ALGORITMA PENCARIAN SIMPUL SOLUSI DALAM GRAF

Anthony Rahmat Sunaryo – NIM: 13506009
Jurusan Teknik Informatika ITB, Bandung
email : if16009@students.if.itb.ac.id

Abstract -- *Makalah ini membahas tentang analsis algoritma-algoritma pencarian solusi pada graf. Pencarian solusi pada graf berarti mencari jalan menuju sebuahsimpul solusi dalam graf. Beberapa algoritma yang telah ditemukan untuk mengatasi permasalahan ini antara lain Depth-First Search, Breadth-First Search, Iterative Deepening Depth-First Search, algoritma Dijkstra, algoritma A*, Best-First Search, dan lain-lain.*

Pembahasan dalam makalah ini dibatasi hanya graf tak berbobot. Oleh karena itu, algoritma-algoritma yang akan dianalisis dalam makalah ini hanya algoritma khusus untuk graf tak berbobot, yaitu Best-First Search, Depth-First Search, dan Iterative Deepening Depth-First Search. Analisis yang akan dilakukan adalah perkiraan kompleksitas waktu, pemakaian memori, optimalitas, dan completeness.

Graf sudah banyak diaplikasikan dalam kehidupan sehari-hari, terutama dalam bidang pemrograman komputer. Salah satu contoh aplikasi graf adalah pencarian jalan atau lintasan terpendek dari titik awal menuju titik tujuan. Dari sekian banyak algoritma yang ada, maka sangatlah penting untuk menganalisis dan membandingkan ekeftivitas dan efisiensi masing-masing algoritma sehingga dapat ditemukan algoritma yang paling baik untuk digunakan.

Kata Kunci : *Graf, simpul solusi, kompleksitas waktu, pemakaian memori, optimalitas, completeness.*

1. PENDAHULUAN

Kemudahan, kenyamanan, dan kepraktisan adalah keinginan setiap manusia dalam menjalani hidupnya. Manusia selalu ingin menyelesaikan permasalahan yang dihadapi dalam hidupnya dengan jalan yang paling mudah. Sebagai konsekuensi dari keinginan untuk membuat hidup lebih praktis, masalah baru muncul dari waktu ke waktu. Namun seiring dengan perkembangan zaman, solusi dari masalah-masalah tersebut dapat ditemukan. Tetapi ada saatnya dimana sebuah solusi dari suatu masalah terasa tidak praktis lagi untuk diterapkan.

Di lain pihak, perkembangan zaman juga tidak lepas dari perkembangan penyebaran informasi. Dalam penyebaran informasi, selalu ada pengirim, penerima dan rute jalannya informasi. Rute penyebaran informasi adalah tantangan utamanya. Terdapat banyak jalur untuk pengiriman informasi dari suatu tempat ke tempat lain, namun selalu ada rute terpendek dari seluruh kemungkinan yang ada. Di sisi lain, efisiensi rute yang ditempuh juga perlu diperhitungkan.

Sejak ditemukannya teori graf dalam ilmu matematika, telah banyak ditemukan solusi mengenai masalah rute penyebaran informasi. Dengan mengandaikan alur penyebaran informasi sebagai graf, masalah ini dapat dibuat menjadi lebih sederhana. Ditambah lagi dengan pengaplikasian teori graf ke dalam bidang pemrograman komputer, dapat diciptakan algoritma-algoritma yang dapat memecahkan masalah tersebut dalam waktu cepat. Setiap algoritma memiliki keunggulan dan kelemahan tergantung dari jenis masalah yang dihadapi. Oleh karena itu, perlu dilakukan pembelajaran lebih lanjut terhadap algoritma-algoritma tersebut.

2. TERMINOLOGI

2.1. GRAF

Dalam ilmu matematika, graf (G) didefinisikan sebagai himpunan pasangan (V, E) dimana V adalah himpunan simpul-simpul (*vertices* atau *nodes*) yang tidak boleh kosong dan E adalah himpunan sisi (*edges*) yang menghubungkan dua buah simpul. Graf dapat dinotasikan sebagai G = (V,E). Sebagai akibat dari Himpunan simpul yang tidak boleh kosong dan himpunan sisi yang boleh kosong, sebuah graf dapat hanya memiliki satu buah simpul saja tanpa sisi.

