# Optimalisasi *Airport Gate Assignment Scheduling* Menggunakan Algoritma *Greedy*

Makalah IF2211 Strategi Algoritma

Rabbi Fijar Mayoza (13516081)
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*13516081@std.stei.itb.ac.id*

***Abstrak*—Sebuah bandar udara besar memiliki aktivitas penerbangan yang banyak dan sibuk. Hampir selama 24 jam suatu bandar udara beroperasi, setiap selang beberapa menit ada pesawat yang lepas landas dan mendarat secara bergantian. Pesawat-pesawat itu harus parkir di apron untuk menurunkan dan memuat penumpang. Namun jika dibandingkan dengan jumlah penerbangan, jumlah *gate* yang tersedia sebagai pintu masuk dan keluar penumpang dari dan menuju pesawat amatlah terbatas. Oleh karena itu, dibutuhkan sebuah solusi agar pemakaian *airport gate* yang jumlahnya terbatas itu optimal sehingga tidak menghambat kelancaran penerbangan. Makalah ini akan membahas penggunaan Algoritma Greedy sebagai strategi mengoptimalisasi penggunaan *airport gate* di bandara.**

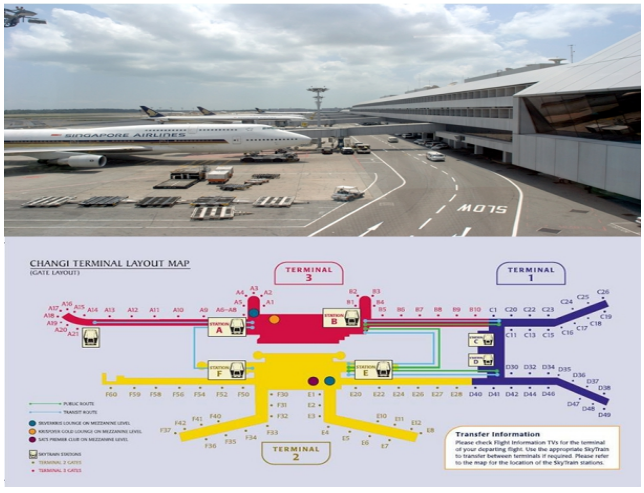
***Kata Kunci — Greedy; Airport gate;* Optimalisasi; *Scheduling***

## I. PENDAHULUAN

Sebuah *gate* atau gerbang dalam penerbangan adalah sebuah bagian dalam terminal bandar udara untuk memindahkan penumpang dan awak maskapai penerbangan ke dalam pesawat terbang. Sebuah gerbang umumnya terdiri dari ruang tunggu, *counter*, pintu keluar, tangga atau *elevator*, dan garbarata. Sebuah gerbang dipakai untuk menampung para penumpang menunggu kedatangan pesawat. Ketika pesawat sudah tiba, penumpang keluar dari pesawat dan menuju terminal melalui gerbang tersebut. Setelah itu akan diumumkan bahwa pesawat telah siap untuk diberangkatkan. Maka penumpang yang telah menunggu di ruang tunggu bergegas masuk ke pesawat. Akhirnya gerbang tersebut berada dalam keadaan kosong. Gerbang yang kosong dapat dipakai untuk aktivitas penerbangan selanjutnya. Begitulah rangkaian aktivitas penggunaan sebuah gerbang di bandar udara. Siklus ini terjadi secara berulang-ulang setiap hari.

Masalah yang muncul kemudian adalah apakah gerbang ini mampu menampung seluruh aktivitas kedatangan dan keberangkatan penumpang di suatu bandar udara. Jumlah gerbang yang terlalu sedikit di bandar udara akan berdampak buruk bagi penerbangan. Akan banyak penerbangan yang terlalu lama mengantre hanya untuk bergantian menggunakan gerbang. Apalagi di sebuah bandar udara internasional yang super sibuk yang memiliki jadwal penerbangan yang sangat padat, seperti Bandara Soekarno-Hatta di Jakarta dan Bandara Changi di Singapura. Kecepatan dan ketepatan waktu dari proses kedatangan dan keberangkatan penerbangan sangat dibutuhkan. Apabila ada keterlambatan, maka akan dihasilkan kerugian yang tidak kecil. Di sisi lain membangun gerbang baru juga membutuhkan biaya yang tidak sedikit sehingga memerlukan pertimbangan yang matang. Pemasalahan ini malah menjadi semakin kompleks. Untuk itu diperlukan strategi yang optimal agar penggunaan gerbang dapat maksimal dengan mengeluarkan biaya seminimal mungkin. Dalam pemasalahan optimalisasi seperti ini, algoritma *greedy* tepat untuk diimplementasikan.

