Implementasi Binary Search Tree pada Struktur Data Priority Queue dalam Bahasa C

Muhammad Hanif Adzkiya / 13517120
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517120@std.stei.itb.ac.id

***Abstrak* — Pohon merupakan salah satu lingkup materi dari matematika diskirit. Ada beragam jenis pohon dengan kegunaannya masing-masing. Salah satu jenis dari pohon adalah *binary search tree. Binary search tree* memiliki banyak manfaat dalam penyelesaian masalah komputasional. *Binary search tree* dianggap sebagai suatu solusi untuk mengefesienkan suatu algoritma, salah satunya adalah *priority queue*. Hal ini dikarenakan pada di kedua struktur data merupakan struktur data yang memiliki elemen terususun secara terstruktur. *Priority queue* merupakan struktur data antrian terurut. Pada makalah ini akan dibahas penerapan binary tree untuk membentuk struktur data antrian terurut, *priority queue*.**

***Kata Kunci* —*Binary Search Tree*, Pohon, *Priority Queue*, Struktur Data**

I. PENDAHULUAN

Sebuah data pada algoritma pemrograman memerlukan suatu tipe data yang dapat menyimpan data tersebut. Secara klasik, dikenal beragam tipe data primitif seperti integer, real, char, dan boolean. Namun, tipe data primitif tidak cukup untuk menyelesaikan semua permasalahan komputasi. Seiring dengan kemajuan teknologi komputasi, penggunaan tipe data tak hanya semakin beragam namun memerlukan struktur data yang tepat dan efisien. Penggunaan struktur data yang efisien bermanfaat ketika menggunakan struktur data dengan elemen yang banyak.

Salah satu strukur data dalam komputasional adalah struktur data antrian. Struktur data antrian memiliki sifat seperti antrian pada umumnya. Terkadang pemrosesan antrian dilakukan dengan menggunakan prioritas tertentu. Salah satu contoh praktek dalam kehidupan sehari – hari adalah pengantrean pada wahana permainan seperti *Disneyland*. Pengunjung yang membayar lebih besar berhak masuk ke wahana lebih dahulu. Praktek ini dapat menggunakan antrian yang memiliki keterututan. Dalam ilmu struktur data, antrean ini dinamakan dengan priority queue.

Terdapat banyak cara dalam pengimplementasian suatu struktur data. Struktur data dapat diimplementasikan secara naif menggunakan definisi suatu struktur data sebagai dasar pengaplikasian. Namun untuk penggunaan dalam lingkup yang luas, pengimplementasian menggunakan cara yang naif tidak seluruhnya tepat. Bentuk pengimplementasian lain adalah dengan memanfaatkan struktur data lain yang dimodifikasi sebagai bentuk implementasi suatu struktur data.

Struktur data priority queue memiliki sifat yang sama dengan struktur data binary search tree yang sama – sama memiliki keterurutan. Pada prosesnya, operasi pada struktur data binary tree seperti pencarian, penyisipan, dan penghapusan dapat diterapkan pada struktur data priority queue.

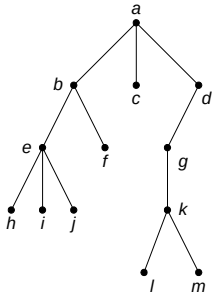
Oleh karena itu, pada makalah ini, akan dibahas bagaimana mengimplementasikan struktur data priority queue dengan bantun struktur data lain yaitu binary search tree.

II. LANDASAN TEORI

2.1 Pohon

2.1.1 Definisi

Pohon merupakan struktur data yang terdapat dalam informatika. Struktur data pohon menyerupai pohon pada umumnya. Setiap pohon memiliki sebuah akar yang terhubung dengan simpul – simpul dan memiliki daun sebagai simpul terakhir pada pohon.



Gambar 2.1. Terminologi Pohon

2.1.2 Terminologi

Terminologi merupakan istilah – istilah yang sering dipakai dalam membuat pohon. Berikut terminologi yang umum digunakan dalam struktur data pohon pada gambar 2.1 :

A. Simpul

Simpul merupakan bagian dari pohon yang memiliki sebuah nilai. Setiap simpul dapat terhubung dengan simpul lainnya. Pada gambar 2.1 memiliki 13 buah simpul, yaitu simpul a, b, c, d, e, f, g, h, i, j, k, l,

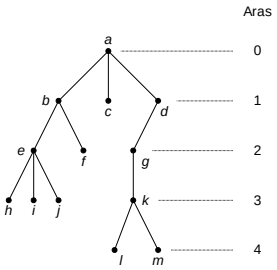
dan m.

