# Implementasi Intuitif Penyelesaian Labirin dengan Algoritma *Depth-First Search*

Joshua Atmadja, 13514098
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*joshuatmadja@gmail.com*

***Abstrak*—Labirin merupakan salah satu persoalan umum dalam dunia informatika. Penyelesaian labirin oleh komputer memerlukan desain algoritma yang efisien dan efektif. Beberapa contoh desain algoritmanya ialah algoritma pencarian melebar (BFS), algoritma pencarian mendalam (DFS), algoritma A* (A Star), dan algoritma pencarian iteratif mendalam (IDS). Akan tetapi, untuk manusia sendiri–termasuk makhluk hidup bergerak lainnya–akan mengalami kesulitan untuk memecahkan masalah labirin bila ia sendiri yang berada dalam labirin tersebut, apalagi bila labirin berukuran sangat luas. Manusia memiliki keterbatasan dalam menghapal rute labirin sehingga beberapa algoritma di atas kurang sesuai untuk diterapkan. Namun demikian, secara intuitif, manusia dapat menyelesaikan labirin dengan pendekatan algoritma DFS. Bagaimana caranya?**

***Kata kunci—algoritma pencarian mendalam; labirin; orientasi arah; pohon ruang status; runut balik.***

## I. PENDAHULUAN

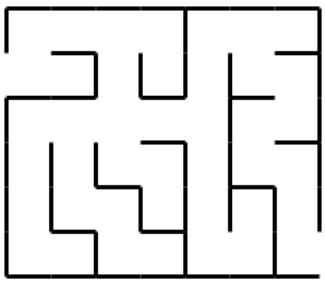
Dalam dunia informatika, kebanyakan persoalan-persoalan yang lazim ada di kehidupan sehari-hari dapat diselesaikan dengan menggunakan komputer. Komputer tersebut pastinya sudah diprogram atau dipasang perangkat lunak yang dirancang khusus untuk melakukan pemecahan persoalan tersebut. Sebagai contoh, kalkulator. Kalkulator diciptakan untuk membantu mempercepat pekerjaan terutama pekerjaan kalkulasi (menghitung). Kalkulator pun semakin berkembangnya zaman, semakin berkembang fitur dan fungsinya, contohnya menggambar kurva fungsi dan menghitung standar deviasi dari sampel sebuah populasi.

Dewasa ini marak aplikasi komputer, baik program kecil maupun perangkat lunak kompleks, membantu manusia dalam memecahkan masalah-masalah yang sederhana dan mudah dikerjakan dalam ukuran yang kecil namun kompleks dan sulit dalam ukuran yang besar. Beberapa persoalan tersebut ialah pengaturan jadwal penerbangan, penyelesaian program linier untuk optimisasi, pencarian rute terpendek, hingga penyelesaian sudoku. Ada satu persoalan yang juga masuk ke dalam himpunan persoalan dengan kriteria demikian, yaitu persoalan labirin.

Labirin (atau *maze*) menurut Google didefinisikan sebagai berikut.

> ***maze (noun):***
>
> "a network of paths and hedges designed as a puzzle through which one has to find a way."

Dari definisi tersebut, labirin diartikan sebagai sebuah teka-teki berbentuk gambar bidang yang terdiri atas kumpulan jalan yang terhubung satu sama lain dan dirintangi oleh tembok yang harus diselesaikan dengan cara mencari jalan keluar. Berikut adalah contoh ilustrasi labirin.



***Ilustrasi 1.1.*** *Labirin*

Beberapa desain algoritma yang sesuai dan populer untuk menyelesaikan persoalan ini ialah algoritma pencarian melebar (BFS) dan algoritma pencarian mendalam (DFS). Akan tetapi, kekurangan algoritma BFS ialah kebutuhan memori yang besar. Objek yang berada dalam labirin harus memiliki kapasitas memori (atau kemampuan mengingat) yang cukup besar untuk menyelesaikan persoalan labirin, yakni untuk mengingat rute perjalanan labirin yang sudah dilalui. Bagaimana bila objek tersebut memiliki kapasitas memori yang terbatas? Algoritma DFS ini dapat diimplementasikan dalam penyelesaian persoalan labirin secara intuitif.

Pada makalah kali ini, penulis akan memaparkan implementasi intuitif penyelesaian labirin dengan algoritma DFS. Asumsi penulis dalam menulis makalah ini ialah bahwa jalan di labirin hanya memungkinkan ke utara, selatan, barat, dan timur. Penulis memperingatkan bahwa tidak semua labirin murni dapat diselesaikan secara intuitif, melainkan ada juga yang membutuhkan memori.