Optimalisasi penggunaan *airport gate* dapat dilakukan menggunakan algoritma *greedy.* Algoritma *greedy* akan mengatur *scheduling* pemakaian sebuah gerbang dengan menyesuaikan jadwal penerbangan yang dimiliki suatu bandara dalam sehari. Dengan optimalisasi ini maka penggunaan gerbang-gerbang di bandara menjadi efektif dan efisien. Optimalisasi ini akan meminimalisir terjadinya antrean panjang pesawat di apron, menumpuknya penumpang di terminal, dan tak ter-*assign*-nya pesawat ke gerbang. Strategi optimalisasi ini juga dapat dikembangkan untuk merumuskan kebijakan pembangunan suatu bandara apabila jumlah penerbangan yang harus dilayani semakin meningkat.




**Gambar 1.** Sebaran gerbang di Bandara Internasional Changi, Singapura. Sumber: https://www.theseatinthemiddle.com/2017/03/28/lounge-review-singapore-changi-airport-dnata-lounge-terminal-1/

## II. DASAR TEORI

### ~~A.~~ *Algoritma Greedy*

Algoritma Greedy adalah salah satu algoritma dasar dalam pemecahan masalah. Secara etimologis, *Greedy* artinya rakus atau tamak. Hal ini sesuai dengan prinsip dari algoritma *greedy*, yaitu langsung mengambil nilai terbaik dari subpersoalan saat itu tanpa memperhatikan konsekuensi ke depan. Algoritma *Greedy* membentuk solusi langkah per langkah. Dari setiap langkah akan diambil nilai yang paling menguntungkan, yaitu nilai optimum lokal. Setelah pindah ke langkah selanjutnya, keputusan yang telah diambil pada langkah sebelumnya tidak dapat diubah lagi. Semua optimum lokal yang diperoleh diasusmsikan mengarah ke solusi optimum global, yaitu solusi paling menguntungkan dari keseluruhan kemungkinan solusi. Prinsip inilah yang menjadikan algoritma *greedy* biasa dipakai untuk memecahkan masalah optimasi.

Persoalan optimasi dalam konteks algoritma *greedy* disusun oleh elemen-elemen sebagai berikut.

*1) Himpunan Kandidat*

Himpunan ini berisi elemen-elemen pembentuk solusi.

*2) Himpunan Solusi*

Himpunan ini berisi kandidat-kandidat yang terpilih sebagai solusi persoalan. Maka himpunan solusi merupakan himpunan bagian dari himpunan kandidat.

*3) Fungsi Seleksi*

Fungsi untuk memilih kandidat yang paling memnungkinkan mencapai solusi optimal. Fungsi ini dipanggil pada setiap langkah. Kandidat yang sudah dipilih pada suatu langkah tidak pernah dipertimbangkan kembali pada langkah selanjutnya. Fungsi seleksi biasanya memilih nilai terbesar ataupun terkecil dalam himpunan bertipe data numerik.

*4) Fungsi Kelayakan*

Fungsi untuk memeriksa kelayakan kandidat yang dipilih. Syarat suatu kandidat layak menjadi solusi yaitu kandidat yang dipilih tidak melanggar *constraints* yang ada. Jika kandidat layak, kandidat dimasukkan ke dalam himpunan solusi. Sedangkan jika tidak layak maka kandidat tersebut dibuang dan tidak pernah dipertimbangkan lagi.

*5) Fungsi Obyektif*

Disebut juga dengan fungsi optimisasi, yaitu fungsi yang memaksimumkan atau meminimumkan nilai solusi.

Secara umum, skema algoritma *greedy* dapat dirumuskan sebagai berikut.

- Inisiasi himpunan solusi dengan himpunan kosong.
- *While* solusi belum lengkap *and* himpunan kandidat belum kosong *do*:
    - Dengan fungsi seleksi pilih sebuah kandidat dari himpunan kandidat.
    - Kurangi himpunan kandidat dengan elemen kandidat yang telah dipilih.
    - Dengan fungsi kelayakan periksa apakah kandidat yang telah dipilih tidak melanggar *constraints.* Jika layak maka masukkan kandidat tersebut ke dalam himpunan solusi. Jika tidak layak maka buang dan jangan pertimbangkan lagi.
    - Dengan fungsi obyektif, periksa apakah himpunan solusi sudah memberikan solusi yang lengkap. Jika ya, keluar dari *loop*. Jika tidak, lanjutkan *loop.*

