TUGAS AKHIR – IF184802

PENERAPAN PEMROGRAMAN DINAMIS UNTUK MENYELESAIKAN PERMASALAHAN KOMPUTASI GEOMETRI: STUDI KASUS SPOJ 7693 *ENVIRONMENTAL ENGINEERING*

Natasha Valentina Santoso
05111640000183

Dosen Pembimbing I :
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing II :
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

TUGAS AKHIR – IF184802

PENERAPAN PEMROGRAMAN DINAMIS UNTUK MENYELESAIKAN PERMASALAHAN KOMPUTASI GEOMETRI: STUDI KASUS SPOJ 7693 *ENVIRONMENTAL ENGINEERING*

Natasha Valentina Santoso
05111640000183

Dosen Pembimbing I :
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing II :
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESIS – IF184802

APPLICATION OF DYNAMIC PROGRAMMING TO SOLVE COMPUTATIONAL GEOMETRY PROBLEM: A CASE STUDY OF SPOJ 7693 ENVIRONMENTAL ENGINEERING

Natasha Valentina Santoso
05111640000183

Supervisor I :
Rully Soelaiman, S.Kom., M.Kom.

Supervisor II :
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

DEPARTEMENT OF INFORMATICS ENGINEERING
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

PENERAPAN PEMROGRAMAN DINAMIS UNTUK MENYELESAIKAN PERMASALAHAN KOMPUTASI GEOMETRI: STUDI KASUS SPOJ 7693 *ENVIRONMENTAL ENGINEERING*

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer pada
Bidang Studi Algoritma dan Pemrograman
Program Studi S-1 Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh:
Natasha Valentina Santoso
NRP: 051116 40000 183

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Soelaiman, S.Kom., M.Kom.
NIP. 197002131994021001 ..... (Pembimbing 1)

M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.
NIP. 197402092002121001 ..... (Pembimbing 2)

SURABAYA
JANUARI 2020

[Halaman ini sengaja dikosongkan]

PENERAPAN PEMROGRAMAN DINAMIS UNTUK MENYELESAIKAN PERMASALAHAN KOMPUTASI GEOMETRI: STUDI KASUS SPOJ 7693
ENVIRONMENTAL ENGINEERING

Nama Mahasiswa	: Natasha Valentina Santoso
NRP	: 051116 40000 183
Departemen	: Teknik Informatika FTEIC – ITS
Dosen Pembimbing 1	: Rully Soelaiman, S.Kom., M.Kom.
Dosen Pembimbing 2	: M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

Abstrak

Permasalahan ini diangkat dari soal pada situs SPOJ "Environmental Engineering" yang membahas mengenai pendistribusian air bersih. Permasalahan tersebut menceritakan cara menyalurkan air dari daerah kaya air ke daerah langka air. Masalah ini akan diselesaikan dengan membangun suatu jaringan pipa yang menghubungkan kedua tempat tersebut dengan syarat jaringan pipa tidak ada yang saling bersilangan. Hasil akhir yang diminta adalah total panjang minimal pipa yang harus dibangun.

Tugas Akhir ini merancang penyelesaian masalah dengan melakukan pemodelan jaringan pipa berdasarkan prinsip kesebangunan matematis. Jaringan pipa akan dihitung dengan cara pemrograman dinamis karena pemodelan dapat dipecah menjadi submasalah optimal dan submasalah tumpang tindih.

Solusi yang dikembangkan berjalan dengan kompleksitas $O(N^3)$, dimana N adalah jumlah titik lokasi. Solusi ini berhasil melakukan komputasi geometri dengan cepat.

Kata Kunci: komputasi geometri, kesebangunan geometri, pemrograman dinamis

[Halaman ini sengaja dikosongkan]

APPLICATION OF DYNAMIC PROGRAMMING TO SOLVE COMPUTATIONAL GEOMETRY PROBLEM: A CASE STUDY OF SPOJ 7693 *ENVIRONMENTAL ENGINEERING*

Student Name	: Natasha Valentina Santoso
NRP	: 051116 40000 183
Department	: Informatics Engineering FIEIT – ITS
Supervisor 1	: Rully Soelaiman, S.Kom., M.Kom.
Supervisor 2	: M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

Abstract

This problem is based on the problem on SPOJ "Environmental Engineering" which discusses the distribution of fresh water throughout the earth. This specific problem covers how to connect water scarce regions to water abundant regions. The idea in this project is to construct non-crossing gigantic pipelines that can transport water to these water scarce regions. The final objective is to calculate the total minimum length of all pipes that connect two adjacent locations.

In this Thesis, a solution will emerge by modeling the pipelines based on the principle of similarity in geometry. These pipelines modeling can be broken down into an optimal substructure and overlapping subproblems, which allows the calculation to be done with dynamic programming.

The complexity of the solution developed is $O(N^3)$, where N is the number of locations. The solution defined has successfully answered the geometry problem.

Keywords: computational geometry, similarity geometry, dynamic programming

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas berkat, bimbingan, dan rahmat-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

PENERAPAN PEMROGRAMAN DINAMIS UNTUK MENYELESAIKAN PERMASALAHAN KOMPUTASI GEOMETRI: STUDI KASUS SPOJ 7693 ENVIRONMENTAL ENGINEERING

Pengerjaan Tugas Akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana Komputer di Departemen Informatika Fakultas Teknologi Elektro dan Informatika Cerdas Institut Teknologi Sepuluh Nopember.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama mengerjakan Tugas Akhir serta selama menempuh masa perkuliahan. Pihak-pihak tersebut antara lain:

1. Kedua orang tua, adik, dan keluarga yang selalu memberikan dukungan selama mengerjakan Tugas Akhir dan selama menempuh kuliah.
2. Bapak Rully Soelaiman, S.Kom., M.Kom. selaku dosen pembimbing Tugas Akhir sekaligus sahabat yang telah memberikan ilmu, nasihat dan motivasi dari masa awal perkuliahan hingga masa pengerjaan Tugas Akhir di Departemen Informatika.
3. Bapak M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil. selaku dosen pembimbing yang memberikan arahan dan motivasi yang bermanfaat kepada penulis selama pengerjaan Tugas Akhir.

4. Seluruh tenaga pengajar, asisten dosen, dan karyawan Departemen Informatika ITS yang telah memberikan ilmu dan waktunya demi keberlangsungan kegiatan belajar mengajar di Departemen Informatika ITS.
5. Seluruh teman penulis di Departemen Informatika ITS yang telah memberikan dukungan dan semangat kepada penulis selama menyelesaikan Tugas Akhir.
6. Teman-teman dari organisasi mahasiswa KMK yang telah memberikan motivasi dan dukungan selama pembuatan Tugas Akhir serta menemani masa perkuliahan.
7. Seluruh pihak-pihak yang tidak dapat disebutkan yang telah membantu selama berkuliah di Departemen Informatika ITS.

Penulis mohon maaf bila masih terdapat kekurangan dalam Tugas Akhir ini. Penulis mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan terhadap Tugas Akhir ini. Semoga melalui Tugas Akhir ini, penulis dapat memberikan kontribusi dan maaf sebaik-baiknya.

Surabaya, Januari 2020

Natasha Valentina Santoso

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR PERSAMAAN

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini akan dijelaskan mengenai garis besar Tugas Akhir yang meliputi latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, dan sistematika penulisan Tugas Akhir.

1.1. Latar Belakang

Salah satu tantangan utama di masa depan adalah menghadapi kelangkaan air minum. Sumber daya air tawar yang memadai sangat dibutuhkan untuk kelangsungan hidup hewan, tumbuhan, dan manusia. Namun, distribusi air yang ada tidak merata di setiap tempat di belahan bumi menyebabkan beberapa wilayah kekurangan air minum yang layak sementara wilayah lainnya memiliki air minum yang berlebihan.

Topik Tugas Akhir ini mengacu pada permasalahan yang terdapat pada situs penilaian SPOJ dengan kode ENVIRON. Permasalahan ini digunakan untuk menggambarkan bagaimana pemerataan distribusi air di bumi secara sederhana. Daerah kaya air dan daerah langka air tersebar secara merata di sekitar lingkar bumi. Kemudian, pipa akan dibangun untuk menghubungkan daerah kaya air dengan daerah langka air sehingga air dapat dibagi secara merata. Permasalahan yang dihadapi adalah bagaimana cara membangun pipa dengan panjang total minimal dan tidak ada yang saling bersilangan.

Pendekatan yang dilakukan adalah dengan memodelkan permasalahan ke dalam bentuk geometri. Setelah pemodelan tersebut, didapatkan bahwa permasalahan dapat dipecah menjadi submasalah yang lebih kecil. Submasalah tersebut kemudian akan diselesaikan dengan pemrograman dinamis karena permasalahan

ini memiliki kriteria submasalah optimal dan submasalah tumpang tindih.

Hasil Tugas Akhir ini diharapkan dapat memberikan gambaran mengenai penerapan pemrograman dinamis untuk menyelesaikan permasalahan komputasi geometri secara optimal dan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

1.2. Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana bentuk pemodelan matematis dalam menyelesaikan permasalahan komputasi geometri pada pendistribusian air?
2. Bagaimana desain algoritma dalam menyelesaikan permasalahan komputasi geometri pada pendistribusian air?
3. Bagaimana cara implementasi algoritma dan struktur data yang digunakan dalam menyelesaikan permasalahan komputasi geometri pada pendistribusian air?
4. Bagaimana hasil kinerja dari implementasi algoritma yang telah dilakukan dalam menyelesaikan permasalahan komputasi geometri pada pendistribusian air?

1.3. Batasan Masalah

Permasalahan yang dibahas pada Tugas Akhir ini memiliki dua topik batasan, yaitu batasan Tugas Akhir dan batasan dari situs penilaian SPOJ. Batasan dari Tugas Akhir adalah sebagai berikut:

1. Permasalahan yang dibahas terbatas pada komputasi geometri dari kasus pendistribusian air.

2. Metode penyelesaian yang digunakan adalah pemrograman dinamis sebagai solusi masalah.
3. Persebaran daerah distribusi air dari daerah kaya air dan daerah langka air harus berkorespondensi satu-satu.

Batasan dari situs penilaian SPOJ adalah sebagai berikut:

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Batas maksimum dari jumlah lokasi adalah sebanyak 500.
3. Batas maksimum waktu perangkat lunak berjalan adalah 5.726 detik.
4. Batas memori yang digunakan untuk memproses kasus uji adalah 1536 MB.
5. Batas ukuran kode sumber yang dikirim adalah 50kB.
6. Kasus uji yang digunakan adalah kasus uji yang terdapat pada permasalahan SPOJ *Environmental Engineering*.

1.4. Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Menentukan pemodelan matematis yang sesuai dengan permasalahan komputasi geometri pada pendistribusian air.
2. Melakukan analisis dan desain algoritma untuk menyelesaikan permasalahan komputasi geometri pada pendistribusian air.
3. Melakukan implementasi algoritma untuk menyelesaikan permasalahan komputasi geometri pada pendistribusian air.
4. Mengevaluasi hasil dari kinerja algoritma untuk menyelesaikan permasalahan komputasi geometri pada pendistribusian air.

1.5. Manfaat

Tugas Akhir ini diharapkan dapat membantu memahami penggunaan pemrograman dinamis dan penerapannya untuk menyelesaikan permasalahan komputasi geometri pada permasalahan pendistribusian air.

1.6. Metodologi

Metodologi yang digunakan yang digunakan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir
   Pada tahap ini dilakukan penyusunan proposal Tugas Akhir yang berisi gagasan penyelesaian dalam permasalahan komputasi geometri pada studi kasus SPOJ 7693 *Environmental Engineering*.
2. Studi literatur
   Pada tahap ini dilakukan pencarian informasi dan studi literatur yang relevan sebagai acuan referensi dalam menyelesaikan masalah. Informasi didapatkan dari materi-materi yang berhubungan dengan algoritma dalam menyelesaikan masalah. Materi tersebut didapatkan dari buku, jurnal, dan internet yang memiliki kaitan dengan metode yang akan digunakan.
3. Desain
   Pada tahap ini dilakukan desain rancangan algoritma yang digunakan dalam solusi untuk memecahkan permasalahan SPOJ 7693 *Environmental Engineering*.
4. Implementasi perangkat lunak
   Pada tahap ini dilakukan implementasi dari rancangan desain algoritma yang telah dibangun pada tahap desain ke dalam bentuk program.

5. Uji coba dan evaluasi
   Pada tahap ini dilakukan uji coba kebenaran dan kinerja solusi yang telah diimplementasikan. Uji coba dilakukan dengan mengirimkan hasil kode implementasi ke situs penilaian daring SPOJ pada permasalahan terkait dan melihat hasil umpan balik. Pengujian dilakukan dengan membandingkan hasil keluaran program dengan hasil umpan balik dari situs penilaian.
6. Penyusunan buku Tugas Akhir
   Pada tahap ini dilakukan penyusunan buku Tugas Akhir yang menjelaskan dasar teori dan metode yang digunakan serta hasil implementasi aplikasi perangkat lunak yang telah dibuat.

1.7. Sistematika Penulisan

Sistematika penulisan buku Tugas Akhir secara garis besar adalah sebagai berikut:

1. Bab I: Pendahuluan

Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, dan sistematika penulisan Tugas Akhir.

2. Bab II: Dasar Teori

Bab ini berisi dasar teori yang digunakan dalam menyelesaikan masalah serta algoritma penyelesaian yang digunakan dalam Tugas Akhir.

