Penerapan Algoritma BFS,DFS dan *Branch and Bound* pada 15-*Puzzle*-Solver

Ade Surya Ramadhani/13514049
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung Jalan Ganesha nomor 10 Bandung 40132, Indonesia
13514049@std.stei.itb.ac.id

***Abstract*—Makalah ini dibuat untuk membandingkan Algoritma BFS*(Breadth-First Search)*, DFS(*Depth-First Search*) dan Branch and Bound dalam menyelesaikan permainan 15-*puzzle*. Dimana ketiga Algoritma tersebut merupakan Algoritma yang sering dijumpai dan digunakan baik penelusuran *tree* ataupun persoalan *least cost search*. Setelah membaca dan mempelajari *paper* ini diharapkan pembaca dapat lebih memahami ketiga Algoritma ini waaupun dengan hanya satu contoh penerapan kasus.**

Keywords—BFS,DFS,Brench and Bound, 15-Puzzle Problem

I. PENDAHULUAN

Algoritma BFS dan DFS sangat umum digunakan dalam metode pencarian dalam struktur data *tree* ataupun *graph*. Pemahaman akan kedua Algoritma ini berguna untuk implementasi berbagai permasalahan yang menggunakan pencarian *tree* ataupun *graph*. Apalagi di dunia Informatika, kedua Algoritma ini lebih sering digunakan dan dijumpai. Selain kedua algoritma tersebut,terdapat algoritma yang meng-*optimize* pencarian dengan adanya suatu *bound /cost* agar melakukan pencarian dapat dipersempit.Waktu yang dibutuhkan algoritma ini juga relatif lebih cepat. Algoritma *Branch and Bound* ini juga memanfaatkan metode *backtracking* dalam pemilihan optimum solusi.Metode ini juga biasa digunakan dalam pencarian *shortest path* atau persoalan *travelinf salesmen problem* Namun apakah algoritma ini selalu lebih efisien dan lebih cepat dibandingkan Algoritma BFS dan DFS ?.

Makalah ini mengkaji dan memfokuskan dalam mengulas ketiga algorita tersebut di suatu penyelesaian kasus yaitu 15-*puzzle* problem. 15-*puzzle* problem sendiri sudah menjadi pembahasan kasus yang umum digunakan dalam mengulas keefektifan suatu algoritma atau dalam hal ini menggunakan Algortima DFS dan BFS. Mungkin sudah banyak beberapa makalah lain yang sudah membahas bagaimana cara menyelesaikan 15-*puzzle problem* namun disini penulis memfokuskan bagaimana cara menyelesaikan *problem* ini dengan menggunakan Algoritma DFS,BFS dan *Brench and Bound* sekaligus membandingkan prilaku ketiga algoritma ini ketika proses *solving* berjalan. Tujuan lainnya agar pembaca yang berkutat di dunia informatika ataupun tidak dapat memahami dimana lebih efisien diantara ketiga algoritma tersebut di satu permasalahan dan kelebihan masing-masing algoritma.

II. DASAR TEORI

A. *Algoritma BFS (Breadth-First Search)*

Algoritma BFS adaalah algoritma pencarian melebar pada graf dimana pencarian dimulai dari akar sampai ke simpul tujuan dilakukan secara horizontal dari satu node ke node yang lain.Jadi pencarian ini dilakukan dengan membangkitkan semua node *child* dari node awal lalu kemudian dicari dari node-node yang telah dibangkitkan tersebut. Konsep BFS mirip dengan struktur data antrian (*queue)* dimana simpul yang pertama kali dibangkitkan akan ditelusuri dan dibangkitkan kembali jika pencarian belum ditemukan dan dimasukkan ke antrian kembali untuk nantinya diproses.Secara umum Algoritma BFS dapat dituliskan dalam *psudo-code*

