TUGAS AKHIR – KI141502

Desain dan Analisis Algoritma Dijkstra dan Struktur Data Two Dimensional Binary Indexed Tree pada Penyelesaian Persoalan: URI Online Judge Journey Through The Kingdom

BRAMA DIWANGKARA
05111540000150

Dosen Pembimbing
Rully Soelaiman, S.Kom., M.Kom.
Fajar Baskoro, S.Kom., M.T.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

TUGAS AKHIR – KI141502

Desain dan Analisis Algoritma Dijkstra dan Struktur Data Two Dimensional Binary Indexed Tree pada Penyelesaian Persoalan: URI Online Judge Journey Through The Kingdom

BRAMA DIWANGKARA
05111540000150

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

Dosen Pembimbing II
Fajar Baskoro, S.Kom., M.T.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2019

[*Halaman ini sengaja dikosongkan*]

FINAL PROJECT - KI141502

Design and Analysis of Dijkstra Algorithm and Two Dimensional Binary Indexed Tree Data Structure on Solving Problem: URI Online Judge Journey Through The Kingdom

BRAMA DIWANGKARA
05111540000150

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

Supervisor II
Fajar Baskoro, S.Kom., M.T.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
Sepuluh Nopember Institute of Technology
Surabaya, 2019

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

Desain dan Analisis Algoritma Dijkstra dan Struktur Data Two Dimensional Binary Indexed Tree pada Penyelesaian Persoalan: URI Online Judge Journey Through The Kingdom

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Algoritma Pemrograman
Program Studi S-1 Departemen Teknik Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
Brama Diwangkara
NRP: 05111540000150

Disetujui oleh Dosen Pembimbing Tugas Akhir:

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

Fajar Baskoro, S.Kom., M.T.
NIP. 197404031999031002 ..... (Pembimbing 2)

SURABAYA
JANUARI 2019

[*Halaman ini sengaja dikosongkan*]

Desain dan Analisis Algoritma Dijkstra dan Struktur Data Two Dimensional Binary Indexed Tree pada Penyelesaian Persoalan: URI Online Judge Journey Through The Kingdom

Nama Mahasiswa : Brama Diwangkara
NRP : 05111540000150
Departemen : Teknik Informatika FTIf - ITS
Dosen Pembimbing 1 : Rully Soelaiman, S.Kom., M.Kom.
Dosen Pembimbing 2 : Fajar Baskoro, S.Kom, M.T.

Abstrak

Permasalahan dalam Tugas Akhir ini merupakan sebuah permasalahan yang melibatkan sebuah rentang pencarian. Dalam permasalahan ini, diketahui terdapat sebuah negara yang masing – masing provinsinya membentuk sebuah pola seperti array dua dimensi. Pada soal ini diberikan jalur dan ongkos untuk berpindah dari satu provinsi ke provinsi lainnya. Dari berbagai jalur yang diberikan, struktur data harus mampu memilih jalur dengan harga terendah.

Pada Tugas Akhir ini, dirancang penyelesaian permasalahan Journey Through the Kingdom dalam pencarian jalur dengan ongkos terendah dari rentang yang diberikan. Permasalahan ini dapat diselesaikan dengan menggunakan Algoritma Dijkstra dan Struktur Data Binary Indexed Tree 2D untuk mendapatkan jalur dengan ongkos terendah secara optimal. Beberapa hal yang harus diperhatikan adalah mengenai cara pemilihan node yang belum dan sudah dipilih oleh Dijkstra melalui Struktur Data Binary Indexed Tree 2D.

Hasil dari Tugas Akhir ini telah berhasil menyelesaikan permasalahan di atas dengan cukup efisien, dengan kompleksitas waktu sebesar O(log(NM)) per query.

Kata kunci: binary indexed tree 2D, dijkstra, node, array.

[*Halaman ini sengaja dikosongkan*]

Desain dan Analisis Algoritma Dijkstra dan Struktur Data Two Dimensional Binary Indexed Tree Pada Penyelesaian Persoalan: URI Online Judge Journey Through The Kingdom

Student Name	: Brama Diwangkara
Registration Number	: 05111540000150
Department	: Informatics Department Faculty of IT – ITS
First Supervisor	: Rully Soelaiman, S.Kom., M.Kom.
Second Supervisor	: Fajar Baskoro, S.Kom, M.T.

Abstract

The problem in this final project is a problem that involves a search range. In this problem, it is known that there is a country where each province forms a pattern such as a two dimensional array. In this case, the route and costs are given to move from one province to another. From the various lines provided, the data structure must be able to choose the path at the lowest price.

In this Final Project, a solution will be designed to solve the Journey Through the Kingdom problem with the lowest cost of the given range. This problem can be solved by using the Dijkstra Algorithm and Binary Indexed Tree 2D data structures to get the path with the lowest cost optimally. Some things to note are about how to select nodes that have not been and have been selected by the Dijkstra through 2D Binary Indexed Tree data structure.

The results of this final project have succeeded in solving the above problems quite efficiently, with a time complexity of O (log (NM)) per query.

Keywords: binary indexed tree 2D, dijkstra, node.

[*Halaman ini sengaja dikosongkan*]

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Allah SWT atas karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul :

Desain dan Analisis Algoritma Dijkstra dan Struktur Data Two Dimensional Binary Indexed Tree pada Penyelesaian Persoalan: URI Online Judge Journey Through The Kingdom

Penelitian Tugas Akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Departemen Teknik Informatika Fakultas Teknologi Informati Institut Teknologi Sepuluh Nopember.

Dengan selesainya Tugas Akhir ini diharapkan apa yang telah dikerjakan penulis dapat memeberikan manfaat bagi perkembangan ilmu pengetahuan terutama di bidang teknologi informasi serta bagi diri penulis sendiri selaku peneliti.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama penulis mengerjakan Tugas Akhir maupun selama menempuh masa studi antara lain:

1. Allat SWT atas segala karuniaNya selama ini.
2. Ayah, ibu, dan keluarga penulis yang selalu memberikan dukungan, perhatian, dan kasih sayang bagi penulis yang menjadi semangat selama perkuliahan maupun pengerjaan Tugas Akhir ini.
3. Bapak Rully Soelaiman, S.Kom., M.Kom. selaku dosen pembimbing yang telah banyak meluangkan waktu untuk memberikan ilmu, nasihat, motivasi, bimbingan dan didikan kepada penulis dengan sabar selama perkuliahan maupun pengerjaan Tugas Akhir ini.
4. Bapak Fajar Baskoro, S.Kom., M.T. selaku dosen pembimbing yang telah memberikan masukan dan

bimbingan kepada penulis selama pengerjaan Tugas Akhir ini.

5. Seluruh tenaga pengajar dan karyawan Departemen Teknik Informatika ITS yang telah memberikan tenaga dan waktunya demi kelancaran proses belajar mengajar penulis selama perkuliahan.
6. Firza Gustama *partner* belajar mengajar penulis selama perkuliahan .
7. Teman-teman Wardug dan berkah *residence* yang telah menerima penulis dan menjadi keluarga penulis selama perkuliahan.
8. Teman-teman dari rumpun mata kuliah dan laboratorium Algoritma dan Pemrograman yang memberikan dukungan dan semangat kepada penulis selama pengerjaan Tugas Akhir ini.
9. Teman-teman angkatan 2015 jurusan Teknik Informatika ITS yang telah menemani perjuangan penulis selama 4 tahun masa perkuliahan.
10. Serta pihak-pihak lain yang tidak dapat disebutkan disini yang telah banyak membantu penulis dalam penyusunan Tugas Akhir ini.

Penulis mohon maaf apabila masih ada kekurangan pada Tugas Akhir ini. Penulis juga mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan di kemudian hari. Semoga melalui Tugas Akhir ini penulis dapat memberikan kontribusi dan manfaat yang sebaik-baiknya.

Surabaya, Januari 2019

Brama Diwangkara

DAFTAR ISI

[*Halaman ini sengaja dikosongkan*]

DAFTAR GAMBAR

DAFTAR TABEL

[*Halaman ini sengaja dikosongkan*]

DAFTAR KODE SUMBER

[*Halaman ini sengaja dikosongkan*]

BAB I
PENDAHULUAN

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

1.1 Latar Belakang

Di era modern ini, sudah banyak algoritma dan struktur data yang ditemukan. Masing-masing algoritma seringkali membutuhkan struktur data yang tepat untuk membantu menyelesaikan permasalahan tersebut. Salah satunya pada permasalahan rentang pencarian.

Topik Tugas Akhir ini mengacu pada permasalahan rentang pencarian pada permasalahan *Journey Through the Kingdom*. Permasalahan ini mencari jalan dengan *cost* paling kecil untuk sebuah perjalanan dari *node* yang ditempati menuju *node* yang akan dituju dengan melewati *node* lainnya yang berbentuk Dua Dimensi untuk posisi setiap *node*. Setiap *cost* merepresentasikan maksimum jarak antar *node* lainnya. Dari permasalah tersebut, dapat disimpulkan bahwa diperlukan solusi *Dijkstra* dan optimasi *Binary Indexed Tree 2D*.

Hasil Tugas Akhir ini diharapkan dapat memberi gambaran mengenai performa dari algoritma pencarian jarak terpendek *Dijsktra* dan struktur data *Binary Indexed Tree 2D* dalam penyelesaian permasalahan *Journey Through the Kingdom* secara optimal dan diharapkan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah:

1. Bagaimana menganalisis serta menentukan algoritma dan struktur data yang tepat untuk menyelesaikan

permasalahan *Journey Through the Kingdom* pada situs penilaian *URI Online Judge*?

2. Bagaimana performa algoritma *Dijkstra* dengan struktur data *Binary Indexed Tree 2D* pada permasalahan *Journey Through the Kingdom* pada situs penilaian *URI Online Judge*?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu:

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Batas waktu komputasi adalah 10 detik.
3. Jumlah R minimum satu.
4. Jumlah C maksimum 500.
5. Jumlah kolom maksimum adalah 500 kolom dan jumlah provinsi yang dituju antara 2 sampai 5 provinsi.
6. Rentang jumlah *cost* V_{ij} adalah $1 \leq V_{ij} \leq 1000$ dengan ij merupakan indeks provinsi.
7. Rentang jumlah baris penelusuran R_{ij} adalah $0 \leq R_{ij} \leq R$ dengan ij merupakan indeks provinsi.
8. Rentang jumlah kolom penelusuran C_{ij} adalah $0 \leq C_{ij} \leq C$ dengan ij merupakan indeks provinsi.

1.4 Tujuan Pembuatan Tugas Akhir

Tujuan dari pembuatan Tugas Akhir ini adalah:

1. Melakukan desain dan implementasi penyelesaian permasalahan *Journey Through the Kingdom* pada situs penilaian *URI Online Judge*.
2. Menganalisis hasil kinerja algoritma *Dijkstra* dengan struktur data *Binary Indexed Tree 2D* penyelesaian permasalahan *Journey Through the Kingdom*.

1.5 Manfaat Tugas Akhir

Tugas Akhir ini diharapkan dapat mengimplementasikan solusi dari permasalahan *Journey Through the Kingdom* sehingga dapat memberikan kontribusi pada pengembangan ilmu pengetahuan dan teknologi informasi.

1.6 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini yaitu:

1. **Penyusunan proposal Tugas Akhir.**
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir yang berisi gagasan untuk menyelesaikan permasalahan rope pada studi kasus permasalahan *Journey Through the Kingdom.*
2. **Studi literatur**
   Pada tahap ini dilakukan pencarian informasi dan studi literatur mengenai pengetahuan atau metode yang dapat digunakan dalam penyelesaian masalah. Informasi didapatkan dari materi-materi yang berhubungan dengan algoritma yang digunakan untuk penyelesaian permasalahan ini, materi-materi tersebut didapatkan dari buku, jurnal, maupun internet.
3. **Desain**
   Pada tahap ini dilakukan desain rancangan algoritma yang digunakan dalam solusi untuk pemecahan permasalahan *Journey Through the Kingdom.*
4. **Implementasi perangkat lunak**
   Pada tahap ini dilakukan implementasi atau realiasi 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 implementasi. Pengujian kebenaran dilakukan pada sistem penilaian dari *Uri Online Judge* sesuai dengan

masalah yang dikerjakan untuk diuji apakah luaran dari program telah sesuai.

6. **Penyusunan buku Tugas Akhir**

   Pada tahap ini dilakukan penyusunan buku Tugas Akhir yang berisi dokumentasi hasil pengerjaan Tugas Akhir.

