Solusi Jika Tersesat di Dalam Sebuah Labirin dengan *Breadth-First Search* dan *Dynamic Programming*

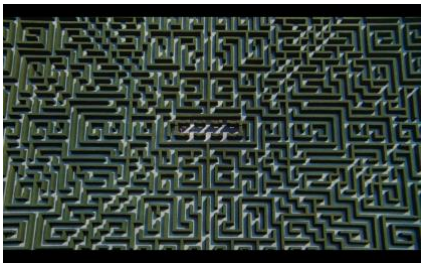
Arik Rayi Arkananta - 13520048
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13520048@std.stei.itb.ac.id

Abstract*—Setelah penulis menaiki transportasi umum angkot di makalah mata kuliah Matematika Diskrit IF2120 milik penulis pada semester lalu, penulis tersesat di dalam sebuah labirin dan memutuskan untuk turun lalu mencari jalan keluarnya sendiri dengan memanfaatkan ilmu-ilmu yang telah dipelajari mata kuliah Strategi Algoritma IF221. Penulis memutuskan untuk memakai algoritma *Breadth-First Search* atau BFS dengan gabungan *dynamic programming* karena permasalahan yang dihadapi mirip dengan *Maze Problem.

Keywords—labirin; Maze Problem; Breadth-First Search; dynamic programming;

I. PENDAHULUAN

Labirin merupakan permainan yang pasti hampir semua orang tahu. Permainan ini pasti pernah anda temui di halaman belakang koran, di buku sekolah, ataupun di film horror. Salah satu film horror terkenal, yaitu The Shining (1980), yang dibintangi oleh aktor legendaris Jack Nicholson, terdapat labirin di filmnya yang sangatlah besar dan menyeramkan sehingga membuat penontonnya merasa takut akan tersesat.



Gambar 1. Labirin pada film The Shining
(sumber: https://www.imdb.com/title/tt0081505/?ref_=nv_sr_srsg_0)

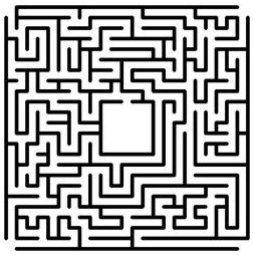
Labirin merupakan permainan yang pasti hampir semua orang tahu. Permainan ini pasti pernah anda temui di halaman belakang koran, di buku sekolah, ataupun di film horror. Salah satu film horror terkenal, yaitu The Shining (1980) terdapat labirin di filmnya yang sangatlah besar dan menyeramkan sehingga membuat penontonnya merasa takut akan tersesat.

II. LANDASAN TEORI

A. *Labirin*

Labyrinth dan *maze* sebenarnya merupakan dua hal yang berbeda, namun jika ditranslasi ke dalam bahasa Indonesia *maze* berarti labirin. Sehingga, penulis akan menyebut labirin yang merupakan *maze*.

Labirin atau *maze* merupakan suatu puzzle dengan suatu titik awal mulai dan juga memiliki banyak kombinasi jalur untuk menemukan jalan keluarnya. Kombinasi jalur yang salah akan mengakibatkan jalan buntu. Sedangkan sebuah *labyrinth* hanya memiliki satu jalan masuk dan jalan keluar.



Gambar 2. Contoh maze dengan banyak jalan keluar
(sumber: https://www.amazeaweek.net/category/mazes/)

Labirin yang diketahui pertama kali sudah ada dari zaman abad ke-5 sebelum masehi di Mesir. Ahli sejarah kuno Hedorotus mengklaim pernah mengunjungi suatu labirin di Mesir dan menggambarkannya sebagai bangunan berliku, terdiri dari ribuan ruangan, banyak di antaranya berada di bawah tanah dan menyimpan makam raja-raja Mesir.

Sehingga, labirin tampaknya hanya satu jalur kompleks pada awalnya. Namun, seiring berjalannya waktu, konsep labirin telah digunakan dan dimodifikasi untuk berbagai tujuan, hingga muncul konsep labirin dengan banyak rute, di mana kita harus menemukan jalur yang benar untuk mencapai titik akhir atau tujuan.