3. Bab III: Desain

Bab ini berisi desain algoritma dan struktur data yang digunakan dalam penyelesaian permasalahan.

4. Bab IV: Implementasi

Bab ini berisi implementasi dari dari rancangan algoritma yang telah dibuat pada tahap desain.

5. Bab V: Uji Coba dan Evaluasi

Bab ini berisi uji coba dan evaluasi dari hasil implementasi yang telah dilakukan pada tahap implementasi.

6. Bab VI: Kesimpulan

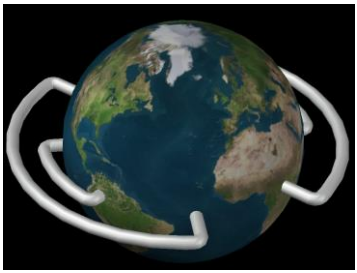
Bab ini berisi kesimpulan dari hasil uji coba yang telah dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

BAB II
DASAR TEORI

Pada bab ini akan dijelaskan mengenai dasar teori yang menjadi dasar pengerjaan Tugas Akhir.

2.1. Deskripsi Permasalahan SPOJ *ENVIRON*

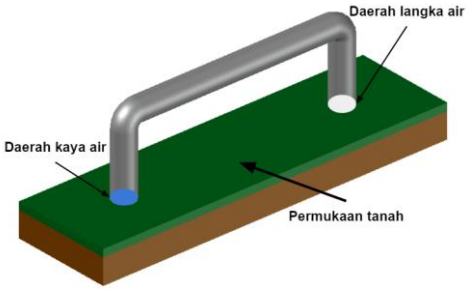
Permasalahan SPOJ *Environmental Engineering* dengan kode ENVIRON membahas mengenai distribusi air yang tidak merata di bumi sehingga menyebabkan beberapa wilayah kekurangan air minum yang layak, sementara wilayah lain air minumnya berlebihan. Diceritakan bahwa *International Committee for Precious Consumables* (ICPC) merencanakan proyek untuk mendistribusikan air secara merata di bumi. Diasumsikan bahwa bumi berbentuk bola sempurna. Daerah kaya sumber daya air dan daerah langka air terletak secara merata di sepanjang lingkaran dengan panjang maksimal di sekitar bumi, di mana pusat bumi sebagai pusat lingkaran. Hasil akhir proyek ini akan berbentuk seperti pada Gambar 2.1.



Gambar 2.1 Ilustrasi jaringan pipa di sekitar bumi

Ide yang dimunculkan adalah membangun jaringan pipa raksasa yang dapat mengangkut air di sepanjang lingkaran tersebut. Setiap pipa memiliki dua ujung yang dimulai dari lokasi kaya air

dan berakhir di lokasi langka air. Namun, mengebor ke dalam bumi pada jarak yang sangat jauh adalah pilihan yang tidak masuk akal, sehingga semua jalur pipa tetap berada di atas permukaan bumi. Selanjutnya, saluran khusus untuk pipa-pipa tersebut berada di ketinggian yang berbeda. Saluran pipa dibangun secara vertikal ke atas sampai salah satu ketinggian yang diijinkan tercapai, lalu berbelok membentuk sudut 90^o, dan mengikuti saluran pada ketinggian konstan hingga mencapai lokasi langka air, kemudian berbelok lagi membentuk sudut 90^o turun secara vertikal ke tanah (Gambar 2.2). Semua proyeksi jaringan pipa ke tanah pada bidang 2 dimensi harus mengikuti lingkaran yang sama sehingga tidak ada saluran pipa yang saling silang. Batas ketinggian pipa adalah sebuah bilangan bulat asli dikali jarak dari lokasi kaya air ke lokasi langka air. Jarak dari kedua lokasi tersebut, yang diukur dari permukaan bumi berbentuk lingkaran, bernilai 1 satuan.



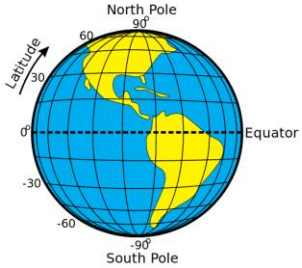
Gambar 2.2 Ilustrasi pipa yang dibangun di atas permukaan tanah

Masukan yang diberikan adalah *N* jumlah dari lokasi yang ada. Kemudian diikuti sebanyak *N* bilangan yang terdiri dari ‘0’ atau ‘1’. Angka ‘0’ merepresentasikan lokasi yang langka air, sedangkan angka ‘1’ merepresentasikan lokasi yang kaya air. Jumlah angka ‘0’ akan sebanyak angka ‘1’ karena setiap lokasi akan terhubung tepat satu. Permasalahan ini dibatasi dengan nilai

maksimal $N = 500$ dan batas waktu maksimal adalah 5.7 detik. Hasil akhir yang didapatkan berupa panjang total minimal pipa yang akan dibangun dengan ketelitian desimal sebesar 10^{-2} dan hubungan pipa-pipa tersebut tidak ada yang saling bersilangan.

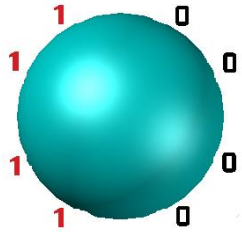
2.2. Penjelasan Permasalahan SPOJ *ENVIRON*

Permasalahan di atas menceritakan cara membangun jaringan pipa air di atas permukaan bumi untuk menghubungkan daerah kaya air dan daerah langka air. Dari permasalahan tersebut, akan dibuat asumsi untuk lebih memperjelas permasalahannya. Daerah kaya air dan daerah langka air akan diasumsikan terletak di sepanjang garis khatulistiwa dari sudut pandang garis lintang. Hal ini sesuai dengan pernyataan bahwa daerah kaya air dan daerah langka air terletak secara merata di sepanjang lingkaran dengan panjang maksimal di sekitar bumi, di mana pusat bumi sebagai pusat lingkaran. Menurut Mitchell (1876:15), garis khatulistiwa (*equator*) adalah garis imajiner yang memanjang dari timur ke barat di sekitar bumi dan memiliki jarak yang sama jauh dari kedua kutubnya, sedangkan garis lintang (*latitude*) adalah garis khayal yang menentukan lokasi di bumi terhadap garis khatulistiwa. Garis lintang khatulistiwa adalah 0^o dan membagi bumi menjadi dua bagian, yaitu belahan bumi utara dan belahan bumi selatan (Gambar 2.3).



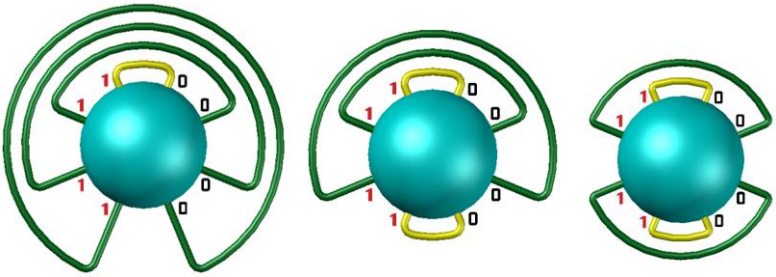
Gambar 2.3 Penggambaran garis lintang dan garis khatulistiwa

Contoh penggambaran permasalahan dapat diambil dari masukan dengan *N=8* yang menunjukkan adanya 8 lokasi dan penggambaran daerah dengan representasi ‘1 1 1 1 0 0 0 0’ seperti pada Gambar 2.4.



Gambar 2.4 Ilustrasi konfigurasi yang terdiri dari 8 lokasi

Sesuai dengan contoh permasalahan di atas, maka hasil penggambaran hubungan pipa menjadi 3 kemungkinan yang dinyatakan pada Gambar 2.5. Gambaran bola adalah representasi dari bumi yang dilihat dari sudut kutub utara, sedangkan pipa-pipa tersebut terletak di sepanjang garis khatulistiwa.



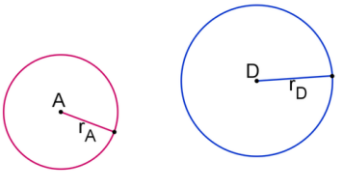
Gambar 2.5 Kemungkinan jalur pipa yang dapat dibangun

Permasalahan ini adalah sebuah permasalahan optimasi yang bertujuan untuk mencari nilai terbaik dari semua kemungkinan yang ada. Pendekatan pemrograman dinamis dapat menyelesaikan permasalahan ini karena permasalahan ini memiliki kriteria submasalah optimal dan submasalah tumpang tindih.

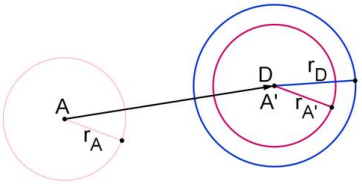
2.3. Prinsip Kesebangunan dalam Geometri

Dua objek geometri dikatakan sebangun, jika keduanya memiliki bentuk yang sama atau salah satu memiliki bentuk sebagai pencerminan dari bentuk lainnya. Bentuk objek geometri yang sebangun didapatkan dari mentransformasikan objek asal, yaitu dengan melakukan translasi, refleksi, rotasi, atau dilatasi diperbesar maupun diperkecil. Kedua objek geometri dapat disebut sebangun, jika memenuhi dua syarat, yaitu: sudut-sudut yang bersesuaian sama besar dan sisi-sisi yang bersesuaian memiliki perbandingan yang sama.

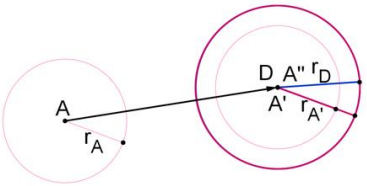
Permasalahan yang diangkat adalah mengenai kesebangunan dalam bidang geometri berbentuk lingkaran. Lingkaran adalah himpunan semua titik di bidang datar yang berjarak sama dari suatu titik tetap di bidang tersebut. Titik tetap lingkaran disebut pusat lingkaran, sedangkan jarak dari suatu titik pada lingkaran ke titik pusat disebut jari-jari (*radius*). Ukuran dari sebuah lingkaran hanya bergantung pada panjang jari-jarinya. Bagian ini akan membuktikan kesebangunan pada objek lingkaran. Misalkan terdapat lingkaran *A*, dengan pusat pada titik *A* dan jari-jari r_A dan lingkaran *D*, dengan pusat pada titik *D* dan jari-jari r_D. Kedua lingkaran ini terpisah pada suatu bidang datar (Gambar 2.6).



Gambar 2.6 Lingkaran mula-mula

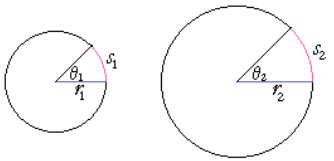


Gambar 2.7 Langkah translasi lingkaran *A* ke Lingkaran *D*



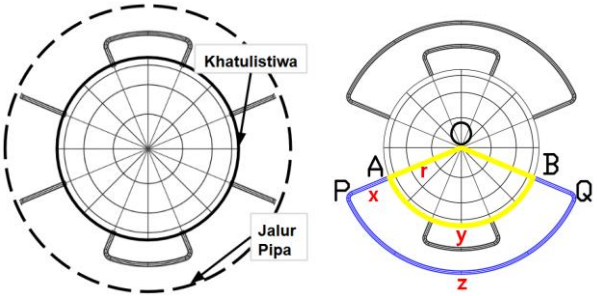
Gambar 2.8 Langkah dilatasi lingkaran *A* ke lingkaran *D*

Terdapat beberapa langkah untuk membuktikan bahwa kedua lingkaran adalah sebangun. Pertama, gambarkan vektor dari pusat *A* ke pusat *D* dan translasikan lingkaran *A* sepanjang vektor untuk membentuk lingkaran *A'* dan $r_A \cong r'_A$ (Gambar 2.7). Lalu, lingkaran *A'* didilatasi sehingga seukuran dengan lingkaran *D* (Gambar 2.8). Karena ukuran lingkaran sepenuhnya ditentukan oleh jari-jari, jari-jari dapat digunakan untuk mencari faktor skala yang benar. Lingkaran *A'* akan didilatasi sebesar faktor skala $\frac{rD}{rA'}$ dan menghasilkan lingkaran *A''*. Lingkaran *A''* adalah lingkaran yang sama ukurannya dengan lingkaran *D* karena $r_{A''} = \frac{rD}{rA'} \cdot r_{A'} = r_D$ dan pusat *A''* terletak pada titik yang sama dengan pusat *D*. Karena sebuah lingkaran didefinisikan oleh pusat dan jari-jari, maka dua lingkaran dengan pusat dan jari-jari yang sama adalah lingkaran yang sama. Hal ini berarti lingkaran *A* sebangun dengan lingkaran *D* karena terjadi transformasi berupa translasi dan dilatasi yang memetakan lingkaran *A* ke lingkaran *D*.



Gambar 2.9 Perbandingan kesebangunan pada busur lingkaran

Legendre (1867:154) mengemukakan, "Pada lingkaran apapun rasio panjang busur terhadap jari-jari menentukan sudut pusat tertentu yang ditarik dari busur tersebut, sebaliknya, sudut pusat yang sama dapat digunakan untuk menentukan rasio panjang busur terhadap jari-jari". Gambar 2.9 menunjukkan rasio busur S_1 dengan jari-jari r_1 sebanding dengan rasio busur S_2 dengan jari-jari r_2 ($\frac{S_1}{r_1} = \frac{S_2}{r_2}$), jika dan hanya jika $\theta_1 = \theta_2$.



