# Penerapan Algoritma BFS dan DFS dalam Permainan Ular Tangga

Christ Angga Saputra – 13512019[1]
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
[1]*christanggasaputra@gmail.com*

***Abstract*—Makalah ini berisi tentang pencarian jalur optimum pada permainan klasik yang terkenal, yaitu permainan ular tangga. Permainan ular tangga berkembang di India dan menjadi terkenal di Eropa. Banyak pertanyaan bermunculan untuk mencari jalur yang optimal dalam menyelesaikan ular tangga ini. Pencarian jalur optimum dalam permainan ular tangga menggunakan dua strategi algoritma, yaitu Breadth-First Search (BFS) dan Depth-First Search (DFS).**

***Index Terms*—BFS, DFS, permainan, ular tangga.**

## I. PENDAHULUAN

Hiburan adalah hal yang tidak dapat dipisahkan dari kehidupan manusia dari masa ke masa. Mulai dari anak-anak hingga usia lanjut hiburan tidak dapat dipisahkan dari seluruh aktivitas. Bahkan ada satu kutipan terkenal yang menyebutkan "*Study hard, play harder*". Hal ini membuktikan semua orang membutuhkan hiburan.

Hiburan yang ada di abad ke-21 ini berbeda dengan abad ke-20 dan masa-masa sebelumnya. Di abad ke-21 ini, yang disebut sebagai era informasi, hiburan yang sedang marak adalah hiburan teknologi, baik dari *TV*, *media social*, *game online*, dan lain-lain. Di kalangan anak muda khususnya, seiring dengan berkembangnya teknologi *smartphone* yang sangat cepat, hiburan anak kecil pun terpengaruhi dengan teknologi tersebut. Contoh-contoh permainan anak saat ini adalah DotA (*Defense of the Ancients*), *Clash of Clans*, *Brave Frontier* dan masih banyak lagi. Akibatnya, anak-anak saat ini cenderung menghabiskan waktunya sendiri dan membuat karakter anak menjadi individualis.

Jika kita bandingkan dengan hiburan abad ke-20, dimana teknologi komputer belum marak, hiburan anak-anak cenderung membutuhkan interaksi satu sama lain. Hal ini akan membuat anak berkembang sebagai anak yang aktif dan sosialis. Contoh-contoh permainan anak pada masa tersebut adalah sondak, monopoli, ludo, ular tangga, dan masih banyak lagi.


Gambar 1. Contoh permainan Monopoli

## II. ULAR TANGGA

Ular tangga adalah sebuah permainan yang melibatkan bidak, dadu, dan sebuah papan permainan khusus. Ular tangga pertama kali ditemukan di India dengan nama *Moksha-Patamu*. Permainan ini adalah permainan moral dengan tangga merepresentasikan kebaikan dan ular merepresentasikan kejahatan. Permainan ini digunakan oleh agama Hindu untuk mengajarkan tentang kebaikan akan mengantarkan pemain ke kehidupan yang lebih tinggi dan kejahatan akan membawa pemain reinkarnasi ke tingkat kehidupan yang lebih rendah. Kotak nomor 100 merepresentasikan *Nirvana*.

Permainan ini cocok dimainkan oleh 2 orang atau lebih. Papan permainan terdiri dari kotak 10x10 dan antar satu kotak dengan kotak lainnya terhubung dengan ular atau tangga. Jika yang menghubungkan adalah ular, pemain yang berada di kotak tempat kepala ular harus turun ke kotak tempat ekor ular tersebut berada. Sebaliknya, jika yang menghubungkan adalah tangga, pemain harus menaiki tangga dari yang kotak lebih kecil nomornya ke kotak yang lebih besar. Permainan dimulai dari kotak nomor 1. Pemain memainkan permainan ini

dengan melempar dadu dan bergerak sesuai lemparan dadu tadi. Pemain yang pertama kali sampai di kotak nomor 100 adalah pemenangnya.

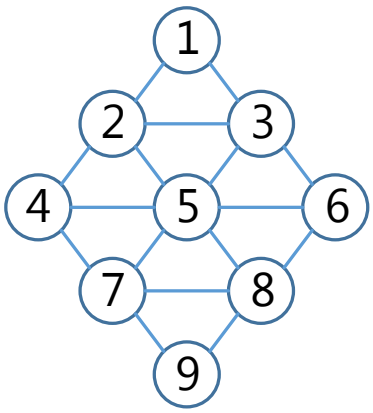
Gambar 2. Contoh permainan ular tangga

## III. Dasar Teori

### A. Breadth-First Search (BFS)

*Breadth-First Search* (BFS) adalah salah satu strategi pencarian dalam suatu graf yang terdiri dari 2 proses utama, yaitu mengunjungi suatu simpul dari graf dan mengeceknya, kemudian mengunjungi lagi semua simpul yang bertetangga dengan simpul sekarang dan mengeceknya. Penelusuran BFS dimulai dari simpul akar, kemudian dilanjutkan dengan penelusuran semua simpul tetangga yang belum pernah dikunjungi sebelumnya, begitu seterusnya hingga seluruh simpul dalam graf dikunjungi atau tujuan penelusuran tercapai. Dalam implementasinya, algoritma ini menggunakan struktur data antrian (*queue*) untuk menyimpan urutan simpul yang akan dikunjungi.