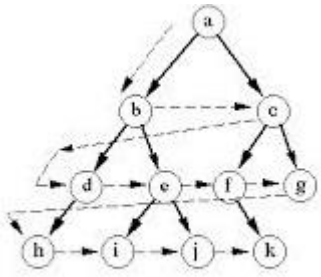
```
procedure BFS(input v:integer)
{ Traversal graf dengan algoritma pencarian BFS.

  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang dikunjungi dicetak ke layar
}
Deklarasi
    w : integer
    q : antrian;

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

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

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

    function AntrianKosong(input q:antrian) → boolean
    { true jika antrian q kosong, false jika sebaliknya }

Algoritma:
    BuatAntrian(q)          { buat antrian kosong }

    write(v)                { cetak simpul awal yang dikunjungi }
    dikunjungi[v]←true      { simpul v telah dikunjungi, tandai dengan
                              true}
    MasukAntrian(q,v)       { masukkan simpul awal kunjungan ke dalam
                              antrian}

   { kunjungi semua simpul graf selama antrian belum kosong }
    while not AntrianKosong(q) do
         HapusAntrian(q,v)    { simpul v telah dikunjungi, hapus dari
                                antrian }
         for tiap simpul w yang bertetangga dengan simpul v do
              if not dikunjungi[w] then
                   write(w)     {cetak simpul yang dikunjungi}
                   MasukAntrian(q,w)
                   dikunjungi[w]←true
              endif
         endfor
    endwhile
   { AntrianKosong(q) }
```

Sumber: Diktat Kuliah Strategi Algoritma Rinaldi Munir

Di *psudocode* diatas menunjukkan traversal pada graph statis yang tidak secara spesifik menunjukkan node tujuan (hanya mengunjungi setiap node) oleh karena itu akan berhenti ketika semua node sudah dikunjungi (antrian kosong) sedangkan jika ditambahkan kasus dimana ada node tujuan maka looping di berhentikan jika node tujuan tersebut sudah dikunjungi dan tidak perlu sampai antrian kosong.berikut langkah-langkah penyelesaian dan penggambaran pencarian simpul solusi dengan BFS :

1. Masukkan simpul akar ke antrian *Queue* jika simpul akar adalah solusi , stop.
2. Jika *Queue* kosong, tidak ada solusi ,stop.
3. Ambil simpul dari *head* dan bangkitkan anak-anaknya. Jika tidak memiliki anak kembali ke langkah 2. Jika ada tempatkan mereka di belakang antrian.
4. Jika simpul head adalah *goal* maka stop jika tidak kembali ke langkah 2.

Gambar Ilustrasi BFS



Breadth-first search

Sumber : https://janav.files.wordpress.com/2014/01/dfs-bfs-difference.jpeg , diakses 07-05-2016 pukul 15:58

Digambar tersebut mengilustrasikan bagaimana node-node tersebut dikunjungi dengan algortima BFS. Dengan sifatnya yang menyimpan setiap anak dari node maka Algortima ini memerlukan memori yang cukup banyak. Waktu yang dibutuhkan untuk sampai ke *goal* juga cukup lama. Kompleksitas algortima ini adalah exponensial yaitu $O(b^d)$ dimana b adalah simpul yang dibangkitkan dari simpul *parent* dan d adalah kedalaman/level dari solusi yang dicari. Namun kelebihan Algoritma ini jika terdapat lebih dari satu solusi akan mendapatkan solusi minimum.

