Implementasi Algoritma *Dynamic Programming* dalam Menyelesaikan *Airport Gate Assignment Problem*

Ainun Fitryh Vianiryzki / 13517057
Informatics Engineering
School of Electrical Engineering and Informatics
Bandung Institute of Technology, Jl. Ganesha no. 10, Bandung, Indonesia
13517057@std.stei.itb.ac.id

***Abstract*— Arus *traffic* pada bandara seringkali sibuk dan memiliki mobilitas yang tinggi. Terutama, pada bandara-bandara internasional. Berangkat dari fakta ini, menempatkan pesawat terbang ke sarana yang terdapat di bandara merupakan hal yang sangat penting. Hal ini bisa disebut dengan *Airport Gate Assignent Problem (AGAP).* AGAP merupakan permasalahan sehari-hari yang sangat serius dan penting yang kita hadapi sehari-hari. Permasalahan utama dalam penentuan ini adalah menempatkan pesawat terbang pada *gate* yang tepat sehingga dapat meminimalisir *time delay* dan meminimalisir pesawat terbang yang tidak ter-*assign* kepada *gate.* Makalah ini akan mencoba menyelesaikan masalah yang telah disebutkan dengan menggunakan algoritma *dynamic programming* atau pemrograman dinamis.**

Keywords — Dynamic Programming, Bandara, Pesawat Terbang, Penjadwalan

I. Introduction

Pesawat terbang merupakan fasilitas transportasi umum yang digunakan untuk berpindah dari satu tempat ke tempat yang lain, terutama bagi mereka yang membutuhkan untuk bepergian jarak jauh namun dengan waktu yang sesingkat-singkatnya. Dengan menggunakan pesawat terbang, orang-orang dapat berpindah dari satu tempat ke tempat lain dalam waktu yang singkat

Ketika kita membahas tentang pesawat terbang dan maskapai penerbangannya, maka pembahasan kita juga tidak akan terlepas dari bandar udara (atau disingkat dengan bandara), yaitu tempat yang digunakan untuk lepas landas dan mendarat oleh pesawat terbang. Pada sebuah bandara, seminimal-minimalnya akan terdapat fasilitas-fasilitas berupa:

1. *Runway*
   Runway dalam bahasa Indonesia adalah landasan pacu, yaitu sebuah area yang dipakai untuk melakukan *landing* (mendarat) dan *take-off* (lepas landas).
2. *Taxiway*
   Taxiway adalah area untuk menghubungkan *runway* dan *apron*. *Taxiway* berguna untuk jalur pesawat berpindah dari *runway* ke *apron*, atau sebaliknya.
3. *Apron*
   Apron adalah area yang digunakan pesawat untuk parkir, mengisi bahan bakar, melakukan pengecekan keadaan pesawat, melakukan pemeliharaan pesawat, menaikkan dan menurunkan penumpang serta barang.
4. *Terminal*
   Terminal adalah tempat bagi penumpang untuk melakukan pengurusan perjalanan udara seperti pembelian tiket, pemeriksaan surat identitas, dan menunggu jadwal keberangkatan. Pada terminal, terdapat fasilitas-fasilitas pendukung seperti ruang tunggu, restoran, toilet, dan musholla.
5. *Crub*
   Crub adalah tempat penumpang untuk naik dan turun dari kendaraan menuju atau meninggalkan bendara.
6. Tempat parkir kendaraan
   Tempat parkir kendaraan diperuntukkan bagi penumpang yang akan menggunakan transportasi udara namun ingin memarkirkan kendaraannya

[4] Gambar I.1 Denah 1 Bandara Soekarno Hatta

[5] Gambar I.2 Denah 2 Bandara Soekarno Hatta

Pada bandara, terdapat sebuah istilah *gate* atau dalam bahasa Indonesia disebut dengan gerbang penerbangan. *Gate* merupakan bagian dari terminal. *Gate* adalah tempat untuk penumpang untuk menunggu keberangkatan pesawat atau tempat ketika penumpang turun dari pesawat terbang. *Gate* juga digunakan sebagai akses keluar menuju pesawat terbang yang diperuntukkan bagi penumpang beserta awak maskapai penerbangan.

Pada bandara, *Gate* digunakan secara bergantian. Hal ini dikarenakan tempat yang disediakan untuk menampung penumpang adalah terbatas. Skema bandara mengenai penggunaan *gate* secara garis besar adalah, pertama, penumpang akan melakukan *check-in* di *check-in counter*. Setelah itu, penumpang akan menuju ke sebuah *gate* tertentu seperti yang telah ditunjuk pada *boarding pass* yang dimiliki oleh penumpang. Pada *boarding pass* juga akan terdapat *boarding time*, yaitu waktu ketika penumpang diperbolehkan memasuki pesawat. Ketika sudah tiba waktu untuk *boarding* atau memasuki pesawat terbang, maka penumpang yang berada di tempat menunggu akan dipersilahkan untuk memasuki pesawat terbang, pada waktu inilah *gate* akan kosong dan dapat digunakan oleh penerbangan lain. Ketika pesawat mendarat pun, penumpang juga akan menuju *gate*.

