

# Penerapan Algoritma *Greedy* Dalam Penjadwalan Pasien Untuk Memaksimalkan Jumlah Pasien

Bryan Bernigen - 13520034  
Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung  
E-mail (gmail): bryanbernigen05@gmail.com

**Abstract**—Penjadwalan merupakan hal yang biasa dalam dunia medis. Biasanya penjadwalan dilakukan dengan antrean berbasis kedatangan tercepat atau dengan perjanjian pada suatu waktu tertentu. Namun kedua cara tersebut tidak dapat memaksimalkan jumlah pasien. Oleh karena itu, pada saat menjelang waktu libur panjang, dibutuhkan algoritma penjadwalan lain untuk memaksimalkan jumlah pasien yang dapat dilayani sebelum libur. Algoritma *greedy* dapat menjadi solusi untuk permasalahan tersebut. Dengan algoritma *greedy*, penjadwalan pasien dapat dilakukan secara optimal. Data yang diperlukan juga tidak banyak yakni hanya perlu data waktu *available*, waktu *unavailable*, dan perkiraan lama waktu dilayani untuk setiap pasien. Dengan data tersebut dapat dilakukan penjadwalan dengan algoritma *greedy* untuk memperoleh jumlah pasien maksimal.

**Keywords**—*greedy*, penjadwalan, pasien, medis

## I. PENDAHULUAN

Sistem penjadwalan sudah digunakan pada dunia medis sejak lama. Sebagai contoh biasanya sebelum kita datang ke rumah sakit, kita akan menelepon bagian pendaftaran untuk menentukan jadwal terlebih dahulu. Atau jika tidak kita dapat langsung datang ke rumah sakit dan langsung menunggu di sana. Namun terdapat kesamaan pada kedua cara tersebut yakni sama-sama perlu menjadwalkan terlebih dahulu ke bagian pendaftaran sebelum akhirnya mendapatkan tindakan dari dokter yang bersangkutan.

Secara umum, biasanya terdapat dua cara penjadwalan yang umumnya dipakai di dunia medis yakni penjadwalan berdasarkan antrean dan penjadwalan berdasarkan perjanjian atau dengan kata lain penjadwalan berbasis *booking* sebuah blok waktu. Kesamaan dari kedua cara penjadwalan tersebut adalah kedua cara penjadwalan tersebut hanya memperhatikan urutan kedatangan atau urutan pengambilan sebuah blok waktu tanpa memperhatikan hal lainnya (Dengan catatan tidak ada sebuah kejadian dengan prioritas tinggi muncul seperti panggilan melahirkan/operasi fatal/dan lain-lain). Kedua cara penjadwalan tersebut mempraktikkan prinsip keadilan yakni siapa cepat dia dapat. Namun kedua cara tersebut dapat membuat jumlah pasien yang dapat ditangani oleh dokter menjadi tidak optimal. Sebagai contoh, untuk cara penjadwalan berbasis antrean, jika terdapat seseorang yang membutuhkan waktu pelayanan lama (3 jam) pada awal antrean, maka pasien

lain harus menunggu lama untuk mendapatkan pelayanan padahal mungkin saja waktu layanan para pasien yang menunggu hanya 5 menit saja. Contoh lain untuk cara penjadwalan berbasis *booking* sebuah blok waktu adalah jika terdapat 2 orang berbeda yang *membooking* waktu dengan jarak antara waktu selesai orang ke-1 dengan waktu mulai orang ke-2 sebesar 59 menit, maka 59 menit akan terbuang dengan percuma karena tidak ada orang lain yang bisa masuk (dengan asumsi 1 orang minimal 1 jam). Kedua hal tersebutlah yang membuat penjadwalan dengan cara konvensional tidak dapat menghasilkan jumlah pasien secara maksimal.

