Evaluasi dan Usaha Optimalisasi Algoritma *Depth First Search* dan *Breadth First Search* dengan Penerapan pada Aplikasi *Rat Race* dan *Web* Peta

Tjatur Kandaga, Alvin Hapendi

Jurusan Teknik Informatika,
Fakultas Teknologi informasi, Universitas Kristen Maranatha
Jl. Prof. Drg. Suria Sumantri No. 65 Bandung 40164
Email : tjatur.k@gmail.com, alvinhapendi@gmail.com

Abstract

Rat Race is a maze game. There is a rat with a main task to search an exit in the maze. The rat is provide with two blind search algorithm, that is Depth First Search and Breadth First Search. The main goal is to find an exit with minimum steps. In this application, Depth First Search has fewer steps to reach an exit than Breadth First Search algorithm. That is because Breadth First Search algorithm had so many backward steps. The blind search algortihm also can be use in another application. Web Peta is an application that use Depth First Search algorithm. The main purpose of this application are to search a route from one place to another place in map. The application also must provide alternative route and the shortest route.

Keywords : Depth First Search, Breadth First Search, Bidirectional Search, Optimalisasi

Pendahuluan

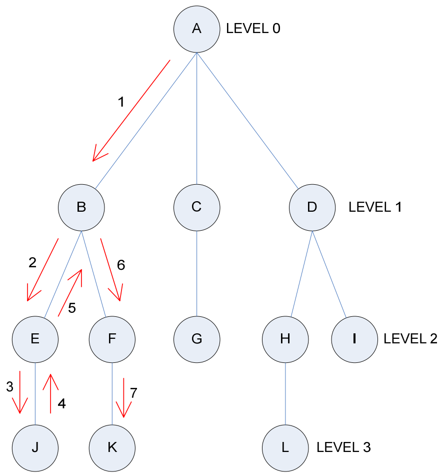
Algoritma *Depth First Search* dan *Breadth First Search* adalah algoritma pencarian buta yang digunakan dalam kecerdasan buatan. Algoritma ini berfungsi untuk menemukan tujuan pada suatu kasus dimana tidak ada informasi tambahan yang dimiliki untuk membantu melakukan pencarian. Pada pencarian buta, pencarian dilakukan dengan cara menjalani satu per satu kemungkinan yang ada. Algoritma tersebut digunakan pada aplikasi *Rat Race* dan *Web* Peta.

Rat Race adalah sebuah permainan dimana terdapat labirin dengan satu jalan masuk dan satu jalan keluar. Terdapat karakter tikus yang bertugas untuk mencari jalan keluar pada labirin. Tujuan utama dari permainan ini adalah mencari jalan keluar secepat mungkin dengan menggunakan kedua algoritma diatas. Hasilnya kemudian akan dibandingkan untuk dilihat algoritma manakah yang lebih baik untuk penerapan pada permainan ini.

Permainan ini memiliki beberapa aturan, antara lain tikus hanya dapat berjalan satu langkah demi satu langkah. Tikus tidak dapat melihat dan berjalan secara diagonal. Pada labirin hanya terdapat satu jalan masuk dan satu jalan keluar.

Aplikasi *Web* Peta juga menggunakan algoritma pencarian buta. Tetapi pada aplikasi *Web* Peta, algoritma yang digunakan adalah algoritma *Depth First Search*. Tujuan utama dari aplikasi ini adalah untuk mencari rute dari satu lokasi ke lokasi lainnya yang terdapat dalam peta. Selain itu juga aplikasi harus dapat menemukan rute alternatif dan rute terpendek untuk mencapai tujuan.