Berikut ini adalah beberapa istilah dalam graf yang harus diketahui :

a. Graf tak berarah (*undirected graph*)

Graf tak berarah adalah Graf yang sisinya tidak memiliki orientasi arah. Pada graf tak berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan. Oleh karena itu, $(v_j , v_k) = (v_k , v_j)$.

b. Graf berarah

Graf berarah adalah graf yang setiap sisinya memiliki orientasi arah. Sisi yang memiliki orientasi arah disebut busur. Oleh karena itu, (v_j , v_k) tidak sama dengan (v_k , v_j) pada graf berarah.

c. Bertetangga (*Adjacent*)

Dua buah simpul disebut bertetangga apabila kedua simpul tersebut dihubungkan langsung oleh sebuah sisi.

d. Bersisian (*Incident*)

Untuk sembarang sisi e = (v_j , v_k), sisi e dikatakan bersisian dengan simpul v_i dan v_j.

g. Lintasan (*Path*)

Lintasan yang panjangnya n dari simpul awal v_0 ke simpul akhir v_n di dalam graf G ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk v_0, e_1, v_1, e_2, …, v_{n-1}, e_n, v_n sedemikian sehingga e_1= (v_0,v_1), e_2= (v_1,v_2), …, e_n= (v_{n-1},v_n) adalah sisi-sisi dari graf G.

Panjang lintasan adalah jumlah sisi yang dilalui dalam suatu lintasan

h. Siklus atau Sirkuit

Sirkuit atau siklus adalah lintasan yang bermula dan berakhir pada simpul yang sama.

i. Graf Berbobot (*Weighted Graph*)

Graf berbobot adalah graf yang setiap sisinya diberikan sebuah bobot atau harga.

j. Terhubung

Graf tak-berarah G disebut terhubung jika utnuk setiap pasang simpul v_i dan v_j dalam himpunan V terdapat lintasan dari v_i ke v_j.

2.2. POHON

Pohon adalah graf tak berarah yag terhubung dan tidak mengandung sirkuit.

Berikut ini adalah beberapa istilah dalam graf yang harus diketahui :

a. Pohon Berakar

Pohon berakar adalah pohon yang salah satu sinpulnya dianggap sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah

b. Simpul anak (*child node*) dan simpul orangtua (*parent node*)

Jika terdapat simpul X dan Y pada sebuah pohon, simpul X dikatakan simpul orangtua jika ada sisi dari simpul X ke Y. Sebaliknya, simpul X disebut simpul anak jika ada sisi dari simpul Y ke simpul X.

c. Tingkat (*level*)

Akar memiliki *level* = 0, simpul-simpul lainnya memiliki *level* = panjang lintasan dari akar ke simpul tersebut

d. Kedalaman (*depth*)

Level maksimum dari suatu pohon disebut *depth*.

3. ALGORITMA PENCARIAN DALAM GRAF

Untuk menganalisis algoritma, perlu beberapa asumsi antara lain:

a. Jumlah child node rata-rata yang dimiliki setiap parent node dianggap sebagai *b* (setiap parent node dianggap memiliki child node sebanyak *b*)

b. Kedalaman maksimum dianggap sebagai *m*

c. Kedalaman tempat ditemukannya solusi dimisalkan sebagai *d*

3.1. ALGORITMA *BREADTH-FIRST SEARCH* (*BFS*)

Algoritma BFS adalah salah satu algoritma pencarian simpul solusi dalam sebuah graf atau pohon. Ciri khas dari algoritma ini adalah pencarian dimulai dari akar lalu dilanjutkan dengan pencarian bertahap level demi level, memeriksa seluruh node pada kedalaman tertentu sebelum masuk ke level yang lebih dalam lagi.