B. Lintasan

Lintasan merupakan jalan yang menghubungkan antar 2 simpul. Dua buah simpul x dan y dikatakan terhubung apabila ada sebuah lintasan yang menghubungkan antara simpul x dan simpul y. Panjang lintasan merupakan banyaknya simpul yang dilewati untuk perjalanan dari simpul x menuju simpul y. Pada gambar 2.1. panjang lintasan dari a ke i adalah 3 karena dari simpul a menuju simpul i, harus melewati simpul b, e, dan i

.

C. Aras / Tingkat



Gambar 2.2. Aras / Tingkat pada Pohon

Aras / Tingkat merupakan level dari sebuah simpul dihitung dari simpul teratas. Pada gambar 2.2, simpul a,b, dan e memiliki aras sebesar 0,1, dan 2.

D. Simpul Anak dan Orangtua

Sebuah simpul x merupakan simpul anak dari y jika ada jarak lintasan dari a ke b adalah 1 dan simpul x memiliki tingkat 1 lebih tinggi dari pada simpul y. Sebaliknya,. Simpul y dikatakan sebagai orang tua dari simpul x.

E. Akar

Akar merupakan simpul yang memiliki tingkat bernilai 0.

.

F. Daun

Daun merupakan semua simpul yang berada paling bawah dan tidak memiliki simpul dibawahnya. Pada gambar 2.1 memiliki daun sebanyak 7, yaitu simpul h, i, j, f, c, l, dan m.

G. Simpul dalam

Simpul dalam merupakan simpul yang memiliki anak.

H. Derajat

Derajat dari simpul x menyatakan banyaknya anak dari dari simpul x. Sebagai contoh, pada gambar 2.1, simpul b memiliki derajat 2, simpul e memiliki derajat 3, dan simpul m memiliki derajat 0. Derajat maksimal adalah derajat terbesar yang dimiliki himpunan simpul – simpul yang membentuk sebuah pohon. Gambar 2.1 memiliki derajat maksimal sebanyak 3.

I. Tinggi / Kedalaman

Aras / Tingkat maksimum dari sebuah pohon dinamakan tinggi / kedalam. Kedalaman dari pohon pada gambar 2.1 adalah 5.

2.1.3 Pohon m-ary

Pohon m-ary merupakan pohon yang setiap simpul dalamnya memiliki tepat m anak.

2.2 Binary Tree

2.2.1 Definisi

Dalam terminologi informatika, Binary tree merupakan struktur data pohon yang setiap simpul dalamnya memiliki tepat 2 buat simpul anak atau dinamakan dengan pohon 2-ary.

2.3 Binary Search Tree

2.3.1 Definisi

Dalam lingkup informatika, Binary Search Tree (BST) merupakan binary tree yang simpul – simpulnya memiliki keterurutan.

Setiap simpul dalam x pada BST memiliki maksimal 2 buah simpul anak, dimana simpul anak kiri memiliki nilai lebih rendah dari simpul x dan simpul anak kanan memiliki nilai lebih tinggi dari simpul x.

2.3.2 Operasi Dasar

Sebagai sebuah struk data, BST memiliki operasi – operasi yang dapat dilakukan terhadap elemen – elemen di dalamnya. Operasi tersebut adalah :

A. Insertion

Insertion adalah proses penyisipan sebuah elemen pada struktur data BST. Langkah – Langkah dalam melakukan penyisipan pada BST adalah :

1. Jika simpul kosong, sisipkan nilai pada simpul tersebut
2. Jika nilai kurang dari nilai pada simpul maka telusuri simpul anak kiri dari simpul tersebut,
3. Jika nilai lebih besar dari nilai pada simpul maka telusuri simpul anak kanan dari simpul tersbut,

B. Deletion

Saat menghapus simpul dari struktur binary search tree, diperhatikan urutan urutan simpul agar pohon masih memiliki urutan sesuai aturan BST. Sebuah metode dari T. Hibbard membagi proses penghapusan elemen menjadi 3 kemungkinan kasus :

1. Menghapus daun

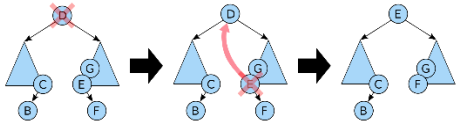
   Menghapus daun cukup dengan menghapus daun dari BST.