Berikut adalah ilustrasi penjelasan algoritma BFS. Terdapat sebuah graf G dengan 9 simpul seperti Gambar 3. Penelusuran dimulai dari simpul nomor 1.




Gambar 3. Ilustrasi graf G

Berikut adalah tabel hasil penelusuran BFS dalam graf G.

| iterasi | V | Q | *visited* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1 | {1} | T | F | F | F | F | F | F | F | F |
| 1 | 1 | {2,3} | T | T | T | F | F | F | F | F | F |
| 2 | 2 | {3,4.5} | T | T | T | T | T | F | F | F | F |
| 3 | 3 | {4,5,6} | T | T | T | T | T | T | F | F | F |
| 4 | 4 | {5,6,7} | T | T | T | T | T | T | T | F | F |
| 5 | 5 | {6,7,8} | T | T | T | T | T | T | T | T | F |
| 6 | 6 | {7,8} | T | T | T | T | T | T | T | T | F |
| 7 | 7 | {8,9} | T | T | T | T | T | T | T | T | T |
| 8 | 8 | {9} | T | T | T | T | T | T | T | T | T |
| 9 | 9 | {} | T | T | T | T | T | T | T | T | T |

Tabel 1. Tabel hasil penelusuran graf G dengan BFS

Dari tabel di atas, iterasi menggambarkan penelusuran dilakukan pada iterasi ke berapa, V menggambarkan simpul (*vertex*) graf yang sedang ditelusuri, Q menggambarkan antrian (*queue*) penelusuran dari graf, dan *visited* menggambarkan simpul mana saja yang telah ditelusuri (T artinya *True*, F artinya *False*).

Penelusuran dimulai dari simpul nomor 1, yaitu iterasi ke 0. Selanjutnya, dari simpul nomor 1 ditemukan simpul nomor 2 dan 3. Maka, simpul 2 dan 3 dimasukkan ke dalam Q dan dianggap telah dikunjungi. Penelusuran dilakukan secara berulang hingga seluruh simpul dikunjungi. Jadi, urutan penelusuran dari graf G di atas dengan menggunakan algoritma BFS adalah 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9.

Berikut adalah *pseudocode* dari algoritma BFS yang telah dijelaskan di atas.

```
procedure BFS(G,s)
Q ← {} // inisialisasi Queue Q
visited[s] ← true // menset state s dengan true
ENQUEUE(Q,s)
while (Q) not empty do
   u ← DEQUEUE(Q)
   foreach v bertetangga to u do
      if not visited[s] then
//   mengecek dan menelusuri seluruh simpul tetangga yang ada dari simpul utama
         ENQUEUE(Q,v)
         visited[v] ← true
      endif
   endfor
endwhile
```

*B. Depth-First Search (DFS)*

Algoritma *Depth-First Search* (DFS) tidak jauh berbeda dengan algoritma BFS, sama-sama menelusuri simpul yang bertetangga dengan simpul sekarang. Berikut adalah penjelasan tentang algoritma DFS. Penelusuran dimulai dari simpul akar. Dari simpul akar dicari satu simpul yang bertetangga dengannya. Saat ditemukan, penelusuran akan dilakukan pada simpul baru tadi dan mengulangi lagi pencarian simpul tetangga dan seterusnya.

Untuk lebih jelasnya, berikut adalah ilustrasi penjelasan algoritma DFS. Diberikan suatu graf G yang sama seperti ilustrasi BFS, yaitu gambar 3. Jumlah simpul pada graf G adalah 9 dan penelusuran dimulai dari simpul nomor 1.