## II. DASAR TEORI

### A. *Algoritma Pencarian Mendalam (DFS)*

Algoritma pencarian mendalam, atau lebih dikenal dengan DFS, merupakan algoritma traversal graf yang melakukan penelusuran simpul dengan pendekatan mendalam. Secara informal, algoritma DFS mengunjungi simpul dari akar ke simpul anak (*child node*) terus menerus hingga ke daun terlebih dahulu, kemudian runut balik ke simpul bapak (*parent node*) bila tidak menemukan solusi, lalu ditelusuri lagi simpul anak yang belum dikunjungi – bila sudah tidak ada yang belum dikunjungi, algoritma melakukan runut balik, begitu seterusnya hingga menemukan solusi.[2]

Secara formal, algoritma DFS diartikan sebagai berikut. Misalkan terdapat graf *G* yang mempunyai *n* buah simpul. Traversal (penelusuran) dimulai dari simpul *v*. Simpul yang berikutnya dikunjungi ialah simpul *w* yang bertetangga dengan simpul *v*, lalu pencarian mendalam dimulai lagi secara rekursif dari simpul *w*. Ketika mencapai pada sebuah simpul, misalnya *u*, sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul *w* yang belum dikunjungi. Pencarian mendalam kemudian dimulai lagi dari *w*. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.[3]

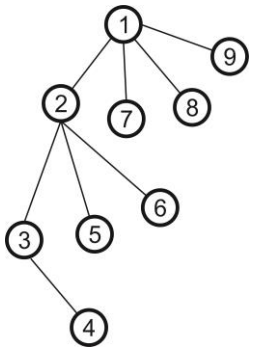
Berikut ini adalah algoritma DFS selengkapnya.[3]

```
procedure DFS(v: integer)

{Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpul yang dikunjungi
tertulis di layar

Asumsi: (deklarasi global)
- terdapat A yaitu matriks ketetanggaan graf
- terdapat visited yaitu array of boolean yang
menyatakan bahwa simpul tersebut sudah/belum
dikunjungi (default false)
- terdapat n yaitu integer yang menyatakan
banyak simpul dalam graf
}

Deklarasi:
  w : integer

Algoritma:
  write(v)
  visited[v]←true
  for w←1 to n do
   if A[v,w]=1 then
    if not visited[w] then
     DFS(w)
    endif
   endif
  endfor
```

Pemanggilan prosedur DFS di program utama, sekaligus inisialisasi variabel globalnya, ialah sebagai berikut.

```
readGraph(A,n)
{prosedur yang digunakan untuk membaca matriks
ketetanggan graf beserta mencari berapa banyak
simpul pada graf tersebut; sudah terdefinisi}

for i←1 to n do
  visited[i]←false
endfor

read(v) {simpul awal}
DFS(v)
```

Ilustrasi 2.1. menggambarkan bagaimana algoritma DFS bekerja pada sebuah graf khusus yaitu pohon berakar. Bila simpul nomor 1 (simpul akar) merupakan simpul yang paling pertama dikunjungi, maka urutan pengunjungan simpul bersesuaian dengan nomor simpul yang tertera pada graf.

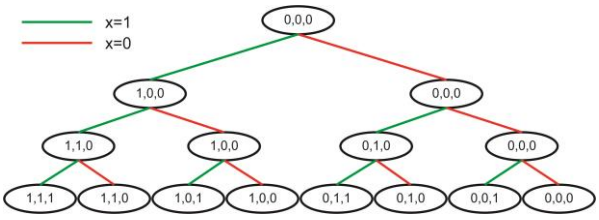



***Ilustrasi 2.1.*** *Algoritma DFS yang diterapkan pada pohon berakar.*
*Dibuat dengan CorelDraw*

### B. *Pohon Ruang Status*

Pohon ruang status ialah sebuah pohon dinamis yang simpul-simpulnya dibentuk dari status persoalan yang ditrans-formasikan menjadi status solusi oleh suatu operator. Akar pada pohon ruang status menyatakan status awal sedangkan daun menyatakan status solusi. [3]

Sebagai contoh, Ilustrasi 2.2 menggambarkan pohon ruang status yang dibentuk untuk memecahkan masalah *integer (0/1) knapsack*. Persoalan ini bergantung dengan banyaknya macam benda yang akan dimasukkan ke dalam *knapsack*. Ilustrasi 2.2 menggambarkan terdapat 3 macam benda yang akan dimasukkan ke dalam *knapsack* ($n$ = 3).




***Ilustrasi 2.2****. Pohon Ruang Status Integer (0/1) Knapsack dengan n=3.*
*Dibuat dengan CorelDraw*

Secara umum, simpul pada pohon ruang status *integer knapsack* ialah sebagai berikut.

$$(x_1, x_2, x_3, \ldots, x_n)$$

dengan *n* adalah banyak macam benda yang ingin dimasukkan ke dalam *knapsack*, dan juga menyatakan aras maksimal yang dapat dibentuk pada pohon dinamis. Operator transformasi pohon dinamis tersebut ialah dengan menyubstitusikan $x_k$ dengan 0 atau 1, dimana *k* menyatakan aras pohon yang akan dibentuk.

