Penerapan Algoritma Pencarian DFS pada Algoritma Trémaux's

Algoritma Trémaux's pada Pencarian Solusi Permasalahan Labirin

Muchammad Ibnu Sidqi - 13518072
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13518072@std.stei.itb.ac.id

***Abstract*—Permasalahan labirin adalah permasalahan yang melibatkan suatu jalur yang dibuat secara paralel sehingga kumpulan jalur tersebut membentuk suatu wilayah yang rumit. Pada permasalahan labirin, seseorang yang terlibat diharuskan mencari jalan keluar agar tidak terjebak oleh rintangan yang terdapat pada labirin. Banyak cara serta algoritma untuk mencari solusi dari permasalahan labirin yaitu Algoritma Backtracking, DFS, A*, Tremaux, dan lainnya. Pada makalah ini akan dibahas mengenai salah satu dari algoritma tersebut yaitu algoritma Tremaux yang merupakan sebuah algoritma hasil penerapan algoritma DFS.**

Keywords—DFS, Tremaux, Labirin, Maze, A*, Backtracking;

I. PENDAHULUAN

Labirin adalah suatu wilayah yang didalamnya terdapat berbagai jalur paralel sehingga jalur tersebut membentuk lintasan yang rumit serta dipenuhi berbagai rintangan. Rintangan yang sering berada pada labirin adalah jalan buntu. Permasalahan labirin adalah suatu permasalahan yang mana bila seseorang terlibat pada permasalahan tersebut maka diharuskan mencari solusi berupa lintasan yang diperlukan agar seseorang tersebut dapat mencapai ujung labirin (suatu titik yang mana titik tersebut merepresentasikan sebagai penghubung langsung antara bagian dalam labirin dengan bagian luar labirin).

Pada permasalahan labirin, telah ditemukan banyak cara serta algoritma agar didapatkan solusi yang mangkus seperti algoritma pencarian DFS, A*, Tremaux, Backtracking, Cheese, dan lainnya. Algoritma pencarian DFS atau *Depth First Search* merupakan algoritma pencarian yang digunakan dengan membangkitkan pohon ruang status secara rekursif berdasarkan status yang menuju kepada solusi yang dicari dan memperhatikan kedalaman dari status yang sedang di iterasi(rekursif). Algoritma A* atau *A-Star* merupakan algoritma pencarian yang digunakan dengan membangkitkan pohon ruang status, seperti pada DFS, perbedaan terletak pada pengambilan status setiap iterasi atau rekursif dengan memperhatikan simpul status berdasarkan nilai heuristik sehingga setiap simpul status yang diambil merupakan simpul terbaik menuju simpul solusi. Algoritma *Backtracking* merupakan algoritma hasil penerapan dari DFS, perbedaan dengan algoritma DFS terletak pada fungsi pembatas yang digunakan untuk mematikan simpul yang tidak mengarah pada solusi. Sedangkan Algoritma *Cheese* merupakan algoritma pencarian yang digunakan dengan memperhatikan simpul solusi sebagai nilai heuristik yang mana nilai heuristik simpul yang terletak di dekat simpul solusi memiliki nilai heuristik tertinggi. Selanjutnya, Algoritma Tremaux merupakan algoritma pencarian hasil penerapan DFS yang mana setiap simpul ditandai dengan sebuah tanda apakah simpul tersebut sudah dilewati atau belum sehingga simpul status yang sudah dilewati atau dibangkitkan tidak akan dibangkitkan kembali. Pada makalah ini, penulis akan membahas dengan mendalam bagaimana penerapan algoritma DFS pada algoritma Tremaux.

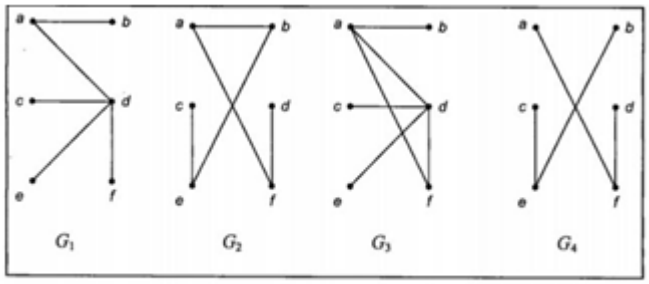
II. DASAR TEORI

A. Pohon

Pohon (*Tree)* adalah suatu graf yang tidak memiliki arah. Graf tersebut saling terhubung tetapi tidak memiliki sirkuit didalamnya. Contohnya terdapat graf yang tidak berarah sederhana G = (V,E) yang memiliki simpul berjumlah n, maka semua pernyataan dibawah ini terbukti:

