Penggunaan BFS untuk Fitur Petunjuk pada Permainan *Maze*

Winson Waisakurnia (13512071)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512071@std.stei.itb.ac.id

***Abstrak*—Pada permainan, terutama permainan teka-teki (puzzle) biasanya terdapat tombol petunjuk untuk membantu pemain yang sudah buntu. Pada makalah ini penulis akan memanfaatkan algoritma BFS untuk membangun petunjuk untuk permainan teka-teki *Maze*. Pemberian petunjuk ini akan memanfaatkan algoritma BFS dan petunjuk jalan yang diberikan kepada pemain adalah yang paling optimal.**

***Index Terms*—BFS, *maze,* petunjuk.**

I. PENDAHULUAN

Permainan *maze* merupakan permainan yang sudah berumur sangat tua, dipercaya ada sejak ribuan tahun yang lalu. Permainan ini masih banyak dimainkan oleh banyak kalangan, terutama anak-anak. Selain untuk dijadikan sebagai permainan, *maze* juga sering dipakai sebagai bahan belajar jurusan Teknik Informatika untuk mempelajari tentang penelusuran graf.

Penulis memilih topik ini karena penulis merasa masih banyak permainan *maze* yang dikembangkan dan permainan ini tidak tertelan oleh waktu. Permainan *maze* juga sering digunakan *game programmer* pemula untuk mempelajari cara membuat permainan komputer.

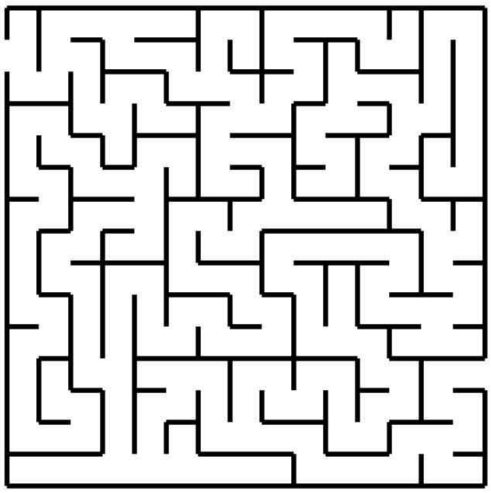
Maze dapat direpresentasikan dalam bentuk graf dan terdapat banyak algortima penelusuran graf. Namun, dari berbagai algoritma penelusuran graf yang ada, penulis memilih salah satu algoritma yang lebih unggul daripada yang lain dalam kasus ini yang tidak lain adalah BFS (*Breadth-first search*).

Diharapkan hasil dari makalah ini dapat membantu pengembang permainan *maze* untuk menambahkan fitur petunjuk ataupun memperbaiki fitur petunjuk pada permainan *maze* yang sudah ada. Hasil makalah ini juga diharapkan dapat dipakai secara luas oleh permainan lainnya yang menyerupai *maze* yang membutuhkan jarak terpendek dari posisi ke finsih.

II. TEORI DASAR

A. Maze

Maze adalah permainan teka-teki yang tersusun dari dinding dan jalan. Susunan dinding dan jalan tersebut membentuk sebuah struktur jalur yang bercabang-cabang dan kompleks.[2] Pemain kemudian diletakkan pada salah satu titik yang merupakan jalan masuk pada *Maze*, dan pemain akan berusaha keluar dari *maze* dengan menelusuri jalur-jalur yang ada.



Gambar 1. Maze (sumber: http://puzzle-house.co.uk/maze.jpg)

B. Graf

Graf adalah sebuah struktur yang terdiri dari himpunan simpul (node atau vertices) dan sisi (edge atau arc).[2] Kumpulan simpul dan sisi boleh disebut graf jika terdapat minimal satu simpul pada himpunan simpul sedangkan himpunan sisi boleh kosong. Graf yang hanya memiliki satu simpul tanpa sisi dinamakan graf trivial.

