Penggunaan BFS dan DFS dalam Penyelesaian Labirin

Nicholas Budiono - 13520121
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13520121@std.stei.itb.ac.id

***Abstrak*—Masa yang ada pada saat ini, membuat kita sebagai manusia memiliki banyak kebutuhan, dengan adanya kebutuhan, akan muncul sebuah inovasi untuk menyelesaikan permasalahn tersebut. Salah satu masalah yang telah diselesaikan adalah navigasi, navigasi pada awalnya diselesaikan dengan peta dan Kompas. Dengan perkembangannya teknologi, muncul cara baru yaitu radar dan GPS. Salah satu masalah yang diselesaikan oleh GPS adalah mencari jalur dari satu titik ke titik lain. Pada makalah ini, akan dibahas sebuah masalah mengenai labirin, dan cara penyelesaiiannya menggunakan BFS dan DFS.**

Kata kunci – Navigasi, GPS, Labirin, BFS, DFS

I. PENDAHULUAN

Pada zaman teknologi modern ini, banyak kegunaan teknologi yang secara langsung membantu kebutuhan sehari-hari, Pada Era Informasi saat ini, begitu banyak data yang keluar dan masuk dari teknologi yang kita gunkan sehari-hari, memberikan informasi yang kita butuhkan seperti informasi tentang lingkungan yang ada di sekitar kita, membantu kita untuk melakukan navigasi ke tujuan tertentu yang kita inginkan. Salah satu contoh dari teknologi tersebut adalah GPS.

Teknologi GPS sendiri ditemukan pada tahun 1970 oleh Ivan Getting dari Departemen Pertahanan Militer Amerika Serikat. Diawali dengan peluncuran satelit GPS yang dilakukan pada 1974, diikuti oleh beberapa satelit lain yang dibutuhkan untuk membuat konstelasi yang mempunyai jangkauan seluruh area di bumi. Teknologi ini baru dibuka oleh umum pada tahun 1994. Dengan memanfaatkan sinyal radio dari satelit Sutnik, ide tersebut dipresentasikan dan disetujui oleh Pemerintahan Amerika Serikat yang bersedia untuk memberikan dana yang besar untuk pengembangan teknologi ini.

Gambar 1.1. Contoh kegunaan dari GPS
(sumber : https://www.geotekno.com/download-gratis-gps-map-indonesia/33)

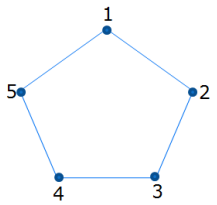
Pada saat ini, sangat banyak kegunaan dari GPS, dari pembuatan peta yang ada di telefon genggam, navigasi di alat lain untuk alat transportasi yang ada, dan banyak yang lain lagi. Salah yang akan dilihat lebih dalam adalah cara GPS untuk mencari jalan dari satu titik ke titk lain, tidak yang paling cepat, tapi mendapatkan jalan keluar.

Alat yang akan digunakan untuk melakukan path finding (menemukan jalan keluar dari jalan masuk ayng telah diberikan) adalah BFS dan DFS, yaitu sebuah algoritma yang menggunakan graf dengan cara pencarian BFS(Breadth First Search) dan DFS(Depth First Search).

II. LANDASAN TEORI

A. *Graf*

Graf adalah sebuah algoritma pencaharian data yang ada dalam bentuk pohon. Pencaharian dilakukan awal dari akar pohon dan dilakukan setiap pencaharian dari satu kedalaman terlebih dahulu, setelah sudah, akan dilanjutkan ke kedalaman berikutnya.

