Aplikasi DFS dan BFS dalam Algoritma Mobil Otomatis

Ayrton Cyril 13516019[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1] *13516019@std.stei.itb.ac.id*

***Abstrak*—Seiring dengan perkembangan teknologi saat ini, semakin banyak teknologi termuka yang ditemukan. Teknologi yang sedang berkembang saat ini adalah mobil otomatis. Mobil otomatis adalah mobil yang tidak memerlukan pengemudi untuk mengerakan mobil. Mobil otomatis yang ditawarkan pada masyarakat ini diharapkan dapat memberikan keselamatan pada seluruh penumpangnya dan juga kenyaman selamat berkendara. Maka dari itu dibuthkan *programmer* dalam menentukan proses pergerakan mobil otomatis. Pergerakan mobil sangat dipengaruhi dari cara program menentukan langkah yang paling optimal dalam menjamin keselamatan penumpang. Proses pencarian solusi yang paling optimal diselesaikan dengan menggunakan algoritma BFS dan DFS.**

***Kata kunci*— BFS,DFS,IDS, mobil otomatis**

I. PENDAHULUAN

Kendaraan saat ini sudah merupakan kebutuhan primer dijaman ini. Kendaraan sudah berkembang dari masa kemasa, dari mobil pertama di dunia hingga mobil saat ini yang sudah sangat canggih. Perkembangan mobil tidak berhenti disitu, pada zaman ini mobil seperti mobil otomatis, mobil listrik dan lainnya sedang marak dikembangkan. Mobil otomatis yang dimaksud disini bukanlah mobil dengan tramisi otomatis, melaikan mobil yang dapat bergerak sendiri tanpa harus dikendarai. Pembuatan mobil otomatis ini diharapkan dapat membantu manusia dalam berkendara dan memberikan keselamatan selama berkendara dan ketertiban dalam berkendara.

Mobil otomatis diharapkan dapat mengurangi resiko berkendara yang diakibat dari *human error*. Kecelakaan yang disebabkan seperti mengantuk saat mengemudi, tidak fokus saat mengemudi dan lainnya dapat diatas dengan adanya mobil otomatis. Mobil otomatis tidak menuntut manusia untuk memperhatikan kondisi sekitar dalam mengemudi sehingga manusia sebagai penumpang dapat lebih santai dengan tidak mengemudi. Penyebab kecelakaan lainnya adalah manusia yang tidak mau mematahui peraturan yang berlaku. Peraturan yang dilanggar dapat menjadi penyebab terjadinya kecelakaan pada saat berkendara. Pelanggaran seperti menerobos lampu merah, maupun menyetir di lajur yang berlawanan. Mobil otomatis yang dibuat diharuskan berekendara sesuai dengan aturan yang berlaku, dan saat mobil otomatis berkendara manusia tidak dapat memaksakan mobil itu untuk melakukan pelanggaran lalu lintas, maka dari itu mobil otomatis diharapkan dapat meningkatkan tingkat keselamatan selama berkendara

Mobil otimatis dituntut untuk memberikan keselamatan lebih kepada penumpangnya, maka dari itu pembuat mobil otmatis harus memperhatikan segala aspek dalam menyusun algoritma mobil otomatis. *Programmer* diharuskan menyusun suatu algoritma yang tepat sehingga mobil dapat bergerak dan sampai ditujuan dalam keadaan selamat. Dalam berkendara apabila mobil dihadapkan pada kondisi ingin bertabrakan mobil diharuskan mengetahui langkah yang paling aman bagi berekndara. Mobil harus melihat jika mobil berbelok ke kiri, maka mobil harus bisa terus berkendara dengan normal, mobil tidak diharapkan akan menabrak mobil lain setelah berbelok ke kiri. Penentuan keputusan diharuskan melihat beberapa langkah kedepan agar mobil dijamin selalu selamat. Mobil juga diharapkan dapat membandingkan suatu kondisi, seperti membandingkan kondisi apabila ia berbelok kiri maupun berbelok kanan. Mobil diharapkan dapat menemukan solusi yang paling tepat diantara beberapa pilhan yang ada.

Dalam menentukan keputusan langkah yang paling tepat dalam berkendara kita dapat menggunakan algoritma DFS dan BFS. Kedua algoritma ini digunakan sebagai metoda dalam pencarian solusi yang paling tepat.

