Pemanfaatan *Directed Acyclic Graph* untuk Merepresentasikan Hubungan Antar Data dalam Basis Data

Winson Waisakurnia (13512071)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512071@std.stei.itb.ac.id

***Abstrak*—Kita sering mencari suatu data dalam basis data. Wikipedia merupakan laman dengan basis data yang besar dan menampung banyak data/pengetahuan. Terkadang kita tidak mengerti penjelasan yang terdapat pada basis data tersebut karena untuk mengerti data tersebut kita perlu membaca data yang mendahului atau menjadi dasar dari data tersebut. Graf dapat digunakan untuk merepresentasikan basis data sehingga kita dapat mengakses suksesor dan predesesor suatu data dengan mudah. *Directed Acyclic Graf*, graf berarah dan tidak bersirkuit adalah salah satu representasi graf yang cocok untuk merepresentasikan basis data dengan kebutuhan pengaksesan suksesor dan predesesor dengan cepat.**

***Index Terms*—data, database, representasi, DAG, graf, berarah.**

I. PENDAHULUAN

Ketidaktahuan suatu istilah atau data seringkali berakhir dengan pencarian data dalam basis data. Salah satu tempat yang paling sering dikunjungi untuk mencari suatu data adalah *Wikipedia*, proyek ensiklopedia multibahasa yang dapat diakses melalui internet. Pencarian data terutama melalui internet dilakukan oleh berbagai kalangan masyarakat untuk memenuhi rasa ingin tahu ataupun karena tuntutan pendidikan atau pekerjaan.

Kadang kita tidak mengerti data yang telah dicari melalui internet. Kejadian seperti ini disebabkan karena kita belum mengetahui atau membaca data lainnya yang menjadi predesesor, menjadi dasar dari data yang sedang kita akses. Contohnya ketika kita ingin mengetahui pengertian DAG (*Directed Acyclic Graf*), kita mencarinya di *Wikipedia*. Namun jika kita belum pernah membaca tentang graf, maka kemungkinan besar kita tidak akan mengerti tentang DAG setelah membacanya di *Wikipedia*.

Kandang kita juga ingin mengetahui manfaat dari suatu data setelah membacanya dari basis data. Akan sangat mudah jika dari data yang kita akses terdapat rujukan ke laman lainnya yang memuat manfaat-manfaat dari data yang sedang kita baca secara langsung. Misalnya dari membaca graf kita mengetahui bahwa terdapat DAG. Dari membaca DAG kita mengetahui bahwa kita dapat melakukan *topological sort* pada DAG. Dari *topological sort* kita dapat menyusun rencana pengambilan mata kuliah.

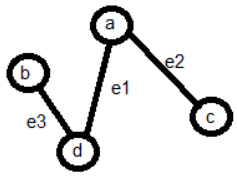
Jika kita dapat menghubungkan setiap data maka kita dapat dengan mudah mengakses dasar dari suatu data dan turunan dari suatu data sehingga pengaksesan data menjadi lebih efisien.

II. TEORI DASAR

2.1 Graf

Graf adalah sebuah struktur yang terdiri dari himpunan simpul (*node* atau *vertices*) dan sisi (*edge* atau *arc*). Kumpulan simpul dan sisi boleh disebut graf jika terdapat minimal satu simpul pada himpunan simpul sedangkan himpunan sisi boleh kosong. Graf yang hanya memiliki satu simpul tanpa sisi dinamakan graf trivial.

Menurut buku *Matematika Diskrit* [1], simpul pada graf dapat dinomori dengan huruf, misalnya a,b,c,…, dengan bilangan asli 1,2,3,.., atau gabungan keduanya. Sisi yang menghubungkan simpul v_i dan simpul v_j dapat dinyatakan dengan pasangan (v_i, v_j) atau dengan lambang e_1,e_2,…



Gambar 2.1 Visualisasi dari graf

Gambar 2.1 memperlihatkan contoh dari graf dengan simpul a,b,c,d dan sisi e_1,e_2,e_3. Sisi e_3 menghubungkan node b dan d. Sisi e_1 menghubungkan node a dan d. Sisi e_2 menghubungkan node a dan c. Graf juga dapat dituliskan dalam bentuk himpunan simpul dan sisi. Jika V adalah himpunan simpul dan E adalah himpunan sisi, maka dapat graf pada Gambar 2.1 dapat dituliskan dalam bentuk V={a,b,c,d} dan E={(b,d), (a,d), (a,c)}.