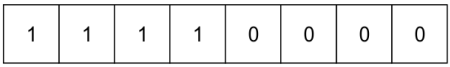
Gambar 2.10 Kesebangunan juring bumi dengan juring jalur pipa

Implementasi dari teori kesebangunan lingkaran dapat dilihat pada penggambaran lingkaran garis khatulistiwa dan lingkaran garis putus-putus jalur pipa (Gambar 2.10). Juring lingkaran adalah bidang yang dibatasi oleh busur lingkaran dan dua jari-jari lingkaran. Karena lingkaran garis khatulistiwa dan lingkar utuh jalur pipa adalah sebangun, maka juring bumi ($\sphericalangle AOB$) dan juring jalur pipa ($\sphericalangle POQ$) juga sebangun. Kesebangunan ini harus memenuhi syarat-syarat bahwa sudut-sudut yang bersesuaian sama

besar dan sisi-sisi yang bersesuaian memiliki perbandingan yang sama. Sudut-sudut yang bersesuaian ditunjukkan dengan $\angle AOB = \angle POQ$, dan jari-jari *OP* adalah sebuah faktor skala dari jari-jari *OA*.

2.4. Analisis Submasalah Optimal pada Permasalahan SPOJ *ENVIRON*

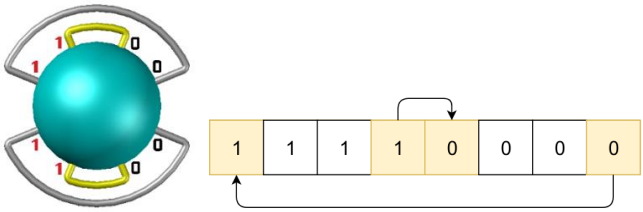
Subbab ini menjelaskan submasalah-submasalah yang jawabannya dapat membangun hasil akhir dari panjang total pipa. Konfigurasi lokasi berdasarkan deskripsi permasalahan (Gambar 2.4) dapat digambarkan sebagai garis lurus seperti terlihat pada Gambar 2.11.



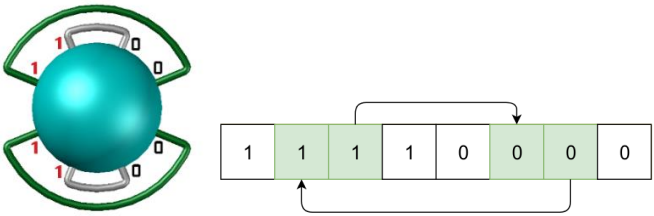
Gambar 2.11 Konfigurasi lokasi berbentuk garis lurus

Pemrograman dinamis menyelesaikan masalah dengan memecahnya menjadi sub-masalah. Pemrograman dinamis dapat diterapkan ketika terdapat sub-masalah yang sama terjadi berulang-ulang. Algoritma pemrograman dinamis memecahkan masing-masing sub-masalah sekali saja kemudian menyimpan jawabannya dalam sebuah tabel, dengan demikian algoritma ini dapat mengurangi perhitungan ulang setiap kali menyelesaikan sub-masalah yang sama. Pada dasarnya, terdapat 2 tipe submasalah yang dapat didefinisikan dari permasalahan yang diangkat. Submasalah pertama adalah menghubungkan 2 titik yang saling bersebelahan dan submasalah kedua adalah menghubungkan 2 titik dimana ada titik-titik lain di bawahnya. Gambar 2.12 menampilkan representasi dari submasalah pertama, dimana kotak berwarna kuning pada representasi garis lurus dan pipa berwarna kuning pada representasi bumi menghubungkan lokasi yang saling

bersebelahan. Kemudian, Gambar 2.13 menunjukkan submasalah kedua, dimana kotak berwarna hijau pada representasi garis lurus dan pipa berwarna hijau pada representasi bumi menghubungkan dua lokasi yang di bawahnya terdapat jalur pipa lain. Terdapat pengecualian dalam penggambaran representasi garis lurus dimana titik di akhir yang dihubungkan ke titik awal akan memiliki jarak 1 karena representasi asli adalah dalam bentuk melingkar. Hubungan ini akan diwakilkan menjadi garis panah yang terletak di bawah kotak.

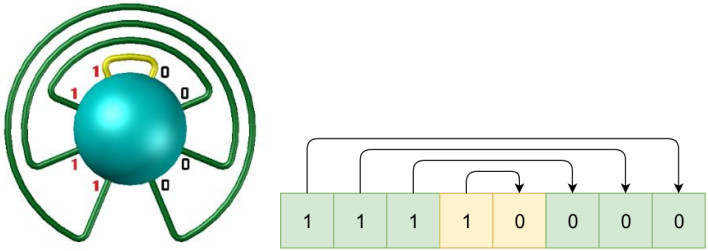


Gambar 2.12 Penggambaran submasalah pertama yang saling bersebelahan

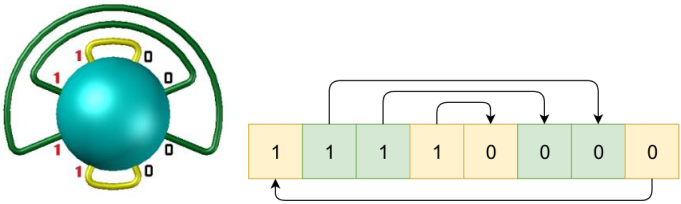


Gambar 2.13 Penggambaran submasalah kedua dimana terdapat titik lain di bawahnya

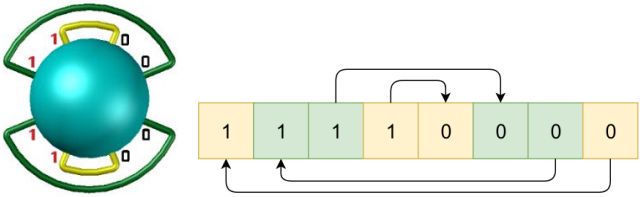
Hasil akhir dapat dibentuk dari penggabungan kedua submasalah tersebut yang disesuaikan dengan penggambaran kemungkinan dari Gambar 2.5.



Gambar 2.14 Ilustrasi kemungkinan pertama



Gambar 2.15 Ilustrasi kemungkinan kedua



Gambar 2.16 Ilustrasi kemungkinan ketiga

Ilustrasi di atas (Gambar 2.14 – Gambar 2.16) menunjukkan terbentuknya 3 kemungkinan dalam membangun jalur pipa dari penggabungan submasalah. Ilustrasi kemungkinan pertama (Gambar 2.14), menunjukkan jalur pipa dibangun dari 1 submasalah pertama dan 3 submasalah kedua. Ilustrasi kemungkinan kedua seperti pada Gambar 2.15 dan ilustrasi kemungkinan ketiga seperti pada Gambar 2.16 merupakan penggabungan 2 submasalah pertama dan 2 submasalah kedua. Yang menjadi pembeda dari kedua

gambar ini adalah garis yang menghubungkan kotak ke-2 dan kotak ke-7. Pada kemungkinan kedua, garis panah berawal dari kotak ke-2 dan berakhir di kotak ke-7 yang menunjukkan terdapat jalur pipa lain di bawahnya dari kotak ke-3 hingga kotak ke-6. Di sisi lain, kemungkinan ketiga menggambarkan garis panah berawal dari kotak ke-7 dan berakhir di kotak ke-2 yang menunjukkan jalur pipa di bawahnya berasal dari kotak ke-8 ke kotak ke-1.

2.5. Definisi dan Notasi

Berikut adalah notasi yang digunakan pada Tugas Akhir:

1. $Excess_i$ adalah notasi untuk nilai hasil *prefix sum*, yaitu selisih jumlah lokasi kaya air dan jumlah lokasi langka air pada posisi ke-i.
2. $DP_{(x,y)}$ adalah notasi untuk nilai hasil perhitungan submasalah, dimana simbol x menyatakan jarak antar-lokasi dan simbol y menyatakan indeks iterasi.
3. $Iterasi\ i.j.k$ adalah notasi yang menyatakan iterasi perulangan, dimana simbol i menyatakan jarak yang tercakup, simbol j menyatakan pergeseran ke-j, dan simbol k menyatakan perulangan dari tahap transisi.

2.6. Penyelesaian Permasalahan SPOJ *ENVIRON*

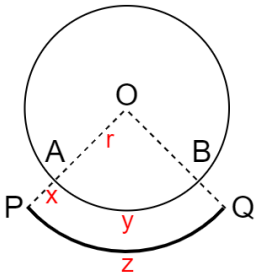
Subbab ini akan membahas mengenai strategi penyelesaian permasalahan SPOJ dengan kode ENVIRON. Strategi penyelesaian masalah akan dibagi menjadi 3 tahap, yaitu:

1. Merumuskan panjang pipa melalui prinsip kesebangunan
2. Menghubungkan lokasi kaya air dan lokasi langka air melalui *prefix sum*
3. Menggabungkan submasalah melalui pemrograman dinamis

Penyelesaian utama dari Tugas Akhir ini adalah dengan cara memodelkan ke dalam bentuk geometri dan menyelesaikan permasalahan komputasi dengan menerapkan pemrograman dinamis untuk mencapai hasil yang paling minimal.

2.6.1. Perumusan Panjang Pipa berdasarkan Prinsip Kesebangunan

Bagian pertama membahas penerapan kesebangunan dari lingkar khatulistiwa bumi dengan jaringan pipa. Bagian ini direpresentasikan pada Gambar 2.17, di mana lingkar terdalam menggambarkan garis khatulistiwa bumi dan juring lingkar luar (busur *PQ*) adalah jalur pipa yang akan dibangun.



Gambar 2.17 Ilustrasi dari lingkar khatulistiwa dan jalur pipa

Dari gambar di atas dapat dinyatakan bahwa juring lingkaran *AOB* dan juring lingkaran *POQ* adalah sebangun. Titik *A* adalah lokasi yang kaya air dan titik *B* adalah lokasi langka air. Garis *OA* (simbol: *r*) menyatakan jari-jari bumi yang juga merupakan jarak lokasi air dengan pusat bumi. Garis *AP* (simbol: *x*) dan garis *BQ* menyatakan ketinggian pipa dari permukaan tanah. Busur *AB* (simbol: *y*) menyatakan jarak lokasi kaya air dan lokasi langka air pada garis khatulistiwa. Busur *PQ* (simbol: *z*) adalah panjang saluran pipa air. Sesuai dengan pernyataan pada soal bahwa jarak dari kedua lokasi bernilai 1 satuan, sehingga busur *AB*

akan bernilai 1 satuan. Untuk menghitung jari-jari *OA* dapat diturunkan dari rumus keliling pada Persamaan 2.1.

$$y = \frac{\theta}{360} \cdot 2\pi r, (\theta = \frac{360}{N})$$
$$y = \frac{1}{N} \cdot 2\pi r$$
$$r = \frac{yN}{2\pi}, y = 1$$
$$r = \frac{N}{2\pi} \quad (2.1)$$

Dari rumus di atas, simbol θ berarti sudut pusat juring, simbol *k* berarti panjang busur *AB*, dan *N* adalah jumlah seluruh titik lokasi. Sesuai dengan rumus kesebangunan juring lingkaran ($\frac{S_1}{r_1} = \frac{S_2}{r_2}$), didapatkan rumus untuk mencari panjang saluran air pada Persamaan 2.2.

$$\diamond AOB \cong \diamond POQ$$
$$\frac{r}{y} = \frac{r + x}{z}$$
$$zr = y\,(r + x)$$
$$zr = yr + xy$$
$$z = y + \frac{xy}{r}$$
$$z = y + xy\left(\frac{2\pi}{N}\right) \quad (2.2)$$

Total pipa (simbol: *T*) untuk menghubungkan titik *A* ke titik *B* adalah penjumlahan dari 2 panjang ketinggian pipa dan panjang saluran air yang dinyatakan pada Persamaan 2.3.

$$Total\ pipa\ A \rightarrow B = PQ + AP + BQ$$
$$Total\ pipa = z + x + x$$
$$T = \left(y + xy\left(\frac{2\pi}{N}\right)\right) + 2x$$
$$T = 2xy\left(\frac{\pi}{N}\right) + 2x + y$$
$$T = \underbrace{2xy}_{a}\ \underbrace{\left(\frac{\pi}{N}\right)}_{k} + \underbrace{2x + y}_{b}$$
$$T = a * k + b \qquad (2.3)$$

Dalam menyelesaikan permasalahan, koefisien variabel *a* dan *b* akan digunakan dalam perhitungan komputasi geometri.