1.7 Sistematika Penulisan

Buku Tugas Akhir ini merupakan laporan secara lengkap mengenai Tugas Akhir yang telah dikerjakan baik dari sisi teori, rancangan, maupun implementasi sehingga memudahkan bagi pembaca dan juga pihak yang ingin mengembangkan lebih lanjut. Sistematika penulisan buku Tugas Akhir secara garis besar antara lain:

Bab I Pendahuluan

Bab ini berisi penjelasan latar belakang, rumusan masalah, batasan masalah dan tujuan pembuatan Tugas Akhir. Selain itu, metodologi pengerjaan dan sistematika penulisan laporan Tugas Akhir juga terdapat di dalamnya.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Desain

Bab ini berisi penjelasan tentang desain dari sistem yang akan dibangun.

Bab IV Implementasi

Bab ini berisi implementasi berdasarkan desain algortima yang telah dilakukan pada tahap desain.

Bab V Uji Coba dan Evaluasi

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

Bab VI Penutup

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakuka.

[*Halaman ini sengaja dikosongkan*]

BAB II
DASAR TEORI

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

2.1 Deskripsi Umum Permasalahan

Permasalahan yang diangkat pada Tugas Akhir ini adalah permasalahan untuk mencari jalur dengan total *cost* terkecil pada sekumpulan indeks berbentuk matriks R×C dengan berbagai *cost* dan batasan penelusuran yang sudah didefinisikan. Setiap indeks provinsi P mempunyai relasi ke indeks provinsi P lainnya dan mempunyai nilai *cost* untuk melakukan penelusuran yang saling berelasi [1].

Setiap indeks mempunyai kesempatan untuk melakukan penelusuran ke indeks lainnya dengan maksimum R_{ij} baris dan C_{ij} kolom. Sebagai contoh persoalan dengan simulasi penyelesaian:

1. Terdapat matriks (3×4) dengan tiap – tiap indeks memiliki *cost* yang ditunjukan pada Gambar 2.1.

1	2	1	1
1	5	3	4
1	1	6	3

Gambar 2.1 Contoh persoalan indeks *cost*

2. Setiap indeks bisa melakukan perjalanan ke R_{ij} baris dan C_{ij} kolom yang ditunjukan pada Gambar 2.2.

R_{ij}

1	4	0	1
2	3	0	1
4	1	3	1

C_{ij}

1	2	3	3
3	3	1	2
0	0	0	1

Gambar 2.2 Contoh persoalan indeks *row* dan *column*

3. Untuk melakukan perjalanan dari indeks (1,1) ke (3, 4) membutuhkan *cost* minimum:
    - Dari indeks (1,1) biaya *cost* yang dikeluarkan adalah 1 dan bisa menempuh 1 baris dan 1 kolom, maka indeks selanjutnya yang bisa dituju adalah (1,2), (2,2) dan (2,1)
    - Dari indeks (2,2) biaya *cost* yang dikeluarkan adalah 5 dan bisa menempuh 3 baris dan 3 kolom, maka indeks (3,4) sudah tercapai dengan total *cost* 1 + 5 = 6.
    - Dari indeks (2,1) biaya *cost* yang dikeluarkan adalah 1 dan bisa menempuh 3 baris dan 2 kolom, maka indeks selanjutnya yang bisa dituju dan belum dilalui adalah (3,1), (3,2), (1,3), (2,3), (3,3)
    - Dari indeks (3,1) biaya *cost* yang dikeluarkan adalah 1 dan bisa menempuh 0 baris dan 4 kolom, maka indeks (3,4) sudah tercapai dengan total *cost* 1 + 1 + 1 = 3.
    - Dari indeks (3,3) biaya *cost* yang dikeluarkan adalah 6 dan bisa menempuh 0 baris dan 3 kolom, maka indeks (3,4) sudah tercapai dengan total *cost* 1 + 1 + 6 = 8.
    - Dari indeks (3,2) biaya *cost* yang dikeluarkan adalah 1 dan hanya bisa kembali ke indeks (3,3). Maka total *cost* 8 + 1 = 9.
    - Indeks (3,2), (1,3) dan (2,3) tidak bisa mencapai indeks (3,4) karena kolom yang bisa ditempuh 0.
    - Maka *cost* terkecil dari indeks (1,1) ke (3,4) adalah 3.

2.2 Deskripsi Umum

2.2.1 *Priority Queue*

Priority Queue adalah sebuah struktur abstrak seperti *queue* atau *stack* pada umumnya, tetapi di mana tambahan setiap elemen memiliki "prioritas" yang terkait denganya. Dalam antrian prioritas, elemen dengan prioritas tinggi berada pada posisi paling atas atau sering disebut *top*. Dalam beberapa implementasi, jika dua elemen memiliki prioritas yang sama, prioritas berada pada elemen yang pertama masuk ke dalam *Priority Queue* [2].

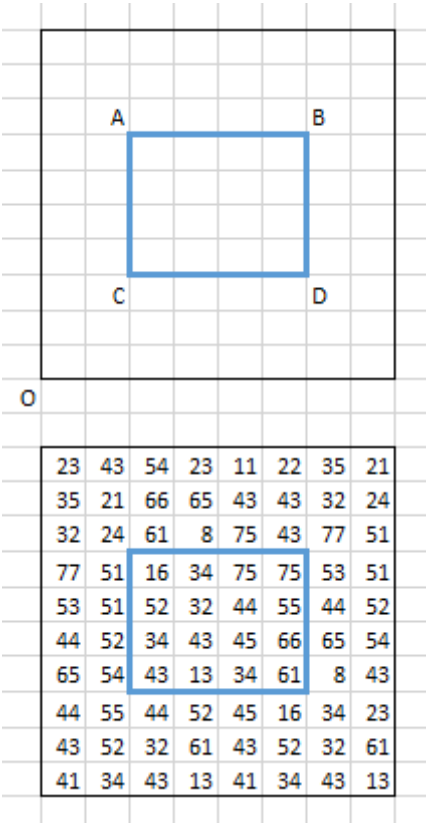
2.2.2 *Struct*

Struct adalah *composite data type* yang mendefinisikan daftar variabel yang dikelompokkan secara fisik untuk ditempatkan di bawah satu nama dalam blok memori, memungkinkan variabel yang berbeda dapat diakses melalui penunjuk tunggal, atau *struct* mendeklarasikan nama yang mengembalikan alamat yang sama. Tipe data *struct* dapat berisi banyak tipe data kompleks dan sederhana [2].

Pada *struct*, operasi deklarasi dan perubahan variabel dapat dilakukan dengan pemanggilan nama *struct* diikuti nama variabel dengan titik.

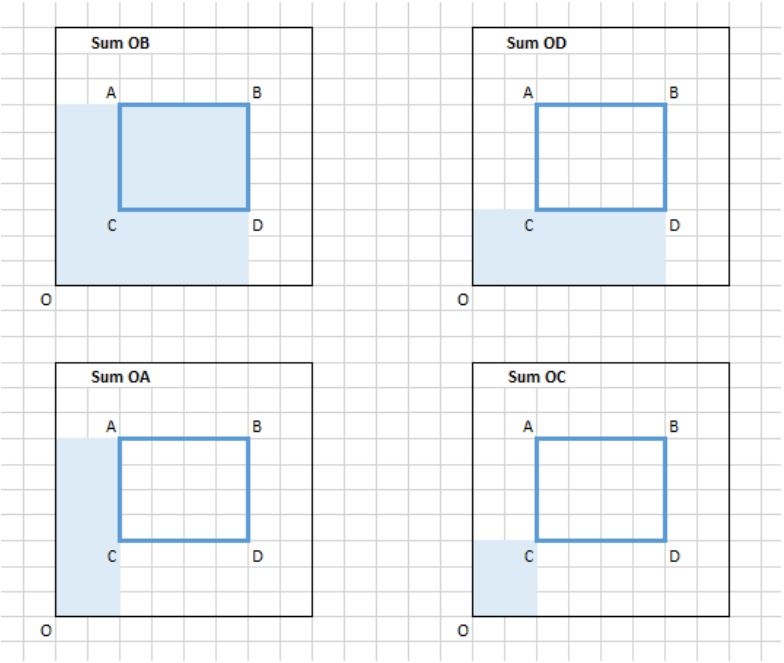
2.3 Struktur Data *Binary Indexed Tree 2D*

Binary Indexed Tree 2D adalah struktur data yang dapat secara efisien memperbarui elemen dan menghitung *prefix sum* yang direpresentasikan melalui *array 2D*.



Gambar 2.3 Penjelasan *Binary Indexed Tree 2D*

Pada Gambar 2.3, gambar merupakan *area* ABCD berisi angka yang akan dijumlahkan menggunakan *Binary Indexed Tree 2D.* Sebagai contoh untuk mencari luas *area* pada persegi ABCD pada Gambar 2.4.



Gambar 2.4 Penjelasan *Query Binary Indexed Tree 2D*

Diperlukan beberapa penjumlahan dan pengurangan untuk memperoleh *area* persegi ABCD, maka:

Sum(ABCD) = Sum(OB) – sum(OD) – sum(OA) + sum(OC)

2.3.1 Sum

Fungsi Sum merupakan penjumlahan dari indeks *area* yang ditentukan oleh fungsi Query pada Gambar 2.3.2. Pada fungsi ini, indeks *Binary Indexed Tree 2D* yang telah dibuat akan dijumlahkan sesuai dengan permintaan yang masuk ke dalam fungsi Sum. *Pseudocode* fungsi Sum ditunjukan pada Gambar 2.5.

```
Sum(x, y)
1.  sum = 0;
2.  for x = x to x > 0
3.          for y = y to y >0
4.            sum += BIT[x][y]
5.              y -= y&-y
6.          endfor
7.          x -= x&-x
8.  endfor
9.  return sum;
```

Gambar 2.5 ***Pseudocode Sum Binary Indexed Tree 2D***

2.3.2 Update

Fungsi Update digunakan untuk indeks yang akan mengalami perubahan pada *Binary Indexed Tree 2D*. *Pseudocode* fungsi Update *Binary Indexed Tree 2D* ditunjukan pada Gambar 2.6.

```
Update(Row, Col , x, y, val)
1.  for x = x to x <= Row
2.          for y = y to y <= Col
3.            BIT[x][y]+= val
4.              y += y&-y
5.          Endfor
6.          x += x&-x
7.  Endfor
```

Gambar 2.6 ***Pseudocode Update Binary Indexed Tree 2D***

2.4 Algoritma *Dijkstra* dengan *Priority Queue*

Algoritma *Dijkstra*, selalu disarankan untuk menggunakan *heap* (atau *Priority Queue*) sebagai operasi yang diperlukan (*extract minimum* dan *decrease key*) yang sesuai dengan spesialisasi *heap*. Namun, masalahnya adalah, *Priority Queue* tidak mendukung *decrease key*. Untuk mengatasi masalah ini, jangan perbarui *key*, tetapi masukkan satu salinan lagi. Jadi, sistem mengizinkan beberapa instance dari titik yang sama dalam *Priority Queue*. Pendekatan ini tidak perlu *decrease key operation*. Berikut merupakan jalanya algoritma *Dijkstra* dengan *Priority Queue*:

1. Inisialisasi semua jarak simpul tidak terhingga.
2. Membuat sebuat *Priority Queue* pq kosong. Setiap element dari pq merupakan sebuah *pair* ataupun *struct* (berisi posisi simpul dan *weight*).
3. Implementasikan prioritas pada *Priority Queue*.
4. *Insert* pq dengan simpul yang akan dimasukan, maka posisi teratas dari pq merupakan hasil dari *Dijkstra* dengan prioritas yang ditetapkan.

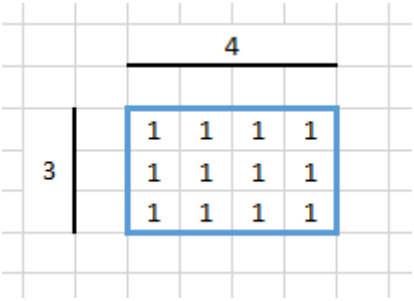
2.5 Strategi Penyelesaian Permasalahan

Strategi yang akan digunakan untuk menyelesaikan permasalahan *Journey Through the Kingdom* ini merupakan gabungan dari algoritma *Dijkstra* dan struktur data *Binary Indexed Tree 2D* yang akan dibagi menjadi tiga bagian strategi utama.

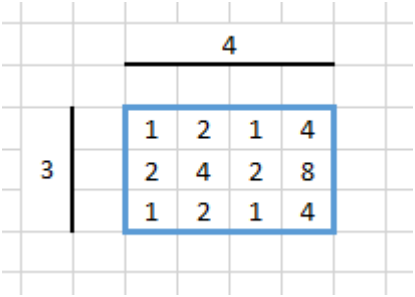
2.5.1 Inisialisasi *Binary Indexed Tree 2D* dan *Priority Queue*

Operasi inisialisasi ini bertujuan untuk mengatur semua *Priority Queue* menjadi kosong. Tujuan menjadikan kosong *Priority Queue* supaya tidak terjadi *crash* untuk kasus persoalan sebelumnya dan memberi semua indeks menjadi angka satu untuk *Binary Indexed Tree 2D*. Indeks bernilai satu merupakan representasi bahwa indeks tersebut belum pernah dilewati oleh fungsi Update pada *Binary Indexed Tree 2D*.

