Implementasi Algoritma BFS dan DFS dalam Penyelesaian *Token Flip Puzzle*

Ali Akbar Septiandri - 13509001[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]mail.septiandri@gmail.com, 13509001@std.stei.itb.ac.id

***Abstract*—Token flip puzzle is a game where we will be given, usually, 16 tokens, black on one side and white on the other, which are arranged on a 4 × 4 board. The objective is to turn all white in as few moves as possible by flipping tokens. The complication is that when a token is turned, the adjacent tokens directly above, below, right, and left of the flipped token are also flipped. In this paper, there are two kinds of algorithm used to help solving the problem: Breadth-First Search (BFS) and Depth-First Search (DFS). The two algorithms mentioned before are general techniques for traversing a graph. Therefore, the problem in the puzzle will be represented as a graph. The graph then will represent the solution tree for the problem.**

***Index Terms*—Breadth-First Search, Depth-First Search, Solution Tree, Token Flip Puzzle.**

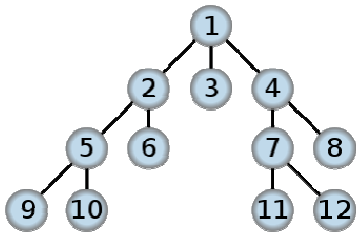
I. PENDAHULUAN

A. Algoritma Breadth-First Search

Algoritma Breadth-First Search (BFS) atau dikenal juga dengan nama algoritma pencarian melebar adalah sebuah teknik umum yang digunakan untuk melakukan traversal pada graf. Secara ringkas, algoritma ini memiliki prosedur sebagai berikut:

1. Traversal dimulai dari simpul v;
2. Kunjungi semua simpul v;
3. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu;
4. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya. [2]

Pencarian dalam algoritma BFS dilakukan secara sistematis, artinya biasanya dikunjungi dalam satu arah, misalnya dari simpul paling kiri ke simpul paling kanan. Penelusuran simpul juga dilakukan dalam satu aras terlebih dahulu sebelum mengunjungi simpul pada aras yang lebih tinggi.



Gambar 1 Pohon Pencarian pada Algoritma BFS

Dalam implementasinya, algoritma BFS memerlukan matriks ketetanggaan $A = [a_{ij}]$ yang berukuran $n \times n$, antrean q untuk menyimpan simpul yang telah dikunjungi, dan tabel *boolean dikunjungi*. *Pseudo-code* dari algoritma BFS adalah sebagai berikut:

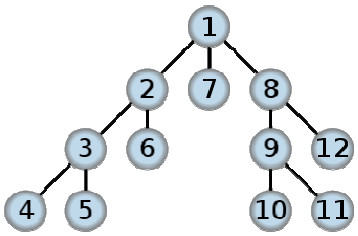
```
procedure BFS(input v:integer)
{
  Traversal graf dengan algoritma pencarian BFS.
  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang dikunjungi dicetak ke
layar
}

Deklarasi
  w : integer
  q : antrean

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

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

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

  function AntreanKosong(input q:antrean) -> boolean
  { true jika antrean q kosong, false jika sebaliknya }

Algoritma
  BuatAntrean(q) { buat antrean kosong }

  write(v) { cetak simpul awal yang dikunjungi }
  dikunjungi[v] <- true { simpul v telah dikunjungi,
        tandai dengan true }

  MasukAntrean(q,v) { masukkan simpul awal kunjungan ke
        dalam antrean }
        { kunjungi semua simpul graf selama antrean
        belum kosong }

  while not AntreanKosong(q) do
    HapusAntrean(q,v) { simpul v telah dikunjungi,
        hapus dari antrean }
    for tiap simpul w yang
        bertetangga dengan simpul v do
      if not dikunjungi[w] then
        write(w) {cetak simpul yang dikunjungi}
        MasukAntrean(q,w)
```

```
        dikunjungi[w] <- true
      endif
    endfor
  endwhile
  { AntreanKosong(q) }
```

B. Algoritma Depth-First Search

Algoritma Depth-First Search (DFS) atau dikenal juga dengan nama algoritma pencarian mendalam adalah teknik penelusuran graf yang mirip dengan algoritma BFS. Namun, algoritma DFS melakukan penelusuran dengan mengunjungi secara rekursif mendalam salah satu dari sejumlah simpul yang dibangkitkan sebelum akhirnya pindah ke anak yang lain dari simpul akarnya. Prosedur dari algoritma DFS dapat digambarkan sebagai berikut:

