PERBANDINGAN METODE PENCARIAN *DEPTH-FIRST SEARCH*, *BREADTH-FIRST SEARCH* DAN *BEST-FIRST SEARCH* PADA PERMAINAN *8-PUZZLE*

Oleh:
Arie S. M. Lumenta
Jurusan Teknik Elektro Fakultas Teknik UNSRAT

Abstrak

Metode pencarian yang banyak diterapkan dan digunakan pada sistem dengan kecerdasan buatan adalah *Depth-First Search*, *Breadht-First Search* dan *Best-First Search.* Dalam tulisan ini akan membandingkan penerapan ke-tiga metode pencarian tersebut diatas pada permainan *8-puzzle*. Untuk melakukan perbandingan, dibuat program 8-puzzle menggunakan bahasa pemrograman Basic yang diterapkan pada Microsoft Visual Basic

Kata kunci: *Depth-First Search*, *Breadht-First Search* dan *Best-First Search*, *8-puzzle*

1. Latar Belakang

Metoda–metoda yang banyak dipakai dalam pencarian/pelacakan adalah *depth-first search*, *breadth-first search* dan *best-first search*.

Metoda – metoda ini dapat di-implementasikan untuk pemecahan masalah *8-puzzle* (kotak delapan). Pemecahan masalah pencarian dan pelacakan dari inisial state sampai dengan goal state dapat digunakan metoda – metoda yang telah disebutkan diatas.

Dengan memahami metoda – metoda diatas, maka dapat dirancang sebuah algoritma yang kemudian digunakan untuk sebuah program. Program ini akan menjalankan proses pelacakan pemecahanan masalah kotak delapan (*8-puzzle*). Program yang akan dibangun berorientasi pada *artificial intelegence* (kecerdasan buatan), yang menggunakan metoda – metoda pencarian *depth-first search*, *breadth-first search*, dan *best-first search.*

Dalam pembuatan program ini menggunakan bahasa pemrograman basic, pada program visual basic. Dimana representasi-nya menggunakan GUI (graphical user interface).

2. Tujuan

Membandingkan penerapan metode- metode *depth-first search*, *breadth-first search* dan *best-first search*. Dalam penyelesaian solusi 8-puzzle.

3. Pencarian *Breadth-First*

Pencarian *Breadth-First* merupakan pencarian tanpa informasi (*uninformed search*) dan digolongkan *brute-force search* atau *exhaustive search*, serta pencarian buta (*blind search*)

Algoritma Pencarian *Breadth-First*

1. Berikan simpul awal pada open.
2. Loop if open n = kosong then exit (fail).
3. n:=*first* open
4. if goal (n) then exit (success).
5. Remove (n, open)
6. Add (n, closed)
7. Ekspansikan n berikan pada ekor open semua simpul anak yang belum muncul pada open atau closed dan bubuhkan pointer ke n.
8. Kembali ke loop

4. Pencarian *Depth - Fisrt*

Pencarian *Depth-First* ini merupakan pencarian tanpa informasi *(uninformed search)* dan digolongkan *brute-force search* atau *exhaustive search,* serta pencarian buta *(blind search).*

Algoritma Pencarian *Depth-First*

1. Berikan simpul awal pada open.
2. Loop if open n = kosong then exit (fail).
3. n:=*first* open
4. if goal (n) then exit (success).
5. Remove (n, open)
6. Add (n, closed)
7. Ekspansikan n berikan semua simpul anak pada kepala open dan bubuhkan pointer dari simpul anak ke n.

8. Kembali ke loop

5. *Best-First Search*

Best-first search termasuk dalam kategori pelacakan *heuristic*. Heuristik adalah suatu perbuatan Pelacakan atau pencarian *heuristic* adalah suatu metode pencarian yang berusaha memperbaiki efisiensi proses pencarian, mungkin dengan cara mengorbankan ketidaklengkapan.

Informasi *heuristic* akan membantu proses dalam proses pencarian:

1. Memutuskan simpul mana yang akan diperluas berikutnya.
2. Memperluas simpul, yaitu memutuskan penyukses mana yang akan dihasilkan dan
3. Memutuskan simpul mana yang akan dipotong dari ruang masalah.

Pada dasarnya *best-first search* merupakan kombinasi dari *breadth-first search* dan *depth-first search* yang mengadopsi kelebihan dari masing-masing pelacakan tersebut.