Permasalahan yang kerap kali terjadi adalah, *gate* tidak cukup untuk menampung sejumlah penumpang yang akan melakukan perjalanan dengan pesawat, baik kepergian maupun kedatangan.. Situasi ini akan menyebabkan adanya antrian penerbangan pada bandara. Hal ini juga yang seringkali menyebabkan pesawat tidak dapat mendarat dan berputar-putar di udara karena belum ada *gate* yang tersedia dan mampu menampung sejumlah penumpang. Akibatnya, akan terjadi keterlambatan waktu kedatangan (*landing*) maupun keberangkatan (*take-off*). Tentu jika terjadi hal seperti ini, yang dirugikan adalah banyak pihak, baik penumpang maupun maskapai penerbangan. Kerugian yang paling dirasakan oleh penumpang ketika terjadi pengantrian *gate* adalah keterlambatan waktu penerbangan dan pendaratan yang seringkali tidak diinformasikan kepada penumpang. Padahal keterlambatan yang terjadi bisa sampai 20 menit. Untuk ukuran bandara internasional seperti bandara Soekarno-Hatta, Husein Sastranegara, atau Ir. Juanda yang setiap harinya terdapat lebih dari 500 penerbangan, problematika ini adalah hal yang serius. Terlebih, apabila terdapat penerbangan yang memerlukan transit terlebih dahulu, atau jika penumpang mengejar penerbangan lain ketika sudah mendarat. Selain itu, jika sarana *gate* pada bandara tidak memenuhi kebutuhan, tentunya akan menimbulkan kerugian pada bandara itu sendiri.

Pada permasalahan ini, diperlukan metode penyelesaian penjadwalan dan penempatan pesawat terbang pada *gate* yang sesuai yang dapat memberikan keuntungan terbesar dan meminimalisasi waktu bagi pesawat terbang ketika mengantri mendapatkan *gate*.

Algoritma yang digunakan pada makalah kali ini adalah algoritma *Dynamic Programming* dan penerapan *job scheduling* dalam menentukan *gate* untuk pesawat terbang. *Dynamic Programming* dipilih karena algoritma in mampu memberikan hasil yang optimum sehingga diharapkan dapat memberikan keuntungan yang maksimum pula bagi penumpang, maskapai penerbangan, dan bandara itu sendiri. Juga, pada makalah kali ini, akan diberikan prediksi berapa waktu yang dibutuhkan bagi penerbangan selanjutnya untuk mendapatkan *gate*. Sehingga kemudian, pada saat melakukan pengantrian di udara, pilot dapat memberikan informasi kepada penumpangnya berapa lama keterlambatan dari waktu mendarat seharusnya. Tentunya, penumpang akan banyak terbantu dengan adanya pemberian informasi demikian. Selain itu, penghitungan waktu antrian ini juga dapat menjadi bahan pertimbangan bagi pengelola bandara untuk menambahkan *gate* baru. Apabila waktu antrian lama, maka merupakan suatu urgensi yang perlu dipikirkan untuk membangun *gate* baru pada bandara.

II. Dasar Teori

A. *Algoritma Dynamic Programming*

Dynamic Programming atau program dinamis adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan *stage* atau tahapan yang menghasilkan solusi optimum dan solusi merupakan hasil dari keputusan yang saling berkaitan. *Stage* atau tahapan ini merupakan sub-masalah yang diproses secara berurutan. Pada dasarnya, program dinamis adalah melakukan optimasi pada penanganan kasus rekursif. Ketika didapat solusi rekursif yang dipanggil beberapa kali untuk input yang sama, maka hal ini dapat dioptimasi menggunakan program dinamis. Ide utamanya sederhana, yaitu menyimpan hasil yang didapatkan ketika

melakukan penghitungan untuk setiap sub-masalah ke sebuah tabel agar tidak perlu dilakukan penghitungan ulang ketika dibutuhkan data yang sama.

Pendekatan yang dimiliki oleh dynamic programming adalah serupa dengan *divide and conquer* pada pemecahan masalah menjadi beberapa bagian sub-masalah. Perbedaannya dengan *divide and conquer* adalah, program dinamis tidak menyelesaikan setiap sub-masalah secara terpisah, namun hasil yang didapat dari setiap sub-masalah akan disimpan untuk digunakan kemudian pada permasalahan yang sama.

Program dinamis dipakai ketika masalah yang dimiliki dapat dipecah menjadi beberapa sub-masalah sehingga hasil yang didapatkan pada setiap sub-masalah dapat digunakan kembali. Utamanya, program dinamis bertujuan untuk optimasi.