3.1.1. CARA KERJA ALGORITMA

Algoritma *BFS* dalam pengerjaannya memiliki beberapa keperluan sebagai berikut :

1. Sebuah struktur data matriks yang menyatakan ketetanggaan. Misalkan matriks A = [a_{ij}] berukuran n x n, maka
   A_{ij} = 1, jika simpul i dan j bertetangga
   A_{ij} = 0, jika simpul i dan j tidak bertetangga
2. Struktur data *queue* (antrian) sebagai tempat menyimpan simpul-simpul yang telah dikunjungi
3. Sebuah *array of boolean* bernama *visited* yang menyatakan apakah sebuah simpul sudah dikunjungi atau belum

Berikut ini adalah algoritma BFS dalam bentuk pseudocode:

```
function BFS (input: v, z :integer)
{I.S.v terdefinisi sebagai simpul
     Awal, z sebagai simpul tujuan
F.S. Seluruh simpul pada graf atau
     Pohon ditelusuri dengan teknik
     BFS sampai ditemukan simpul z}

Kamus Lokal
visited : array [1..n] of boolean
w, i : integer
Q : queue

procedure IsGoal
(input: v,z :integer)
{true jika v = z}
```

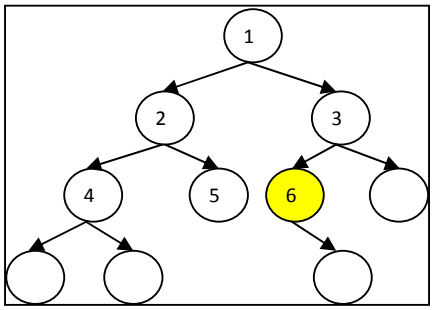
```
procedure CreateEmptyQueue
(input/output : Q : queue)
{Membuat queue Q kosong}

procedure Add
(input/output:Q:queue,      input:v:
integer)
{Memasukkan simpul v ke dalam queue
Q}

procedure Del
(input/output: Q: queue, output: v:
integer)
{Menghapus satu elemen queue Q.
Elemen yang dihapus disimpan sebagai
v}

function IsQueueEmpty
(input: Q: queue)
{Mengembalikan true jika Q kosong}
```

Algoritma

```
for i ← 1 to n do
      visited[i] = false
endfor

CreateEmptyQueue(Q)
if IsGoal(v) then
      return true
endif
visited(v) = true
Add(Q,v)

While not IsQueueEmpty(Q) do
      Del(Q,v)
      for w ← 1 to n do
if A[v,w] = 1 and
            not visited(w) then
      if IsGoal(v) then
            return true
            keluar dari BFS
            else
                  Add(Q,w)
                  visited(w) ← true
            endif
            endif
      endfor
endwhile
return false
```

3.1.2. REALISASI ALGORITMA

Berikut ini adalah gambaran cara kerja algoritma *BFS* dalam pencarian simpul solusi pada sebuah pohon. Simpul berwarna kuning adalah simpul solusi, simpul berwarna hitam adalah simpul yang sudah dikunjungi, dan simpul berwarna merah adalah simpul yang sedang dikunjungi.



Gambar 1. Langkah pengerjaan algortima *BFS*

3.1.3. ANALISIS ALGORITMA

Analisis akan dibagi ke dalam bagian-bagian sebagai berikut:

a. Kompleksitas Waktu
Kasus terburuk adalah ketika solusi berada di level maksimum.
Jika diukur dari jumlah simpul yang dikunjungi per level:
Akar : 1 simpul
Level-1 : b simpul
Level-2 : b^2 simpul
...
Level-m : b^m simpul
maka kompleksitas waktunya adalah $\max(O(1)+O(b)+O(b^2) + ... + O(b^m)) = O(b^m)$
Kompleksitas waktu algoritma *BFS* adalah eksponensial.

b. Pemakaian Memori
Karena algoritma *BFS* menggunakan *queue* untuk menyimpan simpul-simpulnya, maka pemakaian memori dapat dilihat dari jumlah simpul yang disimpan di queue pada setiap level.
Akar : terdapat b simpul di queue
Level-1 : terdapat b^2 simpul di queue
Level-2 : terdapat b^3 simpul di queue
...
Level-d : b^{d+1} simpul di queue, dimana $d < m$

Jumlah simpul yang disimpan dalam memori bersifat eksponensial.