B. Algoritma DFS(Depth-First Search)

Algoritma DFS adalah Algoritma pencarian mendalam dimana pencarian dimana pencarian dimulai dari akar lalu ke anak-anak yang dibangkitkan dan melakukan hal yang sama sampai simpul daun. Kemudian akan mengunjungi ke simpul anak selanjutnya dengan cara yang sama sampai simpul tujuan ditemukan. Konsep DFS mirip dengan *stack* (tumpukan) dimana simpul yang ditumpuk terakhir akan jadi yang pertama diproses. Secara umum *Pseudo-code* DFS dapat dituliskan

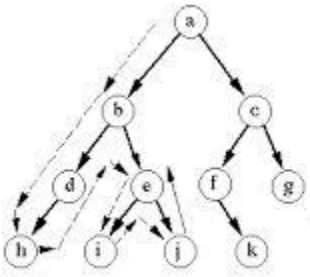
```
procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS

Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpulyang 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
```

Sumber: Diktat Kuliah Strategi Algoritma Rinaldi Munir

Di *Pseudo code* diatas menunjukkan traversal DFS di graph statis yang tidak secara spesifik memiliki simpul tujuan. Jika kasus simpul tujuan ditambahkan maka looping akan berhenti jika simpul telah ditemukan atau didapatkan tumpukan kosong. Langkah-langkah pencarian simpul solusi dengan Algoritma DFS sebagai berikut :

1. Masukkan simpul akar ke tumpukan *Stack* jika simpul akar adlah solusi , stop.
2. Jika *Stack* kosong , tidak ada solusi ,stop.
3. Ambil simpul dari kepala *Stack*. Jika simpul bukan simpul solusi dan tidak memiliki anak kembali ke langkah 2. Jika simpul solusi , stop.
4. Bangkitkan anak-anak dari simpul tersebut dan letakkan di awal *Stack*. Kembali ke langkah 3.

Gambar Ilustrasi DFS



Depth-first search

Sumber : https://janav.files.wordpress.com/2014/01/dfs-bfs-difference.jpeg , diakses 07-05-2016 pukul 15:58

Digambar mengilustrasikan bagaimana pencarian DFS di suatu *tree* . Dengan pencarian yang mendalam tentunya memiliki kekurangan jika kedalaman suatu *tree* atau *graph* yang dicari sangat dalam sehingga menghabiskan banyak waktu.Padahal dapat saja simpul solusi bisa didapat dengan cepat karena dekat dengan *root* .Namun secara umum kecepatan Algoritma ini lebih baik dibandingkan BFS dengan kompleksitas ruang O(bm) dimana b adalah anak yang dibangkitkan simpul *parent* dan m adalah kedalaman dari simpul *goal*.

C. Algortima Branch and Bound

Algoritma *Branch and Bound* atau dapat disingkat dengan B&B merupakan metode pencarian solusi dengan sistematis diimplementasikan ke dalam suatu pohon ruang

status dinamis. Algoritma ini pertama kali dipekenalkan A.H. Land dan A.G. Doig pada tahun 1960. Algoritma ini mengoptimasi permasalahan-permasalahan yang tidak dapat diselesaikan menggunakan algoritma greedy atau dynamic programming.

Meskipun cakupan persoalan yang dapat diselesaikan lebih luas, Algoritma B&B lebih lambat jika dibandingkan dengan algoritma lainnya. Pada kasus terburuk, kompleksitas waktu algoritma B&B dapat berupa eksponensial. Namun, jika diimplementasi dengan baik dan hati-hati, rata-rata algoritma ini dapat berjalan cepat. Secara iteratif Algoritma ini mirip dengan algoritma BFS. Namun ditambahkan pendekatan mematikan cabang-cabang yang tidak mengarah ke solusi. Simpul-simpul ayng dibangkitkan juga bukan semua anak namun simpul yang memenuhi kriteria tertentu. Dua proses utama di Algoritma ini adalah *branching* dan *bounding*. *Braching* adalah proses pencarian solusi optimum secara rekursif sehingga terbentuknya pohon ruang status. *Bounding* adalah proses pencarian nilai batas yang menjadi acuan untuk membangkitkan anak-anak suatu simpul.

