Penggunaan Pengolahan Gambar dan Algoritma Pencari Jalur dalam Pencarian Solusi dari Labirin

Dengan Perbandingan antara Algoritma Depth First Search, Algoritma Breadth First Search, dan Algoritma A-Star

David Timothy Panjaitan 13516075
Program Studi Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
davidtpanjaitan@gmail.com

***Abstract*—Salah satu metode yang umum digunakan untuk mencari solusi dari persoalan labirin adalah pemodelan labirin menjadi graf, kemudian jalur solusi dicari menggunakan algoritma A*. Akan tetapi, labirin dalam bentuk gambar mungkin tidak dapat langsung dimodelkan menjadi graf. Karena itu, dibutuhkan proses pengolahan gambar (*Image Processing*) untuk membentuk graf agar labirin dapat diselesaikan menggunakan algoritma A*. Sebagai perbandingan, selain A* akan digunakan algoritma BFS (*Breadth First Search*) dan algoritma DFS (*Depth First Search*) yang juga menggunakan pemodelan labirin menjadi graf.**

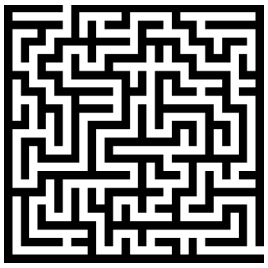
Keywords—Pengolahan gambar, Labirin, Graf, Algoritma A*, Algoritma Breadth First Search, Algoritma Depth First Search.

I. PENDAHULUAN

Pencarian jalur (*pathfinding*) merupakan persoalan mencari lintasan terpendek untuk menghubungkan titik awal dengan titik tujuan. Aplikasi dari persoalan pencarian jalan ini sangat luas dan ada di berbagai bidang, contohnya untuk pencarian rute pada GPS, penentuan jalur pada game, dan pencarian solusi labirin. Untuk menyelesaikan persoalan ini menggunakan program komputer secara mangkus dan sangkil, telah dibuat banyak algoritma pencarian jalan, beberapa di antaranya adalah algoritma BFS (*Breadth First Search*), algoritma DFS (*Depth First Search*), dan algoritma A* (*A-Star*). Ketiga algoritma tersebut menggunakan pemodelan persoalan dengan graf dan menyelesaikan persoalan dengan membangun pohon pencarian (*search tree*).

Pencarian solusi untuk labirin merupakan contoh persoalan yang termasuk kategori persoalan pencarian jalur. Labirin sendiri adalah suatu struktur yang rumit dan bertujuan untuk membingungkan orang yang memasukinya. Terdapat berbagai jenis labirin, contohnya labirin yang mencari jalan keluar dari suatu tempat di tengah labirin, labirin yang memiliki pintu masuk dan pintu keluar, labirin yang memiliki beberapa jalan keluar, labirin yang memiliki perputaran (*looping*) atau labirin yang tidak memiliki perputaran. Labirin yang akan dibahas dalam makalah ini adalah jenis labirin yang memiliki satu pintu masuk dan satu pintu keluar.

Persoalan pencarian jalur (*pathfinding*) solusi pada labirin ini cukup sederhana, namun dapat menjadi persoalan dasar yang dapat dikembangkan untuk memecahkan masalah-masalah pencarian jalur yang lebih kompleks. Salah satu penyebab kesederhanaan dari persoalan labirin ini adalah karena jika tidak ada perputaran (*looping*), labirin pasti hanya memiliki satu jalur solusi, sehingga tidak terdapat jalur yang sub-optimal. Selain itu, pada labirin yang hanya memiliki satu pintu masuk dan pintu keluar, lokasi titik awal dan titik akhir dapat diketahui sebelum pencarian jalur dimulai, sehingga selain algoritma *uninformed search* seperti algoritma BFS dan DFS, dapat juga digunakan algoritma-algoritma *informed search* seperti algoritma A*.