2.6.2. Penerapan *Prefix Sum* pada Masukan Lokasi

Bagian kedua membahas cara menentukan hubungan lokasi kaya air dan lokasi langka air menggunakan *prefix sum*. *Prefix sum*, dengan penamaan variabel $Excess_i$, digunakan untuk menghitung selisih jumlah lokasi kaya air dengan jumlah lokasi langka air pada posisi ke-*i*.

$$Excess_i = Excess_{i-1} + a_i \qquad (2.4)$$

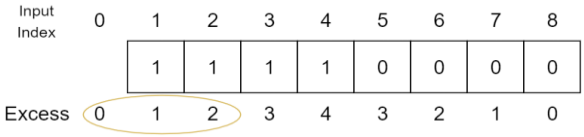
Perhitungan ini menggunakan Persamaan 2.4, dimana a_i dikurangkan dengan 1 bila merepresentasikan lokasi langka air (masukan angka '0') dan ditambahkan 1 bila merepresentasikan lokasi kaya air (masukan angka '1'). Hubungan lokasi kaya air dan lokasi langka air didefinisikan sebagai seimbang, jika dari rentang *x* hingga *y* inklusif (simbol: *S[x,y]*), jumlah lokasi kaya air dan lokasi langka air sama banyak. Pada $S[x, y]$, suatu lokasi dikatakan seimbang jika memenuhi persyaratan Persamaan 2.5.

$$Excess_y = Excess_{x-1} \qquad (2.5)$$

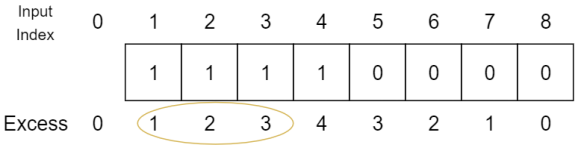
2.6.3. Penerapan Pemrograman Dinamis untuk Menggabungkan Submasalah

Pada bagian ketiga dibahas mengenai penggabungan submasalah dengan cara pemrograman dinamis. $DP_{(x,y)}$ menyimpan nilai dari koefisien variabel *a* dan *b* dari Persamaan 2.3. dan ketinggian pipa. Pada awal inisiasi $DP_{(0,y)}$, seluruh variabel *a*, *b*, dan ketinggian bernilai 0 karena masih berada pada satu titik dan belum ada ketinggian yang dicapai. Penghitungan dimulai dari 2 titik yang saling bersebelahan (menghubungkan lokasi kaya air dan lokasi langka air) dan naik setiap kelipatan 2 hingga mencakup seluruh titik. Setiap iterasi menyimpan biaya minimal yang dibutuhkan berupa koefisien variabel *a*, *b*, dan ketinggian untuk mencakup jarak hingga titik tersebut.

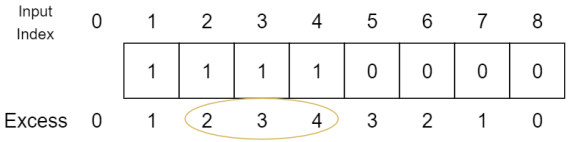
Iterasi dimulai dari tingkat paling bawah yaitu menghubungkan angka '0' dan angka '1' pada ketinggian pipa bilangan asli 1 satuan (iterasi *i=2*). Iterasi dilakukan dengan mencoba ke seluruh titik untuk mencari lokasi yang seimbang menggunakan *prefix sum*. Iterasi untuk mencapai submasalah pertama ditunjukkan pada Gambar 2.18 - Gambar 2.21.



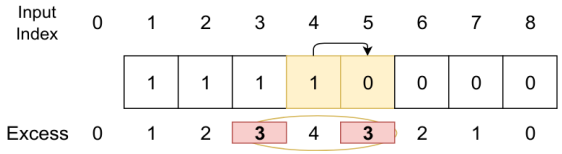
Gambar 2.18 Iterasi 2.0 pada submasalah pertama



Gambar 2.19 Iterasi 2.1 pada submasalah pertama



Gambar 2.20 Iterasi 2.2 pada submasalah pertama



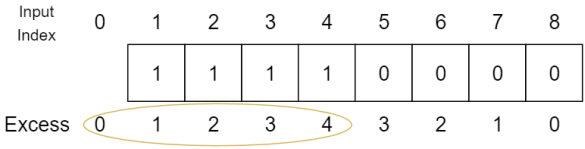
Gambar 2.21 Iterasi 2.3 pada submasalah pertama

Pada gambar tersebut, bentukan persegi adalah representasi masukan dan persegi berwarna kuning menunjukkan submasalah pertama. Variabel $Excess_i$ pada bagian bawah persegi adalah hasil perhitungan *prefix sum*. Pada Gambar 2.21 telah ditemukan lokasi seimbang, yaitu pada *S[4,5]* ditemukan $Excess_5 = Excess_3$. Selanjutnya, dilakukan pengecekan apakah pada masukan ke-4 dan masukan ke-5 berbeda jenis. Pengecekan ini menentukan apakah pada titik *x* dan *y* terdapat lokasi kaya air dan lokasi langka air. Jika berbeda, pipa dapat dibuat dari ujung ke ujung dengan jarak dan ketinggian pipa yang ada. Notasi yang dihasilkan untuk menghubungkan titik pada iterasi ke-*y* terdapat pada Persamaan 2.6.

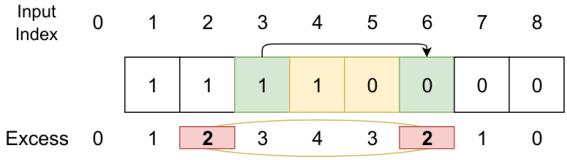
$$DP_{(x,y)} = y \rightarrow (x + y - 1) \tag{2.6}$$

Dari contoh soal pada iterasi *i=2*, masukan ke-4 dapat dihubungkan ke masukan ke-5. Hubungan ini dimasukkan pada Persamaan 2.3 dengan variabel *x* berasal dari jarak yang tercakup dan variabel *y* menyatakan ketinggian pipa. Hasil perhitungan disederhanakan menjadi variabel *a* dan *b* kemudian disimpan pada $DP_{(2,4)}$.

Iterasi dilanjutkan dengan *i=4* (Gambar 2.22) dan ketinggian pipa ditambah 1 satuan agar pipa tidak bersilangan dengan pipa di bawahnya.



Gambar 2.22 Iterasi 4.0 pada submasalah kedua



Gambar 2.23 Iterasi 4.2 pada submasalah kedua

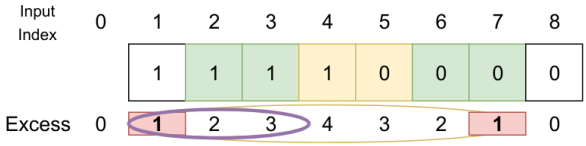
Gambar 2.23 menunjukkan bahwa masukan ke-3 dapat dihubungkan dengan masukan ke-6 dan ditambah dengan hubungan pipa dari masukan ke-4 dan masukan ke-5. Penghitungan submasalah kedua dilakukan ketika pipa dapat dihubungkan dan ditambah dengan hasil perhitungan pipa di bawahnya. Notasi ini dinyatakan dengan Persamaan 2.7.

$$DP_{(x,y)} = DP_{(x,y)} + DP_{(x-2,y+1)} \quad (2.7)$$

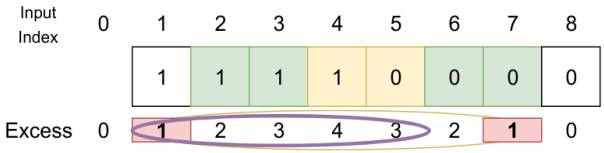
Hasil dari penambahan submasalah kedua ini akan menjadi $DP_{(4,3)} = DP_{(4,3)} + DP_{(2,4)}$. Bagian ini menunjukkan telah terjadi rekurensi yang membutuhkan nilai dari perhitungan masalah sebelumnya.

Setiap iterasi akan melalui tahap transisi yaitu untuk mencari apakah ada nilai yang lebih kecil dari penggabungan submasalah daripada melakukan penambahan pipa di atasnya.

Untuk setiap kemungkinan *k* dari 2 hingga *x-2* dan *k* bilangan genap, jika $DP_{(k,y)}$ tidak seimbang, maka akan pindah ke perulangan selanjutnya. Contoh dari kasus ini ada pada Gambar 2.24 dan Gambar 2.25 yang terjadi pada iterasi *i=6*.



Gambar 2.24 Iterasi 6.1.2 pada tahap transisi tidak seimbang



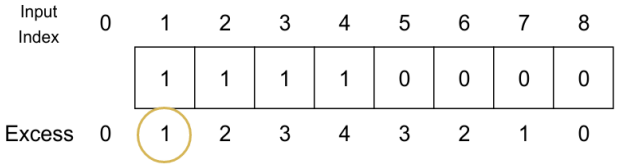
Gambar 2.25 Iterasi 6.1.4 pada tahap transisi tidak seimbang

Gambar di atas menunjukkan adanya hubungan dari masukan ke-2 ke masukan ke-7 melalui *prefix sum* pada $Excess_i$ berwarna merah. Bagian iterasi tahap transisi dinyatakan oleh bentuk elips. Iterasi 6.1.2 menyatakan $Excess_1 \neq Excess_3$ dan iterasi 6.1.4 menyatakan $Excess_1 \neq Excess_5$ sehingga dapat dikatakan tahap transisi ini tidak seimbang.

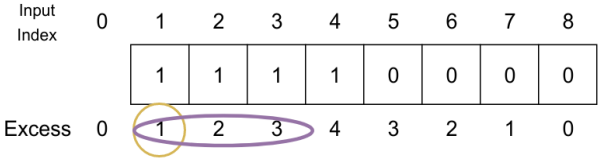
Tahap transisi dikatakan seimbang bila $DP_{(x,y)}$ dan $DP_{(k,y)}$ memenuhi syarat seimbang pada *prefix sum*. Jika $DP_{(k,y)}$ seimbang, $DP_{(x-k,y+k)}$ juga seimbang. Jika salah satu submasalah seimbang, maka submasalah lainnya yang tercakup dalam jarak *x* juga memenuhi syarat seimbang tersebut.

$$DP_{(x,y)} = DP_{(k,y)} + DP_{(x-k,y+k)} \tag{2.8}$$

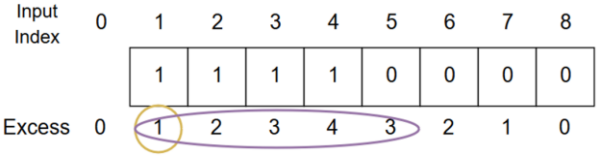
Hasil penggabungan submasalah didapat dari Persamaan 2.8 yang menyatakan penjumlahan submasalah tahap transisi dengan submasalah selain tahap transisi pada jangkauan jarak *x*. Hasil ini akan disimpan jika nilainya lebih minimal dari sebelumnya.



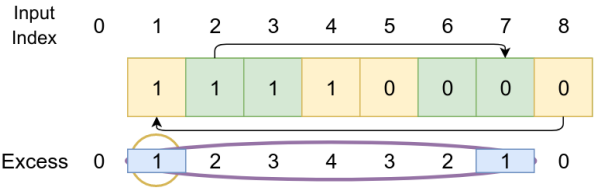
Gambar 2.26 Iterasi 8.1 pada submasalah



Gambar 2.27 Iterasi 8.1.2 pada tahap transisi tidak seimbang



Gambar 2.28 Iterasi 8.1.4 pada tahap transisi tidak seimbang



Gambar 2.29 Iterasi 8.1.6 pada tahap transisi seimbang

Pada Gambar 2.26, iterasi $i=8$ diwakilkan oleh lingkaran berwarna kuning pada $Excess_1$ karena iterasi terakhir ini mencakup seluruh titik lokasi. Gambar 2.27 dan Gambar 2.28

menyatakan tahap transisi tidak seimbang, kemudian pada Gambar 2.29 telah memenuhi syarat seimbang yang diwakili oleh persegi biru pada $Excess_1 = Excess_7$. Tahap transisi dinyatakan seimbang jika $DP_{(8,2)}$ dan $DP_{(6,2)}$ seimbang. $DP_{(6,2)}$ dinyatakan seimbang dengan gambar panah dari masukan ke-2 ke masukan ke-7, sehingga $DP_{(2,8)}$ juga seimbang yang digambarkan oleh panah dari masukan ke-8 hingga masukan ke-2. Hasil tahap transisi seimbang adalah $DP_{(8,2)} = DP_{(6,2)} + DP_{(2,8)}$.

Akhirnya, hasil keluaran didapat dari iterasi linier pada $DP_{(N,y)}$, dimana *N* menyatakan jumlah lokasi total yang mencakup keseluruhan titik lokasi. Iterasi linier ini mencari nilai *a* dan *b* yang paling minimal dan akan dimasukkan ke Persamaan 2.3. Hasil akhir menyatakan panjang total minimal pipa yang akan dibangun.

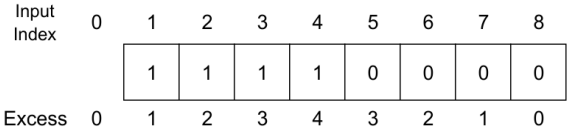
2.7. Contoh Penyelesaian Permasalahan SPOJ *ENVIRON*

Pada subbab ini akan dibahas mengenai tahap penyelesaian permasalahan dari salah satu contoh soal. Contoh masukan soal adalah *N = 8* dan konfigurasi lokasi '1 1 1 1 0 0 0 0'. Ilustrasi dari bentuk konfigurasi pipa contoh soal dapat dilihat pada Gambar 2.4. Hasil perhitungan akan disimpan dalam tabel *DP*. Tabel *DP* akan memuat hasil perhitungan berupa koefisien variabel *a*, *b*, dan ketinggian yang dinyatakan dalam tanda kurung. Tabel *DP* akan memuat 2 bentuk variabel yang dinyatakan sebagai berikut:

1. $x \rightarrow y$ berarti menghubungkan titik *x* ke titik *y* sebagai penggambaran submasalah pertama
2. $x \rightarrow y + [x-2][y+1]$ berarti menghubungkan titik *x* ke titik *y* ditambah dengan hasil pipa di bawahnya sebagai penggambaran submasalah kedua

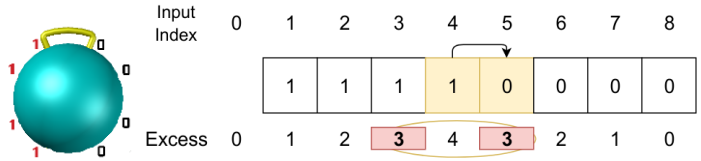
3. $[k][y] + [x-k][y+k]$ berarti menghubungkan 2 sub-masalah sebagai penggambaran tahap transisi

Tahap awal yang dilakukan adalah melakukan inisiasi $DP_{(0,y)}$ dimana *y* dimulai dari 0 hingga *N*. Tahap inisiasi $DP_{(0,y)}$ menyatakan koefisien variabel *a*, *b*, dan ketinggian bernilai 0. Tahap inisiasi juga menyatakan bahwa Excess_0 yang digunakan pada *prefix sum* bernilai 0. Selanjutnya, menghitung *prefix sum* sesuai Persamaan 2.4 pada masukan yang diterima. Hasil dari *prefix sum* adalah sebagai berikut '0 1 2 3 4 3 2 1 0' (Gambar 2.30).