Sisi pada graf dapat memiliki arah. Berdasarkan ada tidaknya arah pada sisi graf, graf dibedakan menjadi dua jenis[4]:

1. Graf tak-berarah (*undirected graph*)
   Graf berarah adalah graf yang sisinya tidak memiliki arah. Pada graf tak berarah, urutan simpul pada penulisan sisi tidak penting. Sisi (v_j,v_k) dengan sisi (v_k,v_j) adalah sama dan kita hanya perlu menuliskan salah satunya pada himpunan sisi jika

hanya satu sisi yang menghubungkan v_k dan v_j.

2. Graf berarah (*directed graph* atau *digraph*)
   Graf berarah adalah graf yang sisinya diberikan arah. Pada graf berarah sisi lebih sering disebut busur. Busur (v_j,v_k) berbeda dengan busur (v_k,v_j). Jika (v_j,v_k) adalah sisi berarah/busur, maka v_j adalah simpul asal (*initial vertex*) dan v_k adalah simpul terminal (*terminal vertex*). Pada busur (v_j,v_k) terdapat jalur dari v_j ke v_k tapi tidak sebaliknya.

Beberapa terminologi (istilah) yang berkaitan dengan graf[4]:

1. Bertetangga (*Adjacent*)
   Dua simpul a dan b pada graf tak-berarah dapat dikatakan bertetangga apabila terdapat sebuah sisi yang menghubungkan simpul a dan b. Pada graf berarah simpul a dikatakan bertetangga dengan simpul b apabila terdapat sisi yang menghubungkan a dan b dengan a sebagai simpul asal dan b sebagai simpul terminal dan b dapat disebut tetangga dari a.
2. Bersisian (*Incident*)
   Untuk sembarang sisi e = (v_j,v_k), sisi e dikatakan bersisian dengan simpul v_j dan v_k.
3. Graf Kosong (*Null Graph* atau *Empty Graph*)
   Graf yang himpunan sisinya merupakan himpunan kosong disebut sebagai graf kosong, E={}.
4. Derajat (*Degree*)
   Derajat suatu simpul graf tak berarah adalah jumlah sisi yang bersisian dengan simpul tersebut. Derajat suatu simpul dari graf berarah dibedakan menjadi dua, derajat masuk dan derajat keluar. Derajat masuk adalah jumlah busur yang masuk ke simpul. Derajat keluar adalah jumlah busur yang keluar dari simpul.
5. Lintasan (*Path*)
   Lintasan yang panjangnya n dari simpul awal v_0 ke simpul akhir v_n pada graf G ialah dapat ditulis sebagai barisan simpul $v_0,v_1,v_2,\ldots,v_n$ dengan syarat terdapat sisi (v_0, v_1), (v_1, v_2), (v_2, v_3),…, (v_{n-1}, v_n). Pada graf tak-berarah, sisi (v_0, v_1) sama dengan sisi (v_1, v_0).
6. Siklus (*Cycle*) atau Sirkuit (*Circuit*)
   Siklus adalah lintasan yang berawal dan berakhir pada simpul yang sama.
7. Terhubung (*Connected*)
   Simpul v_i dan v_j dikatakan terhubung apabila terdapat lintasan dari v_i ke v_j. Graf tak-berarah dikatakan graf terhubung adalah graf yang setiap pasangan simpul pada grafnya terhubung. Graf berarah dikatakan terhubung jika graf tak-berarahnya merupakan graf terhubung.

C. BFS

BFS (*Breath-First Search*) adalah algoritma penelusuran pada graf. Kedua algortima ini memulai pencarian pada graf dimulai dari sebuah simpul. Dari simpul awal tersebut akan diakses data pada simpul tersebut dan kemudian mengakses simpul selanjutnya yang bersebelahan dengan simpul yang sudah pernah dikunjungi.

