Pengaplikasian Algoritma Dijkstra Dalam Pembuatan Agenda Penerbangan

Muhammad Iqbal / 13510064
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
mhmmd.iqbal@students.itb.ac.id

***Abstrak*— Agenda penerbangan memiliki beberapa komponen seperti nomor penerbangan, tujuan dan asal keberangkatan, waktu keberangkatan dan waktu kedatangan, dll. Makalah ini berisi tentang pengimplementasian dan pengaplikasian sebuah Algoritma untuk membuat agenda penerbangan. Masalah pembuatan agenda ini dapat dimodelkan dengan graf, dan dapat diselesaikan dengan algoritma. Tentu saja kita ketahui bahwa perlu dibuat suatu algoritma yang mengatur tentang penjadwalan dalam penerbangan agar tidak kacau. Salah satu Algoritma yang mengatur hal ini adalah algoritma Dijkstra . Agenda ini juga dbuat dengan bantuan priority queue untuk mengendalikan input dan output dari jadwal penerbangan.**

***Kata Kunci*—Algoritma Dijkstra,*Priority Queue*, *prority*, Jadwal Penerbangan.**

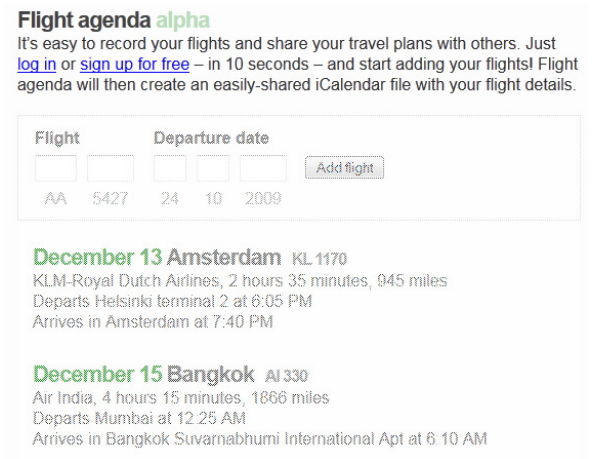
I. PENDAHULUAN

Pada zaman modern ini, tentu saja kita dapat melihat jadwal penerbangan pada setiap airport. Ya, jadwal penerbangan pada airport adalah suatu jadwal yang menampilkan beberapa informasi, yang ditujukan pada para pengunjung dan penumpang. Mungkin secara sekilas kita semua kurang mempedulikan bagaimana jadwal tersebut bisa tersusun secara sistematis dan tidak kacau. Biasanya jadwal penerbangan ini ditampilkan oleh suatu sistem seperti FIDS (Flight Information Display System) yang berfungsi untuk memberitahu para pengunjung informasi tentang jadwal penerbangan baik dari segi *departure* maupun *arrival.* Media yang digunakan pada umumnya LED atau TV plasma.

GA 203	JAKARTA	08:00	04	LAST CALL
JT 565	JAKARTA	08:00	01	SECOND CALL
GA 250	DENPASAR	08:00	03	BOARDING
Y6 567	BATAM	08:05	00	CHECK IN OPEN
AK 595	KUALA LUMPUR	08:55	01	TO WAITING ROOM
Y6 232	PONTIANAK	09:05	00	CHECK IN OPEN
JT 553	JAKARTA	09:25	00	CHECK IN OPEN

Gambar 1. Contoh tampilan dari jadwal penerbangan

Terlepas dari pengoperasian dari jadwal penerbangan ini yang cukup sulit dan rumit, saat ini sudah disediakan berbagai fasilitas untuk memudahkan pembuatan jadwal penerbangan, bahkan jadwal penerbangan pribadi, yaitu jadwal penerbangan yang akan kita lakukan kedepannya. Dengan adanya jadwal penerbangan pribadi ini, kita dapat mempermudah manajemen diri kita. Selain itu, agenda penerbangan ini juga biasanya digunakan oleh agen penerbangan bersangkutan yang kita gunakan, misalnya saja untuk menentukan waktu kedatangan pertama dari list agenda penerbangan para user yang menggunakan agen penerbangan ini. Gambar *screenshot* dibawah ini merupakan salah satu situs pembuat *flight agenda*.



