Analisis Perbandingan Algoritma Rekursif dan Non-Rekursif secara DFS (Depth First Search) dengan Memanfaatkan Graf

Bayu Adi Persada 13505043

Program Studi Teknik Informatika Institut Teknologi Bandung
Jalan Ganesha No 10, Bandung, Indonesia
persada_adi@yahoo.com, if15043@students.if.itb.ac.id

ABSTRAK

DFS (Depth First Search) merupakan salah satu metode untuk penelusuran pencarain solusi persoalan dalam graf dengan memprioritaskan kedalaman graf tersebut. Terdapat dua buah cara pengaplikasian DFS, yaitu dengan cara rekursif dan non-rekursif. Perbedaan utama penggunaan cara rekursif dengan non-rekursif adalah dalam pencarian rekursif, penyelesaiannya dilakukan dengan memanggil fungsi atau prosedur di atasnya, sedangkan untuk pencarian non-rekursif, penyelesaiannya memanfaatkan *stack*.

Perbandingan ini akan didasarkan kepada kompleksitas beserta kelebihan dan kekurangan pada masing-masing algoritma tersebut. Perbandingan inilah yang nantinya menjadi dasar untuk memberikan kesimpulan algoritma mana yang cocok untuk masalah dan dalam komdisi tertentu.

Makalah ini mudah-mudahan memberikan sedikit pemikiran tentang analisa perbandingan penyelesaian suatu masalah dengan penelusuran DFS pada graf dengan menggunakan algoritma rekursif dan non-rekursif. Perbandingan ini akan didasarkan kepada kompleksitas kedua algoritma tersebut.

Kata kunci: DFS, algoritma, graf, rekursif , non-rekursif, *stack*

1. PENDAHULUAN

Graf merupakan kumpulan simpul-simpul yang dihubungkan dengan busur-busur. Setiap busur dihubungkan dengan tepat dua simpul. Persoalan dalam kehidupan sehari-hari banyak yang dapat diimplementasikan oleh graf.

Graf merupakan model matematika yang sangat rumit dan kompleks, tapi bias menjadi solusi yang mangkus dan tepat untuk sebuah permasalahan tertentu. Oleh karena itu, representasi dari suatu graf bergantung dari sifat data dan operasi yang akan dilakukan terhadap sekumpulan data tersebut pada kasus atau permasalahan tertentu.

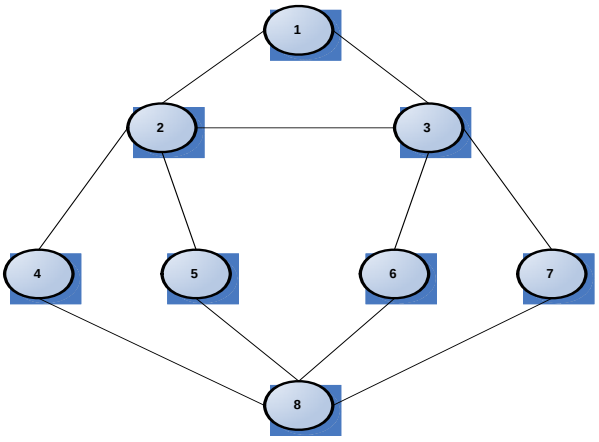
Terdapat banyak operasi yang dapat dilakukan pada graf, diantaranya operasi pembentukan graf , operasi untuk mengirimkan besaran graf, operasi penelusuran graf, operasi penelusuran jalur terpendek, pembuatan pohon merentang minimum, dan sebagainya.

Algoritma traversal pada graf ada 2 buah, salah satunya adalah DFS yang akan dibahas dalam makalah ini. DFS diterapkan pada graf statis, yaitu graf yang sudah tersedia atau dibentuk terlebih dahulu sebelum pencarian solusi.

Makalah yang saya buat ini bertujuan untuk membandingkan kompleksitas algoritma rekursif dan non-rekursif pada penyelesaian masalah dengan memanfaatkan penelusuran graf secara DFS. Saya berharap makalah ini akan memberikan kesimpulan algoritma yang lebih efisien, rekursif atau non-rekursif.