Algoritma *greedy* jauh lebih cepat jika dibandingkan dengan algoritma *exhaustive search,* karena semua alternatif solusi tidak perlu diperiksa melainkan cukup menimbang solusi optimum lokal saja. Selain itu, ide algoritma *greedy* juga sederhana sehingga mudah dipahami. Akan tetapi, tidak semua masalah yang diselesaikan menggunakan algoritma *greedy* menghasilkan solusi yang benar-benar optimum. Walaupun demikian, solusi tersebut dekat dengan solusi optimum sebenarnya sehingga algoritma *greedy* masih bisa digunakan sebagai pendekatan heuristik untuk mengaproksimasi solusi optimum.

### ~~B.~~ *Job Scheduling Problem*

*Job scheduling* atau penjadwalan proses adalah penyusunan jadwal dari sebuah sistem yang terdiri atas *server* yang harus melayani atau memproses sejumlah *client*. Hubungan dalam sistem *client* dan *server* itu bisa berupa *processor* di sistem operasi, TCP/IP, kasir dan nasabah bank, ataupun sistem *airport gate assignment* yang akan dibahas dalam makalah ini. Dalam sistem *airport gate assignment*, gerbang sebagai *server* harus melayani pesawat yang berperan sebagai *client* untuk memproses perpindahan penumpang.

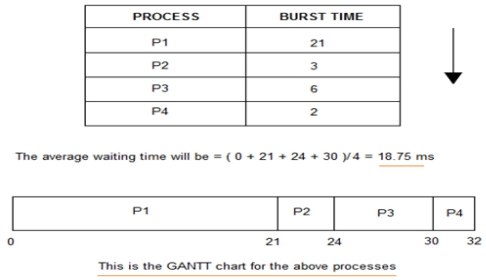
Algoritma *job scheduling* adalah algoritma yang bertujuan untuk memaksimalkan kerja dari *server* dalam melayani *client* dan meminimalkan waktu kerja sistem atau waktu tunggu *client*. Algoritma *job scheduling* terdiri dari dua jenis berdasarkan eksekusinya. Yang pertama adalah *pre-emptive,* yaitu algoritma yang membolehkan sebuah proses yang sedang berlangsung untuk diinterupsi dalam kondisi tertentu. Jenis kedua adalah *non pre-emptive,* yaitu algoritma yang tidak membolehkan sebuah proses yang sedang berlangsung untuk diinterupsi dalam kondisi apapun sampai semua *client* selesai diproses. Berdasarkan tenggat waktu, algoritma *job scheduling* juga dapat dibagi menjadi dua jenis. Yang pertama adalah *job with deadline,* yaitu proses yang memiliki tenggat waktu pegerjaan. Apabila tenggat waktu dilampaui dan pemrosesan belum selesai, maka proses tersebut sudah tidak boleh diselesaikan. Jenis kedua adalah *job without deadline,* yaitu proses yang tidak memiliki tenggat waktu pengerjaan.

Persoalan *job scheduling* dapat diselesaikan menggunakan salah satu varian algoritma-algoritma berikut.

*1) First Come First Serve Scheduling*

Algoritma ini menggunakan prinsip *first in first out (FIFO). Client* yang datang paling awal akan dilayani terlebih dahulu. FCFS merupakan algoritma yang

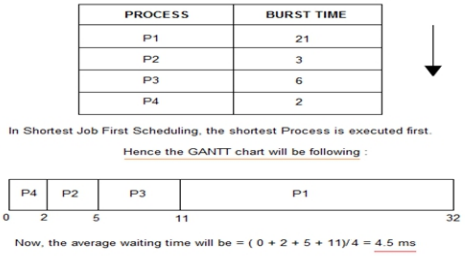
dieksekusi secara *non pre-emptive*. Keuntungan dari algoritma ini yaitu mudah diimplementasikan memakai struktur data Queue. Kerugian dari algoritma ini yaitu memunculkan *Convoy Effect,* yaitu efek yang membuat proses yang memiliki waktu pemrosesan singkat terhalang oleh proses yang memiliki waktu pemrosesan lama. Analoginya seperti mobil sedan yang terhalang di belakang truk besar yang melaju lambat di jalan raya.