1. G merupakan Pohon.
2. Setiap pasang simpul di G memiliki keterhubungan dengan lintasan tunggal.
3. G memiliki keterhubungan dan memiliki sejumlah n-1 busur.
4. G tidak membentuk sirkuit.
5. Penambahan satu busur pada pohon akan menyebabkan terbentuknya 1 sirkuit.
6. G saling terhubung dan setiap busur adalah jembatan (jembatan adalah busur jika dihapus akan menyebabkan pohon terpecah menjadi dua struktur).[1]

Diagram Pohon digunakan sebagai sebuah media untuk menginterpretasikan logika persoalan dengan mengambarkan semua alternative pemecahannya.[2]

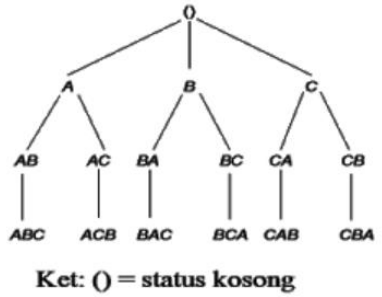


Gambar 9.1 G_1 dan G_2 adalah pohon, sedangkan G_3 dan G_4 bukan pohon

Gambar 1: Contoh Pohon. [1]

B. Ruang Status

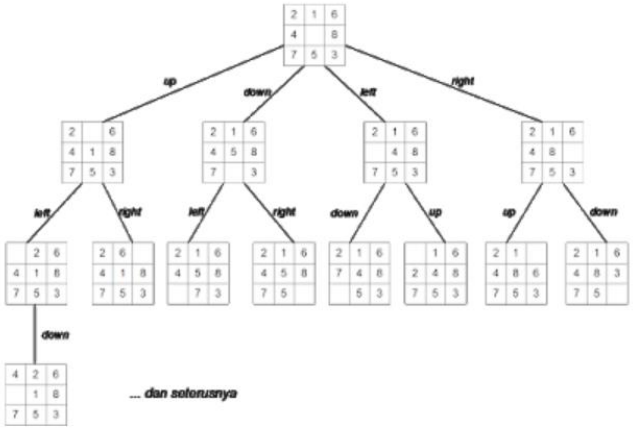
Ruang status adalah dimensi yang terdapat pada pohon (*Tree*) yang mana terdiri atas himpunan status yang melekat pada solusi dari persoalan yang dipetakan pada diagram pohon (*Tree*). Ruang status selalu digunakan dalam pemecahan terkait sehingga didapatka solusi yang diinginkan. Dalam hal ini, status yang terdapat pada ruang status harus sesuai dengan persoalan yang ada. Dalam menciptakan status yang terkait sering kali digunakan pendekatan heuristik agar status yang tercipta tetap berada pada kondisi persoalan. Selain itu, ruang status dapat diartikan sebagai keseluruhan simpul di dalam pohon dinamis dan pohonnya dinamakan juga pohon ruang status (state space tree).[3]



Gambar 2: Ruang Status dari pohon yang dibangkitkan dari permutasi ABC.[3]

Status yang terdapat pada pohon dapat dibagi menjadi 3 yaitu,

1. Status persoalan, merupakan status yang melekat pada persoalan terkait yang mana selalu melekat pada busur dari pohon terkait. Contohnya status persoalan pada persoalan *Puzzle Solver* yang mana status persoalan merupakan arah gerak dari ubin yang memenuhi ubin kosong yaitu atas, bawah, kanan, dan kiri.
2. Status tujuan, merupakan status yang melekat pada solusi terkait persoalan yang ada yang mana selalu melekat pada daun dari pohon ruang status.
3. Status solusi, merupakan gabungan dari himpunan solusi persoalan dengan status tujuan yang menginterpretasikan solusi dari permasalahan yang ada.



Gambar 3: Contoh Status yang Terdapat pada Permasalah *Puzzle*.