B. *Breadth-First Search*

Breadth-First Search atau disingkat BFS merupakan suatu algoritma yang digunakan untuk *tree traversal* untuk struktur data graf atau pohon. Ingat bahwa grafik adalah struktur matematika yang terdiri dari himpunan simpul dan tepi. Tepi menghubungkan tepat dua simpul dan, tergantung pada jenis grafik, memiliki berbagai batasan tambahan. Graf berarah, misalnya, memiliki aturan bahwa sisi yang menghubungkan A dan B tidak menyiratkan bahwa A dan B terhubung, dan graf sederhana memiliki aturan bahwa tidak ada dua sisi yang memiliki titik awal dan titik akhir yang sama. BFS hanya akan

digunakan pada grafik dasar dan berarah untuk menjaga semuanya tetap sederhana.

Sebut simpul tetangga dari simpul A ialah semua simpul B sehingga terdapat suatu sisi dengan simpul awal A dan simpul akhir B. Sesuai namanya, algoritma penelusuran BFS menelusuri simpul-simpul dalam graf dengan urutan lapisan demi lapisan ketetanggaan dari simpul sumber. Dalam artian pertama algoritma akan mengecek simpul suber, kemudian semua simpul yang bertetangga dengan simpul sumber, kemudian semua simpul yang belum ditelusuri dan bertetangga dengan simpul tetangga sumber, dan seterusnya. Ini berbeda dengan DFS yang sesuai namanya menelusuri graf dengan mendalami suatu jalur penelusuran simpul terus hingga buntu, baru kembali hingga mempunyai tetangga yang belum ditelusuri, kemudian dalami jalur tersebut, dan seterusnya.

Algoritma BFS menggunakan struktur data Antrian (Queue) dalam menjalankan tugasnya, berbeda ketimbang DFS yang menggunakan Stack. Queue digunakan karena perilaku algoritma BFS yang menyimpan simpul yang akan ditelusuri dalam urutan FIFO (First In First Out). Penerapan Algoritma BFS yang menggunakan banyak simpul sumber masih dapat hanya menggunakan satu Queue saja yakni memasukkan semua simpul dalam Queue kemudian dilanjutkan biasa.

C. *Dynamic Programming*

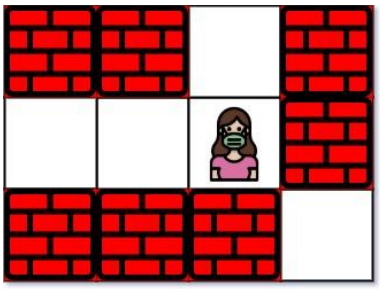
Dynamic Programming merupakan suatu algoritma untuk mengoptimasi solusi dengan cara memecah permasalahan menjadi sub-masalah yang bisa jadi identik secara rekursif lalu menyimpan tiap sub-masalah di dalam suatu struktur data seperti *list* atau *array*. Tergantung urutan penyelesaian, *dynamic programming* dapat dibagi menjadi dua cara, yaitu *top-down* dan *bottom-up*.

Metode *top-down* merupakan memecahkan masalah dari atas, kemudian menyelesaikan sub-masalah jika diperlukan. Sebaliknya, *bottom-up* menyelesaikan sub-masalah yang paling sederhana terlebih dahulu, kemudian membangun solusi untuk masalah yang lebih rumit.

Baris Fibonacchi, misalnya, berisi fungsi rekursif Fn+1 = Fn+Fn-1, tetapi jika perangkat lunak menggunakan cara rekursif untuk menghitung F100 tanpa menyimpan nilai, F98 akan dihitung ulang dua kali, F97 akan dihitung ulang tiga kali, dan segera. Kompleksitas waktu semakin meningkat. Ini dapat diatasi dengan mengalokasikan memori untuk menyimpan 99 fibonachi pertama yang diperlukan untuk menghitung F100. Aplikasi 2 dimensi lain dari algoritma DP adalah perhitungan nilai kombinasi dengan segitiga Pascal, khususnya identitas C(n,m) = C(n-1,m) + C(n-1,m-1), dan masih banyak lagi.