Setiap simpul pada Ilustrasi 2.2 menyatakan status persoalan. Status awal adalah akar yang berupa status (0,0,0). Setiap daun pada pohon tersebut menyatakan status

solusi, dan merupakan ruang solusi. Karena $n$ = 3, aras maksimal yang akan terbentuk pada pohon dinamis ialah 3, dan status persoalannya dinyatakan dalam bentuk

$$(x_1, x_2, x_3)$$

dengan *x* bernilai 0 atau 1.

### C. *Algoritma Runut Balik*

Ketika algoritma DFS menemukan simpul daun yang tidak memberikan solusi, maka algoritma DFS melakukan runut balik (*backtracking*). Algoritma runut balik ini berbasis pada DFS untuk mencari solusi persoalan yang lebih efisien.[3] Algoritma runut balik dapat dilakukan dengan dua pendekatan, yaitu pendekatan iteratif dan pendekatan rekursif. Pada makalah ini, penulis akan membahas hanya algoritma runut balik dengan pendekatan rekursif.

Secara umum, algoritma runut balik rekursif (*recursive backtracking*) berguna untuk persoalan pemuasan batasan (*constraint satisfaction problems*) yang terdiri atas memasukkan nilai kepada peubah-peubah sesuai kumpulan batasan-batasan persoalan. Misalnya, persoalan n-ratu memiliki peubah berupa posisi ratu di setiap baris bidang catur dengan batasan bahwa tidak ada dua ratu berada dalam baris, kolom, dan diagonal yang sama. Contoh lainnya ialah persoalan pewarnaan peta/graf – peubahnya ialah warna masing-masing kota (graf) dengan batasan bahwa tidak ada dua kota bertetangga yang memiliki warna yang sama.[4]

*Pseudocode* umum algoritma runut balik rekursif ialah sebagai berikut.[4]

```
void cariSolusi(n, parameter lainnya){
  if(solusi ditemukan){
    banyakSolusi++;
    tampilkanSolusi();
    if(banyakSolusi >= targetSolusi)
      printf("Solusi sudah ketemu");
    return;
  }

  for (nilai = awal to akhir){
    if(isValid(nilai, n)){
      terapkanNilai(nilai, n);
      cariSolusi(n+1, parameter lainnya);
      hapusNilai(nilai, n);
    }
  }
}
```

*Pseudocode* di atas berlaku untuk persoalan yang ingin dicari satu atau banyak solusi. *Pseudocode* di bawah ini dapat diterapkan hanya untuk mencari solusi tunggal.[4]

```
boolean cariSolusi(n, parameter lainnya){
  if(solusi ditemukan){
    banyakSolusi++;
    tampilkanSolusi();
    return true;
  }

  for (nilai = awal to akhir){
    if(isValid(nilai, n)){
      terapkanNilai(nilai, n);
      if(cariSolusi(n+1, parameter lainnya))
        return true;
      hapusNilai(nilai, n);
    }
  }

  return false;
}
```

Sebagai contoh penerapan *pseudocode* di atas, kita tinjau persoalan n-ratu dalam mencari kolom yang aman. Berikut adalah kode sumbernya.[4]

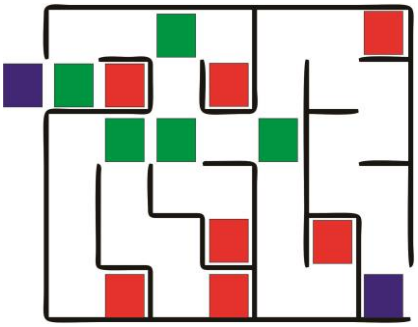
```
public void cariKolomAman(int baris){
  if(baris==ukuranPapan){
    banyakSolusi++;
    tampilkanSolusi();
    if(banyakSolusi >= targetSolusi)
      printf("Solusi sudah ketemu");
    return;
  }

  for (int kolom = 0; kolom < ukuranPapan;
kolom++) {
    if(isAman(baris, kolom)){
      letakkanRatu(baris, kolom);
      cariKolomAman(baris + 1);
      hapusRatu(baris,kolom);
    }
  }
}
```

Terlihat ada kemiripan kode sumber dengan pseudocode umum algoritma runut balik rekursif. Solusi dari contoh persoalan di atas ialah bila nilai *baris* secara lojik sudah sama dengan ukuran papan. Kode sumber di atas menghapus ratu sebelumnya bila ratu berikutnya tidak bisa memenuhi *isAman(baris, kolom)* kemudian mencoba letak yang baru. Bila tidak ada letak yang baru, maka ratu yang sebelumnya lagi dihapus lalu berpindah ke letak baru, dan begitu seterusnya, hingga mencapai solusi.