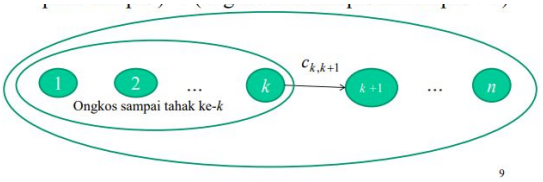
Program dinamis juga memiliki kemiripan dengan algoritma *greedy* dalam menentukan solusi yang paling maksimum dan optimum. Perbedaannya adalah, algoritma *greedy* hanya mempertimbangkan optimasi lokal sehingga hasil yang diberikan tidak selalu optimum. Pada program dinamis, setiap hasil yang didapatkan dari sub-masalah akan digunakan dan dipertimbangkan pada sub-masalah yang lebih besar. Pada program dinamis, hasil yang didapatkan selalu maksimum. Perbedaan lain dengan algoritma *greedy* adalah, pada algoritma *greedy*, hanya terdapat satu rangkaian keputusan yang dihasilkan, sedangkan pada program dinamis, rangkaian keputusan yang dipertimbangkan lebih dari satu.

Karakteristik penyelesaian persoalan dengan program dinamis adalah:

1. Pilihan yang mungkin dilakukan lebih dari satu
2. Solusi dari setiap tahapan sub-masalah dibangun dari hasil solusi tahap sebelumnya
3. Digunakannya persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap

Pada program dinamis terdapat prinsip optimalitas, yaitu setiap rangkaian keputusan yang optimal dibuat dengan menggunakan prinsip optimalitas, yaitu jika solusi total optimal, maka bagian solusi sampai tahap ke-k juga optimal.

Prinsip optimalitas berarti jika kita memulai pekerjaan dari tahap k hingga ke tahap $k+1$, kita dapat menggunakan hasil optimal dari tahap k tanpa harus menghitung dan kembali ke tahap awal.



[1] Gambar II.A.1 Skema program dinamis

Karakteristik persoalan program dinamis adalah:

1. Persoalan dapat dibagi menjadi beberapa tahap atau *stage* yang pada setiap tahapnya hanya diambil satu keputusan
2. Masing-masing tahap terdiri dari sejumlah status atau *state* yang berhubungan dengan tahap tersebut. Umumnya, status merupakan berbagai macam kemungkinan masukan yang ada pada tahap tersebut.
3. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya
4. Ongkos atau *cost* pada suatu tahap meningkat secara teratur atau *seady* seiring dengan peningkatan jumlah tahapan
5. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos pada tahap tersebut
6. Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusna yang dilakukan pada tahap sebelumnya
7. Adanya hubungan rekursif yang mengidentifikasikan keputusan terbaik untuk setiap status pada tahap k yang dapat memberikan keputusan terbaik untuk setiap status pada tahap $k+1$
8. Prinsip optimalitas berlaku pada persoalan tersebut.

Pada program dinamis, terdapat dua tipe pendekatan:

1. *Up-down* atau *forward* (maju)

   Program dinamis yang bergerak mulai dari tahap paling kecil (tahap ke-1) dan terus maju ke tahap 2, 3, 4, dan seterusnya hingga tahap ke-n.

   Prinsip optimalitas pada program dinamis maju adalah:

   ongkos pada tahap $k+1$ = (ongkos yang dihasilkan pada tahap k) + (ongkos dari tahap k ke tahap $k+1$)

   $k = 1, 2, \ldots, n-1$

[1]Gambar II.A.2 Prinsip optimalitas pada pendekatan maju

2. *Bottom-up* atau *backward* (mundur)

   Program dinamis yang bergerak mulai dari tahap paling besar (tahap ke-n) dan terus mundur ke tahap n-1, n-2, n-3, dan seterusnya hingga tahap ke-1.

Prinsip optimalitas pada program dinamis maju adalah:

ongkos pada tahap k = (ongkos yang dihasilkan pada tahap $k+1$) + (ongkos dari tahap $k+1$ ke tahap k)

$k = n, n-1, \ldots, 1$

[1]Gambar II.A.3 Prinsip optimalitas pada pendekatan mundur

B. Job Scheduling

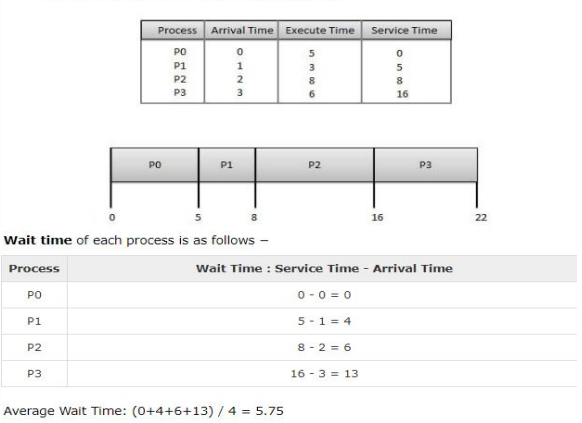
Job scheduling atau biasa juga disebut dengan *job sequencing* adalah sebuah metode penyusunan jadwal yang komponennya ada dua, yaitu *resource* dan *client*. *Resource* adalah sumber yang akan menangani kebutuhan *client*. Sebagai contoh, dapat diambil mesin fotokopi adalah *resource* dan objek yang akan difotokopi adalah *client*. Contoh lain adalah armada travel merupakan *resource* dan penumpang yang akan menaiki travel tersebut adalah *client.* Pada persoalan *Airport Gate Assignment Problem*, yang bertindak sebagai *resource* adalah *gate* yang tersedia pada bandara dan pesawat terbang adalah *client* yang harus dilayani oleh *gate.*