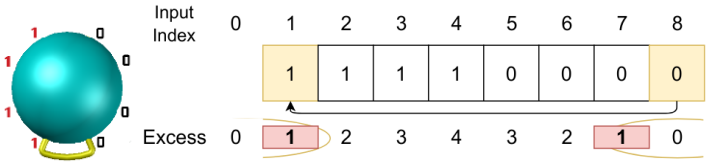


Gambar 2.30 Hasil prefix sum ketika *N=8*

Bagian selanjutnya adalah proses perhitungan submasalah. Iterasi dimulai pada *i=2* dan akan dilakukan pergeseran sebanyak 8 kali. Lokasi yang dapat dihubungkan pada iterasi ini digambarkan pada Gambar 2.31 - Gambar 2.32.



Gambar 2.31 Contoh Soal Iterasi *i=2* (Iterasi 2.3)



Gambar 2.32 Contoh Soal Iterasi *i=2* (Iterasi 2.7)

Dari iterasi di atas didapatkan 2 hubungan yaitu:

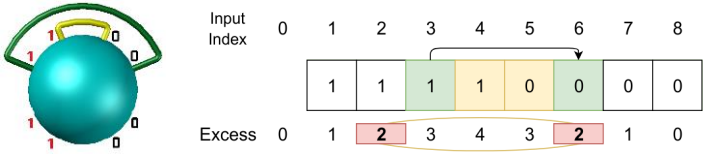
1. Masukan ke-4 menuju masukan ke-5
2. Masukan ke-8 menuju masukan ke-1

Hubungan yang digambarkan dengan persegi berwarna kuning dapat digolongkan sebagai submasalah pertama. Pada tahap ini, tabel *DP* akan dinyatakan sesuai Tabel 2.1 Tabel.

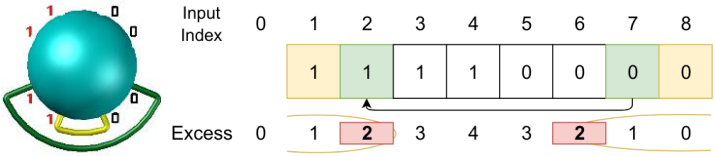
Tabel 2.1 Tabel DP Contoh Soal Iterasi *i=2*

0	1	2	3	4	5	6	7	8
2				4→5 <2, 3, 1>				8→1 <2, 3, 1>
4								
6								
8								

Iterasi dilanjutkan dengan *i=4* dan melakukan pergeseran sebanyak 8 kali. Iterasi saat *i=4* dengan lokasi yang dapat dihubungkan dinyatakan dengan Gambar 2.33 - Gambar 2.34.



Gambar 2.33 Contoh Soal Iterasi *i=4* (Iterasi 4.2)



Gambar 2.34 Contoh Soal Iterasi *i=4* (Iterasi 4.6)

Dari iterasi di atas didapatkan 2 hubungan yaitu:

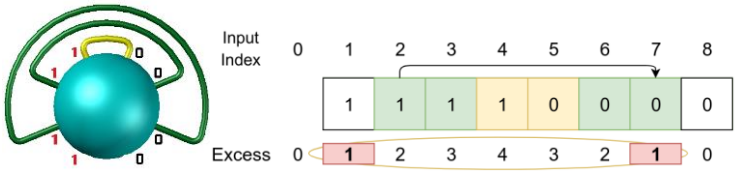
1. Masukan ke-3 menuju masukan ke-6 dengan di bawahnya masukan ke-4 menuju masukan ke-5 sesuai *i=2*
2. Masukan ke-7 menuju masukan ke-2 dengan di bawahnya masukan ke-8 menuju masukan ke-1 sesuai *i=2*

Hubungan yang digambarkan dengan persegi berwarna hijau menunjukkan iterasi telah melakukan perhitungan submasalah kedua. Pada tahap ini tabel *DP* akan dinyatakan pada Tabel 2.2 Tabel.

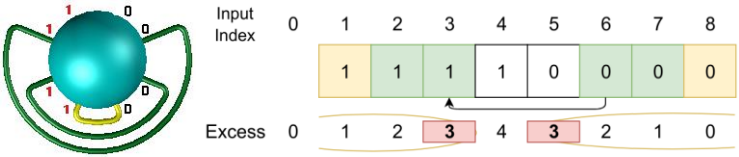
Tabel 2.2 Tabel *DP* Contoh Soal Iterasi *i=4*

0	1	2	3	4	5	6	7	8
2				4→5 <2, 3, 1>				8→1 <2, 3, 1>
4			3→6 + [2][4] <14,10,2>				7→2 + [2][8] <14,10,2>	
6								
8								

Iterasi dilanjutkan dengan *i=6* dan lokasi yang dapat dihubungkan dinyatakan pada Gambar 2.35 - Gambar 2.36.



Gambar 2.35 Contoh Soal Iterasi *i=6* (Iterasi 6.1)



Gambar 2.36 Contoh Soal Iterasi *i=6* (Iterasi 6.5)

Dari iterasi di atas didapatkan 2 hubungan yaitu:

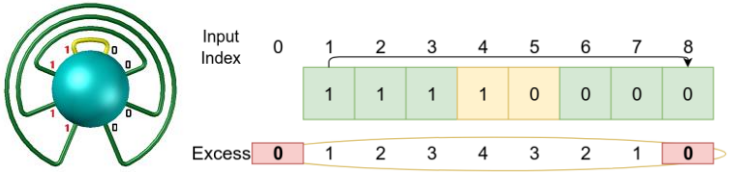
1. Masukan ke-2 menuju masukan ke-7 dengan di bawahnya masukan ke-3 menuju masukan ke-6 sesuai *i=4*
2. Masukan ke-6 menuju masukan ke-3 dengan di bawahnya masukan ke-7 menuju masukan ke-2 sesuai *i=4*

Hubungan yang digambarkan dengan persegi berwarna hijau menunjukkan pada iterasi ini telah dilakukan perhitungan submasalah kedua. Pada tahap ini tabel *DP* akan dinyatakan pada Tabel 2.3.

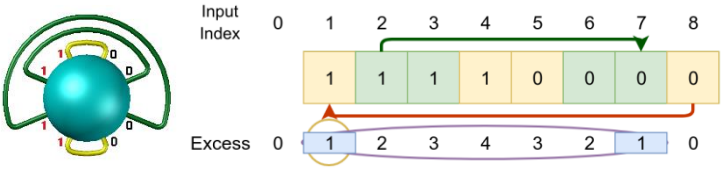
Tabel 2.3 Tabel *DP* Contoh Soal Iterasi *i=6*

0	1	2	3	4	5	6	7	8
2				4→5 <2, 3, 1>				8→1 <2, 3, 1>
4			3→6 + [2][4] <14,10,2>				7→2 + [2][8] <14,10,2>	
6		2→7 + [4][3] <44,21,3>				6→3 + [4][7] <44,21,3>		
8								

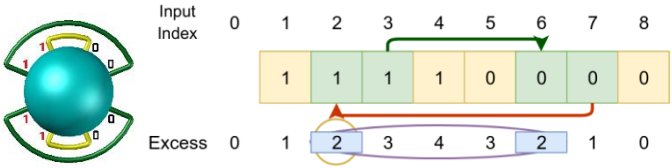
Iterasi terakhir adalah *i=8* yang mencakup keseluruhan lokasi. Lokasi yang dapat dihubungkan dinyatakan pada Gambar 2.37 - Gambar 2.41.



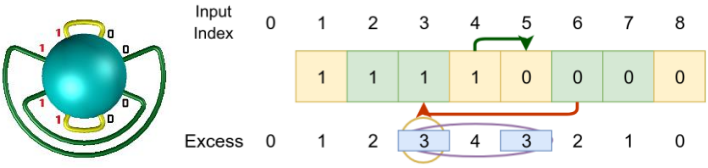
Gambar 2.37 Contoh Soal Iterasi *i=8* (Iterasi 8.0)



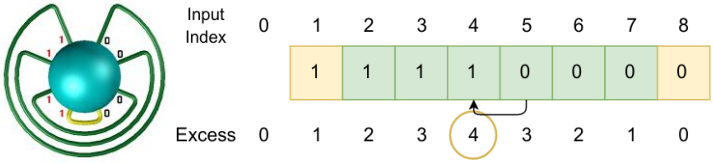
Gambar 2.38 Contoh Soal Iterasi *i=8* (Iterasi 8.1.6)



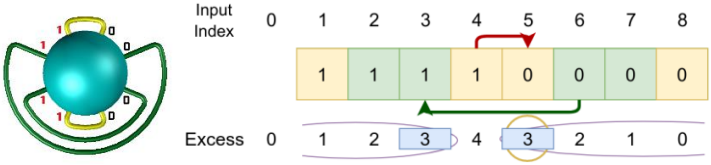
Gambar 2.39 Contoh Soal Iterasi *i=8* (Iterasi 8.2.4)



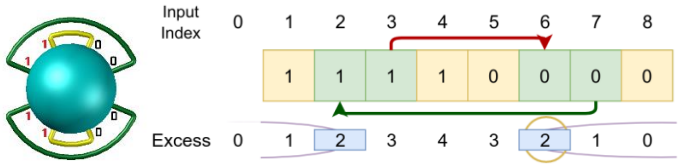
Gambar 2.40 Contoh Soal Iterasi *i=8* (Iterasi 8.3.2)



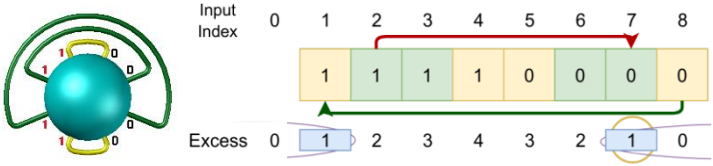
Gambar 2.41 Contoh Soal Iterasi *i=8* (Iterasi 8.4)



Gambar 2.42 Contoh Soal Iterasi *i=8* (Iterasi 8.5.6)



Gambar 2.43 Contoh Soal Iterasi *i=8* (Iterasi 8.6.4)



Gambar 2.44 Contoh Soal Iterasi *i=8* (Iterasi 8.7.2)

Pada gambar di atas, bentukan elips menyatakan bahwa pada tahap iterasi ini telah dijalankan tahap transisi dan seimbang. Seimbang pada tahap transisi ini dinyatakan dengan persegi warna biru pada bagian $Excess_i$. Panah berwarna hijau menyatakan $DP_{(k,y)}$ dan panah berwarna merah menyatakan $DP_{(x-k,y+k)}$. Tabel *DP* pada tahap akhir ini dinyatakan pada Tabel 2.4.

Tabel 2.4 Tabel *DP* Contoh Soal Iterasi i=8

0	1	2	3	4	5	6	7	8
2				4→5 <2, 3, 1>				8→1 <2, 3, 1>
4			3→6 + [2][4] <14,10,2>				7→2 + [2][8] <14,10,2>	
6		2→7 + [4][3] <44,21,3>				6→3 + [4][7] <44,21,3>		
8	1→8 + [6][2] <100,36,4>	[6][2] + [2][8] <46,24,3>	[4][3] + [4][7] <28,20,2>	[2][4] + [6][6] <46,24,3>	5→4 + [6][6] <100,36,4>	[6][6] + [2][4] <46,24,3>	[4][7] + [4][3] <28,20,2>	[2][8] + [6][2] <46,24,3>

Hasil akhir didapat dengan melakukan iterasi sebanyak 8 kali pada $DP_{(8,y)}$ dan dicari nilai yang paling minimal. Hasil paling minimal didapat dari $DP_{(8,3)}$ dengan variabel a = 28, variabel b =

20, dan ketinggian = 2. Hasil ini akan dimasukkan ke Persamaan 2.3 dan didapat hasil akhir sebesar 31,00 (Persamaan 2.9).

$$T = a * k + b$$
$$T = a\left(\frac{\pi}{N}\right) + b$$
$$T = 28\left(\frac{\pi}{8}\right) + 20$$
$$T \approx 31.00 \quad (2.9)$$

Panjang minimal pipa total dengan *N* = 8 dan konfigurasi '1 1 1 1 0 0 0 0' membutuhkan 31,00 satuan.

2.8. Pembuatan Generator Data Untuk Uji Coba

Generator data dibuat untuk menghasilkan kasus uji dalam rangka melihat performa dari algoritma yang dibuat pada Tugas Akhir ini. Generator data memberikan kasus uji yang disesuaikan dengan data masukkan yang dijelaskan pada subbab 4.2.1. Masukan berupa jumlah *N* lokasi diikuti dengan representasi lokasi yang terdiri dari angka '0' dan angka '1'. Proses pembuatan kasus uji dilakukan oleh generator data sesuai *pseudocode* dari Gambar 2.45 hingga Gambar 2.47.