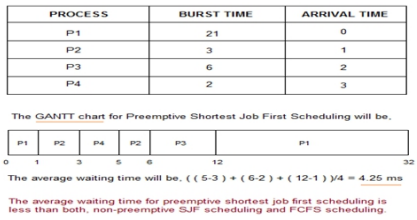

**Gambar 2.** Visualisasi Algoritma FCFS. Sumber: https://www.studytonight.com/operating-system/first-come-first-serve

*2) Shortest Job First Scheduling*

Algoritma ini merupakan algoritma *greedy*. Algoritma ini melayani *client* berdasarkan waktu pemrosesan *(burst time)*. Proses yang waktu pemrosesannya paling kecil akan dilayani terlebih dahulu. Algoritma FJS dapat dieksekusi secara *pre-emptive* dan *non pre-emptive.* Keuntungan dari algoritma ini yaitu dipastikan menghasilkan solusi paling optimal apabila waktu kedatangan semua proses sama. Kerugian dari algoritma ini yaitu jika proses tidak datang secara bersamaan maka akan terjadi *starvation*. *Starvation* adalah peristiwa ketika ada proses yang tidak pernah dilayani atau menunggu terlalu lama untuk dilayani.




(a)

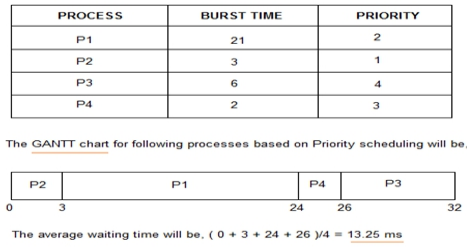



(b)

**Gambar 3.** Visualisasi Algoritma FJS (a) secara *non pre-emptive* dan (b) secara *pre-emptive.* Sumber: https://www.studytonight.com/operating-system/shortest-job-first

*3) Priority Scheduling*

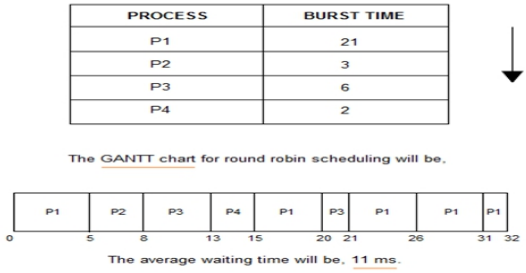
Algoritma ini merupakan algoritma *greedy.* Algoritma ini melayani *client* berdasarkan atribut prioritas. Proses yang memiliki nilai atribut prioritas paling kecil (prioritas paling tinggi) akan dilayani terlebih dahulu. Algoritma ini dieksekusi secara *pre-emptive.* Keuntungan algoritma ini adalah pemrosesan dilakukan berdasarkan tingkat urgensi. Sedangkan kerugian dari algoritma ini yaitu dapat menimbulkan *starvation.*




**Gambar 4.** Visualisasi Algoritma *Priority Scheduling*. Sumber: https://www.studytonight.com/operating-system/priority-scheduling

*4) Round Robin Scheduling*

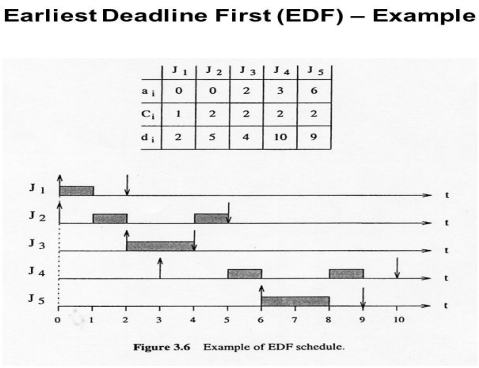
Algoritma ini menggunakan prinsip FIFO tetapi memiliki batasan waktu eksekusi yang disebut *quantum time.* Ketika *quantum time* telah terlampaui maka sebuah proses yang belum selesai diproses akan di-*entry*-kan kembali ke antrean paling belakang. Algoritma ini dieksekusi secara *pre-emptive.*




**Gambar 5.** Visualisasi Algoritma *Round Robin Scheduling* dengan *quantum time* 5. Sumber: https://www.studytonight.com/operating-system/round-robin-scheduling

*5) Earliest Deadline First Scheduling*

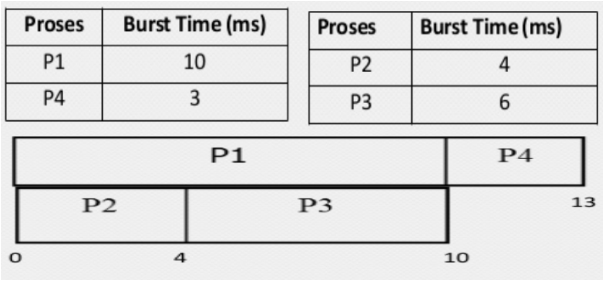
Algoritma ini berprinsip sama dengan *Priority Scheduling.* Yang membedakan adalah algoritma ini merupakan algoritma *Job Scheduling with Deadline.* Nilai prioritas proses ditentukan berdasarkan tenggat waktu. Semakin dekat tenggat waktu maka prioritas akan semakin tinggi. Tujuan algoritma ini adalah untuk meminimasi kemungkinan proses melampaui tenggat waktu.