Berikut adalah tabel hasil penelusuran DFS dalam graf G.

| iterasi | V | *next* V | *visited* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1 | {} | T | F | F | F | F | F | F | F | F |
| 1 | 1 | 2 | T | F | F | F | F | F | F | F | F |
| 2 | 2 | 3 | T | T | F | F | F | F | F | F | F |
| 3 | 3 | 5 | T | T | T | F | F | F | F | F | F |
| 4 | 5 | 4 | T | T | T | F | T | F | F | F | F |
| 5 | 4 | 7 | T | T | T | T | T | F | F | F | F |
| 6 | 7 | 8 | T | T | T | T | T | F | T | F | F |
| 7 | 8 | 6 | T | T | T | T | T | F | T | T | F |
| 8 | 6 | {BT} | T | T | T | T | T | T | T | T | F |
| 9 | 8 | 9 | T | T | T | T | T | T | T | T | F |
| 10 | 9 | {} | T | T | T | T | T | T | T | T | T |

Tabel 2. Tabel hasil penelusuran graf G dengan DFS

Dari tabel di atas, iterasi menyatakan langkah penelusuran, V menyatakan simpul yang sedang ditelusuri, *next* V menyatakan simpul berikutnya yang akan dilakukan penelusuran, *visited* menyatakan simpul mana saja yang telah ditelusuri, dan BT menyatakan *backrack* atau kembali ke simpul sebelumnya.

Penelusuran dilakukan mulai dari simpul nomor 1. Dari simpul nomor 1, ditemukan simpul tetangga berikutnya adalah nomor 2. Sealnjutnya, penelusuran dilakukan di nomor 2 dan ditemukan simpul tetangganya adalah 3, begitu seterusnya. Perlu diperhatikan pada iterasi ke 8, saat penelusuran dilakukan pada simpul 6, tidak ditemukan lagi simpul tetangganya. Oleh karena itu, akan dilakukan *backtrack* dan pencarian kembali ke simpul nomor 8. Dari simpul nomor 8, ternyata ada simpul tetangga lain selain nomor 6, yaitu nomor 9. Maka, penelusuran dilanjutkan ke nomor 9. Sehingga, urutan penelusuran dari graf G di atas dengan menggunakan algoritma DFS adalah 1 – 2 – 3 – 5 – 4 – 7 – 8 – 6 – 9.

Dalam implementasinya, struktur data algoritma DFS biasanya menggunakan struktur data rekursif atau tumpukan (*stack*). Struktur data *stack* biasanya digunakan untuk algoritma DFS yang iteratif.

Berikut adalah *pseudocode* dari algoritma DFS dalam versi rekursif dan versi *stack* yang telah dijelaskan tadi.

```
procedure DFS(G,s) // versi rekursif
visited[s] ← true
foreach simpul w bertetangga dengan s in G do
   if not visited[w] then
      DFS(G,w)
   endif
endfor
```

```
procedure DFS-for(G,s) // versi iteratif
S ← {}
PUSH(S,s)
while (S) not empty do
   POP(S,s)
   if not visited[s] then
      visited[s] ← true
      foreach simpul w bertetangga dengan s in G do
         PUSH(S,w)
      endfor
   endif
endwhile
```

## IV. IMPLEMENTASI BFS DAN DFS DALAM PERMAINAN ULAR TANGGA

Penentuan jalur optimal dalam permainan ular tangga

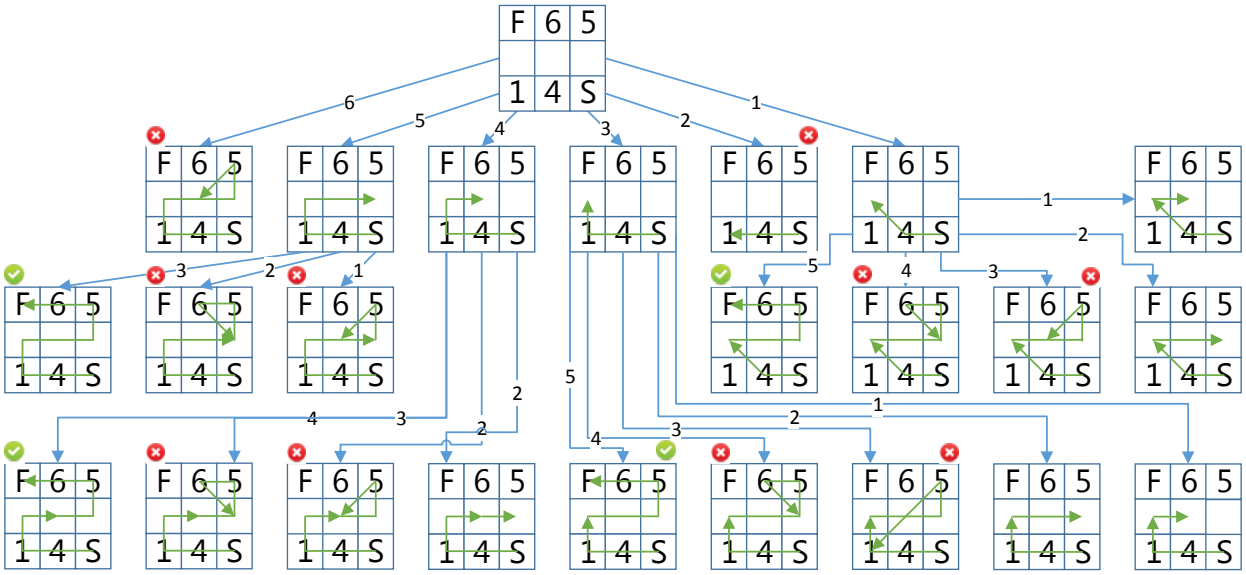
diimplementasi dengan menggunakan algoritma BFS dan DFS.

