

MODUL 9 TUGAS BESAR

Frances Louisa Constantine (13222094)

Nusaiba El Qonitat (13222095)

Farhan Revandi Suhirman (13222096)

Pierre Gavin Tan (13222097)

Kennard Benaya Manli (13222098)

Roger Supriyanto (13222099)

Rubi Naufal Tiandito (13222100)

Asisten: Agape D'Sky (23222031)

Kelompok: E1

EL2208-Praktikum Pemecahan Masalah dengan C

Laboratorium Dasar Teknik Elektro - Sekolah Teknik Elektro dan Informatika ITB

Abstrak

Pada tugas besar ini, sebuah permasalahan berupa pencarian jalur pada sebuah labirin atau maze (lebih dikenal dengan nama maze problem) dipilih untuk diselesaikan dengan beberapa metode traversal. Metode penyelesaian yang digunakan diimplementasikan dengan beberapa jenis algoritma, yaitu BFS, DFS, Dijkstra, Greedy, Dynamic Programming, A (A-star), dan Backtracking. Dengan algoritma-algoritma berikut, jalur-jalur yang didapatkan akan dibandingkan dengan menganalisis kompleksitas waktunya yang berhubungan langsung dengan efisiensi algoritma dalam memecahkan permasalahan ini. Dari hasil analisis, algoritma dengan efisiensi dan efektivitas tertinggi akan digunakan sebagai penyelesaian dari permasalahan ini. Hasil yang didapatkan sudah sesuai dengan output yang diharapkan, meskipun terdapat beberapa kendala seperti tidak bisa menampilkan semua kemungkinan jalur pada beberapa algoritma karena tidak sesuai dengan fungsi algoritma tersebut.*

Kata kunci: maze problem, advanced algorithm, DFS, time complexity

1. PENDAHULUAN

Dalam dunia pemrograman, terdapat banyak algoritma yang dapat digunakan untuk menyelesaikan permasalahan yang ada. Contohnya seperti, algoritma brute force, greedy, depth first search, breadth first search, dan masih banyak lagi. Dalam tugas besar ini, dipilih permasalahan *maze problem* dimana pada sebuah labirin diperlukan untuk mencari semua kemungkinan jalur yang mungkin, jalur terpanjang dan terpendek antara titik mulai 'S' menuju titik akhir 'E' tanpa melewati blokade jalur yang dilambangkan dengan '#'. Untuk menyelesaikan permasalahan ini, digunakan beberapa algoritma sebagai bentuk eksplorasi masing-masing

praktikan dan sebagai bahan pembanding antar-algoritma. Algoritma-algoritma yang diterapkan untuk menyelesaikan permasalahan ini adalah algoritma dijkstra, breadth first search, depth first search, dynamic programming, algoritma a*, backtracking, dan greedy. Beberapa tahapan yang dicapai untuk memenuhi tugas ini adalah membuat batasan-batasan yang akan diterapkan pada permasalahan yang diambil, membagi algoritma yang digunakan pada anggota kelompok sehingga setiap orang akan menyelesaikan permasalahan dengan algoritma yang berbeda-beda, membuat kerangka berpikir untuk masing-masing algoritma dalam bentuk flowchart dan DFD yang kemudian diimplementasikan dalam bentuk *source code*. Setelah tahap implementasi setiap algoritma, hasil yang didapatkan akan dibandingkan dalam segi efisiensi (menurut kompleksitas waktu) dan dipilih satu sebagai solusi terbaik untuk menyelesaikan permasalahan *maze problem*.

Adapun tujuan dari tugas besar ini adalah

1. Membandingkan kecepatan runtime antar algoritma yang digunakan.
2. Mengetahui algoritma terbaik untuk menyelesaikan *maze problem*.

2. STUDI PUSTAKA

2.1 ALGORITMA DIJKSTRA

Algoritma Dijkstra merupakan algoritma yang biasanya digunakan untuk menyelesaikan pencarian jalur terpendek dengan nilai edge yang tidak negatif. Dalam algoritma ini, terdapat suatu set yang berisi vertices yang sudah dikunjungi dan set yang berisi

vertices yang belum dikunjungi. Pencarian dimulai dari vertex sumber kemudian secara iterative memilih vertex yang belum dikunjungi dengan nilai jarak yang paling kecil dari sumber. Setelahnya, tetangga dari vertex tersebut akan dikunjungi dan memperbarui jaraknya jika jalur yang lebih pendek ditemukan. Proses ini terus berlanjut sampai vertex akhir ditemukan atau semua vertices sudah dikunjungi [1].

2.2 BREADTH FIRST SEARCH (BFS)

BFS merupakan algoritma yang melakukan pencarian dengan cara menjelajahi seluruh node atau vertex pada "level" yang sama kemudian melanjutkan pencarian ke "level" berikutnya. Algoritma ini dimulai pada suatu vertex yang spesifik dan mengunjungi semua tetangganya sebelum lanjut ke tetangga pada level berikutnya. BFS merupakan algoritma yang biasanya digunakan untuk menemukan jalur, komponen yang terkoneksi, dan pencarian jalur terpendek dalam graf [2].

2.3 DEPTH FIRST SEARCH (DFS)

DFS merupakan algoritma yang melakukan pencarian dengan cara masuk ke dalam sebuah cabang hingga ke node akhir kemudian berpindah ke cabang lainnya. Algoritma ini dimulai pada node root (tergantung pada graf yang digunakan) kemudian mengeksplorasi sejauh mungkin pada setiap cabang sebelum dilakukan backtracking [3].

2.4 DYNAMIC PROGRAMMING

Dynamic Programming merupakan metode yang digunakan dalam ilmu matematika dan computer science untuk menyelesaikan masalah kompleks dengan memecah-mecahnya menjadi submasalah yang lebih sederhana. Dengan hanya menyelesaikan setiap submasalah sekali dan menyimpan hasilnya, komputasi

berulang dan solusi yang lebih efisien dapat dicapai [4].

Algoritma ini dimulai dengan mengidentifikasi submasalah yang ada (membagi masalah utama menjadi submasalah yang lebih kecil dan independent), menyimpan solusi (menyelesaikan setiap submasalah dan menyimpan solusinya ke dalam tabel atau array), membangun solusi (solusi yang sudah disimpan digunakan untuk membangun solusi yang dapat menyelesaikan masalah utama), menghindari perulangan (dengan menyimpan solusi, algoritma ini dapat memastikan bahwa setiap submasalah hanya diselesaikan sekali sehingga waktu komputasinya dapat berkurang) [4].

2.5 ALGORITMA A*

Algoritma A* merupakan teknik terbaik dan paling populer yang digunakan untuk pencarian jalur dan traversal graf. Jika diberikan suatu matrix yang memiliki rintangan, koordinat awal, dan koordinat akhir, pada setiap langkah, algoritma a* akan memilih node dengan value f yang mana adalah sebuah parameter yang sama dengan penjumlahan 2 variabel lain (g dan h). Di setiap langkahnya, algoritma ini memilih node yang memiliki nilai f paling rendah kemudian memproses node tersebut. Variabel g merupakan pergerakan yang digunakan untuk bergerak dari koordinat awal ke koordinat lain dalam matrix sedangkan variable h merupakan estimasi Gerakan yang digunakan untuk bergerak dari koordinat lain dalam matrix ke koordinat tujuan akhir [5].

2.6 ALGORITMA BACKTRACKING

Backtracking merupakan algoritma yang mencoba mencari setiap jalur yang ada. Ketika ditemui dead end saat traversal, akan dilakukan backtrack ke node terakhir dan dicoba rute lain. Dengan kata lain, backtracking merupakan algoritma penyelesaian masalah yang melibatkan pencarian

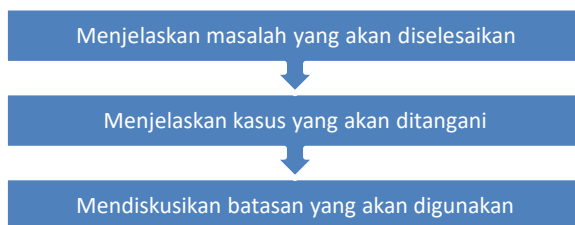
solusi secara bertahap dengan mencoba opsi lain dan membatalkannya bila opsi tersebut mengarah ke dead end. Algoritma ini biasanya digunakan pada situasi dimana dibutuhkan eksplorasi terhadap semua kemungkinan yang ada untuk menyelesaikan suatu masalah, seperti mencari jalur dalam labirin atau menyelesaikan puzzle seperti sudoku [6].

2.7 ALGORITMA GREEDY

Algoritma greedy merupakan algoritma yang membuat pilihan paling optimal secara local pada setiap langkah dengan harapan dapat menemukan solusi global paling optimum. Dalam algoritma ini, keputusan dibuat berdasarkan informasi yang tersedia saat itu tanpa memedulikan konsekuensi dari keputusan tersebut di masa depan. Kuncinya adalah memilih pilihan terbaik di setiap langkahnya agar mengarah pada solusi yang cukup baik meskipun bukan yang terbaik [7].

3. METODOLOGI

3.1 Mendefinisikan Ruang Lingkup Masalah



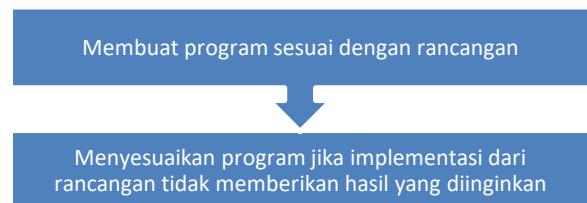
Gambar 3-1 Flowchart tahapan mendefinisikan ruang lingkup masalah

3.2 Merancang Software



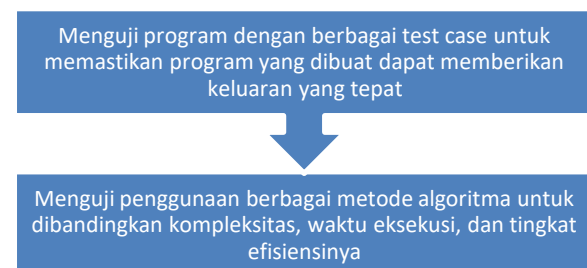
Gambar 3-2 Flowchart tahapan merancang software

3.3 Implementasi Rancangan



Gambar 3-3 Flowchart tahapan implementasi rancangan

3.4 Pengujian



Gambar 3-4 Flowchart tahapan pengujian

4. HASIL DAN ANALISIS

4.1 RUANG LINGKUP MASALAH

Masalah yang dipilih adalah nomor satu, yaitu *maze problem* dimana diharuskan mencari semua kemungkinan jalur, jalur terpanjang, dan jalur

terpendek dari titik awal yang disimbolkan dengan karakter 'S' ke titik akhir yang disimbolkan dengan karakter 'E'. Algoritma yang digunakan adalah, dijkstra, BFS, DFS, dynamic programming, A*, backtracking, dan greedy. Semua algoritma akan mencari jalur terpendek berdasarkan caranya masing-masing sedangkan untuk mencari semua jalur, akan digunakan algoritma DFS saja.

4.2 RANCANGAN

Rencana pembagian tugas dapat disajikan seperti pada tabel berikut.

Task	Pembagian Tugas
Pembacaan file (files.c, files.h)	Designer: - 13222097 Implementer - 13222097 Tester - 13222097
Algoritma Dijkstra	Designer: - 13222095 Implementer - 13222095 Tester - 13222095 - 13222097
Algoritma BFS	Designer: - 13222094 Implementer - 13222094 Tester - 13222094 - 13222097
Algoritma DFS	Designer: - 13222099 Implementer - 13222099 Tester - 13222099 - 13222097
Algoritma Dynamic Programming	Designer: - 13222097 Implementer - 13222097 Tester - 13222097
Algoritma A*	Designer:

	- 13222100 Implementer - 13222100 Tester - 13222100 - 13222097
Algoritma Backtracking	Designer: - 13222098 Implementer - 13222098 Tester - 13222098 - 13222097
Algoritma Greedy	Designer: - 13222096 Implementer - 13222096 Tester - 13222096 - 13222097

4.3 IMPLEMENTASI

4.3.1 Dijkstra

Algoritma Dijkstra menyelesaikan permasalahan *maze* dengan mencari jalur terdekat dari titik awal 'S' menuju titik akhir 'E'. Pencarian dimulai dari titik awal lalu jalur dicari ke 4 arah (atas, bawah, kanan, kiri). Titik yang dikunjungi akan dihitung jaraknya dengan titik sebelumnya, dan titik yang akan dipilih merupakan titik yang memiliki jarak minimum dihitung dari titik awal dan akan terus seperti itu hingga menemukan titik akhir. Dalam menyusun implementasi, dibuat 4 fungsi yaitu fungsi utama, fungsi pembaca file, fungsi dijkstra, dan fungsi pencetak jalur. Berikut penjelasan bagian-bagian kode.

a. Header Libraries

Header	Deskripsi
stdio.h	Library standar untuk input dan output di C. Digunakan untuk fungsi seperti printf, scanf, fopen, fgets, dan fclose.
stdlib.h	Library standar untuk berbagai utilitas umum, termasuk konversi angka, alokasi memori, proses, dan fungsi lingkungan.

limits.h	Library standar yang mendefinisikan berbagai konstanta yang mewakili batas dari tipe data integral.
time.h	Library standar untuk fungsi yang berhubungan dengan waktu dan tanggal.

b. Definisi value dan struct

Bagian	Deskripsi
Value	MAX_SIZE bernilai 100 merupakan konstanta preprosesor yang mendefinisikan ukuran maksimum grid maze. Sedangkan, INT_MAX bernilai infinit (tak hingga) merupakan konstanta preprosesor yang mendefinisikan nilai tak terhingga (infinity) sebagai nilai maksimum dari tipe integer (INT_MAX) yang menyatakan bahwa jarak awalnya tidak diketahui atau tidak dapat dicapai.
Struct Point	Struktur yang merepresentasikan koordinat titik dalam maze dengan dua anggota x dan y.
Struct Maze	Struktur yang merepresentasikan maze dengan berbagai atribut: <ul style="list-style-type: none"> - grid: Matriks karakter untuk menyimpan representasi maze. - dist: Matriks integer untuk menyimpan jarak terpendek dari titik awal. - visited: Matriks integer untuk menandai apakah suatu titik telah dikunjungi. - parent: Matriks Point untuk menyimpan titik asal dari jalur terpendek.

	<ul style="list-style-type: none"> - width dan height: Lebar dan tinggi maze. - start dan end: Titik awal dan akhir dalam maze.
--	---

c. Fungsi readMaze

Bagian	Deskripsi
Input	Input berupa nama file yang berisi struktur maze dan input pointer ke struktur Maze yang akan diisi dengan data dari file.
Proses	Dimulai dengan membuka file dengan nama yang diberikan menggunakan fopen. Lalu, membaca file baris per baris menggunakan fgets. Membaca karakter demi karakter dalam setiap baris dan mengisi grid dengan karakter tersebut. Mengidentifikasi dan menyimpan koordinat titik awal ('S') dan titik akhir ('E'). Kemudian, menghitung lebar (width) dan tinggi (height) maze berdasarkan jumlah kolom dan baris yang dibaca, jika kolom pada baris saat ini berbeda dengan lebar pada baris pertama maka file maze tidak valid.
Output	Struktur Maze yang diisi dengan data dari file: grid, start, end, width, dan height. Jika file tidak dapat dibuka, program akan menampilkan pesan kesalahan dan keluar.

d. Fungsi djikstra

Bagian	Deskripsi
Input	Pointer ke struktur Maze yang berisi data maze dan akan digunakan untuk menjalankan algoritma Dijkstra.
Proses	Dimulai dengan menginisialisasi array dist dengan nilai INF, visited dengan 0, dan parent dengan (-1, -1). Lalu, jarak diatur di titik awal (start) ke 0. Kemudian iterasi dilakukan sebanyak jumlah titik dalam maze (height * width). Dalam

	iterasi dilakukan proses mencari titik dengan jarak minimum yang belum dikunjungi, menandai titik tersebut sebagai titik yang telah dikunjungi, dan memperbarui jarak untuk setiap tetangga yang valid (tidak di luar batas dan bukan dinding).
Output	Struktur Maze yang diperbarui dengan jarak terpendek (dist) dari titik awal ke semua titik lain dan parent dari setiap titik dalam jalur terpendek.

e. Fungsi printShortPath

Bagian	Deskripsi
Input	Pointer ke struktur Maze yang berisi hasil dari algoritma Dijkstra, termasuk jarak terpendek dan parent dari setiap titik.
Proses	Dimulai dengan memeriksa apakah jalur dari titik awal ke titik akhir ditemukan (jarak pada titik akhir tidak INF). Jika jalur ditemukan, membangun jalur terpendek dengan melacak jalur dari titik akhir ke titik awal menggunakan array parent. Lalu, mencetak jarak terpendek dan jalur dari awal ke akhir.
Output	Menampilkan jarak terpendek dari titik awal ke titik akhir, mencetak jalur terpendek dalam bentuk urutan koordinat dari awal ke akhir, dan jika jalur tidak ditemukan, menampilkan pesan "No path found."

f. Fungsi main

Bagian	Deskripsi
Input	Tidak ada input langsung dari fungsi lain, tetapi meminta input nama file dari pengguna.
Proses	Dimulai dengan meminta pengguna memasukkan nama file maze melalui scanf. Lalu, memanggil fungsi readMaze untuk membaca maze dari file. Kemudian, mengukur waktu sebelum dan sesudah menjalankan algoritma Dijkstra.

	Lalu, memanggil dijkstra untuk menemukan jalur terpendek dan memanggil printShortestPath untuk mencetak hasilnya. Diakhiri dengan menghitung waktu eksekusi algoritma.
Output	Menampilkan waktu eksekusi algoritma. Memanggil fungsi lain (readMaze, dijkstra, printShortestPath) untuk menampilkan hasil yang sesuai.