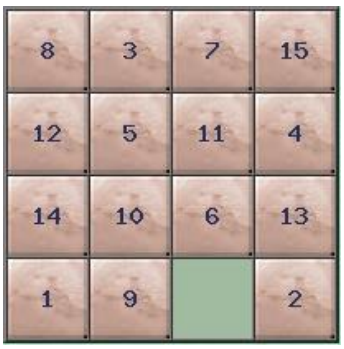
Satu persoalan dengan persoalan lainnya memiliki cara penghitungan batas bawah yang berbeda . Penghitungan ini dilakukan dengan menggunakan naluri otomatis dan tidak memiliki rumus pasti. Ketepatan dan kecepatan dari algoritma ini bergantung dengan penghitungan tersebut. Algoritma untuk mencari fungsi ini tidak bisa dibuktikan optimum atau tidaknya. Selain itu setiap persoalan juga dapat memiliki cara penghitungan batas bawah yang sama. Tantangan sebenarnya dalam merancang algoritma ini adalah menentukan batas bawah tersebut. Secara umum pecarian *cost* untuk suatu persoalan dilambangkan dengan

C^(i) = nilai batas bawah dari *node* ke i

Setiap simpul dimasukkan ke dalam antrian (*queue*). Setiap kali sebuah simpul di ekspansi maka dikeluarkan dari antrian dan proses *branching* mencapai simpul yang sudah tidak bisa di-ekspansi nilai batas bawah itu akan dibandingkan dengan nilai batas bawah simpul lainnya jika simpul tersebut merupakan *cost* paling minimum maka simpul tersebut dianggap sebagai simpul solusi. Jika tidak dan masih ada batas bawah yang lebih kecil maka simpul tersebut akan diekspansi dan proses *branching* dimulai.

D. *15-puzzle problem*

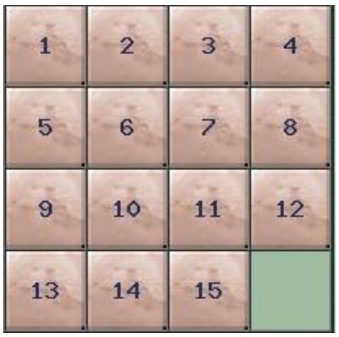
15-*puzzle* (juga sering disebut Gem puzzle,Game of 15,Mystic square) adalah sebuah permainan puzzle dimana sebuah kotak persegi yang berisi ubin-ubin yang dinomori dari 1-16 dan salah satu ubin dihilangkan.Ubin-ubin tersebut diletakkan secara random. Lalu dengan hanya bisa menggeser dari suatu ubin ke *space* yang ubinnya hilang. Suatu kondisi awal harus dapat menuju ke kondisi tujuan dengan hanya aturan pergeseran tersebut. 15-*puzzle* sendiri adalah persoalan yang lebih khusu dari N-*puzzle* dimana N+1 sendiri haruslah bilangan kuadrat sempurna. Permasalahan ini diciptakan dan diperkenalkan oleh Noyes Palmer Chapman pada awal 1874. Lalu kemudia pada tahun 1879 Johnson & Story membuktikan bagaimana permasalahn N-*puzzle* mungkin dan tidak mungkin untuk diselesaikan.



Gambar 2.1 contoh 15-*puzzle*
Summber : http://migo.sixbit.org/puzzles/fifteen/ diakses 08-05-2016 pukul 21.00

III. PENYELESAIAN 15-PUZZLE PROBLEM

Dengan menggunakan algoritma naif maka dapat diperkirakan semua kemungkinan kombinasi posisi dari setiap ubin maka ada 16! Kemungkinan posisi.Hanya setengah dari jumlah tersebut yang bisa dicapai dari state awal yang *random*. Kompleksitas tersebut digolongkan non – polynomial yang tidak realistis untuk diimplementasikan dan digunakan dalam program. Sehingga kita perlu mencari alternative lain dan beberapa diantaranya adalah Algortima BFS, DFS dan B&B. Agar tidak membingungkan maka status goal untuk masalah ini selalu dalam bentuk



