Jurnal TIMES , Vol. IV No 1 : 1-5 , 2015
ISSN : 2337 - 3601

Maze Generator Dengan Menggunakan Algoritma Depth-First-Search

Octara Pribadi, S.Kom
Program Studi Teknik Informatika, STMIK TIME Medan
Jln. Merbabu No 32 AA-BB Medan
Telp. 061-4561932 , Email : octarapribadi@gmail.com

Abstrak

Labirin atau *maze* merupakan tempat yg penuh dengan jalan dan lorong yg berliku-liku dan simpang siur dan dipisahkan oleh tembok. Labirin seringkali dijadikan tantangan dalam permainan seperti puzzle, dimana terdapat objek dalam posisi awal harus menemukan jalan keluar pada posisi yang ditentukan. Dalam ilmu komputer, terdapat beberapa algoritma yang dapat digunakan untuk membuat sebuah labirin misalnya Recursive Backtracker, Kruskal's Algorithm, Prim's Algorithm, dan Depth-First-Search [1]. Depth-First-Search atau DFS merupakah salah satu cara paling mudah dalam membuat sebuah labirin yang tidak terlalu kompleks.

Kata Kunci : Labirin, Depth-First-Search

1. Pendahuluan

Labirin merupakan *puzzle* yang terdiri dari beberapa jalur yang bercabang-cabang, dimana penyelesaian *puzzle* tersebut harus mencari jalan yang sah, dimulai dari titik mulai hingga titik selesai [2]. Sejarah labirin dipercaya sebagian orang berasal dari zaman kuno dimana labirin merupakan struktur bangunan berliku yang dibuat Daedalus untuk raja Minos untuk memenjarakan Minotaur[1]. Labirin yang dibuat Daedalus memiliki sifat unicursal[2].

Dalam kehidupan sehari-hari terdapat beberapa aplikasi labirin yang dapat dijumpai terutama dalam industry *game*, karena labirin menantang pemain untuk mencari jalan keluar dari titik yang ditentukan sebelumnya. Selain itu dibeberapa Negara dibuat labirin dengan ukuran manusia dan menantang orang-orang untuk masuk kedalamnya, sebagai salah satu atraksi untuk menarik wisatawan.

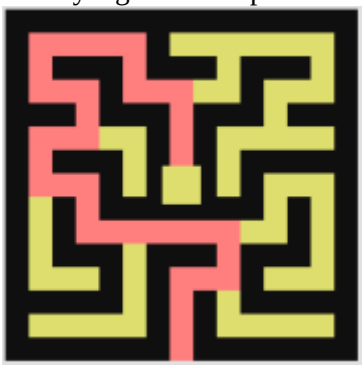
Walter D. Pullen membagi labirin menjadi beberapa jenis berdasarkan properti antara lain [3]:

1. *Dimension*
2. *Hyper-dimension*
3. *Topology*
4. *Tessellation*
5. *Routing*
6. *Texture*
7. *Focus*

Berdasarkan properti *routing* , terdapat 5 jenis labirin yaitu :

1. *Perfect*

   Labirin *perfect* memiliki sifat yaitu tidak terdapat jalan yang berulang(*loop*) selain itu tidak ada sel yang terisolasi pada labirin.



Gambar 1.1 Labirin *Perfect*

2. *Braid*

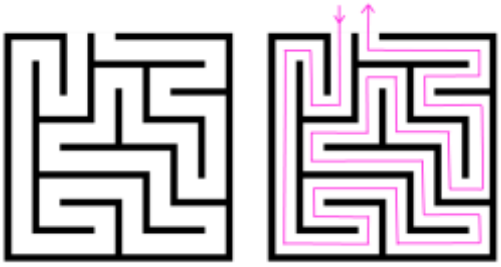
   Labirin *braid* tidak memiliki jalan buntu, semua jalan terhubung satu sama lain, sehingga dapat meningkatkan tingkat kesulitan pada labirin.

Gambar 1.2 Labirin *Braid*

3. *Unicursal*

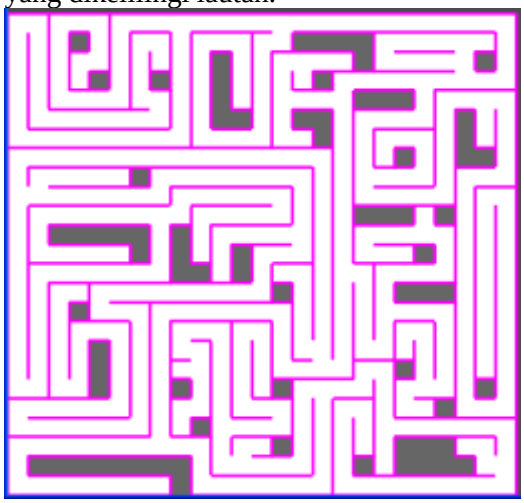
   Labirin *unicursal* adalah labirin yang tidak memiliki persimpangan atau cabang, sehingga untuk mencapai titik selesai dari titik mulai hanya perlu mengikuti jalan yang telah disediakan.