2. Menghapus simpul dengan satu anak

   Penghapusan dilakukan dengan menghapus

dan mengganti posisi simpul tersebut dengan anaknya.

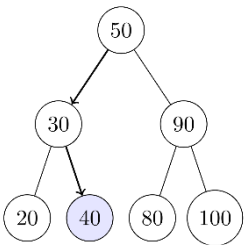
3. Menghapus simpul dengan dua anak



Gambar 2.3. Penghapusan Simpul dengan 2 Anak

Menghapus simpul dengan 2 anak dilakukan dengan memanggil simpul yang akan dihapus, misalkan simpul D. Jangan hapus D. Sebagai gantinya, pilih simpul prediksinya di urutan awal atau simpul pengganti sebagai simpul pengganti E . Salin nilai-nilai simpul dari E ke D. Jika E tidak memiliki anak, cukup menghapus E dari orangtua sebelumnya, G. Jika E memiliki seorang anak, katakanlah F, Ganti E dengan F pada orangtua E.

C. Searching



Gambar 2.4. Pencarian pada BST

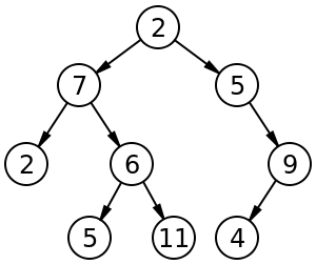
Proses *Searching* / Pencarian selalu dimulai dari simpul yang merupakan akar BST. Jika nilai pada simpul sama dengan nilai yang akan dicari, maka keluarkan nilai benar. Jika tidak, proses pencarian dilakukan ke arah anak kiri atau kanan dari simpul tergantung dari hasil perbandingan. Jika nilai yang akan dicari lebih besar dari nilai simpul, maka pencarian dilanjutkan ke anak kanan dari simpul, sebaliknya pencarian akan dilanjutkan ke anak kiri dari simpul. Jika pencarian telah menemui simpul kosong, maka nilai yang dicari tidak terdapat pada BST.

2.3.3 Representasi

Representasi dari sebuah struktur data pada algoritma pemrograman merupakan cara untuk mengekspresikan struktur data agar bisa dibaca oleh Bahasa pemrograman.

Binary Search Tree dapat direpresentasikan dengan bentuk *pointer* dan *array*.

A. Pointer



Gambar 2.5. Pohon Binary Search Tree

Setiap simpul pada BST akan direpresntasikan menggunakan tipe bentukan struct. Setiap simpul menyimpan 3 buah nilai. Nilai tersebut adalah :

1. Kiri

   Nilai kiri akan menyimpan alamat dari anak di seblah kiri

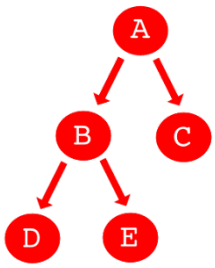
2. Kanan

   Nilai kanan akan menyimpan alamat dari anak di sebelah kanan

3. Value

   Nilai *value* akan menyimpan nilai yang dimiliki oleh simpul

B. Array



Gambar 2.6. Binary Search Tree

Setiap simpul pada BST akan direpresentasikan menggunakan tipe dasar *array*. Setiap elemen dari *array* hanya menyimpan nilai dari sebuah simpul. Untuk menyatakan anak kiri dan kanan dilakukan dengan operasi matematika. Array dengan indeks ke-i memiliki anak kiri pada indeks ke-2*i dan memiliki anak kanan pada indeks ke-2*i+1.

Gambar 2.7. Indeks Anak Kanan dan Kiri

Pohon 2.6 Apabila direpresentasikan dalam bentuk array menjadi seperti gambar 2.7 Simpul 1, A memiliki anak kiri pada indeks 2*1 = 2 dan memiliki anak kanan pada indeks 2*1+1 = 3.