4.3.2 Breadth First Search (BFS)

Penyelesaian masalah *maze problem* dengan BFS dilakukan dengan mencari titik awal 'S' dan titik akhir 'E' terlebih dahulu. Kemudian dilakukan pencarian jalur di sekitar titik awal dan titik-titik selanjutnya yang dipilih sampai koordinat tujuan akhir. Setiap jalur yang dipilih dimana ia belum dikunjungi sebelumnya dan merupakan koordinat yang valid (masih berada dalam peta) akan disimpan dalam queue sedangkan jalur yang sudah dikunjungi akan diubah nilainya menjadi 1 dan tidak akan masuk ke dalam queue lagi. Jalur dipilih dengan menambahkan nilai -1, 1 atau 0 ke koordinat saat ini untuk bergerak ke atas, bawah, kanan, dan kiri. Koordinat baru (yang sudah ditambah dengan nilai tersebut) akan dimasukkan ke dalam queue kemudian dikeluarkan lagi untuk dianalisis jalur di sekitar koordinat itu. Flowchart dan DFD dapat dilihat pada lampiran. Implementasi algoritma ini dibagi menjadi beberapa bagian sebagai berikut.

a. Header Libraries

Header	Deskripsi
stdio.h	Fungsi-fungsi input dan output seperti printf dan scanf
stdlib.h	Untuk penyusunan struct dan alokasi memori
string.h	Penggunaan array of char sebagai string dan fungsi-fungsi untuk seperti strlen, strcpy, dan strcmp untuk mengoperasikan string.
time.h	Untuk mengukur waktu berlalunya proses pencarian jalur dengan clock.

b. Definisi value dan struct

Bagian	Deskripsi
Value	Value untuk MAX_ROWS dan MAX_COLS dibuat menjadi nilai tetap, yaitu 100 agar tidak perlu didefinisikan secara berkala dalam program
Struct Point	Digunakan untuk menyatakan koordinat x dan y bertipe integer
Struct Node	Digunakan untuk menyatakan linked list dimana komponen di dalamnya berupa Point
Struct Queue	Digunakan untuk menyatakan queue yang terdiri dari front dan rear dengan tipe data Node

c. Fungsi initQueue

Bagian	Deskripsi
Input	Input yang dimasukan ke dalam fungsi ini berupa queue yang akan di NULL kan
Proses	Membuat bagian front dan rear queue menjadi NULL
Output	Queue dengan front dan rear yang sudah bernilai NULL

d. Fungsi isEmpty

Bagian	Deskripsi
Input	Input berupa queue yang akan dicek apakah kosong atau tidak
Proses	Pengecekan isi queue
Output	Queue yang sudah dicek, true bila kosong dan false bila queue tidak kosong

e. Fungsi enqueue

Bagian	Deskripsi
Input	Input berupa queue yang ke dalamnya akan ditambahkan data baru berupa point dan point

	yang merupakan data baru yang akan ditambahkan
Proses	Akan dibuat variabel node baru dengan tipe Node dan merupakan array dinamis. Jika node kosong maka program akan berhenti. Pada node baru akan dimasukkan data baru (point). Jika queue kosong, maka node baru akan dimasukkan ke bagian front dan rear queue sedangkan jika tidak, node baru tersebut akan dimasukkan ke bagian rear queue dan rear selanjutnya (dapat dilakukan karena rear bertipe linked list Node) dalam queue
Output	Isi queue yang sudah diupdate

f. Fungsi dequeue

Bagian	Deskripsi
Input	Input berupa queue yang isinya akan dikeluarkan
Proses	Akan dilakukan pengecekan apakah queue kosong atau tidak, jika iya, program akan berhenti. Jika queue tidak kosong, akan dibuat variabel temp bertipe Node dan diisi dengan front dari queue. Dibuat pula variabel p yang berisikan point dari variabel temp. Ubah isi front dari queue menjadi front selanjutnya (bisa dilakukan karena berupa linked list) sehingga nilai paling depan queue yang sudah disimpan dalam temp dapat dikeluarkan. Jika front queue bernilai NULL, buat rearnya menjadi NULL juga
Output	Point dari variabel temp (bagian depan dari queue yang disimpan dalam temp dan sudah dihilangkan dari queue) yang berupa koordinat x dan y

g. Fungsi isValidMove

Bagian	Deskripsi
Input	Input berupa koordinat x dan y yang baru setelah bergerak untuk mencari jalur, jumlah baris dan kolom, matrix peta dan matrix visited (menyimpan koordinat yang sudah dikunjungi)

Proses	Mengecek apakah koordinat x yang baru lebih dari nol dan kurang dari jumlah max baris, apakah koordinat y yang baru lebih dari nol dan kurang dari jumlah max kolom, apakah matrix peta baris ke-x dan kolom ke-y merupakan '.' atau karakter 'E', dan apakah belum dikunjungi.
Output	Akan bernilai true (1) jika pergerakan ke koordinat x dan y yang baru valid dan akan bernilai 0 jika pergerakan tidak valid

h. Fungsi printShortestPath

Bagian	Deskripsi
Input	Input berupa koordinat awal dimana 'S' ditemukan, koordinat tujuan dimana 'E' ditemukan, dan koordinat parent ("orangtua" dari koordinat tertentu)
Proses	Akan dibuat variabel current bertipe Point yang akan diisi dengan nilai koordinat tujuan, variabel pathLength yang bernilai 0, dan array path berukuran MAX_ROWS*MAX_COLS. Ketika koordinat x dan y dari variabel current tidak sama dengan koordinat x dan y awal, isi array path ke-pathlength (dengan nilai pathlength terus bertambah selama kondisi tidak memenuhi) akan diubah menjadi koordinat x dan y yang tersimpan dalam variabel current. Kemudian isi current akan diubah menjadi isi dari matrix parent pada baris ke-(nilai yang sama dengan koordinat x dari variabel current) dan kolom ke-(nilai yang sama dengan koordinat y dari variabel current). Kemudian nilai array ke-pathlength (dengan pathlength merupakan nilai pathlength terakhir setelah keluar dari looping ditambah satu) diubah menjadi sama dengan koordinat awal. Setelahnya akan dicetak jalur

	terpendek dengan format koordinat (x,y) dari array path yang paling belakang karena sebelumnya disimpan secara terbalik (koordinat awal terletak di paling belakang array)
Output	Jalur terpendek dengan format koordinat (x,y) dari titik awal ke tujuan akhir

i. Fungsi bfs

Bagian	Deskripsi
Input	Input berupa koordinat awal dan akhir, jumlah max baris dan kolom, serta matrix peta yang digunakan
Proses	Akan dibuat matrix visited bertipe integer untuk <i>tracking</i> koordinat mana saja yang sudah dikunjungi, matrix parent untuk menyimpan "orangtua" dari setiap koordinat, dan variabel q yang bertipe queue. Pada queue akan dibuat nilai front dan rearnya menjadi NULL terlebih dahulu menggunakan subfungsi initQueue. Kemudian tambahkan koordinat awal pada queue menggunakan subfungsi enqueue. Setelahnya buat matrix visited baris ke-(nilai yang sama dengan koordinat x awal) dan kolom ke-(nilai yang sama dengan koordinat y awal) menjadi 1 (menandakan sudah dikunjungi). Kemudian buat matrix baris ke-(nilai yang sama dengan koordinat x awal) dan kolom ke-(nilai yang sama dengan koordinat y awal) menjadi koordinat awal (menandakan bahwa "orangtua" dari koordinat awal adalah koordinat awal itu sendiri). Akan dibuat pula array dx untuk bergerak dalam baris (ke kanan atau ke kiri) dan array dy untuk bergerak dalam kolom (ke atas dan ke bawah) Ketika queue tidak kosong, akan dilakukan looping berikut (sampai kondisi tidak lagi memenuhi). Nilai pada queue akan dikeluarkan menggunakan

	<p>subfungsi dequeue dan disimpan pada variabel current bertipe Point. Jika koordinat xy current sama dengan koordinat xy tujuan akhir, jalur terpendek akan dikeluarkan menggunakan subfungsi printShortestPath kemudian looping akan berhenti. Jika tidak sama, akan dilakukan perulangan untuk mencari koordinat x dan y baru dengan menambahkan koordinat x dan y yang sekarang dengan array dx dan dy ke-i (dengan i dimulai dari 0 sampai 3 karena terdapat 4 arah. Iterasi ke-0 berarti bergerak ke kiri, 1 bergerak ke kanan, 2 bergerak ke bawah, 3 bergerak ke atas).</p> <p>Pergerakan tersebut akan dicek terlebih dahulu, apakah valid atau tidak menggunakan fungsi isValidMove. Jika valid, koordinat xy baru tersebut akan dimasukkan ke dalam queue menggunakan fungsi enqueue. Kemudian matrix visited baris ke-(koordinat x baru) dan kolom ke-(koordinat y baru) akan diubah menjadi 1 (menandakan sudah dikunjungi). Kemudian "orangtua" dari koordinat baru ini merupakan variabel current, koordinat "orangtua" ini akan dimasukkan ke dalam matrix parent baris ke-(koordinat x baru) dan kolom ke-(koordinat y baru). Jika sudah mencapai akhir kolom dan baris peta, tetapi masih tidak menemukan koordinat tujuan akhir, akan dikatakan bahwa jalur tidak ditemukan.</p>
Output	Jika tujuan akhir ditemukan, akan dikeluarkan jalur terpendek dengan format koordinat (x,y) sedangkan jika tujuan tidak ditemukan, akan dikatakan bahwa tidak ada jalur yang ditemukan

j. Fungsi main

Bagian	Deskripsi
--------	-----------

Input	Nama file eksternal yang berisikan peta
Proses	Akan dicek terlebih dahulu, apakah nama file valid atau tidak. Jika tidak valid, program akan berhenti dan dikatakan bahwa file tidak ditemukan. Jika valid, program akan memproses file dengan pertama-tama memindahkan setiap baris pada file ke dalam sebuah matrix peta, kemudian menghitung jumlah max baris dan kolom dari peta. Setelahnya akan dilakukan iterasi untuk mencari karakter 'S' sebagai titik awal yang disimpan dalam sebuah variabel start bertipe Point, pada iterasi yang sama akan dilakukan pencarian karakter 'E' sebagai titik akhir yang disimpan dalam sebuah variabel end bertipe Point. Sebelum pencarian jalur menggunakan subfungsi bfs dijalankan, clock untuk menghitung waktu yang dibutuhkan untuk mencari jalur terpendek akan dimulai. Setelah subfungsi bfs selesai, clock akan berhenti dan menampilkan jumlah waktu yang dibutuhkan
Output	Jalur terpendek dengan format koordinat (x,y) beserta waktu yang diperlukan.

4.3.3 Depth First Search (DFS)

4.3.3.1 Penjelasan Dari Struct Yang Digunakan

Nama Struct	Deskripsi
Coordinate	<p>Berisi integer row dan integer column.</p> <p>Struct ini akan digunakan di dfs untuk eksplorasi path-path yang ada</p>
Maze	Berisi cells yang akan memuat map yang dibaca dari file, rows dan cols yang merepresentasikan baris dan kolom dari cells tersebut serta coordinate start dan

	coordinate end yang menyimpan titik awal dan titik akhir dari maze.
--	---

4.3.3.2 Penjelasan dari Variabel-variabel pada fungsi

global	Bool visited[100][100] : Sebagai map yang digunakan untuk menandai apakah suatu koordinat sudah dikunjungi atau belum Char direction[] : Untuk merepresentasikan pergerakan sebagai suatu char yang nantinya digunakan untuk currentPath.
Bool isValid	Parameter fungsi : X : sebagai titik x yang dicek Y : sebagai titik y yang dicek. Rows : size baris dari maze. Cols : size kolom dari maze.
Void display_maze	Parameter fungsi : Maze* maze = Maze yang telah dibaca dari file.
Void dfs	Parameter fungsi : Maze* maze = Maze yang telah dibaca dari file. Coordinate* coordinate = sebagai koordinat yang sedang dicek. Char currentPath[] = untuk menyimpan Path dari start ke end. Int pathLength = representasi dari berapa kali path bergerak Char ans[][100] = kumpulan string yang merupakan pergerakan dari pencarian yang

	berhasil sampai ke titik end. Fungsi lokal : Int dx = sebagai pergerakan yang mungkin dititik x (antara maju atau mundur). Int dy = sebagai pergerakan yang mungkin dititik y (antara ke kiri atau kanan). Int newX = nilai X setelah ditambahkan oleh dx. Int newY = nilai Y setelah ditambahkan oleh dy.
Void findPath	Parameter fungsi : Maze* maze = Maze yang telah dibaca dari file. Variabel lokal : CurrentPath[max] = sebagai penyimpanan sementara pergerakan dari titik start ke end yang nantinya akan dicopy ke variabel ans.

4.3.3.3 Penjelasan dari Fungsi-fungsi

Bagian	Deskripsi
Bool isValid	Fungsi ini berfungsi untuk memastikan bahwa pergerakan dari koordinat diantara $0 < \text{koordinat} < \text{jumlah row/jumlah col}$.
Void display_maze	Fungsi ini berfungsi untuk visualisasi maze bagi user.
void dfs	Fungsi ini dijalankan dengan konsep rekursif. Jadi ketika dimulai, fungsi ini akan mengassign syarat basis untuk mengecek apakah tiap program dibacktrack dicek apakah sudah mencapai syarat basis tersebut atau belum. Jadi, pada fungsi ini syarat basisnya adalah ketika struct coordinate yang digunakan

	<p>untuk eksplorasi nilainya sama dengan titik end.</p> <p>Namun ketika belum mencapai syarat basis, karena pada program saya terdapat 2 jenis map yaitu maze.cells dan visited dan visited berfungsi untuk menandai titik koordinat tersebut sudah dikunjungi atau belum maka pada saat ini visited akan ditandai sebagai true. Lalu, dengan variabel pergerakan yaitu dx dan dy, diassign koordinat baru yang merupakan hasil penambahan koordinat lama ditambah dengan pergerakan yang dilakukan. Ketika koordinat tersebut belum dikunjungi dan selama bukan tanda pagar, maka currentPath yang merupakan variabel penyimpan pergerakan sementara ditambahkan ke direction dan akan dijalankan dfs lagi dengan nilai koordinat newX dan newY tersebut.</p>
Void findPath	<p>Fungsi ini berfungsi untuk mencariPath yang mungkin serta menampilkan longest path, shortest path dan jumlah path yang terdapat pada maze.</p> <p>Cara kerja dari fungsi ini adalah terdapat variabel currentPath yang berfungsi untuk menyimpan move yang dijalankan oleh algoritma dfs. Ketika algoritma dfs sudah dijalankan, akan dicek count apakah variabel count yang sebagai jumlah pergerakan sama dengan 0 atau tidak. Apabila tidak, akan dihitung jumlah pergerakan yang terdapat pada variabel ans. Setelah itu, akan dihitung shortest path distance dan longest path distance. Serta di print pathnya satu-satu.</p>
Fungsi main	<p>Pada fungsi, pertama dimulai dengan deklarasi variabel-variabel yang akan digunakan serta struct-struct akan digunakan. Kemudian, dilakukan pengecekan file kosong atau tidak. Setelah itu, akan dilakukan</p>

	<p>pembacaan file ke variabel line serta perhitungan baris dan kolom agar program dapat menetapkan size dari cells dari maze. Setelah itu, memori akan dialokasikan untuk maze.cells dan kemudian akan didapatkan map dari file akan dimasukkan ke maze.cells dan akan dilakukan penetapan titik start dan titik akhir.</p>
--	---

4.3.4 Dynamic Programming

Pendekatan penyelesaian *maze problem* dengan dynamic programming dilakukan dengan konsep tabulasi dan sejenis flood-fill algorithm untuk menyimpan data traversal pada setiap iterasi di sebuah matrix. Matrix ini digunakan untuk memeriksa apabila traversal sudah pernah dilakukan untuk mencapai sebuah titik tertentu sehingga perhitungan jarak dan jalur yang diambil lebih efisien. Implementasi algoritma ini (dapat dilihat pada lampiran laporan) dibagi menjadi beberapa bagian sebagai berikut.

a. Header Libraries

Header	Deskripsi
stdio.h	Fungsi-fungsi input dan output seperti printf dan scanf
stdlib.h	Untuk penyusunan struct dan alokasi memori
string.h	Penggunaan array of char sebagai string dan fungsi-fungsi untuk seperti strlen, strcpy, dan strncpy untuk mengoperasikan string.
stdbool.h	Penggunaan tipe data boolean true dan false
time.h	Untuk mengukur waktu berlalunya proses pencarian jalur dengan clock.

b. Definisi value, struct dan variabel global

Bagian	Deskripsi
Definisi value	Value untuk MAX diatur menjadi 256

	<p>untuk menandakan nilai maksimum yang dapat diproses dalam beberapa hal tertentu seperti pembacaan line dan inisialisasi matrix untuk map yang diterima.</p> <p>Value untuk INF diatur menjadi 999 yang menandakan sebuah angka sangat besar dalam inisialisasi matriks tabulasi (untuk penyimpanan data traversal) yang artinya titik tersebut belum dijelajahi.</p> <p>Kedua value ini tidak memiliki tipe data tertentu dan berfungsi seperti sebuah makro yang mengembalikan nilai yang didefinisi ketika dipanggil.</p>
Definisi struct Cell	Struct Cell menyimpan data integer length yang menandakan jarak dari titik mulai hingga titik pada koordinat tertentu, integer prevX dan prevY untuk menyimpan titik yang ditraversi sebelumnya.
Definisi global matrix of Cell shortestPathLengths	Matrix ini didefinisikan secara global agar tidak perlu dijadikan argumen parameter dalam subfungsi lainnya. Matrix of Cell ini menyimpan data jalur terpendek.

c. Fungsi isValid

Bagian	Deskripsi
--------	-----------

Input	Input yang didapatkan dari argumen parameter fungsi adalah integer x dan y yang menandakan titik yang ingin diuji dalam bentuk koordinat, serta integer rows dan cols yang menandakan batasan baris dan kolom pada map labirin yang dibaca pada file txt.
Output	Fungsi ini memeriksa apabila titik uji x dan y berada dalam batasan map yang digunakan dimana fungsi akan mengembalikan boolean true apabila titik uji x dan y berada dalam batasan map dan false apabila titik uji tidak berada dalam batasan map.

d. Fungsi findShortestPath

Bagian	Deskripsi
Input	Fungsi ini memiliki argumen parameter integer startX, startY, endX, endY yang secara berurutan menandakan titik koordinat character 'S' ditemukan dan character 'E' ditemukan. Selain itu, terdapat juga integer rows dan cols yang menandakan batasan baris dan kolom pada map yang digunakan.
Proses	Pada awalnya, fungsi ini menginisialisasi data pada global matrix yang telah di definisi (shortestPathLengths) dengan nilai length nya semua menjadi INF serta prevX dan prevY menjadi -1. Nilai-nilai dalam matrix ini menandakan bahwa belum ada jalur yang dijelajahi. Setelah itu, length pada titik