1. Traversal dimulai dari simpul v;
2. Kunjungi simpul v;
3. Kunjungi simpul w yang bertetangga dengan v;
4. Ulangi DFS mulai dari simpul w;
5. 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;
6. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi. [2]



Gambar 2 Pohon Pencarian pada Algoritma DFS

Pseudo-code dari algoritma DFS adalah sebagai berikut:

```
procedure DFS(input v:integer)
{
  Mengunjungi seluruh simpul graf dengan algoritma
        pencarian DFS
  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang 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
```

Jika membandingkan algoritma BFS dan DFS, keuntungan dari algoritma DFS adalah penggunaan memori yang lebih kecil, karena pada algoritma DFS tidak perlu menyimpan semua penunjuk simpul anak pada setiap aras [4]. Untuk masalah kecepatan pencarian, algoritma BFS dan DFS memiliki keuntungannya masing-masing, atau dapat dikatakan tergantung kepada spesifikasi data seperti apa yang ingin dicari dalam suatu pohon.

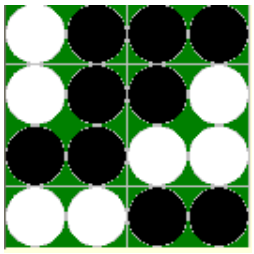
C. Penerapan BFS dan DFS pada Pencarian Solusi

Kemungkinan-kemungkinan solusi dari suatu persoalan akan membentuk ruang solusi (*solution space*). Ruang solusi tersebut diorganisasikan ke dalam struktur pohon. Pencarian solusi dilakukan dengan mengunjungi (*traversal*) simpul-simpul di dalam pohon. Pohon tersebut disebut sebagai pohon dinamis karena dibangun selama pencarian solusi berlangsung. Pohon dinamis menyatakan status-status persoalan pada saat pencarian solusi berlangsung. Beberapa terminologi penting dalam penerapan BFS dan DFS pada pencarian solusi antara lain:

1. Status persoalan (*problem state*): simpul-simpul di dalam pohon dinamis yang memenuhi kendala (*constraints*).
2. Status solusi (*solution state*): satu atau lebih status yang menyatakan solusi persoalan.
3. Status tujuan (*goal state*): status solusi yang merupakan simpul daun.
4. Ruang solusi (*solution space*): himpunan semua status solusi.
5. Ruang status (*state space*): Seluruh simpul di dalam pohon dinamis dan pohonnya dinamakan juga pohon ruang status (*state space tree*).

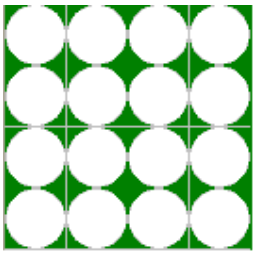
C. Token Flip Puzzle

Token Flip Puzzle merupakan permainan yang biasanya terdiri dari 16 keping koin yang disusun dalam papan 4 × 4. Tujuan dari permainannya adalah mengubah semua koin menjadi putih dengan membalikkan keping yang mungkin dengan langkah pembalikan sesedikit mungkin. Setiap membalikkan satu keping, maka masing-masing satu keping pada setiap penjuru (atas, bawah, kiri, dan kanan) dari keping yang dipilih juga harus dibalik.



Gambar 3 Contoh Soal pada Permainan Token Flip Puzzle

Contoh soal pada Gambar 3 dapat diselesaikan dengan tiga langkah, yaitu dengan membalik keping – secara berurutan – pada kolom ketiga, baris pertama; kolom kedua, baris ketiga; dan terakhir kolom ketiga, baris keempat.



