Implementasi Pohon dalam Bahasa Prolog

Kevin Maulana – 13512044
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
kevhnmay94@students.itb.ac.id

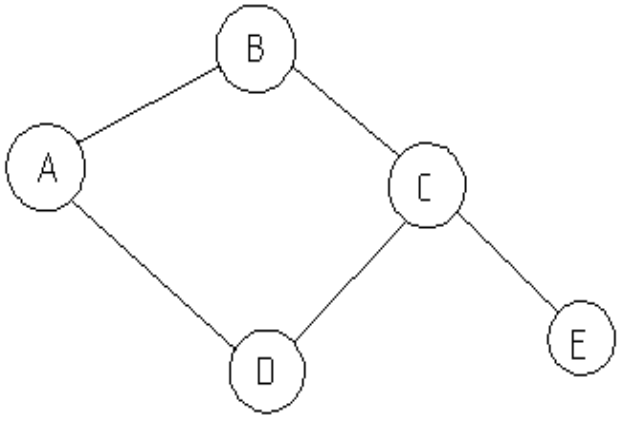
***Abstract*—Prolog merupakan salah satu bahasa pemrograman dengan pendekatan deklaratif yang dikenal oleh banyak orang. Makalah ini membahas tentang implementasi pohon dalam Prolog.**

***Index Terms*—pohon, Prolog, bahasa pemrograman, backtracking**

1. PENDAHULUAN

A. Graf

Graf adalah suatu struktur diskrit yang terdiri atas simpul dan busur yang menghubungkan simpul-simpul tersebut. Secara matematis, G adalah suatu pasangan himpunan (V,E) dimana V adalah himpunan simpul (*vertices*) yang tidak kosong, dan E adalah himpunan busur (*edges*) yang menghubungkan pasangan simpul. Notasi singkatnya dapat ditulis sebagai G = (V,E).



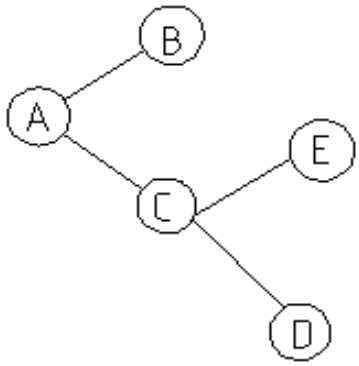
Gambar 1 : Contoh graf.

Graf dalam Gambar 1 dapat direpresentasikan sebagai berikut:

V = {A,B,C,D,E}
E = {(A,B),(A,D),(B,C),(C,D),(C,E)}.

B. Pohon

Pohon adalah bentuk graf tak-berarah terhubung yang tidak mempunyai sirkuit. Graf pada Gambar 1 bukanlah contoh sebuah pohon, karena graf tersebut mempunyai sirkuit yang menghubungkan simpul A,B,C, dan D.



Gambar 2 : Contoh pohon.

B.1. Sifat-sifat Pohon

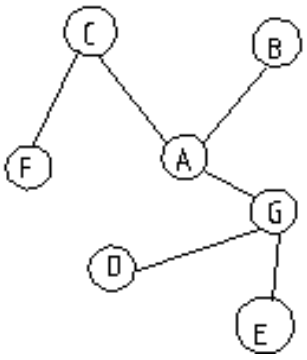
Sebuah pohon mempunyai sifat-sifat berikut:

1. Setiap pasang simpul dalam pohon terhubung dengan lintasan tunggal.
2. Pada pohon dengan jumlah simpul *n*, pohon memiliki *n-1* buah sisi.
3. Pohon tidak mengandung sirkuit dan penambahan satu sisi hanya membuat satu sirkuit.
4. Semua sisi pada pohon adalah jembatan, dimana jika salah satu sisinya dihapus pohon akan terpecah menjadi dua komponen.

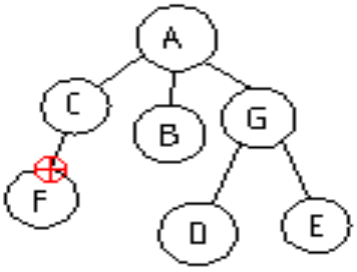
B.2. Pohon Berakar

Pada pengaplikasiannya, simpul tertentu pada pohon diperlakukan sebagai akar (*root*). Ketika sebuah simpul ditetapkan sebagai akar, maka simpul-simpul lainnya pada pohon dapat dicapai dengan memberi arah pada sisi-sisi pohon yang mengikutinya. Pada pohon berakar, setiap simpul dari pohon harus dicapai melalui akar, sehingga petunjuk arah pada pohon berakar dapat dihilangkan.

