MODUL 9 TUGAS BESAR

Wahyu Putra Almaya (13222101)
Poggy Macello Gultom (13222102)
Aldeo Malta Junior (13222103)
Avila Khadhibyan (13222104)
Didan Attaric (13222105)
Jason Theo Salim (13222106)
Fairuz Apuilla Rahagi (13222107)
Asisten: Agape D'Sky
Kelompok: E2
EL2208-Praktikum Pemecahan Masalah dengan C
Laboratorium Dasar Teknik Elektro - Sekolah Teknik Elektro dan Informatika ITB

Abstrak

Tugas besar EL2208 PPMC mengharuskan penyelesaian salah satu dari dua soal yang diberikan, dimana setiap pengerjaan setiap soal harus dilakukan dengan beberapa algoritma. Soal yang dipilih pada tugas besar kali ini adalah soal navigasi maze dari suatu titik awal ke titik akhir, dimana praktikan diharuskan untuk mencari jalur terpendek, terpanjang, dan juga semua jalur yang mungkin dari titik awal ke titik akhir. Pengerjaan dilakukan dengan beberapa algoritma, dan hasil dari tiap-tiap algoritma ini kemudian akan dibandingkan dan dianalisis untuk menentukan karakteristik-karakteristik tiap algoritma dan perbandingan performanya dengan algoritma lain. Tugas besar ini dilakukan secara berkelompok, dengan setiap anggota kelompok memilih satu algoritma untuk diimplementasikan. Hasil yang diperoleh adalah bahwa tiap algoritma memiliki keuntungan dan kerugian masing-masing. Namun algoritma yang paling efektif secara keseluruhan adalah algoritma A.*

Kata kunci: Algoritma, maze problem, struktur data

1. PENDAHULUAN

Tugas Besar praktikum Pemecahan Masalah dengan C ini mewajibkan praktikan memilih dan mengerjakan salah satu dari soal. Soal yang dipilih kelompok ini adalah soal nomor 1. Dalam soal ini, harus dirancang sebuah program yang dapat menavigasi sebuah file (.txt) maze dengan ketentuan-ketentuan tertentu yang sudah diberikan dan akan dibahas lebih lanjut di ruang lingkup masalah.

Pembuatan program untuk menyelesaikan soal 1 ini juga mewajibkan penggunaan beberapa algoritma. Berikut adalah algoritma-algoritma yang akan digunakan:

- Algoritma Djikstra
- Breadth First Search (BFS)
- Depth First Search (DFS)
- Dynamic Programming
- Algoritma A* (A-Star)
- Algoritma Backtracking
- Algoritma Greedy

Dengan memanfaatkan algoritma-algoritma yang berbeda untuk menyelesaikan soal ini, dapat dilakukan perbandingan dan analisis tiap-tiap algoritma sehingga dapat disimpulkan karakteristik tiap algoritma yang digunakan, dan algoritma yang paling efektif untuk pemecehan kasus soal ini.

2. STUDI PUSTAKA

2.1 ALGORITMA DIJKSTRA

Algortima Dijkstra merupakan algoritma yang dapat mencari jalur terpendek pada graf dengan beban. Algoritma ini pertama kali ditemukan oleh Edger W. Dijkstra pada tahun 1956. Satu vertices akan dipilih sebagai titik sumber, lalu semua vertex lainnya akan di ubah menjadi tidak terhingga. Dari titik sumber akan dipilih vertex selanjutnya yang memiliki graf dengan beban terkecil. Proses ini akan diulang hingga mencapai vertex tujuan [1].

Dalam implementasi untuk memecahkan maze pada Bahasa c, setiap sel dalam maze akan ditetapkan sebagai vertices. Kemudian program akan mengambil jarak terdekat dari vertex hingga mencapai tujuan yang diinginkan.

2.2 BREADTH FIRST SEARCH (BFS)

Breadth-first search (BFS) merupakan sebuah algoritma untuk mencari struktur data pohon untuk sebuah simpul yang memenuhi properti tertentu. Implementasi algoritma ini dimulai dari akar pohon dan menjelajahi semua node pada kedalaman saat ini sebelum melanjutkan ke node

pada tingkat kedalaman berikutnya. Memori tambahan, biasanya berupa antrian, diperlukan untuk melacak noda-noda anak yang ditemukan tetapi belum dijelajahi.

Penelusuran *breadth first* dapat digeneralisasikan menjadi graf tidak berarah dan graf berarah dengan simpul awal tertentu (terkadang disebut sebagai 'kunci penelusuran'). Dalam pencarian *state space* dalam kecerdasan buatan, pencarian simpul berulang kali sering kali diperbolehkan, sedangkan dalam analisis teoretis algoritma berdasarkan pencarian luas pertama, tindakan pencegahan biasanya diambil untuk mencegah pengulangan.

2.3 DEPTH FIRST SEARCH (DFS)

Algoritma *Depth First Search* (DFS) adalah suatu metode pencarian pada sebuah tree/pohon dengan menelusuri satu cabang sebuah tree sampai menemukan solusi. Pencarian dilakukan pada satu node dalam setiap level dari yang paling kiri dan dilanjutkan pada node sebelah kanan. Jika solusi ditemukan maka tidak diperlukan proses *backtracking* yaitu penelusuran balik untuk mendapatkan jalur yang diinginkan.

2.4 DYNAMIC PROGRAMMING

Dynamic Programming adalah metode pemrograman algoritmik yang menyelesaikan masalah kompleks dengan cara mendekomposisikan masalah tersebut. Sub-masalah yang dibentuk kemudian dipecahkan secara terpisah, dan setiap solsui sub-masalah ini disimpan dalam sebuah tabel atau array [2]. Hal ini dilakukan untuk memastikan bahwa setiap sub-masalah hanya akan ditemukan solusinya sebanyak satu kali. Solusi-solusi dari sub-masalah ini kemudian akan digabung kembali untuk menemukan solusi utama dari program.

2.5 ALGORTIMA A* (A-STAR)

Algoritma A* adalah algoritma pencarian jalan terdekat yang mirip dengan algoritma Dijkstra. Algoritma ini mengenalkan tambahan heuristik pada perhitungannya. Dengan begitu, ia dapat mengukur jalan yang lebih optimal.

Algoritma ini menghitung nilai sebuah jalan dengan mengukur jarak dari titik awal serta jarak ke titik akhir. Pemilihan heuristic sangat penting untuk performa algoritma ini. Idealnya, nilai heuristik sama dengan jarak suatu titik ke titik akhir. Namun, kita tidak mengetahui jalan ke titik akhir tersebut sehingga kita juga tidak mengetahui jaraknya.

2.6 ALGORITMA BACKTRACKING

Algoritma backtracking adalah metode pencarian solusi dengan mencoba semua kemungkinan solusi satu per satu dan mundur ketika solusi yang sedang dicoba ternyata tidak valid. Dalam konteks mencari jalan di labirin, algoritma ini bekerja dengan mencoba bergerak ke setiap arah yang mungkin (kanan, bawah, kiri, atas) dari posisi saat ini dan menandai jalur yang sudah dicoba untuk menghindari pengulangan.

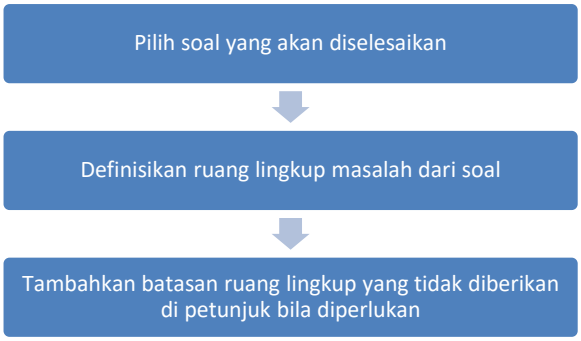
2.7 ALGORITMA GREEDY

Algoritma Greedy adalah pendekatan dalam pemrograman yang berfokus pada pengambilan keputusan lokal yang optimal pada setiap langkah, dengan harapan akan mencapai solusi global yang optimal. Pada setiap langkah, algoritma ini memilih opsi yang paling menguntungkan secara instan, tanpa mempertimbangkan konsekuensi jangka panjang. Dalam implementasinya, algoritma greedy sering digunakan dalam permasalahan optimisasi di mana solusi optimal dapat dicapai dengan memilih opsi terbaik pada setiap langkah. Keunggulan algoritma ini terletak pada kecepatan eksekusi dan kesederhanaan konsep, namun demikian, tidak selalu menghasilkan solusi global yang optimal dalam semua kasus.

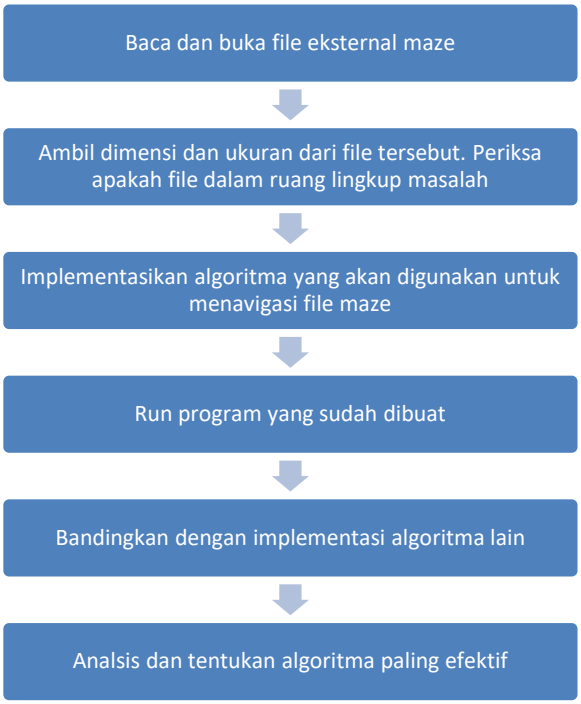
Untuk berbagai bidang, algoritma Greedy telah terbukti efektif untuk menyelesaikan sejumlah permasalahan, seperti pada algoritma Dijkstra dalam graf, Huffman coding dalam kompresi data, dan algoritma penjadwalan pekerjaan. Meskipun demikian, perlu dicatat bahwa kegunaan algoritma Greedy tergantung pada sifat permasalahan yang dihadapi. Dalam beberapa kasus, pendekatan ini dapat menghasilkan solusi yang dekat dengan optimal, sementara dalam kasus lain, bisa jauh dari solusi yang diharapkan.

3. METODOLOGI

Berikut adalah langkah-langkah yang digunakan untuk pengerjaan tugas besar ini:



Gambar 3-1 Diagram proses awal pengerjaan soal tugas besar



Gambar 3-2 Diagram proses pembuatan program soal 1 tugas besar

4. HASIL DAN ANALISIS

4.1 RUANG LINGKUP MASALAH

Format file input maze adalah text berisi '.' (jalan), '#' (dinding), 'S' (titik awal), dan 'E' (titik akhir). Navigasi file hanya dapat dilakukan pada jalan, dan dinding berupa penghalang yang tidak dapat dilewati. File input harus memiliki ukuran MxN (bentuk persegi atau persegi panjang). Program akan memeriksa dan memberitahu pengguna jika nama file yang ingin dibuka tidak ada, file kosong, terdapat karakter tidak valid (di luar keempat karakter yang didefinisikan di awal), dan ukuran maupun dimensi file tidak sesuai.

Jika maze tidak memiliki jalan dari titik awal ke titik akhir, yang artinya seluruh jalan mungkin dihalangi oleh dinding, program akan memberitahu pengguna. Navigasi maze dapat dilakukan dalam 4 arah: atas, bawah, kanan, dan kiri.

Analisis dan perbandingan semua algoritma hanya akan dilakukan untuk eksekusi program yang mencari jalur terpendek. Hal ini dikarenakan terdapat beberapa algoritma yang pemanfaatannya hanya efektif untuk mencari solusi paling optimal. Dengan demikian, dapat dilakukan perbandingan yang lebih efektif.

4.2 RANCANGAN

Rencana pembagian tugas dapat disajikan seperti pada tabel berikut.

Task	Pembagian Tugas
A* Algorithm	Designer: - 13222104 Implementer - 13222104 Tester - 13222104
DFS	Designer: - 13222103 Implementer: - 13222103 Tester: - 13222103
Dynamic Programming	Designer: - 13222106 Implementer - 13222106 Tester - 13222106
Algoritma Dijkstra	Designer: - 13222107 Implementer - 13222107 Tester - 13222107
BFS	Designer: - 13222105 Implementer - 13222105 Tester - 13222105
Greedy	Designer: - 13222102 Implementer

	- 13222102 Tester - 13222102
Backtracking	Designer: - 13222101 Implementer - 13222101 Tester - 13222101

4.3 IMPLEMENTASI

- Algoritma A* (A-Star)

Dalam program ini, file input dibaca dan diperiksa. Jika file input tidak sesuai, program akan segera keluar. Setelah file input dibaca, program menggunakan algoritma backtracking untuk mencari semua jalan dan jalan yang paling panjang. Kemudian, program memberikan data maze, titik awal, dan titik akhir ke algoritma A*.

Algoritma A* menyimpan data node pada sebuah hash table yang berupa matriks. Data node tersebut mencakup parent dari node, nilai heuristik, dan status ekspansi node tersebut. Nilai heuristik yang dipakai adalah Jarak Manhattan atau jumlah selisih x dan y dari suatu titik ke titik akhir. Implementasi ini memanfaatkan bentuk maze yang berupa matriks MxN.

Implementasi A* pada program ini menyimpan daftar node yang akan diekspansi dalam suatu linked list. Implementasi dapat dioptimisasi dengan mengubah linked list menjadi min-heap. Namun, praktikan memilih mengimplementasikan linked list karena struktur heap belum diajarkan.

Algoritma dimulai dengan alokasi memori dan inisialisasi hash table node. Kemudian, titik awal dipush ke list ekspansi. Lalu, algoritma melakukan while loop yang akan berhenti saat list ekspansi kosong.

Dalam loop tersebut, algoritma menyortir list ekspansi sehingga nilai minimal berada di indeks pertama. Jika titik bernilai minimal sudah pernah diekspansi, titik tersebut akan dilewati. Jika tidak, titik akan ditandai sudah diekspansi.

Algoritma kemudian memeriksa semua titik tetangga yang valid. Jika nilai heuristiknya lebih kecil disbanding yang sebelumnya, algoritma akan mengubah parent dan heuristik titik tetangga tersebut. Ini dilakukan karena nilai heuristik yang lebih kecil berarti algoritma menemukan jalan yang lebih efisien dibanding sebelumnya.

Saat list ekspansi sudah kosong, algoritma akan memeriksa apakah titik akhir sudah diekspansi. Jika iya, algoritma akan membentuk sebuah stack dari rantai parent titik-titik jalan optimal, dimulai dengan titik akhir. Jalan tersebut kemudian digambarkan oleh fungsi drawPath. Namun, jika titik akhir belum diekspansi, program akan menyampaikan bahwa ia gagal mencari jalan ke titik akhir.

- Algoritma Dijkstra

Terdapat beberapa problematika dalam penggunaan Algoritma Dijkstra untuk menyelesaikan permasalahan yang diberikan. Naskah soal meminta program untuk melakukan print pada semua jalur yang tersedia. Hal ini bertentangan dengan prinsip dari algoritma Dijkstra itu sendiri, Dimana algoritma akan mencari jalur terpendek saja. Dengan Batasan yang telah disebutkan, algoritma Dijkstra hanya akan mencetak semua jalur terpendek dengan metode *backtracking*.

Pada algoritma Dijkstra yang dibuat, terdapat empat struct yaitu node, Minheap, PathNode, dan Prevlist. Node untuk melambangkan tujuan dan jaraknya, MinHeap melakukan proses heap biner, PathNode untuk jalur dan PrevList agar dapat melakukan backtracking.

MinHeap memiliki fungsi create, swap, extractmin, decreaseKey, dan IsInMinHeap untuk menjalankan algoritma. Create berguna untuk menginisialisasi min-heap, lalu swap berfungsi untuk menukar antar node, extractmin untuk mendapatkan nilai jarak minimum, decreasekey untuk mengurangi jarak node, dan isInMinHeap berfungsi untuk menentukan apakah suaut titik berada didalam heap.[3]

Untuk algoritma Dijkstra dilaksanakan dengan menginisialisasi jarak sebesar takhingga, lalu mengambil node dengan jarak terkecil. Fungsi juga akan memperbarui node tetangga dan menjaga heap. Setelah tujuan tercapai, fungsi akan berhenti dan melakukan print dari semua path terpendek.

Terdapat juga fungsi tambahan seperti fungsi counter untuk membantu visualisasi pada saat melakukan print. Counter juga dapat menentukan jumlah jalur terdekat yang tersedia

- Dynamic Programming

Implementasi algoritma dynamic programming dapat dikatakan sebagai penyokong untuk menyelesaikan soal ini. Pada dasarnya, dynamic programming melakukan dekomposisi masalah, dan setiap solusi yang ditemukan disimpan di dalam tabel untuk menghindari penyelesaian

masalah yang sudah diselesaikan terlebih dahulu. Fokus dari algoritma ini adalah untuk mencari solusi optimal.

Dalam kasus soal ini, tidak terdapat dekomposisi yang dapat dilakukan yang tidak akan meningkatkan kompleksitas dan kesulitan pemecahan masalah. Maka dari itu, pemanfaatan dari dynamic programming pada soal ini dilakukan pada navigasi maze, dimana setelah setiap dilakukan satu pergerakan, koordinat titik yang dikunjungi akan disimpan dalam array '`dp[i][j].path`' yang akan terisi terus setiap pergerakan terjadi.

Karena fokus dari algoritma ini adalah untuk mencari solusi optimal, maka algoritma ini hanya akan dimanfaatkan untuk mencari jalur terpendek yang dapat diambil dalam navigasi maze. Jika dalam proses navigasi ternyata ditemukan jalur yang lebih pendek, maka array dp akan mengupdate isi array tersebut dengan titik koordinat terbaru yang akan menghasilkan jalur terpendek. Karena jalur yang ditempuh sudah disimpan di dalam array, maka iterasi selanjutnya tidak akan lagi melewati jalur yang sama dengan tujuan menghindari navigasi di jalur yang sudah dilewati.

```
if (dp[newX][newY].length > dp[x][y].length
+ 1) {

//Jika ditemukan jalur lebih pendek

dp[newX][newY].length = dp[x][y].length + 1;

for (int i = 0; i < dp[x][y].length; i++) {

//Dilakukan update koordinat navigasi

dp[newX][newY].path[i][0] =
dp[x][y].path[i][0];

dp[newX][newY].path[i][1] =
dp[x][y].path[i][1];

}

//Update koordinat pada array dp

dp[newX][newY].path[dp[x][y].length][0] =
newX;

dp[newX][newY].path[dp[x][y].length][1] =
newY;
```

Cuplikan kode yang mengatur kondisional Dynamic Programming

Untuk mencari jalur terpanjang, tidak digunakan Dynamic Programming, melainkan algortima DFS. Sederhananya, algoritma DFS akan menjalankan program dan mencari jalur dari titik Start ke End. Bila ditemukan suatu jalur, maka panjang jalur dan titik-titik yang dilalui akan disimpan ke variabel '`longestPath`' dan array '`longestPathLength`'. Program ini akan kemudian melakukan iterasi selanjutnya dalam mencari jalur dari Start ke End dan apabila nilai '`longestPath`' yang ditemukan lebih besar, maka jalur tersebut akan diupdate sebagai jalur terpanjang. Hal ini akan terus berulang hingga tidak ada variasi jalur yang dapat diambil lagi.

Jalur-jalur lainnya yang diambil saat melakukan navigasi maze juga memanfaatkan algoritma DFS untuk mencari jalur terpanjang sebelumnya. Bila sebelumnya diperlukan sebuah array '`longestPath`' untuk menunjukkan jalur terpanjang, untuk menampilkan seluruh jalur yang diambil, cukup diakukan print untuk jalur (array path) yang diambil, dan semua variasi jalur lain yang kemudian dilakukan secara rekursif (DFS).

Terdapat juga beberapa fungsi pembantu lainnya, yaitu :

Fungsi readMatrix yang digunakan untuk menentukan dimana titik awal dan akhir (start dan end) file dan mengkonversi nilai itu ke dalam koordinat (`startX`, `startY`) dan (`endX`, `endY`).

Fungsi getMatrixSize yang digunakan untuk menghitung jumlah baris dan kolom file yang dibuka. Fungsi ini juga mengabaikan whitespace bila ada untuk menghindari kesalahan pembacaan file.

Fungsi isSegiempat yang digunakan untuk memenuhi ruang lingkup soal bahwa bentuk maze harus segiempat. Fungsi ini memeriksa apabila setiap baris dalam file memiliki jumlah kolom yang sama, sehingga membentuk segiempat.

Analisis dari program yang memanfaatkan algoritma dynamic programming memberikan beberapa insight yang menarik.

Pertama, waktu yang diperlukan untuk mengeksekusi program, dengan test case maze dengan berbagai ukuran dari rentang (11x12) hingga (20x21) memberikan rentang runtime dari 140ms hingga 800ms. Ukuran maze yang lebih besar, dan titik start yang berada di tengah file memberikan waktu eksekusi yang lebih lama dibandingkan ukuran maze yang lebih kecil dan titik start yang berada di pojokan file.

