Pohon Biner dan Aplikasinya

Muhammad Gema Akbar (13510099)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
mgemaakbar@students.itb.ac.id

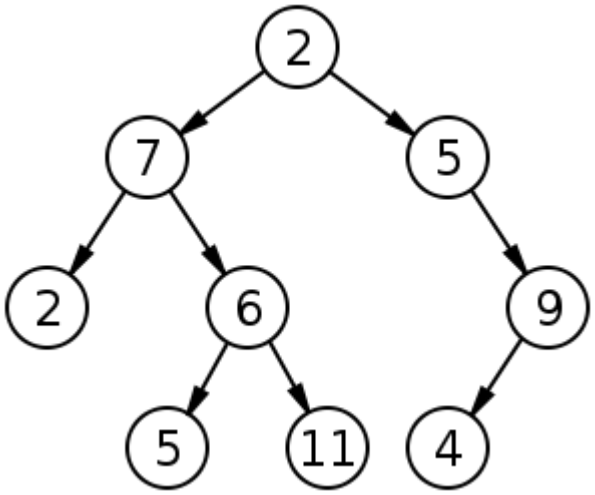
Abstract*—Makalah ini membahas tentang Pohon Biner, definisi istilah, sifat dan penggunaan praktikal dari pohon biner. Pohon biner adalah struktur data pohon dengan anak maksimum 2, beberapa aplikasi dari pohon biner yang dibahas dalam makalah ini adalah *Binary Search Tree, Binary Space Partition, Huffman Coding Tree, Heap, Syntax Tree.

Index Terms*—Binary Search Tree. Binary Space Partition, Graf, *Heap*, *Huffman Coding Tree*, Pohon, Pohon Biner, *Syntax Tree

I. PENDAHULUAN

Pohon biner adalah jenis dari struktur data pohon dimana setiap node hanya bisa paling banyak memiliki 2 anak node. Seperti struktur data lainnya, pohon digunakan untuk menyimpan informasi. Tidak seperti struktur data *Stack* dan *Queue*, yang keduanya adalah struktur data linear, pohon (biner) adalah struktur data hirarkis. Untuk mengatakan bahwa struktur data pohon hirarkis berarti elemen pohon terurut diatas atau dibawah elemen lainnya.

II. DEFINISI KOSAKATA DAN ISTILAH POHON BINER



Gambar 2.1 Contoh Pohon Biner

Anak (child/children) dan Orangtua (*parent*) adalah istilah yang menunjukan bahwa pohon biner adalah struktur data hirarkis, dalam Gambar 2.1 (contoh pohon biner) 7 dan 5 adalah anak dari 5, dan 5 adalah orang tua untuk 5.

Derajat (*degree*) ada perbedaan antara arti derajat pada pohon berakar dan derajat pada graf, derajat sebuah simpul pada pohon berakar adalah jumlah subpohon (atau jumlah anak) pada simpul tersebut. Pada pohon biner, derajat maksimum adalah 2.

Lintasan (*Path*) dari simpul v_1 ke simpul v_k adalah sekuens simpul-simpul $v_1,v_2,....v$ k sedemikian sehingga v_i adalah orangtua dari v_{i+1} untuk 1<=i<k.

Daun (*Leaf*) simpul yang memiliki derajat dapat disebut daun. Pada gambar 2.1 misalnya adalah 5, 11, 4, dan 2.

Aras (*level*) atau tingkat adalah seperti namanya, istilah yang menjukan tinggi suatu pohon. Akar mempunyai aras samadengan 0. Pohon pada gambar 2.1 memiliki aras terbesar 3.

Saudara kandung (*sibling*) adalah simpul yang memiliki orangtua sama. Contoh pada gambar 2.1 5 adalah *sibling* dari 7 (dan begitu sebaliknya) karena memiliki orangtua sama yaitu 2.

III. JENIS-JENIS POHON BINER DAN SIFATNYA

Rooted Binary Tree adalah pohon dengan simpul akar dimana setiap simpul paling banyak hanya bisa memiliki dua anak.

Full Binary Tree adalah pohon dimana setiap simpulnya kecuali daun memiliki 2 simpul.

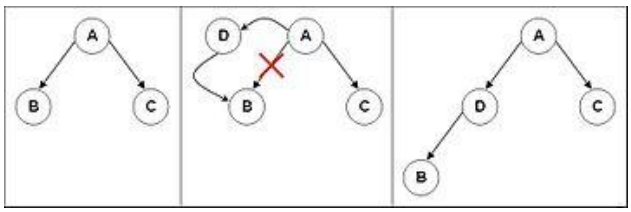
Balanced Binary Tree adalah pohon biner yang tinggi dari 2 subpohon dari setiap simpul tidak pernah lebih dari

Sifat-sifat dari pohon biner:

- Jumlah *n* simpul dari *full binary tree* adalah *n* = 2h+1 -1 dimana h adalah tinggi dari pohon.
- Jumlah simpul minimum dari pohon biner adalah $n = 2^h$ dan paling banyak 2^{h+1} -1 dimana *h* adalah tinggi dari pohon.
- Jumlah daun pada *full binary tree* (pohon biner lengkap) adalah 2^h dimana *h* adalah tinggi pohon.