**Gambar 6.** Visualisasi Algoritma *Earliest Deadline First Scheduling* . Sumber: http://slideplayer.com/slide/5148102/

*6) Multi-Processor Scheduling*

Algoritma ini bekerja dalam sistem *multi-processor* yaitu sistem yang memiliki *server* sejumlah lebih dari satu. Contoh penggunaan algoritma ini yang paling umum adalah *processing* pada komputer *multi-core processor*. Setiap *processor* bekerja secara paralel. Setiap *processor* bisa mengelola satu antrean secara bersama-sama atau bisa memiliki antrean sendiri-sendiri untuk dikelola masing-masing *processor*. Berdasarkan cara kerjanya algoritma *multi-processor scheduling* terbagi menjadi dua jenis. Yang pertama adalah *symmetric multiprocessing,* yaitu algoritma yang membolehkan setiap *processor* memiliki otonomi dalam memroses. Jenis kedua adalah *asymmetric multiprocessing,* yaitu algoritma yang menjadikan satu *processor* sebagai tuan (*master*) dan *processor* lainnya sebagai budak (*slave*).




**Gambar 7.** Visualisasi Algoritma *Multi-processor Scheduling* . Sumber: https://www.slideshare.net/TriSugihartono1/ch-02-multiprocessing-system

## III. Analisis Masalah dan Implementasi Algoritma

### *A. Studi Kasus Permasalahan Airport Gate Assignment Scheduling*

*Airport Gate Assignment Scheduling* merupakan salah satu masalah operasional sehari-hari yang dimiliki setiap bandar udara. Permasalahan ini merupakan permasalahan *job scheduling* antara *client* dan *server.* Gerbang sebagai *server* harus melayani pesawat sebagai *client*-nya. Karena jumlah gerbang di bandara pada umumnya berjumlah lebih dari satu, maka permasalahan ini diselesaikan dengan pendekatan *multi-processor scheduling.* Dalam permasalahan ini, proses yang terjadi adalah sebuah pesawat memakai sebuah gerbang. Asumsikan waktu proses dimulai sama dengan waktu kedatangan sebuah pesawat dan waktu proses berakhir sama dengan waktu keberangkatan sebuah pesawat. Dengan demikian selisih dari kedua waktu tersebut adalah *burst time.* Karena pesawat diharuskan untuk berangkat tepat waktu, maka dapat diasumsikan bahwa waktu keberangkatan merupakan tenggat waktu dari setiap proses. Semua proses diharapkan selesai tanpa melampaui tenggat waktu. *Goal* dari penyelesaian masalah ini adalah meminimasi jumlah *ungated flights* (penerbangan yang tidak dapat dilayani oleh gerbang) dan menentukan efektivitas pelayanan penerbangan.


**Gambar 8.** Jadwal penerbangan. Sumber: https://www.videoblocks.com/video/flight-information-on-airport-arrivals-departures-board-timetable-and-schedules-s4hdghkgin34gpep

Sebagai studi kasus, penulis akan memodelkan permasalahan *Airport Gate Assignment Scheduling* dalam sebuah bandar udara fiktif. Sebut saja namanya dengan Bandara Internasional Wakanda. Bandara Internasional Wakanda memiliki 3 buah gerbang dan hanya beroperasi selama 6 jam per hari. Sebagai prekondisi, jumlah penerbangan per hari selalu lebih dari jumlah gerbang yaitu 3. Untuk menyederhankan permasalahan, asumsikan ukuran dari setiap gerbang adalah sama sehingga pesawat jenis apapun dengan ukuran sebesar apapun dapat menggunakan ketiga gerbang tersebut.

Bandara Internasional Wakanda memiliki aturan khusus untuk penerbangan yang harus transit 4 jam atau lebih. Penerbangan tersebut harus melakukan *towing. Towing* terdiri dari 2 tahap. Tahap pertama adalah penarikan pesawat dari apron menuju *hangar*. Kemudian tahap kedua adalah mengembalikan pesawat dari *hangar* menuju apron. Waktu yang diberikan pesawat untuk memakai *gate* sebelum *towing* dan setelah *towing* adalah masing-masing 1 jam. Pada praktiknya mayoritas bandara di dunia mempunyai aturan *towing.* Tujuannya yaitu untuk memberikan kesempatan bagi pesawat lain untuk memakai gerbang yang sama.