GenerateRandom (N: Integer)	
1.	**initialize** satu = 0, nol = 0
2.	**for** i = 0 to N
3.	input[i] = **random** (0,1)
4.	if input[i] = 1
5.	**increment** satu
6.	else
7.	**increment** nol

Gambar 2.45 *Pseudocode* fungsi *GenerateRandom*

Pseudocode fungsi di atas digunakan untuk mencari masukan nilai sebagai representasi lokasi yang menghasilkan angka acak antara 0 atau 1 dan disimpan sebagai masukan.

ChangeInput ( )	
1.	diff = **absolute** (satu-nol)
2.	diff = diff / 2
3.	**while** diff > 0
4.	**if** satu > nol **and** input[i] = 1
5.	input[i] = 0
6.	**decrement** diff
7.	**else if** satu < nol **and** input[i] = 0
8.	input[i] = 1
9.	**decrement** diff

Gambar 2.46 *Pseudocode* fungsi *ChangeInput*

Pseudocode fungsi di atas digunakan untuk mengganti masukan angka karena jumlah angka 0 dan angka 1 pada masukan harus sama.

RandomTestCase()	
1.	**input** N
2.	t ← 100
3.	while t > 0
4.	print N
5.	generateRandom (N)
6.	changeInput()
7.	for i = 0 to N
8.	**print** input[i]
9.	**print** ‘0’

Gambar 2.47 *Pseudocode* fungsi *RandomTestCase*

Pseudocode fungsi di atas digunakan untuk menampilkan hasil akhir berupa 100 baris kasus uji yang terdiri dari sebuah bilangan *N* diikuti dengan angka 0 atau angka 1 sejumlah *N*.

BAB III
DESAIN

Pada bab ini akan dijelaskan analisis dan desain sistem yang digunakan untuk menyelesaikan permasalahan pada Tugas Akhir.

3.1. Desain Umum Sistem

Sistem menerima masukan berupa satu bilangan *N* yang merepresentasikan jumlah lokasi. Kemudian, masukan diikuti sebanyak *N* bilangan yang terdiri dari ‘0’ atau ‘1’. Angka ‘0’ merepresentasikan lokasi yang langka air, sedangkan angka ‘1’ merepresentasikan lokasi yang kaya air. Jumlah angka ‘0’ dapat dipastikan sebanyak angka ‘1’. Nilai maksimal dari bilangan *N* adalah sebesar 500 lokasi. Input akan berakhir pada kasus uji ketika *N=0*. Setelah itu, sistem akan memproses masukan yang akan diubah menjadi bentuk geometri dan dilakukan perhitungan dengan pemrograman dinamis. Pada pemrograman dinamis, hasil koefisien variabel *a* dan *b* sesuai Persamaan 2.3 dan ketinggian pipa akan dimemoisasi pada sebuah tabel *DP*. Akhirnya, sistem mencari nilai minimal yang mencakup seluruh *N* lokasi dan hasil nilai minimal akan dimasukkan ke persamaan geometri untuk mendapatkan hasil keluaran. Keluaran yang dihasilkan berupa panjang minimal pipa total yang akan dibangun untuk menghubungkan seluruh lokasi kaya air dan lokasi langka air tepat satu, dengan syarat pipa-pipa tersebut tidak ada yang saling bersilangan. *Pseudocode* dari fungsi *main* ditunjukkan pada Gambar 3.1.

Main()	
1.	**for** y = 0 to maxN
2.	DP[0][y] = init(0)
3.	**while** N > 0
4.	**input** N
5.	getInput(N)
6.	computePipe
7.	displayOutput

Gambar 3.1 Desain *Pseudocode* fungsi Main

3.2. Desain Algoritma

Sistem terdiri dari 8 fungsi, yaitu *balance*, seimbang, *getInput*, *setPipeLength*, *addPipe*, *leastLength*, *computePipe*, dan *displayOutput*. Sistem akan didukung oleh tipe data yang ditentukan bernama *Pipe*. Pada subbab ini akan dijelaskan desain algoritma dari masing-masing fungsi.

3.2.1. Desain Tipe Data *Pipe*

Tipe data *Pipe* adalah sebuah tipe struktur data yang menyimpan koefisien variabel *a*, variabel *b*, dan ketinggian pipa. Koefisien variabel *a* dan variabel *b* berasal dari Persamaan 2.3. *Pseudocode* tipe data *Pipe* ditunjukkan pada Gambar 3.2.

```
TYPE Pipe
    DECLARE a : Integer
    DECLARE b : Integer
    DECLARE height : Integer
END TYPE
```

Gambar 3.2 *Pseudocode* tipe data *Pipe*

3.2.2. Desain Fungsi *Balance*

Fungsi *Balance* digunakan untuk memberikan hasil indeks yang sesuai karena konfigurasi lokasi yang diberikan berbentuk linear, sedangkan pada penerapannya berbentuk sirkular. *Pseudocode* fungsi *Balance* ditunjukkan pada Gambar 3.3.

Balance (x: Integer) : Integer	

1.
2.
3.
4.
5.
	if x > N
	  return x-N
	else
	  return x
	end if

Gambar 3.3 *Pseudocode* fungsi *Balance*

3.2.3. Desain Fungsi Seimbang

Perbandingan jumlah lokasi kaya air dan lokasi langka air pada posisi *i* dan posisi *j* diproses pada fungsi Seimbang. Fungsi ini diterapkan pada variabel $Excess_i$ yang menyimpan hasil *prefix sum*. *Pseudocode* fungsi Seimbang ditunjukkan pada Gambar 3.4.

Seimbang (i: Integer, j: Integer) : Boolean	

1.
2.
3.
4.
5.
	If $excess_{j-1} == excess_{balance(j+i-1)}$
	  return true
	else
	  return false
	end if

Gambar 3.4 *Pseudocode* fungsi Seimbang

3.2.4. Desain Fungsi *GetInput*

Menerima masukan konfigurasi pipa dilakukan oleh fungsi *GetInput*. Masukan berupa angka 0 dan angka 1. Fungsi ini juga

sekaligus melakukan perhitungan *prefix sum* untuk setiap masukan. Pseudocode fungsi *GetInput* ditunjukkan pada Gambar 3.5.

getInput (N: Integer)	
1.	**for** i = 0 **to** N
2.	**input** lokasi
3.	**if** lokasi > 0
4.	$excess_i = excess_{i-1} + 1$
5.	**else**
6.	$excess_i = excess_{i-1} - 1$
7.	**end if**

Gambar 3.5 *Pseudocode* fungsi *GetInput*

3.2.5. Desain Fungsi *SetPipeLength*

Perhitungan koefisien variabel *a* dan *b* diambil dari jarak antara dua lokasi dan ketinggian pipa diproses pada fungsi *SetPipeLength*. *Pseudocode* fungsi *setPipeLength* ditunjukkan pada Gambar 3.6.

setPipeLength (distance: Integer, height: Integer) : Pipe	
1.	pipe: Pipe = new()
2.	Pipe.a = distance * 2 * height
3.	Pipe.b = distance + 2 * height
4.	Pipe.height = height
5.	**return** pipe

Gambar 3.6 *Pseudocode* fungsi *setPipeLength*

3.2.6. Desain Fungsi *AddPipe*

Penambahan jalur pipa di atas jalur pipa lain dilakukan pada fungsi *AddPipe*. Fungsi ini digunakan untuk mendefinisikan submasalah kedua dan menggabungkan dua submasalah saat tahap

transisi. *Pseudocode* fungsi *AddPipe* ditunjukkan pada Gambar 3.7.

```
addPipe (pipe1: Pipe, pipe2: Pipe) : Pipe
1.  pipe: Pipe = new()
2.  Pipe.a = pipe1.a + pipe2.a
3.  Pipe.b = pipe1.b + pipe2.b
4.  Pipe.height = max (pipe1.height, pipe2.height)
5.  return pipe
```

Gambar 3.7 *Pseudocode* fungsi *AddPipe*

3.2.7. Desain Fungsi *LeastLength*

Perbandingkan dua jalur pipa dan pencarian hasil paling minimal diproses pada fungsi *LeastLength*. *Pseudocode* fungsi *LeastLength* ditunjukkan pada Gambar 3.8 - Gambar 3.9.

```
leastLength (pipe1: Pipe, pipe2: Pipe) : Pipe
1.  if pipe1.a == -1
2.     return pipe2
3.  else if pipe2.a == -1
4.     return pipe1
5.  else if pipe1.a >= pipe2.a AND pipe1.b >= pipe2.b
6.     return  pipe2
7.  else if pipe1.a < pipe2.a AND pipe1.b < pipe2.b
8.     return pipe1
9.  else if pipe1.a < pipe2.a
10.    atas = pipe1.b - pipe2.b
11.    bawah = pipe2.a - pipe1.a
12.    if (atas * N > bawah *π)
13.       return pipe2
14.    else
15.       return pipe1
16.    end if
```

Gambar 3.8 *Pseudocode* fungsi *LeastLength*

```
17.  else
18.     atas = pipe2.b - pipe1.b
19.     bawah = pipe1.a - pipe2.a
20.     if (atas * N > bawah *π)
21.        return pipe1
22.     else
23.        return pipe2
24.  end if
```

Gambar 3.9 *Pseudocode* fungsi *LeastLength*

3.2.8. Desain Fungsi *ComputePipe*

Perhitungan pemrograman dinamis untuk menggabungkan submasalah diproses pada fungsi *ComputePipe*. *Pseudocode* fungsi *computePipe* ditunjukkan pada Gambar 3.10.

```
computePipe ()
1.  for i = 2 to N; i even
2.     for j = 1 to N
3.        if lokasi i,j seimbang
4.           DP[i][j] = init(-1)
5.           if input[j] ≠ input [i+j]
                COMPUTE setPipeLength
                DP[i][j] = addPipe(pipeLength, DP[i-2][j+1])
             end if

             for k = 2 to i-2; k even
                if lokasi k,j seimbang
                   minPipe = addPipe(DP[k][j], DP[i-k][j+k])
                   DP[k][j] = leastPipeLength(DP[k][j],minPipe)
                end if
        end if
```

Gambar 3.10 *Pseudocode* fungsi *ComputePipe*

3.2.9. Desain Fungsi *DisplayOutput*

Pencaian nilai paling minimal saat seluruh jarak telah tercakup diproses pada fungsi *DisplayOutput*. *Pseudocode* fungsi *displayOutput* ditunjukkan pada Gambar 3.11.

displayOutput ()	
1. 2. 3.	**for** i = 1 **to** N/2 output = leastLength(output, DP[N][i]) **display** output

Gambar 3.11 *Pseudocode* fungsi *DisplayOutput*

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Pada bab ini dijelaskan mengenai implementasi dari desain algoritma penyelesaian SPOJ *Environmental Engineering*.

4.1. Lingkungan Implementasi

Lingkungan implementasi berupa perangkat keras dan perangkat lunak yang digunakan adalah sebagai berikut:

1. Perangkat Keras
   Processor Intel® Core™ i5-5200U CPU @ 2.20GHz
   RAM 11.9 GB
   64-bit Operating System, x64-based processor
2. Perangkat Lunak
   Sistem Operasi Deepin 15.11 Debian 9.0 Stretch
   Visual Studio Code 1.39.1
   GNU C++ compiler 6.3.0

4.2. Rancangan Data

Subbab ini menjelaskan mengenai desain data masukan yang diperlukan untuk melakukan proses algoritma serta data keluaran yang dihasilkan oleh program.

4.2.1. Data Masukan

Data masukan adalah data yang akan diproses oleh program sebagai masukan menggunakan struktur data yang telah dirancang dan akan digunakan dalam algoritma pemrograman dinamis sebagai penyelesaian permasalahan ini. Data masukan berupa:

1. Masukan berupa bilangan bulat *N* yang merepresentasikan jumlah lokasi yang ada. Jumlah lokasi memiliki rentang mulai dari 2 hingga 500 lokasi. Masukan akan berakhir bila *N = 0*. Bilangan *N* dipastikan merupakan bilangan genap.
2. Masukan berupa angka '0' atau angka '1' sebanyak *N*. Angka '0' merepresentasikan lokasi yang langka air dan angka '1' merepresentasikan lokasi yang kaya air. Jumlah angka '0' akan dipastikan sebanyak angka '1'.

Contoh masukan dari permasalahan terdapat pada Gambar 4.1.

```
4
1 0 0 1
8
1 1 1 1 0 0 0 0
0
```

Gambar 4.1 Contoh data masukan

4.2.2. Data Keluaran

Data keluaran yang dihasilkan oleh program berupa nilai dengan ketelitian 2 digit desimal yang merepresentasikan total panjang minimal pipa yang akan dibangun. Pipa-pipa tersebut menghubungkan lokasi kaya air dengan lokasi langka air tepat satu dan tidak ada yang saling bersilangan.