Kedua, dengan mengamati cara kerja algoritma dynamic programming, maka dapat diamati bahwa semakin besar ukuran maze, semakin besar penggunaan memori yang diperlukan, karena tabel yang diperlukan untuk menyimpan data titik yang dikunjungi akan semakin besar. Jika titik awal juga tidak berada di pojok file, artinya ada empat arah awal yang mungkin dapat dijelajahi program, sehingga akan memakan waktu yang lebih lama.

Ketiga, karena penggunaan memori yang bergantung pada ukuran maze, maka sangat mungkin apabila untuk navigasi maze yang sangat besar (misalnya 100x100), pemanfaatan algoritma dynamic programming tidak akan terlalu efektif. Dengan contoh ukuran maze 100x100, artinya inisialisasi tabel dp akan membutuhkan memori untuk 10000 masukan. Ditambah lagi bahwa akan perlu disimpan lebih dari satu jalur (semua jalur yang ditempuh) untuk ditampilkan.

Algoritma DFS yang digunakan untuk mencari jalur terpendek dan semua jalur memiliki masalah utama dalam kompleksitasnya. Kompleksitas waktu program akan meningkat secara eksponensial seiring meningkatnya ukuran maze.

Kesimpulan yang dapat ditarik untuk algoritma Dynamic Programming adalah bahwa algoritma ini memberikan waktu eksekusi program yang secara teoritis akan lebih cepat karena tidak akan menghabiskan waktu untuk menghitung solusi duplikat yang sudah ditemukan. Namun, algoritma ini memiliki batasan saat menghadapi masalah dalam skala yang lebih besar, karena diperlukan alokasi memori untuk menyimpan solusinya, semakin besar memori yang diperlukan untuk memanfaatkan algoritma ini.

- Backtracking

Penggunaan algoritma Backtracking pada persoalan ini memang bisa membantu dalam mencari shortest, longest, dan all path. Tetapi, backtracking memiliki satu kelemahan besar, yaitu efisiensi.

Seperti yang dijelaskan pada studi pustaka, backtracking adalah algoritma yang memeriksa semua percobaan yang bisa terjadi dalam mencari path yang diingingkan. Maka terjadi pemborosan dalam waktu dan memori yang digunakan.

Cara kerja algoritma backtracking pada soal ini dimulai dengan menetukan basis. Jika posisi saat ini adalah 'E', berarti jalur dari 'S' ke 'E' telah ditemukan. Menyimpan koordinat 'E' di jalur saat ini dan menambah hitungan jalur.

Jika jalur saat ini lebih pendek dari jalur terpendek yang ditemukan sebelumnya, memperbarui jalur terpendek.

Jika jalur saat ini lebih panjang dari jalur terpanjang yang ditemukan sebelumnya, memperbarui jalur terpanjang.

Dilanjutkan dengan menyimpan posisi saat ini dengan variabel ‘x’. Dan setelah itu, melakukan iterasi kesegala arah(atas, bawah, kanan, kiri) dan melakukan kembali rekursid penggunaan fungsi findpath dengan nilai x dan y setelah dilakukan pergerakan ke arah yang lain.

Setelah terkumpul semua semua nilai x dan y untuk shortest path dan longest path. Maka dilakukan pengoutputan sehingga dihasilkan map untuk shortest dan longest path. Sedangkan path yang lain hanya ditampilkan dalam bentuk jumlah. Dikarenakan jika kita melakukan hal yang sama seperti longest dan shortes path untuk semua path, maka memori yang digunakan akan membludak. Karna dibutuhkan array integer sebanyak rowxcolxtotal path, dan total path dari semua maze bisa mencapai ratusan ribu path. Karna itu, code ini hanya menampilkan jumlah path.

Jadi kesimpulannya, Algoritma backtracking digunakan dalam berbagai situasi yang melibatkan pencarian solusi dari masalah yang memiliki banyak kemungkinan. Algoritma ini efektif untuk masalah di mana solusi dapat dibangun langkah demi langkah dan di mana keputusan dapat dibatalkan (backtracked) untuk mencoba alternatif lainnya. Jadi, algoritma ini cocok dalam mencari banyak cara atau banyak path untuk mencari solusi dalam persoalaan labirin. Tetapi tidak disarankan dalam mencari shortest path. Karna algoritma ini akan terus mencari semua kemungkinan yang ada, lalu baru menentukan path mana yang terpendek. Alih-alih bukan dengan langsung menentukan jalur mana yang terpendek ketika dalam proses mencari path nya

- DFS

Algoritma DFS sebenarnya tidak dapat menghasilkan semua output yang diinginkan soal (all path, shorthest path, dan juga longest path) karena algoritma DFS bekerja dengan tidak mencoba semua kemungkinan yang ada. DFS bekerja dengan menelusuri satu jalan lalu kembali mundur jika buntu dan mencari cabang terdekat dan mencobanya lagi dan diulang hingga menemukan jalannya lalu berhenti ketika menemukan jalan keluar. Namun DFS dapat dimodifikasi untuk menemukan output-output yang diminta dengan membuatnya tidak berhenti ketika menemukan jalan keluar, sehingga menjadi bruteforce atau mencoba semua jalur.

Pada pengimplementasian DFS, dapat digunakan 2 jenis pendekatan, yaitu rekursid dan iteratif. Pada

kode yang dibuat ini, digunakan pendekatan rekursif. Berikut penjelasan kode yang telah dibuat.

Digunakan 2 jenis structure yaitu pair untuk menyimpan koordinat dari sel dalam maze, dan stack untuk menyimpan jalur yang sedang dieksplorasi

Fungsi yang menjadi pengimplementasian dari DFS di kode yang dibuat adalah findAllPaths, yaitu fungsi rekursif untuk menjelajahi semua jalur dari titik awal ke titik akhir.

DFS diinisialisasikan dengan menandai semua sel sebagai belum dikunjungi, membaca file maze, menemukan posisi awal dan akhir dalam maze, dan menginisialisasikan stack untuk menyimpan jalur yang sedang dieksplorasi.

Lalu dilakukan penandaan sel saat ini sebagai dikunjungi dan menambahkan sel tersebut ke dalam tumpukan jalur. Selanjutnya mengecek apakah sel saat ini adalah sel terakhir. Jika ya, perbarui jalur terpanjang dan terpendek jika diperlukan. Jika tidak, dilanjutkan menjelajahi sel-sel tetangga yang valid. Jika semua sel tetangga sudah dijelajasi, kembalikan dengan menghapus sel saat ini dari stack. Langkah-langkah tersebut dilakukan secara rekursif hingga semua jalur telah dicoba.

Kode berhasil dibuat dan berhasil memecahkan maze_1, maze_3, maze_4, dan maze_6. Dengan waktu yang dapat dilihat pada tabel pengujian. Dari tabel pengujian menunjukkan waktu yang dibutuhkan oleh program untuk menyelesaikan maze. Dapat dilihat maze_1 memerlukan waktu paling banyak. Hal tersebut terjadi karena maze_1 memiliki total path sebanyak 502020. Dapat dilihat pula pada maze_3 dan maze_6 hanya membutuhkan waktu kurang dari 1 ms. Hal tersebut karena maze_3 dan maze_6 merupakan labirin yang tidak banyak percabangannya dan hanya memiliki path sebanyak 2 dan 8 secara berurut.

- BFS

Pada dasar interaksi pengguna dengan program, program digunakan dengan diberikan input nama teks yang mengandung labirin oleh pengguna. Melalui implementasi program, diberikan dua jenis output utama, yaitu output saat semua input benar dan saat ada aspek dari input yang salah.

Saat semua input benar, program mengeluarkan output berupa jalur optimal yang mungkin untuk dilalui dan banyaknya langkah yang diperlukan untuk mencapai titik akhir.

Saat salah satu aspek dari input terdeteksi salah, output error akan dikeluarkan dengan jenis kesalahan yang ada, yang meliputi file teks yang tidak diteukan (kesalahan input nama file), kekosongan isi file teks, tidak terdapatnya titik awal dan akhir dalam labirin, titik awal dan titik akhir di luar labirin, tidak terdapat jalur yang menghubungkan titik awal dan titik akhir labirin, bentuk labirin yang bukan merupakan persegi atau persegi panjang sempurna, dan adanya karakter yang tidak valid dalam labirin.

Pada implementasi algoritma, proses pencarian dilakukan dengan melakukan BFS pada semua noda yang berada di samping noda yang ingin dicek (yang dimulai pada titik awal) yang dimulai dari pengecekan pada sel bagian atas, bagian kiri, bagian kanan, dan kemudian bagian bawah dari sel yang ingin dianalisis. Semua koordinat sel samping dianalisis melalui suatu fungsi pengecekan validitas koordinat labirin dan pengecekan kunjungan terhadap sel. Apabila sel samping tergolong valid dan belum dikunjungi, sel tersebut dimasukkan ke dalam *queue first-in first-out.*

Semua sel yang valid dan belum dikunjungi, termasuk sel awal yang telah dianalisis masuk ke dalam golongan sel yang telah dikunjungi. Kemudian, dilakukan *pop* terhadap *queue* dan diulangi analisis sel samping terhadap sel yang telah di-*pop* tersebut.

Proses ini diulang sampai ditemukan jalur antara titik awal dan titik akhir atau saat tidak ada jalur yang bisa ditemukan (semua sel yang mungkin untuk ditelusuri telah ditelusuri dan *queueu* telah kosong).

Mengenai implementasi untuk menentukan jalur sendiri, dibuat sebuah matriks tiga dimensi yang di dalamnya disebutkan noda "orang tua" untuk masing-masing noda yang telah dikunjungi dalam labirin. Dengan kata lain, masing-masing koordinat dalam labirin diberikan koordinat yang merupakan penunjuk ke koordinat sebelumnya. Semua koordinat diberikan tunjukan dari titik akhir hingga posisi awal.

Mengenai implementasi pengeluaran output, pertama dibuat peta labirin menggunakan data matriks integer (yang awalnya berisikan data 0 dan 1 yang kemudian diubah menjadi karakter 'S','E','#', dan '.'). Titik akhir labirin kemudian

ditelusuri dengan mengecek langkah sebelumnya dari titik tersebut yang kemudian diulangi hingga mencapai titik awal. Semua koordinat yang ditelusuri pada penelusuran tahap diberi tanda '+' pada peta labirin. Peta labirin dengan jalur optimal kemudian di cetak yang diikuti dengan banyaknya langkah.

Mengenai implementasi pembacaan teks menjadi matriks integer yang berisi 0 sebagai noda yang tidak dapat ditelusuri dan 1 sebagai noda yang bisa ditelusuri, pertama dilakukan konkatenasi semua data pada file teks pada sebuah variabel. Variabel tersebut kemudian dianalisis secara satu persatu pada fungsi pembacaan peta untuk menentukan nilai koordinat matriks integer peta dan koordinat titik awal serta titik akhir dalam peta.

Melalui BFS, jalur optimal ditentukan melalui penelusuran semua koordinat yang mampu ditelusuri. Implementasi BFS bisa dibandingkan dengan air yang merambat ke semua posisi yang mungkin ada dengan perambatan yang homogen ke semua noda mulai dari titik awal sampai ke titik air. Oleh karena itu, algortima ini tergolong dalam kategori *informed brute force* karena sifat algoritmanya yang diharuskan mengecek semua koordinat yang berada di jarak yang sama dari titik awal hingga titik akhir.

- Algoritma Greedy

Algoritma Greedy, memberikan keputusan optimal yang bersifat lokal pada setiap langkah, mengarah pada solusi global yang diharapkan. Dalam konteks labirin, algoritma ini terfokus pada gerakan yang paling menjanjikan pada setiap titik, meskipun mencoba semua jalur yang memungkinkan untuk mencapai tujuan akhir. Kelebihannya terletak pada sifatnya yang sederhana dan cepat dalam eksekusi. Namun, kelemahannya adalah tidak menjamin solusi optimal dalam semua kasus, bergantung pada sifat permasalahan yang dihadapi. Dalam implementasi kode ini, algoritma Greedy digunakan untuk mencari semua jalur mungkin dari titik awal ke titik akhir dalam labirin, dengan menyimpan dan memeriksa setiap kemungkinan langkah secara bergantian.

Implementasi algoritma Greedy dalam kode ini melibatkan fungsi-fungsi yang memiliki peran khusus dalam proses pencarian jalur dalam labirin. Pertama, fungsi readMaze() bertanggung jawab membaca labirin dari file dan menentukan posisi titik awal dan akhir. Ini dilakukan dengan memindai setiap karakter dalam labirin dan mencari lokasi titik awal (S) dan akhir (E). Kemudian, isValid() memeriksa apakah suatu posisi dalam labirin valid untuk dilalui, dengan memastikan posisi tersebut berada dalam batas labirin, tidak merupakan dinding, dan belum pernah dikunjungi sebelumnya.

Fungsi findPaths() adalah inti dari algoritma Greedy dalam kode ini. Fungsi ini melakukan pencarian jalur dari titik awal ke titik akhir dengan mengikuti pendekatan Greedy, di mana pada setiap langkah, ia mencoba bergerak ke arah yang terlihat paling menjanjikan. Ini dilakukan dengan mengeksplorasi setiap arah (kanan, bawah, kiri, atas) dari posisi saat ini, menandai sel yang sudah dikunjungi, dan menyimpan jalur-jalur yang mungkin. Fungsi addPath() digunakan untuk menambahkan jalur yang ditemukan ke daftar jalur yang valid.

Setelah menemukan semua jalur, fungsi printPath() digunakan untuk mencetak jalur-jalur tersebut. Fungsi-fungsi ini bersama-sama membentuk proses utama pencarian jalur dalam labirin menggunakan algoritma Greedy. Terakhir, dalam fungsi main(), langkah-langkah tersebut dieksekusi secara berurutan dengan memanfaatkan waktu untuk mengukur kinerja eksekusi algoritma.

Dalam kasus ini, meskipun Greedy digunakan untuk memandu setiap langkah pencarian jalur, ia tidak menjamin solusi terpendek atau terpanjang secara global. Namun demikian, pendekatan ini memberikan gambaran bagaimana algoritma Greedy dapat diterapkan dalam sebuah permasalahan konkret untuk menemukan solusi yang memadai dalam konteks yang diberikan.

4.4 PENGUJIAN

Algoritma yang telah dibuat kemudian diuji dan dibandingkan performanya, khusus hanya untuk jalur terpendek.

Tabel 4-1 waktu eksekusi dari test case

No.	Jenis Algoritma	Hasil uji coba
1	**Algoritma A* (A-Star)**	**Maze 1:** 7.669 milliseconds **Maze 2:** 14.938 milliseconds **Maze 3:** 23.645 milliseconds **Maze 4:** 0.914 milliseconds **Maze 5:** 9.131 milliseconds **Maze 6:** 76.583 milliseconds

2	**Algoritma Dijkstra**	**Maze 1 :** 0.467 milliseconds **Maze 2 :** 0.36 milliseconds **Maze 3 :** 1.295 milliseconds **Maze 4 :** 0.075 milliseconds **Maze 5 :** 0.28 milliseconds **Maze 6 :** 3.16 milliseconds
3	**Algoritma Backtracking**	**Maze 1 :** 217.018000 milliseconds **Maze 3 :** 72.528000 milliseconds **Maze 4 :** 31.372000 milliseconds **Maze 6 :** 207.005000 milliseconds
4	**DFS**	**Maze 1:** 448.944 milliseconds **Maze 3:** 0.213 milliseconds **Maze 4:** 23.458 milliseconds **Maze 6:** 0.957 milliseconds
5	**BFS**	**Maze 1: 0.037 miliseconds** **Maze 2: 0.020 miliseconds** **Maze 3: 0.040 miliseconds** **Maze 4: 0.021 miliseconds** **Maze 5: 0.024 miliseconds** **Maze 6: 0.118 miliseconds**
6	**Dynamic Programming**	**Maze 1:** 1.519 milliseconds **Maze 2:** 1.002 milliseconds **Maze 3:** 8.307 milliseconds **Maze 4:** 0 milliseconds **Maze 5:** 2.614 milliseconds **Maze 6:** 6.214 milliseconds
7	**Greedy**	**Maze 1:** 0.000 milliseconds **Maze 2:** 2689. 395 milliseconds **Maze 3: 0.000** milliseconds **Maze 4: 7.709** milliseconds **Maze 5:** 19.9440 milliseconds **Maze 6:** 109.9020 milliseconds

Dari tabel di atas, dapat diamati bahwa setiap algoritma membutuhkan waktu yang berbeda untuk mencari jalan terdekat. Didapatkan bahwa algoritma dengan performa terbiak untuk mencari jalur tercepat yaitu algoritma Dijkstra. Akan tetapi, efektifitas suatu algoritma tidak hanya murni dinilai dari waktu yang diperlukan untuk mengeksekusi program.

Algoritma yang dapat menyelesaikan seluruh jalur maze dengan performa terbaik yaitu A* algorithm. Hal ini dikarenakan algoritma A* adalah algoritma yang paling efektif dalam *pathfinding*. Meskipun untuk beberapa kasus, algoritma lain seperti Djikstra dan algoritma greedy lainnya memiliki waktu eksekusi yang mungkin lebih cepat, A* tetap menjadi algoritma yang lebih efektif karena konsistensinya. Selain itu, algoritma A* juga memiliki satu keunggulan utama, yaitu memanfaatkan heuristik untuk navigasi file, yang berarti algoritma ini dapat menemukan langsung jalur dari awal ke akhir, sehingga algoritma A* memiliki adaptibilitas yang sangat baik.[4]

Meskipun algoritma A* merupakan pengembangan dari algoritma Dijkstra, implementasi dan pengukuran yang berbeda menyebabkan data yang berbeda dari ekspektasi. Implementasi Dijkstra menggunakan struktur data heap yang lebih efisien dalam operasi yang digunakan pada algoritma. Selain itu, output yang dikeluarkan kedua algoritma ini pun memiliki format berbeda. Pengukuran waktu fungsi berjalan pun menggunakan cara yang berbeda.

5. KESIMPULAN

- Algoritma Dynamic Programming memiliki kecepatan eksekusi yang cukup cepat, namun memiliki keterbatasan untuk kasus masalah yang lebih besar karena memerlukan memori yang semakin besar pula untuk menyimpan solusi tersebut dalam tabel.
- Algoritma backtracking memiliki keunggulan dalam fleksibilitas dan kemampuan menemukan semua solusi potensial untuk suatu masalah, membuatnya sangat efektif untuk problematika seperti pencarian jalur, penyusunan puzzle, dan pengaturan kombinatorial. Namun, kekurangannya adalah kompleksitas waktu yang tinggi, karena bersifat eksponensial, sehingga tidak efisien untuk masalah dengan ruang pencarian yang sangat besar.
- Algoritma Dijkstra memiliki kompleksitas waktu yang rendah karena bersifat logarithmic. Hal ini menyebabkan proses terjadi dengan sangat cepat. Akan tetapi, Algoritma Dijkstra hanya dapat mencari jalur terdekat pada maze, sehingga tidak dapat digunakan untuk menyelesaikan permasalahan yang diberikan
- Algoritma Greedy mengilustrasikan efektivitas pendekatan tersebut dalam menyelesaikan permasalahan pencarian jalur dalam labirin. Kelebihannya terletak pada sifatnya yang sederhana dan efisien, serta kemampuannya dalam menemukan solusi secara cepat. Namun, kelemahannya adalah tidak menjamin solusi optimal dalam semua kasus. Algoritma Greedy pun umumnya memiliki kompleksitas waktu yang relatif rendah jika dibandingkan dengan algoritma lainnya, dikarenakan sering kali hanya memerlukan satu kali iterasi melalui setiap elemen data
- Algoritma BFS, secara algoritmik, merupakan salah satu algoritma yang kurang efisien untuk menganalisis jalur optimal antara dua noda dalam sebuah graf karena sifatnya yang tergolong *brute force.* Skalabilitas algoritma ini kurang baik, dengan peningkatan kompleksitas

waktu yang bersifat kuadratik yang dalam notasi big O dideskripsikan dengan notasi O(M*N) atau O(N^2) sehingga kurang cocok digunakan dalam pencarian jalur dalam peta secara massal. Meskipun begitu, algortima ini masih memiliki kasus penggunaan yang spesifik, seperti pencarian jalur paling optimal dalam sebuah peta atau labirin.

- Algoritma DFS memiliki kompleksitas waktu yang besar, karena bersifat bruteforce. Waktu eksekusi dari algoritma bergantung pada seberapa banyak percabangan yang ada pada maze. Jika percabangan banyak waktu yang diperlukan akan semakin banyak. Namun diluar ini, algoritma ini dapat digunakan untuk menemukan semua jalan sehingga cocok untuk digunakan untuk menyelesaikan masalah yang tidak bisa dipikirkan cara efektifnya.
- Algoritma A* memiliki keunggulan dalam kecepatan pencarian jalan terpendek. Namun, ia menggunakan banyak memori. Penambahan heuristik memungkinkan A* menjadi lebih efektif dibanding Dijsktra. Algoritma ini hanya dapat mencari jalan terpendek.