Gambar 2. Situs pembuat *flight agenda*

Dapat dilihat bahwa situs tersebut menawarkan

kemudahan dalam pembuatan agenda penerbanga Sebenarnya apa yang bisa membuat situs terseb membuatnya dengan mudah? Sedangkan kita tahu bahv membuat jadwal penerbangan dengan berbagai paramet yang berbeda adalah cukup sulit dan rumit.

Untuk mengatur dan membuat jadwal penerbangan i kita menggunakan sebuah priority queue yang berfung untuk me-*maintain* sebuah elemen yang dikumpulk berdasarkan suatu prioritas. Dari sebuah priority queue ini, akan digunakan sebuah algoritma untuk memudahkan pembuatan jadwal penerbangan, yaitu Algoritma Dijkstra.

Dengan mengkombinasikan priority queue dengan pengimplementasian dalam bentuk algoritma Dijkstra, kita dapat membuat sebuah jadwal penerbangan dengan mudah dan sistematis.

Gambar 3. Visualisasi sebuah priority queue

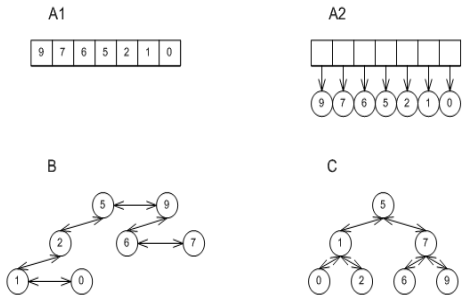
II. DASAR TEORI

A. *Priority Queue*

Priority Queue adalah suatu *Abstract Data Type* biasa seperti halnya array, tabel, queue, stack, dan lain-lain. *Priority queue* ini bisa dikatakan merupakan sebuah modifikasi dari sebuah queue biasa. Perbedaannya terletak pada urutan dari antrian. Ketika kita berlanjut dari suatu elemen, maka elemen selanjutnya yang diakses adalah elemen dengan tingkat keprioritasan tertinggi. Tak jarang *priority queue* ini juga sering direpresentasikan dengan menggunakan pemodelan stack.

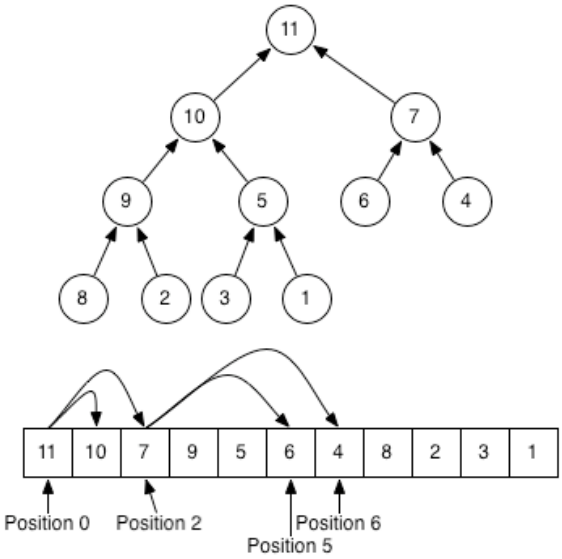
Pada penggunaannya, *priority stack* ini digunakan untuk menjaga sebuah sekumpulan objek agar tetap terurut.

Banyak macam-macam pengimplementasian dari *priority queue* ini. Yang pertama dalah implementasi yang "naif", yaitu pengimplementasian dengan cara tetap membiarkan queue dalam keadaan tidak terurut (*unsorted*). Kapanpun sebuah elemen dengan prioritas tertinggi diminta, maka akan dilakukan pencarian terhadap seluruh elemen dan mengembalikan nilainya. Implementasi ini adalah implementasi yang tidak efisien.



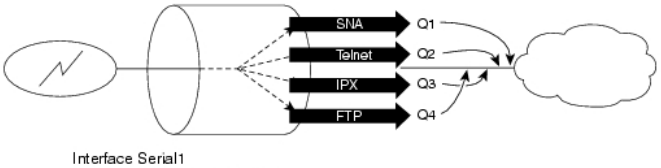
Gambar 4. Berbagai macam pengimplementasian

Implementasi selanjutnya yang lebih umum dan sering dipakai adalah dengan menggunakan *heap.*Meskipun membutuhkan waktu yang lebih lama, namun dengan metode ini, dimungkinkan untuk melihat elemen dengan prioritas tertinggi tanpa harus membuangnya pada saat itu.



