ANALISIS ALGORITMA PEMBANGUN POHON EKSPRESI DARI NOTASI PREFIKS DAN POSTFIKS

R. Raka Angling Dipura – NIM : 13505056

Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
E-mail : if15056@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang kemangkusan dua algoritma yang sering digunakan untuk membangun pohon ekspresi. Algoritma yang diperbandingkan adalah algoritma pembangun pohon ekspresi dari notasi prefiks dan algoritma pembangun pohon ekspresi dari notasi postfiks. Kedua algoritma pembangun pohon ekspresi ini jelas memiliki karakteristik tersendiri, hal ini disebabkan karena notasi prefiks dan postfiks yang dibaca juga memiliki perbedaan mendasar.

Notasi prefiks, yang juga dikenal dengan *Polish notation*, pertama kali ditemukan oleh matematikawan berkebangsaan Polandia, Jan Lukasiewicz, pada tahun 1920-an. Pola penulisan notasi ini adalah dengan mencantumkan *operator* diikuti dua *operand* (*operand* dapat bersifat rekursif, yaitu terdiri dari *operator* dan *operand* yang juga tersusun prefiks).

Berikut ini contoh penulisan notasi prefiks : + * / + 10 4 7 2 - + * 1 2 2 + 6 3
Dengan memperjelas operasi antar *operand*, notasi prefiks di atas dapat kita ubah menjadi notasi yang biasa digunakan dalam kehidupan sehari-hari yaitu notasi infiks. Berikut ini langkah-langkahnya :
⇒ + * / (+ 10 4) 7 2 - + (* 1 2) 2 (+ 6 3)
⇒ + * (/ (+ 10 4) 7) 2 – (+ (* 1 2) 2) (+ 6 3)
⇒ + (* (/ (+ 10 4) 7) 2) (- (+ (* 1 2) 2) (+ 6 3))
⇒ (+ (* (/ (+ 10 4) 7) 2) (- (+ (* 1 2) 2) (+ 6 3)))
⇒ ((((10 + 4) / 7) * 2) + (((1 * 2) + 2) – (6 + 3)))

Notasi postfiks, yang juga dikenal dengan *Reverse Polish notation*, pertama kali ditemukan oleh pakar komputer berkebangsaan Australia, Charles Hamblin, pada tahun 1950-an. Notasi ini merupakan penurunan dari notasi prefiks. Charles Hamblin mempresentasikan hasil kerjanya atas notasi postfiks dalam suatu konferensi pada tahun 1957. Pola penulisan notasi postfiks adalah dengan mencantumkan dua *operand* diikuti satu *operator*. Sama seperti notasi prefiks, *operand* pada notasi postfiks dapat bersifat rekursif tetapi tersusun menurut aturan postfiks.

Berikut ini contoh penulisan notasi postfiks : 10 4 + 7 / 2 * 1 2 * 2 + 6 3 + - +
Dengan memperjelas operasi antar *operand*, notasi postfiks di atas juga dapat kita ubah menjadi notasi infiks. Berikut ini langkah-langkahnya :
⇒ (10 4 +) 7 / 2 * (1 2 *) 2 + (6 3 +) - +
⇒ ((10 4 +) 7 /) 2 * ((1 2 *) 2 +) (6 3 +) - +
⇒ (((10 4 +) 7 /) 2 *) (((1 2 *) 2 +) (6 3 +) -) +
⇒ ((((10 4 +) 7 /) 2 *) (((1 2 *) 2 +) (6 3 +) -) +)
⇒ ((((10 + 4) / 7) * 2) + (((1 * 2) + 2) – (6 + 3)))

***Kata kunci**:* pohon ekspresi, notasi prefiks, notasi postfiks.

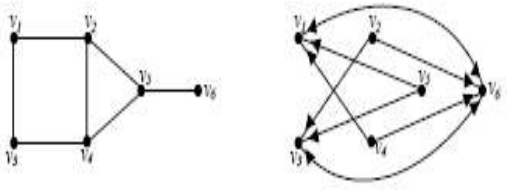
1. Pendahuluan

Di antara sekian banyak konsep dalam teori graf yang ada, mungkin pohon (*tree*) merupakan konsep yang paling penting. Sejak pertama kali digunakan pada 1857 oleh matematikawan Inggris, Arthur Cayley, pohon menjadi salah salah satu pokok kajian intensif. Dalam kehidupan sehari-hari pun sering kita temui penerapan pohon, misalnya untuk menggambarkan silsilah keluarga atau struktur organisasi.