III. Analisis dan Pembahasan

Di permasalahan ini, penulis memberi titik awal di tengah-tengah labirin, karena ceritanya merupakan tersesat di tengah-tengah labirin, lalu akan mencari pintu kelar terdekat dari labirin tersebut dan mengembalikan jumlah pergerakan terkecil tersebut. Terdapat tile kosong yang bisa dilewati oleh pemain, tile tembok yang tidak bisa dilewati oleh pemain, dan juga tile mulai pemain. Labirin akan dibentuk sebagai sebuah matrix m x n sehingga tiap elemen berbentuk koordinat. Tiap tile kosong yang ada di pinggir merupakan jalan keluar. Namun, jika titik mulai pemain ada di pinggir yang merupakan jalan keluar, tidak akan di anggap jalan keluar sehingga tidak terlalu mudah.



Gambar 3. Gambaran jenis labirin yang digunakan (sumber: https://leetcode.com/problems/nearest-exit-from-entrance-in-maze/)

Pada contoh di atas, titik mulai pemain merupakan letak dimana pemain ada. Kotak batu bata merupakan tile tembok. Kotak putih merupakan tile kosong yang dapat dilewati peemain. Dari contoh di atas, jarak terpendek untuk jalan keluar dari labirin merupakan 1 pergerakan yaitu dengan cara bergerak ke atas.

A. *Penerapan Algoritma BFS*

Algoritma BFS akan diterapkan untuk mencari jarak terpendek ke tiap simpul dengan ketetanggaan yang dimulai dari titik awal mulai. Simpul disini merupakan seluruh tile kosong yang ada pada matrix labirin. Berikut gambaran kasar algoritma BFS-nya:

1) Kunjungi semua simpul yang bertetangga dengan simpul awal
2) Nilai untuk simpul yang dikunjungi merupakan nilai simpul awal ditambah 1 (Nilai simpul titik awal pemain merupakan 0)
3) Bandingkan nilai simpul yang dikunjungi yang baru dengan nilai simpul yang dikunjungi pada matrix hasil (Nilai awal simpul yang belum pernah dikunjungi merupakan INT_MAX)
4) Jika nilainya lebih kecil, masukan nilai barunya

TABLE I. TABEL ELEMEN PADA ALGORITMA BFS

Elemen	Representasi
Simpul Akar	Titik awal pemain
Simpul	Seluruh tile kosong
Sisi	Hubungan tile kosong yang terhubung

B. *Penerapan Dynamic Programming*

Penulis menggunakan strategi *dynamic programming top-down* dengan menyimpan nilai-nilai terpendek menuju ke tiap

simpul atau koordinat labirin yang telah dilewati ke dalam sebuah matriks. Penulis memilih jenis *top-down* karena tahapan awalnya lebih sederhana daripada tahap selanjutnya, contohnya pada simpul yang bertetangga dengan titik awal pasti akan menjadi jarak terpendek sedangkan di tahap-tahap berikutnya bisa jadi bukan jarak terpendeknya karena bisa jadi juga dilewati oleh simpul lainnya.

C. Pencarian Jalan Keluar dengan Nilai Terkecil

Pencarian jalan keluar yang memiliki nilai terkecil dilakukan dengan cara mengiterasikan isi dari matriks hasil setelah algoritma BFS selesai. Iterasi dilakukan dengan cara *brute-force* biasa dan mengecek nilai hanya jika koordinat merupakan jalan keluar (dipojok) dan tile kosong.

D. Implementasi Algoritma

Penulis mengimplementasikan algoritma ini pada bahasa pemrograman C++ dengan menjadikan suatu fungsi yang menerima vector dua dimensi yang berupa matriks labirin itu sendiri dan vector ent yang merupakan titik awal pemain dengan elemen ke-0 dan elemen ke-1 adalah x dan y secara berurutan.