Algoritma *Best First Search* adalah:

1. Node terdekat dengan keadaan tujuan, sebagaimana ditentukan oleh h(n) diperluas dahulu.
2. Ciri utamanya:
   a. Cari penyelesaian secara cepat.
   b. Jangan selalu mencari penyelesaian terbaik, karena itu mengevaluasi pilihan terbaik segera, tidak pilihan jangka panjang.
   c. Dapat ke sasaran yang tidak sesuai untuk permulaan yang salah.
3. Perluas sebuah simpul yang sudah pasti simpul akhir. Mirip pencarian *depth-first* (mengikuti sebuah lintasan untuk sebuah solusi atau simpul akhir.)

6. Visual Basic

Visual Basic (dikenal dengan nama VB) adalah bahasa pemrograman berdasarkan bahasa Basic yang sudah menggunakan OOP (*Oriented Object Programing*) dan RAD (*Rapid Application Development*). Versi yang digunakan pada tugas ini ialah VB 6 yang merupakan bagian dari paket Visual Studio 6 yaitu paket pemrograman yang dikembangkan oleh Microsoft.

Seperti pada OOP dan RAD lainnya *programmer* disodorkan suatu form yang di dalamnya bisa ditambahkan berbagai macam komponen (objek) dan juga ada menu editor dimana *programmer* dapat menambahkan menu sesuai dengan kebutuhan.

7. Perancangan Program

Suatu *state* (simpul) pada teori *searching* di *software* ini dipresentasikan dengan suatu variabel *array* dengan tipe TPz. Struktur yang digunakan ialah *linked list* artinya simpul-simpul tersebut saling dihubungkan atau tiap simpul mempunyai pointer ke simpul lainnya. Sedangkan *list open* menggunakan antrian berupa variabel *array* yang isinya adalah nomor simpul dimana *array* paling kiri (index = 0) berisi simpul yang akan di-*ekspand*. Proses pemasukan simpul hasil *expand* ke dalam antrian dan urutan antrian bergantung pada algoritma yang digunakan.

Pada algoritma *Breadth First Search* simpul hasil *expand* diletakan di sebelah kanan, metode yang digunakan ialah FIFO (*First In First Out*), yang pertama masuk maka ia yang akan pertama di-*expand*. Pada algoritma *Depth First Search* metode yang digunakan ialah LIFO (*Last In First Out*). Pada algoritma *Best First Search* setiap ada tambahan pada antrian maka antrian tersebut di-*sort* (diurutkan) berdasarkan nilainya menggunakan metoda *buble* dengan nilai terkecil di sebelah kiri. Nilai yang dimasukkan pada setiap simpul ialah nilai *manhatan distance*.

Permainan ini terdiri atas 9 kotak berisi angka 1 sampai dengan 8 dan ada 1 sisa tempat yang kosong. Tempat kosong tersebut oleh program dianggap sebagai posisi 0. Setiap gerakan pada permainan kotak 8 ini dipresentaskan sebagai perubahan posisi 0. Misalnya; yang dimaksud dengan gerakan ke atas ialah posisi 0 bergerak ke atas atau bertukar tempat dengan kotak yang ada di atasnya.

Setiap *state* dinyatakan dalam variabel *array* Pz( ) dengan tipe TPz. Indeks dari variabel tersebut adalah nomor simpul. Tipe TPz ini di dalamnya terdiri atas :

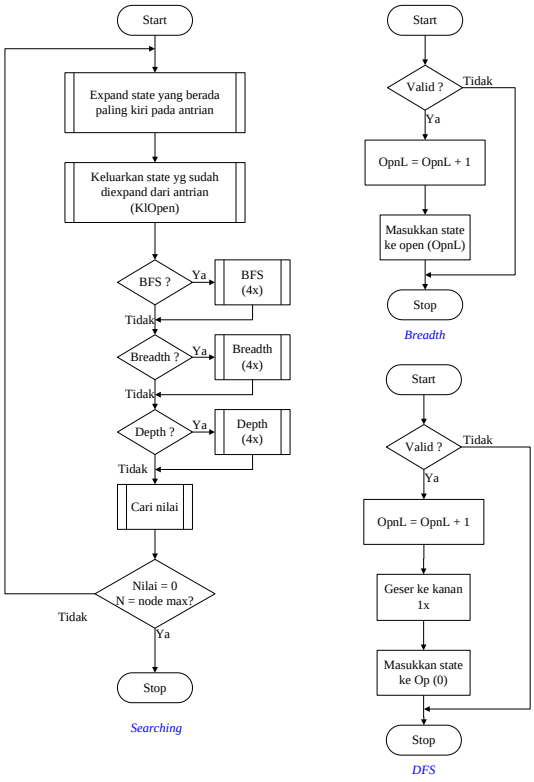
1. Isi (I, J)
   Isi (I, J) adalah suatu *array* berdimensi 2 dengan ukuran 3x3 bertipe *byte* yang menunjukkan isi dari suatu posisi, dimana I menunjukkan posisi baris dan J menunjukkan posisi kolom. Misalnya Isi (2,3) = 7, menunjukkan bahwa isi dari posisi baris ke-2 dan kolom ke-3 adalah 7.