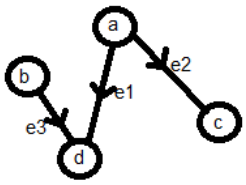
2.1.1 Jenis-Jenis Graf

Sisi pada graf dapat memiliki arah. Berdasarkan ada tidaknya arah pada sisi graf, graf dibedakan menjadi dua jenis [1]:

1. Graf tak-berarah (*undirected graph*)
   Graf berarah adalah graf yang sisinya tidak memiliki arah. Pada graf tak berarah, urutan simpul pada

penulisan sisi tidak penting. Sisi (v_j,v_k) dengan sisi (v_k,v_j) adalah sama dan kita hanya perlu menuliskan salah satunya pada himpunan sisi jika hanya satu sisi yang menghubungkan v_k dan v_j. Gambar 2.1 merupakan visualisasi dari graf yang tergolong graf tak-berarah.

2. Graf berarah (*directed graph* atau *digraph*)
   Graf berarah adalah graf yang sisinya diberikan arah. Pada graf berarah sisi lebih sering disebut busur. Busur (v_j,v_k) berbeda dengan busur (v_k,v_j). Jika (v_j,v_k) adalah sisi berarah/busur, maka v_j adalah simpul asal (*initial vertex*) dan v_k adalah simpul terminal (*terminal vertex*). Pada busur (v_j,v_k) terdapat jalur dari v_j ke v_k tapi tidak sebaliknya. Gambar 2.2 adalah contoh dari graf berarah.



Gambar 2.2 Visualisasi graf berarah

2.1.2 Terminologi Dasar Graf

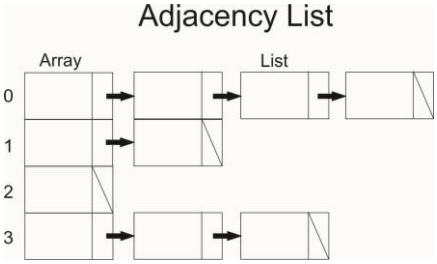
Beberapa terminologi (istilah) yang berkaitan dengan graf [1]:

1. Bertetangga (*Adjacent*)
   Dua simpul a dan b pada graf tak-berarah dapat dikatakan bertetangga apabila terdapat sebuah sisi yang menghubungkan simpul a dan b. Pada graf berarah simpul a dikatakan bertetangga dengan simpul b apabila terdapat sisi yang menghubungkan a dan b dengan a sebagai simpul asal dan b sebagai simpul terminal dan b dapat disebut tetangga dari a. Pada Gambar 2.2 , simpul b bertetangga dengan d dan d adalah tetangga dari b.
2. Bersisian (*Incident*)
   Untuk sembarang sisi e = (v_j,v_k), sisi e dikatakan bersisian dengan simpul v_j dan v_k.
3. Graf Kosong (*Null Graph* atau *Empty Graph*)
   Graf yang himpunan sisinya merupakan himpunan kosong disebut sebagai graf kosong, E={}.
4. Derajat (*Degree*)
   Derajat suatu simpul graf tak berarah adalah jumlah sisi yang bersisian dengan simpul tersebut. Derajat suatu simpul dari graf berarah dibedakan menjadi dua, derajat masuk dan derajat keluar. Derajat masuk adalah jumlah busur yang masuk ke simpul. Derajat keluar adalah jumlah busur yang keluar dari simpul.
5. Lintasan (*Path*)
   Lintasan yang panjangnya n dari simpul awal v_0 ke simpul akhir v_n pada graf G ialah dapat ditulis sebagai barisan simpul $v_0,v_1,v_2,\ldots,v_n$ dengan syarat terdapat sisi (v_0, v_1), (v_1, v_2), (v_2, v_3),…, (v_{n-1}, v_n). Pada graf tak-berarah, sisi (v_0, v_1) sama dengan sisi (v_1, v_0).
6. Siklus (*Cycle*) atau Sirkuit (*Circuit*)
   Siklus adalah lintasan yang berawal dan berakhir pada simpul yang sama.
7. Terhubung (*Connected*)
   Simpul v_i dan v_j dikatakan terhubung apabila terdapat lintasan dari v_i ke v_j. Graf tak-berarah dikatakan graf terhubung adalah graf yang setiap pasangan simpul pada grafnya terhubung. Graf berarah dikatakan terhubung jika graf tak-berarahnya merupakan graf terhubung.