DAFTAR PUSTAKA

[1] https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

[2] https://www.geeksforgeeks.org/dynamic-programming/

[3] https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[4] https://github.com/Vishnuparammal/maze_runner/blob/master/src/maze.c

[5] https://www.geeksforgeeks.org/greedy-algorithms/

[6] https://www.freecodecamp.org/news/greedy-algorithms/

[7] https://www.javatpoint.com/greedy-algorithms

[8] Dian Rachmawati and Lysander Gustin 2020 J. Phys.: Conf. Ser. 1566 012061 "Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem"

LAMPIRAN

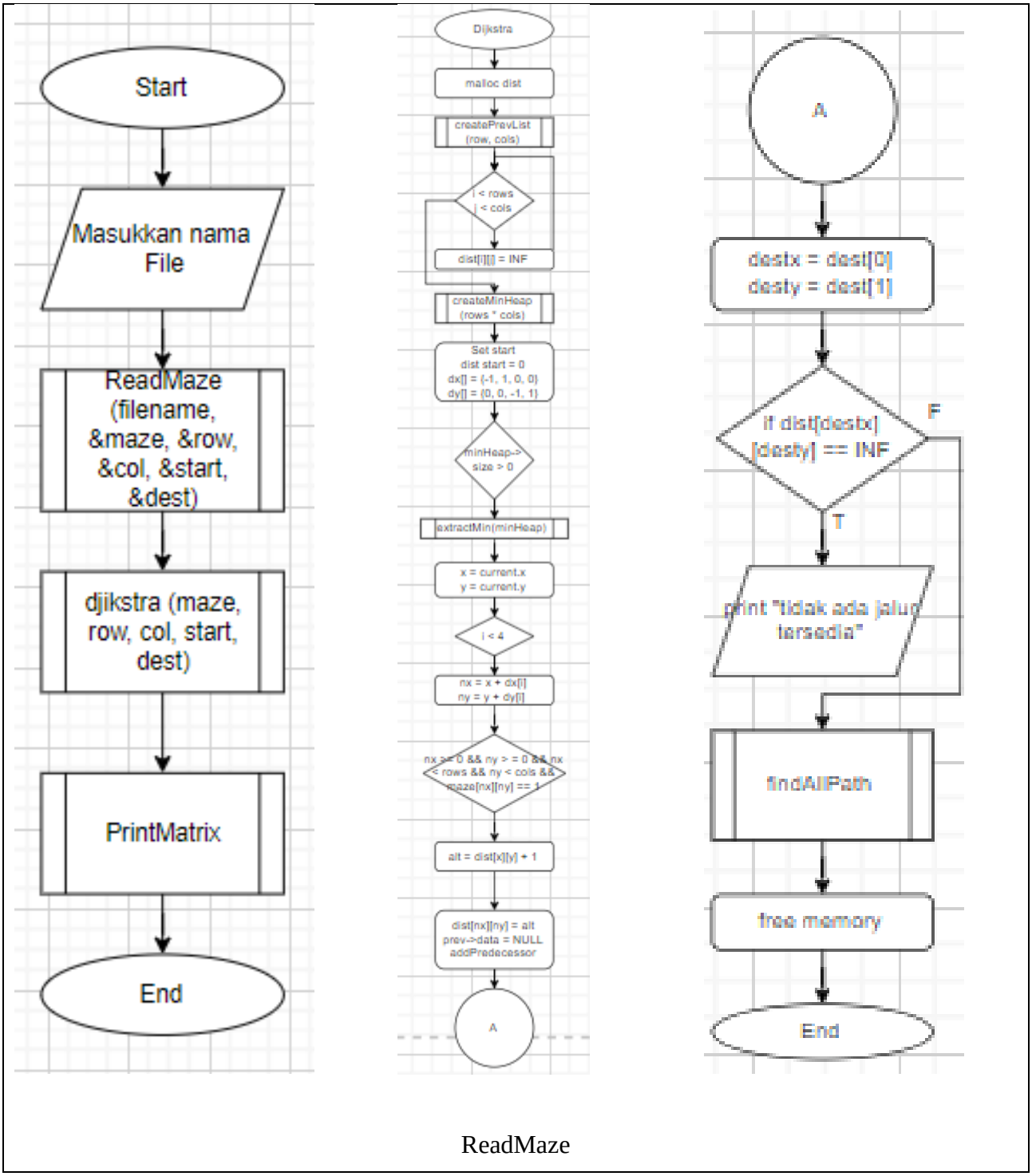
Link ke repository :

https://github.com/AvlKP/EL2208-PPMC-Tubes

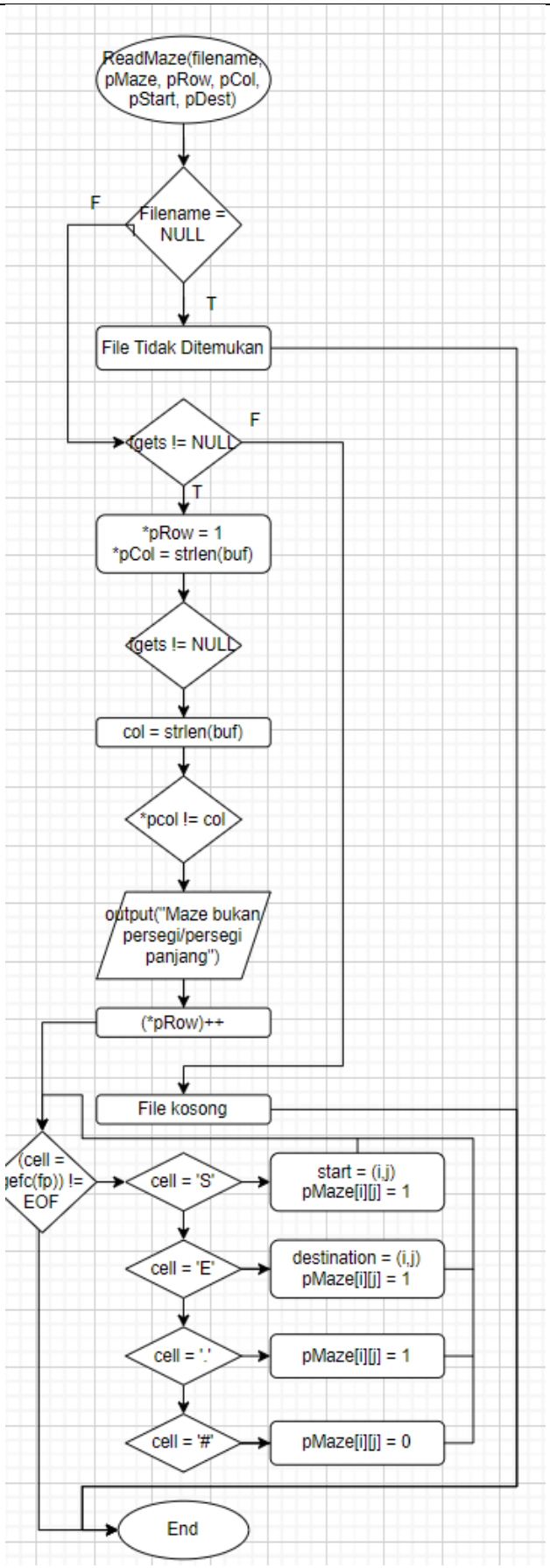
Flowchart

1. Algoritma Dijksta

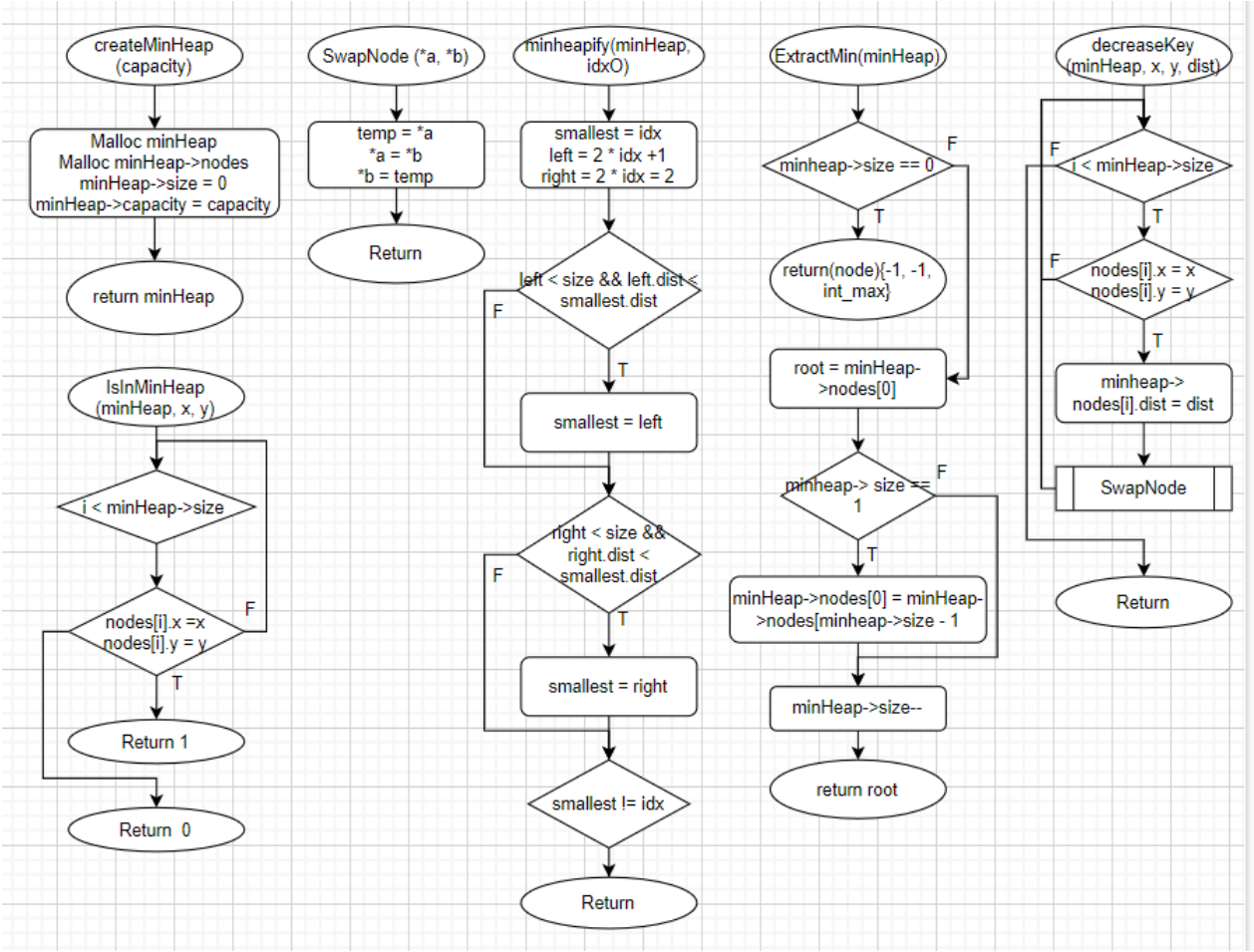
   Main Dijkstra


Start
Masukkan nama File
ReadMaze (filename, &maze, &row, &col, &start, &dest)
djikstra (maze, row, col, start, dest)
PrintMatrix
End
Dijkstra
malloc dist
createPrevList (row, cols)
i < rows
j < cols
dist[i][j] = INF
createMinHeap (rows * cols)
Set start
dist start = 0
dx[] = {-1, 1, 0, 0}
dy[] = {0, 0, -1, 1}
minHeap->size > 0
extractMin(minHeap)
x = current.x
y = current.y
i < 4
nx = x + dx[i]
ny = y + dy[i]
nx >= 0 && ny >= 0 && nx < rows && ny < cols && maze[nx][ny] == 1
alt = dist[x][y] + 1
dist[nx][ny] = alt
prev->data = NULL
addPredecessor
A
A
destx = dest[0]
desty = dest[1]
if dist[destx][desty] == INF
F
T
print "tidak ada jalur tersedia"
findAllPath
free memory
End

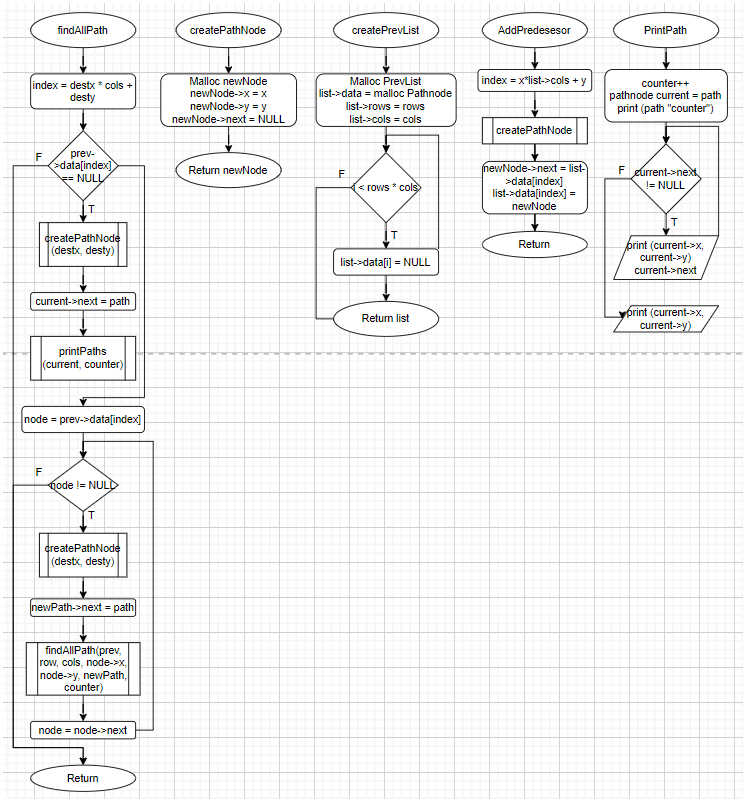
ReadMaze


ReadMaze(filename, pMaze, pRow, pCol, pStart, pDest)
Filename = NULL
F
T
File Tidak Ditemukan
fgets != NULL
F
T
*pRow = 1
*pCol = strlen(buf)
fgets != NULL
col = strlen(buf)
*pcol != col
output("Maze bukan persegi/persegi panjang")
(*pRow)++
File kosong
(cell = fgetc(fp)) != EOF
cell = 'S'
start = (i,j)
pMaze[i][j] = 1
cell = 'E'
destination = (i,j)
pMaze[i][j] = 1
cell = '.'
pMaze[i][j] = 1
cell = '#'
pMaze[i][j] = 0
End