c. *Completeness*
Algoritma BFS terjamin untuk menemukan solusi, meskipun suatu pohon atau graf memiliki kedalaman yang tak terbatas

d. Optimalitas
Setiap solusi yang ditemukan dengan algoritma BFS adalah solusi terpendek, artinya lintasan yang dilalui dari simpul awal ke simpul solusi adalah yang terpendek.
Hal ini disebabkan metode *BFS* yang mencari ada tidaknya solusi level demi level dimulai dari level teratas dilanjujtkan dengan level-level yang lebih dalam. Sebagai akibatnya, jika ditemukan

solusi, lintasan dari simpul awal ke simpul solusi tersebut adalah lintasan terpendek

3.2. ALGORITMA *DEPTH-FIRST SEARCH* (*DFS*)

Algoritma *DFS* hampir sama seperti algoritma *Breadth-First Search*, namun berbeda dalam teknik pencarian simpul solusinya. Algoritma *DFS* memiliki prioritas untuk mngunjungi simpul sampai level terdalam terlebih dahulu. Kemudian jika ditemukan jalan buntu (tidak ada lagi simpul yang bertetangga), algoritma akan memeriksa simpul sebelumnya yang sudah dikunjungi dan masih bertetangga dengan simpul lain yang belum dikunjungi dan menelusuri

3.2.1. CARA KERJA ALGORITMA

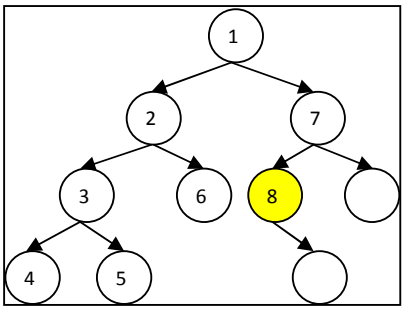
Berbeda dengan algoritma *BFS*, algoritma *DFS* tidak memerlukan struktur data tambahan seperti *queue*. Selain itu, algoritma *DFS* dapat dibuat rekursif (dapat juga dibuat non-rekursif, tetapi lebih tidak mangkus)

Berikut ini adalah algoritma *DFS* dalam *pseudocode* :

```
function DFS(input v,z:integer)
{I.S.    v terdefinisi sebagai simpul Awal, z sebagai simpul tujuan
 F.S.  Seluruh simpul pada graf atau Pohon ditelusuri dengan teknik BFS sampai   ditemukan simpul z}

Kamus Lokal
   w : integer

Algoritma
   if IsGoal(v) then
      return true
      keluar dari DFS
   endif
   visited[v]←true

   for w ← 1 to n do
      if A[v,w]=1 and
      not visited[w] then
         return DFS(w)
      endif
   endfor
   return false
```

3.2.2. REALISASI ALGORITMA

Berikut ini adalah gambaran cara kerja algoritma *DFS* dalam pencarian simpul solusi pada sebuah pohon. Simpul berwarna kuning adalah simpul solusi, angka pada simpul menyatakan urutan simpul tersebut dikunjungi.



Gambar 2. Langkah pengerjaan algortima *DFS*

3.2.3. ANALISIS ALGORITMA

Analisis akan dibagi ke dalam bagian-bagian sebagai berikut:

a. Kompleksitas Waktu
   Kasus terburuk adalah ketika solusi berada di level maksimum. Karena algoritma *DFS* selalu mengunjungi simpul pada level terdalam, maka dapat ditentukan kompleksitas waktunya adalah $O(b^m)$.

b. Pemakaian Memori
   Algoritma *DFS* tidak menggunakan struktur data tambahan, melainkan menggunakan skema rekursif untuk mengunjungi simpul-simpul yang lebih dalam.

   Karena algoritma *DFS* selalu mengunjungi simpul pada level terdalam (level *m*), maka pemakaian memori dapat diukur dengan menggunakan jumlah simpul anak rata-rata (*b*).
   Simpul anak ke-1 : *m* simpul
   Simpul anak ke-2 : *m* simpul
   ...
   Simpul anak ke-*b* : *m* simpul

   Oleh karena itu, jumlah simpul rata-rata yang dimasukkan ke dalam memori adalah sebanyak *bm* simpul, atau bersifat linier.

c. *Completeness*
   Algoritma *DFS* tidak menjamin ditemukannya solusi jika pohon atau graf memiliki level yang sangat dalam (tak terbatas). Pada kasus ini, Algoritma *DFS* ini akan membuat pencarian menjadi tersesat dan tidak menemukan solusi.

d. Optimalitas
   Algoritma *DFS* tidak terjamin menemukan solusi dengan lintasan terpendek karena algoritma *DFS* akan terus mengunjungi simpul-simpul yang lebih dalam dan belum dikunjungi

3.3. ALGORITMA *ITERATIVE DEEPENING DEPTH-FIRST SEARCH* (ITDFS)

Algoritma ITDFS pada dasarnya adalah gabungan dari algoritma BFS dan DFS. Prinsip dasar ITDFS adalah mengunjungi simpul-simpul dengan teknik DFS, namun terdapat batas kedalaman yang semakin lama semakin bertambah sampai ditemukannya simpul solusi.