2.1.3 Representasi Graf

Beberapa representasi graf adalah sebagai berikut [1]:

1. Matriks Ketetanggaan (*adjacency matrix*)
   Matriks ketetanggan merupakan representasi graf dalam bentuk matriks berukuran n x n dengan n adalah jumlah simpul. Bila A adalah sebuah matriks, jika $a_{i,j}$ bernilai 1 berarti simpul i bertetangga dengan simpul j. Jika $a_{i,j}$ bernilai 0 berarti simpul i tidak bertetangga dengan simpul j.
2. Senarai Ketetanggaan (*adjacency list*)



Gambar 2.3 Adjacency List

   Senarai ketetanggaan mengenumerasikan simpul-simpul yang bertetangga dengan setiap simpul pada graf. Gambaran senarai ketetanggaan dapat dilihat pada Gambar 2.3.

2.1.4 *Directed Acyclic Graph* (DAG)

Directed Acyclic Graph adalah graf berarah yang tidak mengandung siklus (cycle).

2.2 Basis Data (*Database*)

Basis data adalah kumpulan informasi yang disusun secara sistematik dengan metode tertentu sehingga mempermudah pengaksesan informasi.

Untuk mengakses informasi dari basis data dibutuhkan *Database Management System* (DBMS). DBMS adalah sekumpulan program yang membantu dalam memasukkan data, mengatur, dan memilih data dari database [3].

Network Data Model (NDM) atau yang dikenal sebagai "*CODASYL Data Model*" adalah DBMS yang menyimpan data dengan struktur seperti DAG yang dengan data sebagai node dan busur sebagai hubungan antar data. NDM memperbolehkan suatu data memiliki beberapa predesesor [4].

III. Representasi Graf Basis Data

A. Pemilihan Jenis Graf

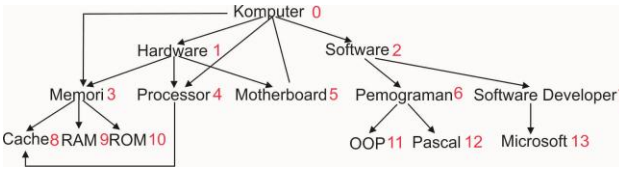
Pada laman yang berisi pengetahuan seperti *Wikipedia* seringkali kita membutuhkan teori dasar atau teori sebelumnya yang mendasari teori yang sedang kita baca karena ketidakmengertian. Terkadang kita juga ingin mengetahui turunan ataupun manfaat dari teori/pengetahuan yang sedang kita baca pada suatu laman.

Jika suatu data direpresentasikan dengan sebuah simpul maka dalam kasus ini kita membutuhkan sisi yang menghubungkan antar data. Untuk membedakan apakah data yang terhubung dengan suatu data adalah teori dasar atau turunan dibutuhkan sisi yang mempunyai arah atau busur dengan simpul asal sebagai teori dasar dari data yang berada pada simpul terminal.

Data dalam database dapat disusun dengan DBMS, direpresentasikan dalam berbagai varian graf dan salah satunya adalah pohon. Namun DBMS yang cocok untuk digunakan sangat bergantung pada kebutuhan. Representasi pohon tidak cocok untuk merepresentasikan data untuk kasus ini karena suatu data/pengetahuan dapat memiliki lebih dari satu teori dasar atau memiliki banyak predesesor. Representasi graf berarah lebih cocok untuk merepresentasikan data dalam kasus ini.

Jika terdapat suatu data direpresentasikan menjadi simpul A yang menjadi teori dasar dari data yang direpresentasikan dengan simpul B, maka tidak akan ada lintasan dari simpul B ke simpul A. Teori dasar dari suatu data bersifat lebih umum daripada data tersebut dan turunan suatu data bersifat lebih khusus daripada data tersebut. Jadi, tidak mungkin terdapat siklus pada representasi graf untuk basis data ini.