Gambar 1.1 Contoh Labirin

Pemodelan labirin menjadi graf juga dapat dilakukan dengan beberapa cara, misalnya dengan menjadikan setiap petak sebagai satu simpul di graf dan semua sisi memilki bobot satu, namun cara ini akan memboroskan memori dan waktu komputasi. Cara lain adalah dengan membentuk satu simpul di setiap persimpangan labirin dan sisi-sisi yang menghubungkan simpul berbobot jarak di antara kedua simpul. Metode yang akan digunakan untuk pemodelan graf dari labirin dalam karya tulis ini adalah metode yang kedua. Namun, karena labirin yang akan dicari solusinya berasal dari gambar, akan digunakan juga pengolahan gambar (*image processing)* dalam pembentukan graf..

Secara umum, pencarian secara *informed search* akan menghasilkan solusi dengan lebih cepat daripada pencarian secara *uninformed search* karena pencarian dapat dilakukan dengan fokus ke arah tujuan. Karena itu, dalam karya tulis ini akan digunakan juga algoritma BFS dan DFS sebagai perbandingan.

II. LANDASAN TEORI

A. Graf

Graf digunakan untuk merepresentasikan objekobjek diskrit dan hubungan antara objek-objek tersebut. Representasi visual dari graf adalah dengan menyatakan objek sebagai noktah, bulatan, atau titik, sedangkan hubungan antar objek dinyatakan dengan garis.

Definisi graf secara matematis adalah sebagai pasangan himpunan (V, E), atau dapat ditulis dengan notasi:

$$G = (V, E)$$

dengan:

V = himpunan tidak-kosong dari simpul-simpul (vertices atau node) = { v1, v2, v3, ... , vn}

dan

E = himpunan sisi (edges atau arcs) yang menghubungkan sepasang simpul = {e1, e2, ..., en}

Dari definisinya, graf minimal memiliki satu simpul dan tidak harus memiliki sisi. Graf minimal seperti itu disebut graf trivial. Sisi pada graf juga dapat didefinisikan sebagai pasangan simpul yang dihubungkan sisi tersebut (misalnya (v1, v2)).

Graf yang akan digunakan dalam karya tulis ini untuk pemodelan labirin adalah graf sederhana tak-berarah yang memiliki bobot pada sisi-sisinya. Graf sederhana berarti dua simpul yang sama hanya bisa dihubungkan oleh maksimal satu sisi. Graf tak-berarah berarti sisi-sisi pada graf tidak memiliki orientasi arah sehingga urutan simpul yang dihubungkan oleh sisi tidak perlu diperhatikan. Bobot pada sisi-sisi graf berbobot mewakili ongkos (*cost*) atau jarak di antara dua simpul yang dihubungkannya.

Beberapa terminologi (istilah) khusus yang sering digunakan saat membahas graf adalah:

1. Bertetangga (*adjacent*)

   Dua simpul disebut bertetangga jika simpul tersebut dihubungkan oleh sebuah sisi. Jika sisi tersebut berarah, simpul tujuan adalah tetangga dari simpul asal, tapi belum tentu sebaliknya.

2. Derajat (*Degree*)

   Derajat dari suatu simpul adalah jumlah sisi yang terhubung dengan simpul tersebut. Dalam graf berarah, derajat dibagi menjadi derajat masuk dan derajat keluar untuk jumlah sisi yang menuju simpul dan jumlah sisi yang keluar dari simpul.

3. Lintasan (*Path*)

   Lintasan dari simpul awal ke simpul tujuan adalah barisan selang-seling antara simpul dan sisi yang dilewati di antara kedua simpul.

4. Siklus (*Cycle*) atau sirkuit (*Circuit*)

   Siklus atau sirkuit adalah lintasan yang berawal dan berakhir di simpul yang sama.

5. Terhubung (*Connected*)

   Suatu graf terhubung bila untuk setiap dua simpul pada graf terdapat lintasan yang menghubungkan keduanya.

6. Upagraf (*Subgraph*)

   Suatu graf G' merupakan upagraf dari graf G jika himpunan simpul pada G' merupakan subset dari himpunan simpul G dan himpunan sisi pada G' merupakan subset himpunan sisi G.