[1] Makhluk mistis setengah kerbau yang berasal dari zaman yunani kuno.
[2] Labirin yang tidak memiliki cabang maupun pilihan, dan hanya memiliki sebuah jalan yang panjang dari titik mulai ke titik selesai.



Gambar 1.3 Labirin *Unicursal*

4. *Sparseness*
   Labirin *sparseness* merupakan labirin dimana tidak seluruh dindingnya dibuat jalan sehingga seolah-olah seperti pulau yang dikelilingi lautan.



Gambar 1.4 Labirin *Sparseness*

5. *Partial Braid*
   Labirin *partial braid* merupakan labirin dengan kombinasi antara jalan buntu, dan jalan perulangan (loop) sehingga meningkatkan kesulitan dalam mencari jalan.

Algoritma DFS merupakan algoritma yang digunakan untuk membuat sebuah labirin *perfect*.

2. Landasan Teori

Dalam teori graf, untuk menelusuri pohon dapat menggunakan algoritma *Depth-First-Search*(DFS) dan *Breadth-First-Search*(BFS).

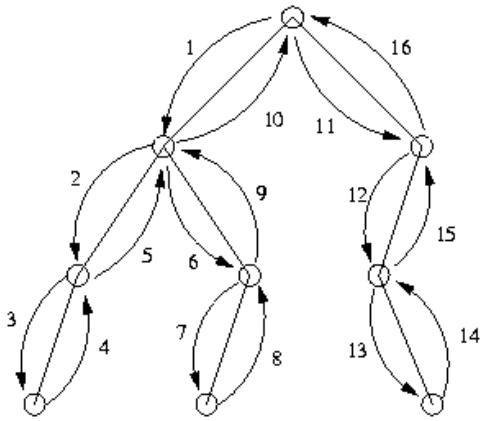
2.1. Depth-First-Search

Menurut Hafid Inggiantowi, DFS adalah pencarian yang berjalan dengan meluaskan anak akar pertama dari pohon pencarian yang dipilih dan berjalan dalam dan lebih dalam lagi sampai simpul tujuan ditemukan, atau sampai menemukan simpul yang tidak punya anak. Kemudian, pencarian *backtracking*, akan kembali ke simpul yang belum selesai ditelusuri [4].

Berikut merupakan beberapa hal tentang DFS yaitu :

1. *Traversal* (penjalaran simpul) dari suatu graf G akan :
   a. Mengunjungi semua simpul dan sisi dari G
   b. Menentukan apakah G terhubung
   c. Memperhitungkan komponen terhubung dari G
   d. Memperhitungkan pohon merentang dari G
2. DFS pada graf dengan n simpul dan m sisi membutuhkan waktu O(n+m)
3. DFS dapat lebih jauh diperluas untuk menyelesaikan masalah-masalah pada graf seperti :
   a. Menemukan dan melaporkan lintasan antara dua simpul yang diberikan
   b. Menemukan sirkuit pada graf
4. DFS untuk menggambar graf Euler ke pohon biner

Berikut merupakan contoh *tree traversal* dari graf G menggunakan algoritma DFS :



Gambar 2.1 Tree traversal graf G dengan DFS

Pada gambar 2.1 terlihat bahwa DFS mengunjungi simpul atas (*ancestor*) dan akan terus menelusuri simpul di bawah-kirinya-nya (*descendant*) hingga simpul yang tidak lagi memiliki cabang(nomor 1, 2, dan 3) kemudian DFS akan melakukan *backtrack* ke simpul atas (nomor 4 dan 5) dan menelusuri simpul dibawah-kanan-nya hingga simpul paling bawah tidak memiliki cabang, kemudian DFS akan melakukan *backtrack* kembali ke simpul atas dan seterusnya hingga kembali lagi ke simpul paling atas.

Salah satu cara dalam mengsimulasikan DFS adalah dengan menggunakan “Stack” yang memiliki sifat LIFO (Last In First Out).