Graf berarah yang sebelumnya telah dipilih untuk merepresentasikan basis data ini kemudian dapat dipersempit menjadi graf berarah tanpa siklus, *directed acyclic graph* (DAG). Gambaran struktur data sederhana dalam bentuk DAG dapat dilihat pada Gambar 3.1. Warna merah merupakan id dari sebuah data.



Gambar 3.1 Representasi Graf dari Basis Data

B. Perancangan Struktur Data Sederhana

Setiap data dikaitkan dengan sebuah id yang unik dalam bentuk integer. Tujuan dalam pemberian id ini adalah agar data mudah diakses pada array. Graf direpresentasikan dengan *adjecency list* seperti pada Gambar 2.3.

Agar pengaksesan predesesor dan suksesor dapat dilakukan dengan cepat, dibutuhkan dua *adjecency list*. *Adjecency list* yang pertama berisi id data sebagai indeks array dengan list yang berisi id dari data turunannya. Misalkan id 0 berisi data tentang graf maka list yang ditunjuk oleh array dengan indeks 0 berisi id dari data tentang graf berarah, graf sederhana, dan kuratowski. *Adjecency list* yang kedua berisi id data sebagai indeks array dengan list yang berisi id dari teori dasar data. Misalkan id 0 berisi data tentang *suffix tree* maka list yang ditunjuk oleh array 0 akan berisi id dari data tentang graf, pohon, dan pemrosesan string. Mengakses suksesor dan predesesor dari sebuah simpul lebih mudah apabila menggunakan *adjecency list* atau *adjecency matrix* dibandingkan representasi graf yang lain. Oleh karena predecessor atau suksesor dari suatu data hanya sedikit, maka akan sangat boros jika menggunakan *adjecency matrix*. *Adjecency list* memerlukan memori sebanyak O(NM) dengan M adalah panjang dari list. Ukuran list biasanya sangat kecil, M<<N, namun kita membutuhkan dua *adjecency list*. *Satu adjecency matrix* bisa digunakan untuk megakses predesesor dan suksesor namun memerlukan memori sebanyak $O(N^2)$. Jika asumsi kita mempunyai $1x10^6$ data, maka *adjecency matrix* memerlukan memori sebesar $1x10^{12}$, sedangkan *adjecency list* memerlukan memori sebesar $2x10^6$.

Selain *adjecency list* untuk menggambarkan hubungan antardata, diperlukan sebuah array dengan dengan indeks sebagai id data dan array tersebut berisi data atau alamat dimana data dengan id tersebut berada.

Index Array	List
0	{3,1,4,5,2}
1	{3,4,5}
2	{6,7}
3	{8,9,10}
4	{8}
5	{}
6	{11,12}
7	{13}
8	{}
9	{}
10	{}
11	{}
12	{}
13	{}

Tabel I. Adjencency List 1 dari Gambar 3.1

Index Array	List
0	{}
1	{0}
2	{0}
3	{0,1}
4	{1,0}
5	{1,0}
6	{2}
7	{2}
8	{3,4}
9	{3}

10	{3}
11	{6}
12	{6}
13	{7}

Tabel II. Adjencency List 2 dari Gambar 3.1

C. Pengaksesan Predesesor dan Suksesor

Pengaksesan predesesor dapat dilakukan dengan cepat melaui adjecency list yang pertama dengan indeks sebagai id data dan list berisi id dari teori dasar data tersebut. Dengan waktu O(M) dengan M adalah banyaknya predesesor, kita dapat mencetak semua predesesor dari sebuah data. Pengaksesan array tidak diperhitungkan karena hanya memakan waktu O(1).

Pengaksesan suksesor juga dapat dilakukan dengan cepat melaui adjecency list yang kedua dengan indeks sebagai id data dan list berisi id dari turunan data tersebut. Dengan waktu O(M) dengan M adalah banyaknya suksesor, kita dapat mencetak semua suksesor dari sebuah data. Pengaksesan array tidak diperhitungkan karena hanya memakan waktu O(1).