Setelah melewati fungsi Init maka indeks akan masuk ke dalam fungsi Modify untuk awal pembentukan *Binary Indexed Tree 2D*. Setelah inisialisasi awal, fungsi Modify digunakan untuk mengubah indeks sesuai permintaan untuk tujuan bahwa indeks tersebut sudah ditelusuri. Sebagai contoh sesuai dengan kasus masukan untuk matriks berukuran 3×4 yang ditunjukan pada Gambar 2.7.

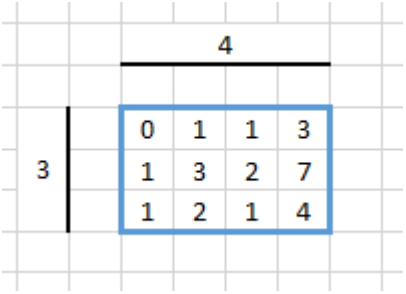


Gambar 2.7 Inisialisasi 1 pada matriks R×C



Gambar 2.8 Modify awal pada matriks R×C sesuai *Binary Indexed Tree 2D*

Jika kita jumlahkan *value* indeks (1,1) sampai (3,4) pada Gambar 2.7 maka jumlahnya adalah delapan yang menandakan semua indeks belum dilakukan penelusuran. Perhitungan juga bisa dihitung dari Gambar 2.8 dengan rumus Sum yang dijelaskan pada sub bab 2.3.1, kompleksitas perhitungan akan lebih cepat dengan jumlah akhir yang sama.

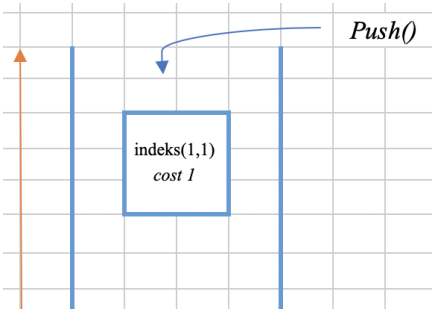


Gambar 2.9 Modify pada indeks (1,1) pada *Binary Indexed Tree* 2D

Gambar 2.9 merepresentasikan bahwa indeks (1,1) telah dilakukan penelusuran, maka dari itu indeks (1,1) masuk pada fungsi Modify dengan *value* negatif satu untuk menunjukan bahwa suatu indeks sudah melakukan penelusuran. Jika kita hitung total indeks (1,1) sampai (3,4) dengan rumus Sum maka akan

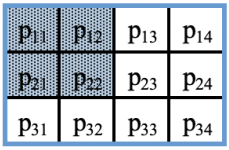
menghasilkan jumlah tujuh yang merepresentasikan ada tujuh provinsi yang belum melakukan penelusuran.

Setelah melakukan inisialisasi pada *Binary Indexed Tree 2D*, akan dilanjutkan memasukan indeks asal ke dalam *Priority Queue* dengan prioritas *cost* yang terkecil. Sebelum melakukan *push* terhadap indeks asal, akan dilakukan *pop* untuk seluruh isi dari *Priority Queue* terlebih dahulu supaya tidak *crash* terhadap kasus soal sebelumnya.



Gambar 2.10 *Push* pada *Priority Queue*

Setelah melakukan *push* terhadap *Priority Queue*, akan dipanggil *top* dan *pop* terhadap prioritas yang paling tinggi pada *Priority Queue* untuk dilakukan penelusuran terhadap indeks tersebut. Untuk menentukan batasan – batasan penelusuran pada R_{ij} dan C_{ij} pada provinsi p_{ij}, akan dilakukan operasi untuk memperoleh batas baris paling atas dan paling bawah serta kolom paling kanan dan paling kiri yang sebagai permisalan ditunjukan pada Gambar 2.11.

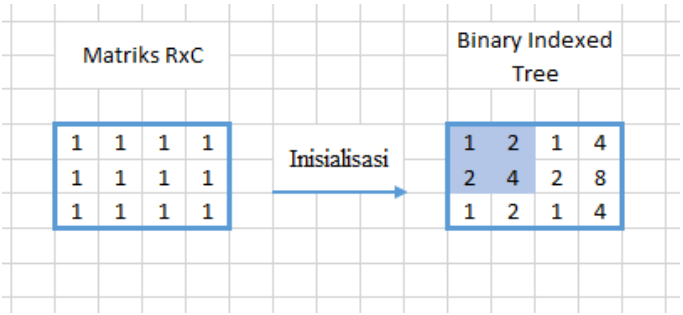


Gambar 2.11 Batas penelusuran *Binary Indexed Tree 2D*

2.5.2 Penelusuran *Dijkstra* dengan *Binary Indexed Tree 2D*

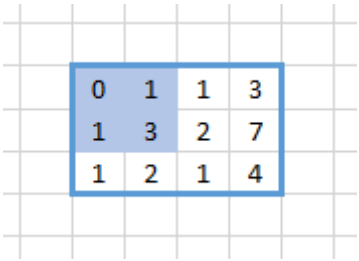
Fungsi Update pada *struct Binary Indexed Tree 2D* ini bertujuan untuk mengubah *cost* pada *Priority Queue* dan memanggil fungsi Modify untuk mengubah nilai indeks dari *Binary Indexed Tree 2D* yang ada untuk tujuan perubahan *unvisited point area*. Sebagai contoh bagaimana fungsi ini bekerja adalah:

1. Sebuah matriks 3×4 akan dilakukan penelusuran dari indeks (1,1) menuju indeks (2,2).
2. Untuk kondisi awal *Binary Indexed Tree 2D* akan dibentuk dari nilai indeks satu yang ditunjukan pada Gambar 2.12.



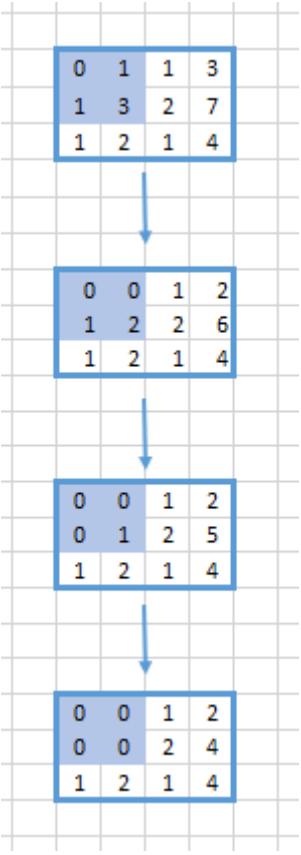
Gambar 2.12 Inisialisasi pembentukan *Binary Indexed Tree 2D*

3. Fungsi Modify akan *update Binary Indexed Tree 2D* pada indeks (1,1) sebagai tanda bahwa indeks tersebut telah ditelusuri dan sudah melakukan *pop* karena merupakan inisiator dari penelusuran. Maka *Binary Indexed Tree 2D* yang berubah ditunjukan pada Gambar 2.13.

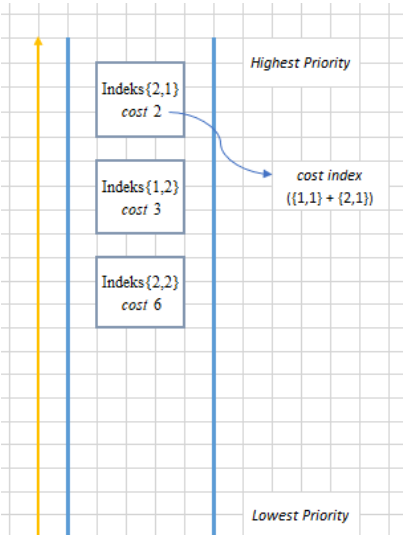


Gambar 2.13 Modify Indeks (1,1) pada *Binary Indexed Tree 2D* pada penelusuran

4. Setelah melakukan fungsi Modify maka fungsi Update akan memastikan indeks yang belum terlewati, jika masih ada yang belum terlewati (fungsi RectangleSum tidak sama dengan 0) maka akan dilanjutkan penelusuran kembali.
5. Penelusuran kembali dilakukan untuk indeks lainnya yang belum dilewati. Indeks yang telah dilewati akan masuk pada *Priority Queue* dengan penambahan *cost value* yang indeks bawa dengan *cost value* yang sebelumnya. Untuk permisalan *cost value* indeks (1,1), (1,2), (2,1), (2,2) adalah 1, 2, 1, 5 dan indeks penelusuran dimulai dari (1,1), maka *cost value* indeks (1,1) akan ditambah oleh indeks lainnya yang masuk pada *Priority Queue*.



Gambar 2.14 Update indeks *Binary Indexed Tree 2D*

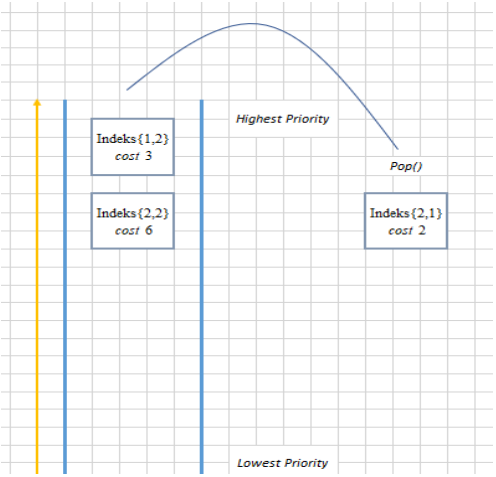


Gambar 2.15 *Priority Queue* update pada penelusuran

2.5.3 *Update Cost* pada *Priority Queue*

Operasi ini bertujuan untuk mencari jarak terkecil yang mungkin untuk sampai ke indeks tujuan, setiap *top* dari list *queue* merepresentasikan jarak terpendek dari penelusuran *Dijkstra*.

Pada saat melakukan *pop* pada list *Priority Queue* fungsi akan memastikan, jika indeks *top* bisa melakukan penelusuran sampai ke indeks yang dituju (setiap indeks mempunyai kemampuan untuk melakukan penelusuran sejumlah X baris dan Y kolom) maka fungsi selesai dan *return cost value* yang ada, jika belum sampai indeks akan menjadi inisiator untuk melakukan penelusuran ulang sesuai dengan penjelasan sub bab 2.5.2**.**



Gambar 2.16 *Pop* pada *Priority Queue*

Dari Gambar 2.16, jika indeks *top* bisa melakukan penelusuran sampai dengan indeks yang dituju maka fungsi selesai dan return *cost* value. Sebaliknya, jika jarak maksimal X baris dan Y kolom indeks *top* tidak bisa mencapai indeks yang dituju maka indeks tersebut kembali melakukan penelusuran *Dijkstra* dengan *Binary Indexed Tree 2D.*

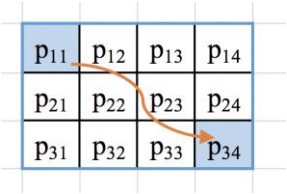
2.6 Simulasi Persoalan dengan Strategi Penyelesaian

Pada sub bab ini akan dijelaskan simulasi sesuai dengan contoh persoalan *Journey Through the Kingdom* pada kasus pertama ditunjukan pada Gambar 2.17.

```
3 4 5
1 2 1 1
1 5 3 4
1 1 6 3
1 2 3 3
3 3 1 2
0 0 0 1
1 4 0 1
2 3 0 1
4 1 3 1
1 1
3 4
1 1
2 2
2 2
```

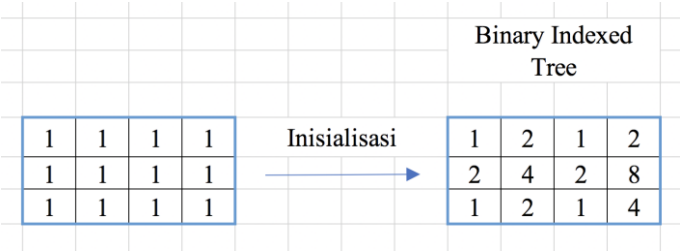
Gambar 2.17 Masukan pada simulasi persoalan

Pada Gambar 2.17, input pada baris pertama (3,4,5) merupakan representasi dari jumlah baris R, jumlah kolom C, dan jumlah provinsi N yang akan ditelusuri. 3 kali R baris selanjutnya merepresentasikan *cost* dari masing – masing provinsi, maksimum R_{ij} penelusuran baris pada provinsi (i, j) dan maksimum C_{ij} penelusuran kolom pada provinsi (i, j). N baris selanjutnya merepresentasikan provinsi P_1, P_2, P_3, …, P_N yang akan ditelusuri dengan mencari penelusuran sesuai dengan urutan dari $P_1 \rightarrow P_2$, $P_2 \rightarrow P_3$, $P_3 \rightarrow P_4$, …, $P_{N-1} \rightarrow P_N$. Kasus pertama dari masukan pada Gambar 2.6.1 merupakan pencarian dari *cost* terkecil dari provinsi p_{11} ke p_{34} yang ditunjukan Gambar 2.18.