### *A. Penentuan jalur optimal permainan ular tangga dengan menggunakan algoritma BFS*

Dengan menggunakan algoritma BFS, kita dapat menentukan seluruh solusi yang ada dari kasus tersebut. Untuk mengimplementasi permainan ular tangga dengan menggunakan algoritma BFS, dibuat batasan-batasan (*constraint*). Batasan-batasan yang dibuat dalam permainan ular tangga ini adalah:

1. **Prioritas dadu**
   Algoritma BFS yang akan diimplementasi mengutamakan dadu dengan angka lebih tinggi dahulu (6 paling tinggi dan 1 paling rendah).
2. **Batas perpindahan**
   Setiap kali pemain bergerak, akan dicek apakah di papan dengan nomor hasil pergerakan terdapat ular atau tidak. Jika terdapat ular, maka jalur tersebut akan dianggap tidak optimal dan tidak akan dilanjutkan penelusurannya. Selain itu, jika saat pelemparan dadu, total pergerakan yang akan dijalankan pemain melebihi garis akhir dianggap tidak optimal dan gagal tidak akan dilanjutkan lagi penelusurannya.

Berikut adalah *pseudocode* dari BFS Solver untuk permainan ular tangga:




Gambar 4. Pohon Ruang Status BFS-Solver dari Ular Tangga

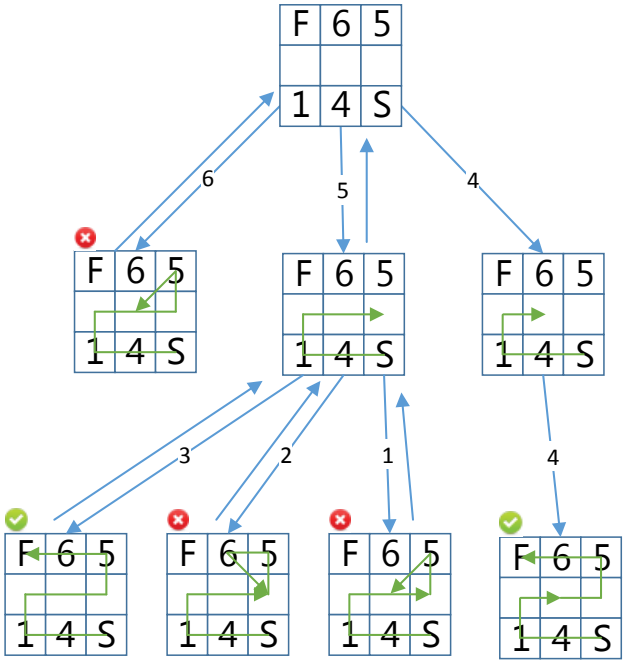
**BFS-Solver**

```
ENQUEUE(Q,s);
while (Q) not empty do {
   DEQUEUE(Q,state);
   if (not visited[state]) {
      visited[state] ← true
      if (state not snake and state
<= finish) then {
//  mengecek  apakah  di  state  ini
masuk ke dalam batasan-batasan yang
ada dan menambahkan ke dalam antrian
jawaban
         answer.add(state);
      }
   }
}
```

### *B. Penentuan jalur optimal permainan ular tangga dengan menggunakan algoritma DFS*

Tidak jauh berbeda dengan algoritma BFS, dengan algoritma DFS kita dapat menentukan seluruh solusi yang ada dari kasus tersebut. Dari sana, kita dapat menentukan solusi optimalnya. Untuk mengimplementasikannya, dibuat batasan-batasan. Berikut adalah batasan-batasan yang dibuat:

1. **Prioritas dadu**
   Sama seperti algoritma BFS, prioritas dadu yang diutamakan adalah 6 (paling besar) dan menurun hingga 1 (paling kecil).
2. **Batas perpindahan**
   Sama seperti pada algoritma BFS, setiap kali pemain bergerak, akan dicek apakah di papan dengan nomor hasil pergerakan terdapat ular atau tidak. Jika terdapat ular, maka jalur tersebut akan dianggap tidak optimal dan tidak akan dilanjutkan penelusurannya. Selain itu, jika saat pelemparan dadu, total pergerakan yang akan dijalankan pemain melebihi garis akhir dianggap tidak optimal dan gagal tidak akan dilanjutkan lagi penelusurannya.