	<p>start diatur menjadi 0 dan sebuah boolean updated di inisialisasikan. Selanjutnya, logika untuk menjalankan traversal jalur diterapkan dengan while-do loop berkondisi boolean updated. Pada setiap iterasinya, nilai updated ini diatur ulang menjadi false dan sebuah nested loop yang mengiterasi sepanjang batasan map (rows dan cols). Dalam nested loop ini, diperiksa apakah titik yang sedang dijelajahi bukan halangan (character '#') dan titik traversal selanjutnya valid dan memiliki jarak baru yang lebih kecil dari jarak saat itu (disimpan dengan integer minNeighbor). Setelah diperiksa, data titik traversi sebelumnya disimpan pada integer prevX dan prevY. Setelah memeriksa traversi pada 4 arah (atas, bawah, kiri, kanan), algoritma memeriksa apabila minNeighbor yang ditemukan lebih kecil dari jarak yang ditempuh pada titik yang ditinjau untuk mengupdate nilai pada global matrix sesuai dengan jarak dan titik prev yang terkecil pada iterasi tersebut. Selanjutnya, nilai boolean updated diatur menjadi true agar iterasi while-do loop tetap berjalan. Pada kondisional yang sama, diperiksa juga apabila titik yang ditinjau saat itu sudah mencapai titik end. Apabila iya, maka fungsi ini akan diterminasi karena traversal sudah selesai mencapai tujuan.</p>
--	---

e. Fungsi printShortestPath

Bagian	Deskripsi
Input	<p>Fungsi ini memiliki argumen parameter integer startX, startY, endX, endY yang secara berurutan menandakan titik kordinat character 'S' ditemukan dan character 'E' ditemukan. Selain itu, terdapat juga integer rows dan cols yang menandakan batasan baris dan kolom pada map yang digunakan.</p>
Proses	<p>Pada awalnya, fungsi ini memeriksa global matrix shortestPathLengths pada titik end telah dijelajahi (jika belum, length akan bernilai INF dan fungsi akan berakhir). Apabila ditemukan jalur, sebuah matrix of integer path[MAX][2] diinisialisasi untuk menyimpan data titik x dan y yang dijelajahi (beserta 3 buah variabel integer length = 0 serta x = endX dan y = endY). Selanjutnya, nilai-nilai dari global matrix dipindah pada variabel lokal dengan iterasi while-do loop dimana setiap iterasi (selama belum mencapai start point yang dilambangkan x dan y = -1) nilai path[length] dimasukkan dengan nilai x dan y serta x dan y dimundurkan ke titik sebelumnya pada prev yang disimpan di global matrix. Setelah iterasi berakhir, kordinat yang disimpan pada matrix path di print dengan for-loop dan output tersebut diubah menjadi bentuk grid map agar dapat</p>

	terlihat bentuk pergerakan yang dijalani dengan lebih jelas.
--	--

f. Fungsi Main

Bagian	Deskripsi
Deklarasi variabel	Fungsi ini merupakan hasil integrasi dari fungsi-fungsi yang dirancang sebelumnya serta pembacaan input (dengan validasi) dan pemindahan data antar fungsi. Pada bagian awal fungsi ini, dideklarasikan string filename dengan panjang MAX, startclk dan endclk dan cpu_time_used.
Pembacaan input	Dengan fungsi scanf, nama file yang dimasukkan user pada terminal akan disimpan dan dibuka dengan fungsi fopen dalam mode read. Apabila file tidak ditemukan, sebuah pesan error akan dikeluarkan dan program akan di-terminate (dimatikan).
Parsing file	Variabel integer row dan col, serta string line dengan panjang MAX diinisialisasikan. Selanjutnya, sebuah line dibaca dari file dan disimpan pada line dengan panjang (tanpa newline character) disimpan pada col. File yang dibaca direwind agar dapat membaca nilai row dan parsing map sepenuhnya. Setelah direwind, sebuah loop akan dijalankan untuk membaca file secara line-per-line. Setiap iterasinya, nilai row akan diinkrementasi dan line

	tersebut akan dicopy ke dalam maze[row] tanpa newline character. Apabila ditemukan baris dengan panjang yang berbeda, program akan memberikan error message dan exit sistem. Setelah mencapai akhir dari pembacaan file, file eksternal akan ditutup dengan fclose. Data yang disimpan pada maze akan diprint dengan iterasi untuk menampilkan grid yang terbaca.
Pencarian nilai awal dan akhir	Integer startX, startY, endX, endY diinisialisasikan dengan nilai -1 dan sebuah iterasi sepanjang maze dijalankan untuk mencari titik 'S' dan titik 'E'. Apabila titik tersebut ditemukan, nilai indeksinya akan disimpan pada variabel yang dideklarasikan sebelumnya. Jika tidak, program akan mengirimkan pesan bahwa titik yang dicari tidak ditemukan dan secara langsung menjalankan terminasi pada fungsi.
Implementasi subfungsi dan perhitungan waktu	Fungsi ini akan mengatur nilai startclk menjadi clock() yang menyimpan waktu pada saat dijalkannya perintah tersebut. Selanjutnya subfungsi findShortestPath dan printShortestPath dijalankan secara berurutan dan dilanjutkan dengan assignment endclk dengan clock(). Kemudian, cpu_time_used diatur agar merupakan nilai rata-rata dari antara startclk dan endclk dengan pembagian oleh CLOCKS_PER_SEC

	(bernilai 1 juta pada sistem 32-bit) dan hasil tersebut ditampilkan sebagai waktu yang dibutuhkan untuk menjalankan algoritma dynamic programming secara keseluruhan.
--	---

4.3.5 Algoritma A*

Algoritma A* bekerja dengan menghitung dua nilai utama untuk setiap titik: G-cost, yaitu jarak atau cost dari titik awal ke titik saat ini, dan H-cost, yaitu perkiraan jarak dari titik saat ini ke titik tujuan (heuristik). Nilai total yang digunakan untuk menentukan jalur terbaik adalah F-cost, yang merupakan penjumlahan dari G-cost dan H-cost. Algoritma A* memilih titik dengan nilai F terendah untuk dievaluasi, memperbarui nilai untuk tetangganya, dan memindahkan titik yang dievaluasi ke daftar tertutup. Proses ini berlanjut hingga mencapai tujuan atau tidak ada lagi titik yang bisa dievaluasi, menjadikan A* lebih cepat dibandingkan algoritma lain karena menggunakan heuristik untuk memandu pencarian.

A) Struktur Data

Definisi Konstanta

MAX_L adalah ukuran maksimal peta, fungsi abs adalah fungsi untuk menghitung nilai absolut

Definisi Struct

Struct Pos digunakan untuk menyimpan posisi dalam koordinat x dan y sebagai int. Struct Cell digunakan untuk menyimpan informasi tentang setiap sel pada peta, open menyatakan apakah cell tersebut ada pada open list atau tidak, closed menyatakan apakah cell tersebut ada pada closed list atau tidak, f g dan h menyatakan nilai cost dan heuristics cell tersebut, dan parent menunjukan posisi parent cell tersebut.

Fungsi Pendukung

a) Fungsi isValid

Fungsi isValid berguna untuk menentukan apakah posisi pos berada dalam batas peta atau tidak. Batas peta yang valid dinyatakan apabila pos berada pada dalam ROW dan COL.

b) Fungsi isObstacle

Fungsi isObstacle berguna untuk menentukan apakah posisi pos merupakan cell obstacle yang ditandai dengan "#". Fungsi ini akan digunakan sebagai syarat sebuah cell akan diturunkan sebagai sebuah Successor.

c) Fungsi isSame

Fungsi isSame digunakan untuk menentukan apakah dua pos merupakan posisi yang sama. Fungsi ini digunakan untuk

d) Fungsi hValue

Fungsi hValue digunakan untuk menghitung nilai heuristik antara sebuah posisi dan tujuan. Untuk implementasi hVal pada fungsi ini menggunakan jarak Manhattan.

e) Fungsi findStartEnd

Fungsi findStartEnd digunakan untuk mengisi posisi yang merupakan posisi start yang ditandai dengan 'S' pada map dan posisi end yang ditandai dengan 'E' pada map. Fungsi ini juga dapat mendeteksi apakah sebuah map valid atau tidak ditandai dengan kelengkapan S dan E pada map.

f) Fungsi findminF

Fungsi ini digunakan untuk mencari sel dengan nilai f terkecil yang masih terbuka. Ini diperlukan sebagai fungsi dasar dari algoritma A* dimana setiap iterasi dimulai dari Node terbuka yang memiliki nilai f terkecil.

g) Fungsi addSuccessor

Fungsi ini digunakan untuk menambahkan cell yang memiliki vertices yang terhubung dengan cell pada suatu saat ke open list yang berarti jalan ke cell tersebut ditemukan dan apabila sudah ada jalan yang lain ditemukan dari iterasi sebelumnya, maka fungsi ini akan mengupdate nilai f g h dari cell tersebut apabila ditemukan jalan dengan nilai f yang lebih kecil.

h) Fungsi start_Astar

Fungsi ini digunakan untuk menginisiasi algoritma Astar. Pertama fungsi ini akan menginisiasi variabel variabel yang akan digunakan seperti Pos start dan end, matrix cellDetails, dan lalu akan memanggil fungsi A_star untuk memulai iterasi. Fungsi ini juga akan mengeluarkan output apakah path ditemukan atau tidak.

i) Fungsi A_star

Fungsi A_star merupakan fungsi implementasi utama algoritma A_star dimana fungsi ini akan mulai dengan menentukan cell mana untuk memulai iterasi dengan menggunakan fungsi findminF untuk menentukan cell open yang memiliki f terkecil. Kemudian fungsi ini akan memindahkan cell tersebut dari open menjadi closed. Lalu akan dilakukan perhitungan tiap successor yaitu untuk UP DOWN LEFT RIGHT atau semua kemungkinan jalan dari sebuah cell. Iterasi ini terus berlanjut hingga cell End dicapai atau tidak ada lagi cell open pada sebuah iterasi.

j) Fungsi tracePath

Fungsi ini digunakan untuk mengeluarkan output hasil dari algoritma Astar dimulai dengan tracing path terdekat yang didapat dari parent cell end sampai pada cell start. Kemudian fungsi ini akan mengeluarkan map dengan path terdekat yang ditandai oleh "*".

k) Fungsi main

Fungsi main digunakan untuk melakukan parsing file eksternal, serta menentukan row dan col yang ada pada map. Kemudian fungsi ini juga digunakan untuk menginisiasi fungsi start_astar dan melakukan perhitungan waktu time taken.

4.3.6 Algoritma Backtracking

Tabel Header

Header	Penjelasan
stdio.h	Library standard input output
stdlib.h	Library for memory manipulation and standard function
string.h	Library for string manipulation
time.h	Library for time related operation

Tabel macro dan typedef

Nama	Penjelasan
# define MAX 256	Nilai maksimal dengan besar 256 sebagai nilai maksimal dari

	panjang dan lebar map
typedef struct pos { int x; int y; } pos;	Tipe variable pos yang terdiri dari int x dan int y sebagai koordinat.

Fungsi Main akan meminta input nama file .txt lalu akan di parse ke dalam matrix. Akan dilakukan inisiasi beberapa variable seperti variable pos sol[MAX] (menyimpan jalur solusi, digunakan seperti stack), pos start, pos end, pos curr, sol_counter, dan finish(nilai 1 jika maze ditaklukan dan 0 jika gagal).

While loop akan terus berjalan selama masih ada posisi dalam stack solusi (sol_counter > 0).

Posisi saat ini diambil (dipop) dari stack (curr = sol[--sol_counter];).

Jika posisi saat ini adalah posisi akhir, fungsi akan keluar dari loop (if (curr.x == end.x && curr.y == end.y) { ... break; }).

Jika posisi saat ini bukan posisi akhir, fungsi akan mencoba bergerak ke posisi baru dalam satu dari empat arah (atas, bawah, kiri, kanan).

Jika bergerak ke posisi baru adalah mungkin (berada dalam batas labirin dan merupakan ruang kosong '.' atau posisi akhir 'E'), posisi saat ini akan dipush kembali ke stack dan posisi baru juga akan dipush ke stack (sol[sol_counter++] = curr; sol[sol_counter++] = next;).

Jika bergerak ke posisi baru tidak mungkin dalam semua arah, loop akan melanjutkan dengan posisi berikutnya dari stack, secara efektif bergerak kembali ke posisi sebelumnya. Gerakan ini adalah langkah backtracking.

4.3.7 Algoritma Greedy

Header dan macro

- **#include <stdio.h>**: Menyertakan pustaka standar untuk input dan output.
- **#include <stdlib.h>**: Menyertakan pustaka standar untuk fungsi umum seperti **malloc**, **free**, dan **exit**.
- **#include <string.h>**: Menyertakan pustaka standar untuk manipulasi string.
- **#include <stdbool.h>**: Menyertakan pustaka untuk mendefinisikan tipe data boolean (**true** dan **false**).

- **#include <time.h>**: Menyertakan pustaka standar untuk operasi terkait waktu.
- **#define MAX 256**: Mendefinisikan konstanta **MAX** dengan nilai 256, digunakan sebagai ukuran maksimum array.

Struct

Cell dan **Pos** adalah tipe data struktural yang digunakan untuk menyimpan koordinat (x, y) dari sel atau posisi dalam maze.

Fungsi isValid

Fungsi ini memeriksa apakah koordinat (**x**, **y**) berada dalam batas-batas maze. Mengembalikan **true** jika valid, **false** jika tidak.

Fungsi Solve Maze

Fungsi ini bertanggung jawab untuk menyelesaikan maze dari posisi start ke end. Menggunakan array **path** untuk menyimpan jalur yang diambil. Fungsi akan memeriksa apakah jalur disekitarnya ada yang valid atau tidak.

Main function

- Terdapat fungsi untuk membaca file dan menyimpan maze ke dalam array maze dan menghitung jumlah baris dan kolom.
- Terdapat fungsi untuk mencari posisi start dan end yang nanti akan digunakan ke dalam fungsi solve maze.
- Terdapat fungsi mengukur waktu eksekusi untuk mencari berapa waktu yang dibutuhkan algoritma untuk menyelesaikan programnya

Bagian ini berisikan penjelasan kontribusi setiap anggota kelompok serta penjelasan dari kode yang Anda buat (struktur *source code*, penjelasan pemetaan kode ke rancangan, penjelasan fungsi, dll.). Jika ada perbedaan antara rancangan dan implementasi rancangan atau perbedaan antara pembagian tugas dan kontribusi anggota, jelaskan alasannya di bagian ini.

4.4 PENGUJIAN

4.4.1 Dijkstra

Program dengan algoritma dijkstra hanya dapat mencari jalur terpendek saja. Untuk mencari keseluruhan jalur, algoritma dijkstra tidak

memungkinkan karena kinerjanya adalah membandingkan nilai setiap jalur dan diambil sesuai jarak terpendek. Untuk pencarian jarak jalur terpanjang, pembuat belum terpikirkan membuat implementasinya karena keterbatasan waktu.

Tabel 4-1 Waktu yang dibutuhkan algoritma Dijkstra untuk setiap test case

Input File	Jarak terkecil	Runtime (s)
maze1	18	0.005
maze2	13	0.007
maze3	18	0
maze4	19	0.005
maze5	-	0
maze6	-	-
maze7	14	0.006

Kompleksitas waktu dari kode dengan algoritma Dijkstra tersebut didominasi oleh fungsi dijkstra, yang memiliki kompleksitas $O((N * M)^2)$, di mana N adalah jumlah baris dan M adalah jumlah kolom dalam maze. Fungsi readMaze memiliki kompleksitas $O(N * M)$ karena membaca setiap karakter dalam file dan memeriksa konsistensi panjang baris. Fungsi printShortestPath memiliki kompleksitas $O(N * M)$ karena melakukan backtracking dari titik akhir ke titik awal untuk membentuk jalur terpendek. Kombinasi fungsi-fungsi ini menyebabkan total kompleksitas waktu program adalah $O((N * M)^2)$, yang berarti waktu eksekusi meningkat secara kuadratik dengan ukuran maze.

4.4.2 Breadth First Search (BFS)

Program ini dibuat hanya untuk menemukan jalur terpendek karena keterbatasan waktu dan tidak terpikirkannya algoritma untuk menemukan semua kemungkinan jalur beserta jalur terpanjang. Seharusnya dengan algoritma ini, pencarian semua jalur yang bisa dilalui dan jalur terpanjang dapat dilakukan dengan menyimpan semua jalur pada suatu struct yang

terdapat 2 linked list di dalamnya, yaitu untuk path selanjutnya dan untuk path sister.

Tabel 4-2 Waktu yang dibutuhkan algoritma BFS untuk setiap test case

Input File	Jarak terkecil	Runtime (s)
maze1	19	0.003
maze2	14	0.003
maze3	19	0.003
maze4	20	0.003
maze5	-	0
maze6	-	-
maze7	15	0.002

Berdasarkan analisis kompleksitas waktu, diketahui bahwa subfungsi `initQueue`, `isEmpty`, `enqueue`, `dequeue`, dan `isValidMove` memiliki kompleksitas $O(1)$ karena tidak terdapat looping di dalamnya dan waktu runtimenya tetap. Pada subfungsi `printShortestPath`, kompleksitas waktunya adalah $O(n)$ dengan n tidak tentu karena akan menyesuaikan jumlah panjang jalur yang diperlukan dari koordinat akhir ke koordinat awal. Pada subfungsi `bfs`, kompleksitas waktunya adalah $O(n*m)^2$ dengan n merupakan banyaknya perulangan yang dilakukan sampai queue kosong sedangkan m merupakan jumlah panjang jalur yang diperlukan dari koordinat akhir ke awal (ditambah dengan perulangan m karena terdapat fungsi `printShortestPath` pada fungsi `bfs`). Pada fungsi `main`, kompleksitas waktu untuk bagian mencari jumlah kolom dan baris serta menyalin isi baris file eksternal ke array peta adalah $O(n)$ dengan n adalah banyaknya baris yang ada pada file eksternal. Untuk bagian mencari koordinat awal 'S' dan akhir 'E', kompleksitas waktunya adalah $O(n*m)$ dengan n adalah jumlah kolom dan m adalah jumlah baris peta. Karena pada `main` digunakan fungsi `bfs`, maka terdapat kompleksitas waktu fungsi `bfs` pula, yaitu $O(n*m)$.

Pada maze1, diperlukan kompleksitas waktu sebanyak $(12*11)^2 = 17424$. Angka yang didapat cukup besar karena terjadi banyak perulangan

ketika dilakukan pencarian titik awal dan titik akhir (jumlah kolom dan barisnya banyak).

Pada maze2, diperlukan kompleksitas waktu sebanyak $(12*10)^2 = 14400$. Angka yang didapat cukup besar karena terjadi banyak perulangan ketika dilakukan pencarian titik awal dan titik akhir (jumlah kolom dan barisnya banyak).

Pada maze3, diperlukan kompleksitas waktu sebanyak $(12*11)^2 = 17424$.

Pada maze4, diperlukan kompleksitas waktu sebanyak $(12*10)^2 = 14400$.

Pada maze5, tidak ada jalur yang terdeteksi sehingga, program tidak menjalankan fungsi `bfs` dan hanya memerlukan perulangan ketika menemukan titik awal dan akhir serta memindahkan peta pada file eksternal ke array

Pada maze6, terdapat baris yang panjangnya tidak seperti baris sebelumnya sehingga tidak masuk ke dalam fungsi `bfs` ataupun pencarian titik awal dan akhir. Perulangan hanya dilakukan ketika mencari jumlah baris dan kolom

Pada maze7, diperlukan kompleksitas waktu sebanyak $(9*7)^2 = 3969$ iterasi. Besar kompleksitas waktu ini terbukti karena pada pengujian maze, waktu runtime paling kecil yang dibutuhkan untuk menjalankan program adalah ketika dimasukkan file maze7.

4.4.3 Depth First Search (DFS)

Telah dilakukan simulasi menggunakan algoritma DFS dan beberapa dari maze dapat disimulasikan dan beberapa error. Contohnya pada maze1.txt dan maze2.txt yang gagal disimulasikan kemungkinan besar karena terlalu banyak memori yang dibutuhkan sehingga program mengalami stuck dipencarian di `FindPath`.

Namun pada maze5.txt, output mengeluarkan path terus menerus dikarenakan program tidak menyiapkan apabila tidak ada path yang dapat dilalui. Sehingga menyebabkan program mengalami pencarian terus menerus tanpa henti.

Tabel 4-1 Waktu yang dibutuhkan algoritma DFS untuk setiap test case

Input File	Jarak terkecil	Runtime (s)
------------	----------------	-------------

maze1	-	-
maze2	-	-
maze3	18	0.014
maze4	19	0.376
maze5	-	infinite
maze6	6	0.005
maze7	14	0.055

4.4.4 Dynamic Programming

Program ini hanya dapat mengkomputasi jalur terdekat antara titik 'S' dan 'E' dan tidak dapat mencari semua jalur karena batasan floodfill algorithm yang digunakan menyebabkan fungsi pencarian jalur diterminasi ketika menemukan titik E pertama kalinya. Program sudah dicoba untuk mengkalkulasi jarak terjauh dengan mengubah logika updating tabel tabulasi menjadi jarak terjauh pada setiap iterasinya. Namun terdapat masalah pada saat iterasi mencapai titik dengan ketinggian sama dengan 'E' yang menyebabkan setiap jalur terulang.