II. LANDASAN TEORI

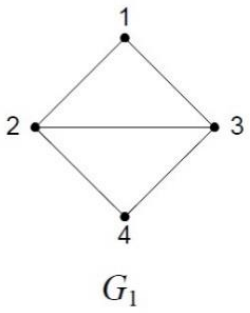
2.1 Definisi Graf

Graf adalah pasangan himpunan (V,E) dengan V adalah himpunan tidak-kosong dari simpul-simpul(*vertices* atau *node*), $V=\{v_1,v_2,v_3,..,v_n\}$ dan E adalah himpuan sisi(*edges* atau *arcs*) yang menghubungkan sepasang simpul = $\{e_1,e_2,e_3,\ldots,e_n\}$. Dari definisi diatas dapat disimpulkan bahwa suatu graf mungkin tidak mempunyai sisi tetapi pasti mempunya satu buah simpul.

Graf dapat dikelompokan menjadi beberapa kategori(jenis) bergantung pada pengelompokannya. Pengelompokan graf dapat dipandak berdasarkan ada tidaknya sisi ganda atau sisi kalang, berdasarkan jumlah simpul, atau berdasarkan orientasi arah pada sisi. Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, maka secara umum graf dapat digolongkan menjadi dua jenis:

2.1.1 Graf sederhana(*simple graph*)

Graf sederhana adalah graf yang tidak mengadung gelang maupun sisi-ganda.

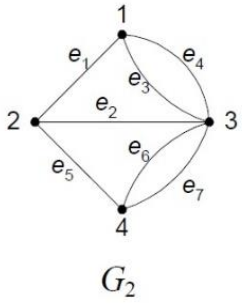


Gambar 2.1 Graf sederhana
Sumber : Diktat Rinaldi Munir

2.1.2 Graf tak-sederhana(un*simple-graph*)

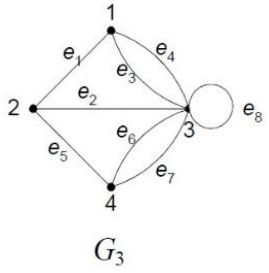
Graf tak-sederhana adalah graf yang mengadung gelang maupun sisi-ganda. Ada dua macam graf tak-sederhana yaitu graf ganda(*multigraph*) dan graf semu (*pseudograph*).

Graf ganda adalah graf yang mengandung sisi ganda. Sisi ganda yang menghubungkan sepasang simpul bisa lebih dari dua buah.



Gambar 2.2 Graf ganda
Sumber : Diktat Rinaldi Munir

Graf semu adalah graf yang mengandung gelang(termasuk apabila memiliki sisi ganda sekalipun). Graf semu lebih umum daripada graf ganda, karena sisi pada graf semu dapat terhubung ke dirinya sendiri.



Gambar 2.3 Graf semu
Sumber : Diktat Rinaldi Munir

Berdasarkan jumlah simpul pada suatu graf, maka secara umum graf dapat digolongkan menjadi dua jenis:

2.1.3 Graf berhingga(*limited graph*)

Graf berhingga adalah graf yang jumlah simpulnya, n, berhingga.

2.1.4 Graf tak-berhingga(un*limited graph*)

Graf tak-berhingga adalah graf yang jumlah simpulnya, n, tidak berhingga banyaknya.

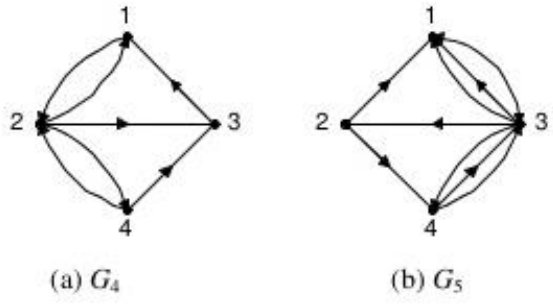
Sisi pada graf dapat mempunyai orientasi arah. Berdasarkan orientasi arah pada sisi, maka secara umum graf dibedakan atas 2 jenis:

