][0] = -1;
        database[89][0] = -1;
        database[90][0] = -1;
        database[91][0] = -1;
        database[92][0] = -1;
        database[93][0] = -1;
        database[94][0] = -1;
        database[95][0] = -1;
        database[103][0] = -1;
        database[104][0] = -1;
        database[105][0] = -1;
        database[106][0] = -1;

        database[18][1] = 99;
        database[19][1] = 99;
        database[20][1] = 99;
        database[21][1] = 99;
        database[22][1] = 99;
        database[23][1] = 99;
```

```
        database[61][1] = 99;
        database[62][1] = 99;
        database[63][1] = 99;
        database[64][1] = 99;
        database[84][1] = 99;
        database[85][1] = 99;
        database[86][1] = 99;
        database[87][1] = 99;
        database[88][1] = 99;
        database[89][1] = 99;
        database[90][1] = 99;
        database[91][1] = 99;
        database[92][1] = 99;
        database[93][1] = 99;
        database[94][1] = 99;
        database[95][1] = 99;
        database[103][1] = 99;
        database[104][1] = 99;
        database[105][1] = 99;
        database[106][1] = 99;

        database[18][2] = 99;
        database[19][2] = 99;
        database[20][2] = 99;
        database[21][2] = 99;
        database[22][2] = 99;
        database[23][2] = 99;
        database[61][2] = 99;
        database[62][2] = 99;
        database[63][2] = 99;
        database[64][2] = 99;
        database[84][2] = 99;
        database[85][2] = 99;
        database[86][2] = 99;
        database[87][2] = 99;
        database[88][2] = 99;
        database[89][2] = 99;
        database[90][2] = 99;
        database[91][2] = 99;
```

```
        database[92][2] = 99;
        database[93][2] = 99;
        database[94][2] = 99;
        database[95][2] = 99;
        database[103][2] = 99;
        database[104][2] = 99;
        database[105][2] = 99;
        database[106][2] = 99;

        database[28][2] = 4;
        database[29][2] = 4;
        database[71][2] = 4;
        database[35][2] = 4;
        database[41][2] = 4;

        database[28][1] = 2;
        database[29][1] = 2;
        database[71][1] = 2;

        database[28][0] = 20;
        database[29][0] = 20;
        database[71][0] = 20;

        database[35][1] = 4;
        database[41][1] = 4;

        database[35][0] = 27;
        database[41][0] = 0;

        database[12][2] = 3;
        database[13][2] = 3;
        database[14][2] = 3;
        database[54][2] = 3;
        database[55][2] = 3;

        database[12][1] = 2;
        database[13][1] = 2;
        database[14][1] = 2;
        database[54][1] = 2;
```

```
        database[55][1] = 2;

        database[12][0] = 14;
        database[13][0] = 14;
        database[14][0] = 14;
        database[54][0] = 14;
        database[55][0] = 14;
```

- Container Loading List 2 di Pelabuhan Makasar

```
        CLL[18][0] = -1;
        CLL[19][0] = -1;
        CLL[20][0] = -1;
        CLL[21][0] = -1;
        CLL[22][0] = -1;
        CLL[23][0] = -1;
        CLL[61][0] = -1;
        CLL[62][0] = -1;
        CLL[63][0] = -1;
        CLL[64][0] = -1;
        CLL[84][0] = -1;
        CLL[85][0] = -1;
        CLL[86][0] = -1;
        CLL[87][0] = -1;
        CLL[88][0] = -1;
        CLL[89][0] = -1;
        CLL[90][0] = -1;
        CLL[91][0] = -1;
        CLL[92][0] = -1;
        CLL[93][0] = -1;
        CLL[94][0] = -1;
        CLL[95][0] = -1;
        CLL[103][0] = -1;
        CLL[104][0] = -1;
        CLL[105][0] = -1;
        CLL[106][0] = -1;
```

```
        CLL[18][1] = 99;
        CLL[19][1] = 99;
        CLL[20][1] = 99;
        CLL[21][1] = 99;
        CLL[22][1] = 99;
        CLL[23][1] = 99;
        CLL[61][1] = 99;
        CLL[62][1] = 99;
        CLL[63][1] = 99;
        CLL[64][1] = 99;
        CLL[84][1] = 99;
        CLL[85][1] = 99;
        CLL[86][1] = 99;
        CLL[87][1] = 99;
        CLL[88][1] = 99;
        CLL[89][1] = 99;
        CLL[90][1] = 99;
        CLL[91][1] = 99;
        CLL[92][1] = 99;
        CLL[93][1] = 99;
        CLL[94][1] = 99;
        CLL[95][1] = 99;
        CLL[103][1] = 99;
        CLL[104][1] = 99;
        CLL[105][1] = 99;
        CLL[106][1] = 99;

        CLL[18][2] = 99;
        CLL[19][2] = 99;
        CLL[20][2] = 99;
        CLL[21][2] = 99;
        CLL[22][2] = 99;
        CLL[23][2] = 99;
        CLL[61][2] = 99;
        CLL[62][2] = 99;
        CLL[63][2] = 99;
        CLL[64][2] = 99;
        CLL[84][2] = 99;
        CLL[85][2] = 99;
```

```
        CLL[86][2] = 99;
        CLL[87][2] = 99;
        CLL[88][2] = 99;
        CLL[89][2] = 99;
        CLL[90][2] = 99;
        CLL[91][2] = 99;
        CLL[92][2] = 99;
        CLL[93][2] = 99;
        CLL[94][2] = 99;
        CLL[95][2] = 99;
        CLL[103][2] = 99;
        CLL[104][2] = 99;
        CLL[105][2] = 99;
        CLL[106][2] = 99;