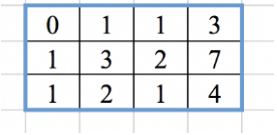
Gambar 2.18 Provinsi penelusuran pada p_{11} sampai p_{34}

Untuk tahap pertama akan dilakukan inisialisasi awal pembentukan *Binary Indexed Tree 2D* sesuai dengan jumlah baris dan kolom berdasarkan masukan (3, 4) dengan nilai satu untuk setiap indeks yang merepresentasikan bahwa indeks tersebut belum melakukan penelusuran. Gambar inisialisasi awal ditunjukan pada Gambar 2.19.



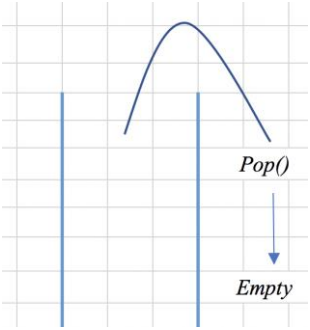
Gambar 2.19 fungsi Init R = 3 C = 4 pada simulasi

Setelah melakukan inisialisasi awal, akan dilakukan modifikasi indeks (1,1) pada *Binary Indexed Tree 2D* sesuai contoh kasus yang merepresentasikan indeks tersebut telah ditelusuri. *Binary Indexed Tree 2D* setelah dipanggil fungsi Modify pada indeks (1,1) ditunjukan pada Gambar 2.20.



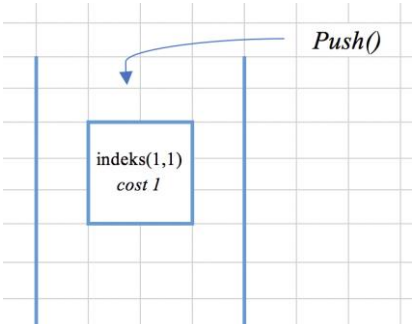
Gambar 2.20 Modify indeks p_{11}

Setelah pemanggilan fungsi Modify pada awal indeks pada *Binary Indexed Tree 2D* akan dilakukan *pop* pada *Priority Queue* sampai *Priority Queue* pada kondisi *empty* yang ditunjukan pada Gambar 2.21.



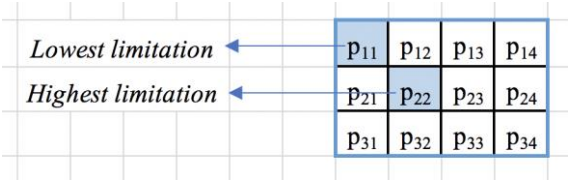
Gambar 2.21 Empty() pada setiap awal kasus persoalan

Setelah *Priority Queue* dalam keadaan *empty* maka akan dimasukan indeks p_{11} dengan *cost* sesuai dengan contoh persoalan. Pada kasus ini, *cost* pada p_{11} untuk melakukan penelusuran adalah satu, maka *Priority Queue* yang terbentuk ditunjukan pada Gambar 2.22.



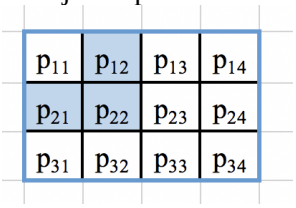
Gambar 2.22 *Push* indeks p_{11} pada *Priority Queue*

Sebelum melakukan penelusuran akan dilakukan *pop* terhadap *Priority Queue* dengan prioritas yang paling tinggi, pada kasus ini p_{11} dengan indeks (1,1), *cost* 1 merupakan prioritas yang paling tinggi. Kemudian akan dilakukan pengecekan apakah p_{11} bisa melakukan penelusuran sampai p_{34} dan pada kasus ini p_{11} tidak dapat melakukan penelusuran sampai ke p_{34}, maka akan dilakukan penelusuran pada provinsi lainnya. Setelah melakukan pengecekan, akan dilakukan inisialisasi batas minimum dan batas maksimum terhadap penelusuran dari p_{11}. Batas minimum merepresentasikan baris dan kolom yang paling kecil yang pada penulisan ini akan direpresentasikan dengan lambang lr dan lc, sedangkan batas maksimum merepresentasikan baris dan kolom yang paling besar yang pada penulisan ini akan direpresentasikan dengan lambang rr dan rc. Pada kasus ini batas indeks batas minimum merupakan (1,1) dan indeks batas maksimum merupakan (2,2). Batas minimum dan maksimum ditunjukan pada Gambar 2.23.



Gambar 2.23 Batas minimum dan maksimum penelusuran p_{11}

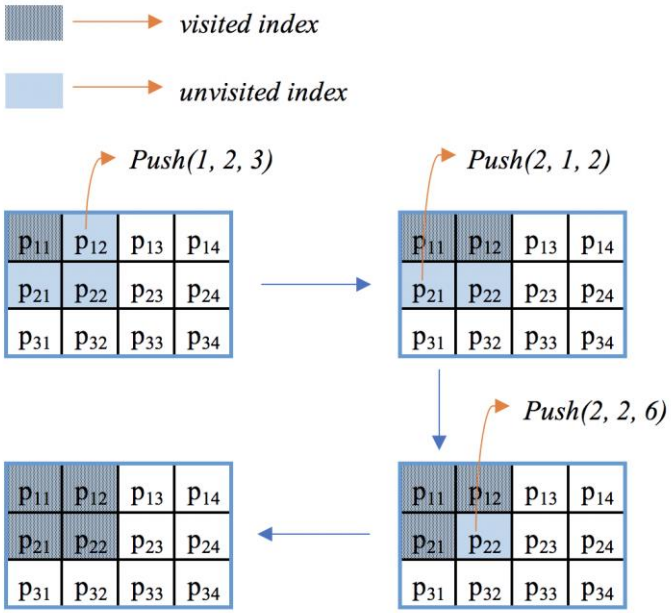
Berdasarkan batas maksimum dan minimum dari yang ditentukan, maka diperoleh indeks yang akan ditelusuri merupakan p_{12}, p_{21} dan p_{22} yang ditunjukan pada Gambar 2.24.



Gambar 2.24 *Adjacency Matrix* penelusuran p_{11}

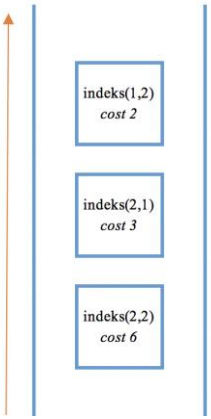
Setelah melakukan identifikasi penelusuran yang bisa dilakukan dari p_{11}, akan di lakukan penelusuran terhadap setiap provinsi yang mungkin ditelusuri. Untuk langkah awal, akan dipanggil fungsi Sum pada *Binary Indexed Tree 2D* untuk menghitung jumlah *unvisited point area* yang merupakan jumlah indeks yang akan dilakukan penelusuran. Pada kasus ini, indeks yang belum dilakukan penelurusan berjumlah 3 yaitu p_{12}, p_{21} dan p_{22}.

Setelah dilakukan identifikasi batas manimum dan maksimum, akan dilakukan penelusuran *Dijkstra* dengan *Binary Indexed Tree 2D*. Langkah – langkah penelusuran akan dilakukan mulai dari lc sampai rc dilanjutkan dengan lr sampai lc. Setiap dilakukan penelusuran, akan dilakukan *push* pada *Priority Queue* terhadap posisi indeks baris x dan kolom y serta *cost* provinsi yang ditelusuri p_{xy} ditambah dengan *cost* indeks posisi awal p_{11}. Penelusuran akan ditunjukan pada Gambar 2.25.



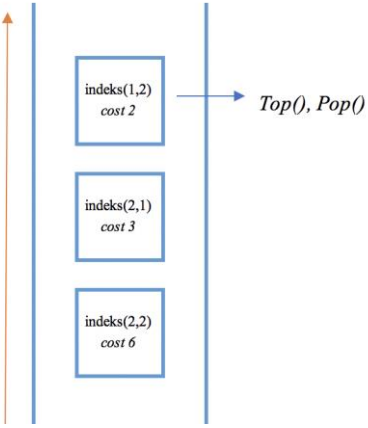
Gambar 2.25 Penelusuran *Dijkstra* pada p_{12}, p_{21} dan p_{22}

Hasil *Priority Queue* pada penelusuran indeks ditunjukan pada Gambar 2.26.



Gambar 2.26 Hasil *Priority Queue* pada penelusuran p_{11}

Setelah semua indeks dari penelusuran p_{11} ditelusuri, akan dilanjutkan penelusuran dengan pemanggilan *top* dan *pop* pada *Priority Queue* untuk memilih *cost* terkecil dari penelusuran. Pada kasus ini p_{12}, dengan total *cost* dua akan dipilih untuk melakukan penelusuran selanjutnya karena mempunyai nilai *cost* penelusuran terkecil dari p_{11}.



Gambar 2.27 *Pop* pada *Priority Queue* pada indeks p_{12}

BAB III
DESAIN

Pada bagian ini akan dijelaskan desain dan algoritma untuk menyelesaikan permasalahan pada Tugas Akhir ini.

3.1 Permasalahan *Binary Indexed Tree* 2D

Pada permasalahan ini sistem akan dibangun untuk menyelesaikan tiga jenis operasi yaitu mengubah indeks penelusuran, fungsi RectangleSum menghitung *unvisited point*, dan fungsi Update untuk untuk perhitungan *cost value* dari masing – masing indeks yang akan masuk pada penelusuran *Dijkstra*.

3.1.1 Deskripsi Umum Sistem

Program akan diawali dengan menerima masukan pada fungsi Main berupa jumlah baris R, jumlah kolom C dan jumlah provinsi T yang akan dituju. Setiap provinsi menempati baris X dan kolom Y yang berbeda – beda. Dari data masukan pertama, program akan lanjut menerima tiga kali X baris baru yang masing – masing baris mempunyai Y kolom untuk representasi:

1. Tiga kali baris pertama merepresentasikan *cost* V untuk setiap provinsi baris X kolom Y
2. Tiga kali baris kedua merepresentasikan X_{ij} yang merupakan jumlah maksimal baris yang bisa ditelusuri dari provinsi baris X kolom Y
3. Tiga kali baris ketiga merepresentasikan Y_{ij} yang merupakan jumlah maksimal kolom yang bisa ditelusuri dari provinsi baris X kolom Y

Berdasarkan inputan yang ada, output merupakan nilai *cost* terkecil dari provinsi$_{x-1}$ sampai dengan provinsi $_{x}$. *Pseudocode* fungsi Main ditunjukan pada Gambar 3.1.

```
Main()
1.  R = input() //jumlah baris
2.  C = input() //jumlah kolom
3.  T = input() //jumlah kota
4.  for i = 1 to R
5.          for j = 1 to C
6.                  V[i][j] = input()
7.  for i = 1 to R
8.          for j = 1 to C
9.                  R[i][j] = input()
10. for i = 1 to R
11.         for j = 1 to C
12.                 C[i][j] = input()
13. for i = 1 to T
14.         X[i] = input()
15.         Y[i] = input()
```

Gambar 3.1 *Pseudocode* Fungsi Main

3.1.2 Desain *Struct* RangeTree

Struct RangeTree digunakan untuk implementasi *Binary Indexed Tree 2D* dengan beberapa fungsi [3] :

1. Init
   Fungi Init untuk identifikasi awal semua indeks belum ditelusuri. Fungsi ini memanggil fungsi Modify untuk inisialisasi semua indeks dengan angka 1 yang berarti semua indeks belum ditelusuri.
2. Modify
   Fungsi ini untuk update data array pada fungsi Init untuk inisialisasi awal *Binary Indexed Tree 2D* dan indeks yang akan di eksekusi pada fungsi Update pada *struct* RangeTree.

3. RectangleSum
   Fungsi ini digunakan sebagai perhitungan indeks yang belum dilewati. Perhitungan ini hanya digunakan pada indeks yang mungkin dilewati.
4. Query
   Fungsi ini untuk menjumlahkan indeks dari *Binary Indexed Tree* 2D sesuai dengan permintaan fungsi RectangleSum.
5. Update
   Fungsi Update digunakan untuk menelusuri indeks yang belum dilewati untuk dimasukan pada data struktur *Dijkstra*. Setelah indeks tersebut dilewati, maka fungsi ini akan memanggil fungsi Modify untuk menandakan bahwa indeks tersebut sudah dilewati.