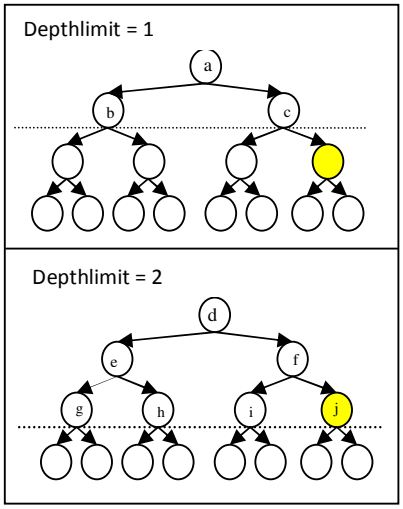
3.3.1. CARA KERJA ALGORITMA

Algoritma ITDFS tidak memerlukan struktur data tambahan, melainkan menggunakan bentuk rekursif.

Berikut ini adalah *pseudocode* algoritma ITDFS :

```
/*fungsi ini terus diulang dengan
bound (batasan level) yang terus
bertambah sampai ditemukan solusi*/

boolean dfsid(vertex v, level l,
bound)
{
      if(level > bound)
            return false;

      visited[v]=1;
      performOperation();
      for(unvisited vertices i
      adjacent to v)
            dfsid(i,level+1,bound);

}
```

3.3.2. REALISASI ALGORITMA



Gambar 3. Langkah-langkah pengerjaan algortima *ITDFS*

3.1.3. ANALISIS ALGORITMA

a. Kompleksitas Waktu

Kasus terburuk adalah ketika solusi berada di level maksimum. Jika dilihat jumlah simpul yang dikunjungi setiap depthlimit:
Depthlimit-1 : $1 + b$
Depthlimit-2 : $1 + b + b^2$
...
Depthlimit = m : $1 + b + b^2 + ... + b^m$
Maka, didapat kompleksitas waktu untuk algoritma ITDFS yaitu :
$= \max(O(m)+O(bm)+O(b^2(m-1))+...+O((1)b^m)$
$= O(b^m)$, bersifat eksponensial

b. Pemakaian Memori
Algoritma *DFS* tidak menggunakan struktur data tambahan, melainkan menggunakan skema rekursif.

Serupa dengan algoritma *DFS*, *ITDFS* selalu mengunjungi simpul pada level terdalam (level *depthlimit*), hanya saja tiap iterasi batasan level (*depthlimit*) bertambah. Jika solusi ditemukan di tingkat kedalaman d, maka saat depthlimit = d, dapat dibuat penghitungan jumlah simpul yang memakai memori sebagai berikut :
Simpul anak ke-1 : d simpul
Simpul anak ke-2 : d simpul
...
Simpul anak ke-b : d simpul.

Oleh karena itu, jumlah simpul rata-rata yang memakai memori adalah sebanyak bd simpul, atau bersifat linier.

c. *Completeness*
Algoritma *ITDFS* menjamin ditemukannya simpul solusi disebabkan oleh adanya batasan level tiap iterasi. Batasan level dapat mengatasi masalah "tersesat" pada algoritma DFS akibat tingkat kedalaman yang tidak terbatas.

d. Optimalitas
Algoritma *ITDFS* terjamin untuk mendapatkan solusi dengan lintasan terpendek karena teknik pencarian simpul dilakukan menggunakan batasan level (*depthlimit*) dimulai dari batasan terkecil sampai level maksimum. Oleh karena itu, jika ditemukan solusi, dapat dipastikan solusi tersebut terletak di level terendah.