2.1.3 Graf tak-berarah(*undirected graph*)

Graf yang sisinya tidak mempunya orientasi arah dsebut graf tak-berarah. Pada graf tak-berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan. Jadi, $(v_j, v_k) = (v_k, v_j)$ adalah sisi yang sama. Contoh graf tak-berarah dapat dilihat pada gambar 2.1, 2.2 dan gambar 2.3

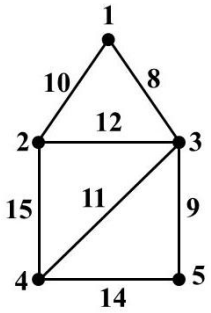
2.1.4 Graf berarah(*directed graph*)

Graf yang setiap sisinya diberikan orientasi arah disebut sebagai graf berarah. Kit alebih suka menyebut sisi berarah dengan sebutan busur(*arc*). Pada graf berarah, (v_j, v_k) dan (v_k, v_j) menyatakan dua buah busur yang berbeda, dengan kata lain $(v_j, v_k) \neq (v_k, v_j)$. Untuk busur (v_j, v_k) simpul v_j dinamakan simpul asal(*intial vertex*) dan simpul v_k dinamakan simpul terminal(*terminal vertex*).



Gambar 2.4 (a)Graf berarah, (b) graf ganda berarah
Sumber : Diktat Rinaldi Munir

Pada graf juga kita dapat memberikan bobot pada setiap sisinya sehingga graf diberi nama graf berbobot(*weighted graph*)



Gambar 2.5 Graf berbobot

Sumber : Diktat Rinaldi Munir

2.2 Definisi DFS dan BFS

Dalam proses pencarian solusi, terdapat dua pendekatan yaitu graf statis dan graf dinamis. Graf statis adalah graf yang sudah terbentuk sebelum proses pencarian dilakukan, sedangkan graf dinamis adalah graf yang terbentuk saat proses pencarian dilakukan.

2.2.1 Pencarian melebar(BFS)

Algoritma pencarian melebar atau *Breadth-First Search*(BFS) adalah algoritma traversal untuk melakukan pencarian di dalam graf tanpa informasi tambahan (*uninformerd search*) dengan cara memilih simpul-simpul yang bertetangga dengan simpul asal dan menelusuri semua simpul tersebut sesuai dengan aturan urutan pembangkitan simpul hingga semua simpul telah dibangkitkan atau solusi telah ditemukan

Kelebihan dari penggunaan algoritma BFS adalah algotima ini pasti selalu menemukan solusi yang optimal dengan berbagai macam bentuk graf atau pohon di dalam suatu permasalahan dan pasti menemukan solusi optimal.

Kelemahan dari penggunaan algoritma BFS adalah kompleksitas memori yang dibutkan sangat besar karena BFS menyimpan semua simpul yang berada di dalam gra

```
procedure BFS(input v:integer)
{ Traversal graf dengan algoritma pencarian BFS.

  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang dikunjungi dicetak ke layar
}
Deklarasi
   w : integer
   q : antrian;

   procedure BuatAntrian(input/output q : antrian)
   { membuat antrian kosong, kepala(q) diisi 0 }

   procedure MasukAntrian(input/output q:antrian, input v:integer)
   { memasukkan v ke dalam antrian q pada posisi belakang }

   procedure HapusAntrian(input/output q:antrian,output v:integer)
   { menghapus v dari kepala antrian q }

   function AntrianKosong(input q:antrian) → boolean
   { true jika antrian q kosong, false jika sebaliknya }

Algoritma:
   BuatAntrian(q)          { buat antrian kosong }

   write(v)                { cetak simpul awal yang dikunjungi }
   dikunjungi[v]←true      { simpul v telah dikunjungi, tandai dengan
                             true}
   MasukAntrian(q,v)       { masukkan simpul awal kunjungan ke dalam
                             antrian}

  { kunjungi semua simpul graf selama antrian belum kosong }
  while not AntrianKosong(q) do
        HapusAntrian(q,v)   { simpul v telah dikunjungi, hapus dari
                              antrian }
        for tiap simpul w yang bertetangga dengan simpul v do
            if not dikunjungi[w] then
               write(w)     {cetak simpul yang dikunjungi}
               MasukAntrian(q,w)
               dikunjungi[w]←true
            endif
     endfor
  endwhile
  { AntrianKosong(q) }
```