Gambar 4 Kondisi Permainan Selesai

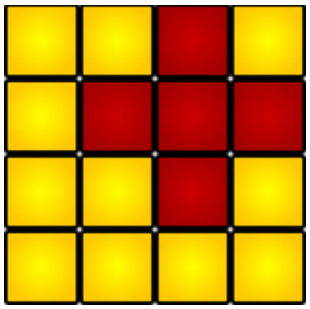
II. CONTOH PERMASALAHAN DAN PENGUJIAN

Pemecahan *Token Flip Puzzle* dapat direpresentasikan sebagai pencarian simpul solusi dari sebuah pohon yang terdiri dari simpul-simpul yang berupa suatu *state* dari papan permainan. Pohon tersebut kemudian akan dapat ditelusuri dengan algoritma BFS atau DFS sehingga dapat dicari solusi untuk persoalan. Untuk mendapatkan gambaran lebih jelas tentang masalah tersebut, saya mencoba membuat sebuah program yang dapat melakukan simulasi penyelesaian masalah dengan menggunakan metode BFS dan DFS. Masalah yang digunakan adalah *Token Flip Puzzle* untuk papan berukuran 4 × 4. *Pseudo-code* dari kedua algoritma dapat dilihat sebagai berikut:

```
procedure solveBFS(input akar: Matrix)

Deklarasi
  antrean: List of Matrix
  temp: Matrix
  i: integer
  solved: boolean

Algoritma
  i <- 0
  solved <- false

  while not solved do
    for it <- 0 to 16 do
      salin antrean_i ke temp
      balik keping pada posisi
        x = it mod 4 dan y = it div 4

      if isSolved() then
        tambahkan temp ke antrean
        solved <- true
      endif

      if temp not in antrean then
        tambahkan temp ke antrean
      endif
    endfor

    i <- i + 1 { Menunjukkan jumlah iterasi }
  endwhile
```

```
procedure solveBFS(input akar: Matrix)

Deklarasi
  antrean: List of Matrix
  temp: Matrix
  i: integer
  solved: boolean

Algoritma
  i <- 0
  solved <- false

  while not solved do
    for it <- 0 to 16 do
      salin antrean_i ke temp
      balik keping pada posisi
        x = it mod 4 dan y = it div 4

      if isSolved() then
        tambahkan temp ke antrean
        solved <- true
      endif

      if temp not in antrean then
        tambahkan temp ke antrean
      endif
    endfor

    i <- i + 1 { Menunjukkan jumlah iterasi }
  endwhile
```

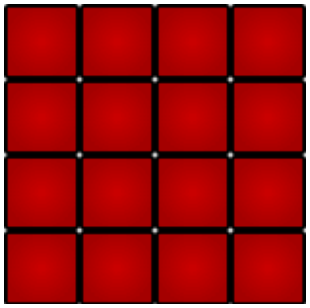
Dari hasil simulasi, didapatkan untuk kasus yang sebetulnya hanya memerlukan 1 langkah, algoritma BFS berhasil menghasilkannya, tetapi algoritma DFS memerlukan 8 langkah untuk menyelesaikannya. Contoh soal yang digunakan dapat digambarkan seperti Gambar 5.



Gambar 5 Contoh Soal 1

Warna merah pada Gambar 5 mewakili warna hitam keping, dan warna kuning pada Gambar 5 mewakili warna putih. Dari soal tersebut dapat dilihat bahwa hanya perlu dilakukan satu langkah untuk menyelesaikan masalahnya, yaitu dengan membalikkan keping pada kolom ketiga, baris kedua.

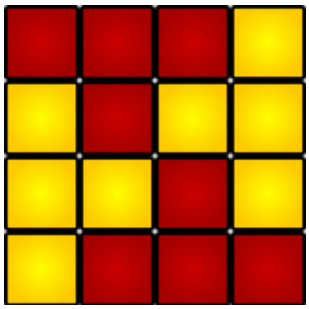
Contoh kasus kedua adalah seperti pada Gambar 6. Untuk menyelesaikan kasus tersebut, sebetulnya hanya diperlukan minimal 4 langkah dengan berbagai variasi, tetapi dengan algoritma BFS dihasilkan 298 iterasi, sedangkan algoritma DFS menghasilkan 169 iterasi.



Gambar 6 Contoh Soal 2

Dalam menyelesaikan contoh kasus seperti pada Gambar 6, salah satu variasi penyelesaian yang dapat digunakan adalah (dengan penulisan kolom-baris) 2-1, 4-2, 1-3, lalu 3-4.