        CLL[96][2] = 4;
        CLL[97][2] = 4;
        CLL[98][2] = 4;
        CLL[99][2] = 4;
        CLL[100][2] = 4;
        CLL[101][2] = 4;
        CLL[0][2] = 4;
        CLL[1][2] = 4;
        CLL[2][2] = 4;
        CLL[3][2] = 4;
        CLL[4][2] = 4;
        CLL[6][2] = 4;
        CLL[7][2] = 4;
        CLL[8][2] = 4;
        CLL[9][2] = 4;
        CLL[10][2] = 4;

        CLL[96][1] = 1;
        CLL[97][1] = 1;
        CLL[98][1] = 1;
        CLL[99][1] = 1;
        CLL[100][1] = 1;
        CLL[101][1] = 1;
```

```
        CLL[96][0] = 21;
        CLL[97][0] = 21;
        CLL[98][0] = 21;
        CLL[99][0] = 21;
        CLL[100][0] = 21;
        CLL[101][0] = 21;

        CLL[0][1] = 4;
        CLL[1][1] = 4;
        CLL[2][1] = 4;
        CLL[3][1] = 4;
        CLL[4][1] = 4;
        CLL[6][1] = 4;
        CLL[7][1] = 4;
        CLL[8][1] = 4;
        CLL[9][1] = 4;
        CLL[10][1] = 4;

        CLL[0][0] = 25;
        CLL[1][0] = 25;
        CLL[2][0] = 25;
        CLL[3][0] = 25;
        CLL[4][0] = 25;
        CLL[6][0] = 0;
        CLL[7][0] = 0;
        CLL[8][0] = 0;
        CLL[9][0] = 0;
        CLL[10][0] = 0;

        CLL[12][2] = 3;
        CLL[13][2] = 3;
        CLL[14][2] = 3;
        CLL[15][2] = 3;
        CLL[16][2] = 3;
        CLL[17][2] = 3;
        CLL[54][2] = 3;
        CLL[55][2] = 3;
        CLL[56][2] = 3;
        CLL[57][2] = 3;
```

```
        CLL[58][2] = 3;
        CLL[59][2] = 3;
        CLL[42][2] = 3;
        CLL[43][2] = 3;
        CLL[44][2] = 3;
        CLL[45][2] = 3;
        CLL[46][2] = 3;
        CLL[48][2] = 3;
        CLL[49][2] = 3;
        CLL[50][2] = 3;
        CLL[51][2] = 3;
        CLL[52][2] = 3;
        CLL[5][2] = 3;
        CLL[47][2] = 3;
        CLL[11][2] = 3;
        CLL[53][2] = 3;

        CLL[12][1] = 2;
        CLL[13][1] = 2;
        CLL[14][1] = 2;
        CLL[15][1] = 2;
        CLL[16][1] = 2;
        CLL[17][1] = 2;
        CLL[54][1] = 2;
        CLL[55][1] = 2;
        CLL[56][1] = 2;
        CLL[57][1] = 2;
        CLL[58][1] = 2;
        CLL[59][1] = 2;

        CLL[12][0] = 14;
        CLL[13][0] = 14;
        CLL[14][0] = 14;
        CLL[15][0] = 14;
        CLL[16][0] = 14;
        CLL[17][0] = 14;
        CLL[54][0] = 14;
        CLL[55][0] = 14;
        CLL[56][0] = 14;
```

```
CLL[57][0] = 14;
CLL[58][0] = 14;
CLL[59][0] = 14;

CLL[42][1] = 4;
CLL[43][1] = 4;
CLL[44][1] = 4;
CLL[45][1] = 4;
CLL[46][1] = 4;
CLL[48][1] = 4;
CLL[49][1] = 4;
CLL[50][1] = 4;
CLL[51][1] = 4;
CLL[52][1] = 4;

CLL[42][0] = 23;
CLL[43][0] = 23;
CLL[44][0] = 23;
CLL[45][0] = 23;
CLL[46][0] = 23;
CLL[48][0] = 0;
CLL[49][0] = 0;
CLL[50][0] = 0;
CLL[51][0] = 0;
CLL[52][0] = 0;

CLL[5][1] = 6;
CLL[47][1] = 6;
CLL[11][1] = 6;
CLL[53][1] = 6;

CLL[5][0] = 8;
CLL[47][0] = 8;
CLL[11][0] = 0;
CLL[53][0] = 0;
```

BIODATA PENULIS

Penulis bernama Nur Laili Asti Pramesti ini bertempat tinggal di Kabupaten Banyuwangi dan lahir pada 8 Juli 1998. Penulis menempuh pendidikan di SDN 4 Penganjuran (2004-2010), SMPN 1 Banyuwangi (2010-2013) dan SMAN 1 Glagah Banyuwangi (2013-2016). Penulis memiliki hobi dalam menikmati musik dan berkegiatan di luar ruangan. Selama menjalani masa perkuliahan di Departemen Matematika ITS, penulis mengambil bidang minat Ilmu Komputer (*Computer Science*). Penulis aktif sebagai pengurum BEM FMKSD ITS dan anggota aktif organisasi Beasiswa Mahaghora. Untuk informasi, kritik atau saran lebih lanjut dapat disampaikan melalui lailiastip@gmail.com