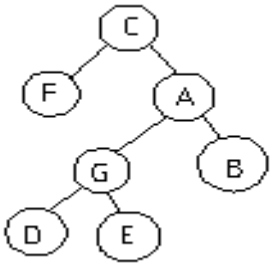
Sembarang pohon tak-berakar dapat diubah menjadi pohon berakar dengan memilih sembarang simpul pada pohon sebagai akar. Pemilihan simpul yang berbeda akan menghasilkan simpul yang berbeda pula, seperti yang diperlihatkan pada Gambar 3 dibawah ini.



Gambar 3 (a): Pohon Tak-Berakar.



Gambar 3 (b): Pohon berakar dengan akar A.



Gambar 3 (c): Pohon berakar dengan akar C.

B.3. Terminologi Pohon Berakar

Pohon berakar mempunyai beberapa terminologi yang penting, seperti:

1. Anak dan Orangtua

Misalkan *x* dan *y* adalah sebuah simpul dalam pohon berakar. Simpul *y* dikatakan anak dari simpul *x* jika ada sisi dari simpul *x* ke *y*. Dalam hal ini, *x* dapat disebut sebagai orangtua dari *y*. Pada Gambar 3(c). F dan A merupakan anak dari simpul C, sedangkan C merupakan orangtua dari simpul F dan A. G dan B merupakan anak dari A, sedangkan A merupakan orangtua dari simpul dari G dan B. Simpul F,D,E, dan B tidak mempunyai anak.

2. Lintasan

Lintasan dari simpul v_1 ke simpul v_k adalah runtunan simpul-simpul $v_1,v_2,...,v_k$ sehingga v_i adalah orangtua dari v_{i+1} untuk $1 \leq i < k$. Dari pohon pada Gambar 3 (c), lintasan dari simpul C ke E adalah C,A,G,D. Panjang lintasan adalah jumlah sisi yang dilalui lintasan, yaitu k-1. Panjang lintasan dari C ke E adalah 3.

3. Keturunan dan Leluhur

Jika terdapat lintasan dari simpul x ke simpul y di dalam pohon, maka x adalah leluhur dari simpul y. Pada Gambar 3 (c), C adalah leluhur dari E, sedangkan E adalah keturunan dari C.

4. Saudara kandung

Simpul yang berorangtua sama adalah saudara kandung. Pada Gambar 3 (a), D merupakan saudara kandung dari E, tetapi F bukan merupakan saudara kandung dari E. karena orangtuanya berbeda.

5. Upagraf

Misalkan x adalah simpul dalam pohon T. Upagraf merupakan pohon $T' = (V',E')$ dengan simpul x sedemikian rupa sehingga V' mengandung x dan semua keturunannya dan E' mengandung sisi-sisi dalam semua lintasan yang berasal dari x. Sebagai contoh, pada Gambar 4, $T' = (V',E')$ merupakan upapohon dengan A sebagai akarnya, dimana:

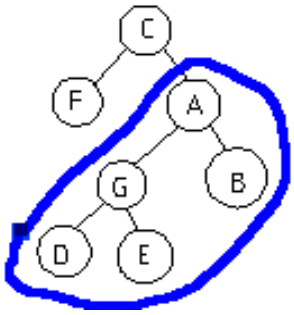
V' = {A,G,B,D,E}

E' = {(A,G),(A,B),(G,D),(G,E)}

Selain T', kita juga dapat mengambil T" = (V",E") dengan G sebagai akarnya, dimana:

V" = {G,D,E}

E" = {(G,D),(G,E)}



Gambar 4: Upapohon T' = (V',E') dengan x sebagai akar.

6. Daun

Daun adalah simpul yang tidak mempunyai anak. Pada Gambar 3(a), daun dari pohon tersebut adalah F,B,D,E.