Pohon adalah graf tak berarah terhubung yang tidak mengandung sirkuit. Salah satu simpulnya (*node*) mendapat perlakuan "khusus" dan disebut akar (*root*). Tiap simpul menjadi orangtua (*parent*) dari simpul anak (*child*), yaitu simpul yang terletak di bawahnya. Simpul yang terletak paling bawah (tidak memiliki anak) disebut daun (*leaf*).

Menurut banyaknya anak pada tiap simpul cabang, pohon terbagi lagi menjadi pohon-pohon *n-ary*, dengan n adalah anak paling banyak yang dimiliki tiap simpul cabang. Jika tiap simpul cabang pohon *n-ary* memiliki tepat n buah anak, maka pohon *n-ary* tersebut dikatakan teratur atau penuh.

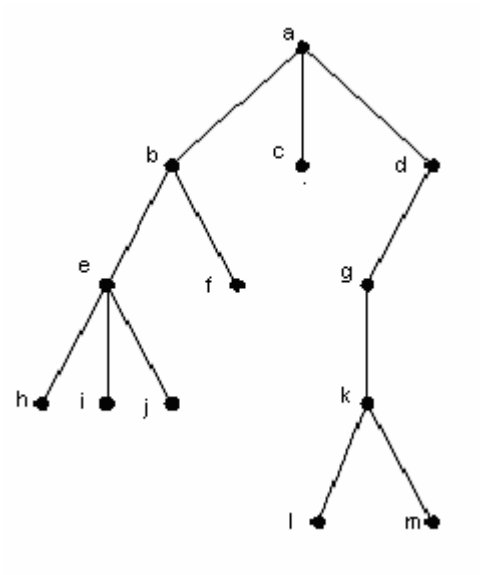
Pohon yang dibahas dalam makalah ini adalah pohon ekspresi, yaitu pohon biner (*2-ary*) dengan daun berupa *operand* dan simpul dalam (termasuk akar) berupa *operator*.



1a 1b

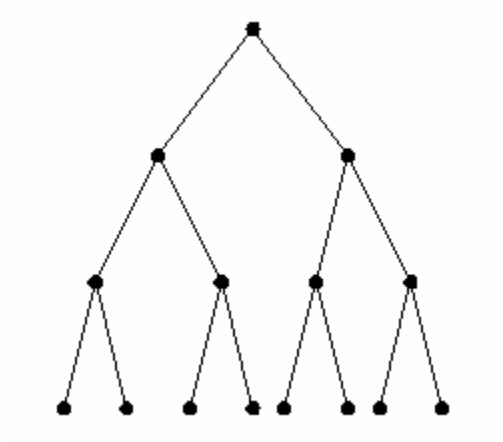
Gambar 1

Gambar 1a adalah contoh graf tak- berarah dengan enam simpul (v_1, v_2, v_3, v_4, v_5, v_6)., Graf tersebut disebut graf tak-berarah karena sisi-sisi yang menghubungkan antar simpul tidak memiliki arah. Sedangkan gambar 1b adalah contoh graf berarah dengan enam simpul (v_1, v_2, v_3, v_4, v_5, v_6). Jelas terlihat bahwa sisi-sisi graf berarah memiliki panah penunjuk.



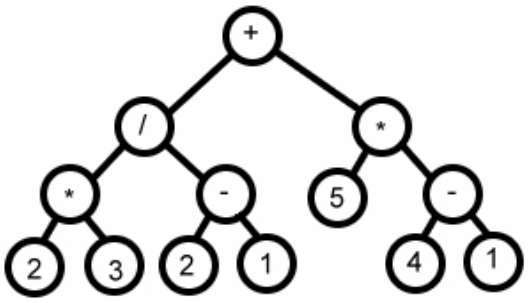
Gambar 2

Gambar 2 adalah contoh pohon dengan a sebagai akar. Sementara h, i, j, f, c, l, dan m adalah daun dari pohon tersebut.



Gambar 3

Gambar 3 adalah contoh pohon *n-ary* dengan nilai n=2 (biner). Karena tiap simpul cabang pohon biner di atas memiliki tepat dua anak, maka pohon biner di atas merupakan pohon biner penuh.