Gambar 2.6 Algoritma BFS
Sumber : Diktat Rinaldi Munir

2.2.2 Pencarian mendalam(DFS)

Algoritma pencarian mendalam atau *Depth-First Search*(DFS) adalah algoritma traversal untuk melakukan pencarian didalam suatu graf tanpa informasi tambahan(*uninformed search*) dengan cara melakukan pencarian dengan memilih salah satu simpul yang ada dan menelusuri anak dari simpul tersebut hingga salah satu solusi ditemukan atau telah mencapai simpul daun, sehingga pencarian akan dilajutkan ke simpul terakhirbyang dikunjungi sebelumnya dan simpul tersebut mempunya satu atau lebih simpul yang belum dijunjungi.

Kelebihan dari penggunaan algoritma DFS adalah pemakaian memori yang lebih sedikit bila dibandingkan dengan algoritma BFS. Untuk persoalan yang memiliki banyak solusi, metode DFS lebih cepat daripada BFS, karena DFS menemukan solusi setelah mengeksplorasi hanya sebagian kecil dari seluruh ruang status.

Kelemahan dari penggunaan algoritma DFS adalah algoritma ini tidak mempunyai jaminan untuk menemukan solusi apabila graf mempunyai level yang sangat dalam sehingga DFS tidak melakukan pada pencari pada simpul lain.

```
procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS

Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpulyang dikunjungi ditulis ke layar
}
Deklarasi
   w : integer

Algoritma:
   write(v)
   dikunjungi[v]←true
   for w←1 to n do
      if A[v,w]=1 then {simpul v dan simpul w bertetangga }
          if not dikunjungi[w] then
             DFS(w)
          endif
      endif
   endfor
```

Gambar 2.7 Algoritma DFS
Sumber : Diktat Rinaldi Munir

2.2.3 *Depth-Limited Search* (DLS)

Algoritma ini adalah hasil dari optimum proses DFS dimana dalam pencarian, kedalaman dibatasi. Misalkan pada pencarian dibatasi sampai kedalaman l. Maka simpul pada level l dianggap tidak memiliki *successor*. Pencarian dengan cara ini memiliki kelebihan yaitu mempercepat proses pencarian DFS. Algoritma DLS juga memilki kelemahan yaitu sulitnya menentukan batas level karena ketemu atau tidaknya solusi sangat ditentukan oleh penentuan level.

```
Function DLS (problem, limit)
→ rec_DLS(make_node(init_state),problem,limit)

Function Rec_DLS (node,problem, limit)
  if isGoal(node) then → solution(node)
  else if depth(node)=limit then → cutoff
  else
      for each successor in Expand(node,problem) do
        result ←rec_DLS(successor,problem,limit)
        if result=cutoff then cutoff_occured← true
        else if result≠failure then → result
  if cutoff_occured then → cutoff
  else → failure
```

Gambar 2.8 Algoritma DLS
Sumber : Diktat Rinaldi Munir

2.2.4 *Iterative Deepening Search*(IDS)

Algoritma IDS melakukan serangkaian DFS, dengan peningkatan nilai kedalaman cutoff, sampai solusi ditemukan. Algoritma ini memiliki asumsi yaitu simpul sebagian besar ada di lvel bawah, sehingga tidak menjadi persoalan ketika simpul pada level-level atas dibangkitkan berulang kali. Algoritma ini memiliki keuntungan dari memori dan menjamin kita

menemukan solusi, hanya saja terjadi pembangkitan berulang kali. Sehingga kompleksitas waktu menjadi sangat tinggi bila dibandingan dengan algoritma lainnya.

```
Depth ← 0
Iterate
  result← DLS(problem,depth)
stop: result ≠ cutoff
  depth← depth+1
→ result
```

Gambar 2.9 Algoritma IDS
Sumber : Diktat Rinaldi Munir

III. APLIKASI DFS DAN BFS DALAM ALGORTIMA MOBIL OTOMATIS

Setelah melihat proses pencarian solusi menggunakan algoritma DFS dan BFS kita dapat melihat banyak manfaat dalam menggunakan dua algoritma tersebut disertai dengan mengetahui kelebihan dan kelemahan dari masing-masing algoritma tersebut. Penentuan suatu keputusan dalam penerapan mobil otomatis memerlukan pengaplikasian DFS maupun BFS. Maka dari itu seiring dengan majunya perkembangan mobil otomatis di dunia saat ini *programmer* membutuhkan algoritma DFS dan BFS dalam memproses keputusan dari mobil otmatis yang dikembangkan oleh mereka.