2.3.4 Kelebihan

Binary Search Tree banyak digunakan dalam mengimplementasikan suatu struktur data pada pemrograman. Berikut berapa keuntungan memiliki struktur data *Binary Search Tree* :

A. BST menampilkan data secara struktur
B. BST memberikan proses *Insertion*, *Deleting*, dan *Searching* secara optimal
C. Pohon yang dibentuk bersifat fleksibel

2.3.5 Kompleksitas

Kompleksitas terbaik dari pencarian binary search tree adalah o(1). Kompleksitas ini diperoleh ketika akar dari *binary search tree* merupakan elemen yang dicari.

Kompleksitas terburuk dari pencarian binary search tree adalah ketika seluruh simpul berada di kiri atau kanan. Kompleksitas pada kasus ini adalah pencarian diharuskan menelusuri semua simpul sehingga kompleksitas berjumlah o(n).

Kompleksitas rata – rata adalah ketika simpul di sisi kiri dan kanan seimbang. Kompleksitas rata – rata memiliki waktu sebesar o(log n).

2.4 Priority Queue

2.4.1 Definisi

Priority Queue adalah tipe data seperti antrian pada umumnya. Setiap penambahan elemen pada *Priority Queue*, elemen akan ditambahkan sesuai dengan urutan prioritas. Elemen dengan nilai yang lebih besar akan menempati urutan di depan elemen yang memiliki nilai yang lebih kecil. Dalam beberapa implementasi, elemen yang memiliki prioritas yang sama dilayani sesuai dengan urutan elemen masuk. Sementara dalam implementasi lainnya, elemen dengan prioritas yang sama tidak terdefinisi dalam antrian.

2.4.2 Operasi

Priority Queue memiliki 3 operasi yaitu :

A. CekKosong
CekKosong digunakan untuk memeriksa kekosongan elemen *priority queue*
B. Push
Push digunakan untuk menambahkan elemen sesuai dengan prioritas elemen.
C. Pop
Pop digunakan untuk mengembalikan nilai dari elemen tertinggi lalu menghapus elemen tersebut dari *priority queue*.

2.4.3 Implementasi

Priority Queue dapat diimplementasikan melalui :

A. Implementasi Naif
Ini merupakan implementasi dengan cara yang paling simpel. Implementasi naif memiliki kompleksitas yang tidak efisien dalam melakukan operasi pop elemen. Setiap elemen yang baru dirtambahkan disimpan dalam array yang tidak terurut. Setiap perintah pop dipanggil, proses yang dilakukan adalah mencari nilai terbesar dalam *array* tersebut lalu mengembalikan nilai tersebut. Kompleksitas dalam melakukan pemasukan elemen dan mengembalikan nilai pop adalah o(n) karena harus dilakukan iterasi terhadap keseluruhan elemen.

B. Implementasi *Binary Search Tree*
Implementasi yang lebih dilakukan dengan menggunakan binary serach tree. Implementasi yang dilakukan adalah :

1. *Push*
Elemen yang akan disisipkan akan ditaruh pada simpul yang tepat menggunakan penyisipan pada *Binary Search Tree*
2. *Pop*
Pengambilan elemen dilakukan dengan cara mengambil elemen paling kanan menggunakan modifikasi fungsi pencarian pada *Binary Search Tree*.

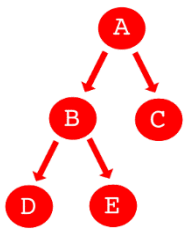
III. ISI DAN PEMBAHASAN

3.1 Deklarasi Tipe Bentukan

```
//Deklarasi tipe data buatan simpul pada BST
struct simpul {
    int nilai;
    struct simpul *kiri, *kanan;
};
```

Gambar 3.1. Deklarasi Tipe Bentukan

Struktur data yang digunakan untuk merepresentasikan *Priority Queue* adalah Struktur Data *Binary Search Tree*. Elemen yang disisipkan pada Priority Queue akan diatur sedemikian rupa sehingga memenuhi aturan pada Binary Search Tree.



Gambar 3.2 Binary Search Tree

Representasi *Binary Search Tree* dilakukan dengan representasi *pointer*. Sebuah simpul pada BST akan memiliki 3 variabel, yaitu :

A. Nilai
Nilai bertipe integer untuk menyimpan nilai yang dimiliki oleh simpul
B. Kiri
Kiri bertipe pointer akan menyimpan address anak kiri dari simpul. Pada gambar 3.2 Nilai kiri pada simpul A menyimpan alamat dari simpul B.
C. Kanan
Kanan bertipe *pointer* akan menyimpan nilai anak kanan dari simpul. Pada gambar 3.2 Nilai kanan pada

simpul A akan menyimpan alamat dari simpul C.