Binary Heap

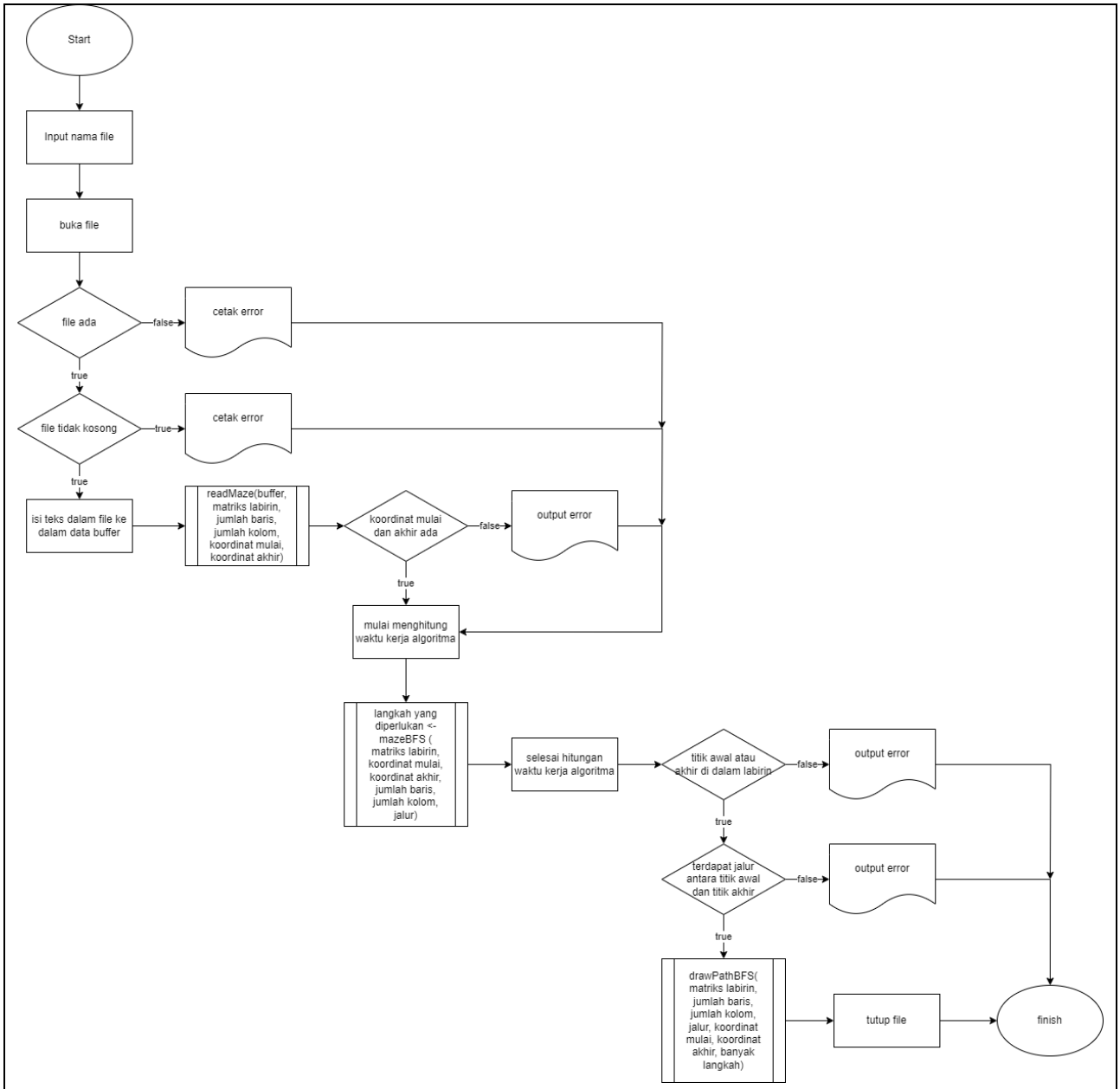


Path

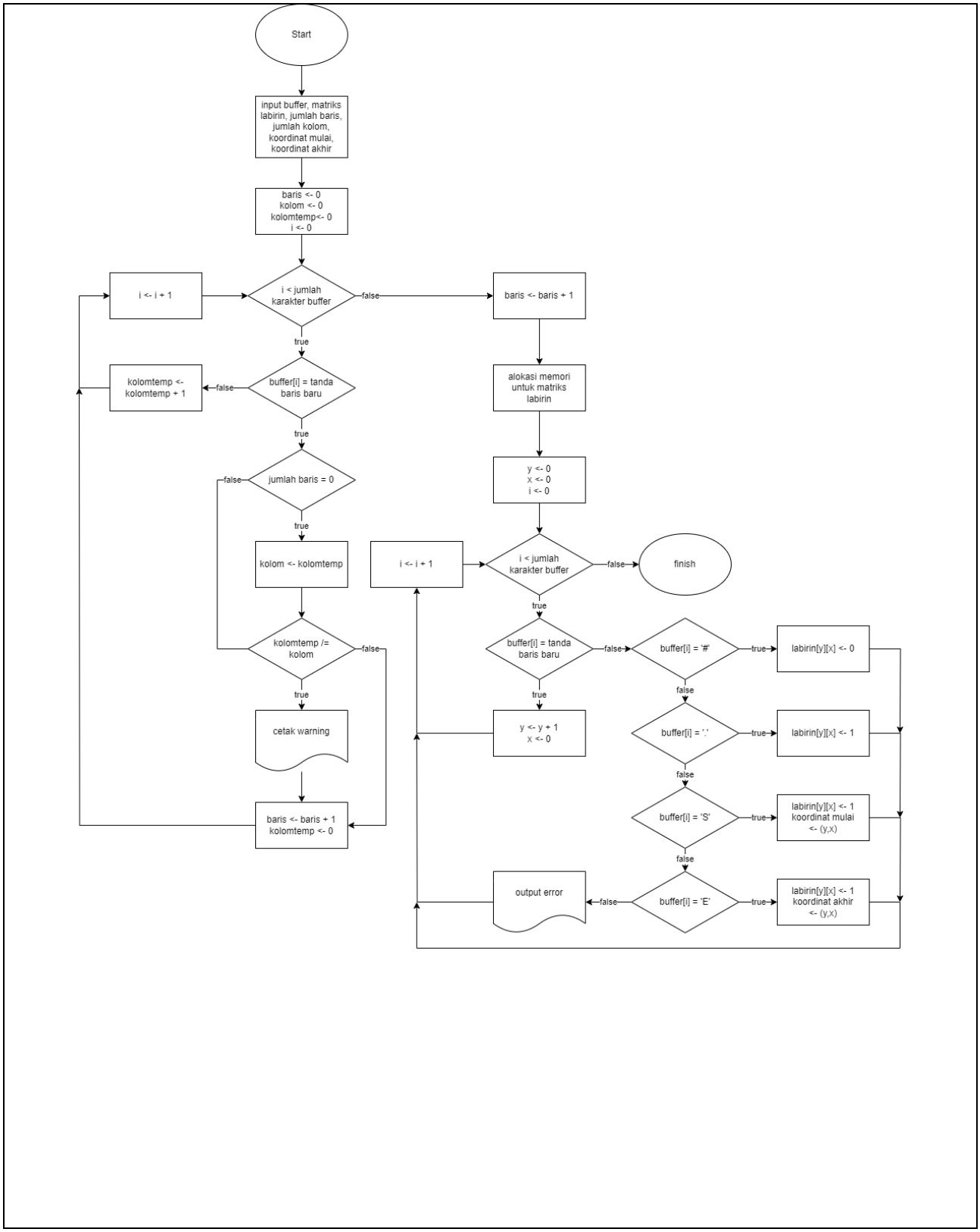


2. Breadth First Search (BFS)

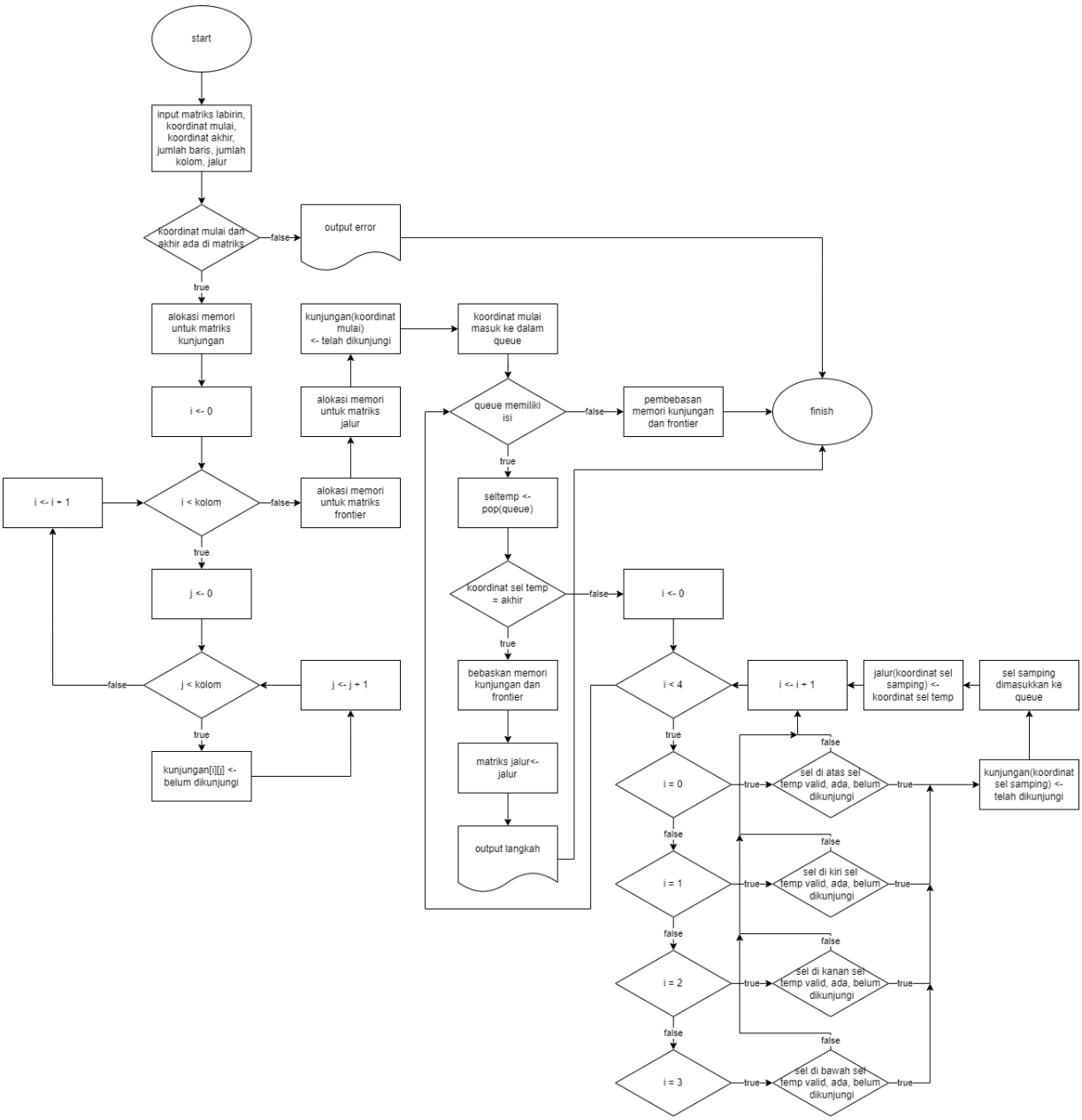
Main


Start
Input nama file
buka file
file ada
false
cetak error
true
file tidak kosong
true
cetak error
true
isi teks dalam file ke dalam data buffer
readMaze(buffer, matriks labirin, jumlah baris, jumlah kolom, koordinat mulai, koordinat akhir)
koordinat mulai dan akhir ada
false
output error
true
mulai menghitung waktu kerja algoritma
langkah yang diperlukan <- mazeBFS (matriks labirin, koordinat mulai, koordinat akhir, jumlah baris, jumlah kolom, jalur)
selesai hitungan waktu kerja algoritma
titik awal atau akhir di dalam labirin
false
output error
true
terdapat jalur antara titik awal dan titik akhir
false
output error
true
drawPathBFS(matriks labirin, jumlah baris, jumlah kolom, jalur, koordinat mulai, koordinat akhir, banyak langkah)
tutup file
finish

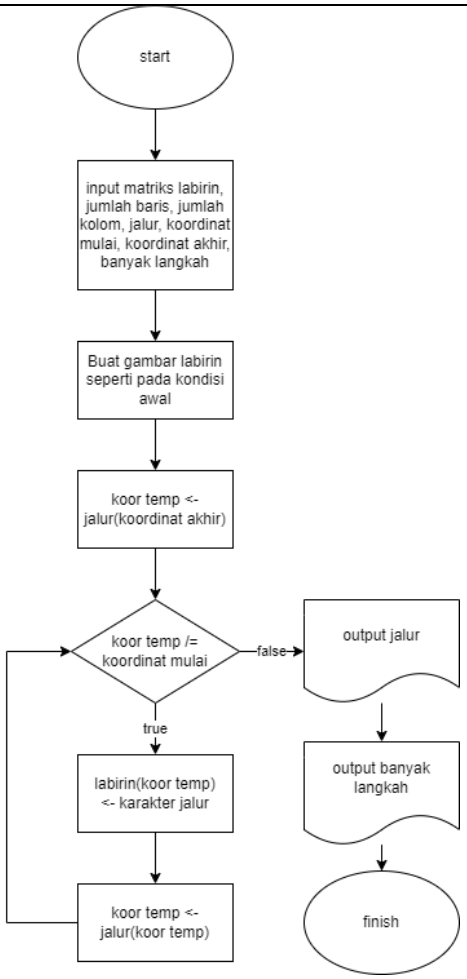
readMaze

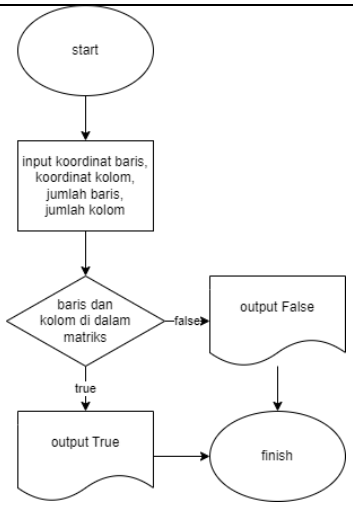

Start
input buffer, matriks labirin, jumlah baris, jumlah kolom, koordinat mulai, koordinat akhir
baris <- 0
kolom <- 0
kolomtemp<- 0
i <- 0
i < jumlah karakter buffer
false
true
baris <- baris + 1
alokasi memori untuk matriks labirin
y <- 0
x <- 0
i <- 0
i <- i + 1
buffer[i] = tanda baris baru
false
true
kolomtemp <- kolomtemp + 1
jumlah baris = 0
false
true
kolom <- kolomtemp
kolomtemp /= kolom
false
true
cetak warning
baris <- baris + 1
kolomtemp <- 0
i <- i + 1
i < jumlah karakter buffer
false
finish
true
buffer[i] = tanda baris baru
false
true
y <- y + 1
x <- 0
buffer[i] = '#'
true
false
labirin[y][x] <- 0
buffer[i] = '.'
true
false
labirin[y][x] <- 1
buffer[i] = 'S'
true
false
labirin[y][x] <- 1 koordinat mulai <- (y,x)
buffer[i] = 'E'
true
false
labirin[y][x] <- 1 koordinat akhir <- (y,x)
output error