B. Pohon

1) Definisi Umum Pohon

Pohon merupakan subset dari graf sederhana tak-berarah terhubung yang tidak memiliki sirkuit di dalamnya. Karena pohon termasuk graf, pohon memiliki definisi matematis:

Jika

$$G = (V, E)$$

Dengan G = graf tak berarah sederhana dengan n buah simpul, pernyataan di bawah ini adalah ekuivalen:

1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan tepat satu lintasan.
3. G adalah graf terhubung yang memiliki n-1 buah sisi.
4. G tidak memiliki sirkuit dan memiliki n-1 buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. G adalah graf terhubung yang semua sisinya merupakan jembatan. (Jembatan adalah sisi yang apabila dihapus akan memecah graf menjadi dua komponen).

Pernyataan nomor 2-6 di atas dapat disebut juga sebagai definisi alternatif dari pohon.

2) Pohon Berakar

Pohon berakar adalah modifikasi dari pohon dengan menentukan satu simpul sebagai simpul akar dan menjadikan semua simpul lain dapat dicapai dari simpul akar dengan memberi arah pada sisi-sisi pohon yang mengikutinya sehingga menjadi graf berarah.

Simpul akar pada pohon berakar memiliki derajat masuk sama dengan nol dan semua simpul lain memiliki derajat masuk sama dengan satu. Simpul yang memiliki derajat keluar sama dengan nol disebut simpul daun dan simpul yang memiliki derajat keluar tidak sama dengan nol disebut simpul dalam atau simpul cabang. Setiap simpul pada pohon berakar dapat dicapai dari akar melalui lintasan yang tunggal (unik).

Pada implementasinya, pohon berakar sering memiliki urutan untuk anak-anaknya. Pohon yang demikian disebut sebagai pohon terurut (*ordered tree*).

Beberapa terminologi khusus pohon berakar adalah:

1. Anak (*child*) dan orangtua (*parent*).
   Suatu sisi di pohon berakar menghubungkan simpul orangtua ke simpul anak.
2. Lintasan (*Path*).
   Lintasan antara dua simpul adalah runtutan simpul-simpul terhubung yang dilewati untuk satu simpul mencapai simpul yang lain.
3. Keturunan (*descendant*) dan Leluhur (*ancestor*).
   Suatu lintasan dalam pohon berakar menghubungkan leluhur ke keturunannya.
4. Saudara kandung (*sibling*).
   Simpul-simpul yang memiliki orangtua sama disebut bersaudara kandung.
5. Aras (*level*)
   Akar mempunyai aras 0 dan semua simpul lain mempunyai aras sepanjang lintasan dari akar simpul.
6. Tinggi (*height*) atau kedalaman (*depth*).
   Aras maksimum dari suatu pohon berakar disebut tinggi atau kedalaman.
7. Upapohon (*subtree*)
   Upapohon dengan akar X dari pohon T adalah suatu upagraf T' yang mengandung X dan semua keturunannya dan sisi-sisi yang berasal dari X.
8. Derajat (*degree*).
   Derajat dari suatu simpul pada pohon berakar adalah jumlah simpul anak dari simpul tersebut, sehingga derajat yang dimaksud di sini merupakan derajat keluar.

C. *Algoritma Pencarian Jalur*

Terdapat banyak algoritma yang dapat digunakan dalm pencarian jalur, beberapa di antaranya adalah:

1) Algoritma BFS

Algoritma BFS (*Breadth First Search*) atau biasa juga disebut algoritma pencarian melebar adalah algoritma yang melakukan pencarian sesuai dengan urutan simpul anak dari suatu simpul. Ilustrasi dari urutan pencariannya adalah sebagai berikut:

1. Kunjungi simpul v.
2. Kunjungi semua simpul yang bertetangga dengan simpul v.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul yang sudah dikunjungi. Sampai pencarian selesai.

Jika graf berbentuk pohon berakar, maka semua simpul pada suatu aras akan dikunjungi terlebih dahulu sebelum simpul pada aras berikutnya dikunjungi.

Algoritma BFS memerlukan sebuah antrian untuk simpul-simpul yang sudah dikunjungi sebagai acuan untuk mengunjungi simpul-simpul yang bertetangga dengannya. Dibutuhkan juga suatu tabel *boolean* untuk menandai simpul yang telah dikunjungi agar tidak dikunjungi lagi.