Berikut merupakan variable-variabel yang dipakai.

- *N* : Himpunan penerbangan
- *M*: Himpunan gerbang yang tersedia di bandara
- *n* : Jumlah penerbangan

- $m$: Jumlah gerbang
- $a_i$ : Waktu kedatangan penerbangan ke-$i$
- $d_i$ : Waktu keberangkatan penerbangan ke-$i$
- $g_k$ : Gerbang nomor $k$

Berikut merupakan *constraints* yang harus diperhatikan.

- Setiap penerbangan hanya boleh di-*assign* ke 1 gerbang saja.
- Setiap waktu kedatangan penerbangan selalu lebih awal daripada waktu keberangkatannya.
- Tidak ada satupun dari 2 penerbangan yang di-*assign* dalam gerbang yang sama saling tumpang tindih.

Berikut merupakan data penerbangan Bandara Internasional Wakanda dalam sehari.

TABLE I. DATA PENERBANGAN BANDARA INTERNASIONAL WAKANDA

| ID | AL | Arrival | | | Departure | | |
|---|---|---|---|---|---|---|---|
| | | FLT | From | Time | FLT | To | Time |
| 1 | JT | 834 | Assen | 9:00 | 306 | Portland | 14:00 |
| 2 | GA | 597 | Waitakere | 9:06 | 226 | Le Mans | 10:00 |
| 3 | QG | 773 | Silverton | 9:10 | 185 | Patiala | 10:15 |
| 4 | GA | 278 | Waiblingen | 9:21 | 850 | Metairie | 10:25 |
| 5 | AA | 728 | São José | 9:40 | 491 | Meerle | 10:50 |
| 6 | SQ | 507 | Maastricht | 10:00 | 251 | Hijuelas | 10:25 |
| 7 | MH | 680 | Frankston | 10:15 | 562 | Pinto | 10:50 |
| 8 | BI | 310 | Meetkerke | 10:25 | 259 | Bandırma | 11:22 |
| 9 | BA | 244 | Badalona | 10:50 | 299 | Aurora | 11:30 |
| 10 | GA | 183 | Wolverhampton | 11:22 | 564 | Toledo | 11:55 |
| 11 | GA | 371 | Zeist | 11:37 | 491 | San Fabián | 12:10 |
| 12 | WA | 110 | İslahiye | 11:52 | 815 | Pontarlier | 12:20 |
| 13 | WA | 405 | Newport | 12:10 | 655 | Plymouth | 12:50 |
| 14 | WA | 808 | Hamburg | 12:20 | 568 | Strasbourg | 13:05 |
| 15 | CJ | 195 | Portland | 12:50 | 837 | Birmingham | 13:10 |
| 16 | CT | 119 | Bo'ness | 13:05 | 506 | Assiniboia | 13:35 |
| 17 | CX | 262 | Recoleta | 13:10 | 469 | Zierikzee | 13:55 |
| 18 | SV | 807 | Gujrat | 13:35 | 715 | Assen | 14:20 |
| 19 | EY | 766 | Cervino | 13:55 | 458 | Hamburg | 14:40 |
| 20 | EL | 315 | Onitsha | 14:00 | 166 | Osaka | 15:00 |