Algoritma *job scheduling* adalah algoritma yang digunakan oleh *CPU* untuk melakukan sebuah kerja tertentu. *Job scheduling* digunakan agar tercipta penjadwalan pada setiap pekerjaan agar memberikan hasil pekerjaan yang optimum. Pada dasarnya, prinsip yang digunakan untuk mendapatkan hasil yang maksimum adalah meminimalisir waktu menunggu *client* dan mengalokasikan *resource* yang tersedia ke setiap pekerjaan yang mungkin sehingga tercipta penjadwalan pekerjaan yang memberikan hasil maksimum. Secara garis besar, *job scheduling* dibedakan menjadi dua jenis, yaitu *preemptive* dan *non-preemptive.* Penjadwalan secara *preemptive* adalah penjadwalan dimana ketika pekerjaan telah menggunakan *resource,* atau dilayani oleh *resource*, maka pekerjaan ini dapat diinterupsi oleh pekerjaan lain yang membutuhkan *resource* yang sama. Sedangkan untuk *non-preemptive,* ketika pekerjaan telah dilayani oleh *resource,* maka pekerjaan ini tidak dapat diinterupsi hingga pekerjaan selesai menggunakan *resource* dan melepas *resource* secara sukarela untuk selanjutnya digunakan oleh pekerjaan lain.

Terdapat beberapa algoritma untuk menyelesaikan permasalahan *job scheduling*, diantaranya adalah:

1. *First-Come First-Serve (FCFS)*

   Pekerjaan dieksekusi atau dijalankan berdasarkan urutan. Pekerjaan yang lebih dulu datang akan dilayani terlebih dahulu. *FCFS* merupakan bagian dari *non-preemptive*. Kelebihan yang ketika menggunakan algoritma ini adalah mudah dimengerti dan diimplementasikan, namun kekurangannya adalah tidak mangkus dan waktu menunggu *client* cenderung lebih lama dan besar



Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

Wait time of each process is as follows –

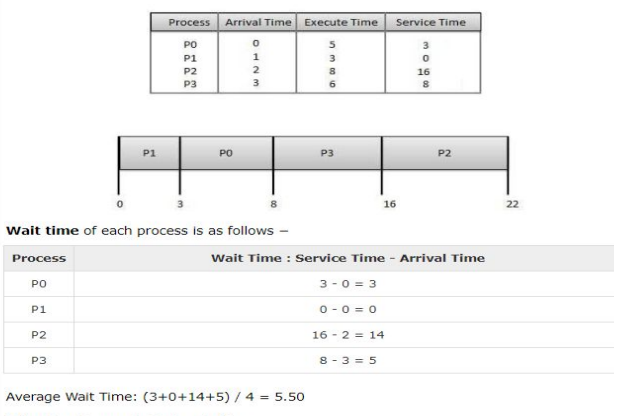
Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

[3]Gambar II.B.1 Ilustrasi *job scheduling* menggunakan *First-Come First-Serve*

2. *Shortest-Job Next (SJN)*

   Dapat disebut juga dengan *shortest-job first (SJF)*. Algoritma ini termasuk ke dalam *non-preemptive*. *Shortest-job first* sangat bagus digunakan untuk meminimalisir waktu menunggu dari *client*. Algoritma ini akan menjalankan pekerjaan yang memiliki waktu eksekusi paling rendah atau paling kecil terlebih dahulu.



Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8

Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	3 - 0 = 3
P1	0 - 0 = 0
P2	16 - 2 = 14
P3	8 - 3 = 5

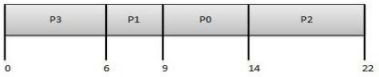
Average Wait Time: (3+0+14+5) / 4 = 5.50

[3]Gambar II.B.2 Ilustrasi *job scheduling* menggunakan *Shortest-Job Next*

3. *Priority-Based Scheduling*

   Priority-Based scheduling merupakan contoh dari *non-preemptive scheduling*. Setiap pekerjaan memiliki prioritasnya masing-masing. Pada algoritma ini, pekerjaan yang memiliki prioritas lebih tinggi akan dieksekusi atau dilayani terlebih dahulu oleh *resource*. Ketika terdapat lebih dari satu pekerjaan yang memiliki prioritas yang sama, maka yang dieksekusi atau dijalankan terlebih dahulu adalah yang waktu kedatangannya lebih awal.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	9 - 0 = 9
P1	6 - 1 = 5
P2	14 - 2 = 12
P3	0 - 0 = 0

Average Wait Time: (9+5+12+0) / 4 = 6.5

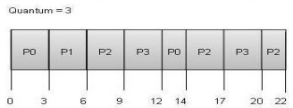
[3]Gambar II.B.3 Ilustrasi *job scheduling* menggunakan *Priority-based scheduling*

4. *Shortest Remaining Time*

Shortest remaining time merupakan versi *preemptive* dari algoritma *shortest-job next. Resource* akan melayani pekerjaan yang memiliki waktu eksekusi paling kecil, namun proses pelayanan ini dapat diinterupsi ketika terdapat pekerjaan baru yang memiliki waktu eksekusi lebih kecil. Jika terjadi interupsi, maka pekerjaan yang mengalami interupsi akan melanjutkan pekerjaannya ketika program yang lain telah selesai.

