Penerapan struktur data *Heap Priority Queue* pada algoritma Djikstra untuk mendapatkan kompleksitas O((n + m)log n)

Aditya Nurcholis H[1], Simon Batara N[2], Mohamad Octamanullah[3]

Laboratorium Ilmu dan Rekayasa Komputasi
Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if13107@students.if.itb.ac.id[2],
if13109@students.if.itb.ac.id[3], if13119@students.if.itb.ac.id[4]

Abstrak

Metode untuk menemukan lintasan terpendek untuk menghubungkan dua titik dari suatu graf telah menjadi bahan pemikiran banyak orang, dikarenakan metoda ini akan banyak membantu persoalan-persoalan yang ada di masyarakat. Seperti menemukan lintasan dengan biaya termurah pada suatu rute penerbangan, atau menemukan lintasan yang harus ditempuh suatu data dalam mencapai komputer tujuan pada suatu jaringan.
Telah banyak ilmuwan yang meneliti tentang metode ini, salah satunya adalah Djikstra yang mempublikasikan kode programnya pada tahun 1959 untuk menemukan lintasan terpendek, yang sampai sekarang lebih dikenal dengan algoritma Djikstra. Ternyata algoritma Djikstra ini masih bisa disempurnakan dengan sebuah immplementasi struktur data Priority Queue. Dalam kesempatan kali ini, kami akan mencoba untuk menyempurnakan algoritma Djikstra yang ada dengan menggunakan implementasi priority queue.

***Kata kunci**: Djikstra, Priority Queue, Heap, lintasan terpendek.*

1. Pendahuluan

Adapun bahan-bahan yang kami pakai adalah algoritma Djikstra dan struktur data Priority Queue.

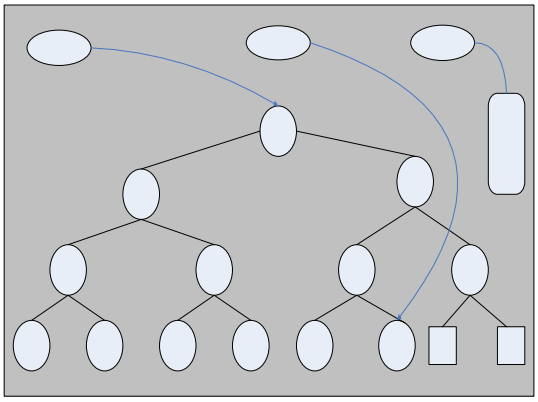
1.1 Algoritma Djikstra

Algoritma Djikstra merupakan algoritma greedy untuk mendapatkan lintasan terpendek. Algoritma ini menggunakan strategi greedy sebagai berikut:
"Pada setiap langkah, ambil sisi yang berbobot minimum yang menghubungkan sebuah simpul yang sudah terpilih dengan sebuah simpul lain yang belum terpilih. Lintasan dari simpul asal ke simpul yang baru haruslah merupakan lintasan yang terpendek diantara semua lintasannya ke simpul – simpul yang belum terpilih."
Kompelsitas waktu dari algoritma Djikstra adalah $O(n^2)$.

1.2. Priority Queue

Priority Queue adalah suatu bentuk struktur data yang berbasiskan struktur Queue pada umumnya. Pada priority Queue, terdapat salah satu bentuk implementasi yaitu implementasi menggunakan struktur data heaps. Dalam implementasi ini ada 3 pokok atribut yaitu heaps, comparator, dan last. Dengan menggunakan fasilitas heap ini kita bisa melakukan insert atau remove elemen dalam kompleksitas waktu logaritmik. Pada priority Queue ini, secara keseluruhan akan dibentuk sebuah pohon biner yang seimbang. Dalam pembuatan pohon biner ini diusahakan ketinggian yang terbentuk adalah minimum. Keminimuman tinggi pohon inilah yang menyebabkan kompleksitas waktu dalam memanipulasi daun menjadi fungsi logaritmik.

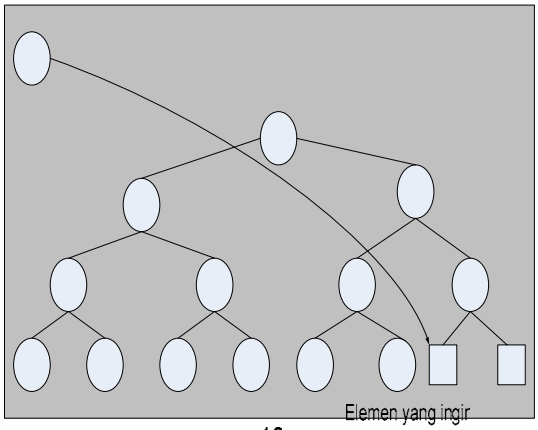
Struktur Priority Queue :



Heap menunjuk pada akar, last menunjuk pada elemen akhir, dan comparator menunjukan pembanding untuk menipulasi posisi.