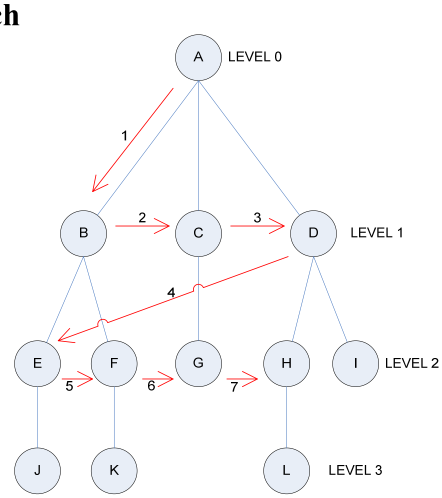
Depth First Search



Gambar 1 Pohon pencarian *Depth First Search*

Gambar diatas adalah gambar pohon pencarian *Depth First Search*. Pada gambar diatas, tiap lingkaran berabjad disebut node. Tiap node bisa memiliki anak berupa node yang lainnya. Pada pencarian dengan algoritma *Depth First Search,* pencarian dimulai dari level paling pertama (level 0) kemudian dilanjutkan ke anak paling kiri pada level berikutnya (level 1) demikian seterusnya sampai tidak terdapat anak lagi atau level yang lebih dalam lagi. Jika pencarian sudah mencapai node atau anak paling dalam maka akan dilakukan penelusuran mundur atau *backtracking* untuk melakukan pencarian ke node anak berikutnya. Pada gambar diatas, pencarian dilakukan mulai dari node A-B-E-J, pada node ini akan melakukan penelusuran mundur menuju E-B kemudian melanjutkan ke node anak berikutnya yaitu F-K. Demikian seterusnya sampai tujuan ditemukan.

Breadth First Search

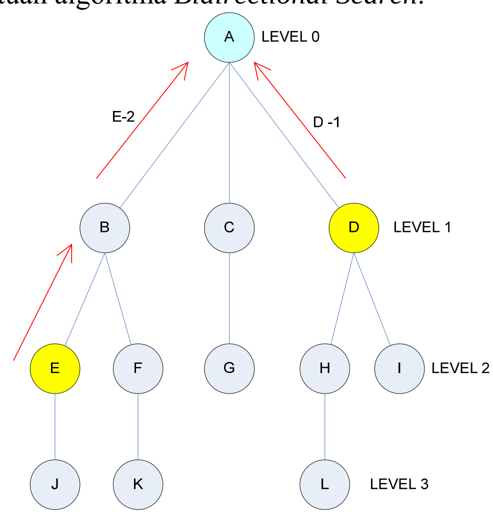


Gambar 2 Pohon pencarian *Breadth First Search*

Gambar diatas adalah gambar pohon pencarian *Breadth First Search*. Sama seperti pada pohon pencarian *Depth First Search*, tiap lingkaran berabjad disebut node. Tiap node bisa memiliki anak berupa node yang lainnya. Pada pencarian dengan algoritma *Breadth First Search,* pencarian dimulai dari level paling pertama (level

0) sampai node pada level tersebut habis. kemudian dilanjutkan ke anak paling kiri pada level berikutnya (level 1) demikian seterusnya sampai node pada level tersebut habis. Jika node pada level tersebut telah habis baru pencarian berpindah ke level berikutnya. Demikian seterusnya samapai tujuan ditemukan.

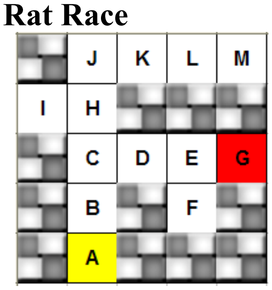
Penerapan pada aplikasi *Rat Race* membutuhkan beberapa modifikasi, terutama untuk algoritma *Breadth First Search*. Hal ini dikarenakan pada algoritma ini tidak dikenal penelusuran mundur. Untuk membantu melakukan penelusuran mundur, maka digunakan bantuan algoritma *Bidirectional Search*.



Gambar 3 Pohon pencarian *Bidirectional Search*

Gambar diatas adalah gambar pohon pencarian *Bidirectional Search*. Pada pencarian *Bidirectional Search*, pencarian dilakukan dengan mencari induk yang sama dari masing-masing posisi awal dan tujuan. Sebagai contoh, pada gambar diatas kita anggap node E sebagai posisi awal dan node D sebagai posisi tujuan. Node E akan mencari induknya yaitu node B, kemudian node B akan mencari induknya yaitu node A. Karena node A tidak memiliki induk maka pencarian induk node pada posisi awal selesai. Pada posisi tujuan node D akan mencari induknya yaitu node A. Sama seperti pada posisi awal, karena node A tidak memiliki induk maka pencarian induk selesai. Setelah itu dapat terlihat bahwa node E dan node D memiliki induk yang sama yaitu node A. Sehingga jalur untuk menuju node D ditemukan. Yaitu induk dari posisi awal B-A dan induk dari posisi akhir A, induk yang sama tidak perlu dicantumkan 2 kali sehingga jalurnya akan menjadi E-B-A-D.