mazeBFS

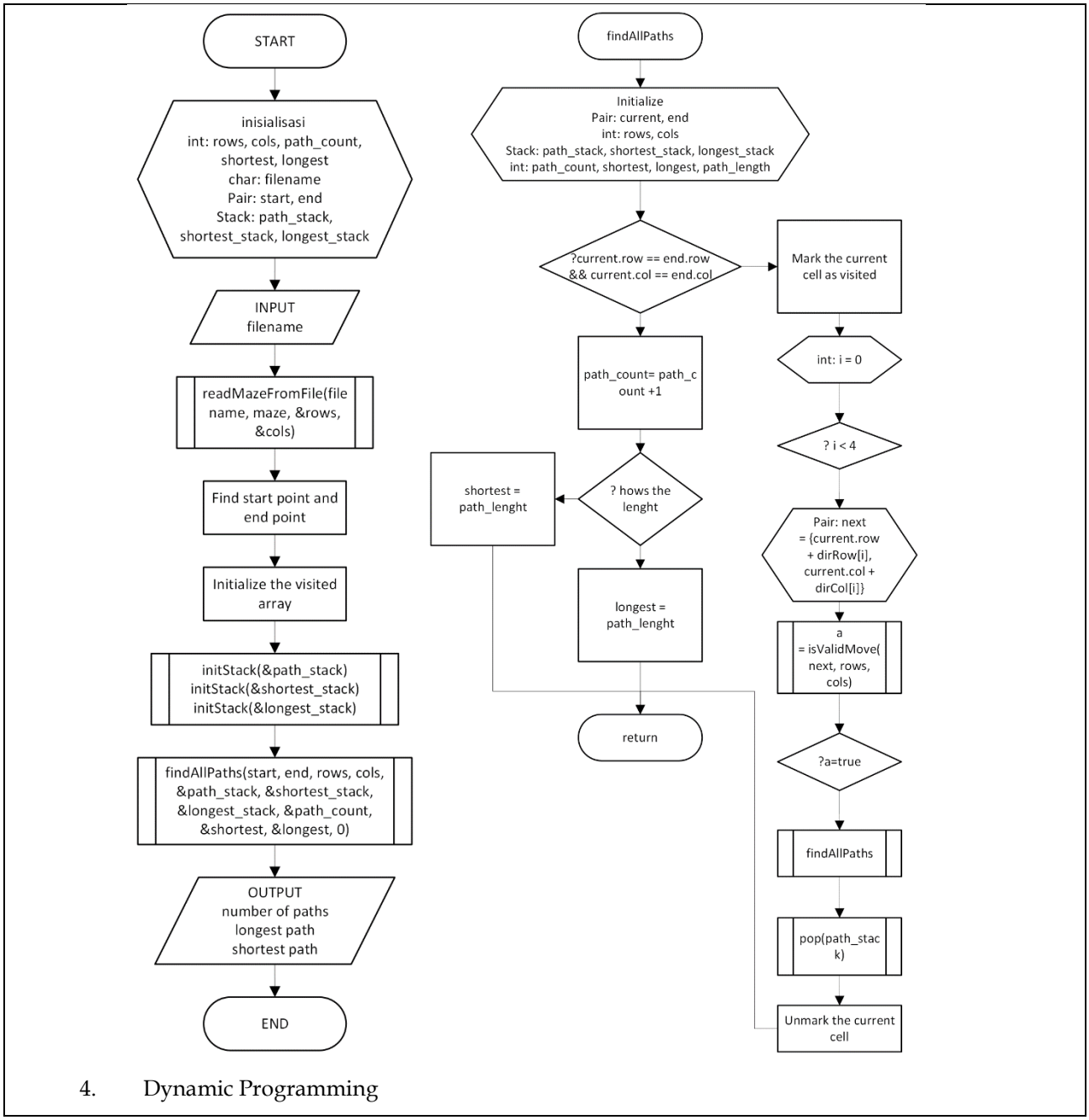


drawPathBFS IsValid

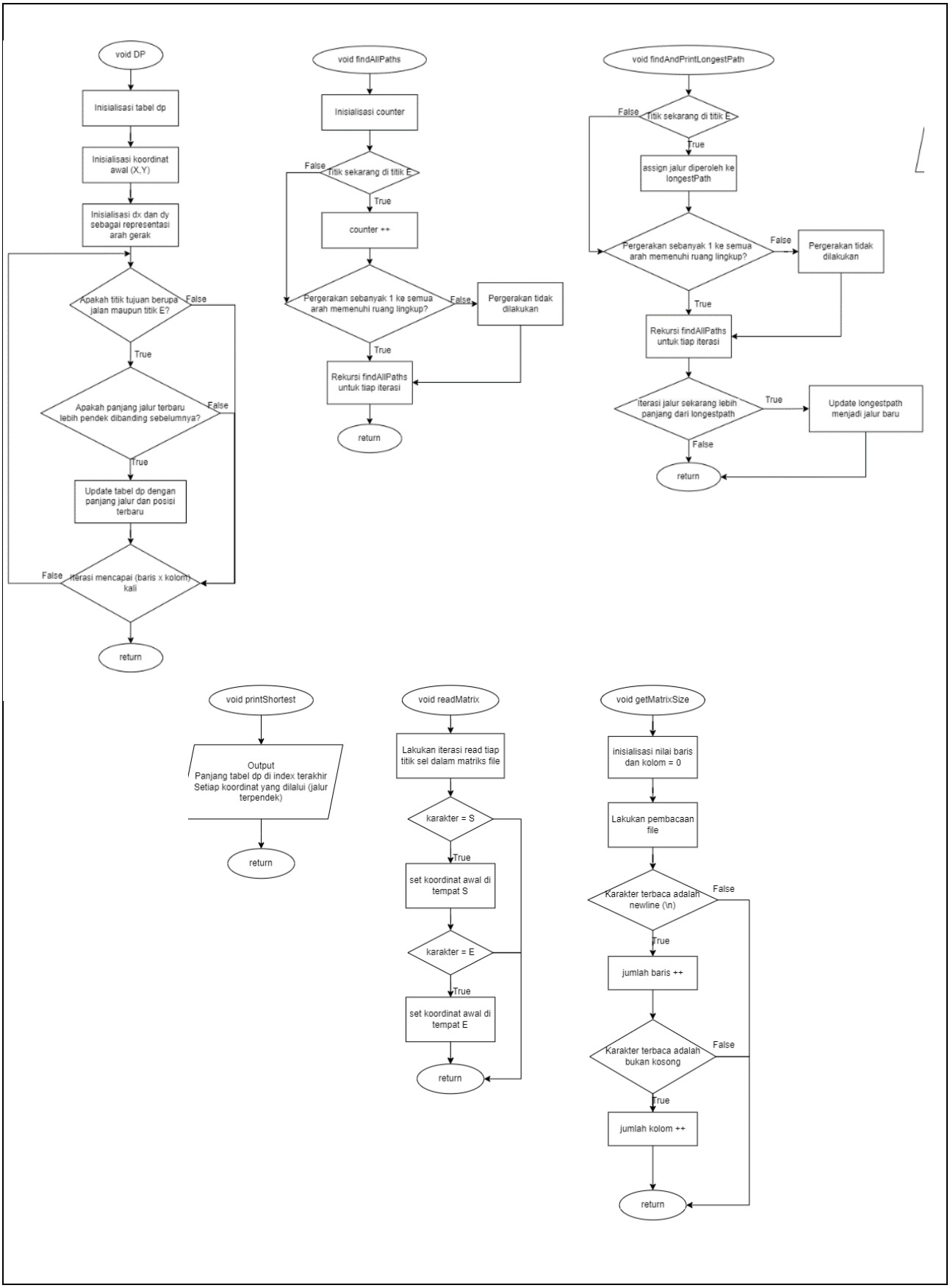


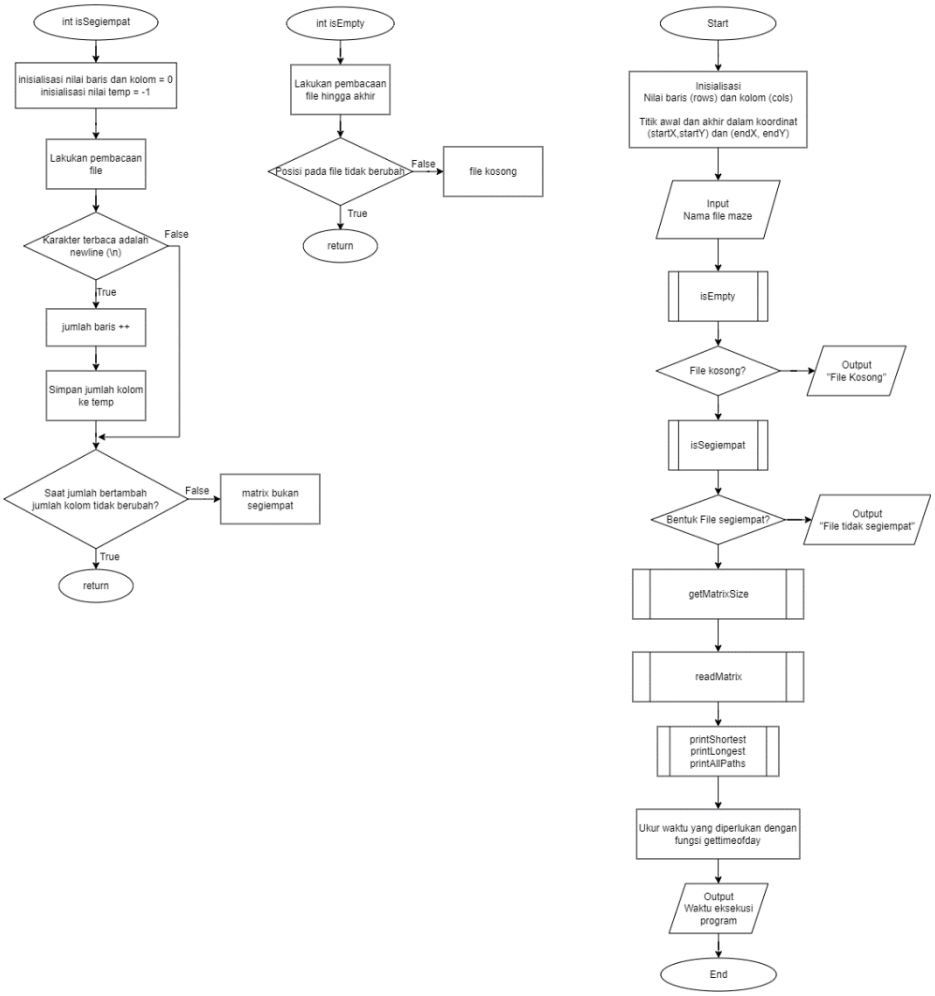


3. Depth First Search (DFS)

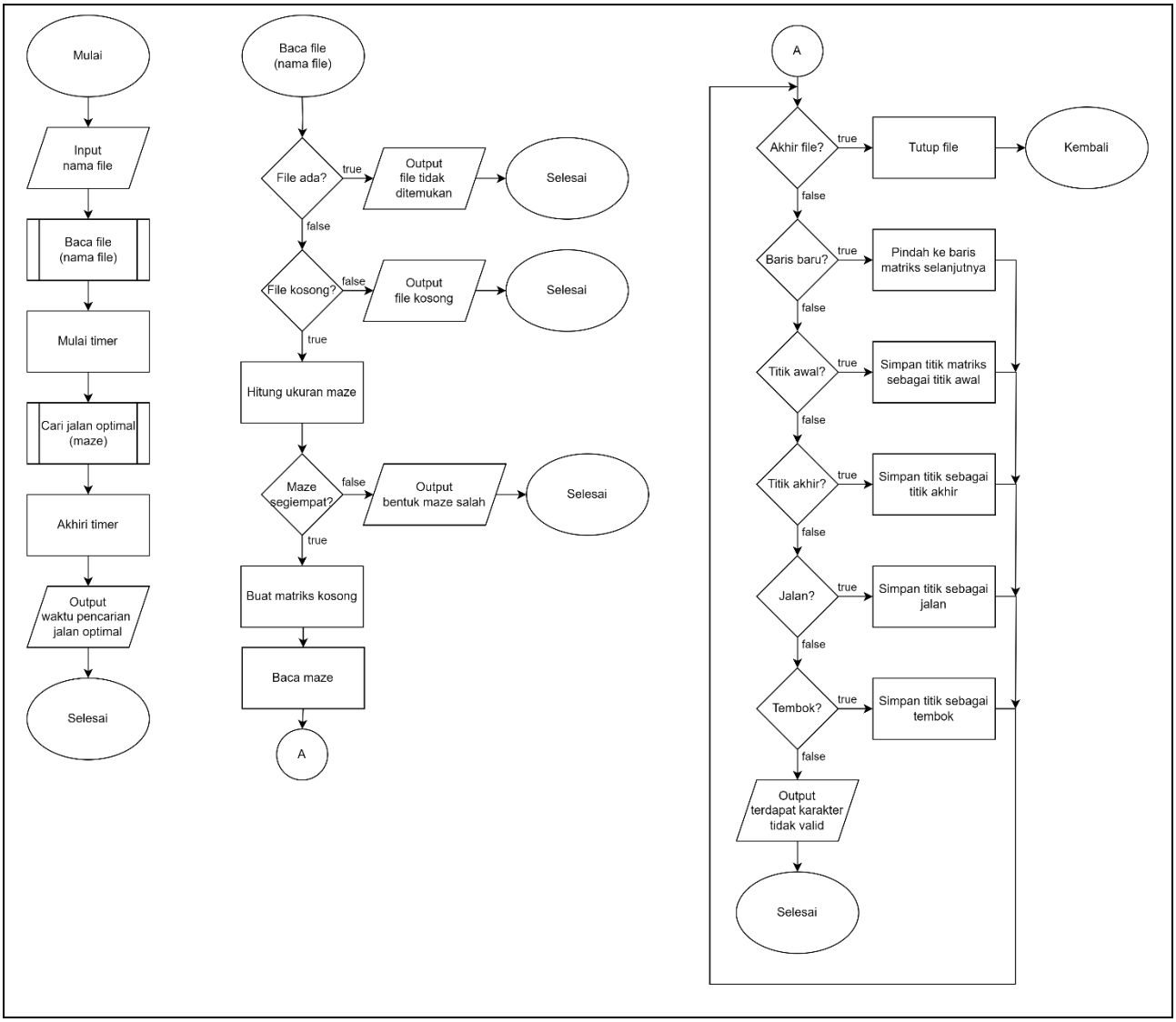


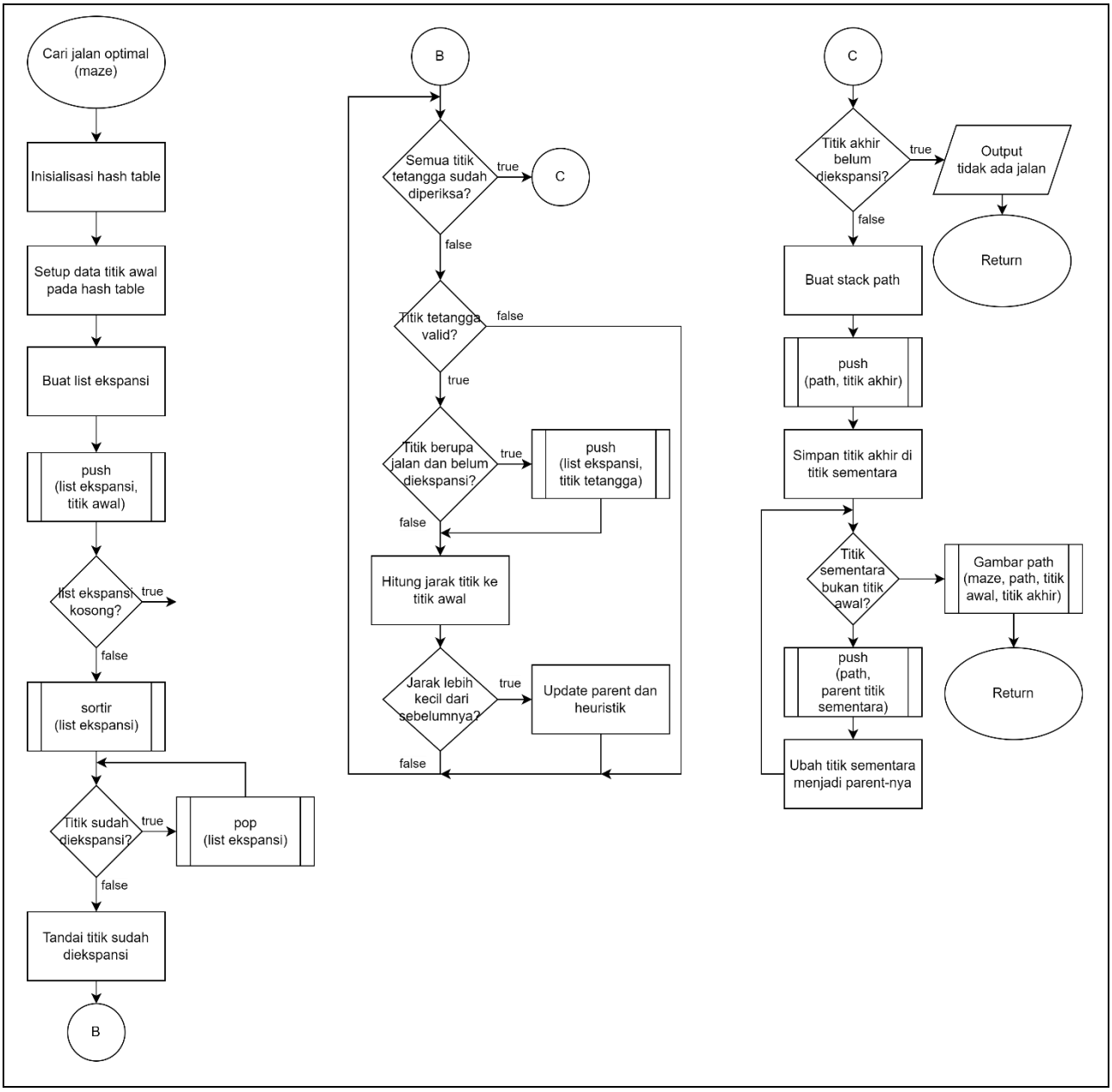
4. Dynamic Programming

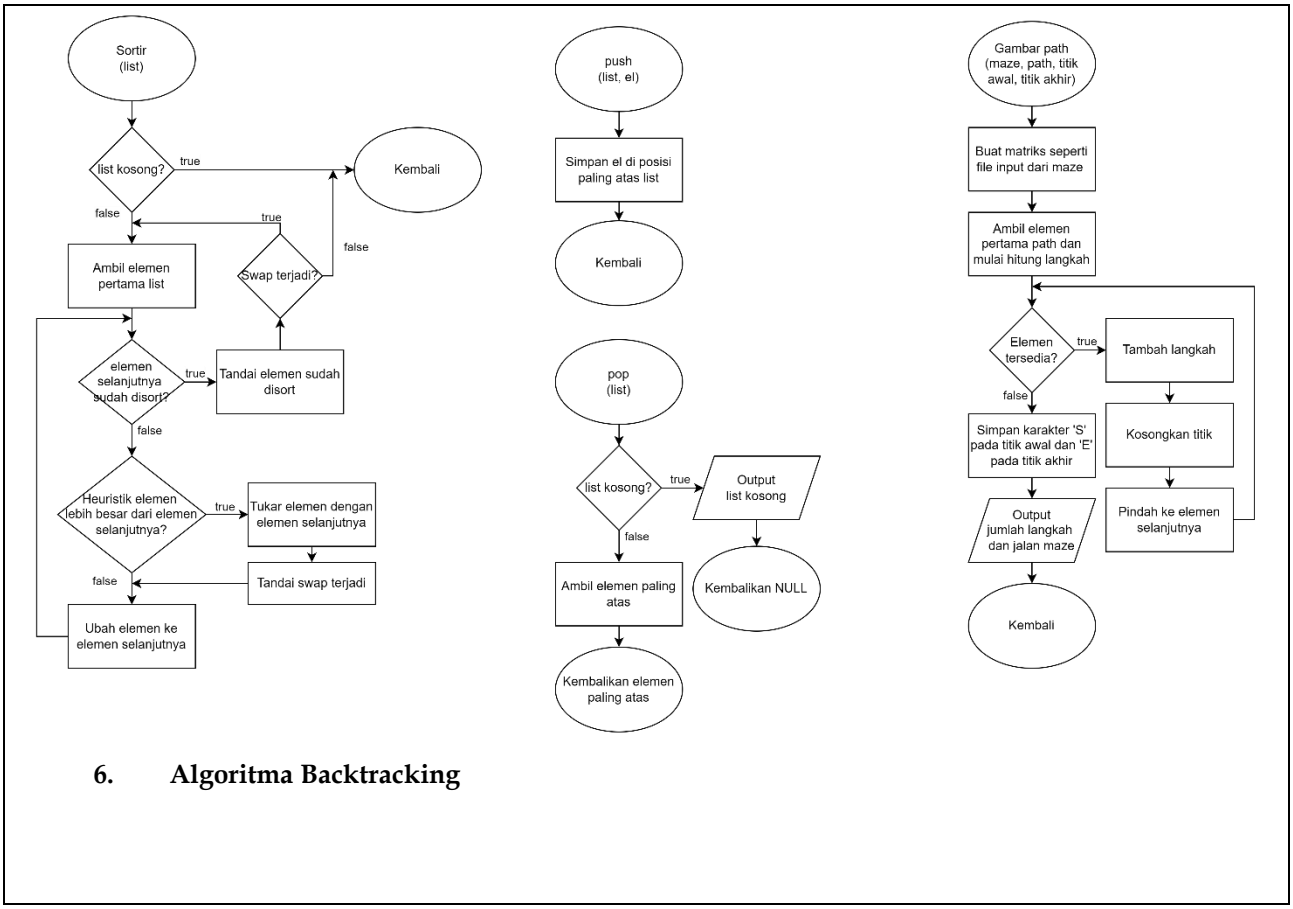

void DP
Inisialisasi tabel dp
Inisialisasi koordinat awal (X,Y)
Inisialisasi dx dan dy sebagai representasi arah gerak
Apakah titik tujuan berupa jalan maupun titik E?
False
True
Apakah panjang jalur terbaru lebih pendek dibanding sebelumnya?
False
True
Update tabel dp dengan panjang jalur dan posisi terbaru
False
Iterasi mencapai (baris x kolom) kali
return
void findAllPaths
Inisialisasi counter
False
Titik sekarang di titik E
True
counter ++
Pergerakan sebanyak 1 ke semua arah memenuhi ruang lingkup?
False
Pergerakan tidak dilakukan
True
Rekursi findAllPaths untuk tiap iterasi
return
void findAndPrintLongestPath
False
Titik sekarang di titik E
True
assign jalur diperoleh ke longestPath
Pergerakan sebanyak 1 ke semua arah memenuhi ruang lingkup?
False
Pergerakan tidak dilakukan
True
Rekursi findAllPaths untuk tiap iterasi
Iterasi jalur sekarang lebih panjang dari longestpath
True
Update longestpath menjadi jalur baru
False
return
void printShortest
Output
Panjang tabel dp di index terakhir
Setiap koordinat yang dilalui (jalur terpendek)
return
void readMatrix
Lakukan iterasi read tiap titik sel dalam matriks file
karakter = S
True
set koordinat awal di tempat S
karakter = E
True
set koordinat awal di tempat E
return
void getMatrixSize
inisialisasi nilai baris dan kolom = 0
Lakukan pembacaan file
Karakter terbaca adalah newline (\n)
False
True
jumlah baris ++
Karakter terbaca adalah bukan kosong
False
True
jumlah kolom ++
return



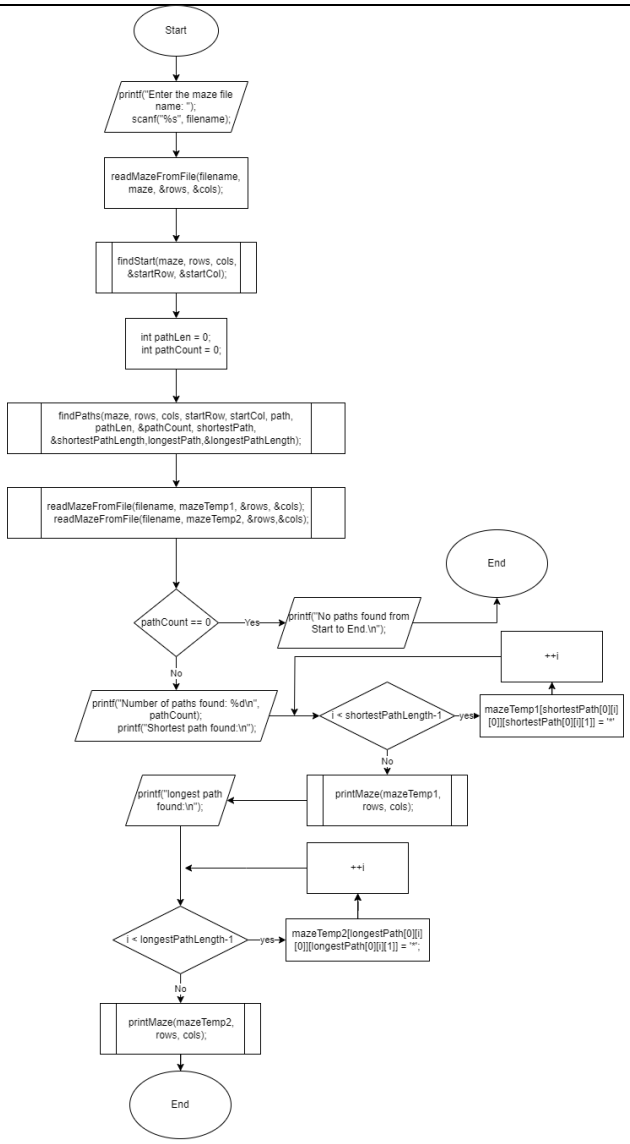
5. Algoritma A* (A-Star)


Mulai
Input nama file
Baca file (nama file)
Mulai timer
Cari jalan optimal (maze)
Akhiri timer
Output waktu pencarian jalan optimal
Selesai
Baca file (nama file)
File ada?
true
Output file tidak ditemukan
Selesai
false
File kosong?
false
Output file kosong
Selesai
true
Hitung ukuran maze
Maze segiempat?
false
Output bentuk maze salah
Selesai
true
Buat matriks kosong
Baca maze
A
A
Akhir file?
true
Tutup file
Kembali
false
Baris baru?
true
Pindah ke baris matriks selanjutnya
false
Titik awal?
true
Simpan titik matriks sebagai titik awal
false
Titik akhir?
true
Simpan titik sebagai titik akhir
false
Jalan?
true
Simpan titik sebagai jalan
false
Tembok?
true
Simpan titik sebagai tembok
false
Output terdapat karakter tidak valid
Selesai

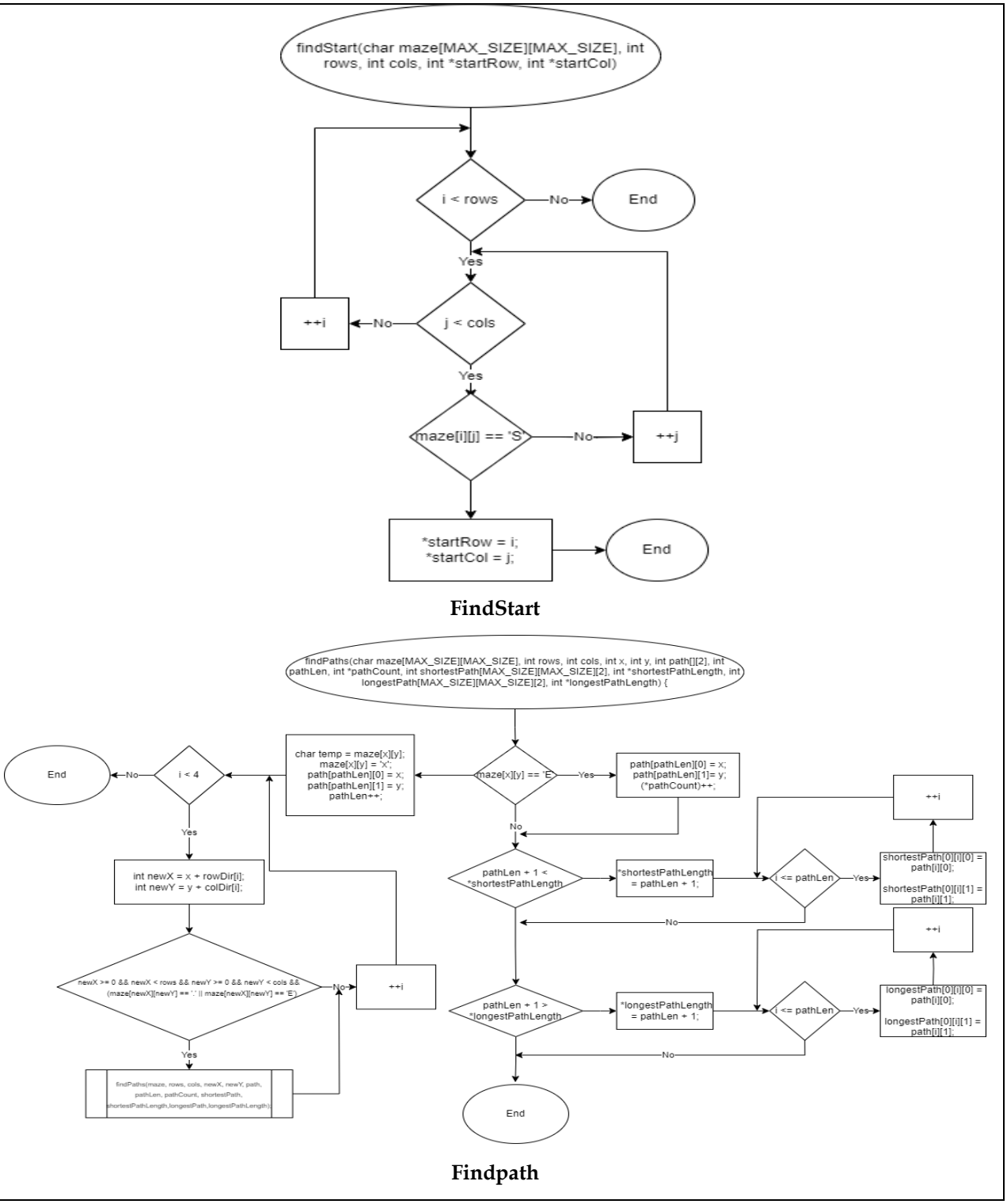

Cari jalan optimal (maze)
Inisialisasi hash table
Setup data titik awal pada hash table
Buat list ekspansi
push (list ekspansi, titik awal)
list ekspansi kosong?
true
false
sortir (list ekspansi)
Titik sudah diekspansi?
true
pop (list ekspansi)
false
Tandai titik sudah diekspansi
B
B
Semua titik tetangga sudah diperiksa?
true
C
false
Titik tetangga valid?
false
true
Titik berupa jalan dan belum diekspansi?
true
push (list ekspansi, titik tetangga)
false
Hitung jarak titik ke titik awal
Jarak lebih kecil dari sebelumnya?
true
Update parent dan heuristik
false
C
Titik akhir belum diekspansi?
true
Output tidak ada jalan
Return
false
Buat stack path
push (path, titik akhir)
Simpan titik akhir di titik sementara
Titik sementara bukan titik awal?
Gambar path (maze, path, titik awal, titik akhir)
Return
push (path, parent titik sementara)
Ubah titik sementara menjadi parent-nya

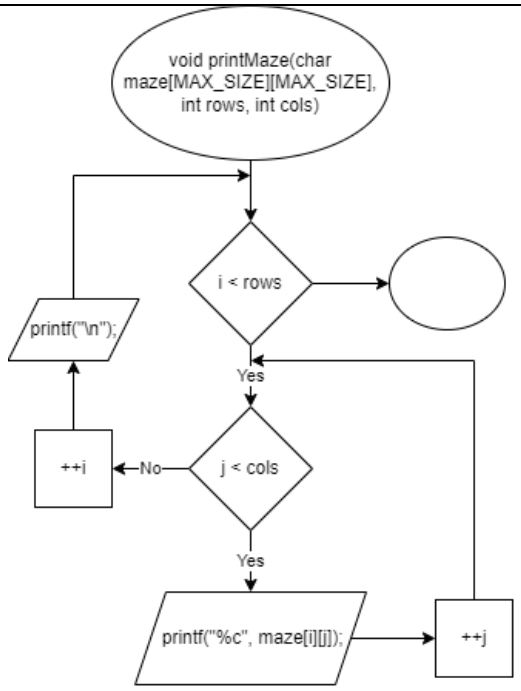


6. Algoritma Backtracking


Start
printf("Enter the maze file name: ");
scanf("%s", filename);
readMazeFromFile(filename, maze, &rows, &cols);
findStart(maze, rows, cols, &startRow, &startCol);
int pathLen = 0;
int pathCount = 0;
findPaths(maze, rows, cols, startRow, startCol, path, pathLen, &pathCount, shortestPath, &shortestPathLength,longestPath,&longestPathLength);
readMazeFromFile(filename, mazeTemp1, &rows, &cols);
readMazeFromFile(filename, mazeTemp2, &rows,&cols);
End
pathCount == 0
Yes
printf("No paths found from Start to End.\n");
No
++i
printf("Number of paths found: %d\n", pathCount);
printf("Shortest path found:\n");
i < shortestPathLength-1
yes
mazeTemp1[shortestPath[0][i][0]][shortestPath[0][i][1]] = '*'
No
printf("longest path found:\n");
printMaze(mazeTemp1, rows, cols);
++i
i < longestPathLength-1
yes
mazeTemp2[longestPath[0][i][0]][longestPath[0][i][1]] = '*';
No
printMaze(mazeTemp2, rows, cols);
End