Contoh kasus terakhir yang diambil adalah seperti pada Gambar 7. Pada kasus ini, sebenarnya minimal hanya 2 langkah yang dibutuhkan untuk menyelesaikannya. Algoritma BFS mendekatinya dengan cukup baik dengan hanya memerlukan 3 langkah untuk penyelesaian. Akan tetapi, algoritma DFS justru membutuhkan 169 langkah untuk penyelesaiannya. Langkah penyelesaian tercepat yang dapat diambil sebenarnya adalah (dengan penulisan kolom-baris) 1-3 lalu 4-2.



Gambar 7 Contoh Soal 3

Dari ketiga contoh kasus yang diambil di atas, kesimpulan sementara yang dapat diambil adalah sesuai dengan yang disampaikan pada bab Pendahuluan, BFS maupun DFS memiliki keuntungannya masing-masing dalam kecepatan, tergantung pada kasus yang dihadapi.

III. Analisis Hasil Pengujian

A. Kompleksitas Waktu Algoritma

Dengan asumsi setiap simpul dapat membangkitkan *b* buah simpul baru, misalkan solusi ditemukan pada aras ke-*d*, maka jumlah maksimum seluruh simpul adalah

$$1 + b + b^2 + \ldots + b^d = (b^{d+1} - 1) / (b - 1) \qquad (1)$$

Dari (1) didapatkan bahwa T(n) untuk waktu algoritma BFS adalah $T(n) = O(b^d)$, atau dapat dikatakan bahwa kompleksitas ruang algoritma BFS sama dengan kompleksitas waktunya, karena semua simpul daun dari pohon harus disimpan di dalam memori selama proses pencarian [3].

Pada algoritma DFS, kompleksitas waktu algoritma pada kasus terburuk adalah O(*bm*) karena kita hanya perlu menyimpan satu buah lintasan tunggal dari akar sampai daun, ditambah dengan simpul-simpul saudara kandungnya yang belum dikembangkan [3].

Dari hasil pengujian, dapat dilihat bahwa hasil yang didapat sangat bervariasi. Untuk kebanyakan kasus, setelah dilakukan pengujian dengan beberapa sampel lagi, dapat dilihat bahwa algoritma BFS cenderung lebih baik dalam menyelesaikan kasus jika penyelesaiannya sederhana dibandingkan dengan algoritma DFS [4]. Akan tetapi, ada juga kasus yang menunjukkan bahwa DFS dapat jauh mengungguli BFS, yaitu 328 berbanding 955 iterasi (DFS banding BFS). Dari berbagai percobaan tersebut, satu hal yang jelas terlihat adalah kedua algoritma ini masih sering terlalu jauh dari hasil minimal yang diharapkan, tetapi masih tergolong cukup baik dalam menghasilkan solusi permasalahan.

B. Analisis Pengembangan

Dari kedua algoritma, seringnya kegagalan mendapatkan langkah yang cukup singkat dalam menyelesaikan kasus dapat terjadi karena sifat kedua algoritma yang terlalu sistematis, sehingga beberapa langkah yang seharusnya menjadi singkat tidak menjadi prioritas pilihan. Untuk itu, dalam pengembangan algoritma ini, dapat digunakan metode heuristik untuk dapat mempersingkat langkah.

Salah satu metode heuristik yang dapat ditawarkan adalah dengan mencari langkah yang dapat menjadikan semua keping berwarna hitam terlebih dahulu. Berdasarkan hasil penelitian yang dilakukan dalam [8], pada kasus dalam papan 4 × 4 dengan semua keping menunjukkan sisi yang berwarna hitam, maka minimal hanya dibutuhkan empat langkah untuk penyelesaiannya.

Metode lainnya yang dapat dilakukan adalah dengan menyimpan beberapa status yang hanya memerlukan satu atau dua langkah penyelesaian, karena dalam beberapa kasus, algoritma DFS masih sering terlalu banyak memerlukan langkah penyelesaian. Hal ini dapat terjadi karena algoritma DFS tidak memperhatikan langkah-langkah singkat yang sebetulnya bisa diambil.