Gambar 3.1 contoh *goal* status

Sumber : http://migo.sixbit.org/puzzles/fifteen/ diakses 08-05-2016 pukul 21.00

A. *Penyelesaian dengan BFS dan DFS*

Secara umum penyelesaian cara penyelesaian masalah Kedua Algoritma BFS dan DFS adalah sama.Karena Pohon ruang status yang diciptakan adalah sama. Perbedaan hanya terletak pada cara penulusuran dan penciptaan pohon dinamis. Ide penyelesaian kasus ini adalah dengan

menganggap ubin yang hilang sebagai acuan pergerakan dan membentuk simpul dari pohon. Simpul maksimum yang diciptakan simpul *parent* maksimum adalah 4 (bergerak ke atas,bawah,kanan, dan kiri) tergantung kondisi dari *parent* apakah mungkin bergeser kearah tersebut. Lalu batasan lain yang harus ditambahkan adalah state yang sudah pernah dicapai suatu *sub-tree* tidak boleh diextend untuk menghindari sirkuit dan pengulangan yang tidak pernah berhenti. Urutan penciptaan simpul juga harus konsisten (misal urutan penciptaan simpul harus atas,kanan,kiri,bawah) hal ini sangat mempengaruhi pencapaian ke suatu simpul tujuan. Berikut adalah pohon ruang status yang diciptakan kedua algoritma ini dalam penyelesaikan kasus ini di gambar 3.2 dan 3.3.

B. *Penyelesaian dengan Algoritma B&B*

Sebelum menelusiri pohon ruang status dalam pencarian simpul tujuan. Maka kita harus menentukan apakah status tujuan tersebut dapat dicapai atau tidak dari statu awal. Pertama-tama kita harus menomori posisi semua kotak penempatan ubin (bukan ubinnya) dengan nomor dari 1-16

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Lalu fungsi POSISI(i) adalah Posisi dari ubin yang bernomor i menurut matriks diatas.Contoh menurut gambar 2.1 di halaman ke 3. POSISI(1) = 13,POSISI(2) = 16, dst. Syarat suatu kondisi awal yang dapat mencapai kondisi simpul *goal* adalah jika

$$\sum_{i=1}^{16} KURANG(i) + X$$

Bernilai genap. KURANG(i) didefinisikan sebagai jumlah ubin "j" demikian sehingga j < i dan POSISI(j) > POSISI(i). contohnya menurut gambar 2.1 KURANG(1) = 0,KURANG(2) = 0, KURANG(3) = 2,..,KURANG(10) = 4,dst. Ubin kosong diberikan nomor i=16. X adalah posisi ubin kosong di awal (POSISI(16) = genap maka X = 0 dan X = 1 jika ganjil). Dengan mengetahui batasan tersebut maka simpul yang tidak dapat menuju ke simpul tujuan tidak perlu diexpansi anak-anaknya. Ide yang diterapkan di BFS dan DFS diterapkan juga disini dimana ubin kosong menjadi acuan dalam menentukan arah atau menuju suatu simpul. Setelah mengetahui batasan tersebut maka langkah selanjutnya adalah menentukan bagaimana menghitung nilai batas / *Bounding.* Hal ini dimaksudkan agar pencarian bisa dilakukan lebih cerdas karena nilai batas ini menentukan expansi yang lebih dulu dicari. Penghitungan nilai ini tidak bisa ditentukan dengan pasti yakni hanya bisa ditaksir