3.1 Implementasi Fungsi

3.1.1 Fungsi CekKosong

```
bool cekKosong(simpul *akar){
    /* Kembalikan nilai true jika simpul kosong */
    return (akar == NULL);
}
```

Gambar 3.3 Fungsi CekKosong

akar yang memiliki tipe data simpul dan memiliki keluaran bernilai *boolean*.

Definisi *priority queue* bernilai kosong apabila tidak ada nilai pada sebuah simpul akar. Fungsi CekKosong mengembalikan nilai *true* jika struktur data kosong dan *false* jika struktur data memiliki isi.

3.1.2 Fungsi Push

```
/* Fungsi untuk menyisipkan sebuah nilai pada BST */
struct simpul* push(struct simpul* simpul, int item)
{
    /* Jika pohon kosong, kembalikan pohon dengan nilai item*/
    if (cekKosong(simpul)) return newsimpul(item);

    /* Selain itu telusuri kembali pohon sesuai
       dengan hasil perbandingan item */
    if (item < simpul->nilai)
        simpul->kiri  = push(simpul->kiri, item);
    else
        simpul->kanan = push(simpul->kanan, item);

    return simpul;
}
```

Gambar 3.4 Fungsi Push

Fungsi *Push* memiliki sebuah variable masukan/keluaran, sebuah variable masukan. Variabel masukan/keluaran berupa variable simpul yang memiliki tipe data simpul. Variabel masukan memiliki nama item dengan tipe data *integer*. Fungsi *push* akan mengeluarakan sebuah keluaran bertipe data simpul.

Proses yang terjadi dalam fungsi *push* adalah :

1. Jika simpul kosong mengembalikan sebuah keluaran berupa simpul baru dengan nilai item
2. Jika nilai item lebih kecil dari nilai simpul, maka fungsi akan melanjutkan proses push ke anak kiri dari simpul. Sebaliknya, fungsi akan melanjutkan proses push ke anak kanan dari simpul. Fungsi akan mengeluarkan keluaran berupa simpul lama yang telah diperbarui anak kiri dan kanan.

3.1.3 Fungsi Pop

```
/* Fungsi untuk melakukan pop pada BST, fungsi ini akan
    memberikan keluaran berupa simpul terbaru dan merubah
    nilai variabel keluaran top dengan nilai top pada
    priority queue */
struct simpul* pop(struct simpul* akar, int* top)
{
    // kasus basis
    if (cekKosong(akar)) return akar;

    if(akar->kanan != NULL)
        /* Selama belum ketemu, telusuri anak kanan dari akar */
        akar->kanan = pop(akar->kanan,top);
    else{
        struct simpul* temp = akar->kiri;
        *top = akar->nilai;
        free(akar);
        return temp;
    }
    return akar;
}
```

Gambar 3.5 Fungsi Pop

Fungsi *Pop* memiliki sebuah variable masukan/keluaran, dan sebuah variabel keluaran. Variabel masukan/keluaran berupa variabel akar yang memiliki tipe data simpul. Variabel keluaran memiliki nama *top* dengan tipe data *integer*. Fungsi *pop* akan mengeluarkan sebuah keluaran bertipe data simpul.

Proses yang terjadi dalam fungsi pop adalah :

1. Jika simpul kosong maka fungsi akan mengembalikan nilai dari akar
2. Simpul yang akan dikeluarkan berada pada simpul yang paling kanan. Fungsi akan menelusuri simpul anak kanan hingga menemukan simpul yang tepat.
3. Jika sudah menemukan simpul yang tepat, fungsi akan memperbarui nilai top dengan nilai fungsi serta menghapus simpul tersebut dari pohon.
4. Fungsi akan mengembalikan nilai akar sebagai nilai akar yang baru.