2. REPRESENTASI MASALAH

Misal contoh graf-nya adalah :

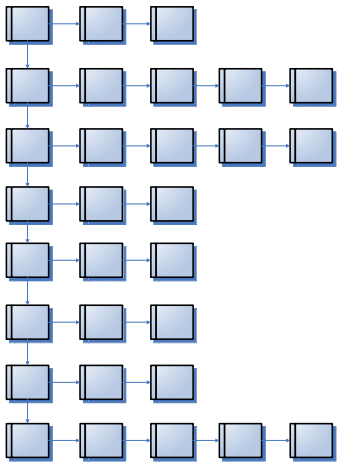


Gambar 1. Contoh Graf dengan 8 Simpul

Representasi Graf dengan matriks Ketetanggaan

	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	0	1	1	0
4	0	1	0	0	0	0	0	1
5	0	1	0	0	0	0	0	1
6	0	0	1	0	0	0	0	1
7	0	0	1	0	0	0	0	1
8	0	0	0	1	1	1	1	0

Representasi Graf dengan List Ketetanggaan



3. ALGORITMA REKURSIF

Algoritma rekursif adalah algoritma yang memanggil dirinya sendiri. Maka algoritma rekursif selalu disusun oleh 2 bagian yaitu *basis* dan *rekurens*. *Basis* merupakan keadaan penghenti dari pemanggilan sendiri algoritma itu sedangkan *rekurens* merupakan pemanggilan algoritma itu sendiri.

Algoritma rekursifnya adalah :

Procedure DFS_Rekursif (input V : char)
{ I.S: Terdefinisi A yaitu matriks ketetanggan dari sebuah graf V adalah simpul yang akan diproses
F.S: Simpul V telah di proses
}

Kamus
W : char

Procedure Proses (input V : char)
{ I.S: V adalah nomor dari simpul yang akan Diproses
F.S: simpul V telah diproses
}

Algoritma
Proses (V)
Visited [V] ← true
for i ←1 **to** N **do**
if (A[V,W] =1) **and** (**not** visited [W])
then DFS_Rekursif (W)
endif
endfor

Penelusuran graf dengan DFS rekursif direpresentasikan dengan tabel sebagai berikut:

Tingkat prosedur dalam graf	Pemanggilan prosedur	Isi simpul	Karakter tercetak
Utama	Output (info (A))	A	A
Utama	DFS_rekursif (left(A))	B	
1	Output (info (B))	B	B
1	DFS_rekursif (left (B))	D	
2	Output (info (D))	D	D
2	DFS_rekursif (left (D))	H	
2	Output (info (H))	H	H
2	DFS_rekursif (left (H))	Nil	
	Kembali ke tingkat 1		
1	DFS_rekursif (right (B))	E	
2	Output (info (E))	E	E
2	DFS_rekursif (left(E))	H	
3	Output (info (H))	H	H

3	DFS_rekursif (left (H))	Nil	
	Kembali ke tingkat 1		
1	DFS_rekursif (right (B))	C	
1	Output (info (C))	C	C
1	DFS_rekursif (left(C))	F	
2	Output (info (F))	F	F
2	DFS_rekursif (left (F))	H	
3	Output (info (H))	H	H
3	DFS_rekursif (left (H))	Nil	
	Kembali ke tingkat utama		
Utama	DFS_rekursif (right (A))	C	
1	Output (info (C))	C	C
1	DFS_rekursif (left (C))	B	
1	Output (info (B))	B	B
1	DFS_rekursif (left (B))	D	
2	Output (info (D))	D	D
2	DFS_rekursif (left (D))	H	
3	Output (info (H))	H	H
3	DFS_rekursif (left (H))	Nil	
	Kembali ke tingkat 1		
1	DFS_rekursif (left (C))	F	
2	Output (info (F))	F	F
2	DFS_rekursif (left (F))	H	
3	Output (info (H))	H	H
3	DFS_rekursif (left (H))	Nil	
	Kembali ke tingkat 1		
1	DFS_rekursif (right (C))	G	
2	Output (info (G))	G	G
2	DFS_rekursif (left(G))	H	
3	Output (info (H))	H	H
3	DFS_rekursif (left (H))	Nil	

Tabel 1. Penelusuran DFS dengan Algoritma Rekursif

Analisis Ruang :

Terlihat pada algoritma penelusuran graf secara rekursif dengan DFS, tidak terlihat unsur *stack* sama sekali yang berbeda dengan algoritma non-rekursif. Hal ini disebabkan karena *stack* bersifat implisit, artinya tidak ditampilkan tetapi sebenarnya dikelola oleh *compiler*. *Compiler* sendiri membatasi besaran penggunaan tumpukan *stack* itu sendiri.

Jika pemanggilan rekursif tersebut melebihi batas, maka *compiler* sendiri yang tidak mampu menampung kelebihan tersebut, sehingga terjadi *stack overflow*. Setiap terjadi pemanggilan rekursif, memori untuk tumpukan elemen yang baru dialokasikan untuk menyimpan alamat kembali peubah lokal dan parameter prosedur dan fungsi.

4. ALGORITMA NON-REKURSIF

Penelusuran DFS pada graf dapat juga diselesaikan dengan algoritma non-rekursif, yaitu dengan memanfaatkan definisi stack dengan representasi *array*.