Breadth-first search adalah salah satu algoritma paling sederhana untuk pencarian pada graf dan merupakan pola dasar untuk banyak algoritma graf yang penting.[3] Algorima Prim untuk *minimum-spanning-tree* dan algoritma Dijkstra untuk *single-source shortest-path* juga menggunakan ide yang sama dengan BFS.[3]

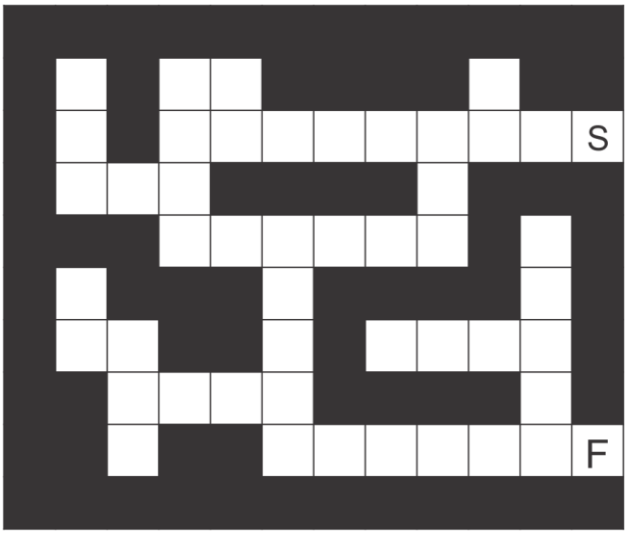
BFS biasanya diimplementasikan dengan memanfaatkan struktur data *queue*. Simpul yang dikunjungi dimasukkan ke dalam *queue* dan diwarnai. Simpul yang dikeluarkan dari *queue* akan diekspansi sampai pada kondisi *queue* menjadi kosong dan semua simpul yang dapat dikunjungi sudah telusuri. Untuk mengetahui apakah suatu simpul sudah pernah dikunjungi, belum dikunjungi, atau sedang diekspansi, biasanya dilakukan pewarnaan pada graf.

Pseudocode BFS:

```
1.  procedure BFS(G,v)
2.      buat queue Q
3.      tandai v sudah pernah dikunjungi
4.      tambahkan simpul v ke dalam queue
5.      while Q tidak kosong do
6.          w adalah elemen yang dikeluarkan dari queue
7.          if w adalah yang dicari then
8.              return w
9.          for semua simpul x yang bersebelahan dengan simpul w do
10.             if x tidak pernah dikunjungi then
11.                 tandai x sudah pernah dikunjungi
12.                 tambahkan simpul x ke dalam queue
```

III. PENGGUNAAN BFS UNTUK HINT PADA PERMAINAN *MAZE*

A. Representasi Maze dalam Bentuk Graf



Gambar 2. Graf Awal yang Disegmentasi

Maze dapat direpresentasikan dalam bentuk graf. Jalur pada *maze* dapat dibagi menjadi segmen-segmen yang dapat dibedakan satu sama lain seperti pada *Gambar 2*. Untuk setiap petak jalan yang ada pada graf, berikan sisi yang menghubungkan petak tersebut dengan petak jalan yang bersebelahan tepat di kiri, kanan, atas, dan bawah. Petak *start* yang dilambangkan dengan 'S' menjadi node awal dan petak *finish* yang dilambangkan dengan 'F' menjadi simpul tujuan. Graf yang dihasilkan dari segmenasi *maze* pada *Gambar 2* adalah graf tidak berarah dan tidak berbobot yang memiliki sirkuit.

B. Pembangunan Hint

Setelah merepresentasikan *maze* dalam bentuk graf, tahapan selanjutnya adalah menemukan algoritma yang cocok untuk membangun fitur petunjuk pada permainan *maze*. Petunjuk yang akan dibangun adalah petunjuk yang optimal, yang memberikan satu langkah yang merupakan bagian dari jalan terpendek menuju *finish*.

Penemuan jalan terpendek dari suatu simpul ke simpul yang tujuan dapat menggunakan beberapa algoritma *shortest-path* yang sudah ada. Petunjuk pada permainan dapat memberikan pilihan langkah yang optimal antara empat arah yang ada dengan menjalankan algoritma *shortest-path* dari simpul-simpul yang berada pada kiri, kanan, atas, dan bawah dari simpul tempat player berdiri sekarang. Simpul yang memberikan jalur paling pendek ke tujuan merupakan simpul yang akan disarankan kepada pemain pada saat pemain menekan fitur petunjuk. Namun jika pencarian jalur terpendek dari empat arah terus dilakukan selama permainan, maka permainan maze ini akan menjadi lebih macet karena *loading*. Walaupun hal tersebut tidak terasa jika *maze* yang dipakai masih berukuran kecil.