### ~~B.~~ *Penyelesaian Masalah Menggunakan Algoritma Greedy*

#### *1) Langkah-langkah Penyelesaian*

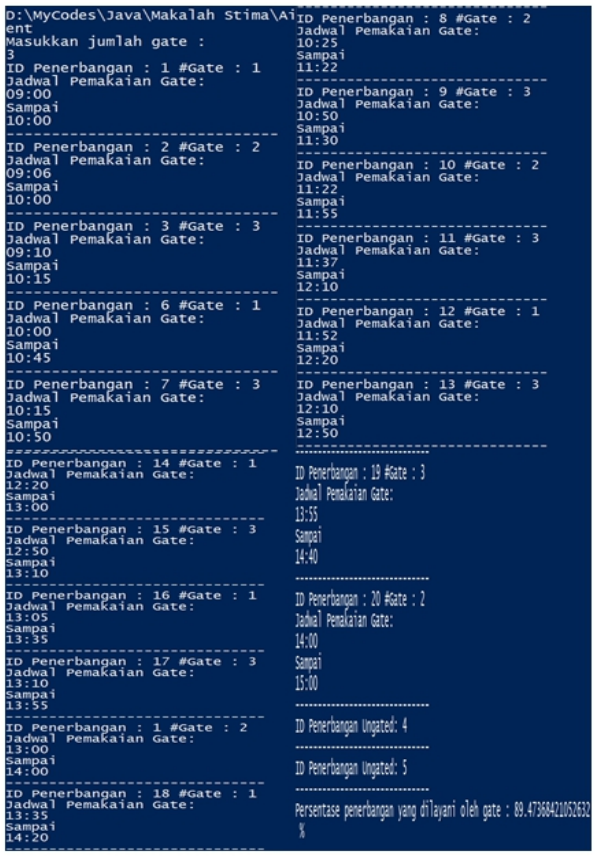
Untuk menyelesaikan masalah *gate assignment scheduling* di Bandara Internasional Wakanda, maka akan digunakan algoritma *greedy* dengan pendekatan heuristik untuk meminimasi jumlah penerbangan yang tidak dapat di-*assign* ke sebuah gerbang atau *ungated flights* menggunakan variabel dan data dari bagian A. Berikut merupakan langkah-langkah penyelesaian masalahnya.

1. Sort $N$ terurut membesar berdasarkan waktu keberangkatan $d_i$ *(1 <= i <= n)*
2. Set nilai $g_k$ = -1 Untuk $g_k$*(1 <= k <= m)*
3. Untuk setiap penerbangan *i* dalam *N*:
   a) Temukan $g_k$ sedemikian sehingga $g_k <= a_i$ dan pilih $g_k$ yang merupakan **nilai terdekat** dengan $a_i$.
   b) Jika $g_k$ ditemukan:
      Jika $d_i - a_i >= 4$ , aplikasikan *towing*: Penerbangan dibagi 2, bagian pertama interval waktunya menjadi $[a_i, a_i + 1]$, bagian kedua interval waktunya menjadi $[d_i - 1, d]$
      *assign* penerbangan *i* ke *gate* $g_k$, *update* $g_k = d_i$.
   c) Jika $g_k$ tidak ditemukan, *assign* penerbangan *i* ke *list "Ungated"*.
4. Tampilkan hasil *gate assignment*.

Source code dari program dapat dilihat melalui tautan yang penulis lampirkan di bagian Apendiks.

#### *2) Hasil dan Analisis*

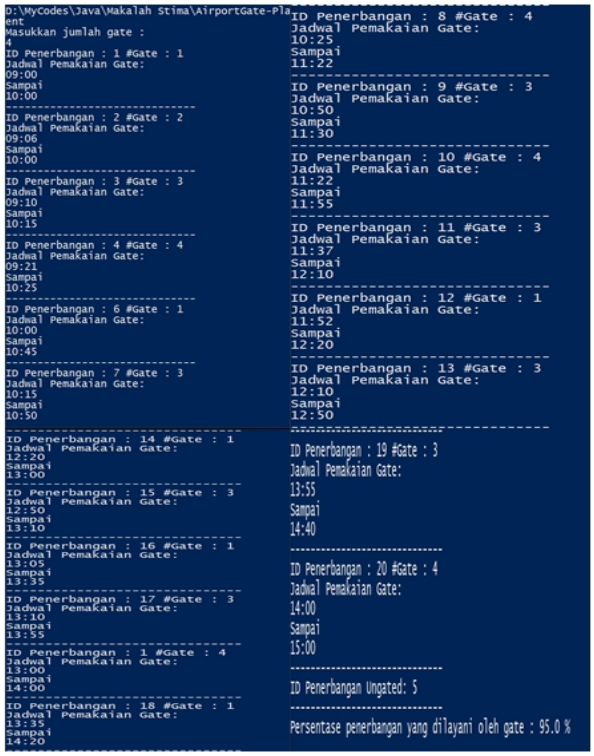
Dari hasil eksekusi program dengan masukan 20 penerbangan dan 3 buah gerbang, diperoleh *output* sebagai berikut.



**Gambar 9.** Hasil Eksekusi Program dengan $m = 3$.

Dari 20 penerbangan ada 18 penerbangan yang bisa di-assign ke gerbang, sedangkan ada 2 penerbangan yang tidak bisa di-assign ke gerbang, yaitu penerbangan ber-ID 4 dan 5. Dapat kita perhatikan bahwa penerbangan ber-ID 1 telah melakukan towing. Selanjutnya didapatkan pula persentase penerbangan yang dapat dilayani oleh semua gerbang adalah 89.47%. Angka ini sangat bermafaat bagi pihak pengambil kebijakan. Angka ini dapat menyimpulkan tingkat keefektivan dan efisiensi sebuah bandar udara dalam melayani penerbangan.