Gambar 4

Gambar 4 adalah contoh pohon ekspresi. Daun-daun dari pohon ekspresi merupakan *operand*, sedangkan simpul cabangnya (termasuk akar) merupakan *operator*. Dengan menelusuri pohon ekspresi di atas, notasi prefiks dan postfiks dari pohon tersebut dapat langsung ditentukan.
Notasi prefiks : + / * 2 3 – 2 1 * 5 – 4 1
Notasi postfiks : 2 3 * 2 1 – / 5 4 1 – * +

2. Algoritma Pembangun Pohon Ekspresi

Algoritma pembangun pohon ekspresi biner menurut notasi yang "dibaca" dikelompokkan menjadi dua. Algoritma yang membangun pohon ekspresi biner dari notasi prefiks dan algoritma yang membangun pohon ekspresi biner dari notasi postfiks, sementara notasi infiks tidak memiliki algoritma yang dapat langsung mengubahnya menjadi pohon ekspresi biner.

2.1 Algoritma Pembangun Pohon Ekspresi dari Notasi Prefiks

Berikut ini suatu prosedur (*procedure*) yang membangun sebuah pohon ekspresi dari notasi prefiks yang diberikan, dengan memanfaatkan tumpukan (*stack*).

```
procedure FromPrefiks
(
  input P1, P2, …, Pn : elemen prefiks,
  output T :  pohon
)
{
  Membangun pohon ekspresi dari notasi prefiks.
  Masukan : notasi prefiks, setiap elemennya disimpan di dalam tabel P
  Keluaran : pohon ekspresi T
}
Deklarasi
    i          : integer
    S          : tumpukan
    T1, T2     : pohon

Algoritma
    for i ← 1 to n do
        if P(n-i+1) = operand then
            createNode ( P(n-i+1)   T )
            push ( S   T )
        else  {P(n-i+1) = operator}
            pop ( S   T1 )
            pop ( S   T2 )
            createTree ( T1   T2   T )
            push  ( S   T)
        endif
    endfor
```

2.2 Algoritma Pembangun Pohon Eksprsi dari Notasi Postfiks

Berikut ini suatu prosedur yang membangun sebuah pohon ekspresi dari notasi postfiks yang diberikan, dengan memanfaatkan tumpukan (*stack*).

```
procedure FromPostfiks
(
  input P1, P2, …, Pn : elemen postfiks,
  output T :  pohon
)
{
  Membangun pohon ekspresi dari notasi postfiks.
  Masukan : notasi postfiks, setiap elemennya disimpan di dalam tabel P
  Keluaran : pohon ekspresi T
}
Deklarasi
    i          : integer
    S          : tumpukan
    T1, T2     : pohon

Algoritma
    for i ← 1 to n do
        for i ← 1 to n do
        if P(i) = operand then
            createNode ( P(n-i+1)   T )
            push ( S   T )
        else  {P(i) = operator}
            pop ( S   T1 )
            pop ( S   T2 )
            createTree ( T2   T1   T )
            push  ( S   T)
        endif
    endfor
```

3. Analisis Algoritma Pembangun Pohon Ekspresi

Setelah menetapkan kedua algoritma pembangun pohon ekspresi di bagian 2, pada bagian 3 ini akan dilakukan analisis mendalam terhadap algoritma tersebut.

Berikut ini beberapa definisi dan spesifikasi algoritma yang digunakan "*procedure FromPrefiks*" dan "*procedure FromPostfiks*" :

```
procedure createNode
(
  input P          : elemen notasi
  output T         : pohon (node-nya)
)
{
  membuat simpul pohon T dari elemen notasi P
}

procedure push
(
  input T          : pohon
  output S         : tumpukan (stack)
)
{
  memasukkan pohon T ke dalam tumpukan S
}

procedure pop
(
  input S          : tumpukan (stack)
  output T         : pohon
)
{
  mengeluarkan pohon T dari dalam tumpukan S
}

procedure createTree
(
  input  T1  T2    : pohon
  output T         : pohon
)
{
  membuat pohon T dari T1 dan T2
}
```

3.1 Analisis "*procedure FromPrefiks*"

Algoritma pada bagian 2.1 menerima masukan dalam notasi prefiks, dengan n adalah jumlah elemen notasi prefiks (jumlah *operand* + jumlah *operator*).

Cara kerja "*procedure FromPrefiks*" adalah dengan memeriksa elemen $P_{(n-i+1)}$ dari notasi prefiks.

Jika elemen $P_{(n-i+1)}$ adalah *operand*, maka dijalankan *createNode*, lalu dilanjutkan dengan *push* sehingga elemen $P_{(n-i+1)}$ yang telah diubah menjadi simpul masuk ke dalam tumpukan.