Main

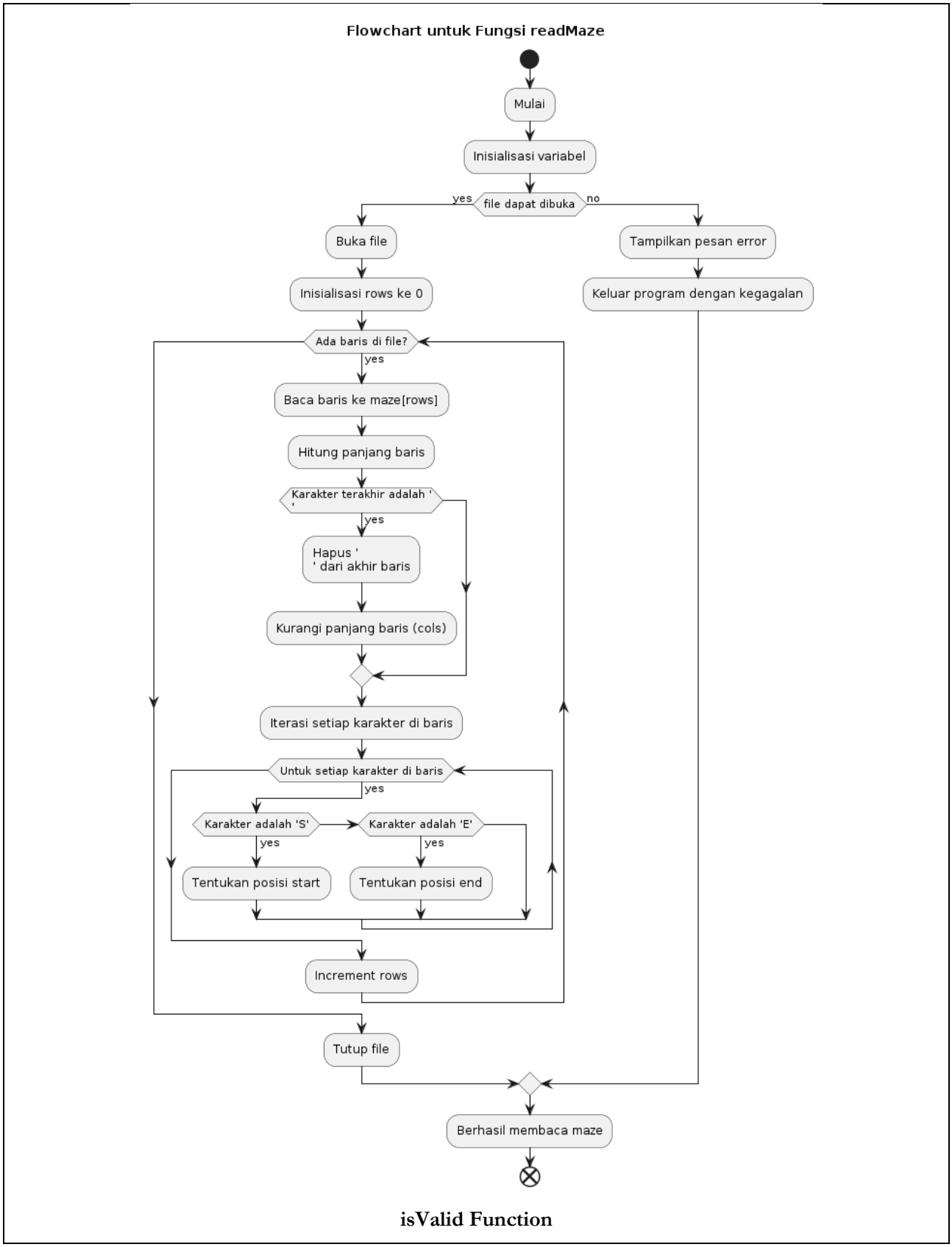

findStart(char maze[MAX_SIZE][MAX_SIZE], int rows, int cols, int *startRow, int *startCol)
i < rows
No
End
Yes
++i
No
j < cols
Yes
maze[i][j] == 'S'
No
++j
*startRow = i;
*startCol = j;
End
FindStart
findPaths(char maze[MAX_SIZE][MAX_SIZE], int rows, int cols, int x, int y, int path[][2], int pathLen, int *pathCount, int shortestPath[MAX_SIZE][MAX_SIZE][2], int *shortestPathLength, int longestPath[MAX_SIZE][MAX_SIZE][2], int *longestPathLength) {
End
No
i < 4
char temp = maze[x][y];
maze[x][y] = 'x';
path[pathLen][0] = x;
path[pathLen][1] = y;
pathLen++;
maze[x][y] == 'E'
Yes
path[pathLen][0] = x;
path[pathLen][1]= y;
(*pathCount)++;
Yes
int newX = x + rowDir[i];
int newY = y + colDir[i];
No
pathLen + 1 < *shortestPathLength
*shortestPathLength = pathLen + 1;
i <= pathLen
Yes
shortestPath[0][i][0] = path[i][0];
shortestPath[0][i][1] = path[i][1];
++i
No
newX >= 0 && newX < rows && newY >= 0 && newY < cols && (maze[newX][newY] == '.' || maze[newX][newY] == 'E')
No
++i
Yes
findPaths(maze, rows, cols, newX, newY, path, pathLen, pathCount, shortestPath, shortestPathLength,longestPath,longestPathLength);
pathLen + 1 > *longestPathLength
*longestPathLength = pathLen + 1;
i <= pathLen
Yes
longestPath[0][i][0] = path[i][0];
longestPath[0][i][1] = path[i][1];
++i
No
End
Findpath