Contoh keluaran dari permasalahan terdapat pada Gambar 4.2.

```
9.14
31.00
```

Gambar 4.2 Contoh data keluaran

4.3. Penggunaan *Library*, Konstanta, dan Variabel Global

Subbab ini menjelaskan mengenai *library*, konstanta dan variabel global yang digunakan dalam sistem. Implementasi algoritma akan membutuhkan 3 *library*, yaitu *cstdio*, *algorithm* dan

cmath. *Library cstdio* digunakan untuk menerima masukan dan menampilkan keluaran, *library algorithm* digunakan untuk membandingkan 2 nilai yang lebih besar dari fungsi *max*(a, b), dan *library cmath* digunakan untuk mendapatkan konstanta M_PI (simbol: π). *Library* yang digunakan akan ditampilkan pada Kode Sumber 4.1.

```
#include <cstdio>
#include <algorithm>
#include <cmath>
using namespace std;
```

Kode Sumber 4.1 *Library* yang digunakan

Variabel global digunakan untuk memudahkan dalam mengakses data yang digunakan antar fungsi, seperti pada Kode Sumber 4.2. Variabel *N* akan merepresentasikan jumlah lokasi yang ada, konstanta *MAXN* digunakan untuk mendefinisikan nilai maksimal dari total lokasi. Variabel *input[]* digunakan untuk menyimpan data masukan dan variabel *excess[]* digunakan untuk menyimpan hasil dari *prefix sum*.

```
int N=1;
const int MAXN = 505;
bool input[MAXN];
int excess[MAXN];
```

Kode Sumber 4.2 Variabel global yang digunakan

4.4. Implementasi Desain Algoritma

Pada subbab ini akan dijelaskan mengenai implementasi proses algoritma secara keseluruhan berdasarkan desain yang telah dijelaskan pada bab sebelumnya.

4.4.1. Implementasi Fungsi *Main*

Fungsi *main* adalah implementasi fungsi *pseudocode* dari subbab 3.1. Fungsi ini menjalankan algoritma pemrograman

dinamis yang digunakan dalam penyelesaian permasalahan. Implementasi fungsi *Main* terdapat pada Kode Sumber 4.3.

```
1.  int main() {
2.    for (int i=0;i<=MAXN;i++) {
3.      DP[0][i] = init(0, 0, 0);
4.    }
5.    excess[0] = 0;
6.
7.    while (N>0) {
8.      scanf("%d", &N);
9.      if (N==0) break;
10.
11.     getInput(N);
12.     computePipe();
13.     displayOutput();
14.   }
15.   return 0;
16. }
```

Kode Sumber 4.3 Implementasi fungsi *Main*

Fungsi main di atas pada baris ke-2 hingga ke-5, menetapkan $DP_{(0,y)}$ bernilai 0 yang berarti hubungan titik lokasi hanya ke diri sendiri dan menetapkan variabel Excess_0 yang digunakan pada *prefix sum* bernilai 0. Selanjutnya, mendapatkan masukan bilangan *N*. Fungsi *getInput* akan menerima masukan, fungsi *computePipe* akan melakukan perhitungan pemrograman dinamis, dan fungsi *displayOutput* akan menampilkan hasil total.

4.4.2. Implementasi Struct *Pipe*

Struct *Pipe* adalah implementasi tipe data struktur yang digunakan untuk menyelesaikan masalah sesuai *pseudocode* dari subbab 3.2.1. Struct menyimpan koefisien variabel *a* dan *b* dengan tipe data *Integer*. Selain itu, struct juga akan menyimpan ketinggian pipa maksimum. Struct *Pipe* akan digunakan pada tabel

DP dalam pemrograman dinamis. Implementasi struct *Pipe* terdapat pada Kode Sumber 4.4.

```
struct Pipe {
  int a;
  int b;
  int height;
} DP[MAXN][MAXN];
```

Kode Sumber 4.4 Implementasi atribut *struct Pipe*

4.4.3. Implementasi Fungsi *Balance*

Fungsi *Balance* adalah implementasi fungsi *pseudocode* dari subbab 3.2.2. Fungsi ini mengembalikan hasil indeks yang sesuai dengan bentuk sirkular. Implementasi fungsi *Balance* terdapat pada Kode Sumber 4.5.

```
1.  int balance(int x) {
2.    if (x>N) return x-N;
3.    return x;
4.  }
```

Kode Sumber 4.5 Implementasi fungsi *Balance*

4.4.4. Implementasi Fungsi Seimbang

Fungsi Seimbang adalah implementasi fungsi *pseudocode* dari subbab 3.2.3. Fungsi ini digunakan untuk membandingkan variabel $Excess_j$ dengan $Excess_{i-1}$ yang menyimpan hasil *prefix sum*. Implementasi fungsi Seimbang terdapat pada Kode Sumber 4.6.

```
1.  bool seimbang(int i, int j) {
2.   return excess[j-1] == excess[balance(j+i-1)];
3.  }
```

Kode Sumber 4.6 Implementasi fungsi Seimbang

4.4.5. Implementasi Fungsi *GetInput*

Fungsi *GetInput* adalah implementasi fungsi *pseudocode* dari subbab 3.2.4. Fungsi ini digunakan untuk mendapatkan masukan dan melakukan perhitungan *prefix sum* terhadap masukan. Implementasi fungsi *getInput* terdapat pada Kode Sumber 4.7.

```
1.  void getInput(int N) {
2.    for (int i=1;i<=N;i++) {
3.      scanf("%d", &input[i]);
4.      excess[i] = excess[i-1];
5.      if (input[i] == 1) {
6.        excess[i]++;
7.      } else {
8.        excess[i]--;
9.      }
10.   }
11. }
```

Kode Sumber 4.7 Implementasi fungsi *GetInput*

4.4.6. Implementasi Fungsi *SetPipeLength*

Fungsi *SetPipeLength* adalah implementasi fungsi *pseudocode* dari subbab 3.2.5. Fungsi ini digunakan untuk mendapatkan koefisien variabel *a*, *b*, dan ketinggian pipa berdasarkan parameter masukan berupa jarak lokasi dan ketinggian pipa. Implementasi fungsi *setPipeLength* terdapat pada Kode Sumber 4.8.

```
1.  Pipe setPipeLength (int distance, int height)
2.  {
3.    Pipe pipe;
4.    pipe.a = distance * 2 * height;
5.    pipe.b = distance + 2 * height;
6.    pipe.height = height;
7.    return pipe;
8.  }
```

Kode Sumber 4.8 Implementasi fungsi *SetPipeLength*

4.4.7. Implementasi Fungsi *AddPipe*

Fungsi *AddPipe* adalah implementasi fungsi *pseudocode* dari subbab 3.2.6. Fungsi ini digunakan untuk menambahkan pipa diatas jalur pipa lain. Ketinggian pipa didapatkan dari nilai terbesar hasil perbandingan pipa pertama dan pipa kedua. Implementasi fungsi *AddPipe* terdapat pada Kode Sumber 4.9.

```
1.  Pipe addPipe(Pipe pipe1, Pipe pipe2) {
2.    Pipe pipe;
3.    pipe.a = pipe1.a + pipe2.a;
4.    pipe.b = pipe1.b + pipe2.b;
5.    pipe.height = max(pipe1.height,
    pipe2.height);
6.    return pipe;
7.  }
```

Kode Sumber 4.9 Implementasi fungsi *AddPipe*

Berdasarkan kode sumber di atas, pipa akan disimbolkan sebagai $Pipe_{(a,\ b,\ height)}$. Contoh adalah ketika melakukan *addPipe* pada $Pipe1_{(2,3,1)}$ dan $Pipe2_{(12,7,2)}$ akan memberi hasil kembalian berupa pipa dengan nilai $Pipe_{(14,10,2)}$.

4.4.8. Implementasi Fungsi *LeastLength*

Fungsi *LeastLength* adalah implementasi fungsi pseudocode dari subbab 3.2.7. Fungsi ini digunakan untuk mencari pipa terkecil dari parameter masukan dua pipa. Implementasi fungsi *LeastLength* terdapat pada Kode Sumber 4.10 - Kode Sumber 4.11.

```
1.  Pipe leastLength (Pipe pipe1, Pipe pipe2) {
2.    if (pipe1.a == -1) {
3.      return pipe2;
4.    } else if (pipe2.a == -1) {
5.      return pipe1;
```

Kode Sumber 4.10 Implementasi fungsi *LeastLength*

```
6.      } else if (pipe1.a >= pipe2.a && pipe1.b >=
        pipe2.b) {
7.        return pipe2;
8.      } else if (pipe1.a < pipe2.a && pipe1.b <
        pipe2.b) {
9.        return pipe1;
10.     } else if (pipe1.a < pipe2.a) {
11.       double atas, bawah;
12.       atas = pipe1.b - pipe2.b;
13.       bawah = pipe2.a - pipe1.a;
14.       if (atas*(double)N > M_PI*bawah){
15.         return pipe2;
16.       } else {
17.         return pipe1;
18.       }
19.     } else {
20.       double atas, bawah;
21.       atas = pipe2.b - pipe1.b;
22.       bawah = pipe1.a - pipe2.a;
23.       if (atas*(double)N > M_PI*bawah){
24.         return pipe1;
25.       } else {
26.         return pipe2;
27.       }
28.     }
29.   }
```

Kode Sumber 4.11 Implementasi fungsi *LeastLength*

Berdasarkan kode sumber di atas, diantara dua pipa, disimbolkan dengan $Pipe_{(a,\ b,\ height)}$, akan dicari pipa yang bernilai minimal. Fungsi ini membagi kondisi menjadi 3 bagian utama. Pernyataan kondisi pertama, baris ke-3 hingga baris ke-6, adalah mengembalikan pipa lainnya, jika salah satu koefisien variabel *a* pada salah satu pipa bernilai -1. Contoh dari kejadian ini adalah $Pipe1_{(-1,-1,0)}$ dan $Pipe2_{(2,3,1)}$ akan memberi kembalian $Pipe2$. Pernyataan kondisi kedua, baris ke-7 hingga baris ke-10, adalah mengembalikan pipa yang memiliki koefisien variabel *a* dan *b* terkecil untuk keduanya. Contoh dari kondisi ini adalah

$Pipe1_{(14,10,2)}$ dan $Pipe2_{(4,6,1)}$ akan menghasilkan $Pipe2$. Pernyataan kondisi ketiga, baris ke-11 hingga baris ke-30, adalah mencari pipa terkecil bila dikalikan dengan koefisien $k = \frac{\pi}{N}$ dari Persamaan 2.3. Contoh dari kondisi ini adalah $Pipe1_{(164,65,4)}$ dan $Pipe2_{(184,63,4)}$ akan memberi kembalian $Pipe2$. Kondisi ini biasanya ditemui ketika jumlah N lokasi sudah besar saat N $\gtrsim$ 40.

4.4.9. Implementasi Fungsi *ComputePipe*

Fungsi *ComputePipe* adalah implementasi fungsi *pseudocode* dari subbab 3.2.8. Fungsi ini digunakan untuk melakukan perhitungan pemrograman dinamis pada submasalah. Implementasi fungsi *computePipe* terdapat pada Kode Sumber 4.12.

```
1.  void computePipe() {
2.   for (int i=2; i<=N; i+=2) {
3.    for (int j=1;j<=N;j++) {
4.     if (seimbang(i,j)) {
5.      DP[i][j]=init(-1,-1,0);
6.      if (input[j]!=input[balance(j+i-1)]){
7.       Pipe innerPipe = DP[i-2][balance(j+1)];
8.       Pipe outerPipe = setPipeLength(i-1,
         innerPipe.height+1);
9.       DP[i][j] = addPipe(innerPipe, outerPipe);
10.     }
11.     for (int k=2;k<i;k+=2) {
12.      if (seimbang(k,j)) {
13.       Pipe minPipe = addPipe(DP[k][j], DP[i-
          k][balance(j+k)]);
14.       DP[i][j] = leastLength(DP[i][j], minPipe);
15.      }
16.     }
17.    }
18.   }
19.  }
20. }
```

Kode Sumber 4.12 Implementasi fungsi *ComputePipe*

Dari kode soal di atas, perulangan digunakan untuk menghubungkan 2 titik lokasi setiap kelipatan genap dan mencoba setiap pergeseran titik. Jika ditemukan input yang berbeda angka, input tersebut dapat dihubungkan. Baris ke-7 hingga baris ke-9 menunjukkan hubungan pipa ditambahkan dengan pipa dibawahnya sesuai Persamaan 2.5. Tahap transisi pada baris ke-11 hingga ke-16 mencoba untuk meminimalisasi hubungan pipa dengan menggabungkan pipa yang tercakup pada jarak *k* dan jarak *i-k* sesuai Persamaan 2.8.

4.4.10. Implementasi Fungsi *Init*

Fungsi ini digunakan untuk memberi nilai pada koefisien variabel *a*, *b*, dan ketinggian pipa dan mengembalikan hasil berupa tipe data *Pipe*. Implementasi fungsi *setPipeLength* terdapat pada Kode Sumber 4.13.

```
1.  Pipe init(int a, int b, int height) {
2.    Pipe pipe;
3.    pipe.a = a;
4.    pipe.b = b;
5.    pipe.height = height;
6.    return pipe;
7.  }
```

Kode Sumber 4.13 Implementasi fungsi *Init*

4.4.11. Implementasi Fungsi *Output*

Fungsi *Output* digunakan untuk mendapatkan hasil akhir total panjang pipa minimal yang harus dibangun sesuai Persamaan 2.3. Implementasi fungsi *output* terdapat pada Kode Sumber 4.14.

```
1.  double output(Pipe pipe) {
2.    double k = M_PI / N;
3.    double ans = pipe.a * k + pipe.b;
4.    return ans;
5.  }
```

Kode Sumber 4.14 Implementasi fungsi *Output*

4.4.12. Implementasi Fungsi *DisplayOutput*

Fungsi *DisplayOutput* adalah implementasi fungsi *pseudocode* dari subbab 3.2.9. Fungsi ini digunakan untuk mencari pipa terpendek yang mencakup seluruh jarak *N*. Hasil akhir menampilkan panjang total minimal pipa yang diperlukan. Implementasi fungsi *displayOutput* terdapat pada Kode Sumber 4.15.

```
1.  void displayOutput() {
2.    Pipe ans = init(-1,-1,0);
3.    for (int i=1;i<=N/2+1;i++) {
4.      ans = leastLength(ans, DP[N][i]);
5.    }
      double totalPipe = output(ans);
      printf("%.2f\n", totalPipe);
    }
```

Kode Sumber 4.15 Implementasi fungsi *DisplayOutput*

[Halaman ini sengaja dikosongkan]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan tentang uji coba dan evaluasi dari implementasi sistem yang telah dilakukan pada Tugas Akhir.