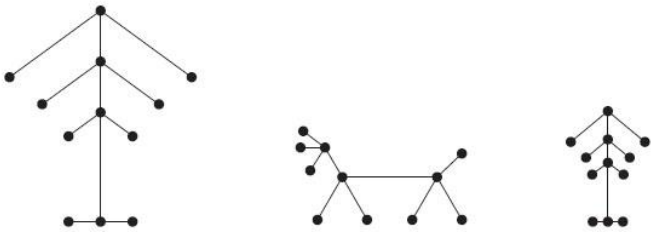


Gambar 2.1. Graf

(sumber : https://bermatematika.net/2016/05/20/graf-dan-komplemennya/)

B. *Pohon*

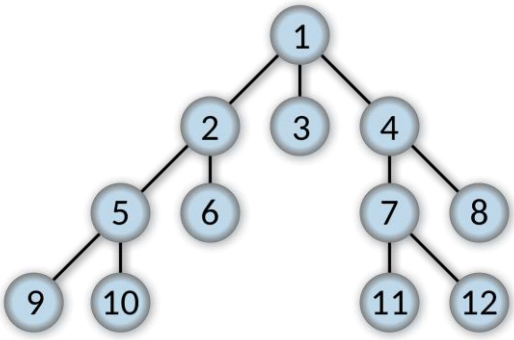
Pohon adalah satu jenis dari graf, dengan syarat tertentu yang berberda dengan graf. Graf pohon memiliki syarat seperti graf tidak boleh memliki sisi ganda dan sirkuit. Graf pohon sangat berguna karena dapat diaplikasikan dengan banyak cara dan kegunaan nya masing-masing.



Gambar 2.2. Grafik Pohon
(sumber : https://www.haimatematika.com/2018/12/graf-pohon-teori-graf.html)

C. *BFS*

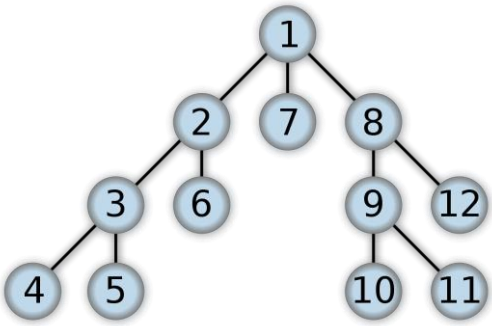
BFS adalah sebuah algoritma pencaharian data yang ada dalam bentuk pohon. Pencaharian dilakukan awal dari akar pohon dan dilakukan setiap pencaharian dari satu kedalaman terlebih dahulu, setelah sudah, akan dilanjutkan ke kedalaman berikutnya.



Gambar 2.3. BFS algotithm
(sumber: https://en.wikipedia.org/wiki/Breadth-first_search)

D. *DFS*

DFS memiliki sifat awal yang sama seperti BFS, merupakan sebual algoritma pencaharian data yang berbentuk pohon. Perbedaannya adalah cara algoritma tersebut bekerja. DFS melakukannya dengan mencari node yang ada sampai ke daun, setelah sampai ke daun, algoritma akan back-tracking ke node sebelumnya dan melakukan pencaharian lanjut ke daun sampai semua daun telah di-cek.

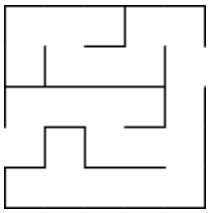


Gambar 2.4. DFS Algorithm
(sumber: https://en.wikipedia.org/wiki/Depth-first_search)

III. Implementasi BFS dan DFS Dalam Penyelesaiian Labirin

A. *Perkenalan Permasalahan*

Dalam pengaplikasiannya, navigasi memang memiliki banyak kegunaan, tetapi, demi kemudahan dalam penjelasan, akan digunakan permasalah labirin. Labirin sendiri adalah sebuah jenis puzzle yang memiliki spesifikasi seperti jalan masuk dan jalan keluar, terdapat jalan berliku-liku. Tujuan dari puzzle tersebut adalah mencari jalan keluar dari titik masuk ke titik keluar.



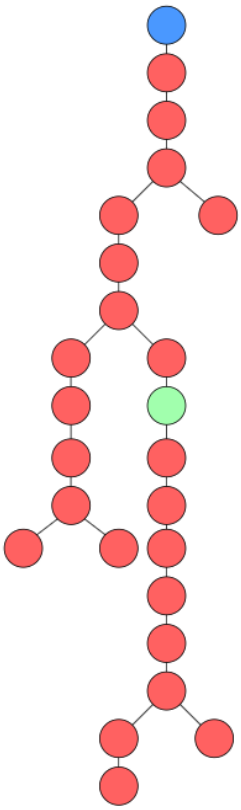
Gambar 3.1. Labirin

(sumber : lokal)

Pada makalah ini, akan dicontohkan dengan menggunakan labirin berukuran 5x5 untuk kemudahan penjelasan.