Gambar 5. Gambar *Heap*

Pengaplikasian *priority queue* ini sendiri cukup banyak, salah satunya adalah dalam manajemen *bandwidth* . Priority queue dapat me-*manage* suatu bandwidth dalam suatu saluran transmisi dari router jaringan. Jika *traffic* dari sebuah jaringan sudah tidak memnuhi bandwidth, maka secara otomatis antrian lain dapat dihentikan, hal ini dilakukan untuk mengirimkan traffic pada antrian lain dengan prioritas tertinggi. Hal ini untuk memastikan bahwa traffic dengan prioritas tertinggi akan diteruskan dengan keterlambatan yang minimum. Sedangkan traffic lain dapat dibereskan saat antrian prioritas tertinggi kosong.




```
Interface Serial1
ip address 20.0.0.1 255.0.0.0
  priority-group 1
!
priority-list 1 protocol ip high tcp 2065
priority-list 1 protocol ip medium tcp 23
priority-list 1 protocol ipx normal
priority-list 1 protocol ip low tcp 21
```

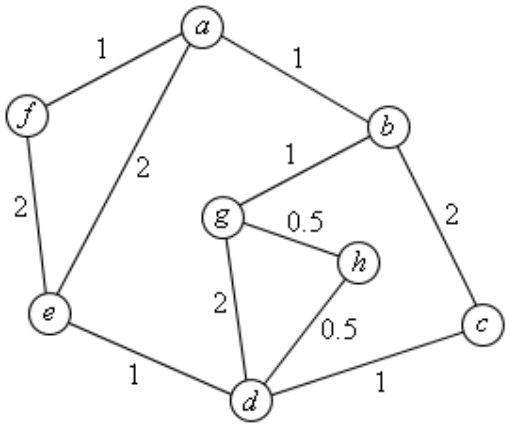
Gambar 6. Bandwidth Management dengan menggunakan priority queue

Selain itu, priority queue juga diimplementasikan pada Algoritma Dijkstra. Ketika sebuah bentuk graf diimplementasikan dalam sebuah *priority queue* maka hal ini bisa digunakan untuk mengekstrak efisiensi minimum ketika kita mengimplementasikannya pada Algoritma Dijkstra.

B. Algoritma Dijkstra

Algoitma Dijkstra, ditemukan oleh Edsger Dijkstra pada tahun 1956 dan dipublikasikan pada tahun 1959, adalah sebuah algoritma pencarian dalam bentuk graf yang memecahkan sebuah masalah *shortest path problem*. Algoritma ini sering digunakan pada *routing*.

Untuk sebuah simpul tertentu, algoritma ini akan menemukan lintasan terpendek dari jarak antara simpul dengan simpul lainnya. Bisa juga untuk menemukan nilai dari jarak terpendek dari sebuah simpul tunggal ke tujuan tunggal dengan menghentikan algoritma ketika jarak terpendek ke simpul tujuan telah ditentukan.



Gambar 7. Dijkstra's Algorithm

Algoritmanya adalah sebagai berikut.

Misal simpul yang akan kita mulai adalah simpul awal. Lalu misalkan Jarak Y adalah jarak dari simpul awal ke Y. Algoritma Dijkstra ini akan mencoba untuk mencari nilai dari Jarak Y melalu tahap-tahap berikut.

1. Tetapkan nilai dari jarak tentatif untuk setiap simpul. Nol untuk simpul awal dan tak hingga untuk simpul lain.

2. Tandai semua simpul yang belum dikunjungi. Tandai simpul awal sebagai simpul yang sedang dikunjungi. Buatlah satu set simpul yang terdiri dari simpul yang belum dikunjungi kecuali simpul awal.

3. Untuk simpul yang sedang dikunjungi, perhitungkan semua simpul tetangga yang belum dikunjungi dan hitung jarak tentatif mereka.

4. Ketika sudah memperhitungkan semuanya, tandai simpul tadi sebagai simpul yang sudah dikunjungi. Dan hapus dari set simpul yang belum dikunjungi. Simpul yang sudah dikunjungi tidak akan diperiksa lagi.

5. Ketika simpul selanjutnya sudah ditandai atau jarak tentatif terkecil antar simpul yang belum dikunjungi adalah tak hingga, maka berhenti. Algoritma telah selesai.