Jika graf yang digunakan tidak berbobot, algoritma BFS bersifat komplit, yaitu pencarian pasti akan berakhir, dan optimal, yaitu pencarian pasti menghasilkan solusi terbaik. Jika graf berbobot, tidak ada jaminan solusi optimal. Selain itu, pencarian menggunakan BFS juga memiliki kompleksitas memori yang besar karena bisa saja semua simpul harus disimpan pada waktu yang sama.

Algoritma BFS yang digunakan di karya tulis ini akan menggunakan graf yang statis, berarti graf harus sudah tersedia (dibentuk dahulu) sebelum algoritma BFS bisa dimulai.

2) Algoritma DFS (Depth First Search)

Algoritma DFS (*Depth First Search*) atau biasa juga deisebut pencarian mendalam adalah algoritma yang mendahulukan pencarian ke simpul anak sebelum ke simpul saudara kandung suatu simpul. Ilustrasi dari proses pencariannya adalah sebagai berikut:

1. Kunjungi simpul v
2. Kunjungi simpul tetangga dari simpul v
3. Ulangi proses secara rekursif sampai mengunjungi simpul yang tidak memiliki tetangga yang belum dikunjungi.
4. Runut balik pencarian ke simpul yang masih memiliki tetangga yang belum dikunjungi.
5. Ulangi proses secara rekursif sampai pencarian selesai.

Pada pencarian dengan algoritma DFS, simpul pada aras yang sama tidak akan dikunjungi selama masih ada simpul keturunan yang belum dikunjungi.

Algoritma DFS membutuhkan suatu stack untuk simpul-simpul yang dikunjungi, atau suatu fungsi rekursi untuk memproses semua simpul yang dikunjunginya. Seperti algoritma BFS, algoritma DFS juga membutuhkan tabel *boolean* untuk menandai simpul-simpul yang sudah dikunjungi.

Solusi yang didapat dari pencarian DFS bersifat komplit, yaitu pencarian pasti berakhir, tapi tidak ada jaminan solusi yang didapat merupakan solusi optimal. Akan tetapi, kompleksitas memori dari pencarian menggunakan DFS relatif lebih kecil dar BFS, karena jumlah simpul maksimal yang dihidupkan pada suatu waktu adalah sebanyak kedalaman pohon.

3) Algoritma A (A-Star)*

Algoritma A* merupakan algoritma pencarian pada graf yang mencari jalur dari suatu titik awal menuju suatu titik akhir yang sudah diketahui. Algoritma ini berjalan seperti algoritma BFS, dengan adanya antrian (*queue)* simpul simpul untuk dikunjungi, namun dalam penentuan simpul berikutnya yang akan dikunjungi digunakan prioritas berdasarkan ongkos total yang dicari menggunakan fungsi heuristik.

Fungsi heuristik adalah fungsi yang berfungsi memberikan pendekatan untuk mengarahkan ke solusi sebenarnya. Karena berupa pendekatan, fungsi heuristik tidak selalu benar, tapi biasanya fungsi heuristik cukup baik untuk meningkatkan kecepatan proses pencarian.

Ilustrasi proses pencarian dengan A* adalah sebagai berikut:

1. Kunjungi simpul v.
2. Hitung perkiraan ongkos total untuk semua simpul yang bertetangga dengan simpul v.
3. Tambahkan simpul-simpul tetangga v ke dalam antrian berprioritas berdasarkan urutan perkiraan ongkosnya secara menaik.
4. Kunjungi simpul berikutnya dalam queue dan ulangi sampai antrian kosong.

Perkiraan ongkos total suatu simpul didapat dari menjumlahkan ongkos dari awal ke simpul tersebut dengan perkiraan ongkos dari simpul tersebut ke simpul akhir. Ongkos dari awal ke suatu simpul disimpan untuk setiap simpul yang sudah dilewati. Sementara, ongkos dari simpul tersebut ke simpul akhir diperkirakan menggunakan fungsi heuristik.