Jika elemen $P_{(n-i+1)}$ bukan *operand* (*operator*), maka dijalankan *pop* sebanyak dua kali untuk mengambil T_1 dan T_2 (T_1 adalah hasil *pop* yang pertama dan T_2 adalah hasil *pop* yang kedua). Selanjutnya dijalankan *createTree* dengan T_2 sebagai upapohon kanan dan T_1 sebagai upapohon kiri, pohon T hasil *createTree* di-*push* ke dalam tumpukan.

Setelah memproses P_1 (saat i = 1), maka "*procedure FromPrefiks*" selesai.

3.2 Analisis "*procedure FromPostfiks*"

Algoritma pada bagian 2.2 menerima masukan dalam notasi postfiks, dengan n adalah jumlah elemen notasi postfiks (jumlah *operand* + jumlah *operator*).

Cara kerja "*procedure FromPostfiks*" adalah dengan memeriksa elemen P_i dari notasi postfiks.

Jika elemen P_i adalah *operand*, maka dijalankan *createNode*, lalu dilanjutkan dengan *push* sehingga elemen P_i yang telah diubah menjadi simpul masuk ke dalam tumpukan.

Jika elemen P_i bukan *operand* (*operator*), maka dijalankan *pop* sebanyak dua kali untuk mengambil T_1 dan T_2 (T_1 adalah hasil *pop* yang pertama dan T_2 adalah hasil *pop* yang kedua). Selanjutnya dijalankan *createTree* dengan T_1 sebagai upapohon kanan dan T_2 sebagai upapohon kiri, pohon T hasil *createTree* di-*push* ke dalam tumpukan.

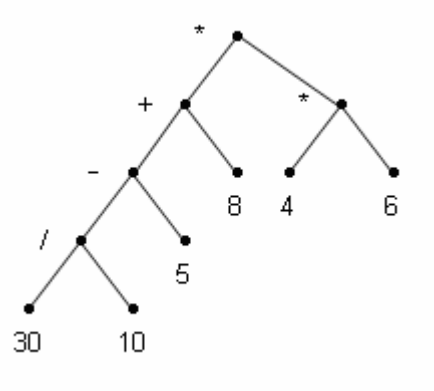
Setelah memproses P_n (saat i = n), maka "*procedure FromPostfiks*" selesai.

4. Pengujian Algoritma Pembangun Pohon Ekspresi

Setelah menentukan algoritma untuk membangun pohon ekspresi, baik dari notasi prefiks maupun notasi postfiks, selanjutnya akan dilakukan suatu uji kasus.

4.1 Pengujian Algoritma Pembangun Pohon Eksprsi dari Notasi Prefiks

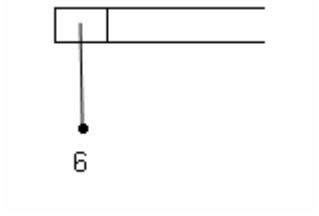
Untuk membuktikan kebenaran algoritma pembangun pohon ekspresi dari notasi prefiks yang telah ditentukan pada bagian 2.1, akan dibangun sebuah pohon ekspresi dari notasi prefiks yang tersedia. Sebelumnya didefinisikan sebuah pohon ekspresi awal untuk diambil notasi prefiks-nya.



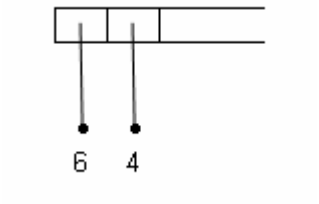
Dengan menelusuri gambar di atas, didapatkan notasi prefiks : * + - / 30 10 5 8 * 4 6

Penyelesaian :

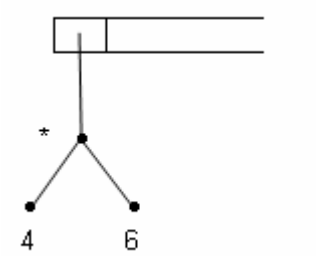
(i) Mulai dari elemen prefiks terakhir, baca P_{11}, P_{11} = 6. Karena P_{11} *operand*, buat simpul untuk P_{11}, *push pointer* P_{11} ke dalam tumpukan.



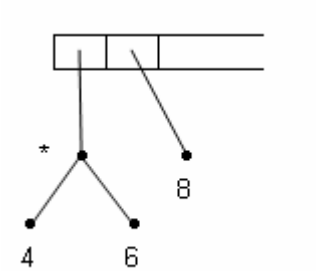
(ii) Baca P_{10}, P_{10} = 4. Karena P_{10} *operand*, buat simpul untuk P_{10}, *push pointer* P_{10} ke dalam tumpukan.