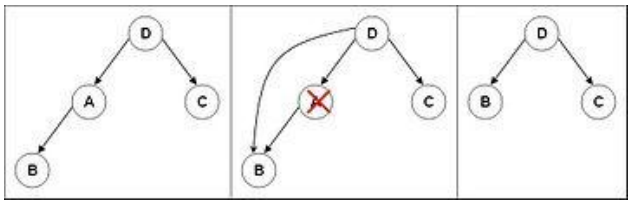
IV. OPERASI PADA POHON BINER

***Insertion* (penyisipan)** adalah penyisipan elemen pada pohon biner antara dua simpul atau penambahan pada simpul eksternal.



Gambar 4.1 Insertion/peenyisipan elemen pada pohon biner.

***Deletion* (penghapusan)** adalah penghapusan elemen dari pohon.



Gambar 4.2 Deletion/penghapusan elemen pada pohon biner.

Iterasi/*traversal* adalah proses pengunjungan elemen pohon. Ada tiga cara, yaitu dengan cara *Pre-Order*, *In-Order*, dan *Post-Order* . Cara *Post-Order* : anak kiri, anak kanan, akar. Cara *In-Order* : Anak kiri, akar, anak kanan. Cara *Pre-Order*: Akar, Anak kiri, anak kanan.

Depth-first order adalah algoritma pencarian pada pohon yang memiliki cara berawal dari akar, lalu menjelajahi sejauh mungkin setiap ranting sebelum mem-*backtrack*.

```
1  procedure DFS(G,v):
2      label v as explored
3      for all edges e in
G.incidentEdges(v) do
4          if edge e is unexplored
then
5              w ← G.opposite(v,e)
6              if vertex w is
unexplored then
7                  label e as a
discovery edge
8                  recursively call
DFS(G,w)
9          else
10             label e as a back edge
```

Gambar 4.3 Pseudocode/notasi algoritmik untuk Depth-first order

Breadth-first order adalah algoritma pencarian yang berawal dari simpul akar, lalu menjelajahi simpul terdekat, dan pada setiap simpul yang dikunjungi, diulangi langkah yang sama sampai ditemukan hasil pencarian.

```
1  procedure BFS(Graph,v):
2      create a queue Q
3      enqueue v onto Q
4      mark v
5      while Q is not empty:
6          t ← Q.dequeue()
7          for all edges e in
G.incidentEdges(t) do
8              o ← G.opposite(t,e)
9              if o is not marked:
10                 mark o
11                 enqueue o onto Q
```

Gambar 4.4 Pseudocode/notasi algoritmik utnuk Breadth-first order

V. APLIKASI POHON BINER

V.I Binary Search Tree

Dalam sains komputer, *Binary Search Tree* yang disebut juga pohon biner terurut (*ordered binary tree*) adalah pohon yang memiliki sifat:

- Anak pohon kiri memiliki nilai yang lebih kecil dari orang tuanya
- Anak pohon kanan memiliki nilai yang lebih besar dari orangtuanya
- Anak kiri dan kanan adalah *binary search tree* juga.

Keunggulan utama dari BST dibanding struktur data lainnya adalah efisiensinya dalam menjalankan algoritma pencarian dan algoritma pengurutan. BST adalah struktur data pokok yang dapat digunakan dalam pembentukan struktur data abstrak (ADT) lainnya seperti himpunan, multiset.

V.1.1 Operasi pada *Binary Search Tree*

V.1.1.1 *Searching*/Pencarian

Pencarian pada BST dapat dilakukan dengan cara rekursif atau iteratif.

Langkah –langkah pencarian:

1. Periksa apakah nilai pada simpul yang sekarang sedang diperiksa sesuai dengan yang dicari. Jika ya maka nilai ditemukan, jika tidak:
2. Jika nilai yang dicari lebih kecil daripada nilai yang sedang diperiksa sekarang: jika simpul yang sekarang sedang diperiksa tidak memiliki anak

kiri, maka **nilai yang dicari tidak ada pada BST.**

Jika ada, ulangi langkah yang sama pada anak kiri.