B. *Penyelesaian Permasalahan*

Dari labirin yang menjadi masalah sebelumnya, kita dapat menvisualisasikan permasalahannya menjadi sebuah pohon sebagai berikut.



Gambar 3.2. Graf Pohon
(sumber : lokal)

Dapat dilihat bahwa warna biru dilambangkan sebagai jalur masuk, dan hijau dilambangkan sebagai jalan keluar. Graf pohon di atas menandakan bahwa dengan menggunakan cara yang tepat, kita bisa mencari secara pasti jalan keluar dari labirin tersebut.

Dari Penjabaran masalah yang telah dilakukan sebelumnya, akan dilakukan penyelesaian dengan menggunakan dua cara, yaitu BFS dan DFS.

C. *BFS*

Diketahui dalam cara BFS(Breadth First Search), algoritma akan melakukan pengecekkan dari kedalaman pertama secara keseluruhan, setelah itu, akan dilanjutkan ke kedalaman selanjutnya sampai ke daun dari graf pohon tersebut. Dalam makalah ini, akan dibuat kode sebagai algoritma BFS dan DFS untuk melakukan pengetesan. Dapat dilihat dibawah, kode python yang digunakan untuk BFS.

```python
# BFS
def bfs(Grid, dest: Grid_Position, start: Grid_Position):
    adj_cell_x = [-1, 0, 0, 1]
    adj_cell_y = [0, -1, 1, 0]
    m, n = (len(Grid), len(Grid))
    visited_blocks = [[False for i in range(m)]
                      for j in range(n)]
    visited_blocks[start.x][start.y] = True
    queue = deque()
    sol = Node(start, 0)
    queue.append(sol)
    cells = 4
    cost = 0
    while queue:
        current_block = queue.popleft()
        current_pos = current_block.pos
        if current_pos.x == dest.x and current_pos.y == dest.y:
            print("Algorithm used = BFS")
            print("Path found!!")
            print("Total nodes visited = ", cost)
            return current_block.cost

        if current_block not in visited_blocks:
            visited_blocks[current_pos.x][current_pos.y] = True
            cost = cost + 1
        x_pos = current_pos.x
        y_pos = current_pos.y
        for i in range(cells):
            if x_pos == len(Grid) - 1 and adj_cell_x[i] == 1:
                x_pos = current_pos.x
                y_pos = current_pos.y + adj_cell_y[i]
            if y_pos == 0 and adj_cell_y[i] == -1:
                x_pos = current_pos.x + adj_cell_x[i]
                y_pos = current_pos.y
            else:
                x_pos = current_pos.x + adj_cell_x[i]
                y_pos = current_pos.y + adj_cell_y[i]
            if x_pos < 12 and y_pos < 12 and x_pos >= 0 and y_pos >= 0:
                if Grid[x_pos][y_pos] == 1:
                    if not visited_blocks[x_pos][y_pos]:
                        next_cell = Node(Grid_Position(x_pos, y_pos),
                                         current_block.cost + 1)
                        visited_blocks[x_pos][y_pos] = True
                        queue.append(next_cell)
    return -1
```

Gambar 3.3. kode BFS

(sumber : lokal)

Dari kode diatas, akan dihasilkan dari input sebagai matriks binary yang melambangkan labirin dalam bentuk pohon, hal ini membuat kode dapat bekerja sesuai dengan ketentuan yang ada pada BFS. Kode tersebut membuat hasil sebagai berikut.

```
Algorithm used = BFS
Path found!!
Total nodes visited =  25
Shortest path steps =  18
```

Gambar 3.4. Hasil kode BFS

(sumber : lokal)

Didapat bahwa dari labirin yang telah kita gunakan sebagai contoh, total node yang di jangkau adalah 25, dengan langkah yang digunakan untuk mencapai tujuan adlaah 18.