5.1. Lingkungan Uji Coba

Lingkungan uji coba dilakukan pada situs penilaian daring SPOJ dengan kluster *Cube* yang memiliki spesifikasi sebagai berikut:

1. Perangkat Keras
   Processor Intel Xeon E3-1220 v5
   Memori limit 1536 MB
2. Perangkat Lunak
   Compiler C++ (g++ 4.3.2)

Lingkungan uji coba kinerja menggunakan komputer dengan spesifikasi perangkat keras dan perangkat lunak sebagai berikut:

1. Perangkat Keras
   Processor Intel® Core™ i5-5200U CPU @ 2.20GHz
   RAM 11.9 GB
   64-bit Operating System, x64-based processor
2. Perangkat Lunak
   Sistem Operasi Deepin 15.11 Debian 9.0 Stretch
   Visual Studio Code 1.39.1
   GNU C++ compiler 6.3.0

5.2. Evaluasi Kebenaran

Evaluasi dilakukan dengan mengecek masukan yang diberikan dibandingkan dengan keluaran yang dihasilkan dari

implementasi program. Hasil evaluasi harus sesuai dengan hasil keluaran dari permasalahan SPOJ *Environ*. Kasus uji yang digunakan sesuai dengan Tabel 5.1.

Tabel 5.1 Tabel kasus uji

Masukan	Keluaran
8 1 1 1 1 0 0 0 0	31.00

Dari implementasi program yang telah dibuat, sistem membutuhkan variabel untuk menyimpan *prefix sum* dalam bentuk *array* dengan nama *Excess* dan tabel *DP* dengan dimensi [*N/2*][*N*]. Tabel *DP* akan berisi tipe data *Struct* dengan atribut variabel *a*, *b*, dan ketinggian. Saat inisiasi awal, *Excess[0]* akan bernilai 0 dan tabel *DP* pada baris 0 akan bernilai (0, 0, 0). Setelah menerima masukan, lalu dilakukan perhitungan *prefix sum* dan disimpan pada variabel *Excess*. Hasil variabel *Excess* seperti pada Tabel 5.2.

Tabel 5.2 *Array Excess* menyimpan *prefix sum*

N	8							
Konfigurasi	1	1	1	1	0	0	0	0
Excess	1	2	3	4	3	2	1	0

Selanjutnya dilakukan iterasi yang dimulai 2 untuk menghubungkan lokasi yang saling bersebelahan hingga iterasi terakhir yang mencakup keseluruhan lokasi. Iterasi ini akan melakukan penggabungan submasalah. Hasil perhitungan dari setiap iterasi akan disimpan dalam tabel *DP* yang menyatakan hasil setiap iterasi (Tabel 5.3 - Tabel 5.6).

Tabel 5.3 Hasil tabel *DP* iterasi *i* = 2

1	2	3	4	5	6	7	8
(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(2, 3, 1)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(2, 3, 1)
(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)

Tabel 5.4 Hasil tabel *DP* iterasi *i* = 4

1	2	3	4	5	6	7	8
(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(2, 3, 1)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(2, 3, 1)
(0, 0, 0)	(0, 0, 0)	(14, 10, 2)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(14, 10, 2)	(0, 0, 0)
(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)

Tabel 5.5 Hasil tabel *DP* iterasi *i* = 6

1	2	3	4	5	6	7	8
(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(2, 3, 1)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(2, 3, 1)
(0, 0, 0)	(0, 0, 0)	(14, 10, 2)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(14, 10, 2)	(0, 0, 0)
(0, 0, 0)	(44, 21, 3)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(44, 21, 3)	(0, 0, 0)	(0, 0, 0)
(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)

Tabel 5.6 Hasil tabel *DP* iterasi *i* = 8

1	2	3	4	5	6	7	8
(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(2, 3, 1)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(2, 3, 1)
(0, 0, 0)	(0, 0, 0)	(14, 10, 2)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(14, 10, 2)	(0, 0, 0)
(0, 0, 0)	(44, 21, 3)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(44, 21, 3)	(0, 0, 0)	(0, 0, 0)
(100, 36, 4)	(46, 24, 3)	(28, 20, 2)	(46, 24, 3)	(100, 36, 4)	(46, 24, 3)	(28, 20, 2)	(46, 24, 3)

Setelah tabel *DP* telah mencakup keseluruhan lokasi, sistem akan memilih nilai paling minimal dari hasil tabel *DP* pada baris ke-*N*. Sesuai gambar di atas, hasil terkecil yang didapat

adalah pada tabel *DP[8][3]* dengan nilai (28, 20, 2). Hasil ini kemudian akan dimasukkan ke perhitungan sesuai dengan Persamaan 5.1.

$$T = 2xy\left(\frac{\pi}{N}\right) + 2x + y$$
$$T = a * k + b$$
$$T = a\left(\frac{\pi}{N}\right) + b$$
$$T = 28\left(\frac{\pi}{8}\right) + 20$$
$$T = 30.9955..$$
$$T \approx 31.00 \quad (5.1)$$

Hasil uji coba di atas yang telah dibulatkan dengan ketelitian 2 desimal sesuai dengan hasil keluaran yang diharapkan dari permasalahan yaitu sebesar 31,00 satuan.

5.3. Skenario Uji Coba

Pada subbab ini akan dijelaskan skenario uji coba yang dilakukan. Skenario akan digunakan untuk melakukan pengujian terhadap implementasi yang dibuat untuk menyelesaikan permasalahan *Environmental Engineering*. Skenario uji coba terdiri dari uji coba kebenaran dan uji coba kinerja.

5.3.1. Uji Coba Kebenaran

Uji coba kebenaran dilakukan dengan mengirimkan kode sumber program kedalam situs penilaian SPOJ. Permasalahan yang diselesaikan adalah permasalahan klasik SPOJ 7693 *Environmental Engineering*. Hasil uji coba dengan waktu terbaik pada situs penilaian SPOJ ditunjukkan pada Gambar 5.1.

24636929	□	2019-10-18 18:11:05	Environmental Engineering	**accepted** edit ideone it	0.01	4.6M	C++ 4.3.2

Gambar 5.1 Hasil uji coba pada situs penilaian SPOJ *Environ*

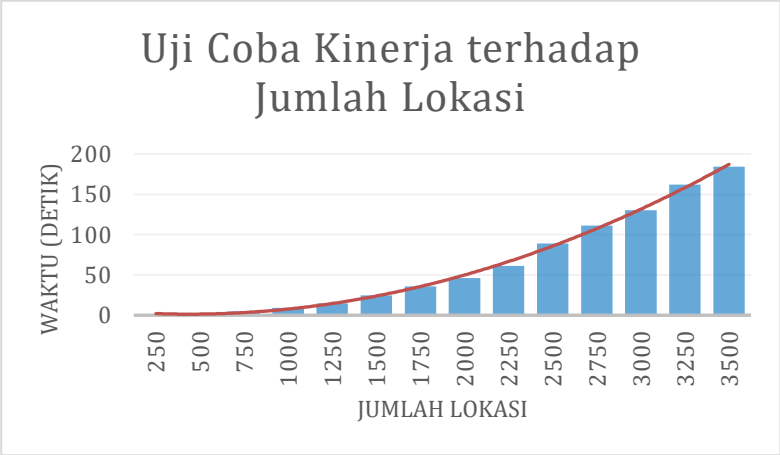
Dari hasil uji coba yang telah dilakukan, kode sumber program mendapatkan umpan balik *Accepted*. Waktu yang dibutuhkan program adalah 0.01 detik dengan memori sebesar 4.6 MB. Setelah itu, dilakukan pengujian dengan mengirimkan kode sumber sebanyak 10 kali pada situ penilaian SPOJ untuk melihat variasi waktu dan memori yang dibutuhkan program. Hasil uji coba sebanyak 10 kali terdapat pada Lampiran A.

5.3.2. Uji Coba Kinerja

Kompleksitas waktu untuk mendapatkan total panjang pipa minimal dengan menggunakan pemrograman dinamis adalah $O(N^3)$. Uji coba dilakukan dengan membuat variasi pada jumlah *N* lokasi dimulai dari 250 lokasi hingga 3.500 lokasi. Pada uji coba ini terdapat 100 kasus uji untuk setiap variasi yang dijalankan. Masukan untuk setiap uji coba berasal dari hasil generator data yang terdapat pada subbab 2.7. Uji coba kinerja dilakukan dengan menambahkan library C++ *std::chrono* pada program yang telah diimplementasikan sebelumnya. Hasil uji kinerja dinyatakan pada Tabel 5.7 yang dihitung dalam satuan detik dan disertai dengan penggambaran grafik pada Gambar 5.2.

Tabel 5.7 Hasil uji coba kinerja waktu terhadap jumlah N

Percobaan	Jumlah N	Waktu Hasil Uji Coba (detik)
1	250	0.384106
2	500	1.855981
3	750	4.807223
4	1000	9.136836
5	1250	14.838220
6	1500	24.549945
7	1750	35.722316
8	2000	46.271231
9	2250	61.331472
10	2500	89.185877
11	2750	111.213936
12	3000	130.517880
13	3250	162.061452
14	3500	184.240786



Gambar 5.2 Grafik uji coba kinerja terhadap jumlah *N* lokasi

Hasil yang didapatkan adalah waktu untuk komputasi meningkat secara kubik terhadap banyaknya jumlah *N* lokasi.

BAB VI
KESIMPULAN

Pada bab ini dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan serta saran yang dapat dilakukan.

6.1. Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap implementasi algoritma penyelesaian permasalahan *Environmental Engineering* dapat diambil beberapa kesimpulan sebagai berikut:

1. Pemodelan matematis yang digunakan dalam menyelesaikan permasalahan komputasi geometri adalah bentuk kesebangunan pipa air terhadap bumi untuk pendistribusian air.
2. Permasalahan *Environmental Engineering* dapat dipecah menjadi submasalah optimal dan submasalah tumpang tindih sehingga diselesaikan dengan menggunakan algoritma pemrograman dinamis.
3. Implementasi algoritma pemrograman dinamis yang ditunjang dengan struktur data yang memuat koefisien dari model matematis kesebangunan dapat menyelesaikan permasalahan dengan cepat dan tepat.
4. Pemrograman dinamis untuk menyelesaikan permasalahan komputasi geometri pada pendistribusian air memiliki kompleksitas waktu $O(N^3)$ dibuktikan dengan hasil uji coba kinerja yang menunjukkan grafik meningkat secara kubik terhadap jumlah lokasi kaya air dan lokasi langka air.

6.2. Saran

Saran yang diberikan dalam pengembangan algoritma pada permasalahan komputasi geometri adalah mencari algoritma yang memiliki kompleksitas waktu yang lebih baik dari $O(N^3)$.

DAFTAR PUSTAKA

[1] C. Kauth, "Sphere Online Judge," 27 October 2010. [Online]. Available: https://www.spoj.com/problems/ENVIRON/. [Accessed 11 September 2019].

[2] e. a. Thomas H. Cormen, Introduction to Algorithms Third Edition, London: The MIT Press, 2009.

[3] S. A. Mitchell, The New Primary Geography, Philadelphia: J.H. Butler & Co., 1876.

[4] A. M. Legendre, Elements of Geometry and Trigonometry, New York: A.S. Barnes & Co., 1867.

[Halaman ini sengaja dikosongkan]

LAMPIRAN A

Hasil Uji Coba pada *Sphere Online Judge* Sebanyak 10 Kali

Tabel 8.1 Hasil pengujian SPOJ *Environ* dengan C++ 4.3.2

No	Tanggal	Hasil	Waktu (detik)	Memori (MB)
1	2019-10-18 18:11:05	Accepted	0.01	4.6
2	2019-10-19 11:05:59	Accepted	0.01	4.5
3	2019-11-04 04:43:13	Accepted	0.01	4.4
4	2019-11-12 17:15:43	Accepted	0.01	4.7
5	2019-11-26 10:29:14	Accepted	0.01	4.6
6	2019-12-04 10:34:55	Accepted	0.01	4.6
7	2019-12-10 14:57:28	Accepted	0.01	4.7
8	2019-12-10 14:58:14	Accepted	0.01	4.4
9	2019-12-10 14:58:38	Accepted	0.01	4.5
10	2019-12-10 14:58:50	Accepted	0.01	4.4

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Natasha Valentina Santoso, lahir pada 17 Desember 1998 di Surabaya. Penulis menempuh studi kuliah program sarjana di Departemen Informatika Institut Teknologi Sepuluh Nopember (ITS). Penulis pernah menjadi asisten dosen dan praktikum untuk mata kuliah bagi mahasiswa Pendidikan Informatika dan Komputer Terapan ITS. Penulis juga aktif mengikuti organisasi kemahasiswaan dengan menjadi staff Departemen Teknologi pada Himpunan Mahasiswa Teknik Computer-Informatika (HMTC) ITS, staff Organisasi Kerohanian Katolik ITS, dan wakil ketua BEM Fakultas Teknologi Informasi dan Komunikasi (FTIK) ITS. Penulis dapat dihubungi melalui surel berikut: natasha.valentina1998@gmail.com