3. Jika nilai yang dicari lebih besar daripada nilai yang sedang diperiksa sekarang , periksa anak kanan. Jika simpul yang sedang diperiksa tidak punya anak kanan maka **nilai yang dicari tidak ada dalam BST.** Jika punya, ulangi lankah yang sama pada anak kanan.

```
bool BinarySearchTree::search(int val)
{
    Node *next = this->root();

    while (next != NULL) {
        if (val == next->value()) {
            return true;
        } else if (val < next-
>value()) {
            next = next->left();
        } else {
            next = next->right();
        }
    }

    //not found
    return false;
}
```

Gambar 5.1 Algoritma Pencarian Pada Binary Search Tree

V.I.II *Insertion*/Penyisipan

Awal dari cara kerja penyisipan sama dengan pencarian, jika akar yang sedang diperiksa tidak sama dengan nilai yang ingin disisipi, kunjungi anak pohon kiri atau kanan hingga akhirnya kita menemukan simpul eksternal dan menyisipkan nilai sebagai anak kiri atau kanan—bergantung dari nilai.

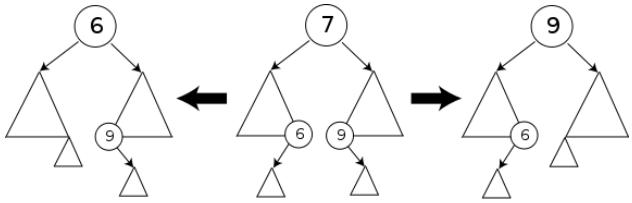
```
void InsertNode(Node* &treeNode, Node
*newNode)
 {
     if (treeNode == NULL)
       treeNode = newNode;
     else if (newNode->key < treeNode-
>key)
       InsertNode(treeNode->left,
newNode);
     else
       InsertNode(treeNode->right,
newNode);
 }
```

Gambar 5.2 Algoritma Penysipan pada Binary Search Tree

V.I.III *Deletion*/Penghapusan

Terdapat 3 kasus yang mungkin dan perlu diatasi dalam penghapusan elemen pohon BST:

- Menghapus simpul yang merupakan daun
- Menghapus simpul dengan 1 anak
- Menghapus simpul dengan 2 anak

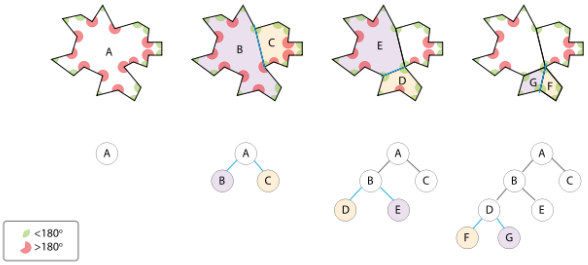


Gambar 5.3 Skema penghapusan pada binary search tree.

V.II Binary Space Partitioning

Dalam grafika komputer untuk menggambar dengan cepat dan benar dapat digunakan *Painter's algorithm. Z-buffering* dapat memastikan gambar tergambar dengan benar, tapi memakan banyak memori. Pohon BSP dapat membagi objek agar *painter's algorithm* dapat menggambarnya tanpa butuh *Z-buffer*.

Pohon BSP banyak digunakan dalam *video game komputer*.



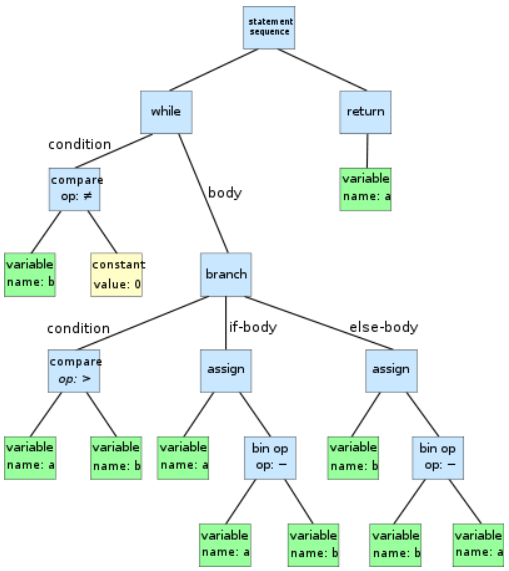
Gambar 5.4 Ilustrasi proses partisi sebuh poligon tidak beraturan menjadi bagian-bagian konveks oleh BSP.

V.III Syntax Tree

Syntax Tree adalah representasi pohon dari struktur sintaksis dari kode yang ditulis dengan bahasa pemrograman. Setiap simpul dari pohon mewakili sebuah kontruk yang terjadi dalam kode.