Algoritma A* secara teori pasti menemukan solusi yang optimal jika memang ada jalur yang menghubungkan kedua simpul. Jumlah simpul yang dikunjungi juga akan lebih sedikit dibandingkan menggunakan BFS karena hanya simpul yang tidak menuju tujuan tidak akan dikunjungi sehingga proses pencarian akan lebih cepat.

D. Pengolahan Gambar (Image Processing)

Pengolahan gambar adalah metode melakukan operasi-operasi pada suatu gambar untuk mendapatkan informasi yang berguna ari gambar tersebut. Pengolahan gambar mengambil sebuah masukan berupa gambar dan menghasilkan keluaran yang juga berupa gambar atau informasi lain yang dapat diasosiasikan dengan gambar masukan.

Proses dari pengolahan gambar dapat dibagi menjadi 3 langkah, yaitu:

1. Meng-*import* gambar
2. Analisis dan manipulasi terhadap gambar
3. Menghasilkan keluaran yang berupa gambar yang telah diubah atau informasi dari gambar.

Dalam karya tulis ini, informasi keluaran yang diarapkan adalah graf yang dibentuk berdasarkan gambar labirin yang dimasukkan.

III. IMPLEMENTASI

Implementasi dari algoritma pencarian jalan dari gambar labirin dimulai dengan pengolahan gambar untuk menghasilkan pemodelan graf dari labirin, kemudian dilanjutkan dengan penerapan algoritma.

A. Implementasi Pengolahan Gambar Labirin

Dalam karya tulis ini, pengolahan gambar labirin akan dibatasi untuk gambar labirin yang memenuhi kriteria berikut:

1. Terdiri dari dua warna, satu warna untuk dinding dan satu warna untuk jalan.
2. Lebar dinding labirin sama dengan lebar jalan, sehingga labirin bisa dimodelkan menjadi petak (*cell)*.
3. Jalan hanya menyentuh ujung gambar di pintu masuk dan pintu keluar.

Implementasi untuk pembuatan graf dari gambar labirin akan menggunakan algoritma BFS dan dituliskan dalam pseudocode berikut:

```
Image <- import image "maze.png"
Info <- extractMazeInfo(Image)
G <- createGraph(Info)

Function extractMazeInfo(Image)
for every border in Image do
   found <- search for entrance/exit paths
   if found then
        psize <- count path width
        msize <- count maze size in paths
        entrance <- entrance coordinate
        exit <- exit coordinate
Info <- (psize, msize, entrance, exit)
Return info

Function createGraph(Info)
listnode <- list of node
listedge <- list of edge
nodequeue <- a queue of (node, parent)
insert entranceNode to listnode
insert (entranceNode, null) to nodequeue
while nodequeue is not empty do
  currNode, parentDir <- pop from nodequeue
  for each direction except parentDir do
        x,y <- currNode
        while (x,y) is not a Node and is valid do
            move (x,y) at (direction)
        endwhile
        if (x,y) is not valid then
            move (x,y) at reverse(direction)
        endif
        if (x,y) not in listnode then
            N <- (createNode(x,y))
            V <- createEdge(currNode, N)
            insert N to listnode
            insert V to listedge
            insert (N, direction) to nodequeue
        endif
   endfor
endwhile
G <- graph of (listnode, listedge)
return G
```

Pengolahan gambar dimulai dengan menentukan di mana pintu masuk dan pintu keluar pada gambar labirin, kemudian ditentukan ukuran lebar jalan labirin dan ukuran labirin dalam satuan jumlah petak.

Setelah informasi tentang labirin diketahui, dimulailah penelusuran labirin dengan menggunakan algoritma BFS untuk menghasilkan graf G. Graf yang dihasilkan memiliki simpul di setiap ujung jalan dan persimpangan labirin dan setiap jalan di antara simpul diwakili satu sisi graf.

B. Implementasi Algoritma Pencarian

Setelah graf terbentuk, pencarian dengan algoritma pencarian dapat dimulai.