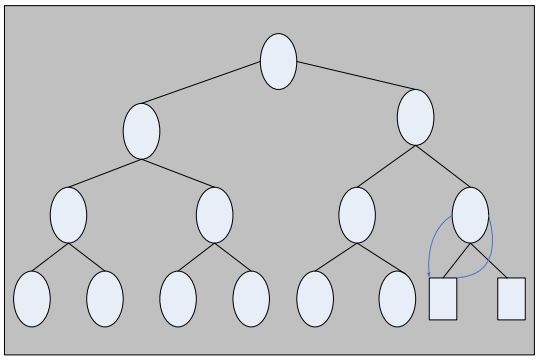
Metode Insert :

awal :



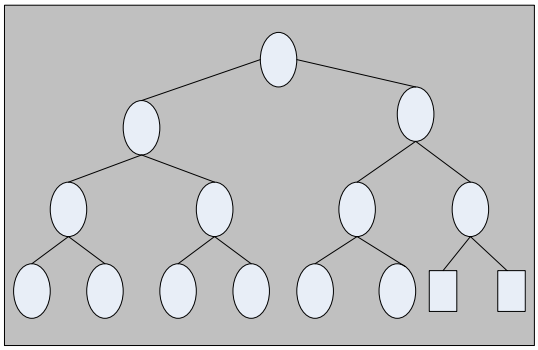
10

selanjutnya :



15

Posisi Akhir :

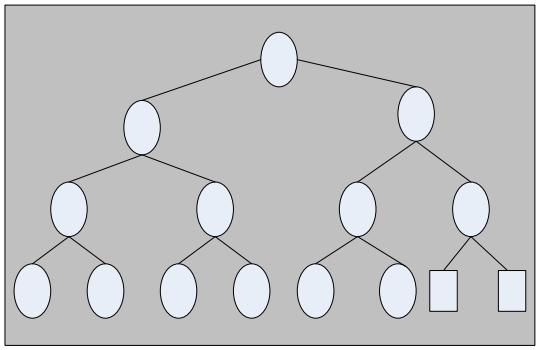


14

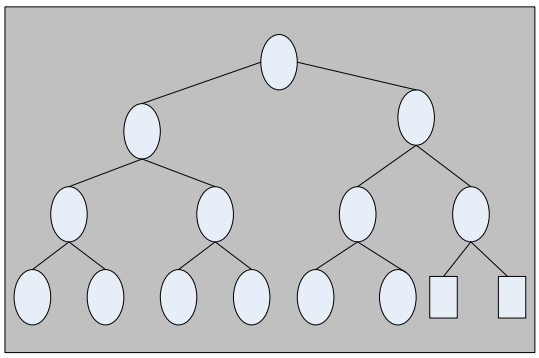
Jadi pohon biner tetap terurut dari akar sampai ke daun selalu menurun.

Metode Remove_Min :

Posisi Awal :



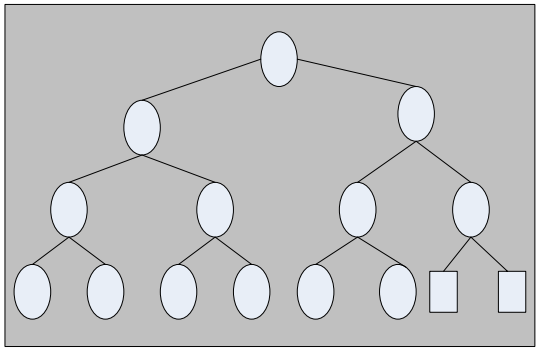
Posisi I :



Elemen 4 diambil dan elemen last yaitu 20 dijadikan heap.

9 7 20

Posisi II :



Elemen 5 dan 20 ditukar. 5 sebagai heap.