Tabel 4-4 Waktu yang dibutuhkan algoritma dynamic programming untuk setiap test case

Input File	Jarak terkecil	Runtime (s)
maze1	18	0.006
maze2	13	0.01
maze3	18	0.008
maze4	19	0.008
maze5	-	0
maze6	-	-
maze7	14	0.004

Dari analisis kompleksitas waktu, algoritma yang dirancang memiliki kompleksitas waktu yang bergantung pada dimensi file input txt yang

digunakan. Pada fungsi pencarian jalur, kompleksitas waktu dapat dianalisis dari penggunaan while loop yang mengiterasikan sepanjang grid (worst-case $N * M$ iterasi) dan saat pathfindingnya dengan nested loop (worst case $N * M$ iterasi) yang bergantung pada besar row dan col yang digunakan. Artinya, fungsi ini memiliki kompleksitas waktu $O(N^2 * M^2)$ yang setara dengan $O((M*N)^2)$ pada worst-case nya. Untuk fungsi menampilkan jalur terpendek, kompleksitas waktunya dapat dianalisis dari beberapa penggunaan loop yaitu while loop untuk memindahkan data dari variabel global ke lokal (worst-case N iterasi, tergantung pada panjang jalur), for loop untuk print jalur yang ditemukan (worst-case N iterasi, tergantung pada panjang jalur), dan nested loop untuk menampilkan jalur terpendek. Oleh karena itu, fungsi ini memiliki worst-case time complexity $O(N * M * K)$ yang bergantung pada besar row, col, dan panjang path yang ditemukan. Sehingga time complexity yang signifikan dibandingkan kompleksitas lainnya adalah $O((M*N)^2)$.

Pada maze1, jalur yang diberikan memiliki belokan yang cukup banyak namun untuk beberapa belokan tersebut akan disimpan dalam tabel tabulasi sehingga nilainya tidak perlu dihitung ulang saat traversal selanjutnya (kompleksitas waktu $O((M*N)^2)$ $(12 \times 11)^2 = 17424$ iterasi).

Pada maze2, jalur yang diberikan sangat banyak daerah terbuka sehingga penyimpanan data pada tabel tabulasi perlu dilakukan berulang kali pada tahap pemeriksaan kondisional nilai minimumnya (kompleksitas waktu $(12 \times 10)^2 = 14400$).

Pada maze3, jalur yang diberikan dibatasi sedikit sehingga arah yang perlu dijelajahi juga lebih sedikit dibandingkan maze lainnya (kompleksitas waktu $(12 \times 11)^2 = 17424$).

Pada maze 4, struktur labirin yang diberikan mirip dengan maze2 namun dengan pembatasan jalur yang lebih banyak lagi sehingga runtimenya juga ikut berkurang (kompleksitas waktu $(11 \times 10)^2 = 12100$).

Pada maze 5, file yang dimasukkan tidak terdapat masalah dalam kelengkapan character dan baris, namun tidak ada jalur langsung yang menghubungkan antara 'S' dan 'E' sehingga fungsi

path finding yang digunakan dengan cepat diterminasi.

Pada maze 6, terdapat sebuah baris dengan jumlah character yang berbeda sehingga parsing dikatakan gagal dan program berhenti.

Pada maze 7, dapat dilihat bahwa waktu yang dijalankan kode paling kecil (kompleksitas waktu $(9 \times 7)^2 = 3969$). Dari hasil perkalian row, col, dan length dari path kita dapat melihat bahwa pernyataan bahwa algoritma ini memiliki kompleksitas waktu $O((M \times N)^2)$ betul karena hasil perkalian worst-case nya sesuai dimana runtime berbanding lurus dengan kompleksitas worst-case nya.

4.4.5 Algoritma A*

Algoritma digunakan untuk pencarian jalur yang paling dekat. Algoritma A* memilih titik dengan nilai F terendah untuk dievaluasi, memperbarui nilai untuk tetangganya, dan memindahkan titik yang dievaluasi ke daftar tertutup. Proses ini berlanjut hingga mencapai tujuan atau tidak ada lagi titik yang bisa dievaluasi, menjadikan A* lebih cepat dibandingkan algoritma lain karena menggunakan heuristik untuk memandu pencarian.

Dengan adanya tambahan fungsi heuristics atau h, maka algoritma ini walaupun memiliki worst case yang sama dengan algoritma yang lainnya, kemungkinan untuk menemukan jalur terdekat lebih cepat sangat besar sehingga cocok untuk implementasi pada dunia nyata.

Dengan menganalisis Kompleksitas waktu, ditemukan bahwa kompleksitas waktu algoritma ini adalah sebagai berikut.

- Inisialisasi

Saat melakukan proses inisialisasi yaitu saat mengalokasikan memori untuk cellDetails

melakukan proses dengan $O(n)$ dengan n adalah jumlah cell atau $n = \text{ROW} \times \text{COL}$.

- Find Start End

Saat melakukan pencarian Start dan End memerlukan waktu $O(n)$ dengan n adalah jumlah cell.

- Find minF

Saat melakukan penentuan cell open dengan F yang minimum, maka saat worst case semua cell open sehingga perlu dilakukan scanning untuk setiap cell dengan kompleksitas waktu $O(n)$

- Fungsi lainnya

Fungsi fungsi yang lainnya seperti memproses successors, isValid, isObstacle dan lainnya hanya menggunakan kompleksitas waktu $O(1)$ karena tidak terpengaruh oleh jumlah data.

- Main Loop

Pada main loop atau fungsi Astar utama, dilakukan iterasi yang berulang ulang secara inf sampai tidak ada cell yang open atau cell end sudah ditemukan. Pada worst case, akan dilakukan iterasi untuk setiap cell yang ada sehingga melakukan $O(n)$ perhitungan dengan didalamnya terdapat fungsi find minF dengan kompleksitas waktu $O(n)$ pula sehingga kompleksitas waktu utama adalah $O(n^2)$ dengan n adalah jumlah cell pada map.

Berikut adalah hasil runtime untuk tiap test case.

Input File	Jarak	Runtime (s)
maze1	18	0.005
maze2	13	0.003
maze3	18	0.008
maze4	19	0.005
maze5	No solution	-
maze6	Uneven line	-
maze7	14	0.002

Tabel Runtime Algoritma A*

4.4.6 Backtracking

Algoritma ini hanya mencari sebuah solusi dan memiliki prioritas gerakan bawah kanan atas kiri.

Berikut adalah hasil runtime tiap testcase

Input File	Size	Jarak	Runtime (s)
maze1	12x11	20	0.000068
maze2	12x10	19	0.000061
maze3	12x11	18	0.000059
maze4	12x10	19	0.000063
maze5	9x7	No solution	-
maze6	uneven	Uneven line	-
maze7	9x7	14	0.000051

Kompleksitas waktu dari backtracking tidak dapat diberi rumus yang pasti karena sangat bergantung pada jumlah backtracking yang dilakukan.

4.4.7 Greedy

Pada program ini algoritma hanya akan mencari satu jalur yang ditemukan meskipun ada banyak jalur yang dapat dilalui. Algoritma ini membandingkan posisi start dengan end dan memilih jalur berdasarkan perbedaan tersebut. Ketika sudah menemukan jalur yang menghubungkan S dengan E maka program akan selesai. Dalam pengujian, program ini mengalami kesulitan ketika diberikan banyak jalan buntu dan melakukan infinite loop. Waktu untuk setiap eksekusi dengan contoh maze1.txt sampai dengan maze3.txt adalah 0,001 sekon.

Analisis Fungsi solveMaze

Loop Utama:

```
while (current.x != end.x || current.y != end.y)
```

Loop ini berlanjut sampai posisi **current** mencapai posisi **end**. Dalam kasus terburuk, loop ini bisa

berjalan sebanyak jumlah total sel dalam maze, yaitu **rows * cols**.

Pemeriksaan Arah:

```
if (abs(dx) > abs(dy))
```

```
else
```

Di dalam loop utama, terdapat blok **if-else** yang memeriksa apakah pergerakan lebih ke arah horizontal (**dx**) atau vertikal (**dy**). Dalam setiap iterasi, satu sel akan ditandai, kecuali ketika menemui jalan buntu, di mana algoritma akan kembali ke jalur sebelumnya (**backtrack**).

Pengecekan Validitas:

```
if (isValid(...))
```

Fungsi **isValid** dipanggil beberapa kali untuk memastikan pergerakan tetap dalam batas-batas maze. Panggilan ini memiliki kompleksitas konstan, $O(1)$.

Penambahan ke Jalur dan Penghapusan dari Jalur:

```
path[pathLength++] = current;
```

Menambahkan sel ke jalur atau menghapus dari jalur dalam array **path** adalah operasi dengan kompleksitas konstan, $O(1)$.

Kompleksitas Waktu Total

Pencarian Jalur: Dalam kasus terburuk, algoritma akan mencoba setiap sel dalam maze, yang berarti kompleksitasnya adalah $O(\text{rows} * \text{cols})$. Ini mencakup skenario di mana algoritma perlu backtrack beberapa kali sebelum menemukan jalur atau menyadari bahwa tidak ada jalur yang valid.

Pemeriksaan dan Validasi: Meskipun ada beberapa pemeriksaan dan validasi dalam setiap iterasi, semuanya adalah operasi $O(1)$, sehingga tidak menambah kompleksitas total secara signifikan.

Kompleksitas waktunya adalah $O(\text{rows} * \text{cols})$

Di mana **rows** adalah jumlah baris dalam maze dan **cols** adalah jumlah kolom dalam maze. Ini karena dalam kasus terburuk, setiap sel dalam maze mungkin perlu diperiksa satu kali.

Dalam pengujian maze4.txt dan maze7.txt disini program mengalami infinite loop ketika

koordinatnya titik y sekarang sudah sama dengan end.y maka program akan memaksa hanya akan berjalan di sumbu x. Walaupun jika di dalam sumbu x memiliki '#'. Hal inilah yang menyebabkan program berada di dalam kondisi infinite loop.

Berdasarkan ke-7 algoritma yang telah diimplementasikan, dapat dilihat bahwa algoritma backtracking memiliki waktu runtime paling cepat dan selalu mengambil jalur terpendek untuk mencapai tujuan akhir dari koordinat awal sehingga algoritma ini paling efisien untuk digunakan. Algoritma dijkstra dan dynamic programming memiliki jumlah panjang jalur yang sama persis dan merupakan yang terpendek juga dari semua algoritma (mengambil jalur yang sama seperti backtracking), tetapi algoritma dynamic programming membutuhkan waktu sedikit lebih banyak dibandingkan dijkstra untuk menemukan jalurnya. BFS memiliki jumlah jalur yang berbeda sedikit (lebih panjang) dari backtracking, dijkstra dan dynamic programming, tetapi memiliki waktu yang relatif sama dan cukup cepat untuk setiap testcasenya.

Begitupun pada A* yang memilih jalur sedikit lebih banyak (biasanya kelebihan 1 jalur) dibandingkan ketiga algoritma yang dapat menemukan jalur paling pendek, tetapi algoritma ini bekerja lebih cepat dibandingkan dijkstra dan dynamic programming. Namun tidak lebih cepat dari BFS. Di lain sisi DFS juga mampu menemukan jalur terpendek, tetapi membutuhkan waktu yang sedikit lebih lama sampai di suatu saat terjadi infinite sama seperti yang terjadi pada algoritma greedy.

Maka dari itu dapat disimpulkan bahwa algoritma yang paling efisien untuk menemukan jalur

tercepat pada *maze problem* adalah algoritma backtracking.

KESIMPULAN

- a. Algoritma backtracking memiliki waktu runtime yang paling cepat dibandingkan algoritma lain, diikuti oleh BFS, kemudian A*, setelahnya dijkstra dan dynamic programming, kemudian DFS, dan terakhir greedy.
- b. Tingkat keakuratan dengan jalur terpendek dan tercepat dipunyai oleh algoritma backtracking. Oleh karena itu untuk menyelesaikan permasalahan *maze problem* lebih cocok digunakan algoritma backtracking.

DAFTAR PUSTAKA

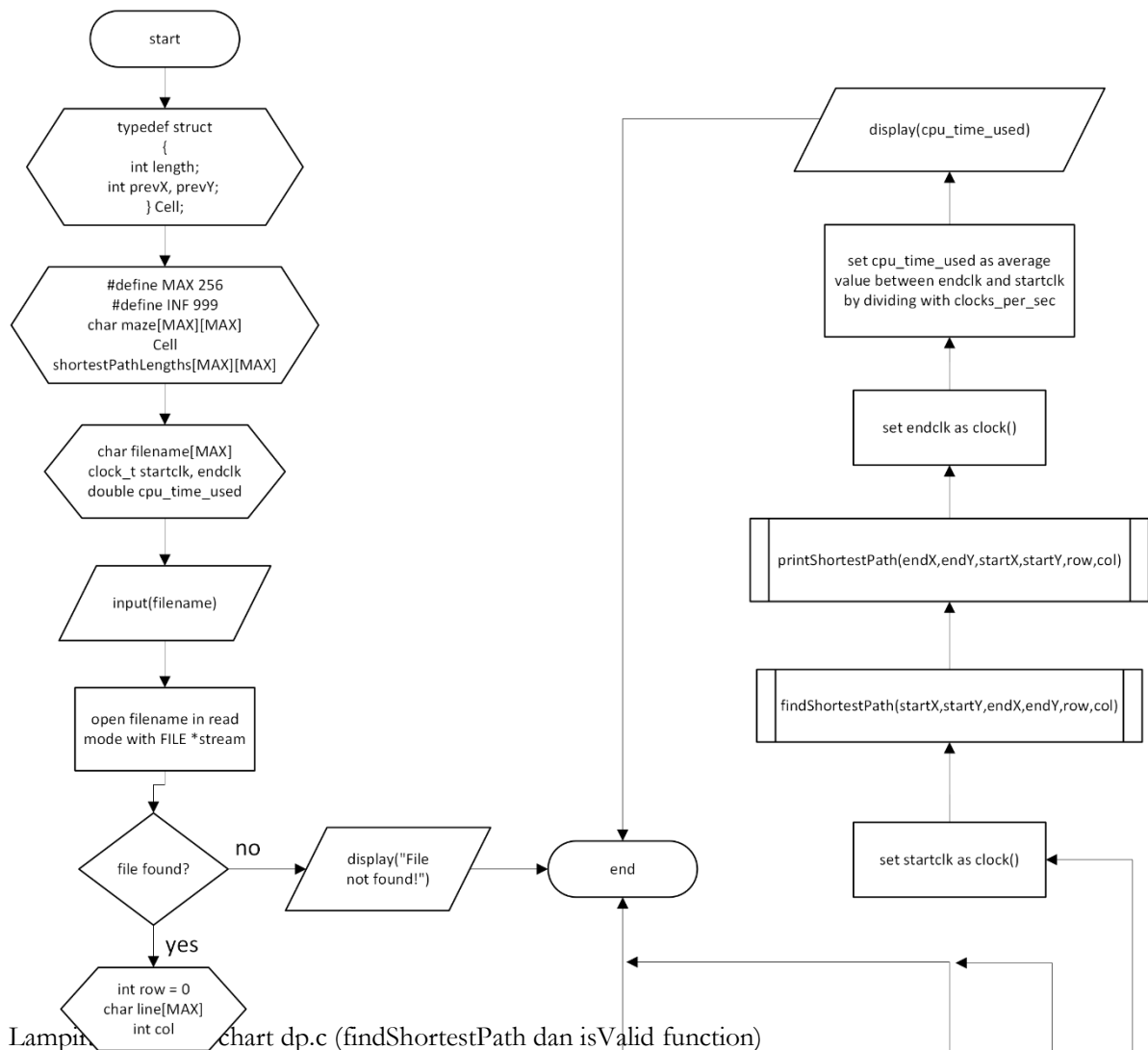
- [1] <https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/>, 16 Mei 2024, 22.30.
- [2] www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/, 16 Mei 2024, 22.30.
- [3] www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/, 16 Mei 2024, 22.30.
- [4] <https://www.geeksforgeeks.org/dynamic-programming/>, 16 Mei 2024, 22.30.
- [5] <https://www.geeksforgeeks.org/a-search-algorithm/>, 16 Mei 2024, 22.30.
- [6] <https://www.geeksforgeeks.org/introduction-to-backtracking-data-structure-and-algorithm-tutorials/#what-is-backtracking>, 16 Mei 2024, 22.30.
- [7] <https://www.geeksforgeeks.org/greedy-algorithms/#what-is-greedy-algorithm>, 16 Mei 2024, 22.30