C. DFS

DFS (*Depth First Search)* merupakan algoritma pencarian yang digunakan dengan metode pencarian mendalam yang mana memiliki skema dengan mencari dari berbagai simpul secara iteratif.

DFD (*Depth First Search*) memiliki algoritma sebagai berikut:

1. Kunjungi simpul akar.
2. Kunjungi simpul w yang bertetangga dengan simpul akar.
3. Ulangi DFS mulai dari simpul w.
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.[4]

D. BFS

BFS (*Breadth First Search*) merupakan algoritma pencarian yang digunakan dengan metode pencarian menyebar yang mana memiliki skema dengan mencari dari berbagai simpul secara iterative.

BFS (*Breadth First Search*) memiliki algoritma sebagai berikut:

1. Kunjungi simpul akar.
2. Kunjungi semua simpul yang bertetangga dengan simpul akar terlebih dahulu.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.

*E. Algoritma A**

Algoritma A* adalah algoritma yang dikemukakan oleh Hart, Nilsson, dan Raphael pada tahun 1968.

Algoritma A* merupakan salah satu algoritma Branch and Bound atau disebut juga sebagai sebuah algoritma untuk melakukan pencarian solusi dengan menggunakan informasi tambahan (heuristik) dalam menghasilkan solusi yang optimal.[5]

Ada beberapa variasi A* seperti berikut :

1. Iterative Depeening A* (IDA*)
2. Simplified Memory-Bounded A* (SMA*)
3. Bi-Directional A* (BDA*)
4. Modified Bi-Directional A* (MBDA*)
5. Dynamic Weighting A* (DWA*)
6. Beam A* (BA*)

F. Algoritma Backtracking

Backtracking dapat dipandang sebagai salah satu dari dua hal berikut:

1. Sebagai sebuah fase di dalam algoritma traversal DFS.
2. Sebagai sebuah metode pemecahan masalah yang mangkus, terstruktur, dan sistematis.

Algoritma runut-balik pertama kali diperkenalkan oleh D. H. Lehmer pada tahun 1950. Algoritma runut-balik merupakan perbaikan dari exhaustive search. Pada exhaustive search, semua kemungkinan solusi dieksplorasi satu per satu. Pada backtracking, hanya pilihan yang mengarah ke solusi yang dieksplorasi, pilihan yang tidak mengarah ke solusi tidak dipertimbangkan lagi dengan memangkas (pruning) simpul-simpul yang tidak mengarah ke solusi.[4]

Langkah-langkah dari algoritma *Backtracking* adalah sebagai berikut:

1. Dari titik start pilih titik terdekat yang bisa dikunjungi atau tidak terhalang oleh tembok pada hal ini kita hanya bisa bergerak ke atas, belok kanan, belok kiri, dan turun kebawah.
2. Apabila titik yang kita kunjungi merupakan titik yang belum pernah dikunjungi sebelumnya, push titik tersebut kedalam stack solusi stack penanda untuk menandakan bahwa posisi tersebut sudah pernah dilewati, kemudian pilih lagi titik didekatnya.
3. Apabila titik yang kita kunjungi merupakan jalan buntu atau bertemu dengan tembok, pop sekali dari stack solusi untuk kembali ke posisi sebelumnya.
4. Ulangi langkah 2–3 sampai menemukan titik keluar dari labirin.[7]

```
while belum sampai pada tujuan do
  if terdapat arah yang benar sedemikian sehingga kita belum pernah
     berpindah ke sel pada arah tersebut
  then
     pindah satu langkah ke arah tersebut
  else
     backtrack langkah sampai terdapat arah seperti yang disebutkan
     di atas
  endif
endwhile
```

Gambar 4: Notasi Algoritmik algoritma *Backtracking*. [7]

G. Algoritma Cheese

Secara umum, cara kerja algoritma ini mirip dengan algoritma pencarian melebar (BFS) setelah didapatkan himpunan solusi menuju petak akhir. Pada algoritma BFS, jalur solusi kemudian di-generate dari himpunan solusi tadi (akhir-awal), sehingga didapatkan jalur solusi yang sebenarnya (awal-akhir). Pada cheese algorithm, kita tidak mencari himpunan solusi terlebih dahulu, disini kita berperan sebagai tikus yang mencari keju dalam labirin. Petak akhir atau yang kita anngap sebagai keju inilah yang memberitahukan keberadaan berdasarkan intensitas bau yang menyebar dalam labirin sesuai dengan jaraknya terhadap tempak keju (petak akhir) berada.