Dalam penulisan algoritma mobil otomatis kita mengharuskan program menentukan sebuah algoritma yang memiliki waktu cepat dalam menentukan solusi dan juga memastikan kita mendapatkan suatu solusi. Algoritma BFS memastikan kita mendapatkan solusi tetapi memiliki kelemahan yaitu memakan banyak memori dan memiliki kompleksitas waktu yang cukup tinggi. Algoritma DFS sebenranya tidak menjamin kita menemukan solusi maka dari itu penggunaan algoritma ini harus dihindari. Pencarian yang tidak dapat menemukan solusi akan berakibat program menjadi *error* dan mobil akan mencelakanan penumpangnya. Maka dari itu apabila ingin menggunakan DFS, kita harus menggunakan algortima DFS yang sudah dioptimasi seperti IDS agar kita pasti mendapatkan solusi. Algoritma IDS adalah algoritma yang mengambungkan kelebihan BFS dan DFS tetapi memiliki kelemahan pada komplesitas waktunya. BFS maupun IDS memiliki kelebihan apabila setiap langkah adalah biaya yang harus dibayarkan. Dalam proses pembuatan aplikasi mobil otomatis setiap langkah yang dikerjakan sangatlah berarti bagi mobil otomatis tersebut. Dalam pembuatan algoritma mobil otomatis *programmer* dituntut untuk memberikan kenyaman dan menjamin keselamatan para penumpangnya maka dari itu apabila algoritma yang kita buat memiliki response time yang lama ataupun solusi yang diberikan membutuhkan banyak langkah untuk mencapainya maka algoritma kita tidak dapat diggunakan. Dalam proses pencarian ini solusi yang dicari adalah menjamin mobil dapat selamat sampai ketempat tujuan.

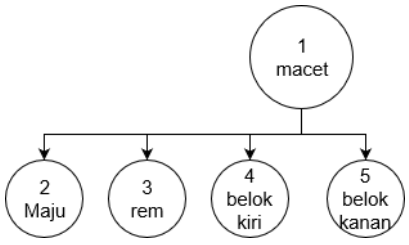
Algoritma IDS lebih baik digunakan pada saat mobil bergerak pada kondisi normal. Pada saat kondisi ini mobil tidak terlalu dituntut untuk merespon segala sewaktu dalam waktu yang cepat, tetapi lebih menuntut untuk mengurangi memori yang diggunakan agar program tidak menjadi berat apabila harus melakukan pencarian menggunakan BFS. Untuk mengoptimalkan IDS sebaiknya kita tidak memulai proses iterasi dari angka satu dikarenakan algoritma mobil otomatis diharapkan tidak menemukan solusi yang aman hanya untuk level satu saja, tetapi kita harus memastikan bahwa untuk proses kedepannya tetap ditemukan solusi sehingga keselamatan mobil tetap terjaga, maka dari proses pencari secara iterative sebaiknya dimulai dari berapa banyak langkah kedepan yang ingin dicari.

Algoritma BFS lebih baik digunakan pada saat mobil dihadapkan pada suatu kondisi yang mendadak sehingga mobil harus bergerak cepat. Penggunaan algoritma ini lebih baik daripada penggunaan DFS maupun IDS walaupun BFS juga memakan waktu yang cukup lama. Algoritma ini dpilih karena kita harus melakukan pencarian melebar agar reaksi mobil lebih cepat dalam menentukan solusi. Pencarian ini dapat membandikan berbagai solusi yang sudah muncul dan dapat menemukan solusi yang paling optimal dikarenakan pencari BFS memungkinkan kita menemukan banyak solusi.