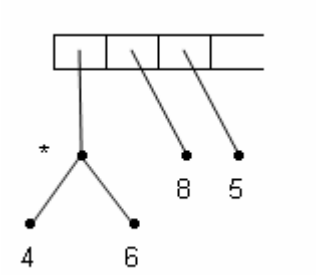
(iii) Baca P_9, P_9 = *. Karena P_9 *operator*, *pop pointer* P_{10} dan nyatakan sebagai T_1, *pop pointer* P_{11} dan nyatakan sebagai T_2, Buat pohon dengan T_1 sebagai upapohon kiri dan T_2 sebagai upapohon kanan. *Push pointer* pohon ke dalam tumpukan.



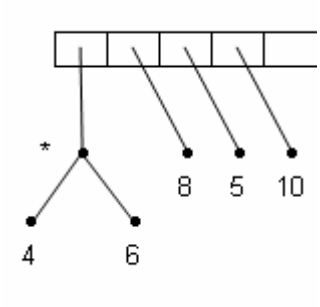
(iv) Baca P_8, P_8 = 8. Karena P_8 *operand*, buat simpul untuk P_8, *push pointer* P_8 ke dalam tumpukan.



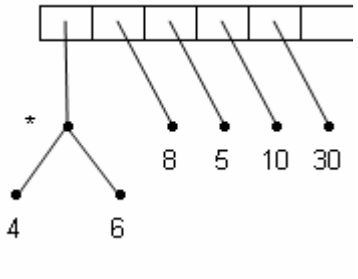
(v) Baca P_7, P_7 = 5. Karena P_7 *operand*, buat simpul untuk P_7, *push pointer* P_7 ke dalam tumpukan.



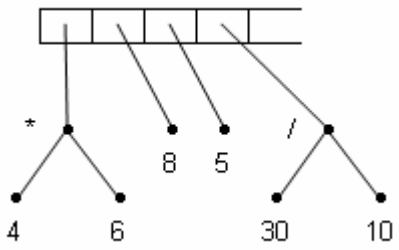
(vi) Baca P_6, P_6 = 10. Karena P_6 *operand*, buat simpul untuk P_6, *push pointer* P_6 ke dalam tumpukan.



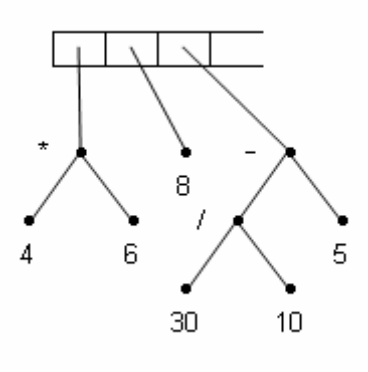
(vii) Baca P_5, $P_5 = 30$. Karena P_5 *operand*, buat simpul untuk P_5, *push pointer* P_5 ke dalam tumpukan.



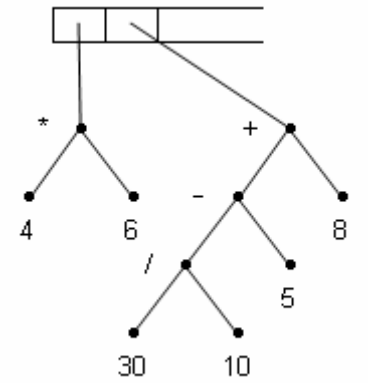
(viii) Baca P_4, $P_4 = /$. Karena P_4 *operator*, *pop pointer* P_5 dan nyatakan sebagai T_1, *pop pointer* P_6 dan nyatakan sebagai T_2, Buat pohon dengan T_1 sebagai upapohon kiri dan T_2 sebagai upapohon kanan. *Push pointer* pohon ke dalam tumpukan.



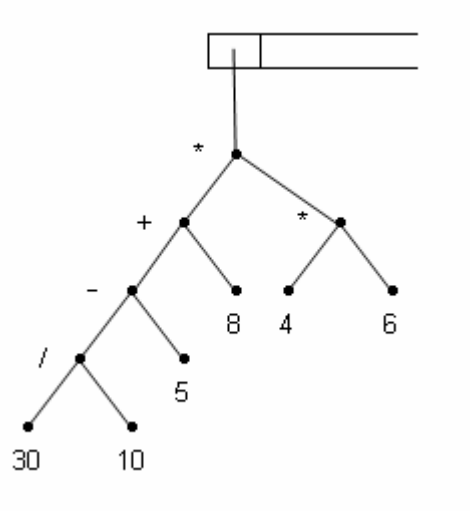
(ix) Baca P_3, $P_3 = -$. Karena P_3 *operator*, *pop pointer* pohon sebelumnya dan nyatakan sebagai T_1, *pop pointer* P_7 dan nyatakan sebagai T_2, Buat pohon dengan T_1 sebagai upapohon kiri dan T_2 sebagai upapohon kanan. *Push pointer* pohon ke dalam tumpukan.