Penerapan pada Aplikasi Rat Race



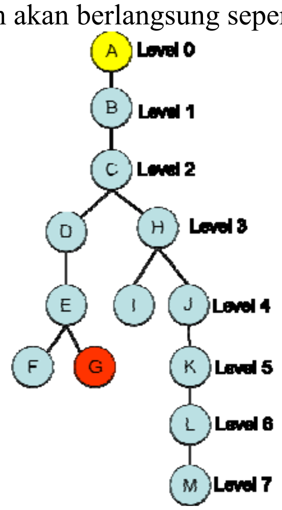
Gambar 4 Contoh Labirin *Rat Race*

Gambar diatas merupakan contoh labirin pada permainan *Rat Race*. Labirin diatas didefinisikan dalam sebuah variabel array dua dimensi bertipe data *integer*. Dengan ketentuan:

0 adalah jalan
1 adalah tembok
2 adalah posisi awal (pada gambar adalah node A)
3 adalah posisi tujuan (pada gambar adalah node G)

Pada saat mulai berjalan, tikus belum memiliki gambaran mengenai labirin. Tikus tidak tahu jalan keluar dimana, 2 langkah ke depan terdapat tembok atau jalan. Tikus hanya memiliki pengetahuan tentang yang dilihatnya pada pertama kali. Pohon *Depth First Search* ataupun *Breadth First Search* pada pertama kali pun belum terbentuk. Pada saat tikus berjalan, barulah pohon pencarian sesuai dengan algoritma mulai terbentuk. Agar tikus melakukan pencarian dengan lebih pintar maka tikus diberi kemampuan untuk mengingat. Dalam hal ini mengingat langkah dan jalan yang telah diambil ataupun dilihat.

Pada pencarian dengan algoritma *Depth First Search* dengan soal seperti gambar labirin sebelumnya, pencarian akan berlangsung seperti berikut:



Gambar 5 Pohon Pencarian *Breadth First Search* lama

- Posisi Awal A
  - Tikus melihat : B
  - Tikus berjalan ke B, posisi sekarang B
- Posisi B
  - Tikus melihat C dan A (tetapi A tidak dianggap sebagai langkah berikutnya karena tikus memiliki kemampuan mengingat).
  - Tikus berjalan ke C, posisi sekarang C
- Posisi C
  - Tikus melihat D, H dan B (B tidak dianggap karena kemampuan mengingat tikus).
  - Tikus berjalan ke D (karena D adalah node anak dari C paling kiri), posisi sekarang D
- Posisi D

    - Tikus melihat E dan C (C tidak dianggap karena kemampuan mengingat tikus).
    - Tikus berjalan ke E, posisi sekarang E
- Posisi E
    - Tikus melihat F, G dan D (D tidak dianggap karena kemampuan mengingat tikus).
    - Tikus berjalan ke G walaupun posisi node anak paling kiri dari E adalah F, karena tikus telah melihat G sebagai tujuan.

Pola diatas didapat karena tikus dianggap memiliki kebiasaan untuk selalu melihat dan berjalan dengan pola: kanan, bawah, kiri, kanan. Jadi ketika kanan ada jalan tikus akan mengangap itu adalah anak pertama, demikian seterusnya.

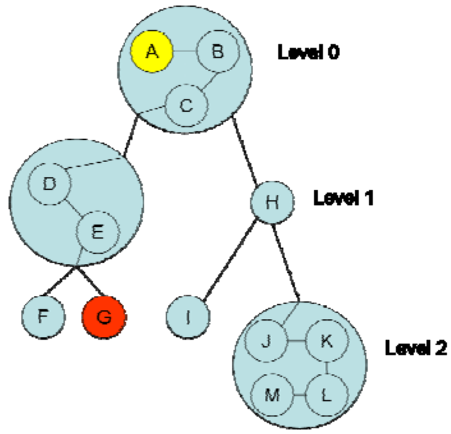
Pada pencarian dengan algoritma *Breadth First Search*, pohon pencarian mengalami modifikasi. Hal ini bertujuan agar pencarian dapat berlangsung lebih optimal.

Pencarian dengan algoritma *Breadth First Search* jika menggunakan pohon pencarian sama seperti *Depth First Search* akan menjadi:
A-B-C-D

Kemudian akan ke node H, karena node H memiliki level yang sama dengan D. Algoritma *Breadth First Seach* akan melakukan pencarian ke node yang selevel terlebih dulu. Karena tujuan berikutnya adalah H, maka tikus harus menuju C terlebih dulu kemudian berpindah ke H (*backtracking*). Untuk dapat melakukannya, dibutuhkan bantuan algoritma *bidirectional search*. Posisi awal D, tujuan H. Masing-masing akan mencari induknya sampai ditemukan induk pertama yang sama, sehingga jalurnya bisa ditemukan. Langkah tikus akan menjadi:
A-B-C-**D-C-H**

Setelah dari node H tikus harus berpindah menuju node anak berikutnya yaitu E. Untuk dapat menuju E, dibutuhkan kembali bantuan algoritma *bidirectional search*. Demikian seterusnya sampai selesai.