IV. STUDI KASUS APLIKASI DFS DAN BFS PADA ALGORITMA POHON KEPUTUSAN

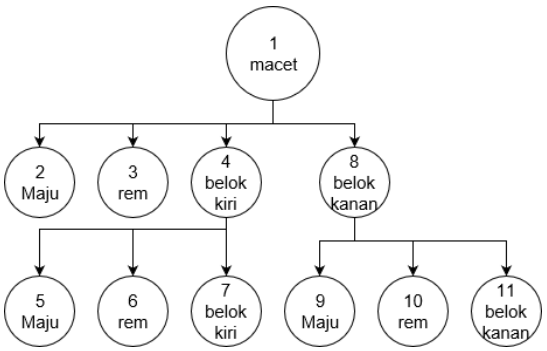
Mobil otomatis dalam penerapannya selalu dihadapkan pada berbagai langkah yang mungkin untuk dilakukan mulai dari belok kiri, belok kanan, berhenti, ganti perseneling sampai menambah kecepatan. Pergerakan mobil otomatis ini selalu didasari oleh keadaan sekitar mobil. Setiap kali ditemukan suatu kondisi baru seperti lampu merah, orang menyebrang, mobil berhenti mendadak dan ainnya diperlukan keputusan baru untuk mengerakan mobil otomatis tersebut.

Pada kondisi macet mobil diharapkan dapat menghindar dari kondisi ini. Dalam pencari solusi ini digunakan algoritma pencari IDS dikarena algoritma ini hemat dari segi memori walaupun membuang banyak waktu, tetapi pada kondisi ini mobil tidak dihadapkan pada kondisi yang membutuhkan waktu cepat.



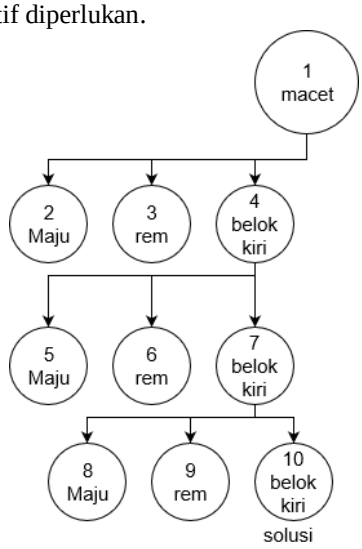
Gambar 4.1 IDS iteratif ke satu
Sumber : Koleksi pribadi penulis

Pada iterative pertama mobil dihadapkan pada pilihan untuk maju, rem dan berbelok, tetapi walaupun mobil sudah berbelok kiri maupun kanan mobil masih terjebak oleh macet maka dari itu algoritma dilanjutkan pada iteratif berikutnya



Gambar 4.2 IDS iteratif ke dua
Sumber : Koleksi pribadi penulis

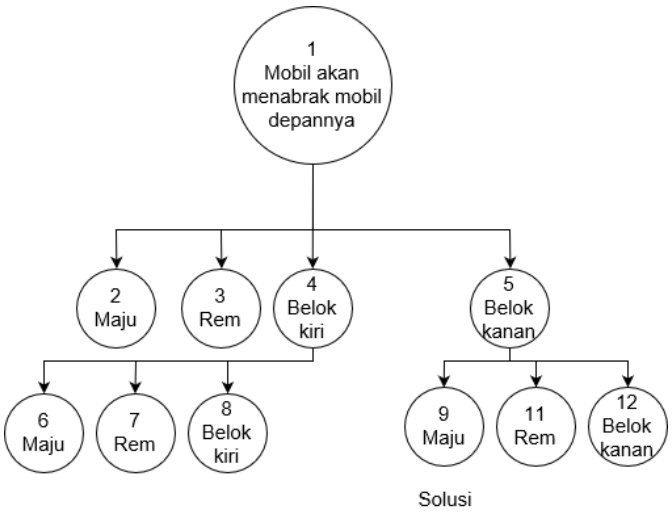
Pada iteratif kedua kita melakukan pencarian hingga level kedua. Urutan pencarian dapat dilihat dengan mengamati nomor pada simpul. Pada kondisi ini pun solusi belum ditemukan dikarenakan mobil masih terjebak kemacetan maka penambah iteratif diperlukan.



Gambar 4.3 IDS iteratif ke tiga
Sumber : Koleksi pribadi penulis

Pada iteratif ketiga ditemukan solusi pada simpul nomor 10. Urutan pencarian dilakukan sesuai dengan nomor simpul. Simpul – simpul yang tidak diekspansi dikarenakan simpul tidak menghasilkan solusi lagi. Pada pengunaan algoritma IDS ini akan memakan waktu lebih lama tetapi proses ini tidak memakan banyak memori dan menjami mobil otomatis menemukan solusi yang tepat.