Khusus untuk algoritma DFS juga, pengembangan algoritma dapat dilakukan dengan memberi batas maksimum kedalaman pohon ruang status, sehingga algoritma DFS berkembang menjadi algoritma yang lebih dikenal dengan nama *Iterative Deepening Search* (IDS). Batas yang dapat diberikan untuk algoritma IDS ini antara lain adalah membatasi langkah hingga 6, karena sejauh percobaan yang saya lakukan dengan aplikasi yang telah dibuat dan disertakan dalam [8], langkah terbanyak yang diperlukan dalam menyelesaikan kasus dalam papan 4 × 4 hanya 6 langkah. Metode heuristik ini akan membantu dalam meningkatkan performa algoritma, tetapi dapat juga menjadi penghalang karena algoritma bisa berujung pada kegagalan jika ternyata memang ada kasus yang membutuhkan minimal 7 langkah dalam penyelesaiannya.

C. Alternatif Algoritma

Dengan pertimbangan bahwa algoritma BFS dan DFS tidak selalu menghasilkan solusi minimum untuk persoalan *Token Flip Puzzle* ini, mencari alternatif algoritma untuk penyelesaian kasus ini bisa jadi merupakan solusi yang baik. Referensi [8] menunjukkan bahwa dalam kasus *Token Flip Puzzle* ini ternyata langkah terpendeknya dapat dicari dengan algoritma yang menggunakan ilmu aljabar linear. Dalam pengembangannya, algoritma tersebut bahkan dapat membantu dalam menyelesaikan kasus untuk papan yang berukuran lebih dari 4 × 4.

IV. SIMPULAN

Algoritma BFS dan DFS merupakan contoh algoritma yang dapat digunakan dalam menyelesaikan masalah pencarian solusi, dalam hal ini kasus *Token Flip Puzzle*. Namun, kedua algoritma ini masih sering belum mendapatkan solusi minimum dari persoalan karena sifatnya yang sistematis. Untuk itu, dalam pengembangannya perlu ditambahkan unsur heuristik dalam penggunaan kedua algoritma ini agar lebih mangkus.

Untuk persoalan yang memiliki banyak solusi, algoritma DFS akan lebih cepat daripada BFS, karena DFS dapat menemukan solusi setelah mengeksplorasi sebagian kecil dari seluruh ruang status, sedangkan BFS harus menelusuri semua lintasan pada aras $d - 1$ sebelum memeriksa solusi pada aras d.

V. UCAPAN TERIMA KASIH

Pertama, penulis ingin mengucapkan terima kasih sebesar-besarnya kepada Bapak Rinaldi Munir selaku dosen mata kuliah [IF3051] Strategi Algoritma atas ilmu yang telah disampaikan selama ini sehingga karya ini dapat terwujud. Di samping itu, penulis juga ingin mengucapkan terima kasih kepada keluarga dan teman-teman penulis yang senantiasa memberikan semangat dalam proses penyelesaian makalah ini.

REFERENCES

[1] Rinaldi Munir, "Diktat Kuliah IF3051 Strategi Algoritma", Bandung: Penerbit ITB

[2] "BFS dan DFS"
URL: http://www.informatika.org/~rinaldi/Stmik/2006-2007/BFS%20dan%20DFS.pdf, diakses tanggal 8 Desember 2011, pukul 21.25

[3] "Penerapan BFS dan DFS pada Pencarian Solusi"
URL: http://www.informatika.org/~rinaldi/Stmik/2006-2007/Penerapan%20BFS%20dan%20DFS%20pada%20Pencarian%20Solusi.pdf, diakses tanggal 8 Desember 2011, pukul 21.25

[4] "What's the Difference Between DFS and BFS?"
URL: http://www.programmerinterview.com/index.php/data-structures/dfs-vs-bfs/, diakses tanggal 8 Desember 2011, pukul 21.26

[5] "Token Flip Puzzle"
URL: http://delphiforfun.org/programs/Flip.htm, diakses tanggal 8 Desember 2011, pukul 21.42

[6] "Token Flip – Final Chapter"
URL: http://delphiforfun.org/programs/FlipFinal.htm, diakses tanggal 8 Desember 2011, pukul 21.42

[7] "Breadth-First Search"
URL: http://ww3.algorithmdesign.net/handouts/BFS.pdf, diakses tanggal 8 Desember 2011, pukul 22.31

[8] "Depth-First Search"
URL: http://ww3.algorithmdesign.net/handouts/DFS.pdf, diakses tanggal 8 Desember 2011, pukul 22.31

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, 9 Desember 2011

Ali Akbar Septiandri - 13509001