2.2. Breadth-First-Search

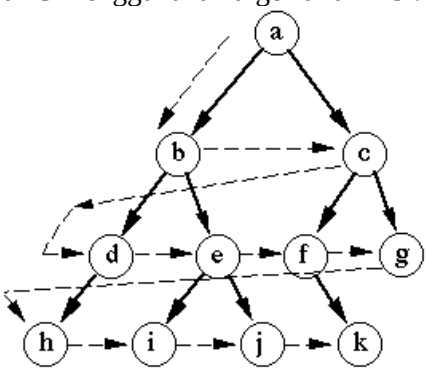
Menurut Hafid Inggiantowi, BFS adalah Algoritma pencarian grafnya dimulai dari simpul akar yang kemudian menelusuri semua simpul tetangganya. Kemudian, untuk setiap simpul yang terdekat itu, ditelusuri simpul tetangga yang belum diperiksa, dan demikian seterusnya, sampai ditemukan simpul tujuan [4].

BFS merupakan metode pencarian yang bertujuan memperluas dan memeriksa semua simpul pada graf G tanpa memikirkan tujuan sampai akhirnya akan menemukan tujuan itu sendiri. Dari sudut algoritma, semua simpul anak yang didapat dengan memperluas simpul ditambahkan ke FIFO (First In First Out) *queue* (antrian). Simpul yang belum diperiksa tetangganya diletakkan pada suatu penampung terbuka(*open*) dan jika telah diperiksa akan dimasukan kedalam penampung tertutup (*close*)[3].

Beberapa hal mengenai *tree traversal* BFS yaitu :

1. Traversal BFS dari suatu graf G akan :
   a. Mengunjungi semua simpul dan sisi dari graf G
   b. Menentukan apakah G terhubung
   c. Memperhitungkan komponen terhubung dari G
   d. Memperhitungkan pohon merentang dari G
2. BFS pada graf G dengan n simpul dan m sisi membutuhkan waktu O(n+m)
3. BFS dapat lebih jauh diperluas untuk menyelesaikan masalah-masalah pada graf seperti :
   a. Menemukan dan melaporkan lintasan dengan angka minimum dari sisi antara dua simpul yang diberikan
   b. Menemukan sirkuit sederhana pada graf jika ada

Berikut merupakan contoh *tree traversal* dari graf G menggunakan algoritma BFS :



Gambar 2.2 Tree Traversal graf G dengan BFS

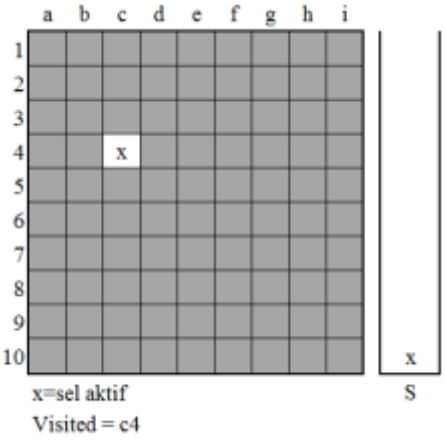
Pada gambar 2.2 terlihat bahwa BFS mengunjungi terlebih dahulu seluruh simpul cabang dari kiri ke kanan, kemudian BFS akan mengunjungi setiap cabang lagi dari simpul cabang yang telah dikunjungi sebelumnya, dan seterusnya hingga seluruh simpul dikunjungi.

3. Metode Penelitian

Untuk membuat labirin dengan metode DFS, secara garis besar dapat dibuat sebagai berikut :

1. Pilih sembarang sel X dan tandai sebagai sel aktif dan *visited*(telah dikunjungi).
2. Selama ada sel yang belum dikunjungi (*unvisited cells*) maka :
   a. Jika sel aktif X memiliki tetangga T[4] yang belum dikunjungi maka :
      i. Pilih acak salah satu T
      ii. Push(T) kedalam Stack S
      iii. Hapus dinding pemisah antara sel X dan T
      iv. Tandai T sebagai aktif dan *visited*
   b. Selain itu jika Stack S tidak kosong maka
      i. Pop() dari Stack S
      ii. Tandai keluaran Pop() sebagai sel aktif
   c. Selain itu maka
      i. Pilih acak sel yang belum dikunjungi, dan tandai sebagai sel aktif dan *visited*

Berikut merupakan ilustrasi dari algoritma diatas :

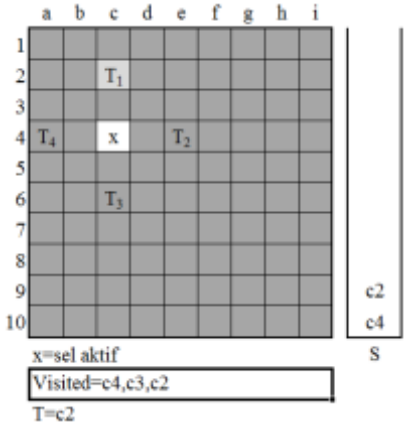


Gambar 3.1 Pilih acak sel dan tandai aktif

Pada gambar 3.1 pilih secara acak salah satu sel dalam *grid* dan tandai sebagai aktif dan *push* kedalam Stack S.