1) Implementasi BFS

Pseudocode untuk algoritma BFS adalah sebagai berikut:

```
Function BFS(graph)
Nodelist : list of nodes in graph
Edgelist : list of edges in graph
Nodequeue <- queue of nodes
currNode <- entranceNode
While currNode != goalNode do
    childNodes <- raise children of currNode
    insert childNodes to nodequeue
    currNode <- pop from nodequeue
endwhile
path <- currNode.path
return path
```

Setiap node di nodequeue juga menyimpan jalur yang dilewati dari entranceNode ke simpul tersebut saat node dibangkitkan dari node orangtua, sehingga ketika goalNode dicapai, jalur solusi bisa diambil dari simpul tujuan.

2) Implementasi DFS

Pseudocode untuk algoritma DFS adalah sebagai berikut:

```
Function DFS(graph, node)
Nodelist : list of nodes in graph
Edgelist : list of edges in graph
If  node is goalNode then
   Return path of node
Else
   nodePath <- path of node
   insert node to nodePath
   For each child of node do
        resultPath <- DFS(graph, child)
        If resultPath > nodePath then
                Break
   If resultPath <= nodePath then
        Pop from resultPath
   Return resultPath
```

Implementasi DFS untuk karya tulis ini dibuat secara rekursif, sehingga hanya perlu satu list untuk menyimpan jalur solusi yang sedang dibentuk. Setiap fungsi rekursi selesai, jika tidak mencapai jawaban, elemen terakhir dari list path dihapus dari list.

*3) Implementasi A**

Pseudocode untuk implementasi A* adalah sebagai berikut:

```
Function AStar(graph)
Nodelist : list of nodes in graph
Edgelist : list of edges in graph
Nodequeue <- priority queue of nodes
currNode <- entranceNode
While currNode != goalNode do
    childNodes <- raise children of currNode
    for each child in childNodes do
        compute total cost for child
        insert child to nodequeue based on cost
    endfor
    currNode <- pop from nodequeue
endwhile
path <- currNode.path
return path
```

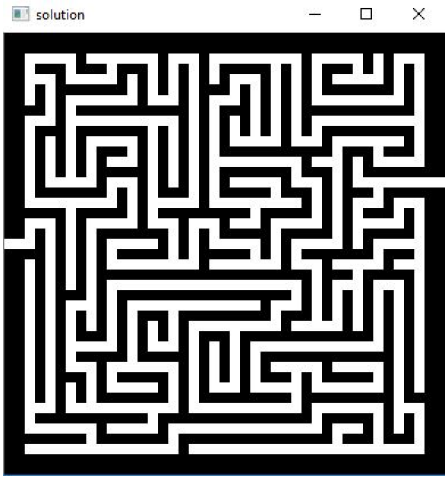
Implementasi dari algoritma A* di sini hampir sama dengan implementasi BFS, tetapi terjadi perubahan struktur data queue pada BFS yang diubah menjadi struktur data priority queue pada A*, sehingga simpul anak yang dibangkitkan dimasukkan ke queue sesuai urutan prioritas. Prioritas ditentukan berdasarkan fungsi heuristik.

IV. Hasil Percobaan

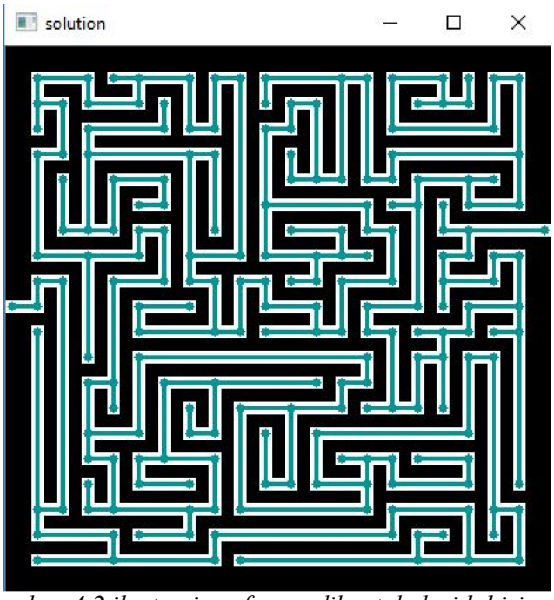
Percobaan dilakukan dengan implementasi menggunakan bahasa pemrograman Python 3.5 dengan kakas tambahan OpenCV. Program dijalankan pada sistem operasi windows 10 pada mesin berprosesor AMD A8-4555M 1.6GHz dan memori RAM 8 Gb.