2. ParPos
   ParPos adalah suatu variabel bertipe *byte* yang digunakan untuk menyimpan informasi gerakan sebelumnya. Informasi ini diperlukan agar dalam mengekspand atau mengacak suatu *state*, tidak akan kembali ke *state* sebelumnya. Isi dari ParPos adalah sebagai berikut :
   a. 1, bila posisi 0 sebelumnya berada di atas posisi 0 sekarang.
   b. 2, bila posisi 0 sebelumnya berada di bawah posisi 0 sekarang.
   c. 3, bila posisi 0 sebelumnya berada di kiri posisi 0 sekarang.
   d. 4, bila posisi 0 sebelumnya berada di kanan posisi 0 sekarang.
3. Pointer
   Pointer ini bertipe long, yang digunakan untuk link dengan cara menyimpan nomor simpul dari parent *state*, yaitu state darimana *current state* di-*ekspand*.
4. Nilai
   Nilai dipresentasikan dengan variabel bertipe *byte*, yang digunakan untuk mengurutkan *state* di dalam *array open* pada metoda *best first search*.

Proses *expand* suatu *state* dilakukan dengan cara menggerakan posisi 0 ke empat arah (atas, bawah, kiri, kanan). Hasil *expand* tersebut di simpan sebagai *state* baru (variabel Pz baru) dengan catatan : posisi 0 tidak menembus batas (misalnya bila posisi 0 sudah paling kiri maka bila digerakkan ke kiri ia akan menembus batas) atau posisi 0 tidak kembali ke lokasi sebelumnya (*current state* = *parent state*).
Array open seperti disebutkan dalam teori dipresentasikan dengan variabel *array* Opn( ) dimana indeksnya adalah urutan (antrian) state di array open dan isinya adalah nomor simpul. Misalnya Opn(0) = 3 dan Opn(1) = 2, berarti simpul nomor 3 (variabel Pz(3)) berada paling kiri dalam antrian dan simpul nomor 2 berada di kanannya.

8. Proses Pelacakan

Ketika program melakukan *search* maka akan dijalankan beberapa proses. Dimulai dengan memeriksa apakah current state yaitu state yang disimpan di antrian paling kiri sama dengan *goal state*, bila tidak maka *state* tersebut akan di-*expand* dan setelah di-*expand* ia akan dikeuarkan dari antrian (*array open*). Hasil *expand* dimasukkan ke dalam antrian yang cara memasukannya sesuai dengan algoritma yang digunakan. Proses ini kemudian akan diulang terus-menerus dan baru akan berhenti bila *current state* = *goal state* atau jumlah simpul sudah mencapai maksimum.
Diagram alir (*flowchart*) dari prosedur-prosedur yang terdapat pada proses pelacakan dapat dilihat pada gambar 1.



Gambar 1.Diagram Alir Proses Pelacakan 1

9. Pengujian dan Analisa

Pengujian terhadap program *8-puzzle* ini dilakukan dengan mengambil 3 buah contoh kasus. Masing-masing kasus berlaku untuk ketiga metoda pelacakan, yaitu *Best first search*, *Breadth first search* dan *Depth first search*. Setelah dilakukan pengujian akan dilakukan analisa terhadap masing-masing kasus dengan penerapan ketiga metoda *searching* yang dilakukan. Setiap hasil pengujian akan disertakan dengan proses pelacakan yang dilakukan program berdasarkan metode yang dipilih.

Setelah pengujian selesai dilakukan, maka akan dilakukan analisa terhadap masing-masing metode untuk setiap kasusnya. Analisa akan diberikan berdasarkan penilaian keberhasilan program menemukan solusi dari setiap kasus (*completeness*), banyaknya simpul yang harus di-*ekspand* oleh program sampai solusi ditemukan (*space complexity*) dan terakhir *optimality* (apakah solusi yang ditemukan adalah solusi terbaik atau bukan).