7. Aras/Tingkat

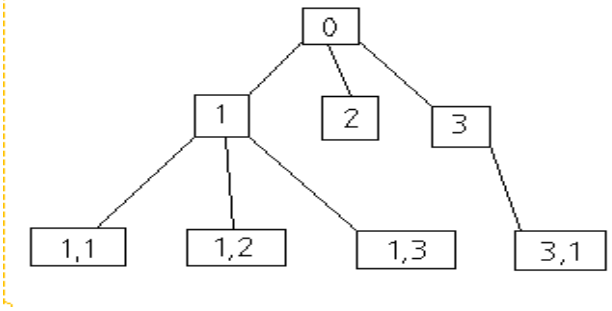
Akar mempunyai aras = 0, sedangkan simpul lainnya mempunyai aras sebanyak panjang lintasan dari akar ke simpul tersebut.

B.5. Pohon Terurut

Pohon berakar yang urutan anak-anaknya penting disebut pohon terurut. Pada pohon terurut, urutan anak-anak dispesifikasikan dari kiri ke kanan.

Sistem yang universal dalam spesifikasi simpul-simpul pada pohon terurut adalah dengan memberi nomor setiap simpulnya seperti penomoran bab

dalam sebuah buku. Simpul akar diberi nomor 0, simpul yang mengikuti akar diberi nomor 1, 2,3 dan seterusnya, anak-anak dari simpul 1 diberi nomor 1.1,1.2,..., anak-anak dari simpul 2 diberi nomor 2.1,2.2,..., begitu seterusnya. Sistem penomoran dimulai dari urutan kiri ke kanan.



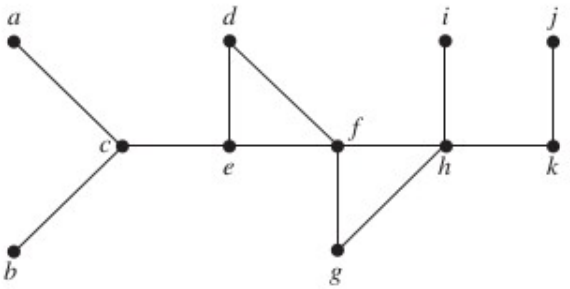
Gambar 5: Pohon terurut

B.6. Backtracking

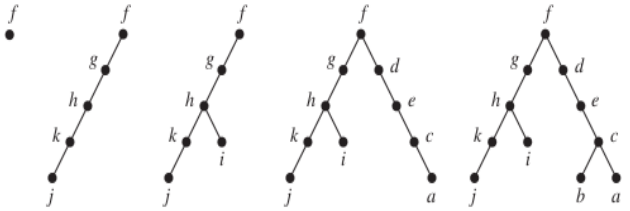
Banyak cara untuk membuat sebuah pohon dari sebuah graf, dan masing-masing mempunyai kelebihan tersendiri. Salah satu cara yang akan dibahas dalam makalah ini adalah metode *backtracking*.

Backtracking adalah salah satu metode traversal dengan cara melacak lintasan hingga ditemukan apa yang dicari atau sampai tidak ada lintasan baru yang dapat dilacak. Cara membuat pohon dari graf dengan backtracking adalah sebagai berikut:

1. Pilih satu simpul sembarang sebagai akar dari pohon.
2. Pilih salah satu simpul yang terhubung dari simpul asal (jika ada) dan buatlah sebuah lintasan yang menghubungkan kedua simpul tersebut. Ulangi hingga simpul baru yang tidak terhubung sudah tidak ditemukan.
3. Jika semua simpul belum terhubung, dan tidak ada simpul baru yang terhubung dari simpul asal,kembalilah ke simpul sebelumnya yang telah dilalui, hingga ditemukan simpul baru yang belum terhubung. Ulangi prosedur 2 dan prosedur ini hingga semua simpul telah terhubung.



Gambar 5(a): Graf G.



Gambar 5(b): Proses backtracking pada graf G.

C. Unifikasi dalam Logika Relasional

Unifikasi merupakan sebuah pondasi penting dalam beberapa bahasa pemrograman berbasis logika, seperti Prolog.

C.1. Definisi Logika Relasional

Logika relasional adalah salah satu pendekatan logika dengan menunjukkan hubungan satu atau lebih objek.

C.2. Terminologi Logika Relasional

1. Variabel

Variabel adalah suatu objek yang dapat digantikan oleh objek apapun. Dalam makalah ini, variabel ditulis dengan awalan huruf kapital.