3.1.3 Desain Fungsi Findpath

Pada saat semua input sudah selesai, maka fungsi utama (*int main*) akan memanggil fungsi Findpath dengan parameter:

1. Posisi indeks asal X[i] dan Y[i]
2. Posisi indeks tujuan X[i+1] dan Y[i+1]
3. Jumlah baris R dan kolom C

Langkah awal adalah melakukan *statement* jika indeks asal dan tujuan sama maka *return nol*, kemudian Fungsi Findpath akan memanggil fungsi Init pada *struct* RangeTree untuk inisialisasi pembentukan indeks matriks R×C dan memanggil fungsi Modify dengan parameter negatif satu untuk menandakan bahwa indeks asal sudah ditelusuri. Setelah fungsi pemanggilan fungsi Init dan Modify akan dimasukan data parameter pada *Priority Queue* dengan parameter indeks asal X[i] dan Y[i] dan *cost value* yang telah pada variabel global, sebelum input awal ke dalam *Priority Queue*, *queue* dikosongkan terlebih dahulu.

Pada langkah selanjutnya, fungsi Findpath akan memastikan *node* yang ada pada *Priority Queue* apakah sudah bisa mencapai tujuan indeks yang dituju. Jika sudah tercapai tujuan maka akan

return cost value yang pada ada *node* yang telah di keluarkan dari *queue*, jika tidak maka fungsi Findpath akan memanggil fungsi Update untuk melakukan penelusuran ke indeks tujuan. Parameter dari fungsi Update yang dipanggil adalah:

1. Maksimal penelusuran terhadap indeks y dan -y dengan indeks y = 0 berada pada posisi awal indeks
2. Maksimal penelusuran terhadap indeks x dan -x dengan indeks x = 0 berada pada posisi awal indeks
3. *Cost value* indeks awal
4. *Parameter* -1 yang menunjukan indeks awal yang belum ditelusuri.

Pseudocode fungsi Findpath ditunjukan pada Gambar 3.2.

```
goal index row = ex
goal index column = ey
start index row = sx
start index column = sy

Findpath(R, C, sx, sy, ex, ey)
1.  If sx == ex and sy == sx
2.          return 0
3.  Init(R, C)
4.  Modify(sx, sy, -1)
5.  while priority queue as pQ not empty
6.          pQ.Pop()
7.  priority queue Push(sx, sy, cost value)
8.  while priority queue as pQ not empty
9.          u = pQ.top(), pQ.pop()
10.         fx = abs(ex – u.sx)
11.         fy = abs(ey – u.sy)
12.         if fx <= R[u.sx][u.sy] and fy <= C[u.sx][u.sy]
13.                 Return cost value u
14.         lr = max (1, u.sx - R[u.sx][u.sy]) //top of row
15.         rr = min(R, R[u.sx][u.sy]) //bottom of row
16.         lc = max (1, u.sy - C[u.sx][u.sy]) //left of column
17.         rc = min(C, C[u.sx][u.sy]) //right of column
18.         Update(lr, rr, lc, rc, value cost u , -1)
19. Return -1
```

Gambar 3.2 *Pseudocode* Fungsi Findpath

3.1.4 Desain Fungsi Init

Fungsi Init pada *struct* RangeTree adalah untuk pembentukan indeks awal yang merepresentasikan bahwa semua indeks belum melakukan penelusuran. Fungsi Init memanggil

fungsi Modify untuk pembentukan *Binary Indexed Tree 2D* dengan nilai satu. *Pseudocode* Fungsi Init ditunjukan pada Gambar 3.3.

```
Init(R, C)
1.  for i = 1 to R
2.          for j = 1 to C
3.                  Modify(i, j, 1)
```

Gambar 3.3 *Pseudocode* Fungsi Init

3.1.5 Desain Fungsi Modify

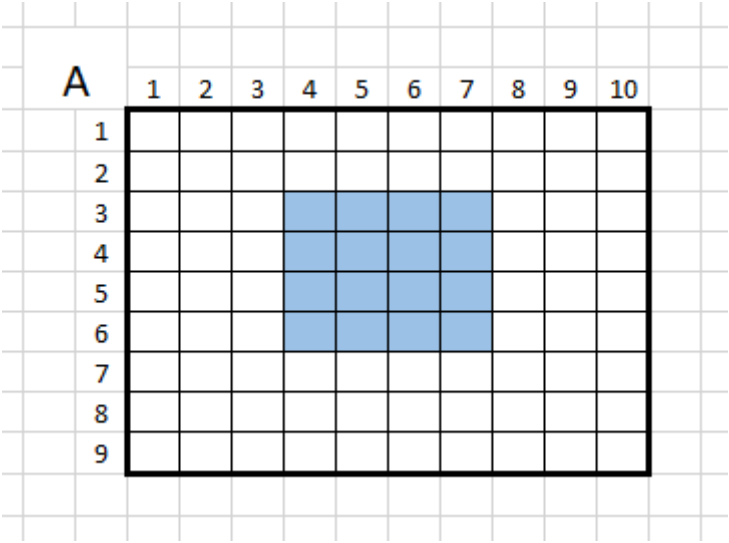
Fungsi Modify pada *struct* RangeTree digunakan untuk perubahan indeks pada matriks *Binary Indexed Tree 2D* yang bertujuan untuk mengubah *visited point area* yang menunjukan jumlah *area* yang sudah ditelusuri oleh penelusuran *Dijkstra* pada fungsi Update. Fungsi Modify dipanggil oleh fungsi Init untuk inisialisasi bahwa semua indeks belum ditelusuri. Untuk memanggil fungsi Modify dibutuhkah parameter posisi indeks (x, y) dan value (1 untuk inisialisasi awal, -1 untuk perubahan bahwa indeks posisi telah dilewati penelusuran). *Pseudocode* fungsi Modify ditunjukan pada Gambar 3.4.

```
Modify(x, y, val)
1.  for i = x to R
2.          for j = y to C
3.                  Indexed Tree[i][j] += val
4.                  j += j&(-j)
5.          i += i&(-i)
```

Gambar 3.4 *Pseudocode* Fungsi Modify

3.1.6 Desain Fungsi RectangleSum

Fungsi RectangleSum digunakan untuk pencarian *unvisited point area* yang merupakan jumlah indeks yang belum dilewati pada *Binary Indexed Tree 2D*. Fungsi RectangleSum memanggil fungsi Query pada *struct* RangeTree untuk menghitung jumlah *area* dari pencarian masing – masing *Binary Indexed Tree 2D*. Penjelasan dan perhitungan dari fungsi RectangleSum ditunjukan pada sub bab 2.3.



Gambar 3.5 Batasan RectangleSum

Sebagai permisalan, untuk menghitung *area* berwarna biru pada Gambar 3.5 maka fungsi RectangleSum akan memanggil fungsi Query dengan parameter:

$$Query(6,7) - Query(2,7) - Query(6,3) + Query(2,3)$$

Pseudocode fungsi RectangleSum ditunjukan pada Gambar 3.6.

```
RectangleSum (lx, rx, ly, ry)
    1. x = query(rx, ry) - query(lx-1, ry) - query(rx, ly-1) +
       query(lx-1, ly-1)
    2. return x
```

Gambar 3.6 *Pseudocode* Fungsi RectangleSum

3.1.7 Desain Fungsi Query

Setelah dirumuskan oleh Fungsi RectangleSum, perhitungan penjumlahan *area* indeks dilakukan oleh Fungsi Query. Fungsi Query merupakan fungsi yang berisi parameter posisi indeks yang akan dilakukan kalkulasi jumlah *visited point area* dari posisi indeks sampai indeks (0,0). *Pseudocode* dari fungsi Query ditunjukan pada Gambar 3.7.

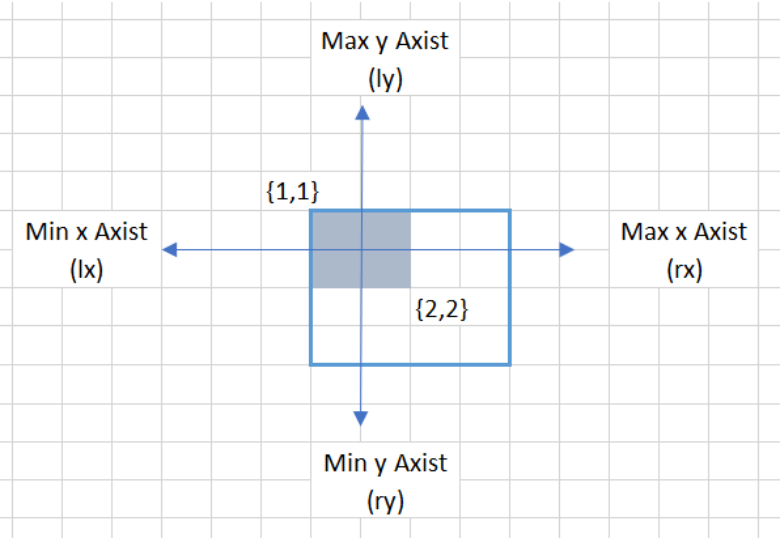
```
Query (x, y)
    1. Ret = 0
    2. for i = x to 0
    3.         for j = y to 0
    4.                 Ret += Index Tree [i][j]
    5.                 j -= j & (-j)
    6.         i = i&(-i)
    7. Return Ret
```

Gambar 3.7 *Pseudocode* Fungsi Query

3.1.8 Desain Fungsi Update

Fungsi Update pada *struct* RangeTree digunakan untuk penelusuran *Dijkstra*. Pada awalnya akan fungsi Update akan memanggil fungsi RectangleSum untuk menghitung total indeks yang belum ditelusuri. Setelah mendapatkan jumlah indeks, fungsi akan melakukan pengecekan jika tidak ada indeks yang belum ditelusuri maka fungsi akan *return* 0.

Parameter yang ada pada fungsi Update adalah sesuai yang dijelaskan pada sub bab 3.1.3 dengan mencari batas – batas penelusuran yang membawa sebuah *cost value* dan *unvisited point area*. Batas penelusuran merupakan maksimal dan minimal aksis y maupun x, sebagai contoh *Dijkstra* akan menelusuri indeks dari (0,0) dengan maksimal penelusuran terhadap indeks x adalah 1 dan maksimal penelusuran terhadap indeks y adalah 1. Maka *area* yang terbentk dari data tersebut ditunjukan pada Gambar 3.8.



Gambar 3.8 Batasan - batasan *area* fungsi Update

Setelah perhitungan *area* selesai, akan dilanjutkan dengan penelusuran untuk mendapatkan nilai indeks tersebut dan memasukanya pada *Priority Queue* dengan penambahan *cost value* pada penelusuran sebelumnya. *Pseudocode* dari fungsi Update dapat ditunjukan pada Gambar 3.9.

```
Update(lx, rx, ly, ry, value , visited)
1.  if tot == 0
2.      return
3.  if lx == rx
4.      if ly == ry
5.          pQ.push(Node(lx, ly, v[lx][ly] + value))
6.          Modify (lx, ly , -1)
7.          Return
8.      int count = RectangleSum (lx, rx, ly, (ly+ry)/2,
    value, count)
9.      if count
10.         Update(lx, rx, ly, (ly+ry)/2, value, count)
11.     if count < visited
12.         Update(lx, rx, (ly+ry)/2, ry, value, visited -
    count)
13. else
14.     int count = RectangleSum (lx, (rx+lx)/2, ly, ry)
15.     if count
16.         Update(lx, (lx+rx)/2, ly, ry, value, count)
17.     if count < visited
18.         Update((lx+rx)/2, rx, ly, ry, value, visited -
    count)
```

Gambar 3.9 *Pseudocode* Fungsi Update

3.2 Permasalahan *Priority Queue Dijkstra*

Pada bab ini akan dijelaskan bagaimana operasi *Dijkstra* dengan *Priority Queue* dengan prioritas yang ditetapkan pada *struct* Node.

3.2.1 Deskripsi umum sistem

Setelah melakukan penelusuran *Dijkstra* dengan *Binary Indexed Tree 2D* maka setiap kali melakukan penelusuran terhadap

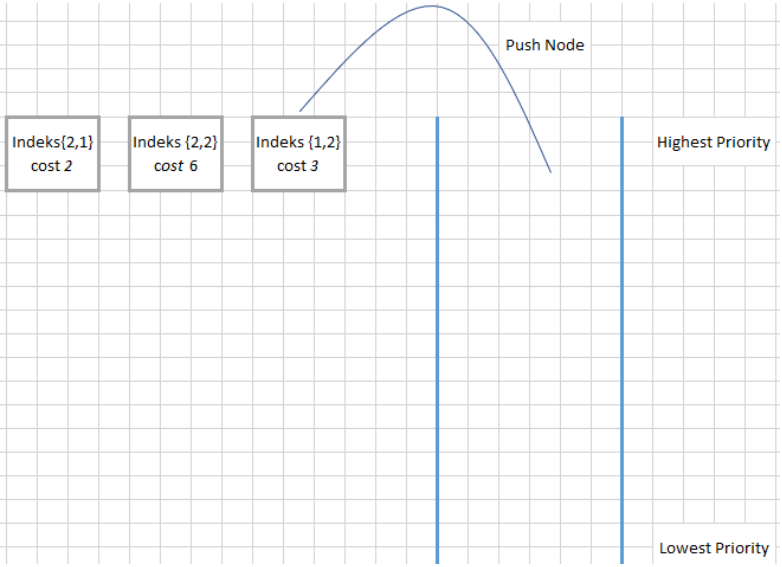
indeks yang belum ditelusuri indeks tersebut akan ditambahkan pada *queue* dengan prioritas yang ditetapkan berdasarkan besaran *cost value*.