Untuk mengetahui keberhasilan pada program *8-puzzle* ini maka akan dilakukan pengujian yang akan dilakukan dengan mengambil 3 contoh kasus. Untuk setiap kasus akan dilakukan pengujian terhadap tiga metode pelacakan, yaitu *Best first search*, *Breadth first search* dan *Depth first search*. Dari setiap hasil pengujian akan dicantumkan 3 hal yang penting, yaitu : keberhasilan/kegagalan dalam menemukan solusi, jumlah simpul yang harus di-*ekspand* agar dapat menemukan solusi, dan yang terakhir adalah jumlah langkah yang harus diambil untuk menemukan solusi.

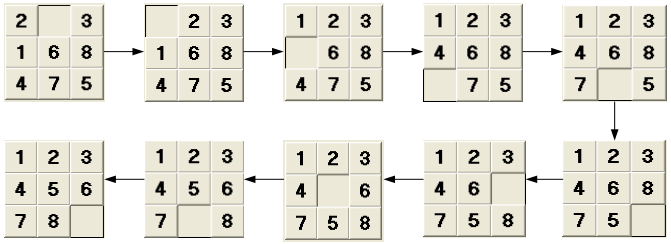
Kasus 1 :

Pengujian pada kasus 1 dilakukan dengan mengambil *initial state* dan *goal state* seperti pada gambar 2.

Gambar 2 *Initial* dan g*oal state* kasus 1

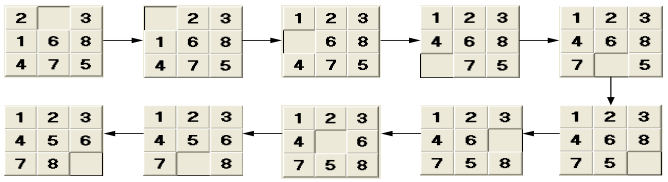
Proses pencarian dengan menggunakan metode *depth first search* pada kasus 1 ternyata tidak dapat memberikan solusi seperti yang diharapkan.

Proses pencarian dengan metode *breadth first search* **berhasil,** dengan jumlah simpul yang telah di-*ekspand* sebanyak **1219** simpul, dan jumlah langkah untuk menemukan solusi sebanyak **10 langkah**. Jalur solusi yang ditemukan adalah sebagai berikut (gambar 3):



Gambar 3 Solusi *breadth first search* kasus 1

Selanjutnya proses pencarian dengan metode *best first search* juga **berhasil** menemukan solusi, dengan jumlah simpul yang di-*ekspand* sebanyak **20** simpul dan solusi ditemukan dalam **10 langkah**. Solusi yang ditemukan dapat dilihat seperti pada gambar 4.



Gambar 4 Solusi *best-first search* kasus 1

Kasus 2 :

Pengujian akan dilakukan untuk kasus dengan *initial state* dan *goal state* yang terlihat pada gambar 5.

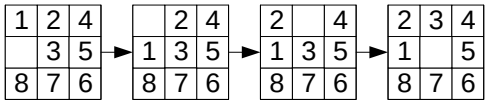
Gambar 5 *Initial* dan *goal state* kasus 2

Untuk kasus 2, proses pencarian dengan metode *depth first search* berhasil menemukan solusi dalam **4 langkah** setelah meng-*ekspand* **7** simpul.

Pelacakan dengan menggunakan metode *Breadth* pada kasus 2 ini berhasil menemukan solusi setelah meng-*ekspand* simpul sebanyak **30** simpul dan solusi ditemukan dalam **4 langkah.**

Jalur solusi yang didapatkan dari proses pelacakan dengan metode *breadth-firdt search* ternyata sama dengan solusi yang didapatkan dengan menggunakan pelacakan *best first*

search, yaitu seperti yang tampak pada gambar 6.
Pada metode *best-first search*, proses pencarian juga **berhasil** dengan jumlah simpul yang di-*ekspand* sebanyak **7** simpul dan jumlah langkah untuk menemukan solusi sebanyak **4 langkah**. Jalur solusinya adalah sebagai berikut (gambar 6):



Gambar 6. Solusi *breadth-firdt search* dan *best-first search* kasus 2

Kasus 3 :
Pengujian akan dilakukan terhadap kasus dengan *initial state* dan *goal state* sebagai berikut (Gambar 7):