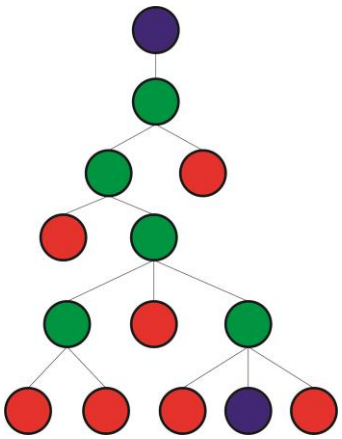
## III. PEMBAHASAN

### A. *Representasi Labirin menjadi Graf*

Labirin dapat direpresentasikan menjadi graf dalam berbagai macam cara dan pendekatan. Pada makalah ini, dengan asumsi yang telah disebutkan sebelumnya, labirin direpresentasikan menjadi graf sedemikian sehingga simpulnya ialah merupakan persimpangan, baik pertigaan maupun perempatan, atau jalan buntu; dan sisinya ialah jalan penghubung persimpangan satu dengan lainnya, termasuk dengan jalan buntu. Labirin memiliki 3 jenis simpul, yaitu simpul pintu masuk atau pintu keluar, simpul persimpangan, dan simpul jalan buntu. Ilustrasi 3.1. menggambarkan simpul-simpul pada labirin di Ilustrasi 1.1. yang akan menjadi graf yang digambarkan di Ilustrasi 3.2.



***Ilustrasi 3.1.*** *Labirin dengan blok warna simpul.*
*Sumber: open.kattis.com, dimodifikasi dengan CorelDraw*



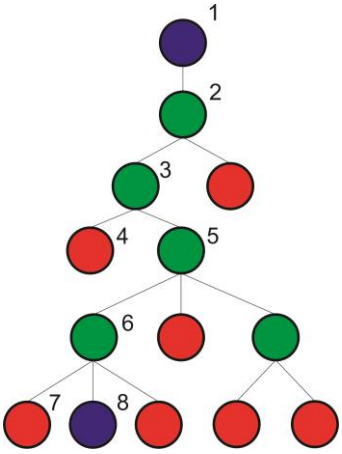
***Ilustrasi 3.2.*** *Graf hasil representasi labirin pada Ilustrasi 3.1. Dibuat dengan CorelDraw*

Warna merah pada labirin menandakan jalan buntu, warna hijau menandakan persimpangan, dan warna biru menandakan pintu masuk dan pintu keluar dari labirin tersebut. Terlihat pada Ilustrasi 3.2. bahwa simpul berwarna hijau akan membangkitkan simpul baru, sedangkan simpul berwarna merah tidak membangkitkan simpul lagi. Kasus untuk simpul biru berbeda, yakni simpul yang merupakan status awal pasti akan membangkitkan satu simpul saja, sedangkan simpul yang merupakan status solusi tidak akan membangkitkan apapun. Graf pada Ilustrasi 3.2. juga menggambarkan pohon ruang status dimana operator transformasinya ialah arah belokan yang diambil ketika bertemu simpangan, antara lain kiri, lurus, dan kanan.

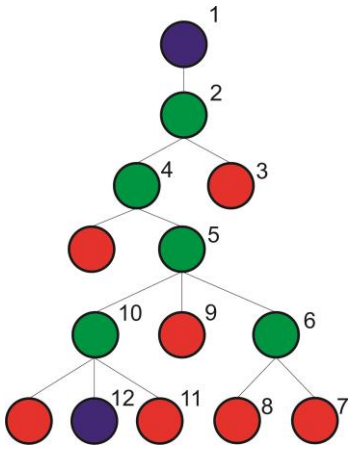
*B. Prioritas Arah*

Implementasi intuitif algoritma DFS ini dapat menyelesaikan persoalan labirin jika sejak awal algoritma ini diselesaikan, terdapat prioritas simpul yang harus dikunjungi dan diproses terlebih dahulu. Maksud dari prioritas di sini ialah orientasi arah pengunjungan simpul pada graf, antara lain: kiri ke kanan atau kanan ke kiri. Prioritas arah penyelesaian algoritma ini haruslah konsisten sejak awal tahap penyelesaian. Bila di tengah-tengah penyelesaian arah berganti, maka algoritma DFS tidak dapat menyelesaikan labirin secara intuitif.

Ilustrasi 3.3. dan 3.4. menggambarkan penyelesaian labirin dengan algoritma DFS dan prioritas arahnya.