3.1 Driver Fungsi

```
int main()
{
    int x,top;
    struct simpul *akar = NULL;
    akar = push(akar, 50);
    printf("Nilai 50 disisipkan pada priority queue\n");
    akar = push(akar, 30);
    printf("Nilai 30 disisipkan pada priority queue\n");
    akar = pop(akar,&top);
    printf("Pop nilai paling atas, Nilai bernilai %d\n",top);
    akar = push(akar, 20);
    printf("Nilai 20 disisipkan pada priority queue\n");
    akar = push(akar, 40);
    printf("Nilai 40 disisipkan pada priority queue\n");
    akar = pop(akar,&top);
    printf("Pop nilai paling atas, Nilai bernilai %d\n",top);
    akar = pop(akar,&top);
    printf("Pop nilai paling atas, Nilai bernilai %d\n",top);
    akar = pop(akar,&top);
    printf("Pop nilai paling atas, Nilai bernilai %d\n",top);
    return 0;
}
```

Gambar 3.6 Driver Fungsi Priority Queue

```
Nilai 50 disisipkan pada priority queue
Nilai 30 disisipkan pada priority queue
Pop nilai paling atas, Nilai bernilai 50
Nilai 20 disisipkan pada priority queue
Nilai 40 disisipkan pada priority queue
Pop nilai paling atas, Nilai bernilai 40
Pop nilai paling atas, Nilai bernilai 30
Pop nilai paling atas, Nilai bernilai 20
```

Gambar 3.7 Hasil Compile dari Driver Fungsi

Untuk melakukan percobaan pada fungsi yang telah dibuat, penulis membuat driver yang berisi deretan perintah yang manfaatkan fungsi pada struktur data priority queue.

1. Struct simpul *akar = NULL

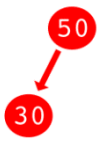
Pada perintah pertama, diinisialisasi variable yang menggunakan struktur data priority queue bernama pq. Pada awalnya pq belum memiliki nilai karena simpul akar pada pohon BST pq memiliki nilai NULL.

2. Pq = push(pq,50)

Gambar 3.8 Hasil dari Langkah 2

Perintah kedua, driver akan menyisipkan nilai 50 pada pq. Pemanggilan fungsi ini akan membuat pohon pq memiliki sebuah simpul dengan nilai 50. Seperti gambar 3.8 , simpul ini masih berupa simpul daun karena tidak memiliki anak.

3. Pq = push(akar,30)



Gambar 3.9 Hasil dari Langkah 3

Perintah ketiga, driver akan menyisipkan nilai 30 pada pq. Nilai ini akan dibandingkan dengan simpul pertama, 50. Ternyata, nilai yang dimasukkan lebih kecil dari 50 sehingga akan dilakukan penyisipan nilai 30 dengan menjadi anak kiri dari simpul bernilai 50.

4. Pq = pop(pq, &top)

Perintah keempat, driver meminta nilai dari simpul yang memiliki prioritas tertinggi pada priority queue pq.

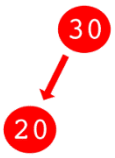
Gambar 3.10 Nilai Top pada Langkah ke-4

Seperti pada gambar 3.10 , pencarian simpul dilakukan dari akar pohon pq. Karena simpul 50 tidak memiliki anak kanan, maka simpul 50 menjadi simpul yang akan diambil nilainya. Nilai 50 akan disimpan pada variable *top*.

Gambar 3.11 Penghapusan Simpul Top Pada Pohon PQ

Setelah nilai disimpan pada variable top, simpul 50 dihapus dari pohon, sehingga pohon kini menjadi seperti pada gambar 3.11

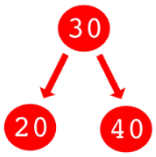
5. Pq = push(akar,20)



Gambar 3.12 Hasil dari Langkah 5

Perintah kelima, driver akan menyisipkan nilai 20 pada pohon pq. Nilai ini akan dibandingkan dengan simpul pertama, 30. Ternyata, nilai yang dimasukkan lebih kecil dari 30 sehingga akan dilakukan penyisipan nilai 20 dengan menjadi anak kiri dari simpul bernilai 30.

6. Pq = push(akar,40)



Gambar 3.13 Hasil dari Langkah 6

Perintah keenam, driver akan menyisipkan nilai 40 pada pq. Nilai ini akan dibandingkan dengan simpul pertama, 40. Ternyata, nilai yang dimasukkan lebih besar dari 40, sehingga akan dilakukan penyisipan simpul bernilai 40 dengan menjadi anak kanan dari simpul bernilai 30.

7. Pq = pop(pq,&top)

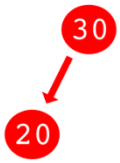
Perintah ketujuh, driver meminta nilai dari simpul yang memiliki prioritas tertinggi pada priority queue pq.