Dapat dilihat, jumlah langkah yang dibutuhkan oleh algoritma *Breadth First Search* menjadi sangat banyak karena banyaknya langkah kembali. Usaha perbaikan dilakukan pada pengelompokan node pada pohon pencarian. Pada pohon pencarian yang baru, level baru berganti ketika tikus menemukan cabang.



Gambar 6 Pohon pencarian *Breadth FIrst Search* baru

Sehingga pohon pencariannya akan menjadi seperti diatas. Pada pohon pencarian diatas, pada saat tikus bertemu dengan jalan yang memiliki cabang, maka level dari node berikutnya baru berubah. Pada pohon pencarian diatas langkah yang ditempuh oleh tikus adalah:
A-B-C-D-E-G (6 langkah).

Sedangkan jika menggunakan pohon pencarian yang lama, langkah yang ditempuh oleh tikus:
A-B-C-D-H-C-D-E-G (9 langkah).

Walaupun dengan kemungkinan terburuk dimana posisi tujuan terdapat pada node M, dengan perbaikan pada algoritma *Breadth First Search* maka pencarian akan menjadi:
A-B-C-D-E-D-C-H-C-D-E-F-E-G-E-D-C-H-I-H-J-K-L-M (24 langkah).

Sedangkan pada pohon pencarian yang lama, langkah yang ditempuh tikus akan menjadi:
A-B-C-D-C-H-C-D-E-D-C-H-I-H-J-H-C-D-E-F-E-G-E-D-C-H-I-H-J-K-L-M (32 langkah).

Hasil Aplikasi Rat Race

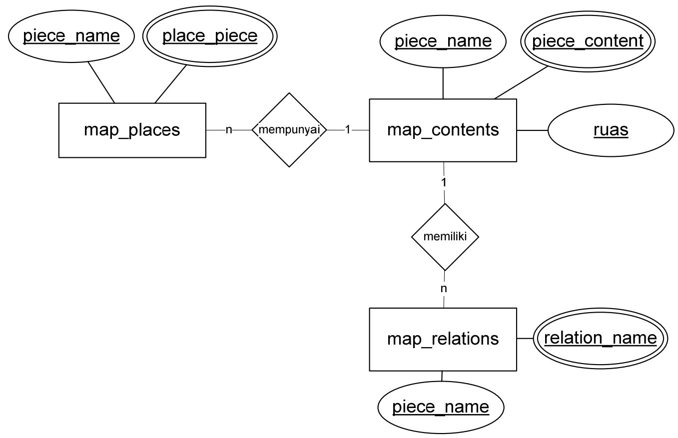
	Depth First Search lama	Depth First Search baru	Selisih	Breadth First Search lama	Breadth First Search baru	Selisih
Peta 5x5						
Jumlah Langkah	10	10	0	78	44	34
Pemakaian Memori	27762688	27049984	712704	27066368	27058176	8192
Pengecekan Tujuan	10	9	1	-	-	-
Peta 20x20						
Jumlah Langkah	88	88	0	490	210	280
Pemakaian Memori	32501760	32067584	434176	32923648	32874496	49152
Pengecekan Tujuan	88	77	11	-	-	-
Pengecekan Tujuan	88	77	11	-	-	-

Pada aplikasi *Rat Race* output yang dihasilkan menunjukan perbaikan pada fungsi *backtracking* berhasil dilakukan, hal ini terlihat dari jumlah pengecekan tujuan yang dilakukan oleh fitur yang baru lebih sedikit dibandingkan dengan fitur yang lama. Pada fitur *Breadth First Search*, output yang dihasilkan menunjukan jumlah langkah yang dibutuhkan oleh tikus untuk mencapai tujuan berkurang banyak menjadi. Pada pengujian dengan ukuran labirin 20 x 20, algoritma *Breadth First Search* yang telah diperbaiki bisa memiliki jumlah langkah kurang dari setengah algoritma yang lama. Pengunaan memori pada algoritma yang baru pun lebih sedikit dibandingkan dengan penggunaan memori pada algoritma yang lama, baik pada fitur *Depth First Search* atau *Breadth First Search*.

Penerapan pada Aplikasi *Web* Peta

Pada aplikasi *Web* Peta, algoritma yang digunakan adalah algoritma *Depth First Search*. Pada aplikasi ini tidak terdapat labirin dan tikus sebagaimana terdapat pada aplikasi *Rat Race*. Tetapi dalam aplikasi ini labirin dapat kita analogikan sebagai peta pada aplikasi ini. Soal dengan bentuk 0,1,2,3 dapat kita analogikan sebagai relasi antar gambar.