6. Tandai simpul yang belum dikunjungi dengan jarak tentatif terkecil sebagai simpul yang sedang dikunjungi dan kembali ke langkah 3.

Dalam pseudocode, algoritmanya adalah sebagai berikut :

```
1  function Dijkstra(Graph, source):
 2      for each vertex v in Graph:
// Initializations
 3          dist[v] := infinity ;
// Unknown distance function from
source to v
 4          previous[v] := undefined ;
// Previous node in optimal path from
source
 5      end for ;
 6      dist[source] := 0 ;
// Distance from source to source
 7      Q := the set of all nodes in
Graph ;   // All nodes in the graph
are unoptimized - thus are in Q
 8      while Q is not empty:
// The main loop
 9          u := vertex in Q with
smallest distance in dist[] ;
10          if dist[u] = infinity:
11              break ;
// all remaining vertices are
inaccessible from source
12          end if ;
13          remove u from Q ;
14          for each neighbor v of u:
// where v has not yet been removed
from Q.
15              alt := dist[u] +
dist_between(u, v) ;
16              if alt < dist[v]:
// Relax (u,v,a)
17                  dist[v] := alt ;
18                  previous[v] := u ;
```

```
19                  decrease-key v in
Q;       // Reorder v in the Queue
20              end if ;
21           end for ;
22       end while ;
23       return dist[] ;
24   end Dijkstra.
```

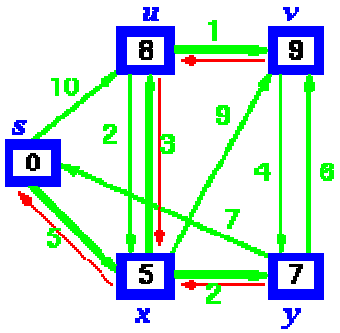
Dalam algoritma diatas, kode u := vertex in Q with smallest dist[], mencari simpul u dalam kumpulan simpul Q yang mempunyai jarak[u] paling kecil. Simpul tersebut dihapus dari kumpulan simpul Q dan dikembalikan pada user. dist_between(u,v) menghitung jarak antara dua simpul tetangga u dan v. Variabel alt pada line 15 adala jarak dari simpul root ke simpul tetangga v jika melewati u.

Jika kita hanya tertarik dalam jarak terpendek antara simpul asal dan simpul tujuan, kita bisa menterminasi lin 13. Sekarang kita bisa mendapatkan jarak terpendek dari simpul asal ke tujuan dengan cara :

```
1   S := empty sequence
2   u := target
3   while previous[u] is defined:
4       insert u at the beginning of S
5       u := previous[u]
6   end while ;
```

Sekarang S adalah yang merepresentasikan jarak terpendek dari simpul awal sampai tujuan, atau kosong apabila jarak tidak ada.

Penerapan Algoritma Dijkstra ini antara lain adalah pada prinsip dasar dari *link-state routing protocols,* OSPF, dan IS-IS.



Gambar 8. Ilustrasi algoritma Dijkstra

III. PEMBAHASAN

Pada Bab ini akan disampaikan sebuah pembahasan mengenai masalah yang ada, masalah yang dibahas disini adalah bagaimana ketika kita ingin membuat sebuah agenda penerbangan, khususnya untuk para agen. Misalkan saja ketika sebuat agen perjalanan bisa mengakses semua data agenda perjalanan, dan mereka ingin menentukan kedatangan paling pertama dengan kota asal dan tujuan telah diketahui.

Caranya adalah dengan memodifikasi algoritma Dijkstra

Misal graf adalah sebuah di-graf.

Vertices merepresentasikan bandara-bandara.

di-edges adalah sebuah penerbangan dengan dua nilai, yaitu waktu kedatangan dan waktu keberangkatan.

Fungsi dari jarak yang digunakan haruslah waktu kedatangan (T) , karena waktu kedatanganlah yang diinginkan oleh sang agen. Yaitu waktu kedatangan pesawat di suatu bandara.

Disini kita menggunakan "minimum" *priority queue* Q yang berisi daftar bandara dengan isi T .