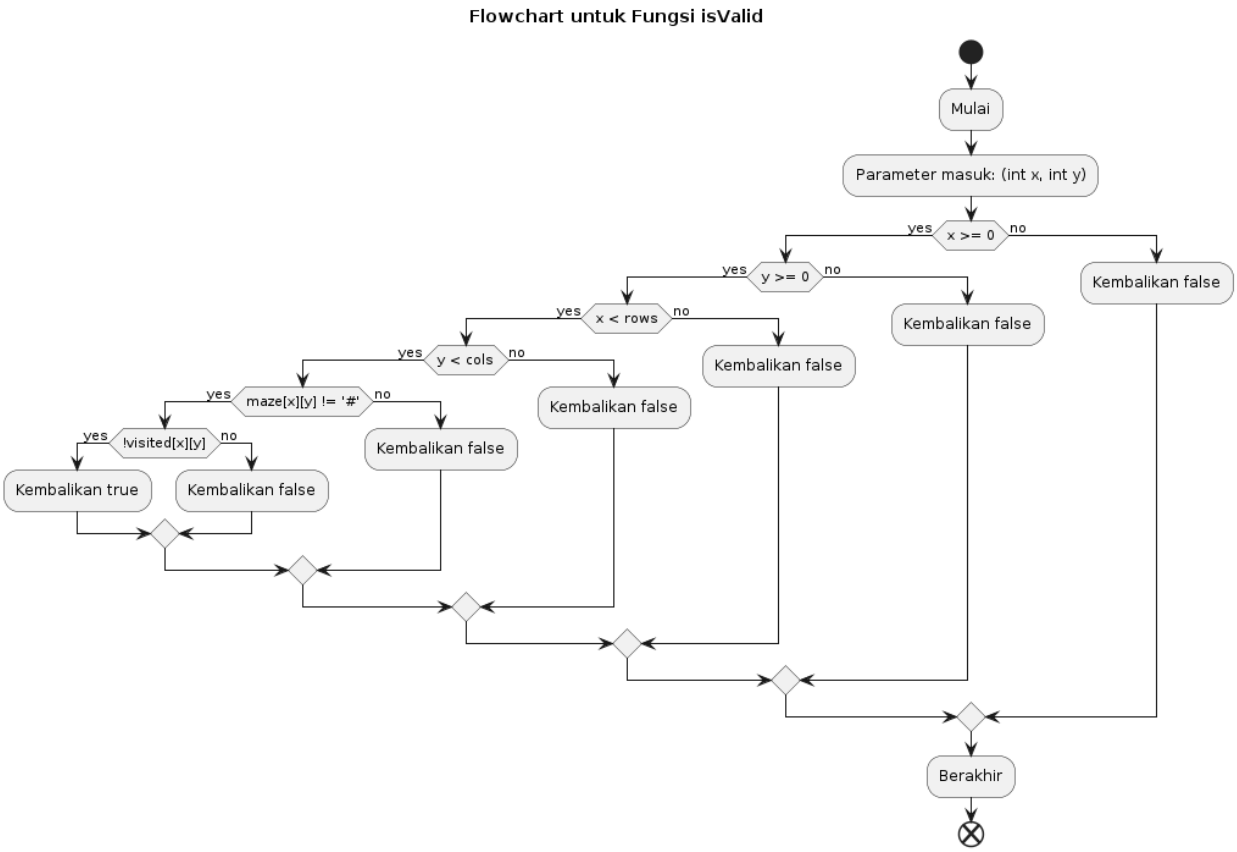
Findmaze

7. Algoritma Greedy

readMaze function

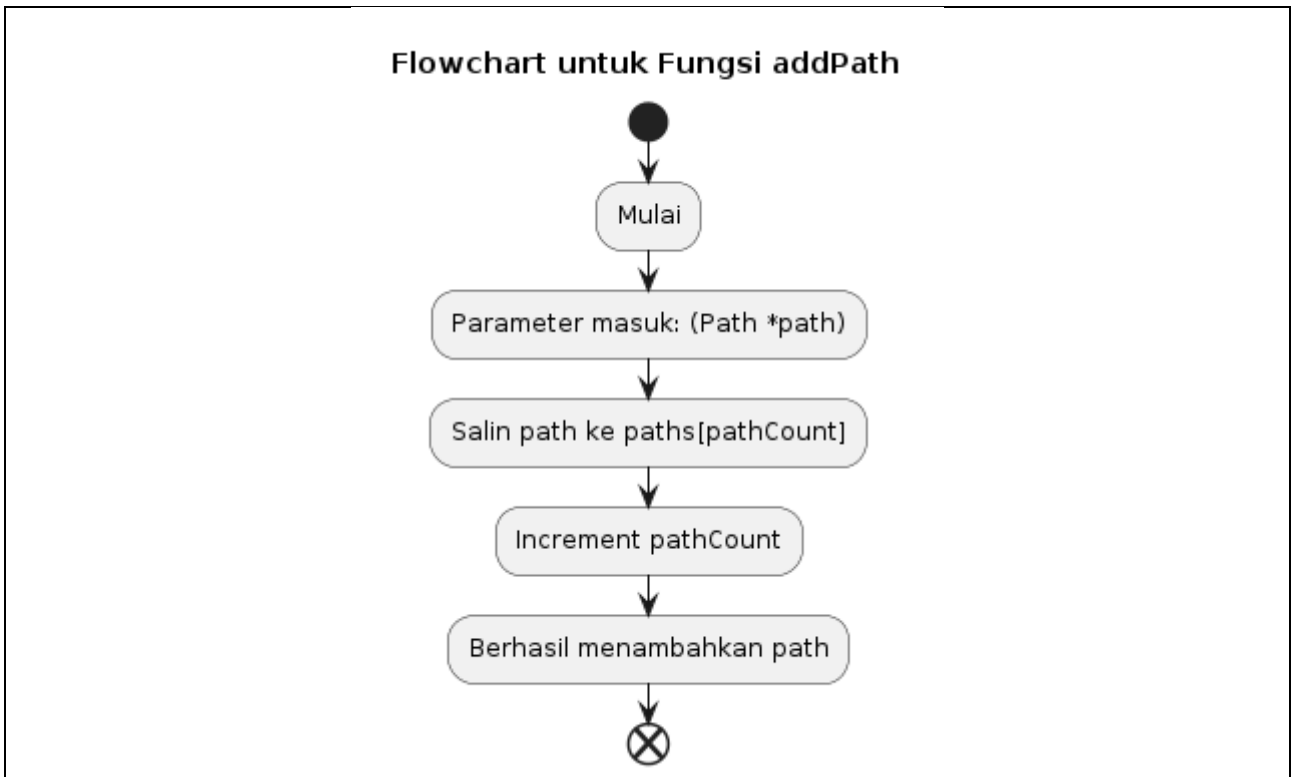


isValid Function

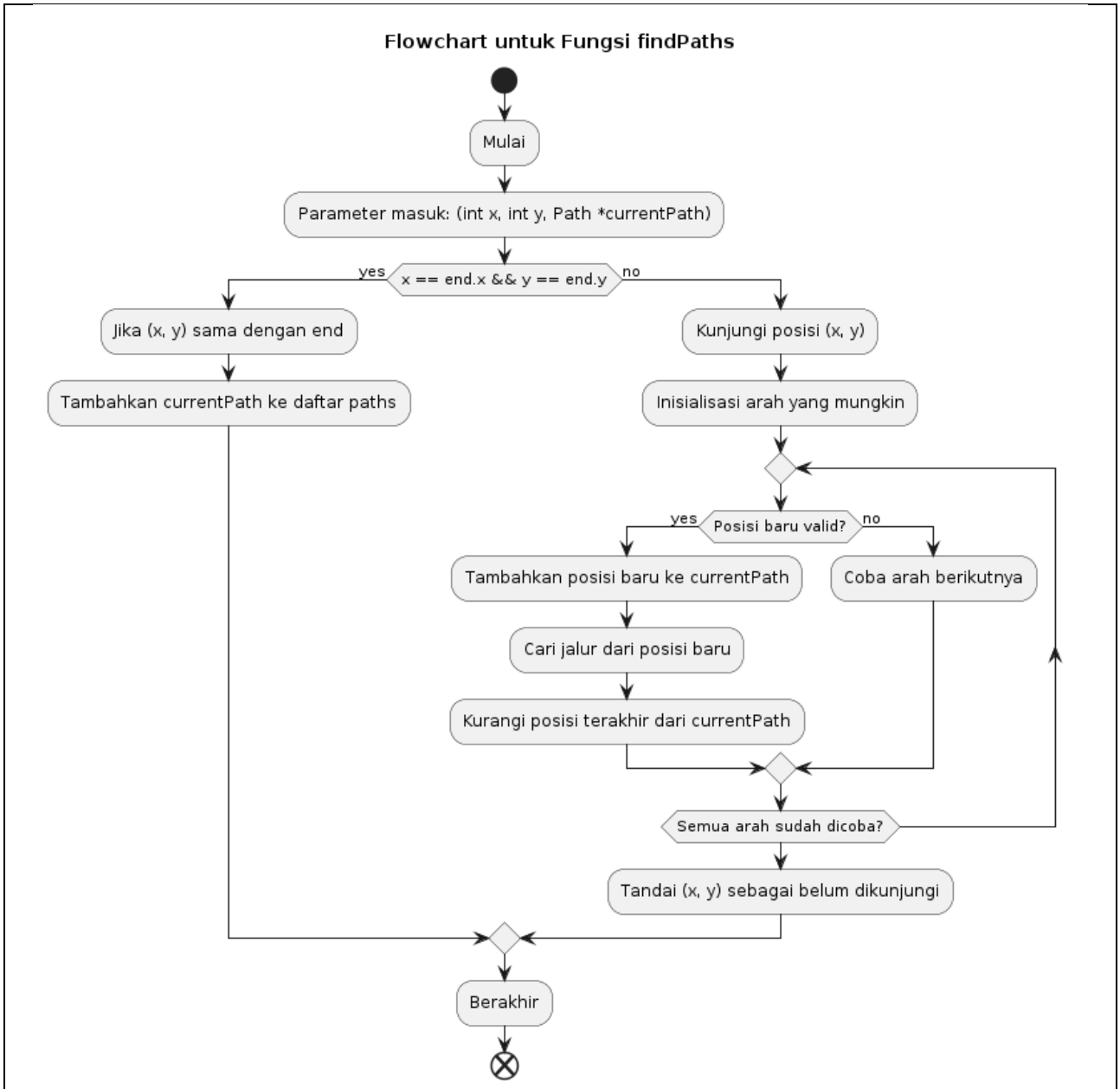


copyPath Function

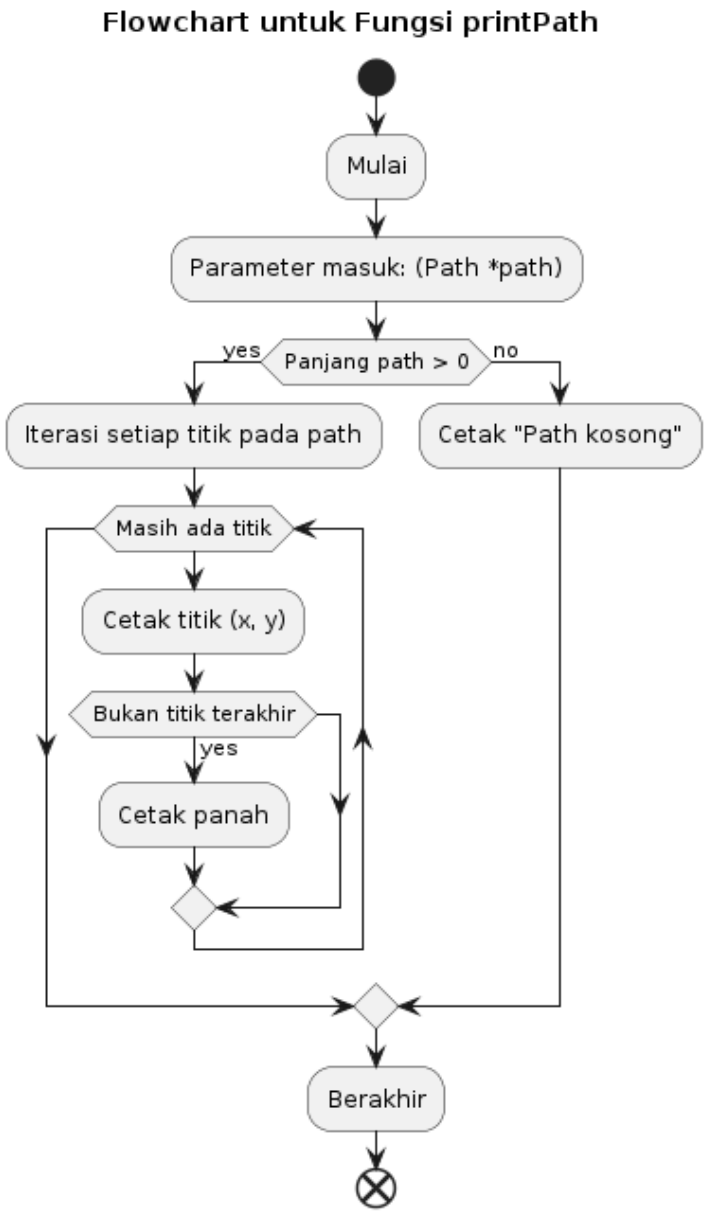
addPath Function



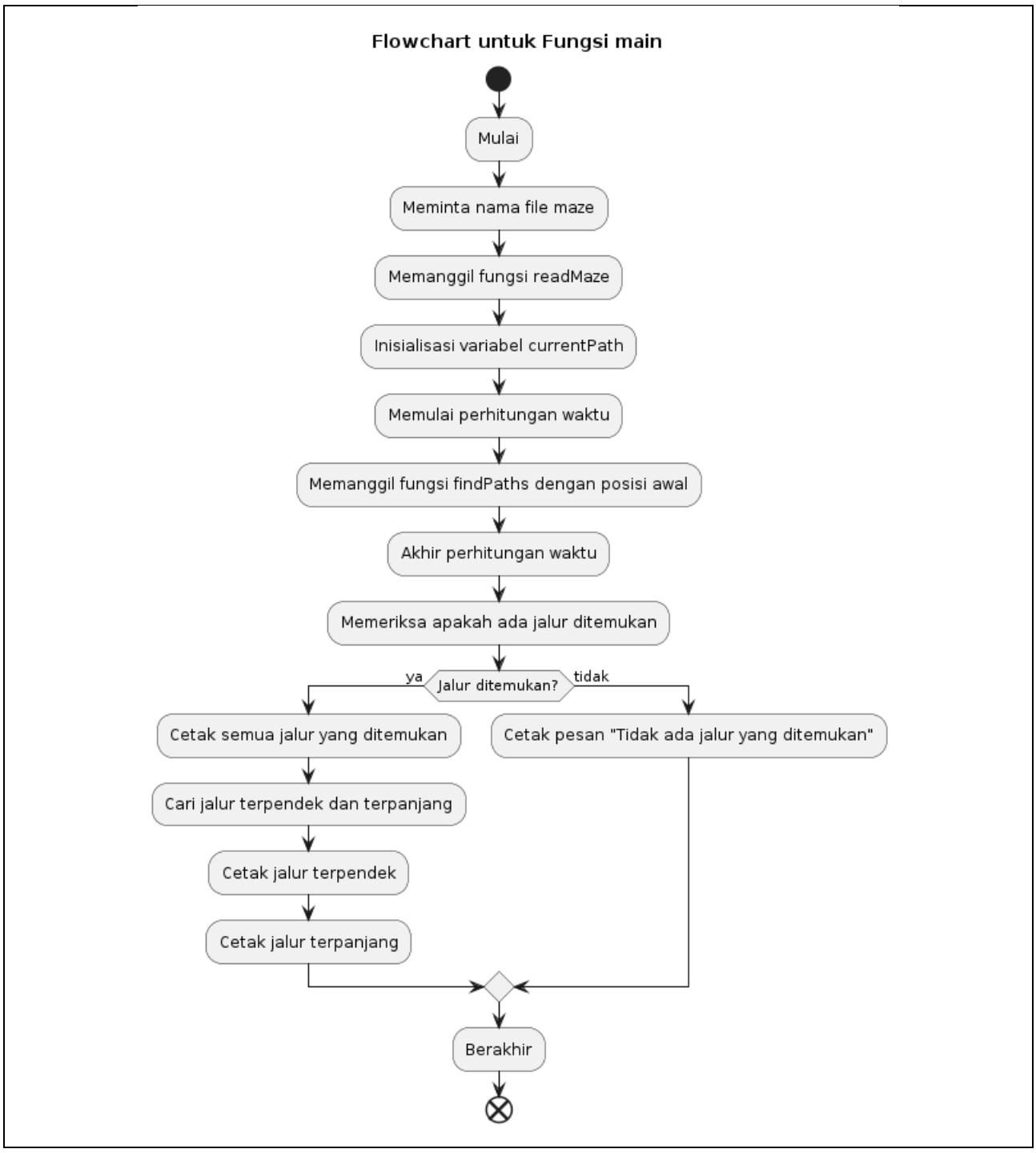
findPaths Function



printPath Function

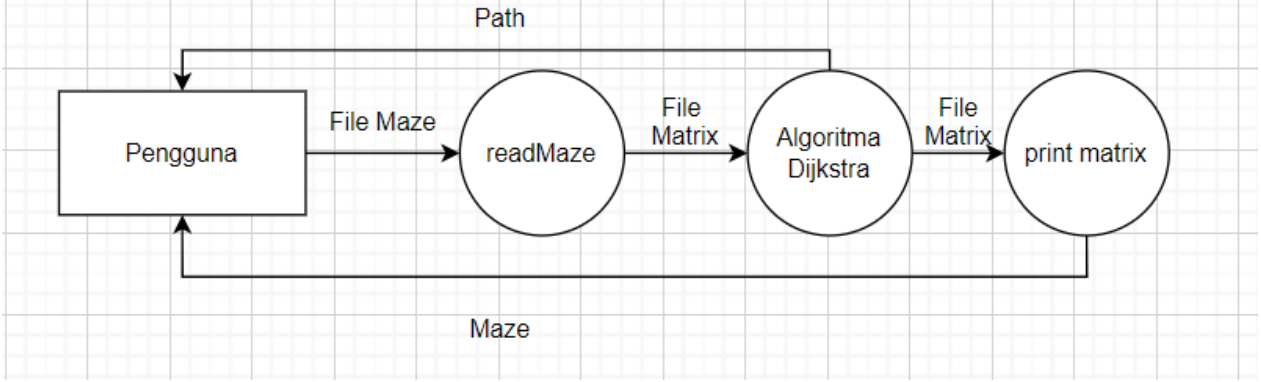


main Function

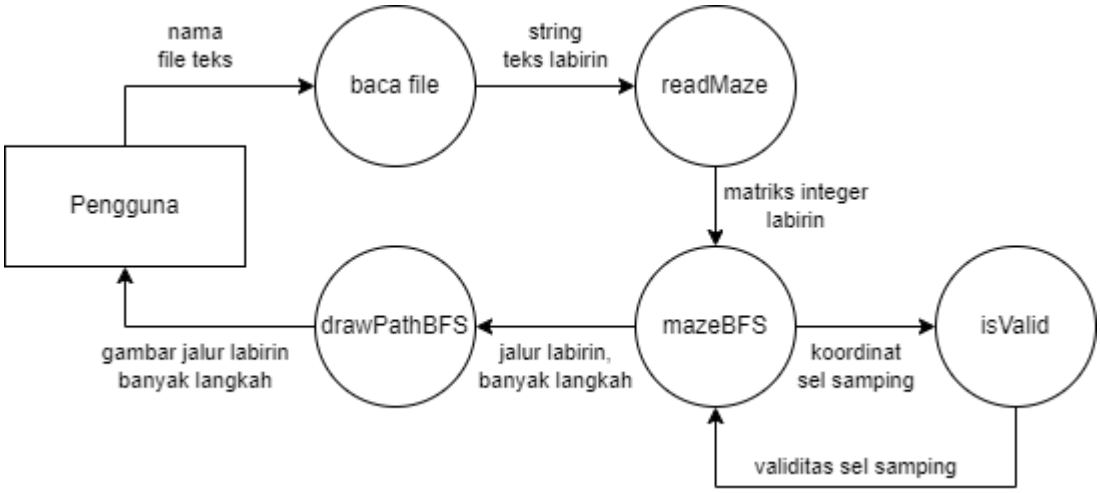


DFD

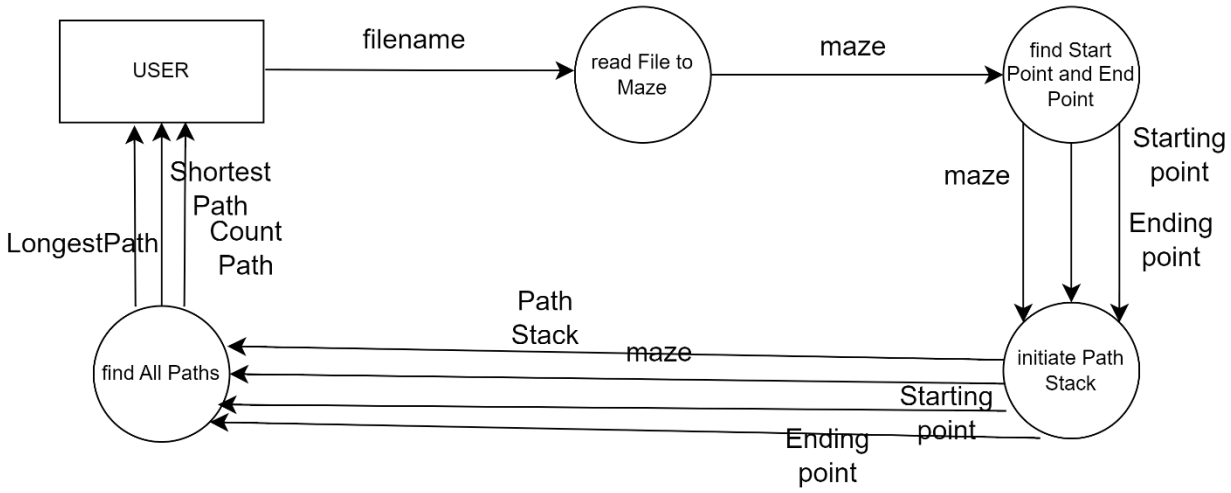
1. Algoritma Djikkstra



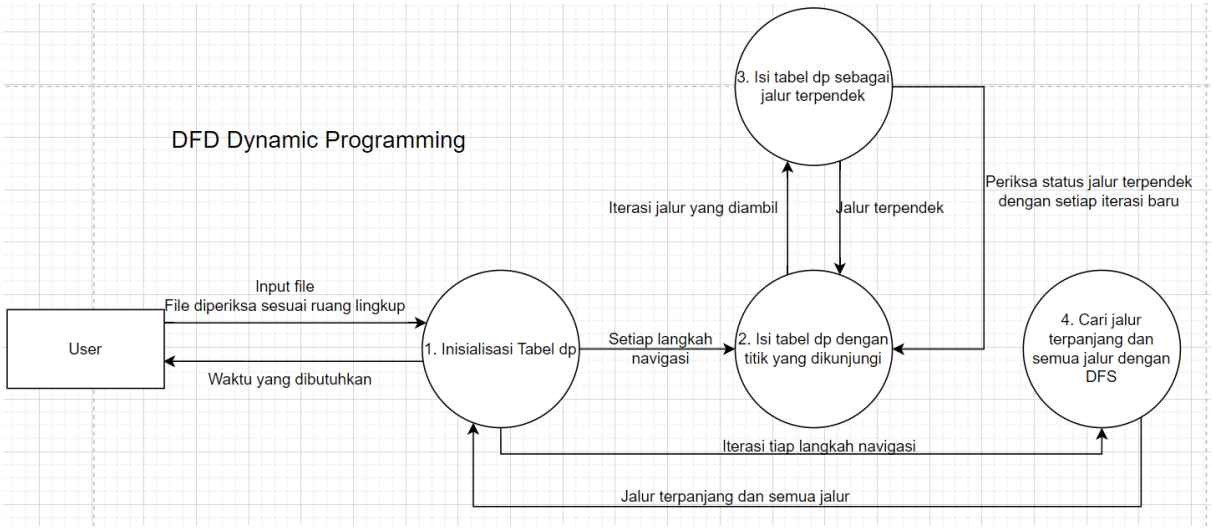
2. Breadth First Search (BFS)



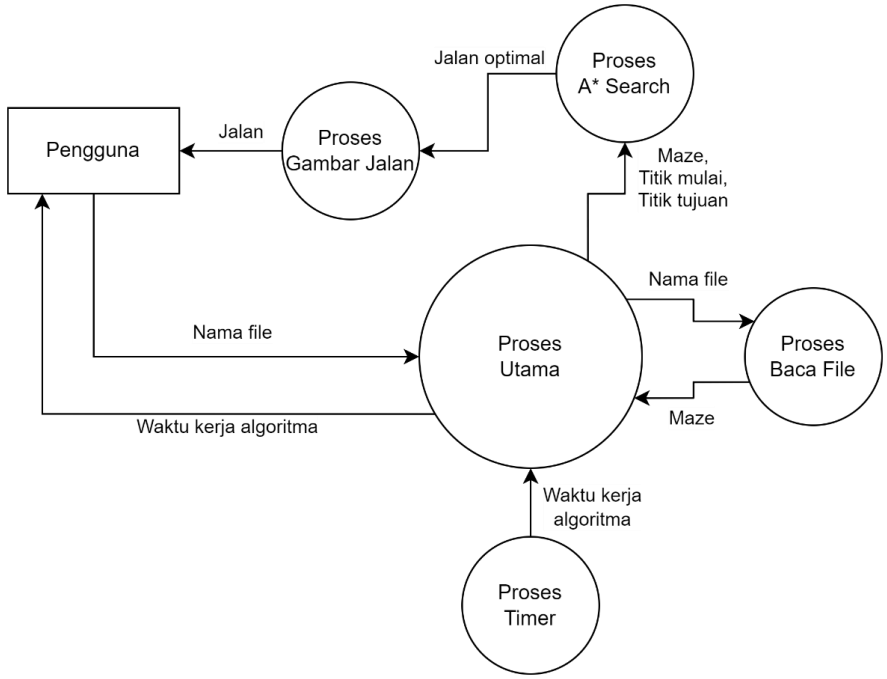
3. Depth First Search (DFS)



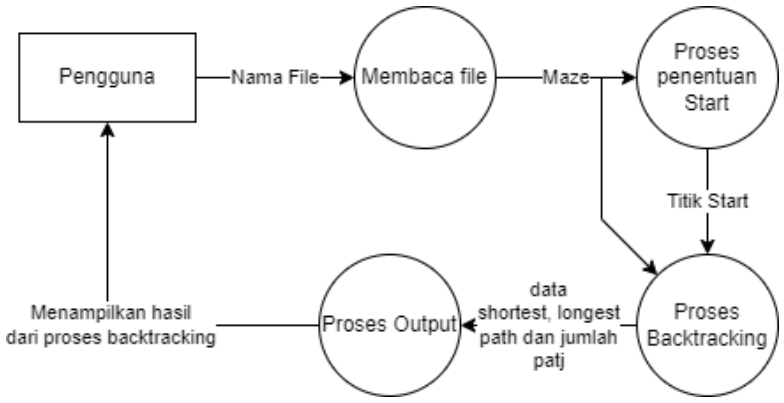
4. Dynamic Programming



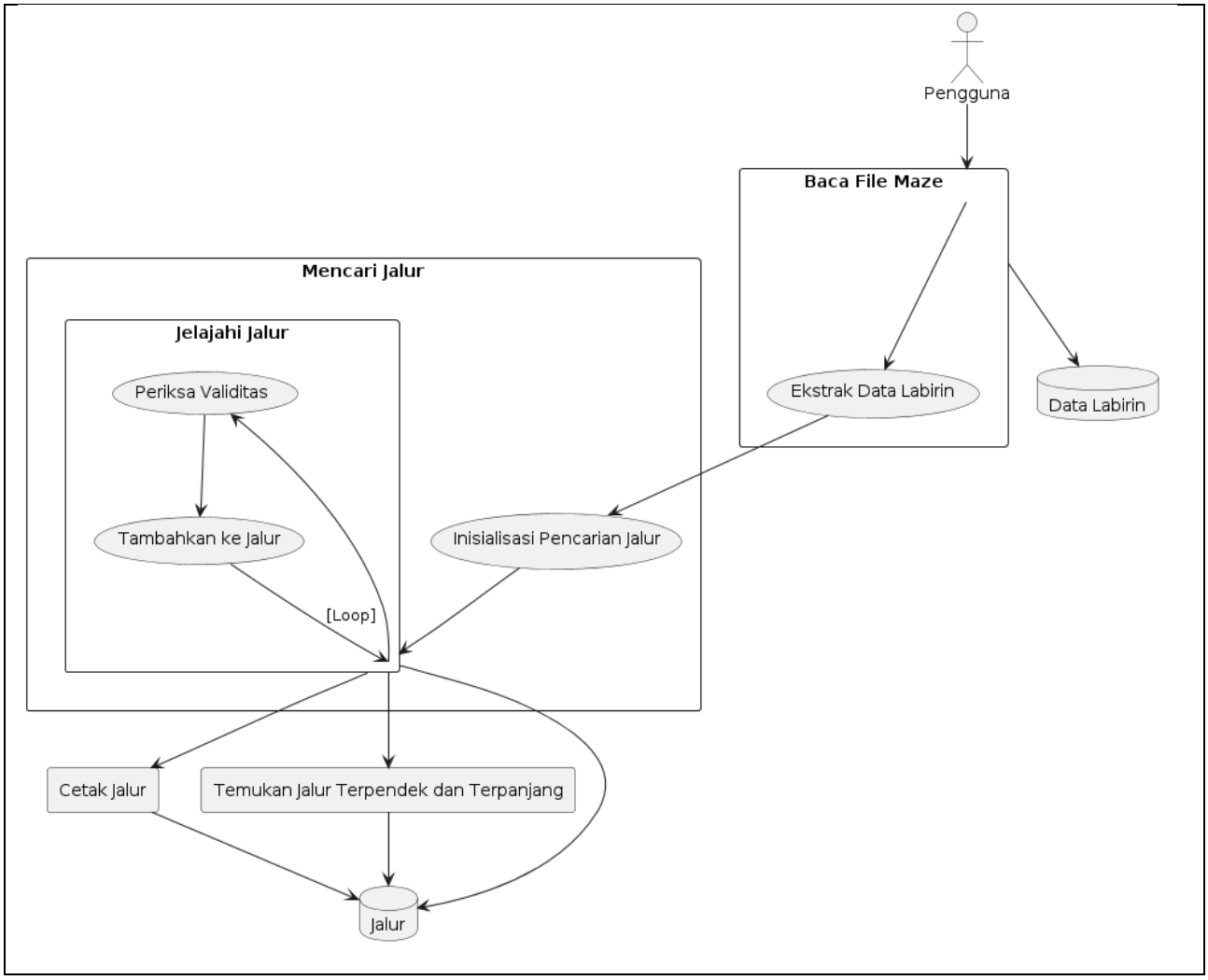
5. Algoritma A* (A-Star)



6. Algoritma Backtracking



7. Algoritma Greedy


Pengguna
Baca File Maze
Ekstrak Data Labirin
Data Labirin
Mencari Jalur
Jelajahi Jalur
Periksa Validitas
Tambahkan ke Jalur
[Loop]
Inisialisasi Pencarian Jalur
Cetak Jalur
Temukan Jalur Terpendek dan Terpanjang
Jalur

Hasil output program

1. Algoritma Djikstra

```
Path 81: (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 3) -> (3, 3)
 -> (3, 4) -> (3, 5) -> (3, 6) -> (4, 6) -> (4, 7) -> (4, 8) -> (5, 8) -> (
6, 8) -> (7, 8) -> (8, 8) -> (9, 8) -> (9, 9)

Jumlah jalur terpendek tersedia : 81
```

2. Breadth First Search (BFS)

```
Masukkan nama file input: maze_1.txt
Waktu yang diperlukan: 0.037000 ms

Jalur:
S + + . . # . . # . . .
# # + # . # . # . . . #
. . + + # . # . # . . .
. # . + + + + # . # . #
. # . # . . + + + + # #
. . . # . . . # . + # .
. # . . . # . # . + . #
. # . # . . . . . + . .
. # . # . # . # . + . .
. . . . # . # . . E . #
. . . . . . . . . . # #
# # # # # # # # # # # #
Banyaknya langkah: 18
```

```
Masukkan nama file input: maze_2.txt
Waktu yang diperlukan: 0.020000 ms

Jalur:
. . . . S + # . . . . .
# . . . # + # . # . # .
# . . . # + + + # . # .
. . . . . . . + + + + +
# . . . # . . . . . # +
# . # . . . # . # # . +
. . . . # . # . . . . E
. . . . . . . . # . . .
. . . . . # # . . # . .
. # . . . . . . . . . .
# # # # # # # # # # # #
Banyaknya langkah: 13
```

```
Masukkan nama file input: maze_3.txt
Waktu yang diperlukan: 0.040000 ms

Jalur:
. S # # # # # # # # # # # # # # # # # #
. + # . . . . . # . . . . . . . . . . #
# + # . # # # . # # # . # # # . # # # . #
# + . . # . . . . . # . # . . . . . # . #
# + # . # # # # # # # . # . # . # # # . #
# + # . . . # . . . . . # . # . # . . . #
# + # # # # # # # # # # # # # . # # # . #
# + + + # . . . . . . . # . . . . . # . #
# # # + # . # # # # # # # # # . # # # . #
# + + + . . # . . . . . . . . . . . # . #
# + # # # # # # # # # # # # # # . # # # . #
# + . . # . . + + + # . . . # . # . . . #
# + # # # # # # + # + # # # . # # # . # . #
# + # . # . # + # + . . # . . . # . # . #
# + # . # . # + # + # # # . # # # . # # #
# + # + + + + + # + + + . . # . . . . . #
# + # + # # # # # # # + # # # # # # # . #
# + + + # . # . . . . + # . # . . . # . #
# . # # # . # # # # # + # . # . # # # . #
# . . . . . . . # . . + + + + + + + + + #
# # # # # # # # # # # # # # # # # # # E #
# # # # # # # # # # # # # # # # # # # # #
Banyaknya langkah: 54
```

```
Masukkan nama file input: maze_5.txt
Waktu yang diperlukan: 0.024000 ms

Jalur:
S + + + + + + + + + . .
. . . . . . . . . . + . .
. . . . . . . . . . + . .
. . . . . . . . . . + . .
. . . . . . . . . . + . .
. . . . . . . . . . + . .
. . . . . . . . . . + . .
. . . . . . . . . . + . .
. . . . . . . . . . + . .
. . . . . . . . . . E . .
. . . . . . . . . . . . .
# # # # # # # # # # # #
Banyaknya langkah: 18
```

```
Masukkan nama file input: maze_4.txt
Waktu yang diperlukan: 0.021000 ms
Tidak terdapat jalur yang menghubungkan titik awal dan titik akhir di dalam labirin
```

```
Masukkan nama file input: maze_6.txt
Waktu yang diperlukan: 0.118000 ms

Jalur:
S + # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
. + # . . . . . . . # . . . # . # . . . . . . . # . . . . . . . # . . . # . . . #
# + # . # . # # # . # . # # # . # # # . # . # . # . # . # . # # # . # # # . # . #
# + # . # . . . # . # . . . . . . . # . # . # . # . # . # . # . . . . . # . # . #
# + # . # # # # # # # . # . # # # . # # # . # # # # # . # . # # # # # . # # # . #
# + # . # . . . # . . . # . # . # . . . . . # . . . . . # . . . # . # . . . # . #
# + # . # . # # # . # . # # # . # # # # # . # # # # # . # # # # # . # # # . # . #
# + + + + + # . . . # . # . . . . . . . . . . . # . # . . . # . # . . . # . . . #
# . # . # + # . # . # # # . # . # # # # # . # # # . # # # . # . # # # . # # # . #
# . # . # + # . # . # . . . # . . . # . # . . . # . . . # . . . # . . . . . . . #
# # # # # + # # # # # # # # # . # # # . # . # # # # # . # . # # # # # # # . # . #
# . . + + + . . # . . . . . # . . . . . # . # . . . . . . . # . . . # . . . # . #
# . # + # # # # # # # # # . # . # # # # # # # . # # # # # . # . # # # . # . # . #
# . # + + + + + # . . . # . # . . . # . . . # . . . # . . . . . . . # . # . # . #
# # # # # . # + # . # # # . # . # . # . # # # . # . # # # # # . # # # . # # # # #
# . . . . . # + + + # . . . # . # . # . . . . . # . . . . . # . # . . . . . . . #
# . # # # # # . # + # . # . # . # . # # # . # # # # # # # # # . # . # . # # # . #
# . . . . . # . # + # . # . # . # . . . . . . . . . . . # . # . . . # . . . # . #
# . # . # # # # # + # . # . # . # # # . # . # . # # # . # . # . # # # . # # # # #
# . # . # . # . . + # . # . # . # . # . # . # . # . # . # . . . # . # . . . . . #
# # # # # . # . # + # . # # # # # . # . # # # # # . # # # . # . # . # # # . # . #
# . . . # . . . # + + + + + + + + + # . # . . . # . # . . . # . . . # . . . # . #
# . # # # # # . # . # # # . # . # + # . # # # . # . # . # # # # # # # . # . # . #
# . . . . . . . # . # . # . # . # + + + + + + + + + # . # . . . . . . . # . # . #
# . # . # . # # # . # . # # # . # # # # # . # # # + # # # # # . # # # . # . # # #
# . # . # . # . . . . . # . . . # . . . # . # . # + # . . . . . # . . . # . . . #
# # # # # # # # # # # . # . # # # # # . # . # . # + # # # . # . # . # # # . # # #
# . # . . . . . . . # . # . . . . . . . # . # . # + . . # . # . # . # . # . # . #
# . # # # . # # # # # # # # # # # # # . # # # . # + # . # . # # # # # . # . # . #
# . . . . . # . . . . . . . # . . . . . . . # + + + # . # . . . # . # . # . # . #
# # # . # . # # # # # . # # # # # # # . # # # + # # # # # . # # # . # . # . # . #
# . . . # . . . . . . . . . # . . . . . . . # + + + + + + + . . # . . . . . . . #
# # # . # # # . # . # . # . # . # . # # # # # . # # # # # + # # # # # . # . # . #
# . . . # . # . # . # . # . # . # . . . . . . . . . . . # + + + # . # . # . # . #
# . # # # . # # # # # . # # # . # # # # # # # . # # # # # # # + # . # # # # # # #
# . # . . . # . . . # . . . . . . . . . . . # . . . . . # . # + + + + + + + # . #
# . # # # . # . # # # . # # # # # # # # # # # . # # # . # . # . # # # # # + # . #
# . # . . . . . # . . . # . # . # . . . . . . . . . # . # . . . . . # . . + + + #
# . # . # . # # # . # . # . # . # # # . # # # . # # # # # # # # # . # # # . # + #
# . # . # . . . . . # . . . . . # . . . # . . . . . . . # . . . . . # . . . # + #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # E #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
Banyaknya langkah: 87
```