Intensitas bau bergantung pada jarak kita terhadap keju. Sehingga makin dekat kita terhadap keju (petak akhir) akan semakin besar nilai intensitasnya. Katakanlah nilai intensitas ini berada antara 0 dan 1, dimana 1 adalah nilai petak dimana keju itu berada. Misal intensitas petak berikutnya kita tetapkan bernilai setengah dari nilai kotak yang lebih dekat dengan keju. Maka kita akan mendapatkan nilai petak-petak tersebut semakin kecil ketika menjauhi petak tempat keju berada.

Persebaran Intensitas bau tersebut membentuk suatu pola yang mana akan mengarah kepada solusi yang diinginkan "*Cheese*" sehingga status permasalahan akan menuju pada intensitas bau tertinggi.

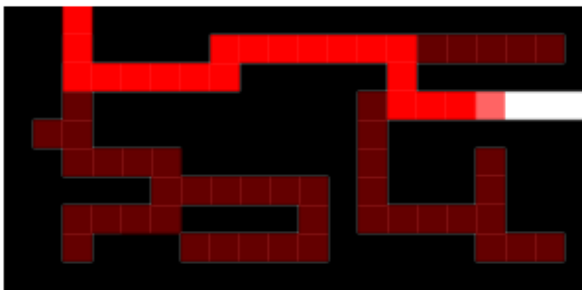
III. PEMBAHASAN

A. Algoritma Trémaux's

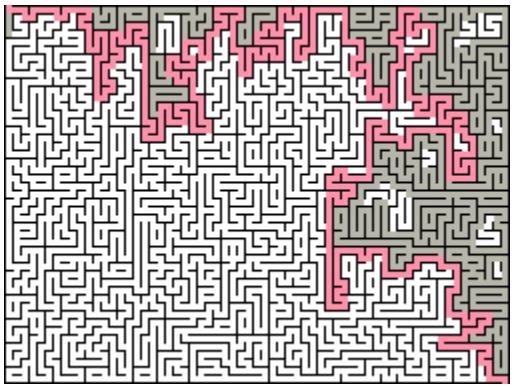
Pada permasalahan labirin, banyak algoritma yang dapat digunakan seperti algoritma A*, *backtracking*, *random mouse*, *wall follower*, *pledge*, *Dead-end filling*, hingga *maze-routing*. Beberapa algoritma memiliki kemiripan dengan algoritma pencarian DFS dan BFS bahkan beberapa algoritma memiliki dasar yang sama dengan DFS dan BFS. Hal tersebut dapat terjadi karena DFS dan BFS merupakan algoritma dasar dalam pencarian menggunakan graf sehingga abstraksi yang digunakan dapat ditemukan pada berbagai algoritma pencarian. Salah satu algoritma pencarian yang memiliki dasar yang sama dengan DFS adalah Algoritma *Trémaux's*. Algoritma *Trémaux's* adalah algoritma yang ditemukan oleh Charles Pierre Trémaux, algoritma tersebut digunakan dalam pemecahan persoalan labirin. Langkah-langkah pada algoritma *Trémaux's* adalah sebagai berikut:

1. Tentukan prioritas status/langkah yang digunakan pada labirin berdasarkan pendekatan heuristik terkait persoalan yang ada. Bila tidak digunakan pendekatan secara heuristik untuk mencari status persoalan maka gunakan *default* prioritas dari algoritma *Trémaux's* yaitu kanan, bawah/atas, kiri, langkah Kembali/*backtrack*.

2. Mulai lakukan penelusurn dengan menggunakan prioritas status yang sudah ditentukan sesuai dengan poin 1.
3. Pada setiap status yang dibuat maka titik yang sudah dilalui beri tanda bahwa tempat tersebut sudah dilalui sebanyak 1 kali.
4. Lakukan iterasi/rekursif hingga ditemukan rintangan atau jalan buntu. Bila telah ditemukan rintangan atau jalan buntu maka segera ambil langkah untuk kembali/*backtrack*.
5. Untuk setiap titik yang sudah dilewati lebih dari 1 kali maka blokir titik tersebut dan anggap bahwa titik tersebut sudah dilewati.
6. Lakukan iterasi/rekursif hingga mencapai status tujuan. Bila status tujuan sudah dicapai maka setiap titik yang sudah diberi tanda 1 kali adalah titik solusi dari permasalahan.