Jika seseorang sedang membuka data tentang memori dengan id 3, maka pada laman tersebut juga akan menampilkan "Teori Dasar: Komputer, Hardware" diakses melalui *adjecency list* pada Tabel II pada array indeks 3. Juga ditampilkan "Turunan: Cache, RAM, ROM" diakses melalui *adjecency list* pada Tabel I pada array indeks 3.

D. Pemasukkan Data

Pemasukan data dapat dilakukan dengan memasukkan data pada indeks array yang belum terdapat data dengan melakukan traversal dari indeks awal dengan kompleksitas O(N) agar array yang kosong akibat penghapusan data dapat dipergunakan dengan optimal.

Apabila tidak memanfaatkan sel-sel yang bekas yang tidak terpakai, pemasukan data dapat dilakukan dengan kompleksitas O(1) namun akan menyebabkan array menjadi *sparse*. Pilihan ini akan berakibat pada banyaknya memori yang dialokasikan namun tidak terpakai.

Contohnya apabila ingin ditambahkan data tentang BIOS yang merupakan turunan dari Motherboard dan ROM, maka BIOS akan diberikan id 14 dan disediakan datanya pada array kontigu. Pada *adjecency list* 2, Tabel II, akan digunakan array indeks 14 dengan isi list {10,5}. Pada *adjecency list* 1, Tabel I, akan ditambahkan id pada list indeks 10 menjadi {14} dan list indeks 5 menjadi {14}. Pemasukkan data pada masing-masing *adjecency list* ini membutuhkan kompleksitas waktu sebesar O(M) dengan M adalah panjang dari list.

E. Penghapusan Data

Penghapusan data dapat dilakukan dengan menghapus data dengan indeks tertentu dengan memberikan sebuah nilai yang menandakan bahwa di sel tersebut tidak lagi terdapat data atau data yang terdapat pada indeks tersebut tidak valid.

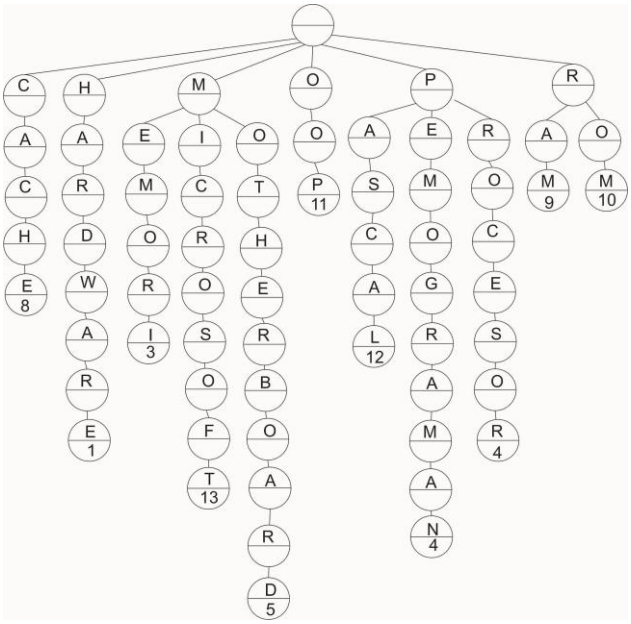
Selanjutnya, penghapusan berlanjut pada *adjecency list*. Dari *adjecency list* 2, Tabel II, kita dapat mengetahui di mana saja pada *adjecency list* 1, Tabel I, yang mengandung id yang akan dihapus. Begitu pula dari *adjecency list* 1, Tabel I, kita dapat mengetahui di mana saja pada *adjecency list* 2, Tabel II, mengandung id yang akan dihapus. Selanjutnya semua data pada list yang sudah tidak digunakan kemudian didealokasi.

Contohnya apabila ingin menghapus data mengenai *software developer* dengna id 7, maka pada array kontigu tempat penyimpanan data akan diganti nilainya untuk menandakan bahwa data pada indeks tersebut tidak valid dan dapat digunakan apabila akan melakukan penambahan data. Selanjutnya dari Tabel I, pada list indeks 7 terdapat id 13 sehingga pada Tabel II pada list indeks 13 dilakukan penghapusan data dengan id 7. List indeks 7 Tabel I kemudian didealokasi menjadi list kosong. Kemudian dari Tabel II pada indeks 7 terdapat list dengan id 2 sehingga pada Tabel I pada list indeks 2 dilakukan penghapusan data dengan id 7. List Tabel II dengan indeks 7 kemudian didealokasi menjadi list kosong.