Pada aplikasi *Web* Peta, untuk dapat menemukan rute alternatif dan rute terpendek kita harus mendefinisikan relasi antara pecahan gambar yang satu dengan pecahan gambar yang lain. Karena hal itu maka labirin dalam aplikasi ini perlu untuk menggunakan *database*. *Entity Relationship Diagram* untuk aplikasi ini dapat dilihat pada gambar dibawah.

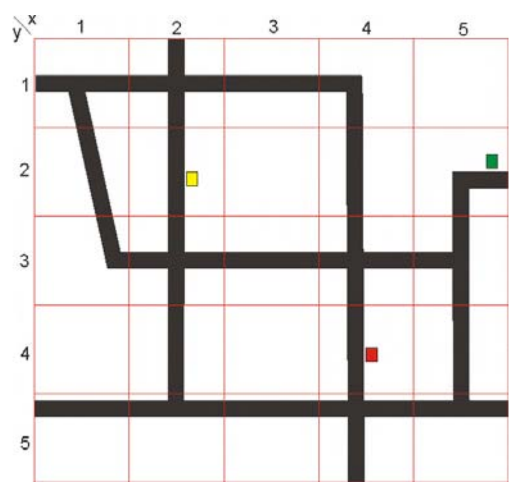


Gambar 7 *Entity Relationship Diagram Web* Peta

Pada Aplikasi *Web* Peta digunakan database *MySQL*. Pada database, terdapat 3 buah tabel. Tabel pertama adalah map_contents, didalamnya terdapat 3 buah atribut yang semuanya adalah *unique*. Fungsi dari tabel ini adalah untuk mencatat *variant* dari pecahan gambar dan mencatat ruasnya. Pencatatan ruas dibutuhkan jika dalam satu pecahan gambar terdapat lebih dari 1 jalan dan pada pecahan tersebut tidak saling berhubungan.

Tabel kedua adalah map_relations, terdiri dari 2 atribut yang keduannya juga *unique*. Tujuan dari tabel ini adalah sebagai pencatat hubungan antara satu pecahan gambar dengan pecahan gambar di sebelah kiri, kanan, atas, bawah.

Penamaan atribut pada database menggambarkan kode yang akan memudahkan aplikasi untuk menjalankan fungsinya. Peta pada aplikasi dibagi beberapa bagian dan memiliki lokasi koordinat x dan y. Sebagai contoh bisa dilihat pada gambar dibawah.



Gambar 8 Penamaan gambar

Pada gambar diatas, penamaan pecahan gambar yang akan disimpan dalam database dimulai dari posisi koordinat x diikuti tanda “-“ kemudian posisi koordinat y. Sebagai contoh adalah 1-2. 1 adalah lokasi x, 2 adalah lokasi y. Berarti pecahan gambar tersebut memiliki lokasi baris ke 1 kolom ke 2 pada peta. Untuk penamaan rute, dimulai sama dengan penamaan jalan biasa. Tambahannya adalah nomor pecahan gambar. Sebagai contoh 1-2-1. Artinya adalah pecahan gambar pertama pada baris 1 kolom 2. Contoh lainnya adalah 2-3-1, 2-3-2.

Kesimpulan

Pada hasil percobaan ditemukan bahwa algoritma *Depth First Search* memiliki keunggulan dalam jumlah langkah untuk mencapai tujuan dibandingkan dengan algoritma *Breadth First Search*. Hal ini dikarenakan banyaknya penelusuran mundur yang dilakukan oleh algoritma *Breadth First Search*.
Algoritma pencarian buta juga dapat diterapkan tidak terbatas hanya pada aplikasi *Rat Race*. Penerapan algoritma pencarian buta pada aplikasi *Web* Peta memberikan gambaran bahwa algoritma pencarian buta dapat diterapkan secara luas, termasuk dalam aplikasi *web*.

Daftar Pustaka

[Cor88] Cormen, Thomas H. dkk. *Introduction To Algorithms*

[Dia06] Diablo, Senior. 2006. *Beginners Guide to Pathfinding Algoritm*. Available from: http://ai-depot.com/Tutorial/PathFinding-Blind.html

[Ken06] Kendall, Graham. 2006. *Defining and Implementing Search*. Available from: http://www.cs.nott.ac.uk/~gxk/courses/g5aiai/003blindsearches/implement.htm

[Lug05] Luger, George F. 2005. *Artificial Intelligence Structures and Strategies for Complex Problem Solving, 4th Ed*. Pearson Education Limited.

[Rus03] Russell, Stuart J., Peter Norvig. 2003. *Artificial Intelligence a Modern Approach*. Prentice Hall.

[Wat05] Watson, Mark. 2005. *Practical Artificial Intelligence Programming in Java*. Available from: http://www.markwatson.com