4

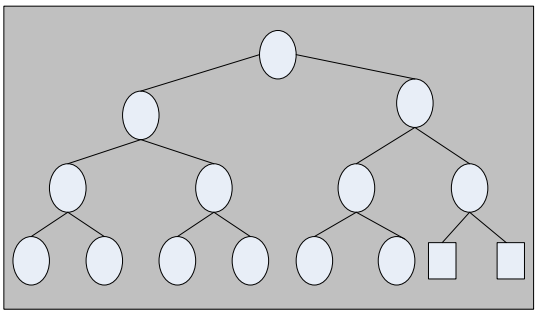
6

5

Terjad
penukarar
elemen

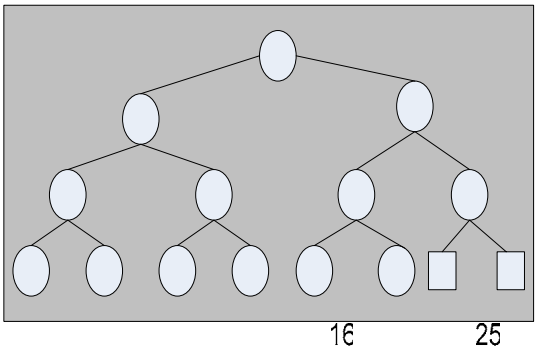
15 9 7 20

Posisi III :



20 ditukar dengan 15.

Posisi Akhir :



16 ditukar dengan 20. terbentuk pohon biner yang valid karena selalu terdapat lintasan yang menurun dari akar ke setiap daunnya.

2. Penggunaan Priority Queue dalam Algoritma Djikstra

Procedure Djikstra(input M : matriks, a : simpul_awal) → tabel
{mencari lintasan terpendek adri simpul awal ke semua simpul lainnya}

```
Deklarasi  :
  D, S : tabel;
  i       : integer;

Algoritma :
{langkah 0 inisialisasi}
For i ← 1 to n do
     S[i] ← 0
     D[i] ← m[a,i]
Endfor

{langkah 1}
S[a] ← 1
D[a] ← ∞

{langkah selanjutnya..}
For i ← 2 to n-1 do
    Cari j sedemikian sehingga S[j] = 0
dan D[j] =     minimum
Hitung D[i] yang baru dari a ke simpul
i bukan elemen S :
    D[i]    ← Minimum (D[i], D[j] +
M[j,i])
Endfor
```

Dalam algoritma djikstra ini terdapat pencarian jarak minimum untuk simpul yang belum terpilih. Dalam hal ini kita bisa menggunakan *priority Queue* untuk mendapatkan elemen terkecil dan menggunakannya sebagai pembanding untuk langkah update jarak.
Jadi seharusnya kompleksitas yang dibutuhkan dalam langkah ini adalah O(n) tetapi dengan menggunakan priority Queue, kita bisa mereduksi kompleksitas menjadi O(log n).

Setelah menggabungkan metode dalam priority Queue dengan algoritma djikstra, maka akan didapatkan algoritma yang berisi :

```
//Tahap inisiasi dilakukan seperti pada
//algoritma djikstra biasa
//Lalu pada tahap langkah selanjutnya :
//D[u] adalah jarak dari simpul awal ke
//simpul u

while (Q is not empty) do
       u = Q.removeMin() // menggunakan
             metode pada priority Queue
      for    (setiap    simpul    yang
bertetangga dan
ada di Q)
                 {//z adalah simpul yang
                       belum terpilih
      if (D[u] + w ((u,z)) < D[z] then
                 D[z] = D[u] + w((u,z))
            }
return D;
```

Jadi dengan implementasi ini kita bisa mereduksi kompleksitas pencarian jarak minimum O(n) dengan menggunakan removeMin() pada priority Queue yang memiliki kompleksitas O(log n).

Kompleksitas algoritma djikstra biasa :
$T(n) = O(n) + (n-2)[O(n) + O(n)]$
$T(n) = O(n^2)$

Kompleksitas algoritma dengan implementasi priority Queue:
Inisiasi membentuk pohon bisa didapatkan dengan mengalikan jumlah suku yang ditambahkan dikalikan log n : n (log n).
Untuk melakukan removeMin : O(log n)
Untuk melakuakn update jarak : O(degree(v) log n).
Jadi untuk keseluruhan kompleksitasnya adalah :
T(n) = O(n (log n)) + (O(degree(v) log n) + O(log n))
T(n) = O(n (log n)) + (O(1 + degree(v)))log n)
T(n) = O ((n+m) log n)

4. Kesimpulan

Kompleksitas waktu Algoritma Djikstra dapat direduksi dari $O(n^2)$ menjadi $O((n+m) \log n)$ dengan menggunakan struktur data *priority queue* sehingga waktu pencarian jarak minimum yang seharusnya $O(n)$ menjadi $O(\log n)$.

Daftar Pustaka

1. Goodrich, Tamassia, Mount. *Data Structure and Algorithms in C++*, Wiley, 2004.
2. Munir, Rinaldi. *Diktat Kuliah IF2251 Strategi Algoritmik*, 2005.
3. Schildt, Herbert, The Complete Reference Java, Mc.Graw Hill, 2005.
4. Sedgewick, Robert. *Algorithms in C*, 1990.