Gambar 5. Pohon Ruang Status DFS untuk pencarian 3 jalur dalam permainan ular tangga

Berikut adalah pseudocode dari DFS-Solver untuk permainan ular tangga:

```
DFS-Solver
PUSH(S,s);
while (S) not empty do {
   POP(S,state);
   if (not visited[state]) {
      visited[state] ← true
      if (state not snake and state
<= finish) then {
//  mengecek  apakah  di  state  ini
masuk ke dalam batasan-batasan yang
ada dan menambahkan ke dalam antrian
jawaban
         answer.add(state);
         DFS-Solver(S, state);
      }
   }
}
```

## V. Pengujian dan Analisis Hasil Pengujian

Berikut adalah pengujian yang dilakukan oleh penulis terhadap algoritma BFS dan DFS. Bahasa yang digunakan oleh penulis adalah C++ dengan papan permainan berukuran 10x10.

```
Soal permainan ular tangga:

FF 13 00 08 07
03 00 00 05 00
23 09 00 17 00
20 00 00 00 02
00 18 00 00 SS
```

Gambar 6. Soal permainan ular tangga dalam C++

Berikut adalah hasil eksekusi program untuk persoalan ular tangga dengan menggunakan algoritma BFS:

```
Solusi ular tangga dengan algoritma BFS:

Ditemukan 2177 jalur penelusuran

Jalur yang optimal:
1 - 6 - 20 - 25

Pelemparan dadu yang optimal:
5 - 5
```

Gambar 6. Solusi dari BFS-Solver versi C++

Dan berikut adalah hasil eksekusi program dengan menggunakan algoritma DFS:

```
Solusi ular tangga dengan algoritma DFS:

Ditemukan 2177 jalur penelusuran

Jalur yang optimal:
1 - 6 - 20 - 25

Pelemparan dadu yang optimal:
5 - 5
```

Gambar 7. Solusi dari DFS-Solver versi C++

Kedua algoritma di atas menemukan jumlah jalur penelusuran yang sama, yaitu 2177 jalur penelusuran. Kedua algoritma ini dianggap cukup efektif untuk menemukan solusi optimal dalam permainan ular tangga, yaitu hanya dengan 2 langkah. Perbedaan waktu eksekusi antara kedua algoritma ini juga tidak jauh berbeda.

## VI. Ucapan Terima Kasih

Pertama, penulis mengucapkan terima kasih kepada Tuhan Yesus Kristus yang telah menyertai penulis sehingga berhasil menyelesaikan tugas makalah ini. Penulis juga mengucapkan terima kasih kepada Bapak dan Ibu pengajar mata kuliah IF2211 Strategi Algoritma, yaitu Bapak Ir. Rinaldi Munir dan Ibu Dr. Masayu Leylia Khodra yang telah mengajarkan banyak pelajaran kepada penulis di satu semester ini. Atas bimbingan Bapak dan Ibu, penulis akhirnya berhasil menyelesaikan mata kuliah ini. Penulis juga ingin berterima kasih atas pelajaran eksplorasi strategi algoritma yang Bapak dan Ibu berikan yang dapat penulis bawa sebagai bekal di masa depan. Tanpa tugas eksplorasi yang Bapak dan Ibu berikan,

penulis tidak dapat mendapatkan pengalaman lebih dari strategi algoritma saja. Semoga seluruh perjuangan yang telah penulis dapatkan di dalam mata kuliah ini dapat bermanfaat bagi penulis dan lingkungan sekitar.

## DAFTAR PUSTAKA


[1] http://www.tradgames.org.uk/games/Moksha-Patamu.htm diakses tanggal 17 Mei 2014 pukul 21.00

[2] Munir, Rinaldi. 2009. Diktat Kuliah Strategi Algoritma, Penerbit Informatika: Bandung.

[3] http://opendatastructures.org/versions/edition-0.1e/ods-java/12_3_Graph_Traversal.html#SECTION00153200000000000000 diakses tanggal 18 Mei 2014 pukul 10.00

[4] http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch14.html#QQ1-46-92 diakses tanggal 18 Mei 2014 pukul 13.00


## 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, 18 Mei 2014

Christ Angga Saputra – 13512019