Simbol untuk tembok pada labirin di dalam vektor masukan akan berupa ‘+’ dan tile kosong akan berupa ‘.’.

```
int nearestExit(vector<vector<char>>& maze, vector<int>& ent) {
    // Inisiasi vector untuk semua pergerakan mungkin
    vector< pair<int, int> > dirs = {{0,1}, {0,-1}, {-1,0}, {1,0}};
    int brs = maze.size(), klm = maze[0].size();
    int ans[brs][klm];
    // Inisiasi nilai awal INT_MAX untuk semua titik
    for(int i=0; i<brs; i++)
        for(int j=0; j<klm ; j++)
            ans[i][j] = INT_MAX;

    deque< pair<int, int> > dq;
    dq.push_back({ent[0], ent[1]});
    ans[ent[0]][ent[1]] = 0;

    while(!dq.empty()){
        int r = dq[0].first;
        int c = dq[0].second;
        dq.pop_front();
        if(r < 0 || c < 0 || r >= brs || c >= klm || maze[r][c] == '+') continue;
        for(auto dir : dirs){
            int p = r + dir.first;
            int q = c + dir.second;
            if(p >= 0 && q >= 0 && p < brs && q < klm && maze[p][q] == '.' && ans[p][q] > ans[r][c] + 1){
                ans[p][q] = ans[r][c] + 1;
                dq.push_back({p,q};
            }
        }
    }
    int res = INT_MAX;
    for(int i=0; i<brs; i++){
        if(ans[i][0] != 0) res = min(res, ans[i][0]);
        if(ans[i][klm-1] != 0) res = min(res, ans[i][klm-1]);
    }
    for(int j=0; j<klm; j++){
        if(ans[0][j] != 0) res = min(res, ans[0][j]);
        if(ans[brs-1][j] != 0) res = min(res, ans[brs-1][j]);
    }
    return (res == INT_MAX) ? -1 : res;
}
```

E. Masukan Program

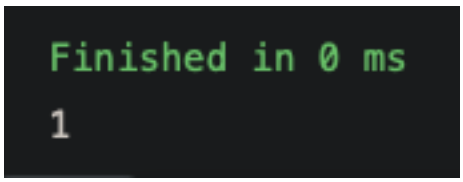
Pada contoh pertama, program menerima masukan vektor dua dimensi yang berupa peta labirin dan juga titik awal berupa maze = {{'+','+','.','+'}, {'.','.','.','+'}, {'+','+','+','.'}} dan ent = [1,2].

Untuk contoh kedua, program menerima masukan vektor dua dimensi yang berupa peta labirin dan juga titik awal berupa maze = {{'+','+','+','+','+'}, {'+','.','.','+','.'}, {'+','.','+','+','.'}, {'.','.','.','.','+'}, {'+','+','+','.','+'}} dan ent = [1,0].

F. Keluaran Program

1. Contoh satu

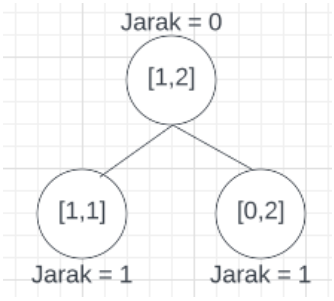
Lama untuk menjalankan contoh satu adalah tidak sampai 0 milidetik. Berikut adalah *screenshot* hasil program.



Gambar 4. Screenshot hasil contoh 1
(sumber: dokumen pribadi)

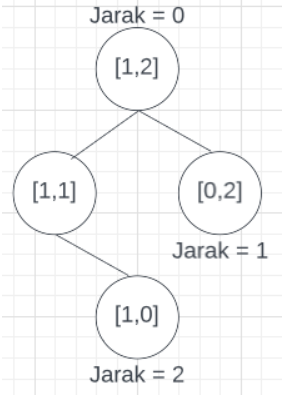
Keluaran dari program adalah jalur terpendek yang dibutuhkan untuk keluar dari labirin. Sesuai dengan dugaan dibutuhkan satu kali gerak yaitu ke atas. Proses algoritma adalah sebagai berikut:

1) Dicari semua simpul yang bertetangga dengan simpul awal, yaitu titik mulai pemain (1,2). Dapat dilihat simpul (0,2) merupakan jalan keluar karena berada di pojok.