Contoh: *X,Y,Z,Orang*

2. Konstanta

Konstanta adalah suatu objek yang tetap, tidak dapat diganti oleh objek apapun. Dalam makalah ini, konstanta ditulis dengan kata dengan awalan huruf *lowercase* atau angka. Ada 3 jenis konstanta, yaitu:

- Konstanta Objek: Konstanta yang menunjuk pada suatu objek. Contoh: *alvin, timothy, 4*.
- Relasi: Konstanta yang menunjukkan hubungan beberapa objek. Contoh: *cinta(mary,X), bunga(melati)*.
- Fungsi: Konstanta yang menghasilkan objek baru berdasarkan beberapa objek dalam fungsi tersebut. Contoh: *ibu(peter)* menghasilkan mary, *plus(2,2)* menghasilkan 4.

3. Aritas

Aritas adalah jumlah objek yang dapat dimasukkan ke dalam fungsi atau relasi.

Contoh: *bapak(putra,putri)* mempunyai aritas 2, *antara(kevin,steve,bangkukosong)* mempunyai aritas 3.

C.3. Substitusi

Seperti yang telah dijelaskan sebelumnya, suatu variabel dapat digantikan oleh objek apapun. Penggantian inilah yang disebut *substitusi*. Dalam logika relasional, substitusi digambar sebagai himpunan pengganti variabel, seperti (X ← a, Y ← f(b), Z ← m). Berikut contoh penggunaan substitusi pada logika relasional:

$$r(X,Y)(X \leftarrow 3, Y \leftarrow 4) = r(3.4)$$
$$g(P,Q)(P \leftarrow police) = g(police,Q)$$

C.4. Unifikasi

Unifikasi adalah proses penyamaan pada dua pernyataan relasional. Kedua pernyataan dinyatakan dapat terunifikasi apabila ada suatu substitusi yang menyebabkan kedua pernyataan tersebut menjadi sama. Substitusi yang menyebabkannya disebut juga sebagai *unifier*. Berikut contoh unifikasi pada logika relasional:

$$f(X, konrad)\{X \leftarrow lisar, Y \leftarrow konrad\} = f(lisar, konrad)$$
$$f(lisar, Y)\{X \leftarrow lisar, Y \leftarrow konrad\} = f(lisar, konrad)$$

$$\begin{matrix} g(delfador, kalenz) \\ g(asheviere, deoran) \end{matrix}$$ Tidak terunifikasi

$$\begin{matrix} h(X, elf) \\ h(dwarf, X) \end{matrix}$$ Tidak terunifikasi

2. PROLOG DAN IMPLEMENTASINYA

A. Introduction to Prolog

Prolog (Programming in Logic) adalah bahasa pemrograman berbasis logika. Prolog dikembangkan oleh Alain Colmerauer dan Phillipe Roussel dari Universitas Aix-Marseille, Prancis, serta Robert Kowalski dari Universitas Edinburgh sekitar tahun 1972. Kowalski membuat gambaran teoritis dari Prolog, sedangkan Colmerauer membuat formalitas dari bahasa Prolog dan Roussel membuat interpreter Prolog pertama. Compiler Prolog pertama dibuat oleh David Warren, ahli Artificial Intelligence di Universitas Edinburgh.

Prolog terdiri dari serangkaian aturan dan fakta yang direpresentasikan dalam sebuah file. Sebuah program dijalankan dengan memasukkan sebuah query, kemudian mengecek apakah query tersebut dapat dibuktikan dengan aturan dan fakta yang diketahui.

B. Fakta dalam Prolog

Dalam Prolog kita dapat membuat suatu pernyataan dengan membuat suatu program. Misalkan kita merepresentasikan suatu fakta bahwa sekarang hujan dan waktu masih pagi dengan menulis program:

```
hujan.
```

Kita dapat menanyakan query pada interpreter Prolog dengan:

```
?- hujan.

yes
```

Interpreter Prolog akan mengembalikan *yes* karena query tersebut terdapat dalam fakta yang diketahui program tersebut.

Kita juga dapat membuat fakta yang berbentuk relasi. Misalkan pada suatu dunia imaginer, Pat adalah bapak dari Peter, Mary adalah ibu dari Peter, Peter adalah bapak dari Tracy, Piper, dan Anne dengan menulis program:

```
bapak(pat,peter).
bapak(peter,tracy).
bapak(peter,piper).
bapak(peter,anne).
ibu(mary,peter).
```

Misalkan kita ingin mengetahui apakah Peter adalah bapak dari Pat:

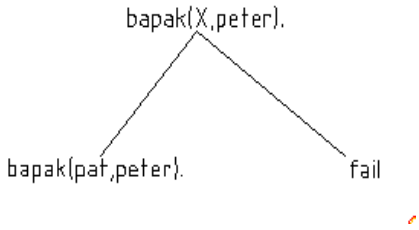
```
?- bapak(peter,pat).

no
```

Kita juga dapat mengetahui siapa bapak dari Peter dengan memasukkan variabel dalam query:

```
?- bapak(X,peter).

X = pat ?
```

Pada tahap tersebut program menemukan solusi dari X yaitu Pat. Program kemudian menanyakan apakah kita menerima solusi tersebut atau tidak. Kita dapat menerima solusi tersebut dengan menekan Enter, mencari solusi berikutnya dengan menekan `;` atau *'a'* untuk menunjukkan semua solusi Karena tidak ada solusi lain, program akan mengembalikan *no* jika kita mencari solusi lain selain solusi X = pat.

```
X = pat ? /* Tombol Enter ditekan */

yes

X = pat ? ;

no

X = pat ? a

no
```

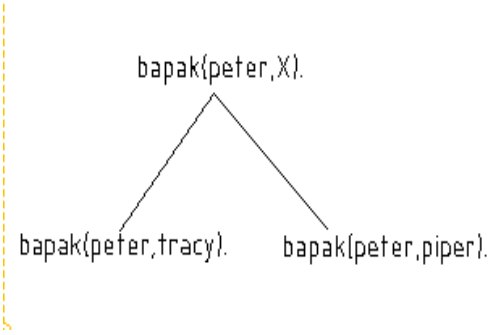


Gambar 6: Backtracking 1.

Contoh lain adalah ketika kita ingin mengetahui seseorang yang mempunyai bapak Peter.

```
?- bapak(peter,X).

X = tracy ? ;
```

```
X = piper

yes
```



Gambar 7: Backtracking 2.

C. Aturan dalam Prolog

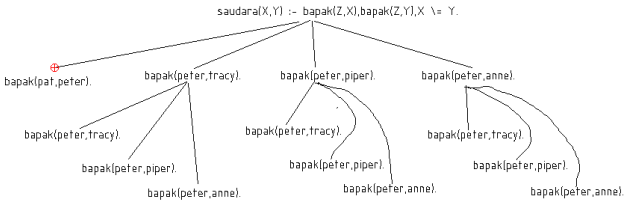
Aturan dalam Prolog memungkinkan kita untuk membuat suatu pernyataan kondisional berdasarkan fakta yang ada, sehingga kita tidak perlu menuliskan fakta baru. Misalkan kita akan membuat pernyataan tentang saudara kandung.

```
saudara(X,Y) :-
   bapak(Z,X),
   bapak(Z,Y),
   X \= Y.
```

Program akan mengecek apakah Z merupakan bapak dari X, Z merupakan bapak dari Y, dan X tidak sama dengan Y, karena kita tidak mengharapkan adanya solusi yang sama.

Misalkan kita ingin mengetahui semua relasi saudara dari program, kita dapat menuliskan query sebagai berikut:

```
?- saudara(A,B).

X = tracy
Y = piper ? a

X = tracy
Y = anne

X = piper
Y = tracy

X = piper
Y = anne

X = anne
Y = tracy

X = anne
Y = piper

no
```



Gambar 8: Backtracking 3.

Perhatikan bahwa jumlah anak pada pohon sesuai dengan banyaknya fakta yang ada, dalam hal ini, fakta yang mempunyai predikat *bapak* ada empat. Pada tingkat pohon yang kedua, Z telah diketahui, sehingga kita hanya mencari predikat *bapak* dengan aritas pertama Z.

REFERENSI

Munir, Rinaldi. "Matematika Diskrit". 2010. Penerbit Informatika.

Rosen, Kenneth H. "Discrete Mathematics and Its Applications Seventh Edition". 2012. McGraw-Hill.

http://www.doc.gold.ac.uk/~mas02gw/prolog_tutorial/prologpages/index.html, Diakses tanggal 17 Desember 2013, 05:44.

http://www.mta.ca/~rrosebru/oldcourse/371199/prolog/history.html, Diakses tanggal 17 Desember 2013, 05:23.

http://kur2003.if.itb.ac.id/file/lecture09.pdf, Diakses tanggal 16 Desember 2013, 19:50.

http://kur2003.if.itb.ac.id/file/lecture05.pdf, Diakses tanggal 16 Desember 2013, 20:02.

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, 17 Desember 2013

Kevin Maulana 13512044