(x) Baca P_2, $P_2 = +$. Karena P_2 *operator*, *pop pointer* pohon sebelumnya dan nyatakan sebagai T_1, *pop pointer* P_8 dan nyatakan sebagai T_2, Buat pohon dengan T_1 sebagai upapohon kiri dan T_2 sebagai upapohon kanan. *Push pointer* pohon ke dalam tumpukan.



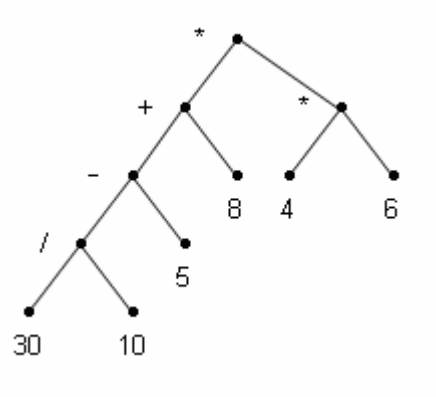
(xi) Baca P_1, $P_1 = *$. Karena P_1 *operator*, *pop pointer* kedua pohon sebelumnya, nyatakan sebagai T_1 dan T_2, Buat pohon dengan T_1 sebagai upapohon kiri dan T_2 sebagai upapohon kanan. *Push pointer* pohon T ke dalam tumpukan.



Pohon telah berhasil dibangun dari notasi prefiks * + - / 30 10 5 8 * 4 6. Daun-daun dari pohon ekspresi di atas terdiri dari *operand* 30, 10, 5, 8, 4, dan 6. Pohon di atas sama persis dengan pohon yang didefinisikan di awal bagian 4.1 ini, dengan demikian dapat disimpulkan bahwa algoritma yang diuji berhasil membangun pohon ekspresi biner dari notasi prefiks.

4.2 Pengujian Algoritma Pembangun Pohon Eksprsi dari Notasi Postfiks

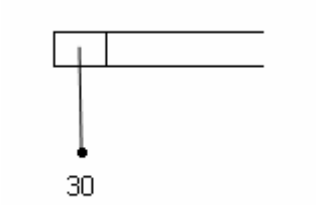
Untuk membuktikan kebenaran algoritma pembangun pohon ekspresi dari notasi postfiks yang telah ditentukan pada bagian 2.2, akan dibangun sebuah pohon ekspresi dari notasi postfiks yang tersedia. Sebelumnya didefinisikan sebuah pohon ekspresi awal untuk diambil notasi postfiks-nya.



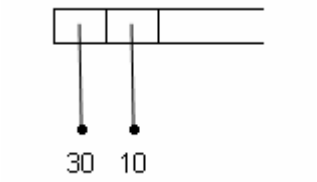
Dengan menelusuri gambar di atas, didapatkan notasi postfiks : 30 10 / 5 – 8 + 4 6 * *

Penyelesaian :

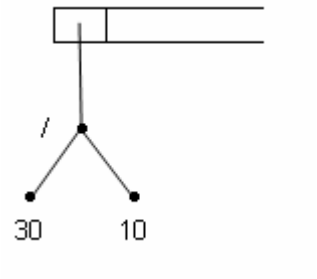
(i) Mulai dari elemen postfiks pertama, baca P_1, P_1 = 30. Karena P_1 *operand*, buat simpul untuk P_1, *push pointer* P_1 ke dalam tumpukan.



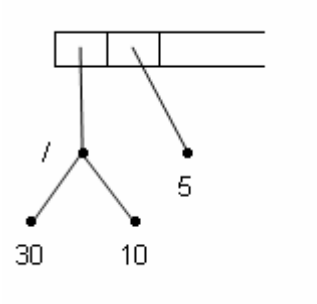
(ii) Baca P_2, P_2 = 10. Karena P_2 *operand*, buat simpul untuk P_2, *push pointer* P_2 ke dalam tumpukan.