Gambar 4.14 *Initial* dan *goal state* kasus 3

Pelacakan dengan menggunakan metode *depth first search* pada kasus 3 ini telah berhasil menemukan solusi dalam **16 langkah** setelah meng-*ekspand* **29** simpul.
Proses pelacakan kasus 3 dengan menggunakan metode *breadth first search* juga berhasil menemukan solusi dalam **16 langkah** dan setelah mengekspand **23718 simpul**. Jalur solusi yang ditemukan sama dengan solusi yang ditemukan dengan menggunakan pelacakan *best first search.*
Proses pencarian dengan menggunakan metode *best-first search* berhasil menemukan solusi dalam **16 langkah** setelah mengekpand simpul sebanyak **29** simpul.
Hasil analisa akan diberikan berdasarkan 3 kasus yang telah dibahas pada proses pengujian. Analisa akan dilakukan terhadap 2 hal yang penting, yaitu keberhasilan program dalam menemukan solusi (*completeness*), dan jumlah simpul yang harus di-*ekspand* agar dapat menemukan solusi (*space complexity*).

Analisa *Depth First Search*
Untuk pelacakan dengan menggunakan metode *depth first search* tidak dapat selalu menemukan solusi yang dicari. Hal ini terjadi karena metoda *depth first search* pada program ini tidak menggunakan *backtracking* (kembali ke simpul yang sebelumnya jika tidak menemukan solusi).
Solusi dari pelacakan *depth* juga belum tentu mendapatkan solusi, karena pelacakan dengan metode ini hanya mengevaluasi simpul yang berada paling kiri saja. *Space* yang dipergunakan untuk melakukan pencarian dengan menggunakan metode *depth* ini relatif kecil begitu juga waktu yang dibutuhkan.

Analisa *Breadth First Search*
Berdasarkan hasil pengujian dari ketiga kasus tersebut pelacakan dengan menggunakan metode *breadth first search* selalu dapat menghasilkan suatu solusi dari setiap permasalahan. Bila jumlah simpul yang di-*ekspand* tidak dibatasi maka pelacakan dengan menggunakan metode ini pasti akan berhasil menemukan solusi dari setiap permasalahan yang ada. Namun karena keterbatasan memori pada komputer yang digunakan maka sangat diperlukan untuk membatasi jumlah simpul yang dapat di-*ekspand*.
Ruang atau space yang dipergunakan pada metoda ini sangat besar, sebab semua simpul harus di-*ekspand* sebelum melangkah ke level berikutnya dan semua simpul yang telah di-*ekspand* tersebut harus disimpan ke dalam memori. Sedangkan waktu yang dibutuhkan dalam menemukan solusi akan sangat besar pula..

Analisa *Best First Search*
Berdasarkan hasil pengujian dari ketiga kasus yang menggunakan metode *best first search* dapat disimpulkan bahwa metode *best first search* selalu berhasil dalam menemukan solusi dalam setiap permasalahan kotak 8. Namun keberhasilan ini dimungkinkan karena dalam program tidak diperbolehkan meng-*ekspand* simpul yang merupakan parent dari simpul tersebut, atau dengan kata lain tidak diperkenankan untuk kembali ke simpul yang sebelumnya. Sedangkan pada pelacakan murni *best first search* diperbolehkan untuk kembali ke simpul yang sebelumnya telah dievaluasi.

Karena waktu yang dibutuhkan hampir sebanding dengan simpul yang dibuat dan ternyata simpul yang dibuat relatif sedikit, maka *time complexity*-nya relatif cepat.

10. Penutup

1. Pelacakan dengan menggunakan metode *best first search* adalah pelacakan yang selalu dapat menghasilkan solusi.
2. Pelacakan *breadth first search* memerlukan memori yang lebih besar dibandingkan dengan metoda yang lainnya sehingga meskipun secara teori selalu menemukan solusi tetapi bila memori komputer tidak cukup maka solusi tidak akan ditemukan.
3. Pelacakan dengan menggunakan metode *depth first search* adalah proses pelacakan yang jarang menemukan solusi karena pelacakan ini hanya mengevaluasi simpul-simpul pada satu pencabangan saja (tidak ada *back tracking*).

DAFTAR PUSTAKA

1. Microsoft. (1999). ***Mastering Visual Basic 6 Development***, Microsoft Press.
2. Microsoft. (1999). ***Mastering Visual Basic 6 Fundamentals***, Microsoft Press.
3. Russell, S., Norvig, P.(2003), ***Artificial Intelligence : A Modern Approach*** 2nd Ed., Prentice Hall.
4. Schildt, Herbert. (1987), ***Artificial Intelligence Using C***, McGrawHill.