4. PERBANDINGAN ALGORITMA *PENCARIAN SIMPUL SOLUSI DALAM GRAF*

Perbandingan akan dibagi menjadi dua bagian. Bagian pertama adalah perbandingan berdasarkan analisis, yaitu perbandingan ketiga algoritma pencarian berdasarkan analisis yang sudah dilakukan sebelumnya untuk dicari keunggulan dan kelemahan masing-masing algoritma. Bagian kedua adalah perbandingan berdasarkan studi kasus untuk membandingkan performa setiap algoritma untuk

kasus-kasus yang diberikan.

4.1. PERBANDINGAN ALGORITMA BERDASARKAN ANALISIS

	Breadth-First Search	*Depth-First Search*	*Iterative Deepening Depth-First Search*
Kompleksitas Waktu	eksponensial	eksponensial	eksponensial
Pemakaian Memori	tidak efisien	Efisien	efisien
Completeness	ya	Tidak	Ya
Optimalitas	ya	Tidak	tidak

Tabel 1. Perbandingan algoritma pencarian pada graf tak berbobot

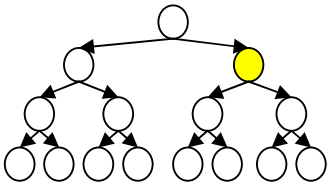
	Breadth-First Search	*Depth-First Search*	*Iterative Deepening Depth-First Search*
Keunggulan	Solusi pasti dapat ditemukan Lintasan menuju solusi adalah lintasan terpendek	Pemakaian memori efisien	Pemakaian memori efisien Solusi pasti dapat ditemukan Lintasan menuju solusi adalah yang terpendek
Kelemahan	Pemakaian memori tidak efisien	Belum tentu akan mendapatkan solusi Lintasan menuju solusi belum tentu yang terpendek	Metode pengunjungan simpul mubazir jika level maksimum sangat dalam

Tabel 2. Perbandingan keunggulan dan kelemahan algoritma pencarian pada graf tak berbobot

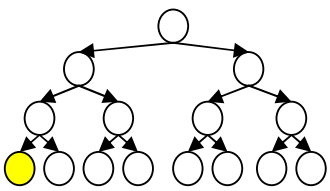
4.2. PERBANDINGAN ALGORITMA DENGAN STUDI KASUS

Untuk menguji performa algoritma *BFS*, *DFS*, *ITDFS* akan digunakan contoh-contoh kasus. Faktor yang akan dilihat pada penentuan performa adalah jumlah simpul (*node*) yang dikunjungi sampai simpul solusi didapatkan.

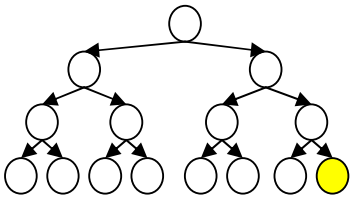
Contoh kasus akan dibagi menjadi dua bagian:
a. Kasus mudah



b. Kasus sulit-1

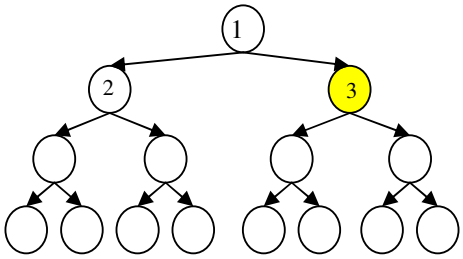


d. Kasus sulit-2

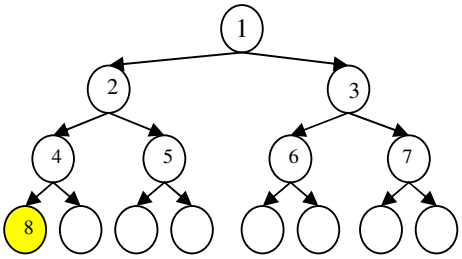


Bilangan yang terletak di dalam simpul menyatakan urutan simpul tersebut dikunjungi. Simpul berwarna kuning adalah simpul tujuan atau solusi.