5. *Round Robin Scheduling*

Round robin merupakan sebuah contoh dari algoritma *preemptive.* Pada setiap pekerjaan, terdapat waktu eksak untuk melakukan eksekusi atau pelayanan oleh *resource*. Waktu ini disebut dengan *quantum*. Ketika terdapat suatu pekerjaan yang dieksekusi atau dilayani oleh *resource*, maka pekerjaan ini dapat diinterupsi oleh pekerjaan lain hingga pekerjaan lain ini selesai dieksekusi atau dilayani oleh *resource*. Kemudian, pekerjaan yang telah terinterupsi akan melanjutkan pekerjaannya hingga selesai. Pada algoritma *round robin* digunakan *context switching* untuk menyimpan status dari pekerjaan yang diinterupsi.



Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

[3]Gambar II.B.4 Ilustrasi *job scheduling* menggunakan *Round-robin*

6. *Multiple-Level Queues Scheduling*

Multiple-Level Queues Scheduling bukan merupakan algoritma *scheduling* yang independen. Algoritma ini menggunakan algoritma lain untuk mengelompokkan pekerjaan yang memiliki karakteristik yang sama. *Multiple queues* digunakan untuk pekerjaan yang memiliki ciri dan karakteristik yang sama. Pada setiap *queue*, bisa saja terdapat algoritma *scheduling* yang berbeda. Setiap *queue* memiliki prioritasnya masing-masing.

C. Binary Search

Binary search atau dalam bahasa Indonesia disebut dengan pencarian biner adalah algoritma pencarian yang paling populer. Algoritma pencarian biner memiliki efisiensi yang tinggi sehingga algoritma ini sering digunakan untuk penyelesaian berbagai macam persoalan. Pencarian biner hanya berfungsi pada sekumpulan elemen yang terurut, sehingga ketika pencarian biner akan digunakan, elemen harus terurut terlebih dahulu. Prinsip dasar dari pencarian biner adalah melakukan proses pembagian ruang pencarian menjadi dua secara berulang-ulang hingga elemen yang dicari ditemukan atau hingga tidak dapat dibagi lagi dan elemen yang dicari tidak dapat ditemukan. Penggunaan dari pencarian biner sangat efektif untuk mengurangi waktu eksekusi program dalam pencarian elemen. Algoritma untuk pencarian biner adalah:

```
Kamus
    Const N : integer = 8  { misalkan jumlah elemen array maksimum = 8 }
    Type A = array [ 1 ..... N ] of integer
    Cari, BatasAtas, BatasBawah, Tengah : Integer
    Ketemu : boolean
ALGORITMA
    Input (cari) { meminta nilai data yang akan dicari}
    BatasAtas   ← 1 { indeks array dimulai dari 1 }
    BatasBawah  ← N
    Ketemu  ← False
    While (BatasAtas < BatasBawah) and (not ketemu) do
        Tengah ←  (BatasAtas  + BatasBawah) div 2
        If A [Tengah] = cari then
          Ketemu ← true
                Else
                    If ( A [Tengah]  < cari ) then { cari di bagian kanan }
                        BatasAtas ← Tengah + 1
                    Else
                        BatasBawah ← Tengah – 1 { cari di bagian kiri }
                    Endif
                Endif
    EndWhile
    If (ketemu) then
            Output ( 'Data berada di index nomor', Tengah )
    Else   Output ( 'Data tidak ditemukan' )
    Endif
```

Gambar II.C.1 Algoritma pencarian dengan *binary search*

III. Analisis Masalah dan Implementasi Algoritma Dynamic Programming

A. Analisis Masalah *Airport Gate Assignment Problem*

Airport Gate Assignment Problem (AGAP) merupakan permasalahan yang dihadapi oleh bandara setiap harinya. Berdasarkan dasar teori yang telah dibahas pada sebelumnya, permasalahan *Airport Gate Assignment Problem (AGAP)*

merupakan permasalah *job scheduling* dengan menggunakan prioritas waktu kedatangan. Dalam permasalahan ini, yang bertindak sebagai *resource* adalah *gate* pada bandara, sedangkan yang bertindak sebagai *client* adalah pesawat terbang itu sendiri. Pada sebuah bandara, jumlah *gate* yang tersedia lebih sedikit daripada pesawat terbang. Pada setiap pesawat terbang terdapat jadwal mendarat dan jadwal untuk kembali lepas landas. Selain itu, pada pesawat terbang juga terdapat bahan bakar yang hanya dapat digunakan untuk sekian waktu perjalanan. Berdasarkan beberapa aspek yang perlu dianalisis ini, maka *job scheduling* yang sesuai untuk menyelesaikan permasalahan ini adalah *multiple-level queue scheduling* dengan pemanfaatan *first-come first-serve* dan *priority-based scheduling*. Tipe yang digunakan adalah *non-preemptive*, yaitu pekerjaan tidak dapat diinterupsi oleh pekerjaan lain ketika telah berjalan.