A. Hasil Pengolahan Gambar

Pengolahan gambar yang dilakukan pada labirin menghasilkan graf yang benar dengan satu simpul di setiap persimpangan dan satu sisi menghubungkan setiap dua simpul yang memiliki jalan di antaranya.



Gambar 4.1 Contoh labirin masukan



Gambar 4.2 ilustrasi graf yang dibentuk dari labirin
Lingkaran = simpul graf dan garis = sisi graf

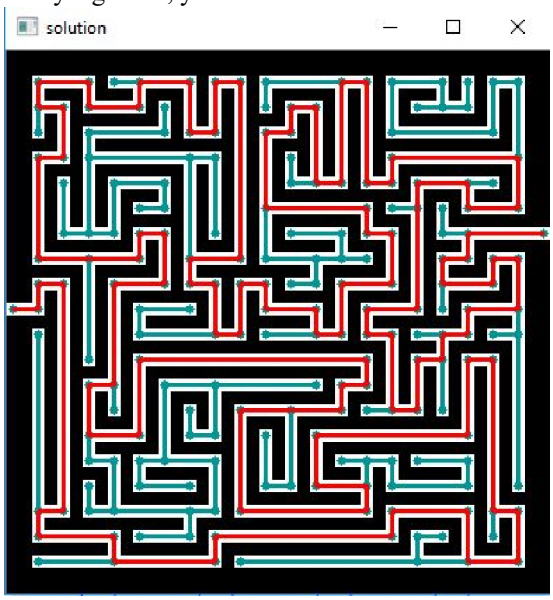
Data berikut untuk waktu proses pengolahan gambar:

Simpul yang dihasilkan: 206 simpul

Percobaan ke	Waktu (detik)
1	0.12909
2	0.11709
3	0.10101
4	0.19800
5	0.11201
Rata-rata	0.13144

B. *Hasil Algoritma Pencarian Jalan*

Karena hanya ada satu jalur solusi, setiap algoritma yang digunakan untuk pencarian jalur menghasilkan jawaban yang sama, yaitu:



Gambar 4.3 Jalur solusi Labirin

1) Algoritma BFS

Data untuk pencarian siolusi menggunakan algoritma BFS yang didapat dari percobaan dituliskan di tabel berikut.

Percobaan ke	Waktu (detik)
1	0.04505
2	0.07201
3	0.04708
4	0.04800
5	0.05901
Rata-rata	0.05423

2) Algoritma DFS

Data untuk pencarian siolusi menggunakan algoritma BFS yang didapat dari percobaan dituliskan di tabel berikut.

Percobaan ke	Waktu (detik)
1	0.01103
2	0.01400
3	0.01102
4	0.00999
5	0.01200
Rata-rata	0.01161

*3) Algoritma A**

Data untuk pencarian siolusi menggunakan algoritma A* yang didapat dari percobaan dituliskan di tabel berikut.

Percobaan ke	Waktu (detik)
1	0.04997
2	0.05101
3	0.05106
4	0.05098
5	0.06999
Rata-rata	0.05460

V. ANALISIS

A. *Pengolahan Gambar*

Demi kesederhanaan, pembuatan graf dari gambar labirin dalam karya tulis ini hanya menggunakan algoritma BFS. Kenyataannya, algoritma BFS mungkin bukan merupakan algoritma yang terbaik untuk pembentukan graf dari labirin.

Selain itu, algoritma BFS yang digunakan untuk membentuk graf juga tidak mengangani kasus jika labirin mengandung siklus. Karena dalam algoritma yang digunakan sisi ditambahkan setiap simpul baru dibentuk, jika dalam penelusuran labirin ditemukan simpul yang pernah dikunjungi, tidak akan dibentuk sisi untuk menghubungkan ke simpul terebut.

Untuk bisa menangani pembentukan graf yang memiliki siklus, algoritma dapat dimodifikasi untuk menentukan posisi semua simpul sebelum mulai

menentukan sisi-sisi untuk menghubungkan semua simpul tersebut. Pembentukan sisi perlu dilakukan jika simpul yang terhubung oleh jalan tidak memiliki sisi di antaranya.