3.2.2 Desain *Struct* Node

Tipe data yang masuk pada *Priority Queue* berupa *struct* yang mempunyai tiga variabel yaitu:

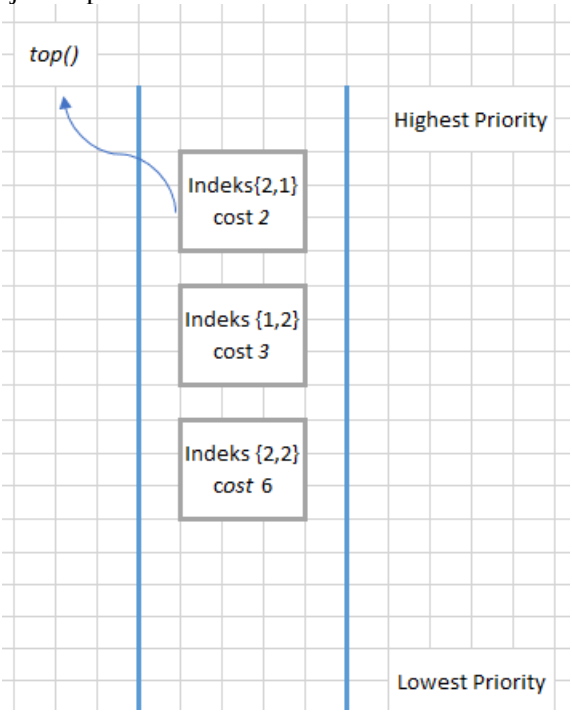
1. Posisi x pada baris
2. Posisi y pada kolom
3. *Cost value* dari indeks

Setiap *struct* Node masuk pada *queue* akan melakukan operasi *Boolean* untuk mendapatkan *cost value* paling kecil untuk prioritas paling tinggi. Sebagai contoh terdapat tiga Node yang akan masuk ke *queue* yang ditunjukan pada Gambar 3.10.



Gambar 3.10 Contoh input Node pada *Priority Queue*

Maka setelah melakukan operator *Boolean* intuk *set* prioritas *queue* akan menjadi seperti Gambar 3.11.



Gambar 3.11 Contoh *Priority Queue*

Pseudocode dari *struct* Node ditunjukan pada Gambar 3.12.

```
struct Node
1. int r, c, v
2. bool Operator <(const Node &priority) const
3.         return v > priority.v
```

Gambar 3.12 *Pseudocode Struct* Node

[*Halaman ini sengaja dikosongkan*]

BAB IV
IMPLEMENTASI

Pada bab ini akan dijelaskan implementasi dari algoritma dan struktur data berdasarkan desain yang telah dilakukan.

4.1 Lingkungan Implementasi

Lingkungan implementasi menggunakan sebuah komputer dengan spesifikasi perangkat lunak dan perangkat keras seperti tercantum pada Tabel 4.1.

No.	Jenis Perangkat	Spesifikasi
1	Perangkat Keras	• *Processor* Intel Core i5-5200U CPU @ 2.20GHz • *Memory* 8GB 1333MHz DDR3
2	Perangkat Lunak	• Sistem operasi Windows 10 Pro 64-bit • *Integrated Development Environment* Dev-C++ 5.3.0.3

Tabel 4.1 Lingkungan Implementasi

4.2 Rancangan Data

Pada subbab ini dijelaskan mengenai desain data masukan yang diperlukan untuk melakukan proses algoritma, dan data keluaran yang dihasilkan oleh program.

4.2.1 Data Masukan

Data masukan adalah data yang akan diproses oleh program sebagai masukan menggunakan algoritma dan struktur data yang telah dirancang dalam Tugas Akhir ini. Data masukan dari persoalan ini:

1. Baris pertama merupakan inputan jumlah R, C, dan N yang merepresentasikan jumlah *row*, jumlah *column* dan jumlah provinsi.
   ($1 \leq R, C \leq 500$ and $2 \leq N \leq 5$)

2. 3 × R baris selanjutnya merupakan representasi dari *value cost*, R_{ij} dan C_{ij} dengan pembagian 3 group yang masing – masing representasi mempunyai R baris.
   ($1 \leq \mathbf{V_{ij}} \leq 1000$, $0 \leq \mathbf{R_{ij}} \leq \mathbf{R}$and $0 \leq \mathbf{C_{ij}} \leq \mathbf{C}$, for **i** = 1, 2, . . . , **R** and **j** = 1, 2, . . . , **C**)
3. N baris selanjutnya merupakan inputan indeks posisi provinsi berupa X_i dan Y_i .
   ($1 \leq X_i \leq R$ and $1 \leq Y_i \leq C$ for i = 1, 2, . . . , N)

Berikut merupakan contoh dari masukan yang diperlukan pada persoalan ini:

```
3 4 5
1 2 1 1
1 5 3 4
1 1 6 3
1 2 3 3
3 3 1 2
0 0 0 1
1 4 0 1
2 3 0 1
4 1 3 1
1 1
3 4
1 1
2 2
2 2
```

R, C, dan N

Cost sesuai dengan indeks provinsi

R_{ij} yang merupakan jumlah baris yang bisa dituju pada provinsi dengan indeks (i, j)

C_{ij} yang merupakan jumlah kolom yang bisa dituju pada provinsi dengan indeks (i, j)

Provinsi dengan indeks (i, j)

4.2.2 Data Keluaran

Data keluaran yang dihasilkan oleh program berupa *cost value* minimal yang dihasilkan oleh penelusuran dari provinsi N sampai provinsi N+1 yang setiap *case* di pisahkan dengan *white space* dan *enter* jika sudah mencapai *case* terakhir. Jika tidak ada kemungkinan untuk melakukan penelusuran dari provinsi N ke N+1 maka data keluaran adalah -1 [1].

4.3 Implementasi Algoritma

Pada sub bab ini akan dijelaskan tentang implementasi proses algoritma secara keseluruhan berdasarkan desain yang telah dijelaskan pada Bab III.

4.3.1 *Header* yang diperlukan

Implementasi algoritma dengan pemanfaatan struktur data *Binary Indexed Tree 2D* membutuhkan empat buah *header* yaitu stdio.h, stdlib.h, string.h dan queue, seperti yang ditunjukan oleh Kode Sumber 4.1.

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <queue>
```

Kode Sumber 4.1 *Header* yang diperlukan

Header stdio.h berisi modul untuk menerima masukan dan memberikan keluaran. *Header* stdlib.h berisi modul untuk manajemen memori dinamis, generasi bilangan acak, pemilahan dan konversi. *Header* string.h berisi modul yang memiliki fungsi-fungsi untuk melakukan pemrosesan string serta *memori set* pada *array*. *Header* queue berisi modul untuk pembuatan *Priority Queue*.

4.3.2 Variabel Global

Variabel dengan *scope* global dibutuhkan untuk kemudahan pengaksesan sebuah variabel oleh setiap fungsi yang ingin mengaksesnya. *Array* 2D V, R dan C masing – masing merepresentasikan *cost* untuk melakukan penelusuran pada indeks (x, y), maksimal baris penelusuran dari indeks (x, y) dan maksimal

kolom penelusuran dari indeks (x, y). Kode sumber penggunaan variabel global ditunjukan pada Kode Sumber 4.2.

```
1. using namespace std;
2. #define MAXN 512
3. int V[MAXN][MAXN], R[MAXN][MAXN],
   C[MAXN][MAXN];
```

Kode Sumber 4.2 Variabel Global

4.3.3 Implementasi Fungsi Main

Tugas dari fungsi Main adalah menangani masukan dengan memanggil fungsi Scanint, memanggil fungsi Update dan menangani data keluaran sesuai yang dijelaskan pada sub bab 3.1.1. Kode sumber fungsi Main ditunjukan pada Kode Sumber 4.3.

```
1.  int Main() {
2.      int n, m, c, X[6], Y[6];
3.      scanf("%d%d%d", &n, &m, &c);
4.
5.      for(int i=1; i<=n; i++)
6.          for(int j=1; j<=m; j++)
7.              scanf("%d", &V[i][j]);
8.
9.      for(int i=1; i<=n; i++)
10.         for(int j=1; j<=m; j++)
11.             scanf("%d", &R[i][j]);
12.
13.     for(int i=1; i<=n; i++)
14.         for(int j=1; j<=m; j++)
15.             scanf("%d", &C[i][j]);
16.
17.     for(int i=0; i<c; i++){
18.         scanf("%d%d", &X[i], &Y[i]);
19.     }
20.
21.     for (int i = 1; i < c; i++) {
22.         int r = findPath(n, m, X[i-1], Y[i-1], X[i], Y[i]);
23.         printf("%d%c", r, i == c - 1 ? '\n' : ' ');
24.     }
25.     return 0;
26. }
```

Kode Sumber 4.3 Fungsi Main

4.3.4 Implementasi Fungsi Scanint

Fungsi Scanint merupakan fungsi yang dipanggil main untuk membaca masukan yang ada dengan memanggil *getchar*. *Getchar* lebih cepat dari *scanf* karena merupakan POSIX *function*

[4]. Kode sumber fungsi Scantint ditunjukan pada Kode Sumber 4.4.

```
1.  void Scanint(int &x){
2.      int c = getchar_unlocked();
3.      x=0;
4.      for (; (c<48 || c>57); c=getchar_unlocked());
5.      for (; c>47 && c<58 ; c=getchar_unlocked())
6.          x=(x<<1) + (x<<3) +c-48;
7.  }
```

Kode Sumber 4.4 Fungsi Scanint

4.3.5 Implementasi *Struct* Node

Sesuai yang dijelaskan pada 3.2.2 *struct* Node mempunyai operasi prioritas yang digunakan untuk *Priority Queue*. Kode sumber *struct* Node ditunjukan pada Kode Sumber 4.5.

```
1.  struct Node {
2.      int r, c, v;
3.      Node(int a=0, int b=0, int d=0, int e=0):
4.      r(a), c(b), v(d) {}
5.      bool operator<(const Node &x) const {
6.          return v > x.v;
7.      }
8.  };
9.  priority_queue<Node> pQ;
```

Kode Sumber 4.5 *Struct* Node

4.3.6 Implementasi Fungsi Findpath

Fungsi Findpath digunakan untuk memastikan terhadap *Priority Queue* apakah indeks dalam *queue* tersebut bisa mencapai indeks tujuan atau tidak, jika tidak maka fungsi Findpath akan memanggil fungi Update untuk penelusuran indeks sesuai yang dijelaskan pada sub bab 3.1.3.

Langkah pertama dalam fungsi Findpath adalah *statemen* jika indeks asal dan tujuan sama maka fungsi akan *return 0*. Untuk langkah selanjutnya fungsi Findpath akan memanggil fungsi Init dengan parameter jumlah R dan C untuk inisialisasi awal pembuatan *Binary Indexed Tree 2D* yang merepresentasikan bahwa semua indeks belum ditelusuri. Representasi bahwa indeks belum ditelusuri adalah angka 1 dan 0 jika penelusuran sudah dilakukan pada indeks tersebut. Setelah melakukan pemanggilan fungsi Init, fungsi Findpath akan memanggil fungsi Modify yang bertujuan bahwa ada satu indeks yang sudah dilakukan penelusuran, yaitu indeks asal karena semua penelusuran dimulai dari indeks asal.

Kode sumber dari fungsi Findpath ditunjukan pada Kode Sumber 4.6.

```
1.  int Findpath(int n, int m, int sx, int sy, int ex, int ey) {
2.     if (sx == ex && sy == ey)     return 0; //if from and to
    same return 0
3.     rangeTree.init(n, m);
4.     rangeTree.modify(sx, sy, -1);
5.
6.     while (!pQ.empty()){
7.         pQ.pop();
8.     }
9.     pQ.push(Node(sx, sy, V[sx][sy]));
10.
11.    Node u;
12.    int lr, rr, lc, rc;
13.    while (!pQ.empty()) {
14.       u = pQ.top(), pQ.pop();
15.
16.       if (abs(u.r - ex) <= R[u.r][u.c] && abs(u.c - ey) <=
    C[u.r][u.c])
17.           return u.v;
18.
19.       lr = max(1, u.r - R[u.r][u.c]), rr = min(n, u.r +
    R[u.r][u.c]);
20.       lc = max(1, u.c - C[u.r][u.c]), rc = min(m, u.c +
    C[u.r][u.c]);
21.       rangeTree.update(lr, rr, lc, rc, u.v, -1);
22.    }
23.    return -1;
24. }
```

Kode Sumber 4.6 Fungsi Findpath

4.3.7 Implementasi Fungsi Init

Pada saat pemanggilan fungsi Findpath, fungsi Init merupakan representasi pembentukan *Binary Indexed Tree 2D*

pertama kali untuk setiap satu kasus permasalahan. Sesuai yang dijelaskan pada sub bab 3.1.4 pembentukan indeks awal yang merepresentasikan bahwa semua indeks belum melakukan penelusuran dengan cara mengisialisasi semua indeks dengan angka satu dengan pemanggilan fungsi Modify. Pada saat inisialisasi, jumlah baris dan kolom akan dijadikan variabel global *struct* yang telah dibuat. Kode sumber fungsi Init ditunjukan pada Kode Sumber 4.7.