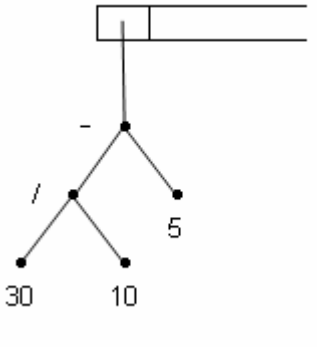
(iii) Baca P_3, P_3 = /. Karena P_3 *operator*, *pop pointer* P_2 dan nyatakan sebagai T_1, *pop pointer* P_1 dan nyatakan sebagai T_2, Buat pohon dengan T_1 sebagai upapohon kanan dan T_2 sebagai upapohon kiri. *Push pointer* pohon ke dalam tumpukan.



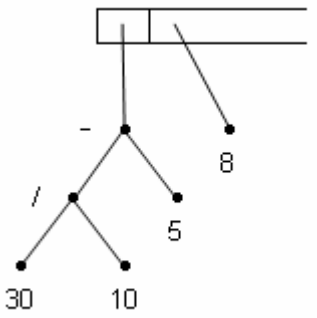
(iv) Baca P_4, P_4 = 5. Karena P_4 *operand*, buat simpul untuk P_4, *push pointer* P_4 ke dalam tumpukan.



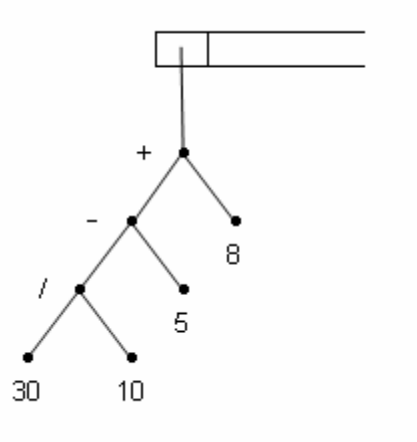
(v) Baca P_5, P_5 = -. Karena P_5 *operator*, *pop pointer* P_4 dan nyatakan sebagai T_1, *pop pointer* pohon sebelum P_4 dan nyatakan sebagai T_2, Buat pohon dengan T_1 sebagai upapohon kanan dan T_2 sebagai upapohon kiri. *Push pointer* pohon ke dalam tumpukan.



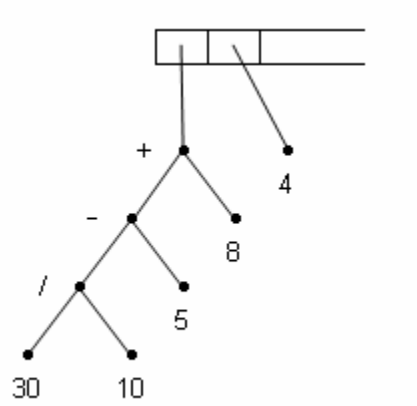
(vi) Baca P_6, P_6 = 8. Karena P_6 *operand*, buat simpul untuk P_6, *push pointer* P_6 ke dalam tumpukan.



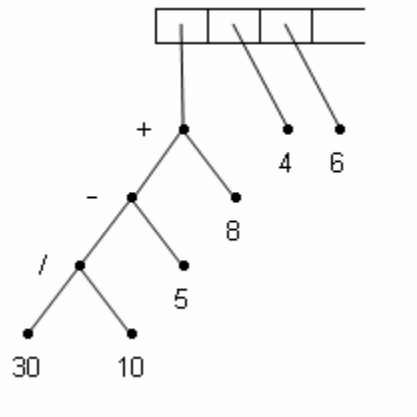
(vii) Baca P_7, P_7 = +. Karena P_7 *operator*, *pop pointer* P_6 dan nyatakan sebagai T_1, *pop pointer* pohon sebelum P_6 dan nyatakan sebagai T_2, Buat pohon dengan T_1 sebagai upapohon kanan dan T_2 sebagai upapohon kiri. *Push pointer* pohon ke dalam tumpukan.



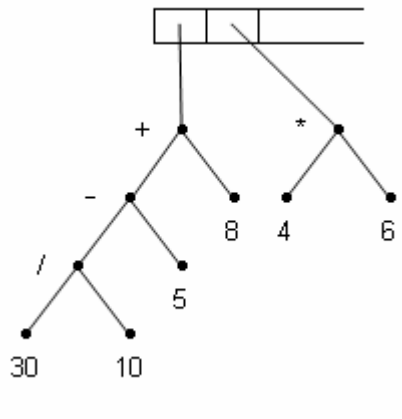
(viii) Baca P_8, P_8 = 4. Karena P_8 *operand*, buat simpul untuk P_8, *push pointer* P_8 ke dalam tumpukan.