LAMPIRAN

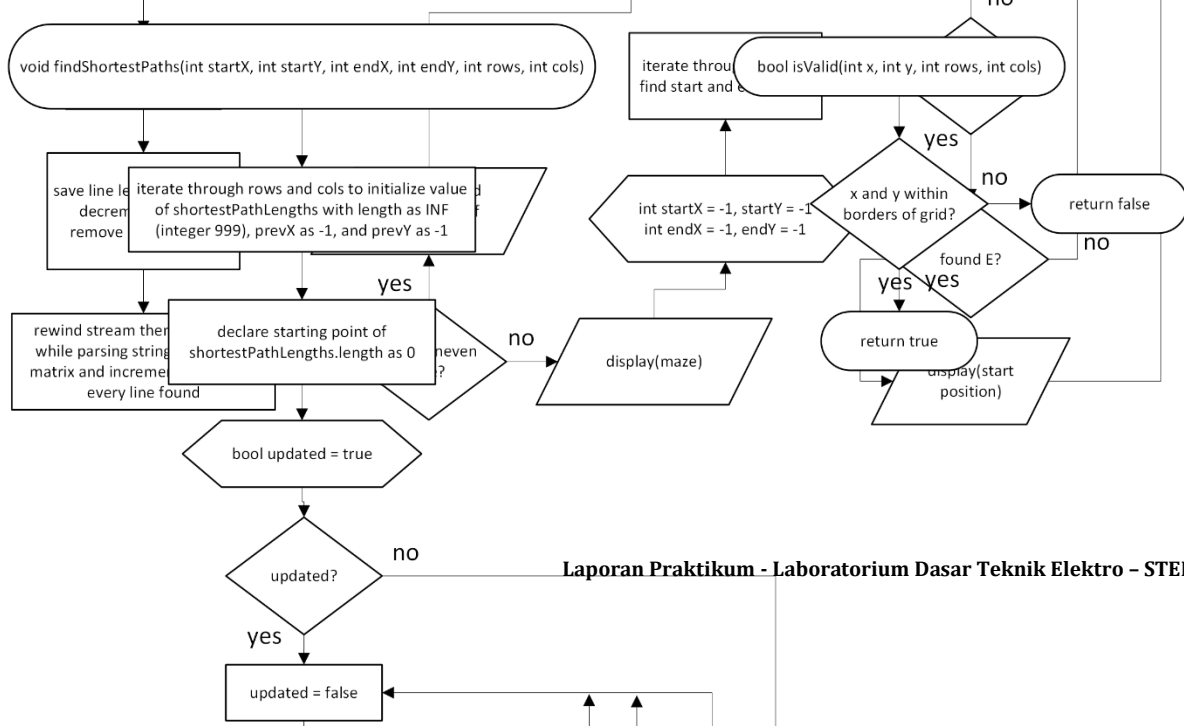
Repository Github :

https://github.com/gav193/TubesPPMC_E1

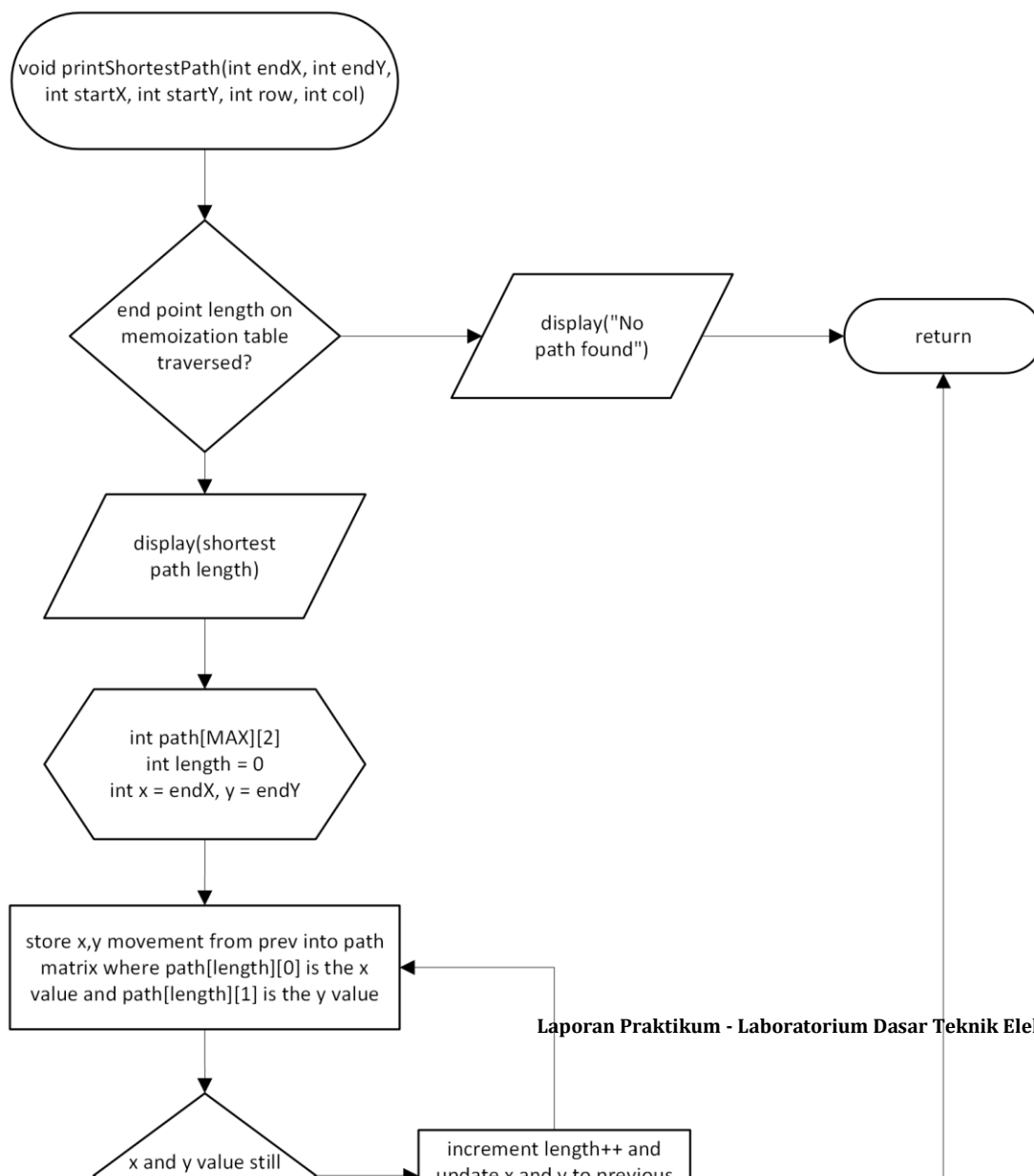
Lampiran 1 – Flowchart dp.c (main)



Lampiran 2 – Flowchart dp.c (findShortestPath dan isValid function)

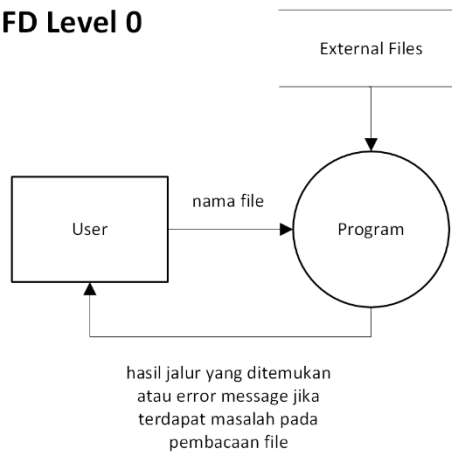


Lampiran 3 – Flowchart dp.c (printShortestPath function)

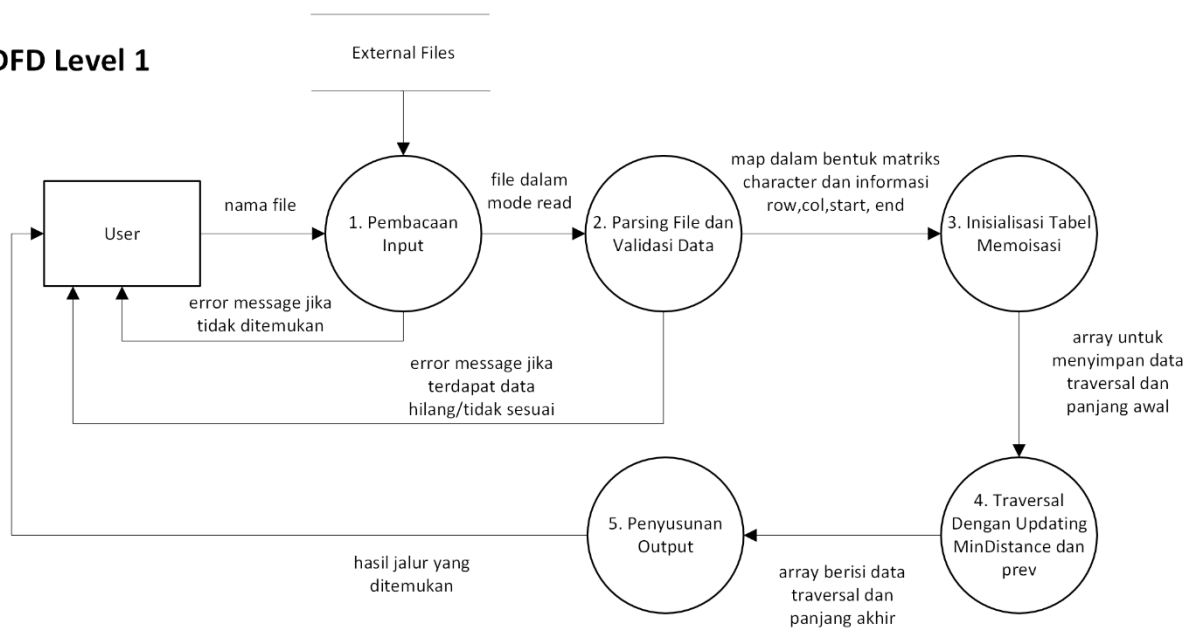


Lampiran 4 – DFD Level 0 dan 1 dp.c

DFD Level 0



DFD Level 1



Lampiran 5 – Hasil Pengujian maze1.txt (dynamic programming)

```

Maze contents:
S....#.#...
###.#.#...#
....#.#.#...
.#.....#.#
.#.#.....##
...#...#.#.
.#...#.#...#
.#.#.....
.#.#.#....
...#.#..E.#
.....##
Start position: (0, 0)
Shortest path length from 'S' to 'E': 18
Path :
(0, 0) -> (1, 0) -> (2, 0) -> (2, 1) -> (2, 2) -> (3, 2) -> (3, 3) -> (4, 3) -> (5, 3) -> (6, 3)
-> (6, 4) -> (7, 4) -> (8, 4) -> (9, 4) -> (9, 5) -> (9, 6) -> (9, 7) -> (9, 8) -> (9, 9)
Shortest path route :
S      *      *      .      #      .      .      #      .      .      .
#      #      *      #      .      #      .      #      .      .      #
.      .      *      *      #      .      #      .      #      .      .
.      #      .      *      *      *      *      #      .      #      #
.      #      .      #      .      .      *      *      *      *      #
.      #      .      #      .      .      .      #      .      *      #
.      #      .      #      .      .      .      .      .      *      .
.      #      .      #      .      .      .      #      .      *      .
.      .      .      .      #      .      #      .      .      E      #
.      .      .      .      .      .      .      .      .      #      #
Time taken: 0.006000 seconds
PS C:\pmc\test> █

```

Lampiran 6 – Hasil Pengujian maze2.txt (dynamic programming)

```

Maze contents:
....S.#.....
#...#.#.#.#.
#...#.#.#.#.
.....
#...#.#.#.#.
#.#...#.#.#..
....#.#....E
.....#...
.....##...
.#.....
Start position: (4, 0)
Shortest path length from 'S' to 'E': 13
Path :
(4, 0) -> (5, 0) -> (5, 1) -> (5, 2) -> (6, 2) -> (7, 2) -> (7, 3) -> (8, 3) -> (9, 3) -> (10, 3)
-> (11, 3) -> (11, 4) -> (11, 5) -> (11, 6)
Shortest path route :
.      .      .      .      S      *      #      .      .      .      .      .
#      .      .      .      #      *      #      .      #      .      #      .
#      .      .      .      #      *      *      .      #      .      #      .
.      .      .      .      .      .      .      *      *      *      *      *
#      .      .      .      #      .      .      .      .      .      #      *
#      .      #      .      .      .      #      .      #      #      .      *
.      .      .      .      #      .      #      .      .      .      .      E
.      .      .      .      .      .      .      .      #      .      .      .
.      .      .      .      .      #      #      .      .      #      .      .
.      #      .      .      .      .      .      .      .      .      .      .
Time taken: 0.010000 seconds

```

Lampiran 7 – Hasil Pengujian maze3.txt (dynamic programming)

```

Maze contents:
S...#.#...
##.#.#...#
###.#.#...
##...#####
##.#.....##
...#####.##
.#####.##.##
.#.#....#.#
.#.#.###.##
.#####E##
#.....##
Start position: (0, 0)
Shortest path length from 'S' to 'E': 18
Path :
(0, 0) -> (1, 0) -> (2, 0) -> (2, 1) -> (2, 2) -> (2, 3) -> (3, 3) -> (4, 3) -> (4, 4) -> (5, 4) -
> (6, 4) -> (7, 4) -> (8, 4) -> (9, 4) -> (9, 5) -> (9, 6) -> (9, 7) -> (9, 8) -> (9, 9)
Shortest path route :
S      *      *      .      .      #      .      .      #      .      .      .
#      #      *      #      .      #      .      #      .      .      .      #
#      #      *      #      #      .      #      .      #      .      .      .
#      #      *      *      *      #      #      #      #      #      #      #
#      #      .      #      *      *      *      *      *      *      #      #
.      .      .      #      #      #      #      #      #      *      #      #
.      #      #      #      #      #      .      #      #      *      #      #
.      #      .      #      .      .      .      .      #      *      #      #
.      #      .      #      .      #      .      #      #      *      #      #
.      #      #      #      #      #      #      #      #      E      #      #
#      .      .      .      .      .      .      .      .      .      #      #

Time taken: 0.008000 seconds

```

Lampiran 8 – Hasil pengujian maze4.txt (dynamic programming)

```

Maze contents:
...#S.#....
#...#.#.#.
#...#.#.#.
....#.#....
#####.#####
#.#...#.#
..#.#...##E
..#...#.#
..#...#.#...
.##.....
Start position: (4, 0)
Shortest path length from 'S' to 'E': 19
Path :
(4, 0) -> (5, 0) -> (5, 1) -> (5, 2) -> (5, 3) -> (5, 4) -> (5, 5) -> (5, 6) -> (5, 7) -> (5, 8) -
> (5, 9) -> (6, 9) -> (7, 9) -> (8, 9) -> (9, 9) -> (10, 9) -> (11, 9) -> (11, 8) -> (11, 7) -> (1
1, 6)
Shortest path route :
.      .      .      #      S      *      #      .      .      .      .      .
#      .      .      .      #      *      #      .      #      .      #      .
#      .      .      .      #      *      #      .      #      .      #      .
.      .      .      .      #      *      #      .      .      .      .      .
#      #      #      #      #      *      #      #      #      #      #      #
#      .      #      .      .      *      #      .      #      #      #      #
.      .      #      .      #      *      #      .      .      #      #      E
.      .      #      .      .      *      #      .      #      #      #      *
.      .      #      .      .      *      #      .      #      .      .      *
.      #      #      .      .      *      *      *      *      *      *      *

Time taken: 0.008000 seconds

```

Lampiran 9 – Hasil Pengujian maze5.txt (dynamic programming) : no path case

```

Maze contents:
S.....#.E
#####
.....
.....
.....
.....
.....
.....
Start position: (0, 0)
No path found from 'S' to 'E'.
Time taken: 0.000000 seconds

```

Lampiran 10 – Hasil Pengujian maze6.txt (dynamic programming) : uneven grid case

```

Enter file name: maze6.txt
Found uneven row of line at 7

```

Lampiran 11 – Hasil Pengujian maze7.txt (dynamic programming)

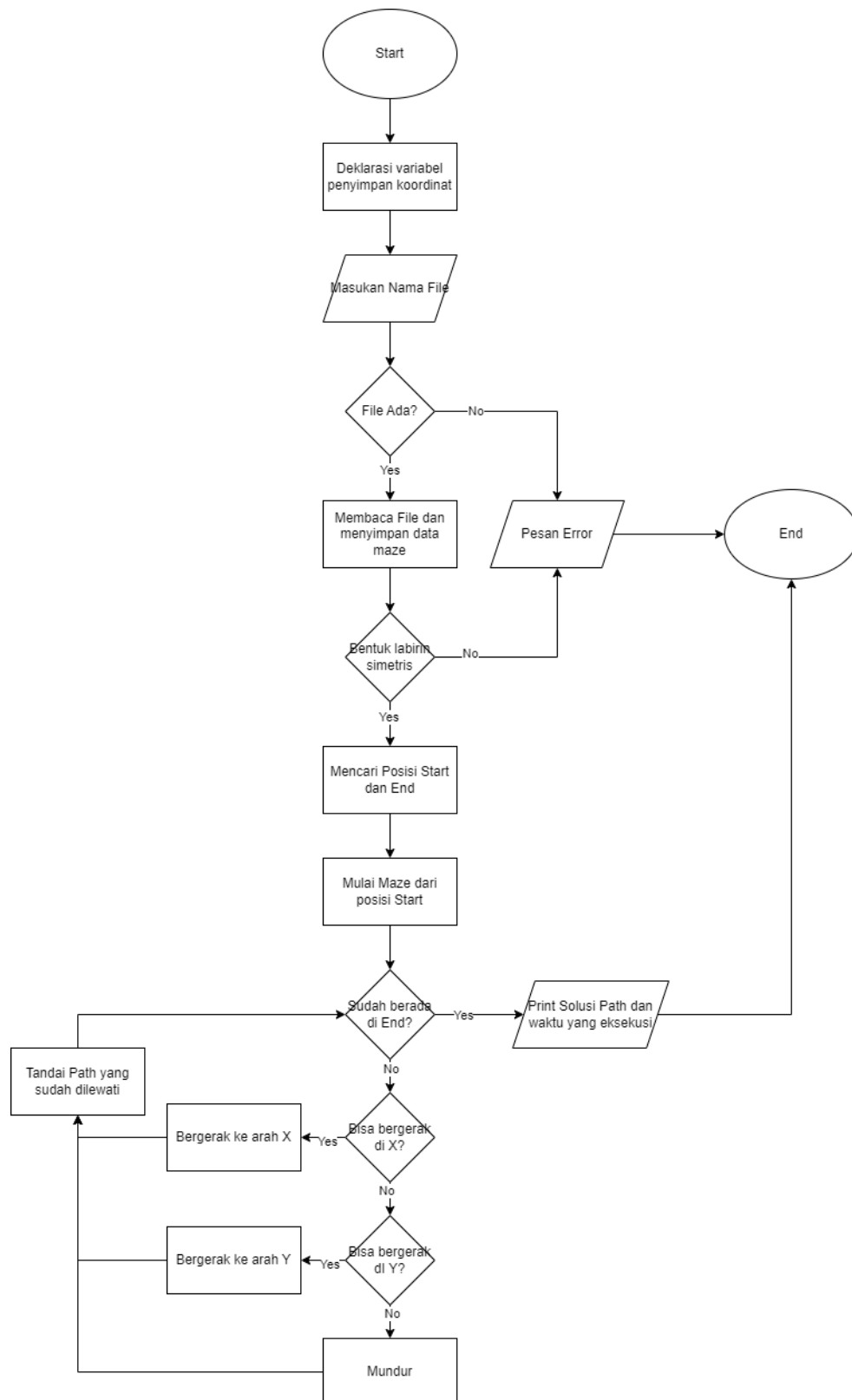
```

Maze contents:
S....#..E
.##..#.#.
..#..#.#.
..#.....
#####
..#.....
..#..###..
Start position: (0, 0)
Shortest path length from 'S' to 'E': 14
Path :
(0, 0) -> (1, 0) -> (2, 0) -> (3, 0) -> (4, 0) -> (4, 1) -> (4, 2) -> (4, 3) -> (5, 3) -> (6, 3) -
> (7, 3) -> (8, 3) -> (8, 2) -> (8, 1) -> (8, 0)
Shortest path route :
S      *      *      *      *      #      .      .      E
.      #      #      .      *      #      .      #      *
.      .      #      .      *      #      .      #      *
.      .      #      .      *      *      *      *      *
#      #      #      #      #      #      #      #      #
.      .      #      .      .      .      .      .      .
.      .      #      .      #      #      #      .      .

Time taken: 0.004000 seconds

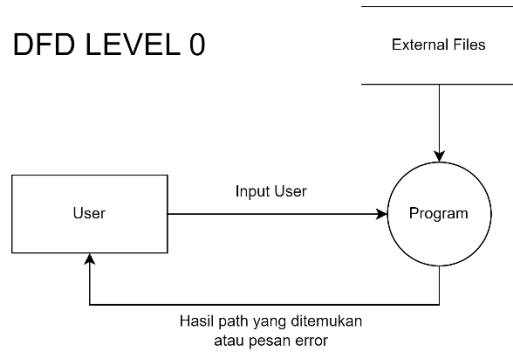
```