D. DFS

Dalam cara DFS ini(Depth First Search), algoritma ini melakukan pengecekkan dari satu cabang langsung lanjut ke cabang di dalmnya sehingga sampai ke daun, setelah sampai ke satu daun, algoritma akan melakukan back-tracking dan mengecek daun selanjutnya sampai semua graf pohon telah dicek.

```
def dfs(Grid, dest: Grid_Position, start: Grid_Position):
    adj_cell_x = [1, 0, 0, -1]
    adj_cell_y = [0, 1, -1, 0]
    m, n = (len(Grid), len(Grid))
    visited_blocks = [[False for i in range(m)]
                      for j in range(n)]
    visited_blocks[start.x][start.y] = True
    stack = deque()
    sol = Node(start, 0)
    stack.append(sol)
    neigh = 4
    neighbours = []
    cost = 0
    while stack:
        current_block = stack.pop()
        current_pos = current_block.pos
        if current_pos.x == dest.x and current_pos.y == dest.y:
            print("Algorithm used = DFS")
            print("Path found!!")
            print("Total nodes visited = ", cost)
            return current_block.cost
        x_pos = current_pos.x
        y_pos = current_pos.y

        for i in range(neigh):
            if x_pos == len(Grid) - 1 and adj_cell_x[i] == 1:
                x_pos = current_pos.x
                y_pos = current_pos.y + adj_cell_y[i]
            if y_pos == 0 and adj_cell_y[i] == -1:
                x_pos = current_pos.x + adj_cell_x[i]
                y_pos = current_pos.y
            else:
                x_pos = current_pos.x + adj_cell_x[i]
                y_pos = current_pos.y + adj_cell_y[i]
            if x_pos != 12 and x_pos != -1 and y_pos != 12 and y_pos != -1:
                if Grid[x_pos][y_pos] == 1:
                    if not visited_blocks[x_pos][y_pos]:
                        cost += 1
                        visited_blocks[x_pos][y_pos] = True
                        stack.append(create_node(
                            x_pos, y_pos, current_block.cost))
    return -1
```

Gambar 3.5. kode DFS
(sumber : lokal)

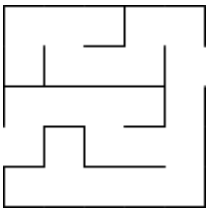
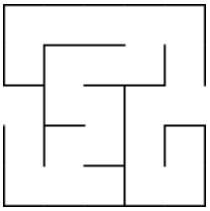
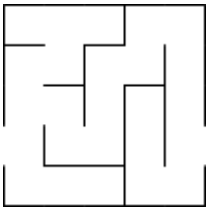
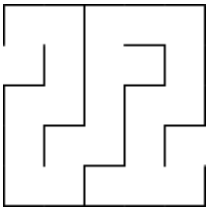
Dari kode algoritma DFS diatas, kode memiliki input yang sama dengan potongan kode yang ada pada algoritma BFS, kode ini akan menghasilkan sebagai berikut.

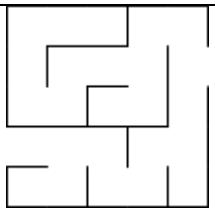
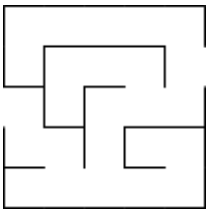
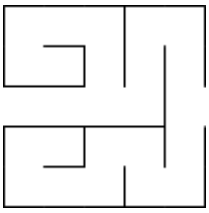
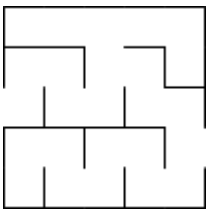
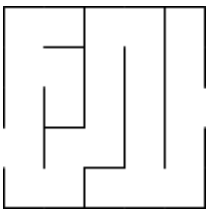
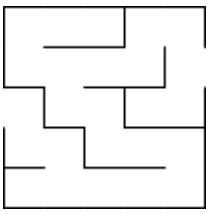
```
Algorithm used = DFS
Path found!!
Total nodes visited =  39
Steps with backtracking =  18
```

Gambar 3.6. Hasil kode DFS
(sumber : lokal)

E. Hasil Pengetesan

Setelah mengetahui perbedaan dari BFS dan DFS serta hasil yang didapat dari kedua cara, akan dilakukan pengetesan dari 10 labirin yang dibuat secara random sehingga kita dapat menentukan dan membuktikan apa kelebihan dan kekurangan dari satu cara dengan yang lain.