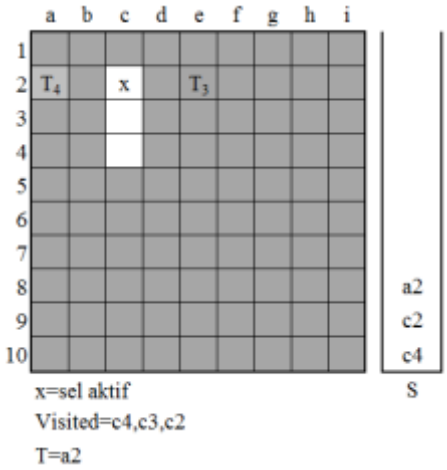
[3] Penampung dapat berupa sebuah larik ataupun list.

[4] Sel tetangga T merupakan sel diseberang sel X yang terdiri dari sel T kiri, T kanan, T atas, dan T bawah. Antara sel X dan sel T terdapat pemisah dinding.



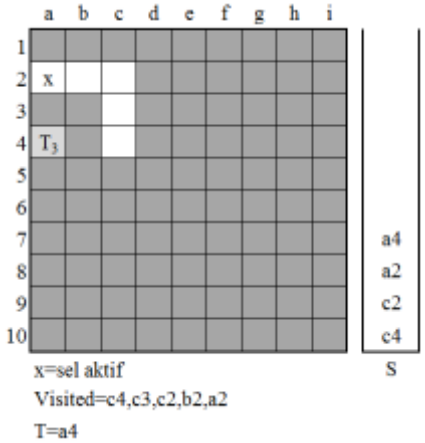
Gambar 3.2 Acak tetangga dan push(T)

Pada gambar 3.2 dari sel aktif atau X, acak salah satu dari kemungkinan sel tetangga T yang belum dikunjungi dan push ke dalam S, kemudian tandai sel T beserta sel antara T dan X dengan *visited*.

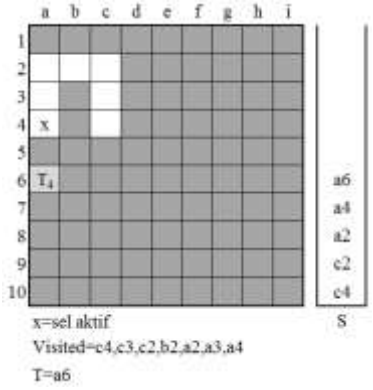


Gambar 3.3 Tandai T beserta sel diantara T dan X sebagai *visited*

Pada gambar 3.3 , pindahkan posisi aktif ke sel posisi teratas dari S. Dari sel X pilih acak sel tetangga T dan push kedalam S.

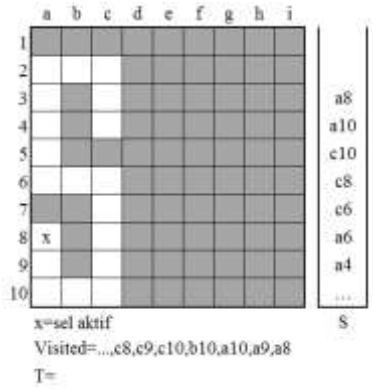


Gambar 3.4 Acak Tetangga dan Push(T)

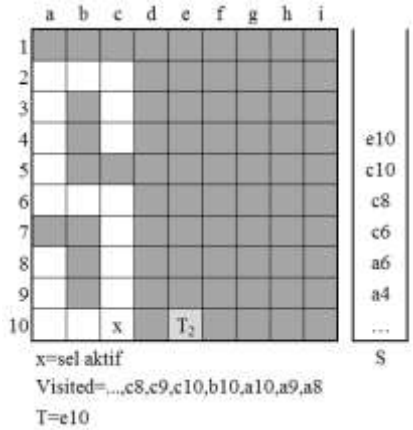


Gambar 3.5 Tandai T beserta sel diantara T dan X sebagai *visited*

Pada gambar 3.4 dan 3.5 tandai sel T berserta sel antara T dan X sebagai *visited*. Kemudian pindahkan posisi aktif ke sel posisi teratas pada S dan pilih salah satu sel tetangga T yang belum pernah dikunjungi.