3. Depth First Search (DFS)

```
Enter the filename of the maze: maze_3.txt
Number of paths: 2
Length of shortest path: 54
Shortest path from start to end:
Path: (0, 1) -> (1, 1) -> (2, 1) -> (3, 1) -> (4, 1) -> (5, 1) ->
(6, 1) -> (7, 1) -> (7, 2) -> (7, 3) -> (8, 3) -> (9, 3) -> (9, 2)
 -> (9, 1) -> (10, 1) -> (11, 1) -> (12, 1) -> (13, 1) -> (14, 1)
-> (15, 1) -> (16, 1) -> (17, 1) -> (17, 2) -> (17, 3) -> (16, 3)
-> (15, 3) -> (15, 4) -> (15, 5) -> (15, 6) -> (15, 7) -> (14, 7)
-> (13, 7) -> (12, 7) -> (11, 7) -> (11, 8) -> (11, 9) -> (12, 9)
-> (13, 9) -> (14, 9) -> (15, 9) -> (15, 10) -> (15, 11) -> (16, 1
1) -> (17, 11) -> (18, 11) -> (19, 11) -> (19, 12) -> (19, 13) ->
(19, 14) -> (19, 15) -> (19, 16) -> (19, 17) -> (19, 18) -> (19, 1
9)
Length of longest path: 56
Longest path from start to end:
Path: (0, 1) -> (0, 0) -> (1, 0) -> (1, 1) -> (2, 1) -> (3, 1) ->
(4, 1) -> (5, 1) -> (6, 1) -> (7, 1) -> (7, 2) -> (7, 3) -> (8, 3)
 -> (9, 3) -> (9, 2) -> (9, 1) -> (10, 1) -> (11, 1) -> (12, 1) ->
 (13, 1) -> (14, 1) -> (15, 1) -> (16, 1) -> (17, 1) -> (17, 2) ->
 (17, 3) -> (16, 3) -> (15, 3) -> (15, 4) -> (15, 5) -> (15, 6) ->
 (15, 7) -> (14, 7) -> (13, 7) -> (12, 7) -> (11, 7) -> (11, 8) ->
 (11, 9) -> (12, 9) -> (13, 9) -> (14, 9) -> (15, 9) -> (15, 10) -
> (15, 11) -> (16, 11) -> (17, 11) -> (18, 11) -> (19, 11) -> (19,
 12) -> (19, 13) -> (19, 14) -> (19, 15) -> (19, 16) -> (19, 17) -
> (19, 18) -> (19, 19)
Time taken to solve the maze: 0.000000 milliseconds
```

```
Enter the filename of the maze: maze_1.txt
Number of paths: 502020
Length of shortest path: 18
Shortest path from start to end:
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)
Length of longest path: 62
Longest path from start to end:
Path: (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 1) ->
(2, 0) -> (3, 0) -> (4, 0) -> (5, 0) -> (6, 0) -> (7, 0) -> (8, 0)
 -> (9, 0) -> (10, 0) -> (10, 1) -> (9, 1) -> (9, 2) -> (10, 2) ->
 (10, 3) -> (10, 4) -> (10, 5) -> (10, 6) -> (10, 7) -> (9, 7) ->
(9, 8) -> (8, 8) -> (8, 9) -> (7, 9) -> (7, 8) -> (7, 7) -> (7, 6)
 -> (6, 6) -> (5, 6) -> (5, 5) -> (4, 5) -> (4, 4) -> (5, 4) -> (6
, 4) -> (6, 3) -> (6, 2) -> (5, 2) -> (4, 2) -> (3, 2) -> (3, 3) -
> (3, 4) -> (3, 5) -> (3, 6) -> (4, 6) -> (4, 7) -> (4, 8) -> (4,
9) -> (5, 9) -> (5, 8) -> (6, 8) -> (6, 9) -> (6, 10) -> (7, 10) -
> (7, 11) -> (8, 11) -> (8, 10) -> (9, 10)
Time taken to solve the maze: 448.944000 milliseconds
```

4. Dynamic Programming

```
Masukkan nama file : maze_1.txt
Jalur terpendek sepanjang : 18
Jalur terpendek yang ditempuh :
(0, 1)
(0, 2)
(1, 2)
(2, 2)
(2, 3)
(3, 3)
(3, 4)
(3, 5)
(3, 6)
(4, 6)
(4, 7)
(4, 8)
(4, 9)
(5, 9)
(6, 9)
(7, 9)
(8, 9)
(9, 9)
```

```
Jalur terpanjang sepanjang : 63
Jalur terpanjang yang ditempuh :
(0, 0)
(0, 1)
(0, 2)
(1, 2)
(2, 2)
(2, 1)
(2, 0)
(3, 0)
(4, 0)
(5, 0)
(5, 1)
(5, 2)
(4, 2)
(3, 2)
(3, 3)
(3, 4)
(3, 5)
(3, 6)
(4, 6)
(4, 7)
(4, 8)
(4, 9)
(5, 9)
(5, 8)
(6, 8)
(6, 9)
(6, 10)
(7, 10)
(7, 11)
(8, 11)
(8, 10)
(8, 9)
(8, 8)
(7, 8)
(7, 7)
(7, 6)
(6, 6)
(5, 6)
(5, 5)
(4, 5)
(4, 4)
(5, 4)
(6, 4)
(6, 3)
(6, 2)
(7, 2)
(8, 2)
(9, 2)
(9, 1)
(9, 0)
(10, 0)
(10, 1)
(10, 2)
(10, 3)
(10, 4)
(10, 5)
(10, 6)
(10, 7)
(9, 7)
(9, 8)
(10, 8)
(10, 9)
(9, 9)
Total jalur yang ditemukan: 502020
Waktu yang dibutuhkan : 402 ms
```

5. Algoritma A* (A-Star)

```
Masukkan nama file input: ../maps/maze_1.txt
Jalan optimal:
Jumlah langkah: 18
S     . . # . . # . . .
# #   # . # . # . . . #
. .     # . # . # . . .
. # .         # . # . #
. # . # . .           # #
. . . # . . . # .     # .
. # . . . # . # .     . #
. # . # . . . . .     . .
. # . # . # . # .     . .
. . . . # . # . . E . #
. . . . . . . . . . # #
Time: 7.669000 ms
```

```
Masukkan nama file input: ../maps/maze_2.txt
Jalan optimal:
Jumlah langkah: 13
. . . . S   # . . . . .
# . . . #   # . # . # .
# . . . #       # . # .
. . . . . . .
# . . . # . . . . . #
# . # . . . # . # # .
. . . . # . # . . . . E
. . . . . . . . # . . .
. . . . . # # . . # . .
. # . . . . . . . . . .
Time: 14.938000 ms
```

```
Masukkan nama file input: ../maps/maze_3.txt
Jalan optimal:
Jumlah langkah: 54
. S # # # # # # # # # # # # # # # # # # #
.   # . . . . . . . # . . . . . . . . . #
#   # . # # # . # # # . # # # . # # # . #
#   . . # . . . . . # . # . . . . . # . #
#   # . # # # # # # # . # . # . # # # . #
#   # . . . # . . . . . # . # . # . . . #
#   # # # # # # # # # # # # # . # # # . #
#       # . . . . . . . # . . . . . # . #
# # #   # . # # # # # # # # # . # # # . #
#       . . # . . . . . . . . . . . # . #
#   # # # # # # # # # # # # # # . # # # . #
#   . . # . .       # . . . # . # . . . #
#   # # # # #   #   # # # . # # # . # . #
#   # . # . #   #     . . # . . . # . # . #
#   # . # . #   #   # # # . # # # . # # #
#   #           #       . . # . . . . . #
#   #   # # # # # # #   # # # # # # # . #
#       # . # . . . .   # . # . . . # . #
# . # # # . # # # # #   # . # . # # # . #
# . . . . . . . # . .                   #
# # # # # # # # # # # # # # # # # # E #
Time: 23.645000 ms
```

```
Masukkan nama file input: ../maps/maze_5.txt
Jalan optimal:
Jumlah langkah: 18
S                   . .
. . . . . . . . . .   . .
. . . . . . . . . .   . .
. . . . . . . . . .   . .
. . . . . . . . . .   . .
. . . . . . . . . .   . .
. . . . . . . . . .   . .
. . . . . . . . . .   . .
. . . . . . . . . .   . .
. . . . . . . . . . E . .
. . . . . . . . . . . . .
Time: 9.131000 ms
```

```
Masukkan nama file input: ../maps/maze_4.txt

Jalan tidak ditemukan!
Time: 0.914000 ms
```

```
Masukkan nama file input: ../maps/maze_6.txt
Jalan optimal:
Jumlah langkah: 87
S   # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
.   # . . . . . . . # . . . # . # . . . . . . . # . . . . . . . # . . . # . . . #
#   # . # . # # # # . # . # # # . # # # . # . # . # . # . # . # # # . # # # . # . #
#   # . # . . . # . # . . . . . . . # . # . # . # . # . # . . . . . # . # . # . #
#   # . # # # # # # . # . # # # . # # # . # # # # # . # . # # # # # . # . # . # . #
#   # . # . . . # . . . # . # . . . . . # . . . . # . . . # . # . . . # . # . # . #
#   # . # . # # # # . # . # # # . # # # # # . # # # # # . # # # # # . # # # . # . #
#           # . . . # . # . . . . . . . . . . . # . # . . . # . # . . . # . # . . . #
# . # . #   # . # . # # # . # . # # # # # . # # # . # # # . # . # # # . # . # . # . #
# . # . #   # . # . # . . . # . . . # . # . . . # . . . # . # . . . # . # . . . . . #
# # # # #   # # # # # # # # # . # # # . # . # # # # # . # . # # # # # # # . # . # . #
# . .       . . # . . . . . # . # . . . . . # . . . # . # . . . # . # . . . # . # . #
# . #   # # # # # # # # # . # . # # # # # # # . # # # # # . # . # # # . # . # . # . #
# . #           # . . . # . # . . . # . . . # . # . . . . . # . # . # . # . # . # . #
# # # # # . # . # # # # # . # . # # # # # . # . # . # . # . # . # . # . # . # . # # # #
# . . . # . . . #                   # . # . . . # . # . . . # . . . # . . . # . # . #
# . # # # # # . # . # # # . # . #   # . # # # . # . # # # # # # # . # . # . # . # . #
# . . . . . . . # . # . # . # .                   # . # . . . . . . . # . # . # . #
# . # . # . # # # . # . # # # . # # # # # . # # #   # # # # # . # # # . # . # . # . #
# . # . # . # . . . # . . . # . # . # . # . # . #   # . . . . . # . . . # . # . # . #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
Time: 76.583000 ms
```

6. Algoritma Backtracking

```
Enter the maze file name: maze.txt
Number of paths found: 502020
Shortest path found:
S**..#..#...
##*#.#.#...#
..**#.#.#...
.#.****#.#.#
.#.#..****##
...#...#.*#.
.#...#.#.*.#
.#.#.....*..
.#.#.#.#.*..
....#.#..E.#
..........##
longest path found:
S**..#..#...
##*#.#.#...#
***.#.#.#...
*######.#.#.#
*#*#######.##
***#***#**#.
.#***#*#***#
.#*#..***.**
.#*#.#.#****
***.#.#**E.#
**********##
Time: 220.092000 ms

Process returned 0 (0x0)   execution time : 2.700 s
Press any key to continue.
```

7. Algoritma Greedy

Maze_1.txt

```
Masukkan nama file: maze_1.txt
Time: 0.000000 ms
All possible paths from start to end:
Path 1: (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 3) -> (3, 3) -> (3, 4) -> (3, 5) -> (3, 6) -> (4, 6) -> (4, 7
) -> (4, 8) -> (4, 9) -> (5, 9) -> (6, 9) -> (6, 10) -> (7, 10) -> (7, 11) -> (8, 11) -> (8, 10) -> (9, 10) -> (9, 9)
Path 2: (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 3) -> (3, 3) -> (3, 4) -> (3, 5) -> (3, 6) -> (4, 6) -> (4, 7
) -> (4, 8) -> (4, 9) -> (5, 9) -> (6, 9) -> (6, 10) -> (7, 10) -> (7, 11) -> (8, 11) -> (8, 10) -> (8, 9) -> (9, 9)
Path 3: (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 3) -> (3, 3) -> (3, 4) -> (3, 5) -> (3, 6) -> (4, 6) -> (4, 7
) -> (4, 8) -> (4, 9) -> (5, 9) -> (6, 9) -> (6, 10) -> (7, 10) -> (7, 11) -> (8, 11) -> (8, 10) -> (8, 9) -> (8, 8) -> (9,
 8) -> (9, 9)
Path 4: (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 3) -> (3, 3) -> (3, 4) -> (3, 5) -> (3, 6) -> (4, 6) -> (4, 7
) -> (4, 8) -> (4, 9) -> (5, 9) -> (6, 9) -> (6, 10) -> (7, 10) -> (7, 11) -> (8, 11) -> (8, 10) -> (8, 9) -> (8, 8) -> (9,
 8) -> (10, 8) -> (10, 9) -> (9, 9)
Path 5: (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 3) -> (3, 3) -> (3, 4) -> (3, 5) -> (3, 6) -> (4, 6) -> (4, 7
) -> (4, 8) -> (4, 9) -> (5, 9) -> (6, 9) -> (6, 10) -> (7, 10) -> (7, 11) -> (8, 11) -> (8, 10) -> (8, 9) -> (8, 8) -> (9,
 8) -> (9, 7) -> (10, 7) -> (10, 8) -> (10, 9) -> (9, 9)
Path 6: (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 3) -> (3, 3) -> (3, 4) -> (3, 5) -> (3, 6) -> (4, 6) -> (4, 7
) -> (4, 8) -> (4, 9) -> (5, 9) -> (6, 9) -> (6, 10) -> (7, 10) -> (7, 11) -> (8, 11) -> (8, 10) -> (8, 9) -> (8, 8) -> (7,
```

```
Shortest path from start to end:
(0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 3) -> (3, 3) -> (3, 4) -> (3, 5) -> (3, 6) -> (4, 6) -> (4, 7) -> (4,
 8) -> (4, 9) -> (5, 9) -> (6, 9) -> (6, 10) -> (7, 10) -> (7, 11) -> (8, 11) -> (8, 10) -> (9, 10) -> (9, 9)

Longest path from start to end:
(0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 3) -> (3, 3) -> (3, 4) -> (3, 5) -> (3, 6) -> (4, 6) -> (4, 7) -> (4,
 8) -> (4, 9) -> (5, 9) -> (6, 9) -> (6, 10) -> (7, 10) -> (7, 11) -> (8, 11) -> (8, 10) -> (8, 9) -> (8, 8) -> (7, 8) -> (
7, 7) -> (7, 6) -> (6, 6) -> (5, 6) -> (5, 5) -> (5, 4) -> (6, 4) -> (6, 3) -> (6, 2) -> (5, 2) -> (5, 1) -> (5, 0) -> (6,
0) -> (7, 0) -> (8, 0) -> (9, 0) -> (9, 1) -> (10, 1) -> (10, 2) -> (9, 2) -> (9, 3) -> (10, 3) -> (10, 4) -> (10, 5) -> (1
0, 6) -> (10, 7) -> (9, 7) -> (9, 8) -> (10, 8) -> (10, 9) -> (9, 9)
```

Maze_3.txt

```
Masukkan nama file: maze_3.txt
Time: 0.000000 ms
All possible paths from start to end:
Path 1: (0, 1) -> (1, 1) -> (2, 1) -> (3, 1) -> (4, 1) -> (5, 1) -> (6, 1) -> (7, 1) -> (7, 2) -> (7, 3) -> (8, 3) -> (9,
3) -> (9, 2) -> (9, 1) -> (10, 1) -> (11, 1) -> (12, 1) -> (13, 1) -> (14, 1) -> (15, 1) -> (16, 1) -> (17, 1) -> (17, 2)
-> (17, 3) -> (16, 3) -> (15, 3) -> (15, 4) -> (15, 5) -> (15, 6) -> (15, 7) -> (14, 7) -> (13, 7) -> (12, 7) -> (11, 7) -
> (11, 8) -> (11, 9) -> (12, 9) -> (13, 9) -> (14, 9) -> (15, 9) -> (15, 10) -> (15, 11) -> (16, 11) -> (17, 11) -> (18, 1
1) -> (19, 11) -> (19, 12) -> (19, 13) -> (19, 14) -> (19, 15) -> (19, 16) -> (19, 17) -> (19, 18) -> (19, 19) -> (20, 19)
Path 2: (0, 1) -> (0, 0) -> (1, 0) -> (1, 1) -> (2, 1) -> (3, 1) -> (4, 1) -> (5, 1) -> (6, 1) -> (7, 1) -> (7, 2) -> (7,
3) -> (8, 3) -> (9, 3) -> (9, 2) -> (9, 1) -> (10, 1) -> (11, 1) -> (12, 1) -> (13, 1) -> (14, 1) -> (15, 1) -> (16, 1) ->
 (17, 1) -> (17, 2) -> (17, 3) -> (16, 3) -> (15, 3) -> (15, 4) -> (15, 5) -> (15, 6) -> (15, 7) -> (14, 7) -> (13, 7) ->
(12, 7) -> (11, 7) -> (11, 8) -> (11, 9) -> (12, 9) -> (13, 9) -> (14, 9) -> (15, 9) -> (15, 10) -> (15, 11) -> (16, 11) -
> (17, 11) -> (18, 11) -> (19, 11) -> (19, 12) -> (19, 13) -> (19, 14) -> (19, 15) -> (19, 16) -> (19, 17) -> (19, 18) ->
(19, 19) -> (20, 19)

Shortest path from start to end:
(0, 1) -> (1, 1) -> (2, 1) -> (3, 1) -> (4, 1) -> (5, 1) -> (6, 1) -> (7, 1) -> (7, 2) -> (7, 3) -> (8, 3) -> (9, 3) -> (9
, 2) -> (9, 1) -> (10, 1) -> (11, 1) -> (12, 1) -> (13, 1) -> (14, 1) -> (15, 1) -> (16, 1) -> (17, 1) -> (17, 2) -> (17,
3) -> (16, 3) -> (15, 3) -> (15, 4) -> (15, 5) -> (15, 6) -> (15, 7) -> (14, 7) -> (13, 7) -> (12, 7) -> (11, 7) -> (11, 8
) -> (11, 9) -> (12, 9) -> (13, 9) -> (14, 9) -> (15, 9) -> (15, 10) -> (15, 11) -> (16, 11) -> (17, 11) -> (18, 11) -> (1
9, 11) -> (19, 12) -> (19, 13) -> (19, 14) -> (19, 15) -> (19, 16) -> (19, 17) -> (19, 18) -> (19, 19) -> (20, 19)

Longest path from start to end:
(0, 1) -> (0, 0) -> (1, 0) -> (1, 1) -> (2, 1) -> (3, 1) -> (4, 1) -> (5, 1) -> (6, 1) -> (7, 1) -> (7, 2) -> (7, 3) -> (8
, 3) -> (9, 3) -> (9, 2) -> (9, 1) -> (10, 1) -> (11, 1) -> (12, 1) -> (13, 1) -> (14, 1) -> (15, 1) -> (16, 1) -> (17, 1)
 -> (17, 2) -> (17, 3) -> (16, 3) -> (15, 3) -> (15, 4) -> (15, 5) -> (15, 6) -> (15, 7) -> (14, 7) -> (13, 7) -> (12, 7)
-> (11, 7) -> (11, 8) -> (11, 9) -> (12, 9) -> (13, 9) -> (14, 9) -> (15, 9) -> (15, 10) -> (15, 11) -> (16, 11) -> (17, 1
1) -> (18, 11) -> (19, 11) -> (19, 12) -> (19, 13) -> (19, 14) -> (19, 15) -> (19, 16) -> (19, 17) -> (19, 18) -> (19, 19)
 -> (20, 19)
```