Lampiran 12 – Flowchart greedy.c

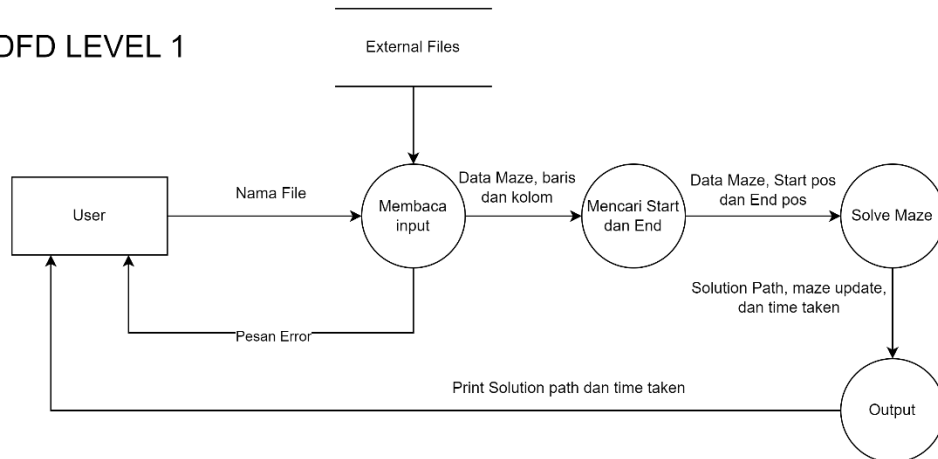


Lampiran 13 DFD greedy.c

DFD LEVEL 0

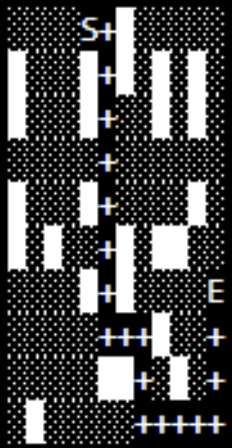
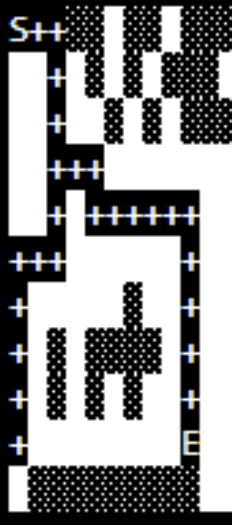

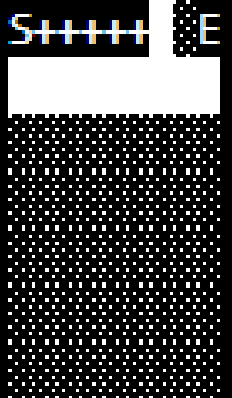


DFD LEVEL 1



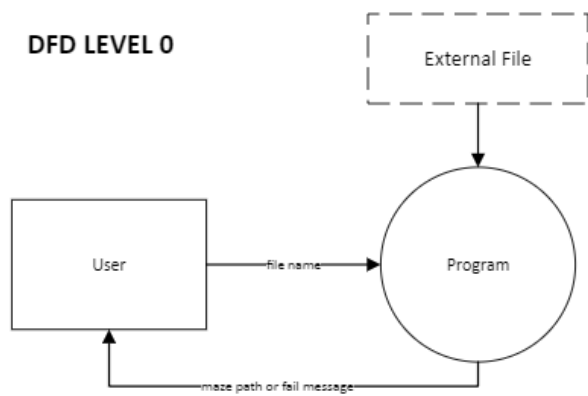
Lampiran 14 Hasil backtracking

Maze	
maze1	

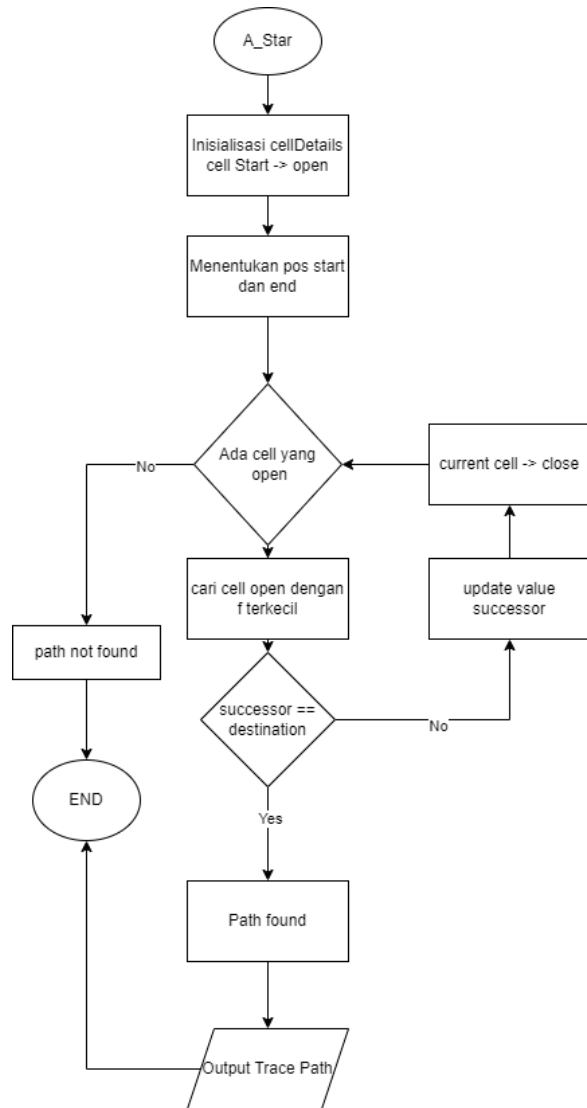
maze2		
maze3		
maze4		
maze5		

Maze 6-7	<pre> Enter file name: maze6.txt Found uneven row of line at 159744 , row PS C:\Users\62812\Documents\C program> .\tttt.exe Enter file name: maze7.txt </pre>
----------	--

Lampiran 16 – DFD backtracking



Lampiran 17 – DFD & Flowchart Algoritma A*



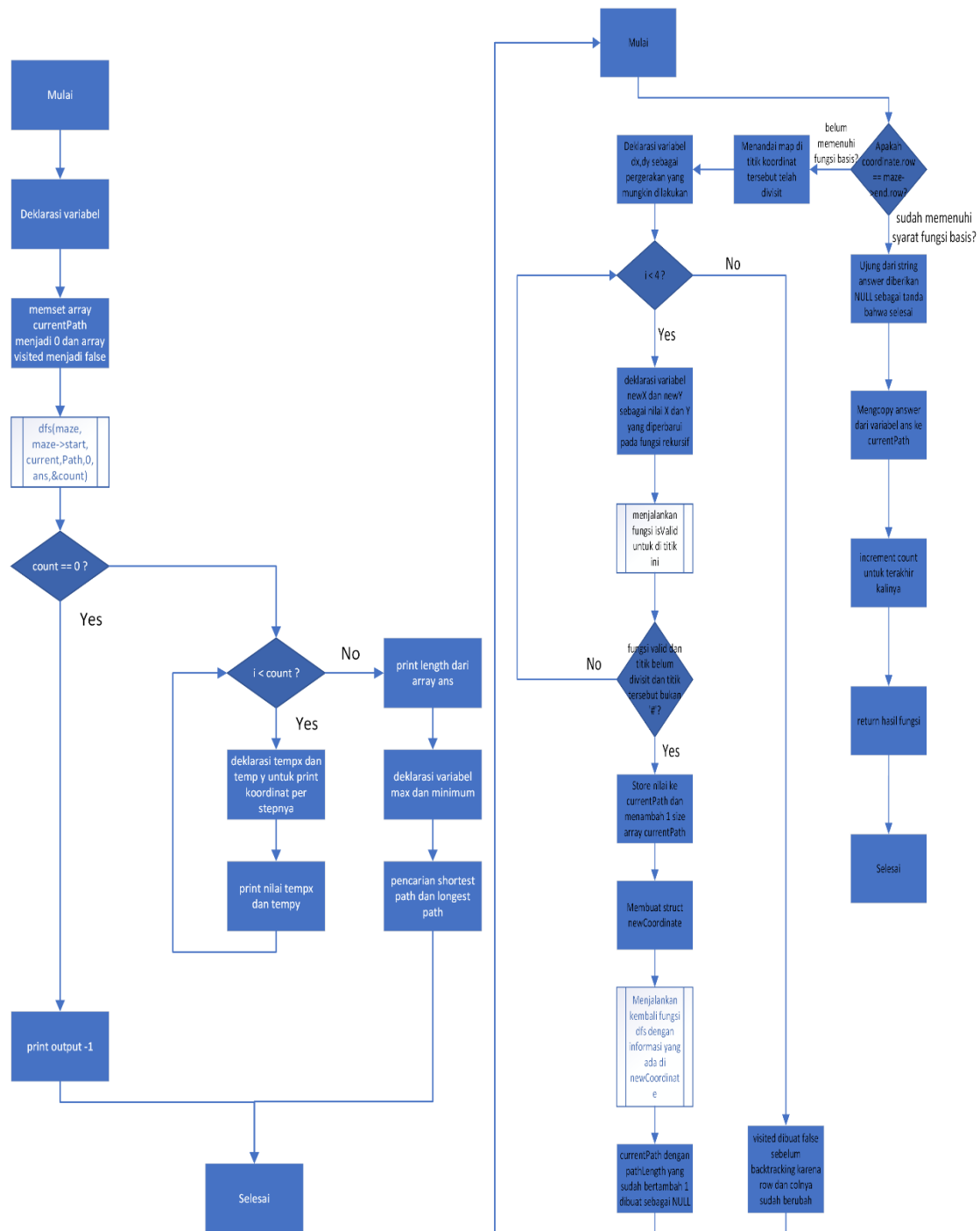
Lampiran 18 – Hasil Pengujian Algoritma A*

```

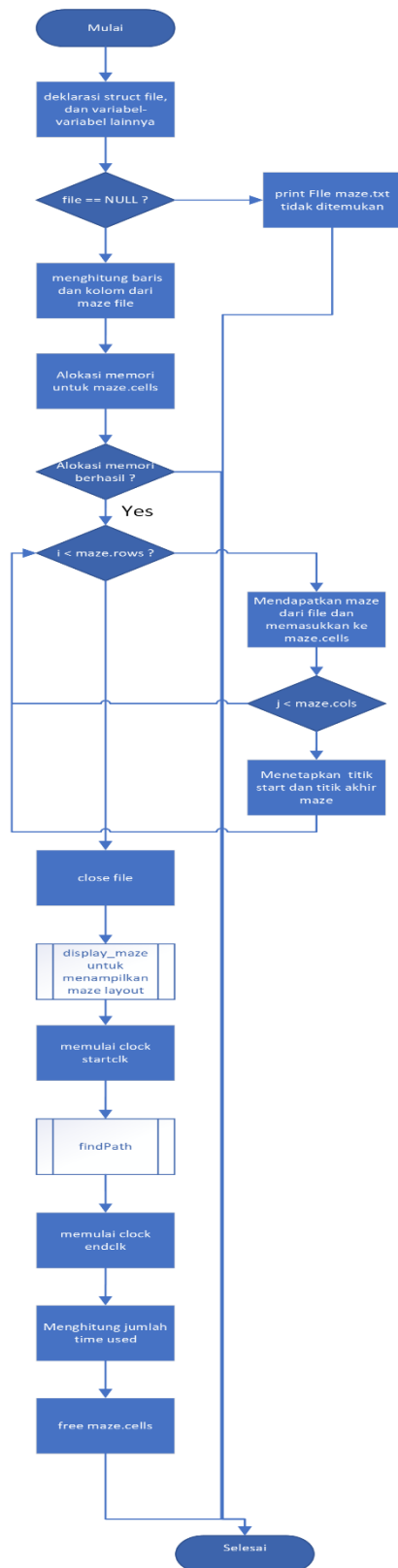
Enter file name: maze3.txt
MAP VALID
S: 0,0
E: 9,9
PATH FOUND
Shortest path length form 'S' to 'E' : 18
Traced Path:
S**#.#.#...
##*#.#.#.#
##*##.#.#...
##***#####
##.#*****##
...#####*##
.#####.#*##
.#.#...#*##
.#.#.#*##
.#####E##
#.....##
Time Taken : 0.004000 sec
PS C:\Users\rubin\Documents\Kullyeah\Sem 4\PMC\Tubes> .\a.exe
Enter file name: maze4.txt
MAP VALID
S: 4,0
E: 11,6
PATH FOUND
Shortest path length form 'S' to 'E' : 19
Traced Path:
...S*#....
#...#*#.#.#.
#...#*#.#.#.
...#*#....
#####*#####
#.#.#*#.####
..#.#*#..##E
..#...*#.####*
..#...#.#***
##..*****
Time Taken : 0.005000 sec

```

[illegible]