```
1.  void Init(int R, int C) {
2.       this->R = R, this->C = C;
3.       memset(A, 0, sizeof(A));
4.       for (int i = 1; i <= R; i++)
5.            for (int j = 1; j <= C; j++)
6.              modify(i, j, 1);
7.    }
```

Kode Sumber 4.7 Fungs Init

4.3.8 Implementasi Fungsi Modify

Fungsi Modify membawa parameter posisi indeks (x, y) dan nilai untuk perbahan *Binary Indexed Tree 2D* pada indeks tersebut.

Fungsi Modify akan melakukan perulangan pada baris dan kolom karena merupaka matriks merupakan *Binary Indexed Tree* 2D. Setiap penelusuran indeks melalui perulangan nilai indeks akan ditambahkan oleh parameter nilai yang masuk pada fungsi ini. Kode sumber fungsi Modify ditunjukan pada Kode Sumber 4.8.

```
1.  void Modify(int x, int y, int val) {
2.        for (; x <= R; x += x&(-x)) {
3.           for (int i = y; i <= C; i += i&(-i))
4.              A[x][i] += val;
5.        }
6.     }
```

Kode Sumber 4.8 Fungsi Modify

4.3.9 Implementasi Fungsi RectangleSum

Untuk menghitung *area* pada *Binary Indexed Tree 2D* digunakan fungsi RectangleSum yang akan memanggil fungsi Query. Tujuan dari fungsi ini untuk menentukan batas – batas *area* yang akan dikirimkan sebagai parameter untuk fungsi Query sesuai dengan penjelasan pada sub bab 3.1.6**.** Kode sumber fungsi RectangleSum ditunjukan pada Kode Sumber 4.9.

```
1.  int RectangleSum(int lx, int rx, int ly, int ry) {
2.        return query(rx, ry) - query(lx-1, ry) - query(rx, ly-
    1) + query(lx-1, ly-1);
3.  }
```

Kode Sumber 4.9 Fungsi RectangleSum

4.3.10 Implementasi Fungsi Query

Fungsi Query mencari *area* dari parameter indeks baris dan kolom yang dikirim oleh fungsi RectangleSum. *Area* yang dihitung merupakan *area* indeks dari parameter sampe indeks (0,0) dengan perhitungan seperti yang dijelaskan pada sub bab 3.1.7 dan hasil *return* dari fungi Query merupakan jumlah *unvisited area*. Kode sumber fungsi Query ditunjukan pada Kode Sumber 4.10.

```
1.  int query(int x, int y) {
2.          int ret = 0;
3.          for (; x > 0; x -= x&-x)
4.              for (int i = y; i > 0; i -= i&(-i))
5.                  ret += A[x][i];
6.          return ret;
7.      }
```

Kode Sumber 4.10 Fungsi Query

4.3.11 Implementasi Fungsi Update

Fungsi Update merupakan fungsi untuk penelusuran *Dijkstra* yang menelusuri matriks dua dimensi dengan optimasi penelusuran dibantu oleh *Binary Indexed Tree 2D* seperti yang dijelaskan pada sub bab 3.1.8. Kode sumber fungsi Update ditunjukan pada Kode Sumber 4.11.

```
1.  void update(int lx, int rx, int ly, int ry, int val, int tot) {
2.      if (tot == -1)
3.          tot = rectangleSum(lx, rx, ly, ry);
4.      if (tot == 0)    return;
5.      if (lx == rx) {
6.          if (ly == ry) {
7.              pQ.push(Node(lx, ly, val + V[lx][ly], abs(lx-TX)
    + abs(ly-TY)));
8.              modify(lx, ly, -1);
9.              return;
10.         }
11.
12.         int cnt = rectangleSum(lx, rx, ly, (ly + ry)/2);
13.         if (cnt)
14.             update(lx, rx, ly, (ly + ry)/2, val, cnt);
15.         if (cnt < tot)
16.             update(lx, rx, (ly + ry)/2 + 1, ry, val, tot - cnt);
17.     }
18.     else {
19.         int cnt = rectangleSum(lx, (lx + rx)/2, ly, ry);
20.         if (cnt)
21.             update(lx, (lx + rx)/2, ly, ry, val, cnt);
22.         if (cnt < tot)
23.             update((lx + rx)/2 + 1, rx, ly, ry, val, tot - cnt);
24.     }
25. }
```

Kode Sumber 4.11 Fungsi Update

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan tentang uji coba dan evaluasi dari implementasi sistem yang telah dilakukan pada bab 4.

5.1 LINGKUNGAN UJI COBA

Lingkungan uji coba menggunakan sebuah komputer dengan spesifikasi perangkat lunak dan perangkat keras seperti tercantum pada Tabel 5.1.

No.	Jenis Perangkat	Spesifikasi
1	Perangkat Keras	• *Processor* Intel Core i5-2670QM CPU @ 2.20GHz • *Memory* 8GB 1333MHz DDR3
2	Perangkat Lunak	• Sistem operasi Windows 10 Pro • *Integrated Development Environment* Dev-C++ 5.3.0.3

Tabel 5.1 Spesifikasi Lingkungan Uji Coba

5.2 SKENARIO UJI COBA

Pada subbab ini akan dijelaskan skenario uji coba yang dilakukan. Skenario uji coba terdiri dari uji coba kebenaran dan uji coba kinerja. Uji coba kebenaran akan menggunakan kasus sama dengan yang ada pada contoh kasus pada permasalahan *Journey Through the Kingdom* ditunjukan pada Gambar 5.1.