F. Pencarian Data

Terdapat berbagai macam proses pencarian string seperti *brute force*, *suffix tree*, *Knuth-Morris-Pratt*, dan alogoritma lainnya. Namun pada saat ini penulis hanya dapat memberikan algoritma pencarian string yang sederhana dengan menggunakan pohon.

Pohon dapat digunakan untuk mencari suatu string yang merupakan nama dari data dengan menyusun setiap huruf sebagai simpul dari pohon. Setiap node ini berisi informasi mengenai huruf dan sebuah interger yang menyatakan apakah node tersebut merupakan akhir dari kata dan dimana lokasi string tersebut berada.



Gambar 3.2 Contoh Pencarian String dengan Pohon

Gambar 3.2 memperlihatkan contoh pencarian string dengan memanfaatkan pohon. Pohon ini akan diperbaharui setiap kali penambahan data baru atau penghapusan data. Pada gambar tersebut beberapa kata telah dihilangkan agar gambar tidak terlalu besar. Jika string yang kita input untuk pencarian memiliki panjang M, maka banyaknya pengecekan maksimal adalah 26xM. Jika panjang string input pada pencarian adalah M, maka dilakukan pencarian huruf pada M aras jika selalu ditemukan. Derajat maksimum dari pohon adalah 26 sehingga pencarian huruf pada tiap aras dilakukan traversal maksimum 26 kali. Kompleksitas waktu yang dibutuhkan adalah O(M).

Namun pencarian seperti ini masih tidak cukup kuat dan masih terdapat banyak kelemahan karena kita tidak dapat mencari data dengan menginput substring yang bukan merupakan awalan dari judul sebuah data. Misalnya mencari data "memori" dengan menginput "emori". Pencarian ini juga tidak dapat mengembalikan data yang namanya paling dekat dengan input jika tidak ditemukan, misalnya jika terdapat kesalahan input menjadi "memoro". Jika kita menginput "memo" kita masih dapat mengembalikan data "memori" dengan bantuan DFS (*Depth First Search*) apabila pada pencarian "memo" berhasil namun huruf 'o' bukan merupakan akhir dari salah satu judul pada data yang terdapat pada pohon.

V. KESIMPULAN

Directed Acyclic Graf (DAG) dapat digunakan untuk merepresentasikan hubungan antar data yang sederhana pada basis data (*database*). Struktur data menyerupai bentuk sederhana dari *Database Management System* (DBMS) yang ada dengan *Network Data Model* (NDM).

Directed Acyclic Graf (DAG) sendiri dapat direpresentasikan dengan dua *adjecency list* untuk mempercepat pengaksesan data. Pada tulisan ini juga telah diberi gambaran sederhana bagaimana proses *query* seperti memasukkan data, menghapus data, pengaksesan suksesor dan predesesor, serta pencarian dapat dilakukan pada graf ini berserta kompleksitasnya.

Dengan model seperti ini maka kita dapat dengan mudah mencari teori dasar dari dan turunan dari sebuah data sehingga penggalian pengetahuan atau data pada suatu basis data dapat dilakukan dengan lebih cepat tanpa harus melakukan pencarian ulang.

REFERENSI

Munir, Rinaldi, *Matematika Diskrit*, 4th, Teknik Informatika ITB, 2006.

Rosen, Kenneth H., *Discrete Mathematics and Its Applications*, 7th, McGraw-Hill, 2012.

http://www.webopedia.com/TERM/D/database.html. Diakses pada tanggal 14 Desember 2013, pukul 18.55

http://coronet.iicm.tugraz.at/wbtmaster/allcoursescontent/netlib/ndm1.htm. Diakses pada tanggal 14 Desember 2013, pukul 18.49

PERNYATAAN

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

Bandung, 16 Desember 2013

Winson Waisakurnia
13512071