Program dinamis digunakan dalam mengimplementasikan pencarian *gate* untuk setiap pesawat agar meminimalisir waktu tunggu dari pesawat untuk mengantri *gate*. Pada makalah ini, akan dibahas miniatur dari sebuah bandara. Miniatur bandara ini akan mulai beroperasi pada pukul 06.00 pagi dan akan mengakhiri operasi pada pukul 12.00 siang. Total waktu operasi dari miniatur bandara ini adalah 6 jam. Pada bandara ini, terdapat total empat *gate* yang dapat digunakan oleh pesawat terbang. Regulasi pada bandara ini adalah, ketika sebuah pesawat terbang memiliki selisih waktu waktu kedatangan dengan waktu keberangkatan lebih dari empat jam, maka pihak operasional bandara akan melakukan *towing*. *Towing* adalah pemindahan pesawat yang semula berada di *apron* menjadi ke *hangar*. Hal ini dilakukan agar pesawat lain dapat menggunakan *apron* yang sama sehingga optimasi *gate* dapat dilakukan. Pada miniatur bandara ini, proses *towing* memakan waktu tiga puluh menit.

Terdapat beberapa variabel yang digunakan dalam menyelesaikan masalah ini:

1. P: himpunan pesawat terbang yang akan beroperasi pada bandara
2. Q: himpunan *gate* pada bandara yang akan digunakan oleh pesawat terbang
3. nP: jumlah penerbangan
4. nQ: jumlah *gate*
5. $t1_i$: waktu kedatangan dari pesawat terbang ke-i
6. $t2_i$: waktu keberangkatan pesawat terbang ke-i

Hal yang perlu diperhatikan dalam melakukan perancangan solusi untuk permasalahan ini adalah:

1. Setiap pesawat terbang hanya dapat dialokasikan ke satu *gate* saja
2. Waktu kedatangan pada pesawat terbang lebih awal daripada waktu keberangkatannya
3. Tidak ada *gate* yang digunakan oleh lebih dari satu pesawat terbang pada waktu yang sama.

No	Air	Kedatangan			Keberangkatan		
		FLT	From	Time	FLT	To	Time
1	GT	347	Surabaya	6.00	235	Jakarta	7.45
2	GH	457	Bandung	6.20	643	Surabaya	7.55
3	TR	784	Pontianak	6.50	677	Makassar	8.10
4	TX	325	Lombok	7.15	745	Bali	8.40
5	DW	570	Bali	7.35	344	Jakarta	9.25
6	XQ	234	Surabaya	8.05	657	Bandung	9.40
7	QZ	458	Jogjakarta	8.15	235	Jakarta	10.05
8	GU	234	Makassar	8.55	678	Pontianak	10.35
9	IK	322	Bandung	9.15	890	Jogjakarta	10.50
10	CR	115	Medan	9.40	778	Surabaya	11.00
11	CT	527	Surabaya	10.20	431	Bandung	11.15
12	FB	489	Jogjakarta	11.00	339	Jakarta	11.40
13	AR	873	Malang	11.25	547	Surabata1	12.00

Tabel 3.A.1 Tabel jadwal keberangkatan dan kedatangan pesawat terbang pada miniatur bandara

B. Penggunaan dan Pengimplementasian Algoritma *Dynamic Programming*

Pada permasalahan kali ini, akan diusulkan penggunaan program dinamis untuk menyelesaikan *Airport Gate Scheduling Problem*.

Pada persoalan ini:

1. Tahap (*k)* adalah proses memilih dan menempatkan pesawat terbang pada gate
2. Status (y_k) menyatakan *gate* yang dialokasikan kepada pesawat pada setiap langkahnya
3. *Time* (*t)* merupakan waktu bagi pesawat terbang untuk menggunakan *gate*
4. *Arrival* (a_i) adalah waktu kedatangan pesawat ke-1
5. *Departure* (d_i) adalah waktu keberangkatan pesawat ke-i
6. *Time Delay* (tD_i) adalah waktu yang digunakan pesawat ke-i untuk mengantri *gate*

Relasi keuntungan rekurens dari permasalahan ini adalah:

$f_1(y_1) = min(a,)$ *basis*

$f_k(y_k) = min(a_k , a_{k-1})$ *rekursif*

dengan k = *nP-1*

Secara garis besar, urutan algoritmanya adalah sebagai berikut:

1. Mengurutkan waktu kedatangan pesawat dari terkecil ke terbesar
2. Membuat larik kosong sejumlah gate yang akan diisi oleh pesawat terbang
3. Pada setiap himpunan pesawat terbang, melakukan pengecekan terhadap himpunan pesawat terbang, apabila ada terdapat *gate* yang kosong, maka pesawat terbang tersebut akan dialokasikan kepada *gate* yang kosong tersebut
4. Apabila semua *gate* telah terisi dan masih terdapat pesawat yang belum mendapatkan *gate*, maka *tD* dari pesawat akan ditambah per satu menit hingga sesuai dengan waktu kekosongan *gate*.
5. Apabila terdapat pesawat terbang yang membutuhkan *towing*, maka setelah mendarat, pesawat terbang akan langsung dipindahkan ke *hangar* agar *gate* dapat digunakan oleh pesawat lain dan membuat jam keberangkatan pesawat menjadi tiga puluh menit lebih awal karena waktu yang dibutuhkan untuk memindahkan pesawat adalah tiga puluh menit.

Kasus uji yang dilakukan pada makalah ini adalah adalah empat *gate* dengan jadwal penerbangan sebagai berikut:

Flight No.	From	To	Arrival	Departure
CT456	Badung	Surabaya	6.00	8.20
RX569	Surabaya	Pontianak	6.20	9.45
ZR908	Makassar	Jakarta	6.40	10.05
DF597	Denpasar	Yogyakarta	6.30	8.10
FG488	Jakarta	Palangkaraya	8.10	11.20
GH763	Malang	Bandung	9.20	11.10
TR578	Jakarta	Surabaya	9.00	11.00
GX511	Makassar	Palu	8.40	11.40
TY476	Bandung	Manado	8.55	11.55
IX909	Balikpapan	Samarinda	9.50	12.15
DF773	Batam	Jakarta	10.30	12.00
CX551	Semarang	Yogyakarta	10.20	11.50

Tabel 3.B.1 Tabel kasus uji

Hasil yang didapatkan dari program dalam bentuk tabel adalah sebagai berikut:

Gate 1:

Flight No.	Arrival	Departure	Delay Time
CT456	6.00	8.20	0 mins
GX511	8.40	11.40	0 mins
CX551	10.30 → 11.40	12.00	70 mins

Tabel 3.B.2 Tabel Hasil kasus uji untuk *gate* 1

Gate 2:

Flight No.	Arrival	Departure	Delay Time
RX569	6.20	9.45	0 mins
TY476	8.55 → 9.45	11.55	70 mins

Tabel 3.B.3 Tabel Hasil kasus uji untuk *gate* 2

Gate 3:

Flight No.	Arrival	Departure	Delay Time
DF597	6.30	8.10	0 mins
FG488	8.10	11.20	0 mins
DF773	10.20 → 11.20	11.50	60 mins

Tabel 3.B.4 Tabel Hasil kasus uji untuk *gate* 3

Gate 4:

Flight No.	Arrival	Departure	Delay Time
ZR908	6.40	10.05	0 mins
TR578	9.00 → 10.05	11.00	55 mins
IX909	9.50 → 11.00	12.15	70 mins

Tabel 3.B.5 Tabel Hasil kasus uji untuk *gate* 4

Pada keluaran yang dihasilkan oleh program, *scheduling* yang paling optimum terdapat pada Tabel 3.5.2 untuk *gate* 1, Tabel 3.5.3 untuk *gate* 2, Tabel 3.5.4 untuk *gate* 3, dan Tabel 3.5.3 untuk *gate* 4. Pada baris pertama di setiap tabel, *delay time* selalu nol menit, sedangkan pada baris selanjutnya, *delay time* bervariasi tergantung pada waktu kedatangan yang seharusnya dengan ketersediaan *gate* pada bandara. *Delay time* ini diperlukan dan sangat penting untuk diinformasikan kepada penumpang agar penumpang dapat mengetahui waktu pasti pesawat terbang akan mendarat di bandara. Terdapat lima pesawat terbang yang mengalami *delay.* Sekarang, akan dicoba untuk menambahkan satu *gate,* sehingga jumlah *gate* adalah 5.

Gate 1:

Flight No.	Arrival	Departure	Delay Time
CT456	6.00	8.20	0 mins
TY476	8.55	11.55	0 mins

Tabel 3.B.5 Tabel Hasil kasus uji untuk *gate* 1

Gate 2:

Flight No.	Arrival	Departure	Delay Time
RX569	6.20	9.45	0 mins
TR578	9.00 → 9.45	10.20	45 mins
IX909	9.50 → 10.20	12.15	30 mins

Tabel 3.B.7 Tabel Hasil kasus uji untuk *gate* 2

Gate 3:

Flight No.	Arrival	Departure	Delay Time
DF597	6.30	8.10	0 mins
GX511	8.40	11.40	0 mins

Tabel 3.B.8 Tabel Hasil kasus uji untuk *gate* 3

Gate 4:

Flight No.	Arrival	Departure	Delay Time
ZR908	6.40	10.05	0 mins
GH763	9.20 → 10.05	11.00	45 mins
CX551	10.20 → 11.00	11.50	40 mins

Tabel 3.B.9 Tabel Hasil kasus uji untuk *gate* 4

Gate 5:

Flight No.	Arrival	Departure	Delay Time
FG488	8.10	11.20	0 mins
DF772	10.30 → 11.20	12.00	70 mins

Tabel 3.B.10 Tabel Hasil kasus uji untuk *gate* 5

Jika menambahkan satu buah *gate*, maka *delay time* setiap pesawat terbang cenderung lebih singkat sehingga ini akan membuat keuntungan yang lebih besar. Dari segi penumpang juga diuntungkan. Informasi-informasi yang didapatkan dari makalah ini berguna untuk pengembangan pariwisata dan transportasi di Indonesia

IV. Kesimpulan dan Saran

Kesimpulan yang didapat dari makalah ini adalah pada *Airport Gate Assignment Problem (AGAP)* dapat diselesaikan dengan menggunakan algoritma *dynamic programming* atau program dinamis dengan memanfaatkan *job scheduling*. Hasil yang didapat pada setiap penyelesaiannya adalah optimum, yaitu setiap *gate* akan digunakan oleh pesawat terbang yang paling maksimum. Selain itu, penambahan *gate* pada bandara juga akan mempengaruhi efektifitas dan efisiensi dari *scheduling* yang dilakukan oleh program.

Saran untuk pengembangan kedepannya adalah dapat digunakan variabel-variabel yang lebih beragam agar hasil yang didapatkan lebih dekat pada kenyataannya. Simulasi bandara yang dibuat oleh penulis pada makalah ini hanyalah miniatur yang sangat sederhana tanpa mempertimbangkan aspek waktu mobilisasi penumpang dari *gate* hingga ke pesawat. Jumlah penumpang pada pesawat terbang juga belum dipertimbangkan. Sehingga, penulis berharap agar kedepannya ada yang mengembangan dengan menambahkan aspek-aspek tersebut sebagai bahan pertimbangan dalam menentukan *gate* yang ditempati oleh pesawat.

V. Appendiks

Implementasi dari program yang telah penulis buat dalam menyelesaikan tugas makalah Strategi Algoritma ini dapat diakses pada halaman GitHub milik penulis pada tautan https://github.com/afvianiryzki/agap-solver. Program ini dikembangakan dengan bahasa Python.

Ucapan Terimakasih

Pertama-tama, penulis mengucapkan syukur kepada Allah swt. yang telah mencurahkan rizki dan rahmatnya sehingga penulis dapat menyelesaikan makalah ini tepat waktu. Penulis juga berterimakasih kepada orangtua penulis yang telah memberikan dukungan dalam proses penyusunan makalah ini. Tak lupa, ucapan terimakasih ditujukan kepada Dosen Pengampu Mata Kuliah Strategi Algoritma K-03, Bapak Rinaldi Munir yang telah memberikan ilmu yang tak terkira berharganya. Terimakasih juga penulis sampaikan kepada teman-teman penulis yang selalu ada untuk memberikan semangat dalam mengerjakan makalah ini.

Akhir kata, penulis memohon maaf karena makalah ini belum sempurna dan masih terdapat kekurangan baik yang disengaja maupun yang tidak disengaja. Semoga kedepannya makalah ini dapat dikembangkan lebih lanjut dan memberikan hasil yang optimum dengan mempertimbangkan banyak aspek seperti lama perpindahan penumpang dan jarak yang harus ditempuh penumpang untuk menuju ke pesawat terbang.

References

[1] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algortima. Program Studi Teknik Informatika ITB. 2018.

[2] A. Silberschatz, P.B. Galvin, and G. Gagne. 2013. Operating System Concepts 9th Edition. New Jersey: John Wiley & Sons, Inc.

[3] https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.html (terakhir diakses pada 26 April 2019)

[4] http://bandarasoekarnohatta.com/peta-terminal-2-bandara-soekarno-hatta-cgk-cengkareng-2.info (terakhir diakses pada 26 April 2019)

[5] https://tangerangonline.id/2018/07/04/progres-pembangunan-apron-cargo-village-bandara-soetta/ (terakhir diakses pada 26 April 2019)

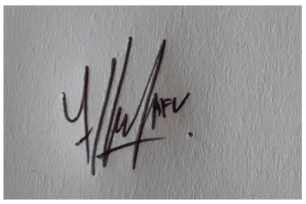
[6] Ding, H., Lim, A., Rodrigues, B. and Zhu, Y. (2004). Aircraft and gate scheduling optimization at airports, 37th Hawaii International Conference on System Sciences, 3, 30074b.

[7] Yan, S. and Chang, C. M. (1998). A network model for gate assignment, Journal of advanced Transportation, 32, 176–189.

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, 26 April 2019



Ainun Fitryh Vianiryzki