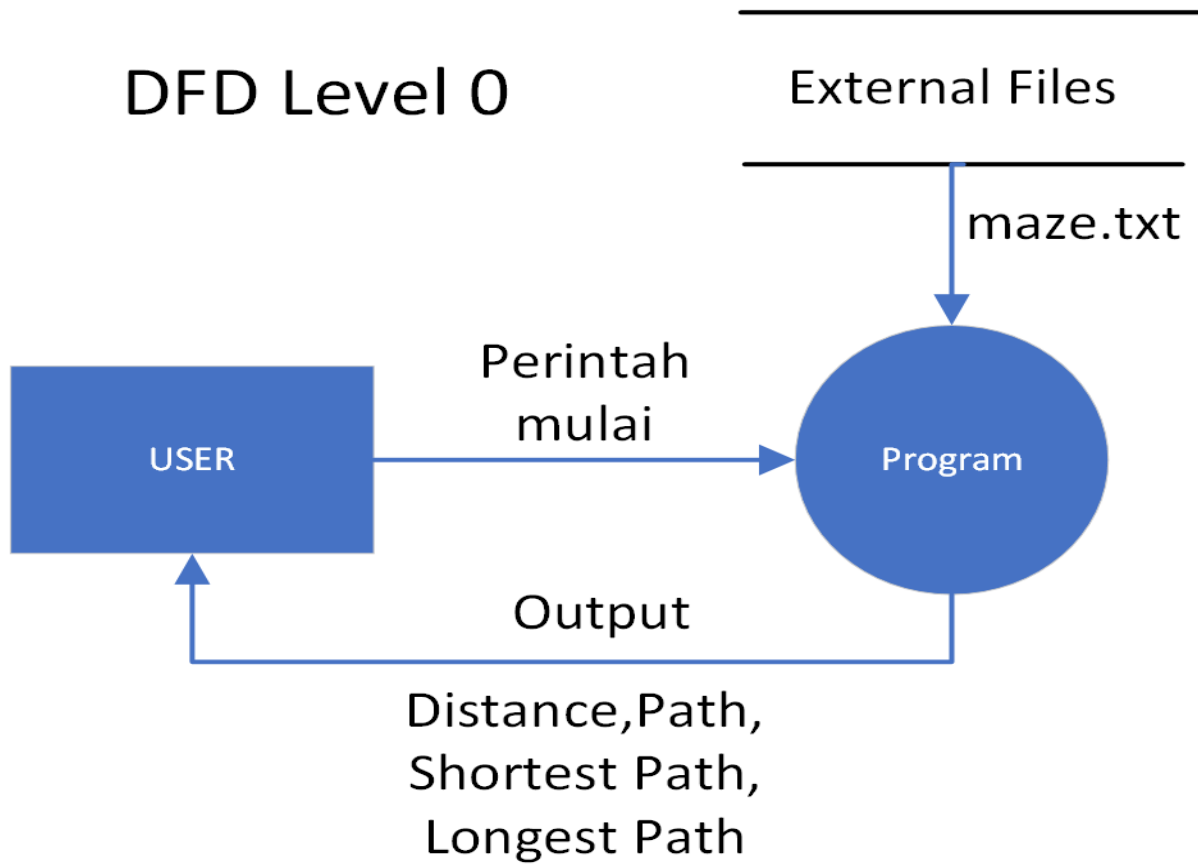


Flowchart fungsi main



Lampiran 20 – DFS DFD Level 0 dan Level 1

DFD Level 0



Lampiran 21 – Hasil Pengujian DFS

Maze1.txt

Maze2.txt

```
Starting path finding...
```

Maze6.txt

Maze7.txt

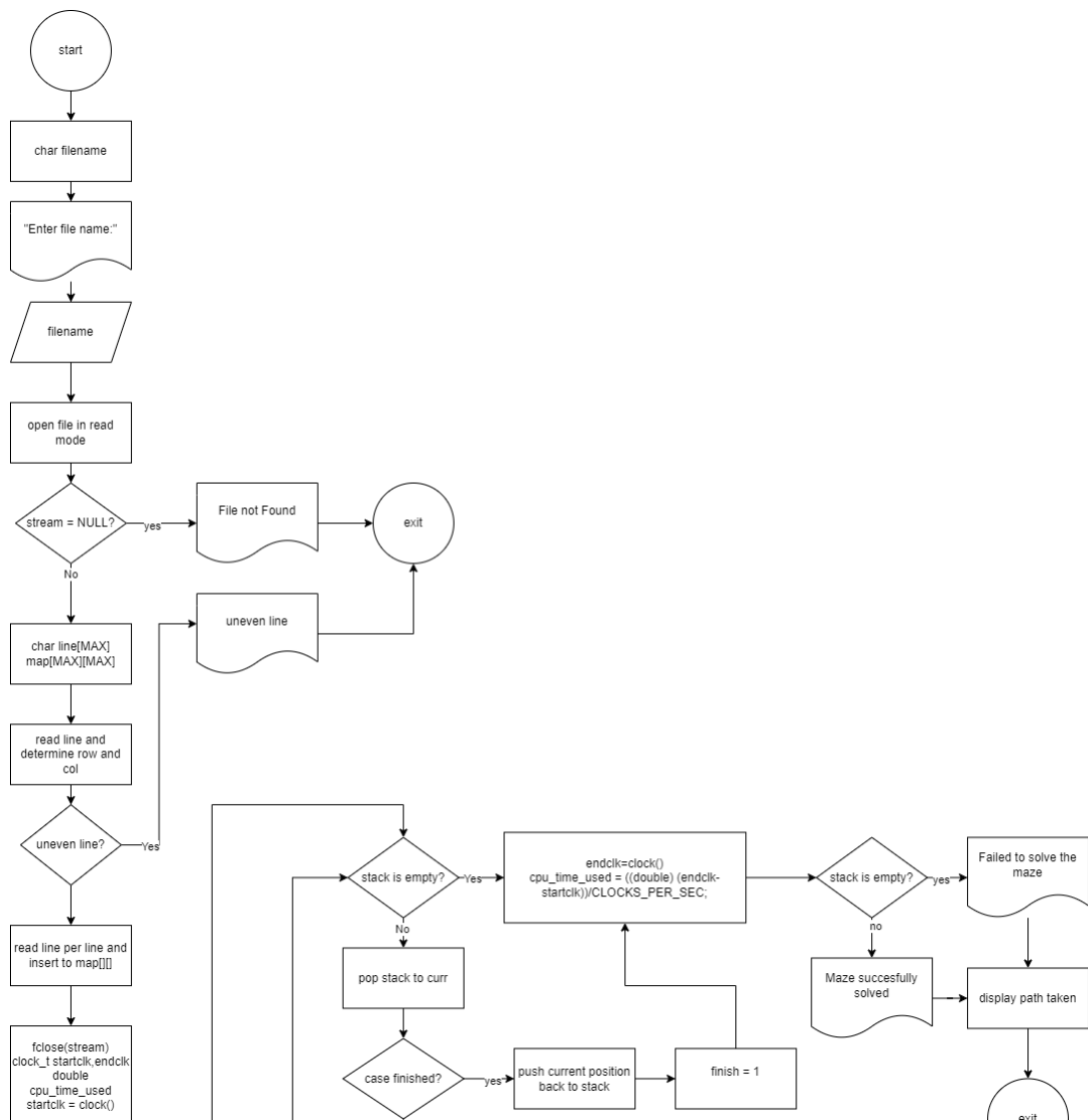
Maze8.txt


```

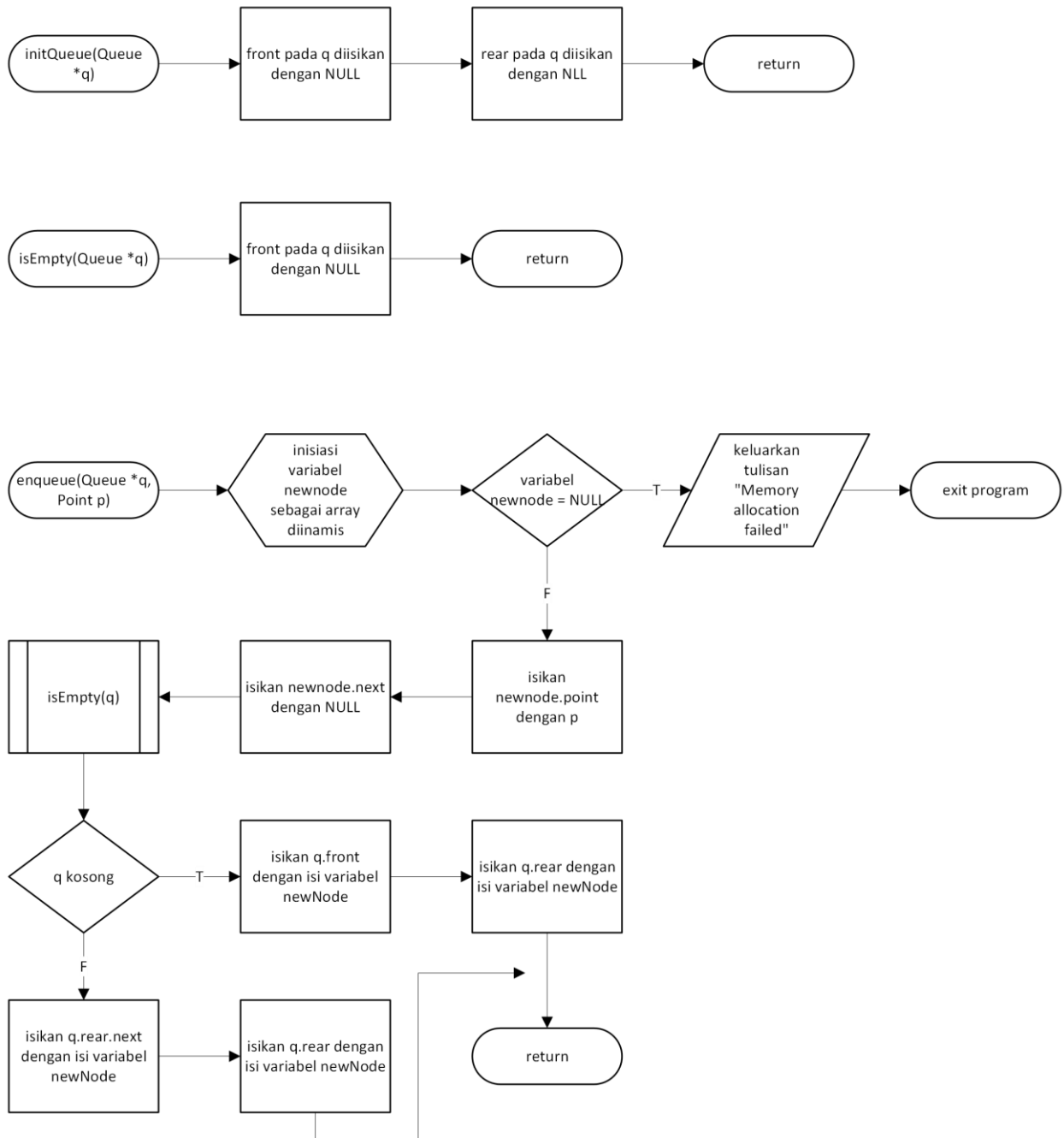
Total number of unique paths from 'S' to 'E': 16
Shortest path distance: 39
Path : (1, 0) -> (1,1) -> (2,1) -> (2,2) -> (2,3) -> (3,3) -> (4,3) -> (4,4) -> (4,5) -> (4,6) -> (4,7) -> (5,7) -> (6,7)
-> (6,6) -> (6,5) -> (7,5) -> (8,5) -> (8,6) -> (8,7) -> (8,8) -> (8,9) -> (9,9) -> (10,9) -> (11,9) -> (12,9) -> (13,9) -
-> (14,9) -> (14,10) -> (14,11) -> (14,12) -> (14,13) -> (14,14) -> (14,15) -> (14,16) -> (14,17) -> (15,17) -> (16,17) ->
(17,17) -> (17,18) -> (17,19)
Longest path distance: 79
Path : (1, 0) -> (1,1) -> (2,1) -> (2,2) -> (2,3) -> (3,3) -> (4,3) -> (4,4) -> (4,5) -> (4,6) -> (4,7) -> (5,7) -> (6,7)
-> (6,6) -> (6,5) -> (7,5) -> (8,5) -> (8,6) -> (8,7) -> (9,7) -> (10,7) -> (11,7) -> (12,7) -> (13,7) -> (14,7) -> (14,6)
-> (14,5) -> (14,4) -> (14,3) -> (13,3) -> (12,3) -> (12,2) -> (12,1) -> (13,1) -> (14,1) -> (15,1) -> (16,1) -> (16,2) -
-> (16,3) -> (16,4) -> (16,5) -> (16,6) -> (16,7) -> (16,8) -> (16,9) -> (17,9) -> (17,10) -> (17,11) -> (16,11) -> (16,12)
-> (16,13) -> (16,14) -> (16,15) -> (17,15) -> (17,16) -> (17,17) -> (16,17) -> (15,17) -> (14,17) -> (14,16) -> (14,15)
-> (14,14) -> (14,13) -> (14,12) -> (14,11) -> (13,11) -> (12,11) -> (12,12) -> (12,13) -> (12,14) -> (12,15) -> (12,16) -
-> (12,17) -> (12,18) -> (12,19) -> (13,19) -> (14,19) -> (15,19) -> (16,19) -> (17,19)
Waktu yang diperlukan: 0.223000

```

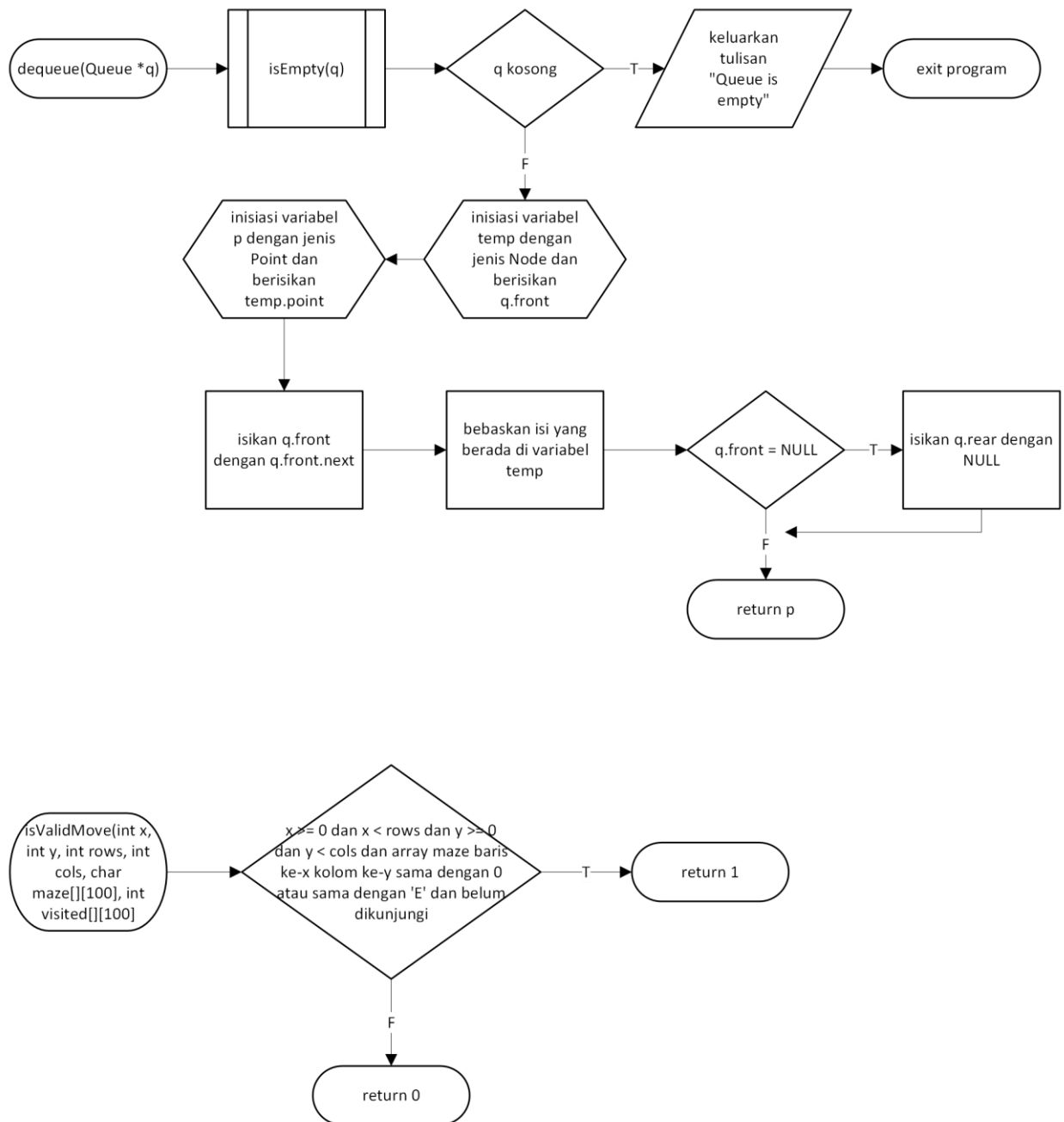
Lampiran 22 – Flowchart backtracking



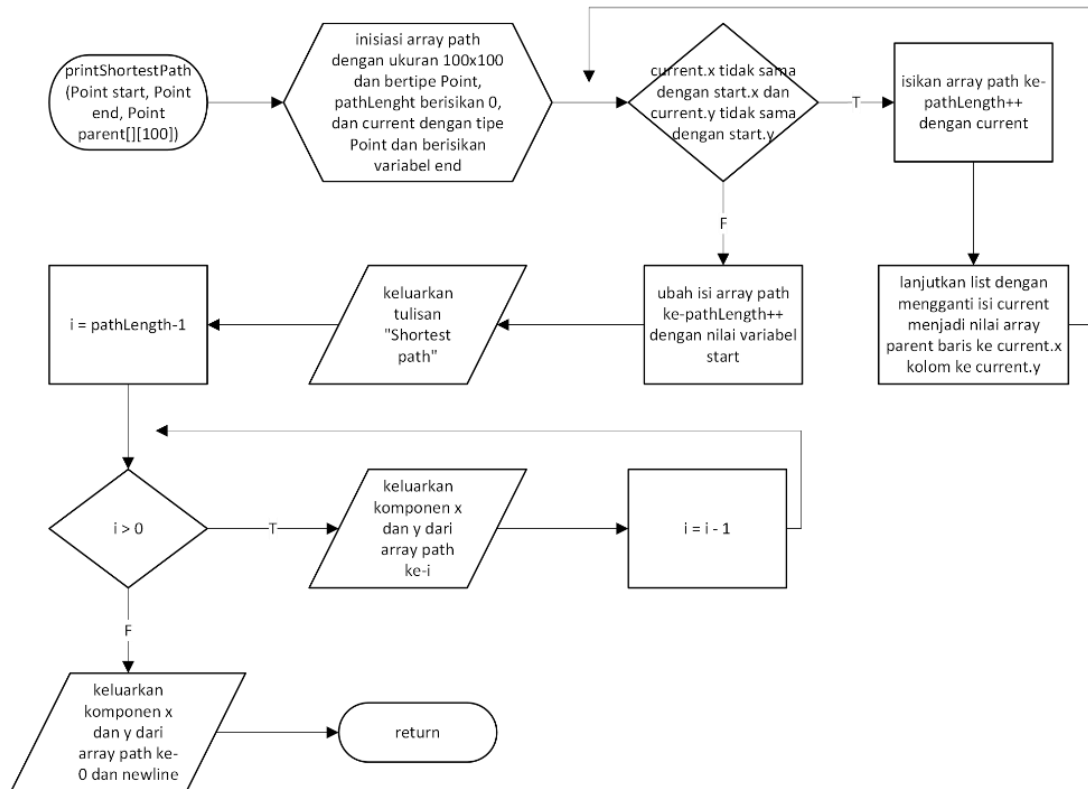
Lampiran 23 – Flowchart BFS (fungsi initQueue, isEmpty, enqueue)



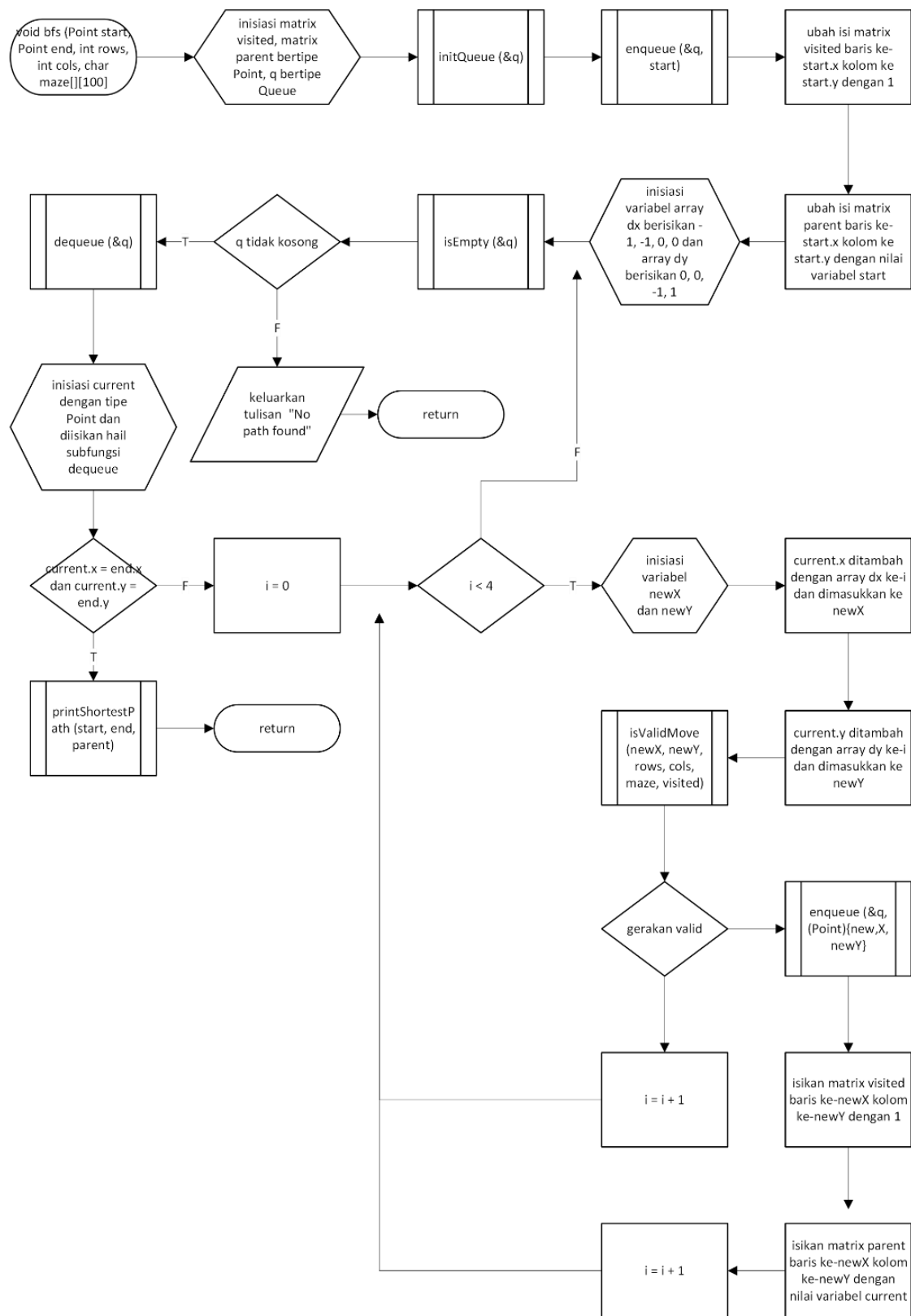
Lampiran 24 – Flowchart BFS (fungsi dequeue, isValidMove)



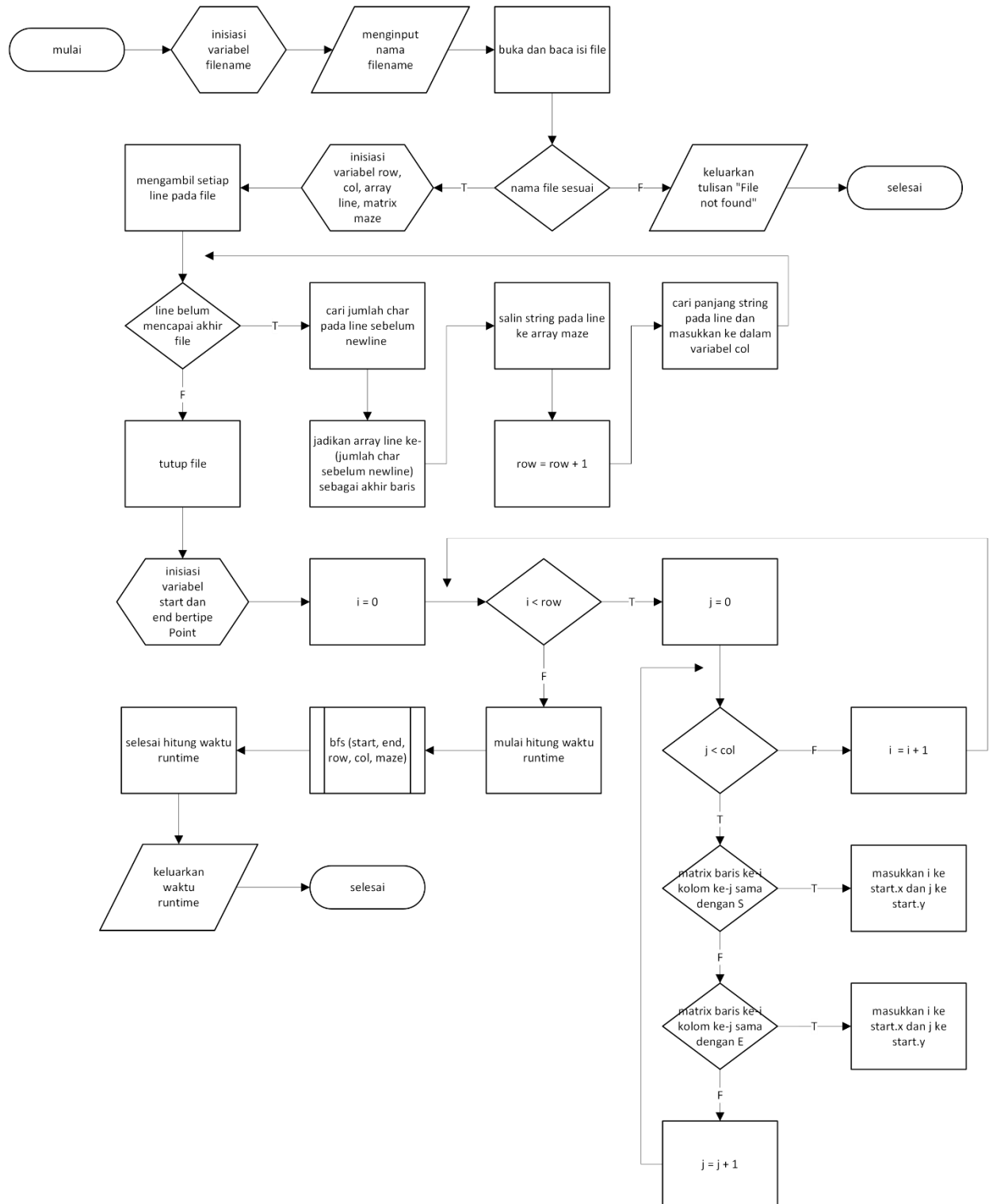
Lampiran 25 – Flowchart BFS (fungsi printShortestPath)