***Ilustrasi 3.3.*** *Urutan pengunjungan simpul algoritma DFS dengan prioritas arah kiri ke kanan. Dibuat dengan CorelDraw*



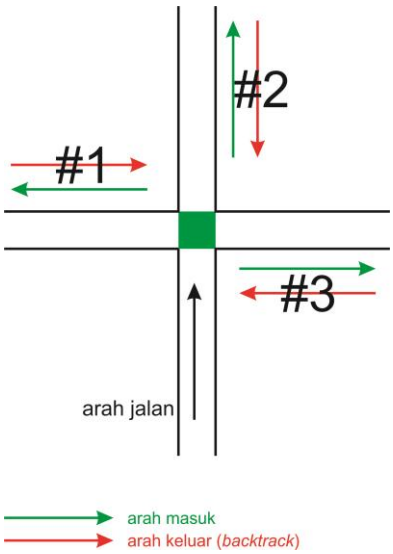

***Ilustrasi 3.4.*** *Urutan pengunjungan simpul algoritma DFS dengan prioritas kanan ke kiri. Dibuat dengan CorelDraw.*

Makalah ini menekankan bahwa prioritas arah sangat penting dalam penyelesaian intuitif labirin. Prioritas arah ini tidak hanya diterapkan pada pengunjungan simpul, melainkan juga harus diterapkan saat pembangkitan simpul graf representasi labirin. Tinjau kembali Ilustrasi 3.1. dan Ilustrasi 3.2. Dari kedua ilustrasi tersebut, terlihat bahwa simpul-simpul pada graf representasi labirin tersebut dibentuk sesuai dengan prioritas arah belokan ke kiri terlebih dahulu.

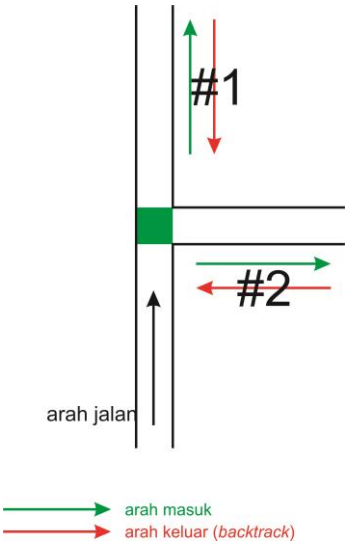
Adapun algoritma pengunjungan simpul dan pembangkitan simpul sesuai dengan prioritas arah, misalnya kiri, adalah sebagai berikut.

- Apabila simpul yang dikunjungi ialah simpul jalan buntu, STOP.
- Apabila simpul yang dikunjungi ialah simpul persimpangan, periksa bentuk simpangannya dan jalan baru yang bisa dilalui oleh objek. Setidaknya dua di antara tiga tahap dibawah ini pasti akan dikerjakan.
- Bila dari arah masuk persimpangan terdapat belokan ke kiri, bangkitkan/kunjungi simpul yang dihubungkan dengan jalan tersebut. Ulangi dari langkah 1 dengan posisi sekarang ialah simpul yang baru dibangkitkan atau dikunjungi. Bila tidak ada belokan ke kiri, lanjutkan ke tahap berikutnya.
- Bila dari arah masuk persimpangan terdapat jalan lurus, bangkitkan/kunjungi simpul yang dihubungkan dengan jalan lurus tersebut. Ulangi dari langkah 1 dengan posisi sekarang ialah simpul yang baru dibangkitkan atau dikunjungi. Bila tidak ada jalan lurus, lanjutkan ke tahap berikutnya.
- Bila dari arah masuk persimpangan terdapat belokan ke kanan, bangkitkan/kunjungi simpul yang dihubungkan dengan belokan ke kanan. Ulangi dari langkah 1 dengan posisi sekarang ialah simpul yang baru dibangkitkan atau dikunjungi. Bila tidak ada belokan ke kanan, lanjutkan ke tahap berikutnya.
- Runut balik ke simpul persimpangan sebelumnya.
- Bila simpul sekarang ialah simpul awal, STOP.

Ilustrasi 3.5. dan Ilustrasi 3.6. menggambarkan algoritma di atas untuk kasus persimpangan berbentuk perempatan dan pertigaan.




***Ilustrasi 3.5.*** *Algoritma pengunjungan persimpangan dengan prioritas arah kiri pada perempatan. Dibuat dengan CorelDraw*




***Ilustrasi 3.6.*** *Algoritma pengunjungan persimpangan dengan prioritas arah kiri pada pertigaan. Dibuat dengan CorelDraw.*

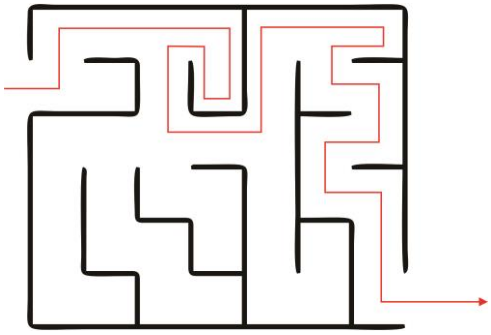
### *C. Penerapan Intuitif Algoritma DFS*

Dengan graf yang sudah dibentuk dari labirin dan prioritas arah yang sudah ditentukan, baik kiri maupun kanan, kita bisa langsung menerapkan algoritma DFS secara intuitif untuk menyelesaikan labirin. Pada makalah ini, penulis menggunakan prioritas arah kiri.