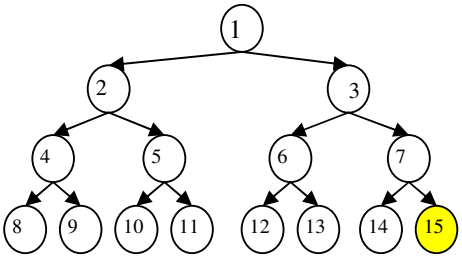
4.2.1. STUDI KASUS ALGORITMA *BFS*



Solusi kasus mudah dengan algoritma *BFS*



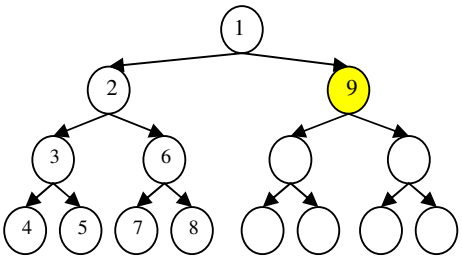
Solusi kasus sulit-1 dengan algoritma *BFS*



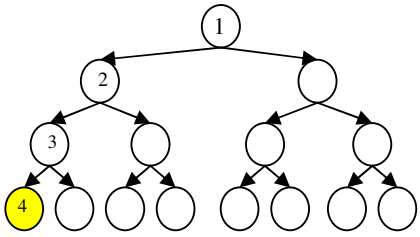
Solusi kasus sulit-2 dengan algoritma *BFS*

Gambar 4. Solusi dengan algoritma BFS

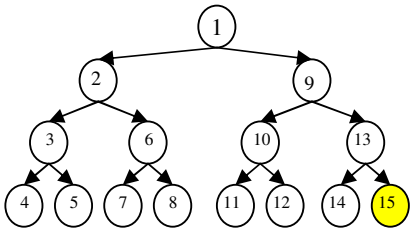
4.2.2. STUDI KASUS ALGORITMA DFS



Solusi kasus mudah dengan algoritma D*FS*



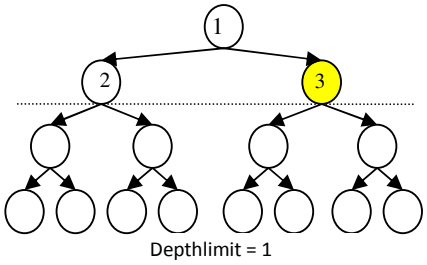
Solusi kasus sulit-1 dengan algoritma *DFS*



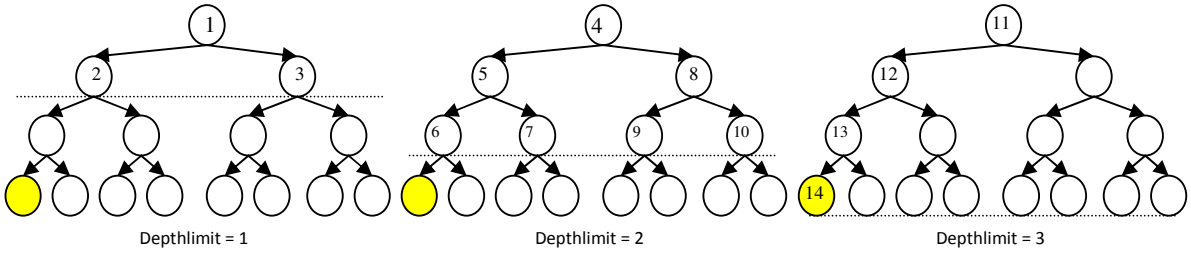
Solusi kasus sulit-2 dengan algoritma *DFS*