Solusi lainnya adalah dengan melakukan pencarian *shortest-path* dari simpul tujuan ke semua simpul lainnya sehingga. Dengan cara tersebut, dapat diketahui jarak setiap simpul dari simpul tujuan. Setelah dilakukan pencatatan jarak semua simpul dari simpul tujuan dengan satu kali penelusuran, tidak lagi diperlukan pencarian *shortest-path* setiap kali penekanan tombol petunjuk.

Jika *maze* mempunyai lebih dari satu simpul tujuan, penelusuran *shortest-path* dapat dilakukan sebanyak simpul tujuan yang ada dengan memanfaatkan tabel jarak setiap simpul yang sudah ada dengan pembatas "jika jarak dari simpul tersebut ke simpul tujuan yang lain lebih dekat daripada dari simpul tujian sekarang, maka jarak terpendek dari simpul tersebut dibatalkan.

C. Pemilihan Algoritma

Yang dibutuhkan untuk mengimplementasi algoritma untuk menghasilkan petunjuk adalah algoritma *single-source shortest-path* dengan simpul tujuan sebagai *source* dan simpul lainnya sebagai simpul yang ingin diketahui jaraknya dari *source*. Terdapat beberapa algoritma *single-source shortest-path* yang sering digunakan yaitu BFS, Dijkstra, Bellman-Ford.

BFS lebih baik dalam menyelesaikan persoalan pencarian jalur terpendek dengan graf yang tidak memiliki bobot dengan kadalaman pencarian dapat digunakan sebagai jarak. BFS merupakan algoritma yang cukup sederhana dan mudah diimplementasikan. Kompleksitas algoritma BFS adalah O(V+E) dengan V adalah jumlah simpul dan E adalah jumlah sisi[3]. Dijkstra merupakan algoritma yang cocok untuk *single-source shortest-path* yang memiliki bobot. Algoritma Dijkstra memiliki kompleksitas O(V logV+E).[3] Kelemahan dari algoritma Dijkstra adalah tidak mampunya mengatasi sirkuit negatif. Bellman-Ford dapat menanggulangi sirkuit negatif Dijkstra. Bellman-Ford memiliki kompleksitas O(VE).[3]

Algoritma yang lebih baik dibayar dengan kompleksitas yang lebih besar sehingga untuk menyelesaikan masalah cukup menggunakan algoritma yang tidak rumit namun dapat menyelesaikan persoalan. Dari pertimanagan tersebut maka BFS dipilih untuk mendapatkan *single-source shortest-path* dari simpul tujuan sebagai fitur petunjuk pada permainan *maze*.

D. Implementasi

Pertama lakukan pambacaan ukuran dan inisialisasi pada array dua dimensi, hintTable, yang digunakan untuk menyimpan jumlah langkah. Karena hanya membutuhkan sebuah tabel berukuran baris x kolom dan sebuah array yang menampung daftar finsih, kompleksitas ruang dari algoritma ini adalah O(BK) + O(B+K) dan kompleksitas secara keseluruhannya adalah O(BK). Karena banyaknya V(simpul) adalah baris x kolom, maka kompleksitas ruangnya dapat ditulis O(V).

```
FOR i=0..baris
  FOR j=0..kolom
    hintTable[i][j] ← MAXVAL
```

MAXVAL adalah nilai yang sangat besar dan tidak dapat dicapai pada saat perhitungan *shortest-path*. Variabel *baris* merupakan banyaknya baris pada maze dan *kolom* menyatakan banyaknya kolom pada maze yang telah disegmentasi.