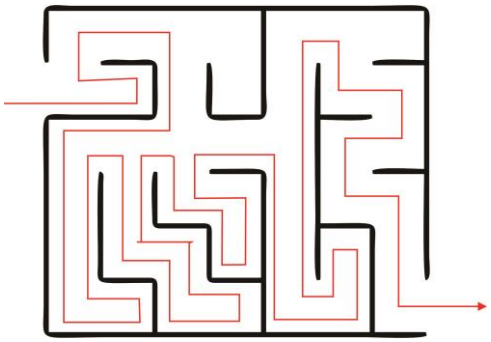
Menurut wikiHow, cara mudah menyelesaikan labirin ialah mengikuti temboknya. Karena kebanyakan labirin, baik yang digambar maupun yang nyata dapat diselesaikan oleh manusia atau objek lain, mulai dan selesai pada pinggir bidang labirin, solusi sederhana ialah mengikuti bentuk temboknya. Bila objek mengikuti tembok di sisi kiri atau kanannya, objek pasti akan mencapai pintu keluar. Perlu dicatat bahwa mengikuti temboknya harus benar-benar dimulai dari awal masuk labirin.[1]

Pernyataan dari wikiHow di atas merangkum subbab di atas yakni mengenai representasi graf dan prioritas arah. Dengan mengikuti tembok di labirin, secara otomatis objek di dalam labirin memprioritaskan arah belokannya. Bila tembok yang diikuti oleh objek ialah tembok di sisi kiri, maka prioritas arah objek ialah kiri, dan begitupun sebaliknya.

Ilustrasi 3.7. dan 3.8. menggambarkan penyelesaian labirin pada Ilustrasi 1.1. dengan prioritas arah kiri dan prioritas arah kanan. Tampak bahwa untuk kasus labirin tersebut, prioritas arah kiri dapat memberikan solusi yang efisien waktu.



***Ilustrasi 3.7.*** *Penyelesaian labirin secara intuitif dengan algoritma DFS dan prioritas arah kiri. Sumber: open.kattis.com, dimodifikasi dengan CorelDraw.*



***Ilustrasi 3.8.*** *Penyelesaian labirin secara intuitif dengan algoritma DFS dan prioritas arah kanan. Sumber: open.kattis.com, dimodifikasi dengan CorelDraw.*

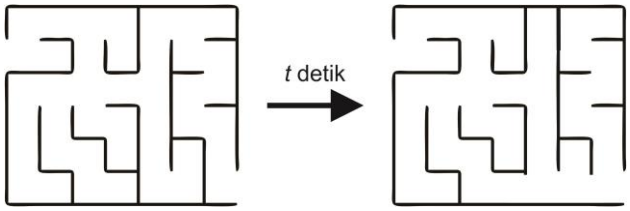
## IV. PENGECUALIAN

Asumsi penulis dalam membahas implementasi di bab sebelumnya pada makalah ini ialah: (1) bahwa labirin bersifat statis yakni labirin tidak mengalami perubahan bentuk dan ukuran jalan ataupun tembok halangan serta tidak ada perubahan mekanis pada kurun waktu tertentu– tidak ada pintu yang dalam waktu tertentu terbuka/tertutup otomatis, dan sebagainya; (2) bahwa tidak ada rintangan atau jebakan di dalam labirin yang membuat objek celaka; (3) bahwa labirin memiliki pintu masuk dan pintu keluar yang berbeda tempat; dan (4) bahwa tidak ada jalan pada labirin yang membentuk sirkuit. Bagaimana bila empat asumsi tersebut dilanggar?

### *A. Labirin Tidak Statis*

Implementasi intuitif berlaku bila kondisi labirin tetap (statis). Maksudnya ialah labirin tetap memiliki jalan dan tembok dengan konfigurasi yang sama selama objek masih berada di dalam labirin untuk menyelesaikan labirin tersebut. Bila labirin mengalami perubahan bentuk jalan dan konfigurasi tembok seperti pada Ilustrasi 4.1. saat objek masih berada di dalam labirin untuk mencari jalan keluar

dalam waktu *t* detik, implementasi intuitif algoritma DFS akan gagal dijalankan.