No	Labirin	BFS(node-visited)	DFS(node-visited)
1.		25	39
2.		40	49
3.		43	39
4.		50	49
5.		25	46

			
6.		36	43
7.		30	31
8.		32	33
9.		47	50
10.		46	47
	Rata-rata	37.4	42.6

IV. Kesimpulan

Dari segi kegunaannya, BFS dan DFS telah terbukti memiliki banyak kegunaan dalam kehidupan sehari-hari, dalam masa saat ini yang kita sebagai manusia memiliki kebergantungan akan teknologi. Sehingga, teknologi yang kit agunakan harus akurat dan mudah digunakan. Salah satu dari teknologi tersebut adalah GPS, GPS bekerja dengan menunjukan jalan atau arah yang si pengguna perlu lakukan untuk mencapai tujuan tertentu. Hal ini sangat berguna karena kebutuhan kita dengan mudahnya terpenuhnya untuk melakukan eksplorasi atau mencari satu tujuan.

Dalam eksperimen ini, didapatkan hasil yang demikian. Dari 10 labirin yang dibuat secara random, didapatkan data dengan hasil node yang dilewati oleh algoritma sebagai berikut. Didapat, dengan menggunakan BFS, rata-rata dari node yang dilalui adalah 37.4, sedangkan dengan menggunakan DFS, akan didapat 42.6.

Dri hasil yang didapatkan, kita dapat mensimpulkan bahwa BFS memiliki kelebihan disbandingkan dengan DFS dalam aplikasi labirin, tetapi kenapa? Menurut saya, saya dapat menyimpulkan bahwa dalam permasalhan labirin, sebaiknya menggunakan BFS karena DFS bekerja baik jika suatu labirin, atua pohon, memiliki lebih sedikit cabang, hal ini dikarenakan cara kerjanya DFS yang membuat algoritma pencarian dengan melalui node sampai ke daun dan hanya menemukan jalan buntu dan harus bak-tracking. Sedangkan BFS tidak mefokuskan ke satu jalan, tapi lebih luas, sehingga jika dilihat dari rata-rata. BFS lebih baik dibandingkan DFS, sedangkan DFS tidak akan memiliki hasil yang konstan, algoritma tersebut akan bekerja sangat baik atau sangat buruk untuk labirin.

V. Link Video Youtube

https://youtu.be/EUgUKzhutnE

VI. Ucapan Terima Kasih

Pertama, saya sebagai penulis panjatkan puji syukur kepada Tuhan Yang Maha Esa karen berkat-Nya yang telah diberikan dalam pembuatan makalah ini. Ucapan terima kasih juga kepada orang tua dan teman yang membantu dalam bentuk dukungan untuk penyelesaian makalah ini. Tidak lupa juga ucapan terima kasih kepada Dr. Masayu Leylia Khodra, S.T., M.T. sebagai dosen dalam mata kuliah IF2211 Strategi Algoritma. Terima kahis atas ilmu yang telah diberikan dan motivasi untuk mendapatkan ilmu baru dalam bentuk apapun. Mohon maaf bila ada salah dalam bentuk apapum, baik yang disengaja maupun tidak disengaja dalam pembuatan makalah ini.

VII. References

[1] Pengertian Graf https://bermatematika.net/2016/05/20/graf-dan-komplemennya/

[2] Graf Pohon https://www.haimatematika.com/2018/12/graf-pohon-teori-graf.html

[3] Breadh-First Search https://en.wikipedia.org/wiki/Breadth-first_search

[4] Depth-First Search https://en.wikipedia.org/wiki/Depth-first_search

[5] Pembuatan Diagram Pohon https://app.diagrams.net/#G1rrVkNV2XAqOuOwEghvDpVvwQmxbN3Zka

[6] Pembuatan Labirin https://puzzlemaker.discoveryeducation.com/maze

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

Nicholas Budiono 13520121