Selanjutnya, masukkan data ke dalam hintTable dan daftarFinish yang mencatat semua lokasi tujuan. Kedua tabel inilah yang kemudian akan diolah untuk mendapatkan jalur terpendek.

```
1.  n_finish ← 0
2.  FOR i = 0..baris
3.    FOR j = 0..kolom
4.      input(c)
5.      IF c = '#' THEN
6.        hintTable[i][j] ← -1
7.      ELSE IF c = 'F' THEN
8.        t_lokasi temp
9.        temp.x ← j
10.       temp.y ← i
11.       daftarFinish[n_finish] = temp
12.       n_finish ← n_finish + 1
```

Pada algoritma ini graf direpresentasikan dengan hintTable. Setiap sel merupakan edge. Setiap sel pada hintTable memiliki edge yang menghubungkan ke sel yang terletak pada posisi kiri, kanan, atas, dan bawah sel tersebut.

Setelah mendapatkan seluruh data, Lakukan pencarian *shortest-path* dari simpul tujuan ke semua simpul lainnya dengan menggunakan algoritma BFS. Pseudocode BFS yang digunakan untuk menghasilkan banyak langkah dari setiap sel ke finish dapat dilihat pada kotak dibawah ini. Kompleksitas penghasilan jarak terpendek pada setiap sel tersebut adalah O(V+E), kompleksitas algoritma BFS.

```
1. moveX[] ← {1,-1,0,0}
2. moveY[] ← {0,0,1,-1}
3. FOR n = 0 .. n_finish
4.   Queue Q
5.   Q.ENQUEUE(daftarFinish[n])
6.   hintTable[daftarFinish[n].y][daftarFinish[n].x]
   ← 0
7.   WHILE NOT Q.EMPTY() DO
8.     t_lokasi depan ← Q.FRONT()
9.     Q.POP()
10.    FOR i= 0 .. 4
11.      nextX ← depan.x + moveX[i]
12.      nextY ← depan.y + moveY[i]
13.      IF nextX < kolom AND nextX >=0 AND
14.        nextY < baris AND nextY >= 0 THEN
15.        IF hintTable[nextY][nextX] >
   hintTable[depan.y][depan.x]+1 THEN
16.          hintTable[nextY][nextX] ←
   hintTable[depan.y][depan.x]+1
17.          t_lokasi temp
18.          temp.x ← nextX
19.          temp.y ← nextY
20.          Q.PUSH(temp)
```

Jika pemain menekan tobol petunjuk maka akan dilakukan pengecekan tabel dengan kompleksitas O(1) yang mempertimbangkan langkah ke kiri, kanan, atas, dan bawah dengan algoritma sebagai berikut:

```
1.  best_move ← -1
2.  FOR i= 0 .. 4
3.    nextX ← posisi_player.x + moveX[i]
4.    nextY ← posisi_player.y + moveY[i]
5.    IF nextX < kolom AND nextX >=0 AND
6.      nextY < baris AND nextY >= 0 THEN
7.      IF best_move = -1 OR
      hintTable[best_move.y][best_move.x] <
      best_move THEN
8.        best_move ← i
```

Variabel best_move menyatakan arah yang menjadi hint bagi pemain yang merupakan satu langkah dari jalur terpendek menuju tujuan. Korespondensi nilai best_move dengan arah dengan konstanta yang terdapat pada pseudocode BFS dapat dilihat pada *Tabel I* Jika best_move adalah -1 maka dapat disimpulkan tidak mungkin ada jalan menuju tujuan.

best_move	arah
0	Kanan
1	Kiri
2	Bawah

3	Atas
-1	-

Tabel 1. Korespondensi nilai best_move dengan arah

E. Pengujian

Pengujian dilakukan untuk memastikan algoritma tersebut dapat berjalan dengan baik. Pengujian dilakukan dengan menggunakan *maze* ukuran baris 10 dan ukuran kolom 12 dengan dua tujuan. *Maze* pengujian dapat dilihat dari *Gambar 2* dan menambahkan satu tujuan tambahan.

```
10 12
#F##########
#.#..####.##
#.#........S
#...####.###
###......#.#
#.###.####.#
#..##.#....#
##....####.#
##.##......F
############
```