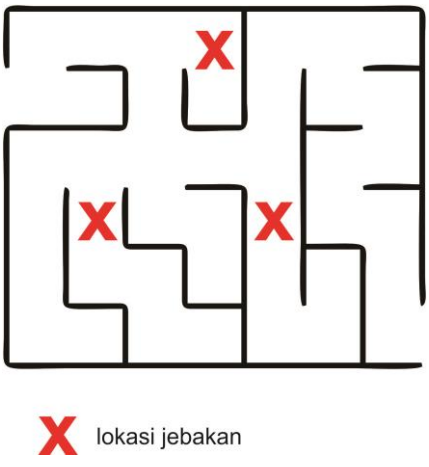

***Ilustrasi 4.1.*** *Labirin dinamis.*
*Sumber: open.kattis.com, dimodifikasi dengan CorelDraw*

Bila terjadi demikian, simpul dan sisi pohon ruang status labirin juga banyak berubah sehingga pohon harus banyak diubah, lalu kunjungan simpul juga jadi berubah yang mengakibatkan algortima DFS gagal dijalankan. Solusi untuk pengecualian ini ialah dengan menerapkan intelegensi buatan kepada objek (terkecuali manusia, yang intelegensinya murni) supaya bisa mendeteksi perubahan konfigurasi pada labirin untuk menuju pintu keluar. Selain itu, objek juga perlu diberi kemampuan mengingat jalan yang sudah dilalui.

### *B. Objek di dalam Labirin Menjadi Celaka*

Lain halnya dengan labirin dinamis, kondisi labirin kali ini ialah terdapat suatu objek lain yang dapat mencelakakan objek yang sedang mencari jalan keluar, misalnya jebakan. Jebakan yang dimaksud bermacam-macam dampaknya, mulai dari terluka, mengurangi kesempatan hidup, hingga membunuh objek sekalipun.

Bila objek menerapkan cara intuitif untuk menerapkan algoritma DFS di kondisi labirin yang demikian, kemungkinan objek hidup dan selamat di pintu keluar sangatlah kecil. Ilustrasi 4.2. menggambarkan contoh labirin yang memiliki jebakan. Bila objek menyelesaikan labirin dengan prioritas arah kiri, maka ia akan terkena jebakan hanya sekali. Sedangkan, bila objek menyelesaikan labirin tersebut dengan prioritas arah kanan, maka ia akan terkena jebakan sebanyak dua kali.

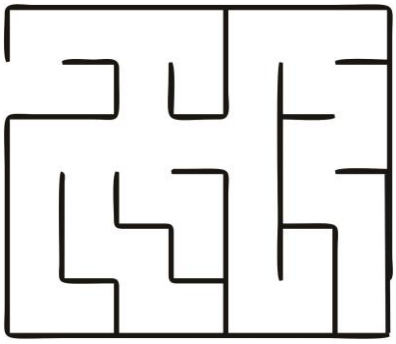



***Ilustrasi 4.2.*** *Labirin yang memiliki jebakan*
*Sumber: open.kattis.com, dimodifikasi dengan CorelDraw*

Solusi dari pengecualian ini ialah dengan tetap menggunakan intelegensi buatan supaya objek menerapkan *trial and error* dengan nyawanya yang tak berhingga. Sangat sulit diterapkan kepada manusia normal pada umumnya karena keterbatasan nyawa dan kesempatan hidupnya. Intelegensi buatan itu tidak hanya memungkinkan nyawa yang tak terbatas melainkan juga mempelajari letak jebakan tersebut sehingga bila objek hidup kembali, ia sudah mengenali di mana jebakan tersebut berada sehingga tidak akan melewati letak jebakan tersebut.

### *C. Pintu Keluar sama dengan Pintu Masuk (Labirin Satu Pintu)*

Tujuan utama dari persoalan labirin ialah mencari jalan keluar. Namun, apabila labirin tersebut hanya memiliki satu pintu saja, yakni pintu masuk yang sama dengan pintu keluar, pengecualian ini, meskipun berhasil menerapakan implementasi intuitif algoritma DFS dan tidak merugikan nyawa objek, sangat merugikan waktu terutama labirin yang ukurannya sangat besar.



***Ilustrasi 4.3.*** *Labirin satu pintu.*
*Sumber: open.kattis.com, dimodifikasi dengan CorelDraw*

Tidak ada solusi dari pengecualian ini tetapi kita dapat melakukan pemeriksaan awal terlebih dahulu. Salah satu bentuk pemeriksaannya ialah dengan menerapkan algoritma Floodfill 4 arah. Salah satu implementasi Floodfill 4 arah ialah dengan rekursif berbasis tumpukan (*stack-based recursive*). Berikut ini adalah pseudocode implementasi Floodfill tersebut.[5]

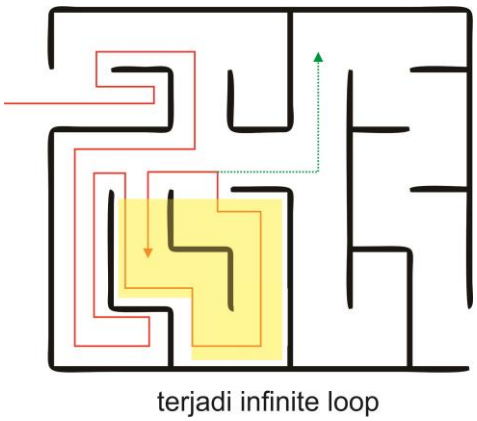
```
void Floodfill(petak, warnaTarget, warnaIsi){
 if(warnaTarget == warnaIsi) return;
 if(petak.warna != warnaTarget) return;
 petak.warna = warnaIsi;

 Floodfill(petak atas, warnaTarget, warnaIsi);
 Floodfill(petak bawah, warnaTarget, warnaIsi);
 Floodfill(petak kiri, warnaTarget, warnaIsi);
 Floodfill(petak kanan, warnaTarget, warnaIsi);
 return;
}
```