Gambar 3.14 Nilai Top pada Langkah ke-7

Pencarian simpul dengan nilai terbesar pada pohon pq dimulai dari simpul akar. Dilakukan pengecekan terhadap

simpul tersebut, apakah memiliki anak kanan atau tidak. Ternyata simpul memiliki anak kanan, fungsi akan memanggil diri sendiri dengan mengganti nilai pq dengan nilai anak kanan dari pq.

Seperti pada gambar 3.14 , simpul baru tiidak memiliki anak kanan, sehingga simpul tersebut adalah simpul yang memiliki nilai terbesar. Nilai pada simpul disimpan pada variable keluaran top.



Gambar 3.15 Penghapusan Simpul Top Pada Pohon PQ

Setelah nilai terbesar disimpan, simpul dengan nilai terbesar akan dihapus dari pohon pq. Kini, simpul menjadi seperti pada gambar 3.15.

8. Pq = pop(pq,&top)

Perintah kedelapan, driver meminta nilai dari simpul yang memiliki prioritas tertinggi pada priority queue pq.

Gambar 3.16 Nilai Top pada Langkah ke-8

Pencarian simpul dengan nilai terbesar pada pohon pq dimulai dari simpul akar. Dilakukan pengecekan terhadap simpul tersebut, apakah memiliki anak kanan atau tidak. Ternyata simpul tidak memiliki anak kanan, sehingga simpul dengan nilai 30 merupakan simpul yang memiliki nilai terbesar.

Nilai pada simpul dimasukkan pada variable keluaran top lalu simpul akan dihapus dari pohon pq. Karena simpul yang akan dihapus merupakan simpul akar dari pohon pq, maka nilai akar pq akan diperbarui dengan nilai simpul anak kiri dari pq seperti terlihat pada gambar 3.16.

9. Pq = pop(pq,&top)

Perintah kedelapan, driver meminta nilai dari simpul yang memiliki prioritas tertinggi pada priority queue pq.

Gambar 3.17 Nilai Top pada Langkah ke-9

Kini, pohon pq hanya memiliki satu simpul sehingga nilai simpul akar dari pq dimasukkan ke dalam variable keluaran top. Setelah penyimpanan, simpul akar akan dihapus dari pohon pq sehingga kini pohon pq tidak memiliki simpul

V. SIMPULAN

Pemanfaatan dari struktur data *binary search tree* yang terdapat dalam ruang lingkup matematika diskrit dpat diaplikasikan sebagai representasi dari struktur data *priority queue*.

Struktur data *priority queue* yang memanfaatkan *binary search tree* sebagai representasi memiliki bentuk pohon yang terstruktur sehingga operasi dapat dilakukan lebih cepat.

V. UCAPAN TERIMAKASIH

Pertama, penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa karena berkat rahmat-Nya penulis dapat menyelesaikan makalah dengan judul “Implementasi Binary Search Tree pada Struktur Data Priority Queue dalam Bahasa C” ini.

Tak lupa, Penulis juga ingin berterima kasih kepada semua pihak yang telah mendukung penulisan makalah ini, baik secara langsung maupun secara tidak langsung. Penulis berterimakasih kepada Pak Rinaldi Munir dan Bu Harlili sebagai pengampu mata kuliah matematika diskrit. Secara khusus, Penulis berterimakasih kepada Pak Judhi Santoso sebagai dosen pengampu mata kuliah matematika diskrit kelas K-3.

DAFTAR PUSTAKA

[1] R. Munir, “Pohon”, dalam Matematika Diskrit, ed. 3. Bandung:Informatika, 2010, hlm. 443-475

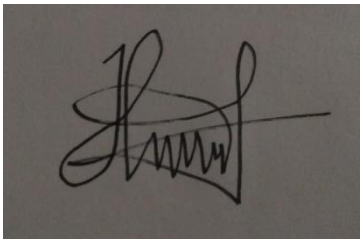
[2] K. Manoj, Binary Search Tree | Set 2 (Delete), https://www.geeksforgeeks.org/binary-search-tree-set-2-delete/ [diakses pada 9 Desember 2018, pukul 08.00 WIB]

[3] S. Robert, 2.4 Priority Queues, https://algs4.cs.princeton.edu/24pq/, [diakses pada 9 Desember 2018, pukul 10.00 WIB]

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, 9 Desember 2018



Muhammad Hanif Adzkiya
13517120