Gambar 3. Input Maze

Dari input tersebut didapatkan tabel jarak dari setiap sel ke tujuan. Tabel tersebut yang kemudian akan digunakan untuk menghasilkan petunjuk dengan kompleksitas O(1) dengan melakukan pengecekan ke kiri, kanan, atas, dan bawah dari posisi pemain pada saat penekanan tombol petunjuk.

```
C:\Users\Winson\Desktop\stima>g++ -o maze mazeHint.cpp

C:\Users\Winson\Desktop\stima>maze < maze.txt
 -1   0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
 -1   1  -1   7   8  -1  -1  -1  -1  13  -1  -1
 -1   2  -1   6   7   8   9  10  11  12  13  14
 -1   3   4   5  -1  -1  -1  -1  12  -1  -1  -1
 -1  -1  -1   6   7   8   9  10  11  -1   5  -1
 -1  13  -1  -1  -1   9  -1  -1  -1  -1   4  -1
 -1  12  11  -1  -1   8  -1   6   5   4   3  -1
 -1  -1  10   9   8   7  -1  -1  -1  -1   2  -1
 -1  -1  11  -1  -1   6   5   4   3   2   1   0
 -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
```

Gambar 4. Tabel output shortest-path BFS

Jika player berada pada posisi baris 3 kolom 4 dan menekan tombol petunjuk, maka berdasarkan *Gambar 4* dilihat nilai disekitar sel tersebut seperti pada *Gambar 5* dan memberikan petunjuk ke arah bawah. Jika terdapat lebih dari satu opsi jalur terpendek dari posisi pemain maka algoritma ini akan memberikan salah satu opsi. Namun algoritma ini dapat dimodifikasi jika ingin memberikan dua opsi tanpa perubahan yang banyak.

```
-1   7   8
-1   6   7
 4   5  -1
```

Gambar 5. Perbesaran tabel baris 3 kolom 4

V. KESIMPULAN

Algoritma BFS cukup efisien dan efektif untuk membuat hint pada permainan *maze*. Hint yang didapatkan adalah hint yang paling optimal. Langkah yang diberikan merupakan bagian dari jalur terpendek menuju final. Kompleksitas waktu algoritma penghasilan tabel jarak terpendek yang dipakai adalah O(V+E) dan kompleksitas ruangnya adalah O(V). Kompleksitas waktu untuk mendapatkan jalan terbaik sebagai hint adalah O(1).

Algoritma ini dapat dimodifikasi dan disesuaikan dengan fitur-fitur permainan yang ada. Selain permainan *maze* algoritma ini juga dapat digunakan untuk permainan yang memiliki titik mulai dan tujuan serta jalan terpendek menuju tujuan merupakan hal yang penting untuk dipertimbangkan.

VI. PENGHARGAAN

Penulis bersyukur kepada Tuhan yang telah memberikan kemampuan kepada penulis untuk dapat menyelesaikan makalah ini. Terima kasih juga kepada orang tua yang telah membesarkan penulis hingga dapat berkarya saat ini. Penulis juga berterima kasih kepada dosen mata kuliah Strategi Algoritma semester genap 2013/2014, Dr. Ir. Rinaldi Munir, MT. dan Dr. Masayu Leylia Khodra, ST., MT. yang telah memberikan bimbingan. Tidak lupa juga penulis berterima kasih kepada teman-teman yang telah memberikan inspirasi untuk makalah ini.

REFERENSI

[1] Munir, Rinaldi, *Strategi Algoritma*, 4th, Teknik Informatika ITB, 2006.

[2] Hermann Kern, *Through the labyrinth: designs and meanings over 5000 years*, 2000.

[3] Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford, *Introduction to Algorithms*, 3rd, MIT Press and McGraw-Hill, 2009.

[4] Munir, Rinaldi, *Matematika Diskrit*, 4th, Teknik Informatika ITB, 2006.

[5] Rosen, Kenneth H., *Discrete Mathematics and Its Applications*, 7th, McGraw-Hill, 2012.

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, 19 Mei 2014

Winson Waisakurnia
13512071