Gambar 5. BFS level pertama
(sumber: dokumen pribadi)

2) Sisa dari simpul yang masih bertetangga hanyalah simpul (1,0) yang bertetangga dengan simpul (1,0)



Gambar 6. BFS level kedua
(sumber: dokumen pribadi)

3) Tabel jarak untuk tiap simpul dan pemilihan jarak terpendek

TABLE II. TABEL JARAK TERPENDEK TIAP SIMPUL

Row\Col	0	1	2	3
0	INT_MAX	INT_MAX	1	INT_MAX
1	2	1	0	INT_MAX
2	INT_MAX	INT_MAX	INT_MAX	INT_MAX

Jika dilihat di atas, semua koordinat/simpul yang berada di pojok, simpul (0,2) memiliki nilai terkecil, yaitu 1.

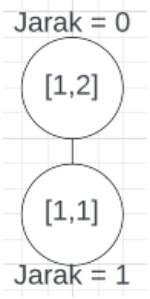
2. Contoh dua

Lama untuk menjalankan contoh satu adalah 4 milidetik. Berikut adalah *screenshot* hasil program.

Gambar 7. Screenshot hasil contoh 1
(sumber: dokumen pribadi)

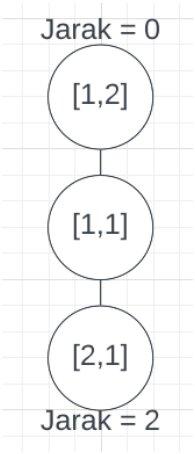
Keluaran dari program adalah jalur terpendek yang dibutuhkan untuk keluar dari labirin. Hasil yang didapat adalah 4. Proses algoritma adalah sebagai berikut:

1) Dicari semua simpul yang bertetangga dengan simpul awal, yaitu titik mulai pemain (1,2).



Gambar 8. BFS level pertama
(sumber: dokumen pribadi)

2) Satu-satunya simpul yang bertetangga dengan simpul (1,1) adalah simpul (2,1)

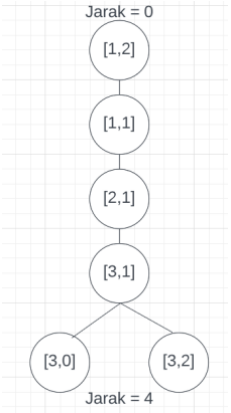


Gambar 9. BFS level kedua
(sumber: dokumen pribadi)

3) Satu-satunya simpul yang bertetangga dengan simpul (2,1) adalah simpul (3,1)

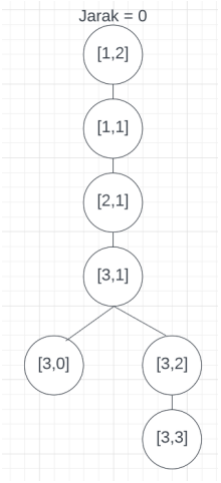
Gambar 10. BFS level ketiga
(sumber: dokumen pribadi)

4) Simpul-simpul yang bertetangga dengan simpul (3,1) salah satunya merupakan jalan keluar, yaitu simpul (3,0)



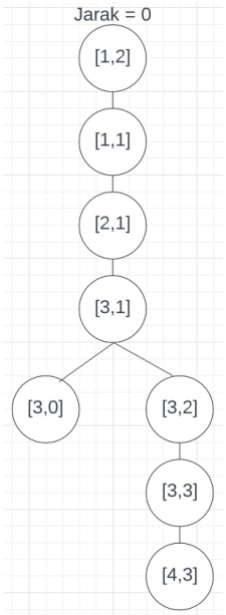
Gambar 11. BFS level keempat
(sumber: dokumen pribadi)

5) Satu-satunya simpul yang bertetangga dengan simpul (3,2) adalah simpul (3,3)



Gambar 12. BFS level kelima
(sumber: dokumen pribadi)

6) Satu-satunya simpul yang bertetangga dengan simpul (3,3) adalah simpul (4,3) yang merupakan salah satu jalan keluar dengan jarak enam gerakan.