Algoritma non-rekursif nya adalah:

```
Procedure DFS_NonRekursif (input V : char)
{   I.S. Terdefinisi A yaitu matriks ketetanggan
    dari sebuah graf V adalah simpul yang akan
    diproses
    F.S. Simpul V telah di proses
}
```

Kamus
W : char
S : stack

Procedure CreateStack (output S : Stack)
{ I.S : sembarang
F.S : sebuah *stack* S kosong siap dipakai dan terdefenisi
}

Function IsEmptyStack (S : Stack) → boolean
{ Test stack kosong: *False* jika tumpukan tidak kosong dan mengirim *true* jika tumpukan kosong
}

Procedure PushStack (input/output S : Stack; Input P : address)
{ Menambahkan elemen baru pada TOP, dengan elemen yang sudah diketahui alamatnya
}

Procedure PopStack (input/output S : Stack; Output P : address)
{ I.S : *stack* tidak kosong
F.S: alamat elemen TOP disimpan pada P, shg Informasi dapat diakses melalui P }
{ Menghapus elemen *stack*, *stack* tidak boleh Kosong dan mungkin setelah penghapusan *stack* menjadi kosong
}

Procedure Proses (input V : char)
{ I.S: V adalah simpul yang akan Diproses
F.S: simpul V telah diproses
}

Algoritma
```
Proses (V)
Visited[ V] ← true
while not IsEmptyStack (S) do
    PopStack (S,V)
    while (V<>Nil) do
        if (A[V,W] =1) and (not visited [W] ) then
                Proses(W) PushStack
        (S,W) Visited[W] ← true
        endif
     endwhile
endwhile
IsEmptyStack (S)
```

Penelusuran graf dengan DFS rekursif direpresentasikan dengan tabel sebagai berikut:

Simpul yang dikunjungi	Isi Simpul	Alamat yang di Push	Isi Stack	Karakter Tercetak
A	A	Right (A) = C	C	A
Left (A)	B	Right (B) = E	CE	B
Left (B)	D	Right (D) = H	CEH	D
Left (D)	Nil	-	CEH	-
Pop	H	-	CE	H
Left (H)	Nil	Nil	CE	-
Pop	E	-	C	E
Left (E)	Nil	Right (E) =H	CH	F
Pop	H	-	C	-
Pop	C	Right (C) =G	G	C
Pop	G	Nil	-	G

Tabel 2. Penelusuran DFS dengan Algoritma nonRekursif

Analisis Ruang :

Berbeda dengan algoritma rekursif, pada algoritma ini, *programmer* harus terlebih dahulu mendefinisikan *stack* beserta operasinya. Hal ini disebabkan *stack* bersifat eksplisit, yang artinya segala sesuatu yang berhubungan dengan *stack* harus dikelola *programmer* sendiri. Pengelolaan mandiri ini menyebabkan proses dan segala operasi pada *stack* akan sangat mudah untuk ditelusuri dan memperkecil alokasi memori yang dibutuhkan.

Analisis Waktu :

Operasi dasar dari algoritma ini adalah pengecekan status *stack*, apakah kosong atau tidak sehingga menyebabkan kompleksitas algoritma ini adalah O(n), dengan n adalah jumlah simpul yang dikunjungi.

5. KESIMPULAN

Sebenarnya dalam prinsip pengaplikasian kedua algoritma DFS ini sama-sama menggunakan *stack*.

Perbedaan mendasarnya adalah dalam algoritma rekursif, *stack* dikelola secara implisit oleh *compiler* sedangkan pada algoritma non-rekursif, *stack* dikelola secara eksplisit oleh *programmer* baik dalam proses maupun operasi di dalamnya.

Dari segi efisiensi dalam kode program, algoritma DFS rekursif lebih hemat karena hanya tinggal memanggil prosedur di atasnya tanpa memperhatikan pengelolaan *stack*. Akan tetapi, melihat dari segi memori yang digunakan, algoritma non-rekursif lebih efisien dan efektif karena memori yang digunakan lebih sedikit.

6. SARAN

Jika jumlah simpul pada graf yang akan ditelusuri secara DFS relatif sedikit, maka sebaiknya digunakan algoritma rekursif karena akan lebih efisien dan efektif dari segi program itu sendiri. Sebaliknya, jika simpul yang akan ditelusuri cukup banyak, maka algoritma non-rekursif merupakan solusi terbaik karena hanya sedikit memakan memori.

REFERENSI

[01] Munir, Rinaldi. 2006. *Matematika Diskrit*. Bandung : ITB.

[02] Munir, Rinaldi. 2007. *Strategi Algoritmik*. Bandung: ITB.