Algoritma yang digunakan pada kondisi saat mobil dihadapkan pada kondisi yang akan bertabrakan adalah algoritma BFS. Mobil diharapkan dapat menghindari tabrakan dengan mobil di depannya. Penghindaran mobil juga diharapkan tidak mengakibatkan tabrakan lain akibat dari penghindaran mobil di depannya.



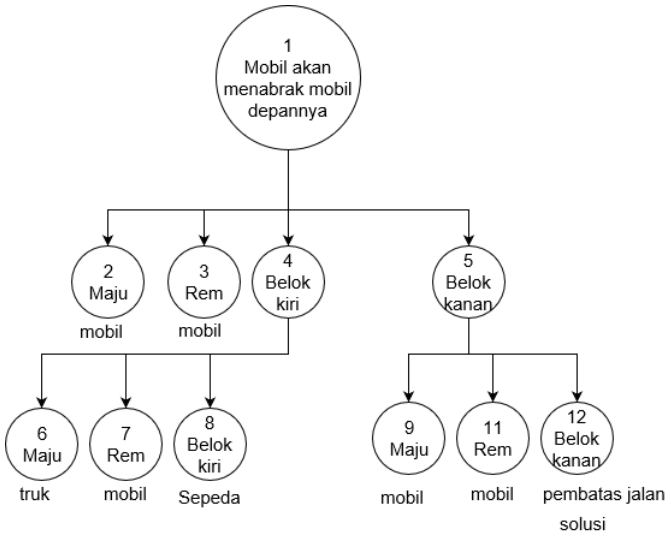
Gambar 4.4 penerapan BFS
Sumber : Koleksi pribadi penulis

Pada gambar 4.4 itu adalah contoh penerapan algoritma BFS dalam pencarian solusi. Kondisi yang dihadapkan adalah mobil akan menabrak mobil depannya. Mobil memiliki beberapa kemungkinan langkah, tetapi saat ini penulis hanya menuliskan sebagian kecil dari kemungkinan agar pembaca dapat mengerti dengan mudah. Pada setiap level mobil dihadapkan pada kemungkinan untuk melakukan gerakan maju, rem, belok kiri dan belok kanan. Pada level 1 mobil tidak mungkin melakukan langkah maju karena akan menabrak mobil depannya dan mobil tidak bisa mengerem karena akan tertabrak oleh mobil belakangnya(jarak mobil dengan mobil belakang tidak jauh). Jadi simpul yang di perluas hanyalah simpul belok kiri dan simpul belok kanan. Pada level dua mobil dihadapkan pada pilihan langkah yang sama, tetapi kita menghilankan langkah – langkah yang mengebalikan mobil ke keadaan semual seperti setelah berbelok kanan mobil akan langsung berbelok kiri lagi. Pada simpul – simpul hasil ekspansi dari simpul belok kiri semua bukan solusi dikarenakan semuanya mengakibatkan mobil mengalami tabrakan dengan mobil lain. Maka dari itu algoritma akan mengecek simpul – simpul hasil ekspansi dari simpul belok kanan, dan didapatkan solusi pada simpul maju karena setelah mobil menghindar tabrakan dengan berbelok kanan mobil dapat selamat dengan maju menghindari tabrakan. Urutan pembuatan simpul dapat dilihat dari penomoran simpul pada gambar 4.4. Proses ini efektif dikarenakan kita membandingkan solusi pada setiap level yang sama. Tidak seperti algoritma DFS yang tidak membandikan solusi pada setiap level. Pada contoh gambar 4.4 solusi ditemukan pada level dua. Pada kondisi ini mungkin terdapat solusi lain pada level berikutnya seperti kita melakukan ekspansi pada simpul 12, tetapi pada penerapan kondisi ini apabila kita membandikan solusi untuk selama antar hanya perlu melakukan dua langkah dan tiga langkah maka lebih optimal hanya melakukan dua langkah saja dikarenakan perbedaan waktu sedikit saja dapat berakibat fatal bagi pengendara mobil.