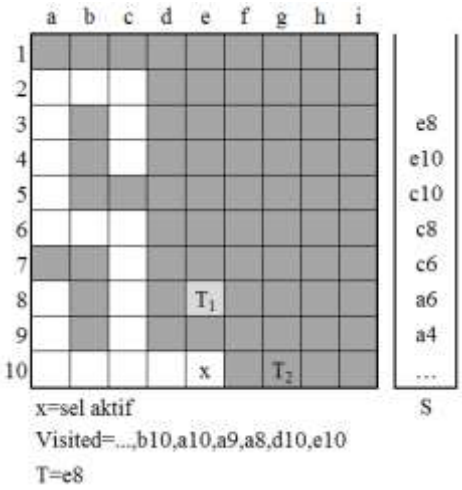
Gambar 3.6 Posisi Sel X tidak memiliki tetangga



Gambar 3.7 Pop() seluruh sel hingga sel yang memiliki tetangga yang belum dikunjungi

Pada gambar 3.6 setelah berulang-ulang dilakukan langkah sebelumnya sel X tidak memiliki tetangga, sehingga pada gambar 3.7

dilakukan pop pada S[5] hingga menjumpai sel dalam S yang masih memiliki tetangga, dan proses diulang kembali.

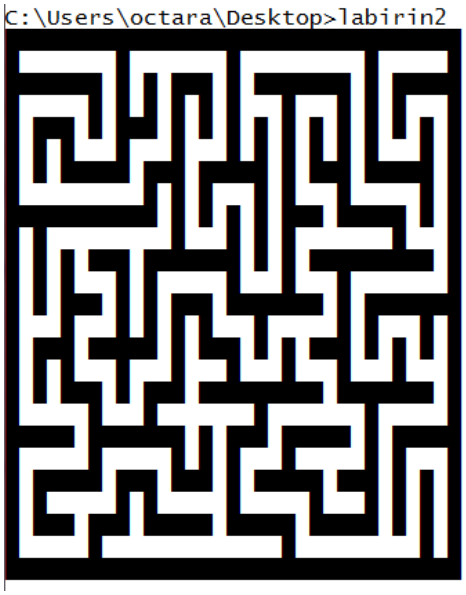


Gambar 3.8 Proses diulang dari awal

Pada gambar 3.8 setelah sel X dipilih, maka proses akan diulang lagi dari awal dan dilakukan terus menerus hingga seluruh sel pada S kosong.

4. Hasil

Berikut gambar 5.1 merupakan hasil yang didapatkan setelah seluruh proses dilakukan hingga akhir [6]:



Gambar 5.1 Labirin dengan ukuran 21 x 31

Hal yang sama tetap berlaku untuk labirin dengan ukuran yang berbeda-beda, hal ini dapat dilihat pada gambar 5.2

Gambar 5.2 Labirin dengan ukuran 17 x 23

5. Kesimpulan

Dari penelitian ini penulis penyimpulkan bahwa dengan algoritma DFS dapat membuat labirin *perfect* dengan sempurna.

Metode DFS ini masih memiliki kelemahan terutama dari sisi *time complexity*, dikarenakan setiap sel tetangga T yang akan ditandai harus mengecek seluruh sel yang telah ditandai *visited* sehingga membutuhkan lebih banyak langkah jika dibandingkan dengan algoritma seperti algoritma Prim dan algoritma Kruskal.

Daftar Pustaka

[1] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Maze_generation_algorithm. [Accessed 7 July 2015].

[2] M. Foltin, Automated Maze Generation and Human Interaction, Brno: Masaryk University Faculty Of Informatics, 2011.

[3] D. W. Pullen, "Think Labyrinth! :-)," [Online]. Available: http://www.astrolog.org/labyrnth.htm. [Accessed 1 July 2015].

[4] H. Inggiantowi, "Perbandingan Algoritma Penelusuran Depth First Search dan Breadth First Search pada Graf serta Aplikasinya," [Online]. Available: informatika.stei.itb.ac.id. [Accessed 2 July 2015].

[5] Sel yang di Pop adalah sel a8 dan a10 karena sel tersebut tidak memiliki tetangga yang belum pernah dikunjungi

[6] Implementasi program menggunakan bahasa C++