Misal akan diuji apakah dengan menambah jumlah gate Bandara Internasional Wakanda dari 3 menjadi 4 akan meningkatkan keefektifan dan efisiensi dalam melayani penerbangan. Dengan mengubah nilai *m* menjadi 4 di dalam program, didapatkan *output* berikut.




**Gambar 10.** Hasil Eksekusi Program dengan $m = 4$.

Dari hasil eksekusi program ternyata persentase penerbangan yang dapat dilayani oleh semua gerbang meningkat menjadi 95%. Tetapi angka 95% ini walaupun tinggi bukanlah nilai terbaik. Sebab jika kita perhatikan gerbang nomor 2 hanya dipakai sebanyak 1 kali. Ini menandakan bahwa dengan menambah jumlah gerbang menjadi 4, maka akan menimbulkan pemborosan. Oleh karena itu, tidak perlu dilakukan penambahan jumlah gerbang. Jadi, angka 89.47% merupakan nilai paling optimal.

## IV. KESIMPULAN

Berdasarkan hasil uji coba dari implementasi algoritma tersebut dan analisis, terdapat beberapa hal yang dapat disimpulkan:

1. Permasalahan *Airport Gate Assignment Scheduling* dapat diselesaikan dengan algoritma *greedy*. Hasil penyelesaiannya optimal dan sesuai dengan harapan.
2. Eksekusi algoritma *greedy* lebih cepat daripada menyelesaikan menggunakan algoritma *bruteforce* yang tidak menyusun urutan jadwal keberangkatan dan kedatangan tetapi langsung mencoba setiap kemungkinan, padahal kemungkinan suatu kombinasi melanggar *constraint* cukup besar sehingga membuang-buang waktu. Sedangkan dengan menggunakan algoritma *greedy,* yang diperhatikan hanyalah urutan dan selisih jam keberangatan dengan kedatangan.
3. Penyelesaian menggunakan algoritma *greedy* dapat menentukan tingkat efektivitas dan efisiensi suatu bandar udara dalam melayani penerbangan. Hasil penyelesaiannya juga bermanfaat bagi otoritas bandara untuk merumuskan kebijakan pembangunan bandara apabila terjadi peningkatan jumlah penerbangan.

## V. APENDIKS

Implementasi program yang dibuat penulis beserta hasil percobaan dapat diakses pada GitHub melalui tautan https://github.com/Pollycarpus/AirportGate-Planner. Program ditulis dengan bahasa Java sehingga diperlukan Java JDK 8 untuk mengeksekusi program.


## UCAPAN TERIMA KASIH

Pertama-tama penulis mengucapkan syukur kepada Allah SWT karena berkat rahmat dan pertolongan-Nya penulis dapat menyelesaikan perkuliahan Strategi Algoritma beserta tugas-tugasnya dengan lancar pada semester ini. Kemudian penulis juga berterima kasih kepada dosen-dosen pengajar mata kuliah IF2211 Strategi Algoritma, Drs. Nur Ulfa Maulidevi, Dr. Ir. Rinaldi Munir, M.T., dan Dr. Masayu Leylia Khodra ST., MT. atas bimbingan dan dukungan yang telah diberikan kepada penulis beserta ilmu-ilmu bermanfaat yang telah diajarkan. Penulis juga berterima kasih kepada keluarga dan teman-teman yang telah memberikan dukungan, motivasi, dan bantuan kepada penulis selama pengerjaan makalah ini.


## REFERENSI


[1] Levitin, Anany. 2011. *Introduction to The Design and Analysis of Algorithms*. London: Pearson.

[2] Steven and Felix Halim. 2010. *Competitive Programming 1st Edition.* Singapore: NUS.

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

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

[5] A. Thanyan Al Sultan. "The airport gate assignment problem: scheduling algorithms and simulation approach," Okayama University. Japan, March 2012.

[6] http://ocw.mit.edu/courses/civil-and-environmental-engineering/1-204-computer-algorithms-in-systems-engineering-spring-2010/lecture-notes/MIT1_204S10_lec10.pdf terakhir diakses pada 11 Mei 2018.

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, 13 Mei 2018

Rabbi Fijar Mayoza - 13516081