```
3 4 5
1 2 1 1
1 5 3 4
1 1 6 3
1 2 3 3
3 3 1 2
0 0 0 1
1 4 0 1
2 3 0 1
4 1 3 1
1 1
3 4
1 1
2 2
2 2
```

Gambar 5.1 Contoh Kasus Uji Coba

Gambar 5.2 Contoh Kasus Uji Coba

5.2.1 Inisialisasi Contoh Kasus Permasalahan

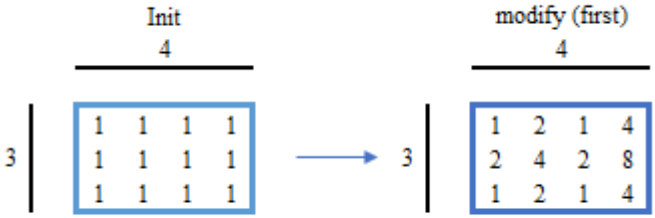
Setelah melakukan data masukan dari fungsi Main, matriks *cost*, matriks R_{ij} dan matriks C_{ij} akan di*set* pada variabel global untuk memudahkan akses pembacaan variabel. Kemudian fungsi Main akan memanggil fungsi Findpath dengan parameter indeks asal, indeks tujuan serta jumlah baris dan kolom.

Setelah masuk ke fungsi Findpath yang memilik parameter n = 3, m = 4, sx = 1, sy = 1, ex = 3 dan ey = 4 yang merepresentasikan:

- n = jumlah baris
- m = jumlah kolom
- sx = indeks baris asal

- sy = indeks kolom asal
- ex = indeks baris tujuan
- ey indeks kolom tujuan

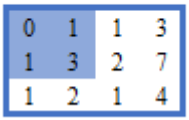
akan dilakukan operasi untuk pemeriksaan apakah parameter indeks asal dan tujuan sama, jika sama maka akan melakukan *return 0*. Pada kasus ini indeks asal dan tujuan tidak sama, maka akan lanjut ke operasi selanjutnya. Selanjutnya akan dipanggil fungsi Init pada *struct* RangeTree dengan parameter jumlah indeks dan kolom untuk inisialisasi awal bentuk *Binary Indexed Tree 2D* yang akan dilanjutkan pemanggilan fungsi Modify. Hasil awal Inisialisasi ditunjukan pada Gambar 5.2.



Gambar 5.3 Pembentukan *Binary Indexed Tree 2D* awal

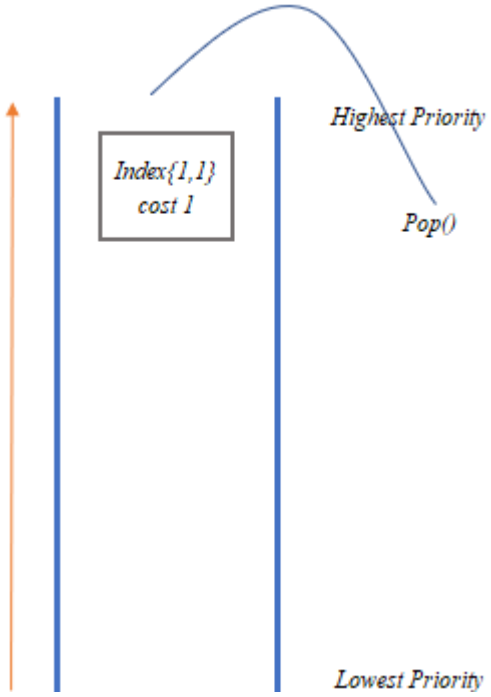
Setelah melakukan pemanggilan fungsi Init yang merepresentasikan semua indeks pada *Binary Indexed Tree 2D* belum dilakukan penelusuran, akan dipanggil fungsi Modify dengan parameter indeks baris asal = 1 dan indeks kolom asal = 1 pada untuk representasi bahwa indeks asal sudah melakukan penelusuran. Setiap dilakukan pemanggilan fungsi Modify, semua indeks yang berkaitan dengan indeks yang diubah akan mengalami perubahan juga seperti yang ditunjukan pada Gambar 5.3.

Modify(root indeks)



Gambar 5.4 Pengubahan indeks asal pada *Binary Indexed Tree 2D*

Indeks tersebut akan dilakukan *push* pada *Priority Queue* dengan parameter indeks asal dan *cost* dari indeks tersebut. Sebelum dilakukan *push*, akan dilakukan *pop* terhadap *Priority Queue* sampai *Empty*. Hasil *push* pada *Priority Queue* ditunjukan pada Gambar 5.4.



Gambar 5.5 *Push* indeks asal pada *Priority Queue*

5.2.2 Pengecekan penelusuran indeks Contoh Kasus

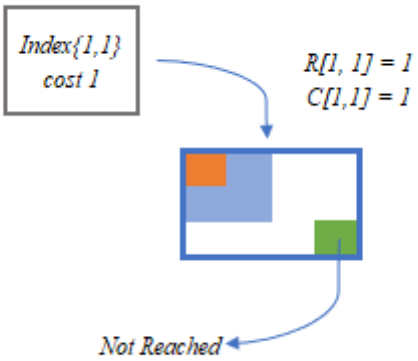
Pada sub bab ini akan ditunjuk elemen yang mempunyai *priority* paling tinggi dengan pemanggilan *top* dan akan dilakukan *pop* untuk elemen tersebut. Setelah elemen dengan prioritas paling tinggi ditunjuk, maka akan dilakukan pengecekan apakah indeks tersebut bisa melakukan penelusuran sampai indeks tujuan dengan *cost* elemen tersebut dengan ketentuan:

$$absolute(u.r - ex) \leq Rij \text{ dan } absolute(u.c - ey \leq Cij)$$

Pada kasus ini, indeks asal dengan R_{ij} dan C_{ij} yang ditentukan tidak dapat melakukan penelusuran sampai indeks tujuan, maka akan dipanggil fungsi Update untuk melakukan

penelusuran indeks lain. Ilustrasi pengecekan penelusuran ditunjukan pada Gambar 5.5 dengan informasi warna ditunjukan pada Gambar 5.6.

Gambar 5.6 Informasi warna



Gambar 5.7 Pengecekan penelusuran terhadap indeks asal *Binary Indexed Tree 2D*

5.2.3 Penerapan *Dijkstra* dengan *Binary Indexed Tree 2D* pada Contoh

Fungsi Update dipanggil dari fungsi Findpath dengan parameter lx = 1, rx = 2, ly = 1, ry = 2, val = 1 dan tot = -1 yang merepresentasikan:

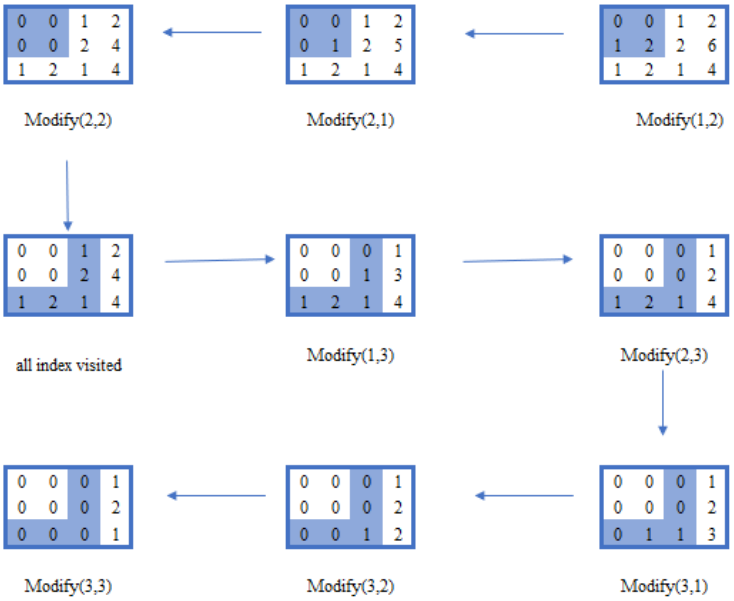
- lx = indeks baris paling kecil
- rx = indeks baris paling besar
- ly = indeks kolom paling kecil
- ry = indeks kolom paling besar
- val = *cost* yang dikenakan pada indeks
- tot -1 merupakan representasi bahwa indeks belum melakukan penelusuran.

Kemudian akan dilakukan pengecekan parameter tot, jika parameter tot bernilai 0 maka semua indeks sudah ditelusuri atau indeks sudah tidak bisa melakukan penelusuran kembali dan akan melakukan *return*, jika tot bernilai -1 menunjukan bahwa penelusuran tersebut masih kasus yang pertama dan akan dihitung *univited point area* dengan pemanggilan fungsi RectangleSum. Pada langkah pertama ini *unvisited point area* yang terhitung berjumlah tiga yang menelusuri indeks (1,1) sampai indeks (2,2).

Fungsi Update akan melakukan strategi rekursif pemanggilan diri sendiri untuk melakukan penelusuran. Penelusuran pertama akan dilakukan pengecekan apakah lx = rx, jika tidak maka dilakukan pengubahan nilai rx menjadi (lx + rx)/2 yang berarti indeks baris paling besar akan beralih ke baris yang lebih kecil dan akan melakukan perhitungan *unvisited point area* kembali untuk pemanggilan fungsi Update dengan tot dan posisi yang berbeda. Jika hasil dari perhitungan *unvisited point area* < tot, maka lx akan diubah menjadi (lx + rx)/2 + 1 dan akan melakukan rekursih pemanggilan fungsi Update dengan tot dan posisi yang berbeda.

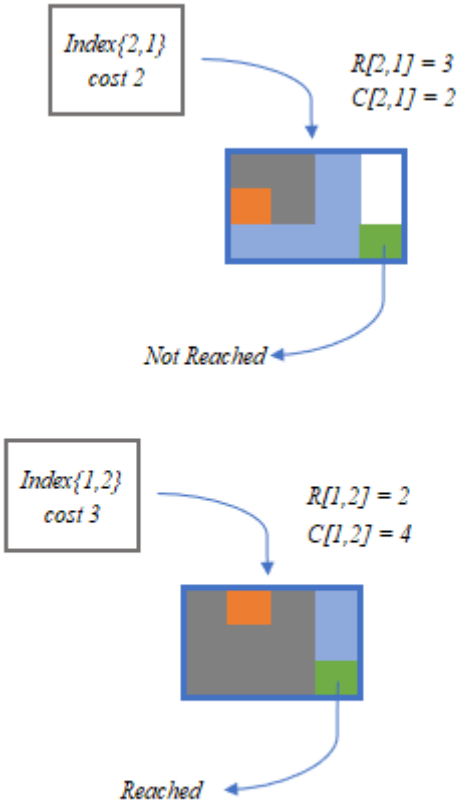
Jika lx = rx maka akan dilakukan pengecekan kembali apakah ly = ry, jika tidak maka dilakukan pengubahan sama seperti rx dan lx yang dijelaskan pada paragraf sebelumnya. Setelah lx =

rx dan ly = ry maka akan diubah indeks pada *Binary Indexed Tree 2D* bahwa indeks tersebut sudah melakukan penelusuran dengan cara pemanggilan fungsi Modify dan akan dilakukan *push* pada *Priority Queue* dengan posisi indeks dan *cost value* yang ada. Setelah semua indeks sudah dilakukan penelusuran, maka akan keluar dari fungsi Update dan akan kembali lagi melakukan *pop* dan kembali menuju bab 5.2.2. Untuk ilustrasi pengubahan *Binary Indexed Tree 2D* pada contoh kasus permasalahan ditunjukan pada Gambar 5.7:



Gambar 5.8 Penelusuran dan perubahan indeks *Binary Indexed Tree 2D* pada kasus pertama

Untuk ilustrasi pengecekan penelusuran indeks seperti sub bab 5.2.2 pada contoh kasus permasalahan ditunjukan pada Gambar 5.8.

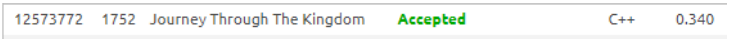


Gambar 5.9 Pengecekan penelusuran terhadap lain pada *Binary Indexed Tree 2D*

Pada Contoh kasus pertama permasalahan *Journey Through the Kingdom* akan dihasilkan keluaran sama dengan contoh yang ada pada keluaran persoalan, yaitu angka 3. Untuk kasus lain akan dilakukan cara yang sama seperti kasus pertama.

5.3 SKENARIO UJI COBA

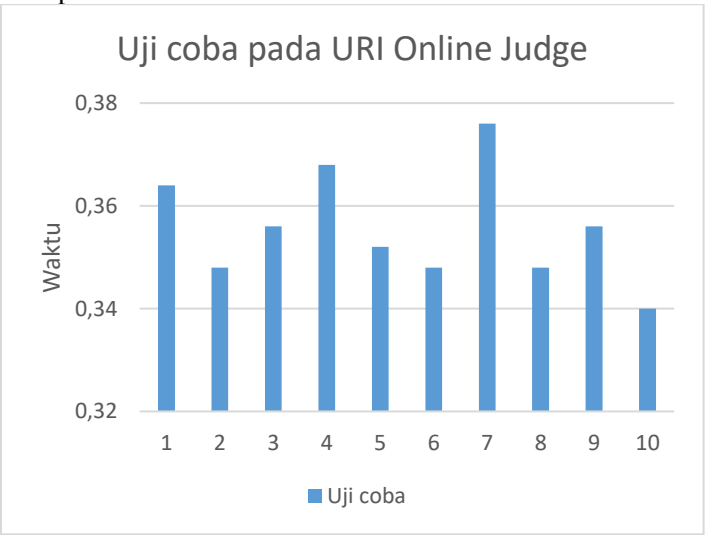
Dari hasil analisis sub bab **5.2** pada contoh permasalahan menghasilkan keluaran yang sesuai dengan keluaran pada contoh kasus. Setelah itu dilakukan uji kebenaran dengan mengumpulkan berkas kode sumber ke situs *URI Online Judge* ditunjukan pada Gambar 5.9.



Gambar 5.10 Hasil uji coba kebenaran pada situs *URI Online Judge*

Hasil uji coba yang telah dilakukan program mendapat umpan balik *Accepted*. Waktu yang dibutuhkan program untuk menyelesaikan permasalahan tersebut adalah 0.34 detik.

Grafik hasil uji coba pengumpulan sebanyak 10 kali ditunjukan pada Gambar 5.10.



Gambar 5.11 Hasil uji coba kinerja

Dari hasil pengumpulan sebanyak 10 kali didapat waktu rata – rata program yaitu 0.3532 detik.

BAB VI
KESIMPULAN

Pada bab ini akan dijelaskan kesimpulan dari hasil uji coba yang telah dilakukan.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap implementasi solusi untuk permasalahan *Journery Through the Kingdom* dalam mencari jalur dengan *cost* terendah dari ketentuan yang sudah diberikan, dapat diambil kesimpulan:

1. Permasalahan mencari jalur dengan *cost* terkecil dari matriks yang telah diberikan telah berhasil diselesaikan dengan algoritma *Dijkstra* dan struktur *Binary Indexed Tree 2D*.
2. Waktu rata-rata yang diperlukan untuk menyelesaikan permasalahan *Journey Through the Kingdom* dari 10 kali uji coba pada dari *Uri Online Judge* adalah 0.3532 detik
3. Rata-rata waktu yang diperlukan untuk menyelesaikan kasus uji pada dari *URI Online Judge* merupakan yang terbaik dibandingkan dengan implementasi lain.

6.2 Saran

Saran yang diberikan untuk implementasi Tugas Akhir ini adalah:

1. Penggunaan penelusuran *Dijkstra* dengan struktur data *Binary Indexed Tree 2D* pada Tugas Akhir ini, struktur data yang digunakan adalah struktur data *searching* sehingga setiap operasi memerlukan waktu logaritmik. Struktur data *Binary Indexed Tree 2D* dapat dipertimbangkan karena pencarian yang digunakan mempunyai kompleksitas yang lebih baik sehingga operasi penelusuran akan menjadi lebih cepat.

[*Halaman ini sengaja dikosongkan*]

DAFTAR PUSTAKA

[1] F. I. S. Massolo, “Journey Through The Kingdom,” [Online]. Available: https://www.urionlinejudge.com.br/judge/en/problems/view/1752.

[2] C. E. Leiserson, C. Stein, R. Rivest and T. H. Cormen, Introduction to Algorithms, 3rd Edition, London: MIT Press, 1989.

[3] R. Belwariar, “https://www.geeksforgeeks.org,” Geeks for Geeks, [Online]. Available: https://www.geeksforgeeks.org/two-dimensional-binary-indexed-tree-or-fenwick-tree/.

[4] D. G. Sunita, “Journal of King Saud University - Computer and Information Sciences,” in *Computer and Information Sciences*, Greater Noida, India, Elsevier, 2018.

[5] B. Gallmeister, “POSIX.4 Programmers Guide,” in *Programming for the Real World*, O'Reilly Media, 1995, p. 570.

[*Halaman ini sengaja dikosongkan*]

LAMPIRAN A
HASIL UJI COBA

Berikut merupakan lampiran hasil uji coba pengumpulan berkas kode solusi sebanyak 10 kali:

#		PROBLEM	ANSWER	LANGUAGE	TIME
12573772	1752	Journey Through The Kingdom	**Accepted**	C++	0.340
12573769	1752	Journey Through The Kingdom	**Accepted**	C++	0.356
12573767	1752	Journey Through The Kingdom	**Accepted**	C++	0.348
12573766	1752	Journey Through The Kingdom	**Accepted**	C++	0.376
12573765	1752	Journey Through The Kingdom	**Accepted**	C++	0.348
12573764	1752	Journey Through The Kingdom	**Accepted**	C++	0.352
12573763	1752	Journey Through The Kingdom	**Accepted**	C++	0.368
12573760	1752	Journey Through The Kingdom	**Accepted**	C++	0.356
12573758	1752	Journey Through The Kingdom	**Accepted**	C++	0.348
12573757	1752	Journey Through The Kingdom	**Accepted**	C++	0.364

Gambar 6.1 Hasil Pengumpulan Kode Sumber Sebanyak 10 Kali

[*Halaman ini sengaja dikosongkan*]

BIODATA PENULIS

Brama Diwangkara, lahir di Semarang tanggal 09 Februari 1997. Penulis merupakan anak keempat dari 4 bersaudara. Penulis telah menempuh pendidikan formal TK Bustanul Alfa Kabupaten Semarang, SD Negeri 01 Kabupaten Semarang (2003-2009), SMP Negeri 01 Salatiga (2009-2012) , SMA Negeri 1 Salatiga (2012-2015). Penulis melanjutkan studi kuliah program sarjana di Departemen Teknik Informatika ITS.

Selama kuliah di Teknik Informatika ITS, penulis mengambil bidang minat Algoritma Pemrograman (AP). Penulis pernah menjadi asisten dosen dan praktikum untuk mata kuliah Sistem Operasi (2017) dan Sistem Basis Data (2017). Selama menempuh perkuliahan, penulis juga aktif mengikuti kompetisi *Business Case Information Technology*. Selain itu, penulis juga pernah menjadi pengembang sistem Gemastik (paGelaran Mahasiswa Nasional TIK) pada tahun 2018, serta menjadi staff ahli REEVA pada tahun 2017. Penulis dapat dihubungi melalui surel di `diwangkarabrama@gmail.com`.