Gambar 13. BFS level keenam
(sumber: dokumen pribadi)

TABLE III. TABEL JARAK TERPENDEK TIAP SIMPUL

Row\Col	0	1	2	3	4
0	INT_MAX	INT_MAX	INT_MAX	INT_MAX	INT_MAX
1	INT_MAX	1	0	INT_MAX	INT_MAX
2	INT_MAX	2	INT_MAX	INT_MAX	INT_MAX
3	4	3	4	5	INT_MAX
4	INT_MAX	INT_MAX	INT_MAX	6	INT_MAX

Jika dilihat di atas, semua koordinat/simpul yang berada di pojok dan tidak bernilai INT_MAX, yaitu simpul (3,0) dan (4,3) merupakan jalan keluarnya. Dari kedua nilai tersebut, nilai yang terkecil merupakan 3, yaitu simpul (3,0).

Dapat dilihat dari dua contoh di atas bahwa program dapat menyelesaikan kasus-kasus sederhana dengan cepat, pada contoh kasus pertama yang memiliki ukuran labirin 3x4 tidak sampai 0 milidetik sedangkan untuk kasus kedua yang memiliki ukuran labirin 5x5 hanya memakan waktu 4 milidetik.

IV. KESIMPULAN

Breadth-First Search atau BFS merupakan salah satu algoritma yang baik untuk digunakan untuk *tree traversal* untuk

struktur data graf atau pohon. Graf berarah, misalnya, memiliki aturan bahwa sisi yang menghubungkan A dan B tidak menyiratkan bahwa A dan B terhubung, dan graf sederhana memiliki aturan bahwa tidak ada dua sisi yang memiliki titik awal dan titik akhir yang sama. BFS hanya akan digunakan pada grafik dasar dan berarah untuk menjaga semuanya tetap sederhana.

Dapat dilihat dari bab sebelumnya, penggunaan algoritma *Breadth-First Search* atau BFS dengan bantuan *dynamic programming* untuk memecahkan variasi dari permasalahan *maze problem* merupakan salah satu cara yang optimal. Dilihat dari durasi untu menyelesaikan contoh-contoh kasus yang diberikan, program mampu menyelesaikannya dengan waktu yang sangat cepat, yaitu 4 milidetik.

V. Acknowledgment

Dengan diselesaikannya makalah ini dengan lancar dan juga tepat watu, Penulis memanjatkan Puji dan Sykur kepada Allah SWT yang Maha Kuasa atas Rahmat-Nya. Penulis juga ingin mengucapkan terima kasih kepada dosen mata kuliah Strategi Algoritma IF2211 penulis, yaitu Dr. Ir. Rinaldi Munir , yang telah membimbing pelajaran Strategi Algorima dan juga atas ilmunya selama satu semester ini.

References

[1] https://www.usi.edu/media/4024497/Mazes-vs-Labyrinths2.pdf. Diakses pada 21 Mei 2022

[2] https://www.educative.io/edpresso/how-to-implement-a-breadth-first-search-in-python. Diakses pada 21 Mei 2022

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Makalah2021/Makalah-Stima-2021-K2%20(19).pdf. Diakses pada 21 Mei 2022

[4] https://www.mentalfloss.com/article/70964/15-intricate-facts-about-mazes. Diakses pada 21 Mei 2022

[5] https://leetcode.com/problems/nearest-exit-from-entrance-in-maze/. Diakses pada 21 Mei 2022

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2022

Arik Rayi Arkananta 13520048