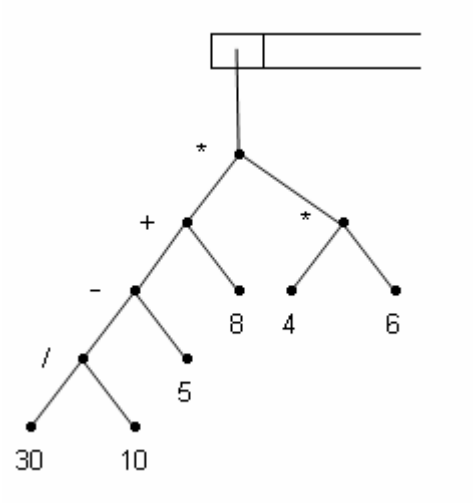
(ix) Baca P_9, P_9 = 6. Karena P_9 *operand*, buat simpul untuk P_9, *push pointer* P_9 ke dalam tumpukan.



(x) Baca P_{10}, P_{10} = *. Karena P_{10} *operator*, *pop pointer* P_9 dan nyatakan sebagai T_1, *pop pointer* P_8 dan nyatakan sebagai T_2, Buat pohon dengan T_1 sebagai upapohon kanan dan T_2 sebagai upapohon kiri. *Push pointer* pohon T ke dalam tumpukan.



(xi) Baca P_{11}, P_{11} = *. Karena P_{11} *operator*, *pop pointer* kedua pohon sebelumnya, nyatakan sebagai T_1 dan T_2. Buat pohon dengan T_1 sebagai upapohon kanan dan T_2 sebagai upapohon kiri. *Push pointer* pohon ke dalam tumpukan.



Pohon telah berhasil dibangun dari notasi prefiks 30 10 / 5 – 8 + 4 6 * *. Daun-daun dari pohon ekspresi di atas terdiri dari *operand* 30, 10, 5, 8, 4, dan 6. Pohon di atas sama persis dengan pohon yang didefinisikan di awal bagian 4.2 ini, dengan demikian dapat disimpulkan bahwa algoritma yang diuji berhasil membangun pohon ekspresi biner dari notasi postfiks.

5. Kesimpulan

Setelah menganalisis algoritma pembangun pohon ekspresi dari notasi prefiks dan postfiks pada bagian 3, serta mengamati pola pembentukan pohon dari uji kasus pada bagian 4, terlihat jelas persamaan maupun perbedaan dari kedua algoritma tersebut.

Persamaan :

- prosedur (*procedure*) bantuan yang digunakan prosedur pembangun pohon ekspresi
- langkah-langkah dalam algoritma

Perbedaan :

- pada algoritma pembentuk pohon ekspresi dari notasi prefiks, pemeriksaan dimulai dari elemen akhir notasi dan selesai setelah memeriksa elemen awal notasi. Pada algoritma pembentuk pohon ekspresi dari notasi postfiks, pemeriksaan dimulai dari elemen awal notasi dan selesai setelah memeriksa elemen akhir notasi
- pada algoritma pembentuk pohon ekspresi dari notasi prefiks, T_1 menjadi upapohon kiri dan T_2 menjadi upapohon kanan dari pohon T. Pada algoritma pembentuk pohon ekspresi dari notasi postfiks, T_1 menjadi upapohon kanan dan T_2 menjadi upapohon kiri dari pohon T.

Dengan membandingkan persamaan dan perbedaan dari kedua algoritma pembangun pohon ekspresi tersebut, dapat disimpulkan bahwa kemangkusan algoritma pembangun pohon ekspresi dari notasi prefiks dan algoritma pembangun pohon ekspresi dari notasi postfiks relatif sama.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. (2003). Diktat Kuliah IF2153 Matematika Diskrit. Program Studi Teknik Informatika, Institut Teknologi Bandung.

[2] Codepedia the developers encyclopedia. (2006). *http://www.codepedia.com/1/Art_Expressions_p1*. Tanggal akses: 24 Desember 2006 pukul 20:00.

[3] Wikipedia, the free encyclopedia. (2006). *http://en.wikipedia.org/wiki/Graph_theory*. Tanggal akses: 24 Desember 2006 pukul 20:00.

[4] Wikipedia, the free encyclopedia. (2006). *http://en.wikipedia.org/wiki/Polish_notation*. Tanggal akses: 24 Desember 2006 pukul 20:00.

[5] Wikipedia, the free encyclopedia (2006). *http://en.wikipedia.org/wiki/Reverse_Polish_notation*. Tanggal akses: 24 Desember 2006 pukul 20:00.