Gambar 5: Contoh Implementasi Algoritma *Trémaux's*. blok merah bata menandakan bahwa blok sudah dilewati 2 kali.



Gambar 6: Contoh Implementasi Algoritma *Trémaux's*. blok abu-abu menandakan bahwa blok sudah dilewati 2 kali.[6]

B. *Algoritma DFS vs Algoritma Trémaux's*

Pada algoritma DFS, langkah-langkah yang digunakan adalah melakukan rekursif pada simpul yang bertetangga hingga mencapai status tujuan atau mencapai simpul yang pernah dikunjungi. Bila mengunjungi simpul yang sudah dikunjungi maka langkah yang diambil adalah melakukan *backtrack* hingga ditemukan simpul tetangga yang belum pernah dikunjungi lalu lakukan rekursif kembali hingga ditemukan status tujuan dari permasalahan terkait. Pada algoritma *Trémaux's* langkah yang diambil adalah pengambilan status berdasarkan pendekatan heuristik yang sudah disepakati. Lalu pada setiap iterasi berikan tanda hingga ditemukan status tujuan atau ditemukan jalan buntu. Bila ditemukan jalan buntu maka lakukan *backtrack* dan beri tanda kembali pada setiap titik yang sudah dilewati. Bila tanda pada suatu titik sudah lebih dari 1 maka blokir jalan tersebut dengan anggapan bahwa jalan tersebut adalah buntu.

Perbedaan pada dua algoritma pencarian tersebut adalah **penggunaan pohon ruang status**. Pada algoritma DFS, setiap langkah terkait pencarian solusi diambil berdasarkan pohon ruang status dengan membuat simpul lalu mengunjungi simpul tersebut secara rekursif sedangkan pada algoritma *Trémaux's*, langkah yang diambil belum tentu berdasarkan pohon ruang status, langkah yang diambil dapat dengan melakukan iterasi sekuensial pada labirin terkait.

Perbedaan selanjutnya pada dua algoritma tersebut adalah **pengambilan status solusi** bila status tujuan telah tercapai. Pada algoritma DFS, setiap langkah yang digunakan dilakukan dengan cara rekursif sehingga pengambilan status solusi menjadi lebih mudah. Pada algoritma *Trémaux's* pengambilan status solusi bila menggunakan iterasi secara sekuensial menjadi lebih sulit sehingga pengambilan status solusi harus dengan melakukan pengecekan seluruh titik dari labirin lalu memilih titik yang baru dikunjungi 1 kali.

Perbedaan selanjutnya dari dua algoritma tersebut adalah **alokasi memori** yang dibutuhkan. Pada algoritma DFS, simpul yang dibuat hanya berdasarkan status yang dipilih sehingga tidak menjamin bahwa seluruh titik akan dibuat sedangkan pada algoritma *Trémaux's*, bila menggunakan iterasi secara sekuensial maka seluruh titik dari labirin harus dibuat terlebih dahulu sehingga alokasi memori pada algoritma *Trémaux's* dapat lebih besar bila dibandingkan dengan algoritma DFS.

Persamaan dari algoritma *Trémaux's* dan DFS yaitu solusi yang didapatkan pada dua algoritma tersebut **tidak dijamin solusi yang optimal**. Solusi optimal pada permasalahan labirin adalah solusi dengan jalur terpendek dari semua solusi yang ada.

C. *Implementasi Algoritma DFS pada Algoritma Trémaux's*

Dari berbagai perbedaan yang telah ditemukan dari algoritma DFS dan pada algoritma *Trémaux's* adalah penggunaan diagram pohon pada implementasi algoritma sehingga dengan menggunakan pohon ruang status pada implementasi algoritma *Trémaux's* maka antara dua algoritma tersebut dapat dikatakan bahwa memiliki dasar yang sama sehingga dapat diambil sebuah pernyataan bahwa implementasi algoritma DFS pada algoritma *Trémaux's* adalah dengan mengimplementasikan pohon ruang status.