Cara memeriksanya cukup sederhana yaitu dengan "menutup" terlebih dahulu pintu masuk, kemudian memulai algoritma Floodfill dari petak (*node*) pertama yang dekat dengan pintu masuk. Bila ada petak yang diwarnai di luar batas bidang labirin, berarti labirin memiliki pintu keluar yang berbeda dengan pintu masuk. Bila Floodfill berhenti tanpa ada petak yang diwarnai di luar labirin, maka labirin tidak memiliki pintu keluar solusi.

### *D. Lintasan Labirin Membentuk Sirkuit*

Lain halnya dengan labirin dinamis, lintas labirin ini statis namun membentuk sirkuit. Bila labirin ini diselesaikan dengan implementasi intuitif algoritma DFS, akan terjadi kemungkinan *infinite loop*. Hal tersebut juga

akan mengubah pohon ruang status representasi labirin menjadi sebuah graf pada umumnya, tidak lagi menjadi sebuah pohon berakar. Ilustrasi 4.4. menggambarkan labirin yang mempunyai lintasan sirkuit diselesaikan secara intuitif dengan algoritma DFS dan prioritas arah kanan.




***Ilustrasi 4.4.*** *Objek mengalami* infinite loop *saat melalui lintasan sirkuit dalam labirin.*
*Sumber: open.kattis.com, dimodifikasi dengan CorelDraw*

Memang tidak semua labirin tidak akan menyebabkan *infinite loop* apabila beruntung namun kondisi ini bisa dicegah dengan menambahkan memori kepada objek supaya dapat mengingat persimpangan yang sudah dilalui oleh objek. Dengan begitu, objek dapat menelusuri jalan lain dan menemukan simpangan baru untuk dilalui.

## V. Simpulan

Implementasi intuitif penyelesaian labirin dengan algoritma DFS dapat membantu manusia dan objek lain untuk menyelesaikan labirin tanpa ada usaha mengingat sama sekali. Efisiensi waktu sangat bergantung terhadap bentuk labirin dan prioritas arah yang diambil oleh objek saat memulai memecahkan masalah ini. Tidak semua kondisi dan bentukan labirin dapat membuat implementasi intuitif ini berhasil diterapkan termasuk dalam hal mengingat lintasan, persimpangan, dan jalan buntu labirin tersebut. Sehingga labirin yang dapat diselesaikan dengan implementasi intuitif algoritma DFS ialah labirin yang sedemikian sehingga dapat dibentuk sebuah pohon ruang status dimana simpul-simpulnya ialah jalan masuk atau keluar, persimpangan, dan jalan buntu; serta memenuhi empat asumsi tersebut di atas.

## VI. Ucapan Terima Kasih

Pertama, penulis mengucapkan syukur dan terima kasih kepada Tuhan Yang Maha Esa karena rahmat dan berkat karunia-Nya yang selalu menyertai penulis hingga makalah ini selesai dirancang dan ditulis. Penulis juga berterimakasih kepada kedua orang tua penulis yang selalu mendukung penulis melalui doa dan bantuan dalam bentuk moral dan material. Akhirnya, penulis juga berterimakasih kepada Bapak Rinaldi Munir dan Ibu Nur Ulfa Maulidevi selaku dosen pengajar Strategi Algoritma.

## Daftar Pustaka

[1] *How to Find Your Way Through A Maze*. (t.thn.). Dipetik Mei 7, 2016, dari wikiHow: http://www.wikihow.com/Find-Your-Way-Though-a-Maze

[2] Kleinberg, J., & Tardos, E. (2006). *Algorithm Design.* Massachussets: Pearson Education, Inc.

[3] Munir, R. (2009). *Diktat Kuliah IF2211 Strategi Algoritma.* Bandung.

[4] Sullivan, Ph. D., D. G. (2012, Fall). Recursion and Recursive Backtracking. United States: Harvard Extention School. Dipetik Mei 7, 2016, dari http://www.fas.harvard.edu/~cscie119/lectures/recursion.pdf

[5] Vandevenne, L. (2004). *Flood Fill*. Dipetik Mei 8, 2016, dari Lode's Computer Graphics Tutorial: http://lodev.org/cgtutor/floodfill.html#4-Way_Recursive_Method_floodFill4

## 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, 6 Mei 2016

Joshua Atmadja 13514098