$$c'(P) = f(p) + g'(p)$$

f(p) adalah panjang lintasan dari simpul akar ke P (simpul yang akan diexpand) sedangkan g'(p) adalah taksiran dari P menuju simpul solusi pada sub pohon yang akaranya adalah P sendiri dengan kata lain g'(p) adalah ubin tidak kosong yang tidak terdapat di simpul tujuan. Ide dari B&B pada persoalan 15-*puzzle* ini adalah dengan metode runut balik dimana hal ini dilakukan ketika kita mengexpand anak-anak dari suatu simpul ternyata nilai *bound* yang didapat dari mereka lebih besar dari suatu simpul yang pernah diciptkan sebelumnya maka akan kembali ke simpul tersebut (*backtracking*) untuk kembali mengexpand anak-anaknya.Metode ini mirip mirip dengan konsep *priority queue* dengan komparasi berdasarkan nilai c'(P) masing-masing simpul. Dengan struktur data ini maka otomatis simpul yang memiliki nilai *bound* yang paling minimal akan berada di *head*. Penggambaran proses B&B dapat dilihat di gambar 3.4.

C. *Implementasi dalam Psudo Code*

15-*puzzle solver* using BFS

```
Procedure solveUsingBFS
 create a queue Q
 input matriks M
 mark M to Q
 init found = false
 while (not found and Q is not Empty)
  delete node in head of Q
  if (GoalCondition of node) then found = true
  else
   if(PossibleMove(up) && NOT(hadbBeenVisited)) make(up,node) add to Q
   if(PossibleMove(down) && NOT(hadbBeenVisited)) make(down,node) add to Q
   if(PossibleMove(right) && NOT(hadbBeenVisited)) make(right,node) add to Q
   if(PossibleMove(left) && NOT(hadbBeenVisited)) make(left,node) add to Q
 {end of while}
{end of procedure}
```

15-*puzzle solver* using DFS

```
Procedure solveUsingDFS
 create a Stack S
 input matriks M
 mark M to S
 init found = false
 while (not found and S is not Empty)
  pop node in head of S
  if (GoalCondition of node) then found = true
  else
   if(PossibleMove(up) && NOT(hadbBeenVisited)) make(up,node) push to S
   if(PossibleMove(down) && NOT(hadbBeenVisited)) make(down,node) push to S
   if(PossibleMove(right) && NOT(hadbBeenVisited)) make(right,node) push to S
   if(PossibleMove(left) && NOT(hadbBeenVisited)) make(left,node) push to S
 {end of while}
{end of procedure}
```

15-*puzzle solver* using B&B

```
Procedure solveUsingBandB
 create a PriorityQueue PQ
 input matriks M
 mark M to PQ
 define Compare of PQ {compare using cost of bound}
 init found = false
 if(can go to goal condition)
  while (not found and PQ is not Empty)
   pop node in head of S
   if (GoalCondition of node) then found = true
   else
    if(PossibleMove(up) && NOT(hadbBeenVisited)) make(up,node) countbound(node) add to PQ
    if(PossibleMove(down) && NOT(hadbBeenVisited)) make(down,node) countbound(node) add to PQ
    if(PossibleMove(right) && NOT(hadbBeenVisited)) make(right,node) countbound(node) add to PQ
    if(PossibleMove(left) && NOT(hadbBeenVisited)) make(left,node) countbound(node) add to PQ
  {end of while}
```

D. Eksperimen

Eksperimen yang diinput ke masing – masing algoritma adalah matriks 15-*puzzle* dengan kecocokan dengan simpul tujuan dari tidak besar , sedang dan lebih besar lagi (dihitung dengan *cost bound* simpul akar)

No Test Case	Bound Awal	Jumlah Node yang diekspansi		
		BFS	DFS	B&B
1	6	24	14	8
2	14	32	18	11
3	22	44	29	18

No Test Case	Waktu		
	BFS	DFS	B&B
1	33ms	27ms	24ms
2	51ms	36ms	30ms
3	74ms	41ms	36ms