Tapi waktu kedatangan dari simpul awal, yaitu bandara asal (s) adalah T[s] = startT. Waktu awal.
Kalang ketetanggaan juga harus diubah,
hanya bisa memilih penerbangan, f, yang masih bisa dikejar , atau f.depart > T[v].
Bagaimana kita dapat menentukan penerbangan yang baik? Buat sebuah priority queue baru P.
Berikut algoritma pembuatan agenda penerbangan:

for all vertices, *w*, adjacent to *v* **do**
make Priority Queue, *P*, of fights, *f*, with *f.departT* ≥ *T*[*v*] keyed by *f.arriveT*
Time *t* ← ∞
if *P* is not empty **then** *t* ← *P*.min().*arriveT*

if *t* < *T*[*w*] **then**
T[*w*] ← *t*
update *w* in *Q*

FlightAgenda(DiGraph *G*, Vertex *s,* Time *startT*)
// Input: *G* di-graph
// vertices adalah bandara
// di-edges penerbangan dengan dua nilai
// *departT* waktu keberangkatan dari bandara awal
// *arriveT* waktu kedatangan dari bandara tujuan
// *s* adalah bandara awal dan *startT* adalah waktu awal

// inisialisasi dari waktu kedatangan awal, *T*
T[*s*] ← *startT*
for all vertex, *v* ≠ *s* **do** *T*[*v*] ← ∞

make Priority Queue, *Q*, of vertices keyed by *T*

while Q is not empty **do**
$v \leftarrow Q$.removeMin()
for all vertices, w, adjacent to v **and** in Q **do**

// menentukan penerbangan selanjutnya
make Priority Queue, P, of fights, f, with $f.departT \geq T[v]$ keyed by $f.arriveT$
Time $t \leftarrow \infty$
if P is not empty **then** $t \leftarrow P$.min().$arriveT$

// relaxation
if $t < T[w]$ **then**
$T[w] \leftarrow t$
update w in Q
return T

Bagaimana dengan *cost* nya?

Menentukan *cost* sedikit susah karena penambahan cost dari pembuatan semua agenda penerbangan yang bertipe priority queue. Dengan berasumsi bahwa untuk kasus terburuk terdapat m buah priority queue penerbangan dibuat dengan *cost* O(m) untuk membuat setiap priority queue, maka kita dapatkan total cost $O(m^2)$ untuk semua priority queue. Tentu saja ini merupakan *cost* yang sangat besar, alasannya adalah karena tidak akan ada m buah penerbangan yang didampingi oleh m buah priority queue. Cara lain untuk mempertimbangkannya adalah dengan menghitung setiap sisi tetapi hanya terhadap satu penerbangan saja. Dengan begitu total cost untuk membuat semua priority penerbangan adalah O(m). Total cost untuk penggunaan heap untuk priority queue bandara adalah O((n+m) log n).

Dengan algoritma yang telah dibuat diatas, kita dapat membuat sebuah agenda penerbangan yang sudah cukup baik , dan salah satu kasus diatas telah dipecahkan, yaitu bagaimana cara membuat agenda penerbangan yang diurutkan sesuai dengan waktu kedatangan masing-masing.

IV. KESIMPULAN

Berdasarkan uraian dan analisis, serta perbandingan diatas, dapat diambil beberapa kesimpulan:

1. Algoritma Dijkstra dapat dimanfaatkan dalam pembuatan agenda penerbangan, serta dalam berbagai pemecahan masalah lainnya.
2. Dalam pembuatan agenda penerbangan, dibutuhkan juga sebuah priority queue yang dipadukan dengan algoritma Dijkstra.
3. Total *cost* untuk penggunaan heap sebagai priority queue adalah O((n+m) log n).

DAFTAR PUSTAKA

[1] http://en.wikipedia.org/wiki/Dijkstra's_Algorithm diakses pada tanggal: 10 Desember 2011.
[2] http://en.wikipedia.org/wiki/Priority_queue diakses pada tanggal: 11 Desember 2011.
[3] http://www.toves.org/books/data/ch08-pq/index.html diakses pada tanggal: 11 Desember 2011.
[4] http://www.csl.mtu.edu/cs2321/www/newLectures/30_More_Dijkstra.htmdiakses pada tanggal: 11 Desember 2011.
[5] Munir, Rinaldi.2008.*Diktat Kuliah Struktur Diskrit.*Bandung: Penerbit Informatika ITB.

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, 11 Desember 2011

Muhammad Iqbal / 13510064