Gambar 5. Solusi dengan algoritma DFS

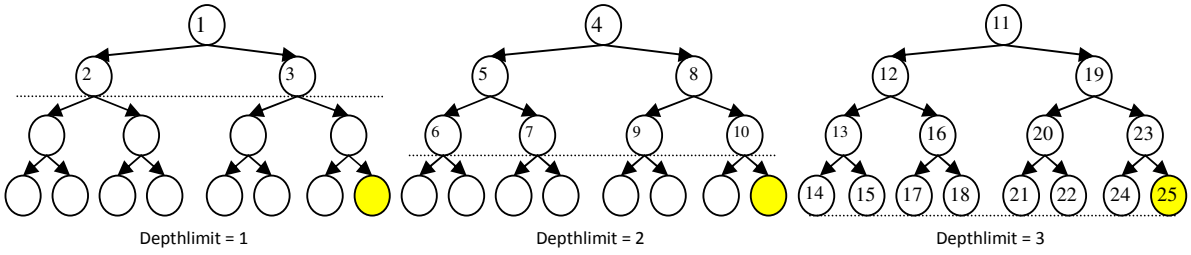
4.2.3. STUDI KASUS ALGORITMA *ITDFS*



Solusi kasus mudah dengan algoritma *ITDFS*



Solusi kasus sulit-1 dengan algoritma *ITDFS*



Solusi kasus sulit-2 dengan algoritma *ITDFS*

Gambar 5. Solusi dengan algoritma ITDFS

4.2.4. HASIL PERBANDINGAN DENGAN STUDI KASUS

Kasus	*BFS*	*DFS*	*ITDFS*
Mudah	3 simpul	9 simpul	3 simpul
sulit-1	8 simpul	4 simpul	14 simpul
sulit-2	15 simpul	15 simpul	25 simpul

Tabel 3. Perbandingan jumlah simpul dari setiap solusi

Dari hasil perbandingan didapatkan bahwa setiap algoritma memiliki performa yang berbeda-beda tergantung dari permasalahan yang dihadapi.

Algoritma BFS sekilas terlihat efisien untuk semua kasus, tetapi karena algoritma ini memerlukan struktur data tambahan untuk menyimpan simpul-simpul yang dikunjungi, maka pemakaian memori menjadi sangat besar

Algoritma DFS memang terlihat unggul pada kasus sulit-1. Hal tersebut disebabkan oleh simpul solusi yang terletak di level terdalam dan pada simpul anak paling kiri. Namun, algoritma *DFS* sangat jelas terlihat kekurangannya pada kasus mudah-2, dimana letak simpul tujuan hanya bertetangga dengan simpul awal. Jika kasus mudah-2 ini memiliki kedalaman tak terhingga, algoritma DFS tidak dapat menemukan solusi.

Algoritma *ITDFS* sangat terlihat tidak efisien pada kasus sulit-2. Karena simpul solusi terletak pada level terdalam dan pada simpul anak paling kanan, terjadi ketidakmangkusan pencarian menggunakan *ITDFS*. Namun jika kedalaman pohon dibuat menjadi tak terhingga, algoritma ini masih tetap dapat mencari solusi karena adanya *depthlimit* yang mencegah terjadinya "tersesat" seperti yang dialami *DFS*.

5. KESIMPULAN

Dari tiga macam algoritma pencarian graf yaitu *Breadth-First Search*, *Depth-First Search*, dan *Iterative Deepening Depth-First Search*, masing-masing memiliki kelebihan dan kekurangan sendiri-sendiri. Selain itu, ketiga algoritma tersebut memiliki performa yang berbeda-beda tergantung dari bentuk pohon atau grafnya.

Algoritma *ITDFS* adalah algoritma yang lebih sering digunakan daripada *BFS* dan *DFS*. Algoritma *BFS* bermasalah dengan pemakaian memori. Di lain pihak, algoritma *DFS* bermasalah dengan graf atau pohon dengan kedalaman tak terbatas sehingga tidak dapat menemukan solusi.

DAFTAR REFERENSI

[1] Munir, Rinaldi. 2003. *Matematika Diskrit*. Departemen Teknik Informatika, Institut Teknologi Bandung.

[2] Liem, Inggriani. 2001. *Diktat Struktur Data*. Departemen Teknik Informatika, Institut Teknologi Bandung.

[3] *Breadth-First Search*. http://en.wikipedia.org/wiki/Breadth-first_search. Tanggal akses 24 Desember 2007 pukul 15.00 WIB.

[4] Munir, Rinaldi. 2004. *Bahan kuliah ke-7 strategi algoritmik, Algoritma traversal dalam graf.* Departemen Teknik Informatika, Institut Teknologi Bandung.

[5] Wikipedia. *Breadth-First Search*. http://en.wikipedia.org/wiki/Breadth-first_search. Tanggal akses 24 Desember 2007 pukul 15.10 WIB.

[6] Manurung, Ruli. *Bahan kuliah 4 Sistem Cerdas, Uninformed Search Strategies*. Fakultas Ilmu Komputer, Universitas Indonesia.