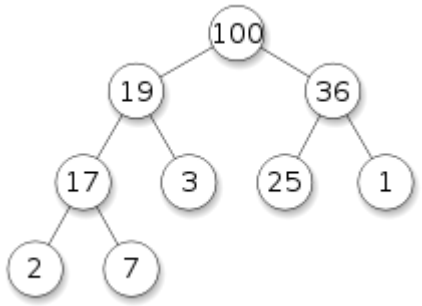
Syntax Tree untuk:

```
while b ≠ 0
if a > b
a := a − b
else
b := b − a
return a
```



Gambar 5.4 Syntax Tree

V.IV Heap



Gambar 5.5 Heap biner lengkap dengan nilai terbesar pada simpul akarnya (max-heap).

Heap adalah struktur data yang berdasar dari pohon biner, memiliki sifat jika *B* adalah anak simpul dari *A* maka kunci(A)=>kunci(B). Ini berarti elemen terbesar pasti terdapat pada simpul akar, atau elemen terkecil pasti pada simpul akar.

Heap memiliki banyak penggunaan, misalnya pada *Heapsort*, *selection algorithm*, dan algoritma graf.

Algoritma graf seprti algoritma Prim, algoritma Dijkstra, dapat dikurangi *runtime* nya dengan menerapkan stuktur data *heap* hingga orde polinomial.

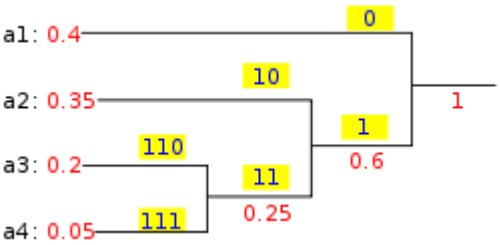
Algoritma seleksi, pencarian nilai terbesar atau terkecil atau median, dapat dieksekusi dengan waktu linear jika *heap* digunakan.

V.V Huffman coding

Huffman coding digunakan dalam pemampatan/*compression* data. Huffman coding menggunakan cara spesifik dalam memilih representasi untuk setiap simbol, yang menghasilkan kode prefix. Kode prefix adalah *bit string* yang merepresenasikan simbol tertentu. String yang lebih pendek menandakan bahwa simbol yang direpresentasikan jarang digunakan, sementara string yang lebih panjang menandakan bahwa simbol sering muncul.

Cara mendapatkan kode Huffman

Jika simbol $\{a_1,a_2,a_3,a_4\}$ memilki peluang kemuncukan dalam suatu string {0.4;0.35;0.2;0.05} maka untuk mendapatkan kode huffman nya, dibangun pohon biner



Gambar 5.6 Pohon biner untuk mendapatkan huffman code

Pohon biner dibangun dengan cara mencari 2 simbol dengan peluang kemunculan paling kecil terlebih dahulu, dalam kasus ini adalah a3 dan a4. A3 memiliki peluang 0.2, a4 memiliki peluang 0.05. Tandai bagian kiri dengan 0, bagian anak kanan dengan 1. A3 sebagai anak kiri dan a4 sebagai anak kanan. Orangtua yang dibentuk adalah string gabungan dari a3 dan a4 yang memiliki peluang kemunculan sebsar jumlah dari kedua peluang a3 dan a4. Lalu ulangi langkah penentuan string yang memiliki peluang paling kecil (selai a3 dan a4 tentunya) dan ulangi langkah penggabungan sampai diperoleh gabungan string dengan peluang 1.

VI. KESIMPULAN

Pohon biner diturunkan dari struktur data lain yaitu pohon, yang diturunkan dari graf. Pada dasarnya pohon biner adalah pohon dengan anak maksimum 2.

Dalam sains komputer, pohon biner memiliki banyak aplikasi, seperti *binary search tree*, *binary space partitioning* dalam grafika komputer untuk *video game* 3D , pembentukan struktur data lainnya, kompresi dengan kode Huffman, *syntax tree* dalam *compiler*.

REFERENCES

[1] En.wikipedia.org/wiki/Binary_tree Minggu, 12 Desember 2011 18.32
[2] En.wikipedia.org/wiki/Binary_search_tree Minggu, 12 Desember 2011 18.32
[3] En.wikipedia.org/wiki/Huffman_coding, Minggu, 12 Desember 2011 18.32
[4] En.wikipedia.org/wiki/Syntax_tree, Minggu, 12 Desember 2011
[5] En.wikipedia.org/wiki/Binary_space_partitioning, Minggu, 12 Desember 2011 18.32

[6] En.wikipedia.org/wiki/Heap, Minggu, 12 Desember 2011 18.32
[7] En.wikipedia.org/wiki/Heapsort, Minggu, 12 Desember 2011 18.32

[8] http://maven.smith.edu/~mcharley/bsp/, Minggu, 12 Desember 2011 18.33

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, 29 April 2010

ttd

Muhammad Gema Akbar 13510099