D. Algoritma Backtracking vs Algoritma Trémaux's

Pada algoritma *Backtracking* pemecahan permasalahan labirin, langkah-langkah yang digunakan seperti pada DFS yaitu, secara rekursif pada simpul yang bertetangga hingga mencapai status tujuan atau mencapai simpul yang pernah dikunjungi. Bila mengunjungi simpul yang sudah dikunjungi maka langkah yang diambil adalah melakukan *backtrack* hingga ditemukan simpul tetangga yang belum pernah dikunjungi lalu lakukan rekursif kembali hingga ditemukan status tujuan dari permasalahan terkait sehingga tidak ada perbedaan besar dengan algoritma *Trémaux's*.

Perbedaan antara algoritma *Backtracking* dengan algoritma *Trémaux's* adalah pada **fungsi pembatas** yang diimplementasikan pada algoritma *backtracking*. Pada algoritma *Backtracking* terdapat fungsi pembatas yang berfungsi untuk membunuh simpul yang tidak mengarah pada status tujuan sehingga dapat mengurangi waktu dalam mencari solusi. Fungsi pembatas yang diimplementasikan harus berdasarkan pendekatan heuristik dari permasalahan terkait sehingga simpul yang dibunuh bukan merupakan simpul yang mengarah pada status tujuan. Untuk permasalahan labirin, penentuan fungsi pembatas cukup rumit karena labirin yang diimplementasikan tidak memuat informasi spesifik mengenai karakteristik dari struktur pada labirin sehingga hampir seluruh labirin memiliki struktur yang sama. Bila fungsi pembatas tidak diimplementasikan maka dapat dikatakan bahwa antara algoritma *Backtracking* dengan algoritma *Trémaux's* adalah sama.

E. Algoritma A vs Algoritma Trémaux's*

Algoritma A* memiliki kemiripan dengan algoritma BFS yang mana pencarian solusi dilakukan dengan iterasi/rekursif pada simpul-simpul yang bertetangga dengan simpul yang sedang diiterasi. Perbedaan dari algoritma A* dengan algoritma BFS adalah penggunaan pendekatan heuristik berupa *cost* untuk mencapai status tujuan untuk menentukan simpul mana yang akan diiterasi sehingga pengambilan simpul tidak selalu berurutan dan memiliki kemungkinan akan mengambil simpul dengan level yang lebih rendah dari simpul yang sedang diiterasi(*backtrack*). *Cost* dari Algoritma A* adalah penjumlahan fungsi $g(n)$ dan $h(n)$ yang mana n menandakan simpul ke-n, g merupakan fungsi heuristik yang menandakan usaha untuk mencapai simpul saat ini dan h merupakan fungsi heuristik yang menandakan usaha untuk mencapai status tujuan.

Berikut beberapa kondisi dari algoritma A*:

1. Jika $h(n)$ bernilai 0, maka $f(n) = g(n)$. Algoritma A* akan berjalan seperti algoritma Dijkstra.
2. Jika $h(n)$ bernilai sangat besar dibandingkan dengan nilai $g(n)$, maka $f(n) = h(n)$. Algoritma A* akan berjalan seperti algoritma Greedy Best-First Search.
3. Jika $h(n)$ bernilai kurang dari atau sama dengan cost dari titik n ke titik tujuan, maka algoritma A* dijamin dapat menemukan solusi optimal. Semakin kecil nilai $h(n)$, pencarian akan semakin lama.
4. Jika $h(n)$ bernilai lebih besar dibandingkan cost dari titik n ke titik tujuan, maka algoritma A* tidak dijamin dapat menemukan solusi optimal. Akan tetapi, pencarian akan berjalan lebih cepat. [8]

Perbedaan antara algoritma A* dengan algoritma *Trémaux's* terletak pada **penggunaan pendekatan heuristik** *cost* dalam penentuan simpul yang diiterasi.

Dasar yang diimplementasikan pada kedua algoritma tersebut juga berbeda, pada algoritma A* dasar yang digunakan adalah algoritma pencarian BFS sedangkan pada algoritma *Trémaux's* dasar yang digunakan adalah DFS. Apakah mungkin algoritma *Trémaux's* mengimplementasikan BFS? Jawabannya adalah tidak mungkin karena iterasi yang digunakan pada algoritma *Trémaux's* adalah iterasi dengan memperhatikan status sebelumnya sehingga status saat ini adalah hasil dari status sebelumnya.