B. *Pencarian Solusi Labirin*

Dari data yang diperoleh dari percobaan, dapat dilihat bahwa algoritma tercepat untuk pencarian jalur solusi labirin adalah algoritma DFS, yang diikuti oleh algoritma BFS dan kemudian algoritma A*. Tidak sesuai dengan perkiraan, ternyata algoritma A* berada di posisi terakhir dibandingkan kedua algoritma lainnya, meskipun tidak berbeda jauh dengan algoritma BFS.

Terdapat beberapa kemungkinan penjelasan untuk hasil percobaan ini, yaitu:

1. *Subset yang besar dari jalur solusi bergerak menjauhi simpul tujuan.*

   Karena fungsi heuristik yang digunakan algoritma A*, simpul tetangga yang lebih dekat ke simpul tujuan akan dikunjungi terlebih dahulu. Namun, pada contoh labirin yang digunakan untuk percobaan, potongan besar dari jalur solusi bergerak menjauhi simpul tujuan sehingga fungsi heuristik justru menjauhkan dari solusi sebenarnya dan solusi harus menunggu lebih lama untuk dikunjungi.

2. *Akumulasi waktu terbuang untuk pemrosesan tambahan pada A*.*

   Dalam algoritma A*, terdapat waktu terbuang untuk pemrosesan tambahan dibanding algoritma BFS dan DFS. Pemrosesan tambahan itu adalah perhitungan fungsi heuristik dan proses memasukkan simpul ke antrian berprioritas. Proses memasukkan simpul tersebut membutuhkan iterasi antrian dan perbandingan ongkos total dengan simpul-simpul lain di antrian sehingga cukup memakan waktu.

3. *Labirin yang membutuhkan penelusuran dalam dan kurang banyak cabang*

   Labirin yang digunakan sebagai contoh memiliki sedikit percabangan dan solusi yang dihasilkan relatif panjang. Fakta ini memberikan keunggulan kepada algoritma pencarian mendalam seperti DFS dibandingkan algoritma pencarian melebar seperti BFS dan A*, meskipun lebar pada pencarian A* sudah dibatasi dengan fungsi heuristik.

Semua alasan di atas dapat menjelaskan penyebab algoritma DFS jauh lebih cepat dibanding algoritma BFS dan algoritma A* dan juga penyebab algoritma A* sedikit lebih lambat dibanding algoritma BFS.

VI. Kesimpulan

Di antara tiga algoritma yang digunakan dalam karya tulis ini, yaitu algoritma BFS, DFS, dan A*, algoritma terbaik untuk mencari solusi dari labirin adalah algoritma DFS. Sementara untuk pengolahan gambar labirin, algoritma yang digunakan sudah cukup baik untuk memproses gambar labirin dengan batasan yang diberikan dan penggunaan algoritma BFS sudah cukup baik untuk menghasilkan pemodelan graf dari labirin.

VII. Ucapan Terima Kasih

Pertama, penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa karena berkat rahmat-Nya penulis dapat menyelesaikan karya tulis ini dengan tepat waktu. Penulis juga mengucapkan terima kasih kepada Bapak Rinaldi Munir sebagai pengajar yang telah membimbing penulis dalam mata kuliah IF2211 Strategi Algoritma selama satu semester ini. Ucapan terima kasih juga kepada teman-teman dan keluarga penulis yang telah mendukung penulis selama ini.

Daftar Pustaka

[1] R. Munir, *Diktat Kuliah Matematika Diskrit,* Bandung, Institut Teknologi Bandung, 2003

[2] R. Munir, *Diktat Kuliah Strategi Algoritma,* Bandung, Institut Teknologi Bandung, 2018

[3] G. Anbarjafari, *Digital Image Processing,* 2014, University of Tartu https://sisu.ut.ee/imageprocessing/book/1

[4] J. Lauro, http://hereandabove.com/maze/mazeorig.form.html, *Maze Maker,* 2018.

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, 13 Mei 2018

David Timothy Panjaitan
13516075