Pada gambar 4.4 simpul 12 juga sebenarnya membawa kita kepada solusi juga, akan tetapi simpul 9 yang dipilih karena memberikan kita resiko keselamatan yang lebih tinggi dan juga

kenyaman bagi pengendara karena resiko untuk maju terus lebih kecil dibandingkan harus berbelok lagi ke kanan dengan resiko dapat tertabrak lagi oleh kendaraan di lajur kanan. Jadi sebagai *programmer* kita tidak menetapkan solusi tujuan dari pencarian hanya bergantung pada kondisi selamat mobil tetapi kita juga memiliki berbagai faktor lain yang mempengaruhi nilai dari solusi yang kita dapatkan ini.

Pada gambar 4.4 merupakan kondisi yang memiliki solusi yang dapat menyelamatkan penumpang. Pada dunia nyata kondisi ini tidaklah selalu dapat terwujud terkadang mobil otomatis dihadapkan pada suatu kondisi yang tidak dapat terhindari. Maka dari itu algoritma juga harus dapat menyimpan kondisi terburuk beserta kerugian yang akan diterimakan mobil.



Gambar 4.5 Kondisi menabrak
Sumber : Koleksi pribadi penulis

Pada gambar 4.5 dapat dilihat bahwa semua kondisi mengakibatkan mobil menabrak sesuatu seperti menabrak truk mobil, sepeda maupun pembatas jalan. Pada kondisi ini algoritma diharuskan menuliskan kerugian yang didapatkan oleh mobil yaitu objek apa yang akan ditabrak. Setelah itu algoritma akan melihat kerugian yang dihasilkan apabila mobil otomatis menabrak objek tersebut. Setelah melakukan pembandingan ternyata didapat bahwa simpul 12 merupakan solusi dimana mobil akan menabrak pembatas jalan. Solusi ini disebut optimal dikarenakan mobil tidak akan mengakibatkan kerusakan yang cukup fatal bila dibandingkan dengan menabrakan mobil kepada objek yang sedang melaju.

Pengunaan strategi DFS dan IDS ini haruslah pada tempatnya. Apabila mobil otomatis menggunakan IDS pada kondisi menabrak maka karena kompleksitas waktunya tinggi maka mobil akan terlambat melakukan suatu aksi yang dapat menyelamatkan nyawa dari penumpangnya. Pada saat kondisi mobil hanya perlu mengamati keadaan sekitar tanpa perlu bertindak cepat kita tidak memerlukan BFS dikarenakan BFS sangatlah memakan banyak memori maka dari itu apabila memori yang diggunakan terlalu banyak maka akan menyebabkan memori dari mobil otomatis akan dipenuhi informasi yan sebenarnya tidak terllau diperlukan

V. KESIMPULAN

Algoritma pencarian solusi menggunakan strategi DFS dan BFS dapat diterapkan dalam penggunaan mobil otomatis. Strategi ini menjami mobil otomatis untuk menemukan solusi yang paling tepat agar dapat menjamin keselamatan pengguna mobil otomatis. Dengan bantuan strategi DFS dan BFS *programmer* juga dimudahkan untuk membandingkan resiko yang dihasilkan dari sebuah kecelakaan sehingga mobil otomatis dapat mengurangi resiko kecelakaan yang dapat terjadi

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepata Tuhan Yang Maha Esa untuk setiap berkatNYa dalam hidup saya sehingga saya dapat menyelesaikan makalah saya ini yang berjudul aplikasi DFS dan BFS dalam algoritma mobil otomatis. Penulis juga tidal lupa untuk berterima kasih pada orangtua saya yang selalu memberikan dukungan bagi saya untuk menyelesaikan makalah ini. Tak lupa penulis juga mengucapkan terima kasih kepada Dr. Masayu Leylia Khodra ST,MT selaku dosen IF2211 Strategi Algoritma penulis, berkat bantuannya penulis dapat memahami konsep dari DFS dan BFS sehingga dapat menyelesaikan makalah ini. Semoga dengan adany makalah ini pembaca dapat mendapatkan manfaat yang berarti dan berguna kedepannya.

REFERENSI

[1] Rinaldi Munir, *Diktat Kuliah Matematika Diskrit*. Bandung:Departemen Teknik Informatika,2003,bab 8.
[2] Rinaldi Munir, *Diktat Kuliah Strategi Algoritma*. Bandung:Departemen Teknik Informatika,2007,bab 6.

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

Ayrton Cyril-13516019