Perbedaan selanjutnya adalah bila $h(n)$ yang diperoleh pada algoritma A* melalui pendekatan heuristik lebih kecil atau sama dengan nilai $h(n)$ tanpa pendekatan heuristik maka solusi yang didapatkan **dijamin optimal**.

Persamaan dari dua algoritma tersebut adalah bila nilai *cost* yang didapatkan pada algoritma A* bernilai $g(n)$ maka algoritma A* tidak dijamin memiliki **solusi optimal**.

F. Algoritma Cheese vs Algoritma Trémaux's

Algoritma Cheese memiliki kemiripan dengan algoritma A* yaitu pemilihan simpul yang akan diiterasi berdasarkan pendekatan heuristik. Perbedaan dari algoritma A* dengan algoritma *Cheese* adalah pada pendekatan heuristik yang digunakan. Bila algoritma A* menggunakan pendekatan heuristik dengan penjumlahan fungsi $g(n)$ dan $h(n)$ yang mana n menandakan simpul ke-n, g merupakan fungsi heuristik yang menandakan usaha untuk mencapai simpul saat ini dan h merupakan fungsi heuristik yang menandakan usaha untuk mencapai status tujuan maka algoritma *Cheese* menggunakan pendekatan heuristik berdasarkan intensitas kedekatan antara status saat ini dengan status tujuan. Algoritma *Cheese* dapat dikatakan berjalan seperti algoritma *Greedy Best-First Search*.

Perbedaan dari algoritma *Cheese* dengan algoritma *Trémaux's* adalah pada penggunaan **pendekatan heuristik** yang digunakan dalam memilih simpul yang akan diiterasi yang mana pada algoritma *Cheese* pemilihan simpul berdasarkan pendekatan heuristik dari intensitas kedekatan antara status saat ini dengan status tujuan.

Perbedaan selanjutnya dari algoritma *Cheese* dengan algoritma *Tremaux's* adalah **dasar yang digunakan** pada kedua algoritma tersebut. Pada algoritma *Cheese*, dasar yang digunakan adalah BFS sedangkan pada algoritma *Trémaux's*, dasar yang digunakan adalah DFS.

Persamaan dari algoritma *Cheese* dengan algoritma *Trémaux's* adalah kedua algoritma tersebut **tidak menjamin mendapatkan solusi optimal**.

VIDEO LINK AT YOUTUBE

https://youtu.be/oDNYjyxbc1M

UCAPAN TERIMAKASIH

Atas berkat rahmat Tuhan Yang Maha Esa pengerjaan karya tulis *"Penerapan Algoritma Pencarian DFS pada Algoritma Trémaux's"* dapat berlangsung dengan baik. Untuk itu, penulis juga mengucapkan terimakasih yang tidak terhingga kepada Dr. Ir. Rinaldi Munir, MT. selaku dosen mata kuliah strategi algoritma dan semua kerabat yang telah memberi saran kritik atas keberjalanan pembuatan karya tulis ini.

REFERENCES

[1] Munir, Rinaldi. 2010. Matematika Diskrit, edisi 3 revisi keempat. Bandung: Informatika Bandung.

[2] Wibisono, Samuel, 2008. Matematika Diskrit, edisi 2, Yogyakarta: Graha Ilmu.

[3] Anonim, https://www.ilmuskripsi.com/2016/05/penerapan-bfs-dan-dfs-pada-pencarian.html diakses pada 1 Mei 2020 pukul 21.05 WIB.

[4] M. Rinaldi, http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/stima19-20.htm diakses pada 1 Mei 2020 pukul 22.05 WIB.

[5] R. H. Fitri, https://fitrihandayani.blog.upi.edu/variasi-algoritma-a/ diakses pada 1 Mei 2020 pukul 22.05 WIB.

[6] http://www.astrolog.org/labyrnth/algrithm.htm diakses pada 2 Mei 2020 pukul 00.35 WIB

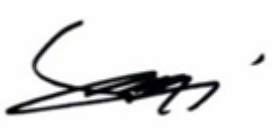
[7] F. Irsyad, https://irsyadf.my.id/maze-solver-menggunakan-algoritma-backtracking-47249d1542af diakses pada 2 Mei 2020 pukul 00.45 WIB

[8] http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html. Diakses pada 2 Mei 2020 pukul 22.10 WIB

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, 04 Mei 2020



Muchammad Ibnu
13518072