IV. Analisis

Dari berabagai test case di percobaan maka algoritma B&B jauh unggul dalam pencarian solusi kasus 15-*puzzle*. Baik dalam hal waktu ataupun ketepatan solusi.Namun dalam hal ini kita juga harus melihat di aspek implementasinya dimana Algoritma B&B ini sangat sulit disusun dan rentan terjadi error.Jika tidak hati-hati dalam menyusun fungsi pencari nilai batas maka solusi menjadi tidak tepat atau bahkan dapat terjadi hal-hal lain yang tidak terduga seperti program yang failure.Memang algoritma BFS dan DFS juga bisa terjadi hal demikian namun persentase terjadinya hal tersebut berada di Algoritma B&B ini. Hambatan lain di kasus ini adalah pengecekan apakah state ini pernah dibangkitkan atau dikunjungi sebelumnya dan tentu hal ini memakan banyak waktu lagi. Pengecekan ini tentu saja tidak dapat dihilangkan karena menjadi pencegah terjadinya node-node yang membentuk sirkuit sehingga menyebabkan *looping* yang tidak berhenti Namun sejatinya ada beberapa testcase yang membuat algoritma B&B tidak lebih cepat dibandingkan BFS ataupun DFS yakni ketika simpul solusi hanya berda di level 1 kiri. Pencarian *cost bound* akan memperlambat perbandingan sehingga sedikit lebih banyak memakan waktu *running*. Untuk komplesitas waktu algoritma B&B bergantung pada banyaknya anak yang diekspand dan penghitungan batas bawah simpul. Kompleksitas B&B secara umum masih dalam Polinomial.

V. Kesimpulan

Algoritma Branch and Bound cocok diterapkan di permasalahan optimizing dan pencarian solusi seperti persoalan knapsack,TSP, dll. Namun implementasi yang rumit dan perancangan perhitungan penatksiran batas bawah yang juga harus hati-hati menjadi pertimbangan sendiri dalam menggunakan algoritma ini. Algoritma DFS dan BFS dapat menjadi alternative solusi untuk permasalahan yang tidak memerlukan banyak pencarian optimum seperti mencari simpul di graph dinamis ataupun problem simple lainnya.

VI. Penutup

Penulis ialah salah satu mahasiswa teknik Informatika Institut Teknologi Bandung dengan nim 13514049 yang bernama Ade Surya Ramadhani. Tujuan ditulis makalah ini adala memenuhi tugas IF2211 Strategi Algoritma tahun ajaran 2015/2016.

Penulis juga mengucapkan banyak terima kasih kepada Dr.Ir.Rinaldi Munir , M.T dan Ulfa . yang telah mengampu mata kuliah IF2211 Strategi Algoritma selama satu semester ini. Tanpa bimbingan dan arahan beliau, makalah ini tidak akan dapat diselesaikan. Penulis juga menyampaikan terima kasih kepada rekan-rekan HMIF ITB (Himpunan Mahasiswa Informatika ITB) atas kesempatan dan kerjasamanya.

Referensi

[1] Munir, Rinaldi. 2009. Diktat Kuliah Strategi Algoritma. Bandung : Program Studi Teknik Informatika Institut Teknologi Bandung. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.

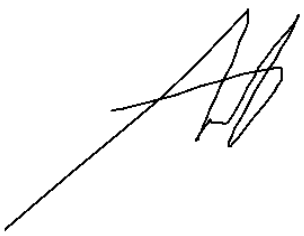
[2] http://jeuxgratuitsenligne.over-blog.com/-continuity-uses-an-analogy-to-the-n-puzzle diakses tanggal 8 Mei 2016 pukul 20.00

[3] https://www.google.co.id/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=&url=http%3A%2F%2Fwiki.cs.mtholyoke.edu%2Fmediawiki%2Fcs201%2Findex.php%3Ftitle%3DGraphs&psig=AFQjCNGUYYO1Yl461JUs2ZUGr7WJ1_UrQA&ust=1462813072367564 diakses tanggal 8 Mei 2016 pukul 20.00

VII. PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahandari makalah orang lain, dan bukan plagiasi.

Bandung, 08 Mei 2016



Ade Surya Ramadhani (13514049)