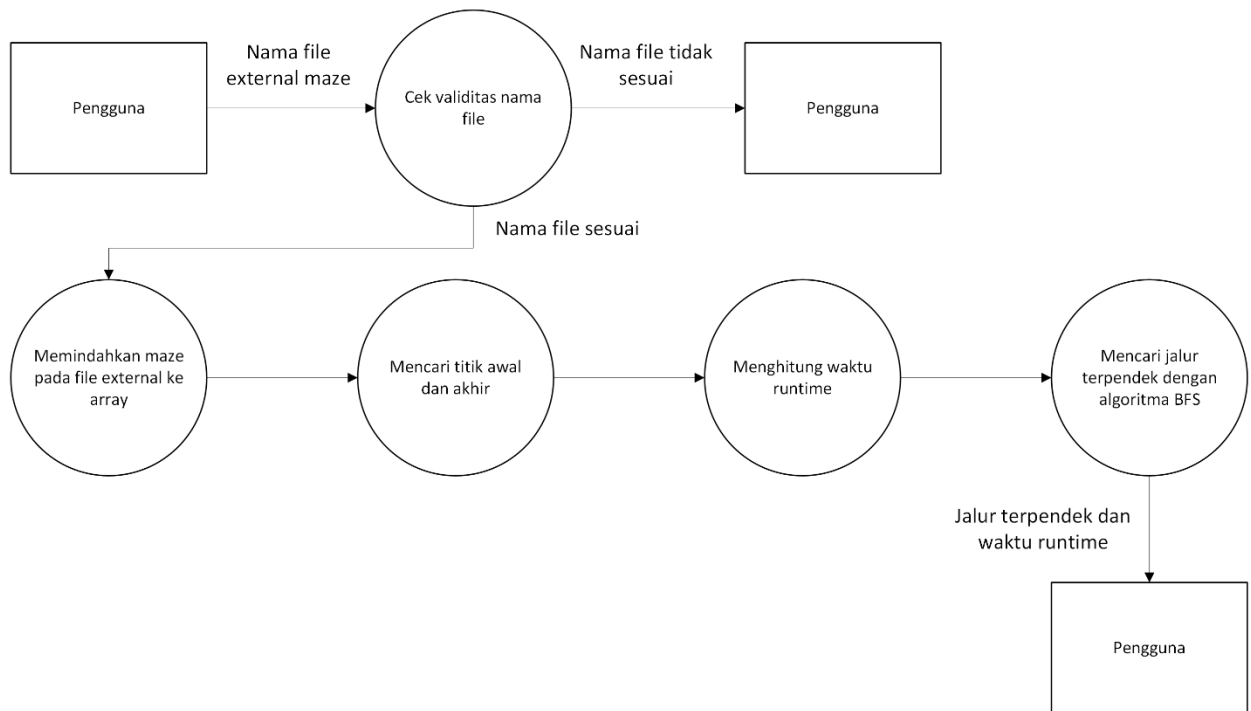
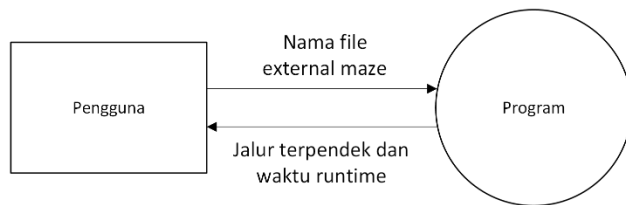
Lampiran 26 – Flowchart BFS (fungsi bfs)



Lampiran 27 – Flowchart BFS (fungsi utama)



Lampiran 28 – DFD BFS Level 0 dan 1



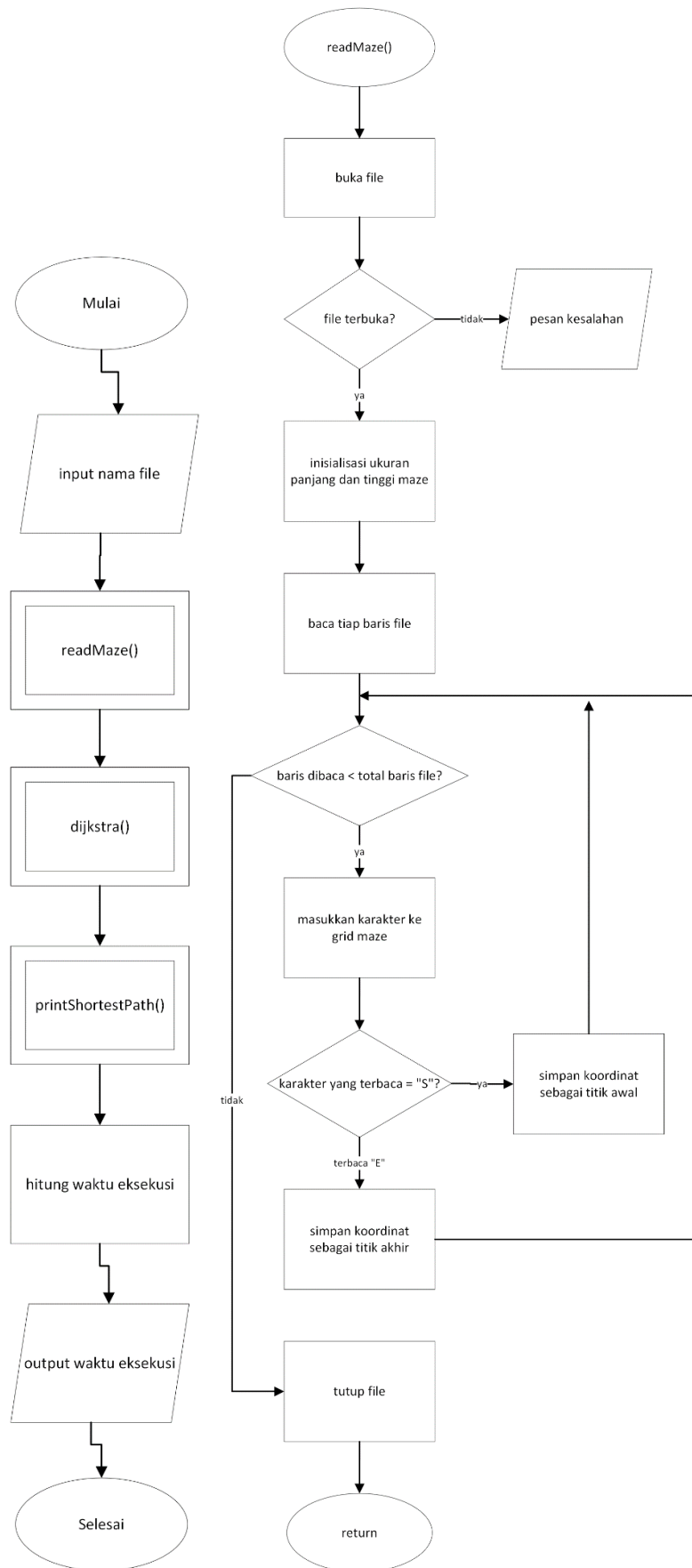
Lampiran 29 – Output BFS


```

PS C:\PMC> cd "c:\PMC\TubesPPMC\" ; if ($?) { gcc BFS.c -o BFS } ; if ($?) { .\BFS }
Enter file name: maze1.txt
Shortest path: (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (3, 2) -> (4, 2) -> (5, 2) -> (6, 2) -> (6, 3) -> (6, 4) -> (7, 4) -> (7, 5) -> (7, 6) -> (7, 7) -> (7, 8) -> (8, 8) -> (9, 8) -> (9, 9)
Waktu yang diperlukan: 0.001000
PS C:\PMC\TubesPPMC> cd "c:\PMC\TubesPPMC\" ; if ($?) { gcc BFS.c -o BFS } ; if ($?) { .\BFS }
Enter file name: maze2.txt
Shortest path: (0, 4) -> (0, 5) -> (1, 5) -> (2, 5) -> (3, 5) -> (4, 5) -> (4, 6) -> (4, 7) -> (5, 7) -> (6, 7) -> (6, 8) -> (6, 9) -> (6, 10) -> (6, 11)
Waktu yang diperlukan: 0.003000
PS C:\PMC\TubesPPMC> cd "c:\PMC\TubesPPMC\" ; if ($?) { gcc BFS.c -o BFS } ; if ($?) { .\BFS }
Enter file name: maze3.txt
Shortest path: (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (3, 2) -> (3, 3) -> (3, 4) -> (4, 4) -> (4, 5) -> (4, 6) -> (4, 7) -> (4, 8) -> (4, 9) -> (5, 9) -> (6, 9) -> (7, 9) -> (8, 9) -> (9, 9)
Waktu yang diperlukan: 0.001000
PS C:\PMC\TubesPPMC> cd "c:\PMC\TubesPPMC\" ; if ($?) { gcc BFS.c -o BFS } ; if ($?) { .\BFS }
Enter file name: maze4.txt
Shortest path: (0, 4) -> (0, 5) -> (1, 5) -> (2, 5) -> (3, 5) -> (4, 5) -> (5, 5) -> (6, 5) -> (7, 5) -> (8, 5) -> (9, 5) -> (9, 6) -> (9, 7) -> (9, 8) -> (9, 9) -> (8, 9) -> (8, 10) -> (8, 11) -> (7, 11) -> (6, 11)
Waktu yang diperlukan: 0.005000
PS C:\PMC\TubesPPMC> cd "c:\PMC\TubesPPMC\" ; if ($?) { gcc BFS.c -o BFS } ; if ($?) { .\BFS }
Enter file name: maze5.txt
No path found.
Waktu yang diperlukan: 0.001000
PS C:\PMC\TubesPPMC> cd "c:\PMC\TubesPPMC\" ; if ($?) { gcc BFS.c -o BFS } ; if ($?) { .\BFS }
Enter file name: maze6.txt
Found uneven row of line at 7
PS C:\PMC\TubesPPMC> cd "c:\PMC\TubesPPMC\" ; if ($?) { gcc BFS.c -o BFS } ; if ($?) { .\BFS }
Enter file name: maze7.txt
Shortest path: (0, 0) -> (0, 1) -> (0, 2) -> (0, 3) -> (1, 3) -> (2, 3) -> (3, 3) -> (3, 4) -> (3, 5) -> (3, 6) -> (2, 6) -> (1, 6) -> (0, 6) -> (0, 7) -> (0, 8)
Waktu yang diperlukan: 0.003000
PS C:\PMC\TubesPPMC> cd "c:\PMC\TubesPPMC\" ; if ($?) { gcc BFS.c -o BFS } ; if ($?) { .\BFS }
Enter file name: MAZE.TXT
File not found!
PS C:\PMC\TubesPPMC> █

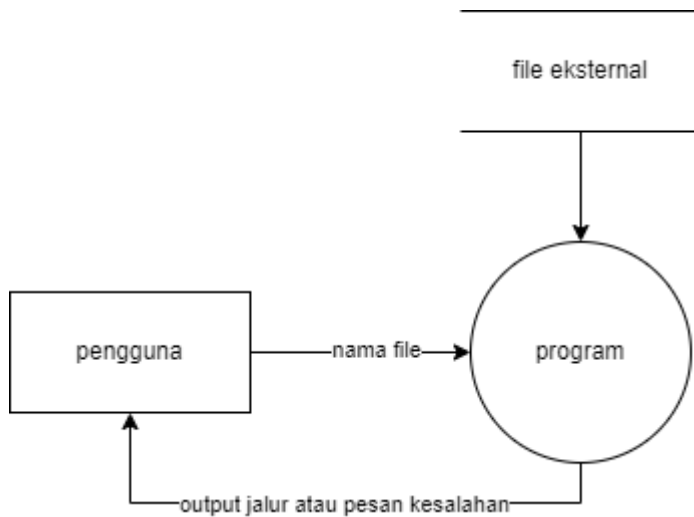
```

Lampiran 30 – Flowchart Dijkstra

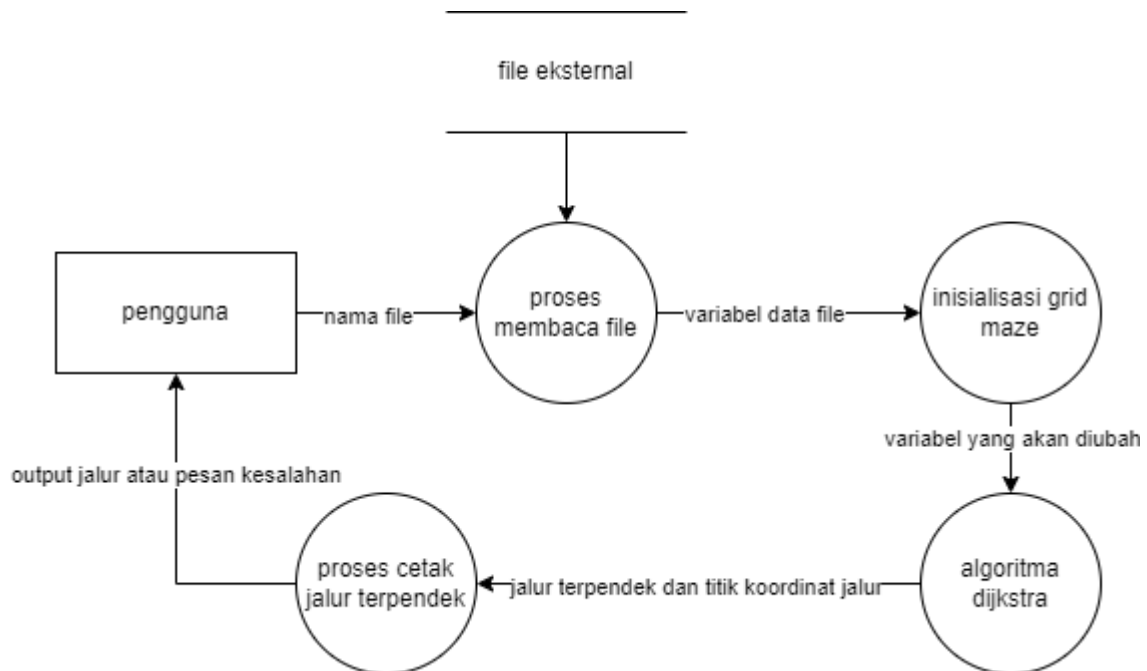




Lampiran 31 – DFD Dijkstra Level 0



Lampiran 32 – DFD Dijkstra Level 1



Lampiran 33 – Output Dijkstra

```

PS C:\Users\Wusaiba El Qonitat> cd "d:\PMC\PPMC\Tubes\" ; if ($?) { gcc dijkstra.c -o dijkstra } ; if ($?) { .\dijkstra }
Masukkan File Txt Struktur Maze : maze1.txt
Shortest path distance: 18
Path: (0, 0) -> (1, 0) -> (2, 0) -> (2, 1) -> (2, 2) -> (3, 2) -> (3, 3) -> (4, 3) -> (5, 3) -> (6, 3) -> (6, 4) -> (7, 4) -> (8, 4) -> (9, 4) -> (9, 5) -> (9, 6) -> (9, 7) -> (9, 8) -> (9, 9)
Waktu yang diperlukan: 0.005000
PS D:\PMC\PPMC\Tubes> cd "d:\PMC\PPMC\Tubes\" ; if ($?) { gcc dijkstra.c -o dijkstra } ; if ($?) { .\dijkstra }
Masukkan File Txt Struktur Maze : maze2.txt
Shortest path distance: 13
Path: (4, 0) -> (5, 0) -> (5, 1) -> (5, 2) -> (6, 2) -> (7, 2) -> (7, 3) -> (8, 3) -> (9, 3) -> (10, 3) -> (11, 3) -> (11, 4) -> (11, 5) -> (11, 6)
Waktu yang diperlukan: 0.007000
PS D:\PMC\PPMC\Tubes> cd "d:\PMC\PPMC\Tubes\" ; if ($?) { gcc dijkstra.c -o dijkstra } ; if ($?) { .\dijkstra }
Masukkan File Txt Struktur Maze : maze3.txt
Shortest path distance: 18
Path: (0, 0) -> (1, 0) -> (2, 0) -> (2, 1) -> (2, 2) -> (2, 3) -> (3, 3) -> (4, 3) -> (4, 4) -> (5, 4) -> (6, 4) -> (7, 4) -> (8, 4) -> (9, 4) -> (9, 5) -> (9, 6) -> (9, 7) -> (9, 8) -> (9, 9)
Waktu yang diperlukan: 0.000000
PS D:\PMC\PPMC\Tubes> cd "d:\PMC\PPMC\Tubes\" ; if ($?) { gcc dijkstra.c -o dijkstra } ; if ($?) { .\dijkstra }
Masukkan File Txt Struktur Maze : maze4.txt
Shortest path distance: 19
Path: (4, 0) -> (5, 0) -> (5, 1) -> (5, 2) -> (5, 3) -> (5, 4) -> (5, 5) -> (5, 6) -> (5, 7) -> (5, 8) -> (5, 9) -> (6, 9) -> (7, 9) -> (8, 9) -> (9, 9) -> (9, 8) -> (10, 8) -> (11, 8) -> (11, 7) -> (11, 6)
Waktu yang diperlukan: 0.005000
PS D:\PMC\PPMC\Tubes> cd "d:\PMC\PPMC\Tubes\" ; if ($?) { gcc dijkstra.c -o dijkstra } ; if ($?) { .\dijkstra }
Masukkan File Txt Struktur Maze : maze5.txt
No path found.
Waktu yang diperlukan: 0.000000
PS D:\PMC\PPMC\Tubes> cd "d:\PMC\PPMC\Tubes\" ; if ($?) { gcc dijkstra.c -o dijkstra } ; if ($?) { .\dijkstra }
Masukkan File Txt Struktur Maze : maze6.txt
Kolom maze memiliki panjang yang berbeda!
PS D:\PMC\PPMC\Tubes> cd "d:\PMC\PPMC\Tubes\" ; if ($?) { gcc dijkstra.c -o dijkstra } ; if ($?) { .\dijkstra }
Masukkan File Txt Struktur Maze : maze7.txt
Shortest path distance: 14
Path: (0, 0) -> (1, 0) -> (2, 0) -> (3, 0) -> (4, 0) -> (4, 1) -> (4, 2) -> (4, 3) -> (5, 3) -> (6, 3) -> (6, 2) -> (6, 1) -> (6, 0) -> (7, 0) -> (8, 0)
Waktu yang diperlukan: 0.006000
PS D:\PMC\PPMC\Tubes> █

```