Pertanyaan yang sering ditanyakan adalah untuk apa memaksimalkan jumlah pasien ketika biaya yang seorang pasien keluarkan biasanya berbanding lurus dengan jumlah waktu yang mereka habiskan. Untuk apa mengerjakan banyak pasien dengan waktu sebentar namun biaya kecil? Bukankah lebih baik mengerjakan satu pasien untuk waktu lama namun dengan biaya yang besar juga? Jawabannya adalah optimasi jumlah pasien di sini bukan ditujukan untuk mendapatkan keuntungan sebesar mungkin, namun optimasi di sini ditujukan untuk melayani sebanyak mungkin pasien. Kepuasan pasien merupakan hal yang jauh lebih berharga dibandingkan keuntungan sesaat. Jika kita dapat memberikan pelayanan yang memuaskan, maka pasien tersebut akan menjadi langganan dan jumlah total biaya yang mereka keluarkan di kemudian hari akan jauh lebih besar dibandingkan keuntungan sesaat yang kita dapatkan jika kita lebih memilih satu kasus besar dengan waktu yang lama.

Untuk melakukan optimasi terhadap penjadwalan pasien, maka dapat diterapkan optimasi dengan algoritma *greedy* terhadap *activity selection* problem. Permasalahan yang diangkat pada algoritma optimasi tersebut cukup mirip dan cukup merepresentasikan permasalahan saat ini sehingga optimasi penjadwalan pasien untuk memaksimalkan jumlah pasien akan di optimasi menggunakan algoritma *greedy* pada persoalan *activity selection* problem.

Optimasi penjadwalan agar memaksimalkan jumlah pasien tidak dapat dilakukan secara rutin atau dengan kata lain menjadikan optimasi ini menjadi penjadwalan *default*. Hal tersebut tidak baik untuk dilakukan karena Penjadwalan tersebut kurang etis untuk dilakukan secara terus menerus karena berpotensi untuk menimbulkan *starvation* atau dengan kata lain berpotensi untuk membuat seorang pasien secara terus

menerus gagal mendapatkan jadwal karena jadwalnya selalu terpengkas ketika dilakukan optimasi. Optimasi tersebut hanya cocok untuk dilakukan ketika waktu yang tersedia terbatas. Contoh ketika dokter mau libur lebaran atau tutup untuk beberapa hari sehingga waktu dokter tersebut tersedia sangat terbatas yakni hanya sampai waktu libur tersebut. Pada saat inilah optimasi cocok untuk dilakukan. Pendaftaran dapat dibuka sampai H-1, lalu setiap pasien yang mendaftar akan diminta waktu *available*-nya dan dimasukkan ke *waiting list* dan pada hari H, optimasi dilakukan. Pasien yang terpilih akan di layani oleh dokter sedangkan pasien yang tidak terpilih akan dikabari. Dengan demikian dokter tersebut dapat melayani sebanyak mungkin orang sebelum libur sehingga jumlah orang yang merasa puas dengan dokter akan maksimal.

Catatan: pendahuluan tersebut ditulis berdasarkan pengalaman penulis menjadi bagian pendaftaran seorang dokter dan bisanya pada waktu dekat libur, dokter akan memilih-milih pasien berdasarkan insting agar dapat melayani sebanyak mungkin pasien sebelum libur.

## II. LANDASAN TEORI

### A. Algoritma Greedy

Algoritma *greedy* adalah algoritma yang memecahkan masalah dengan tamak yakni mengambil solusi terbaik pada setiap iterasi dengan harapan bahwa solusi optimum lokal tersebut dapat menghasilkan solusi optimum global juga. Secara umum, langkah-langkah algoritma *greedy* adalah sebagai berikut:

1. Pada setiap langkah, pilih solusi terbaik pada langkah tersebut tanpa memikirkan konsekuensi dari pilihan tersebut untuk langkah-langkah berikutnya.
2. Lalu pada setiap langkah, kita berharap bahwa dengan memilih optimal lokal pada setiap langkah, maka optimum global akan tercapai.

Prinsip yang digunakan pada algoritma *greedy* adalah “*take what you can take now!*”. Prinsip tersebut membuat algoritma *greedy* selalu mengambil solusi optimal lokal karena algoritma tersebut berprinsip untuk mengambil sebanyak-banyaknya ketika ada kesempatan. Prinsip tersebut juga yang membuat algoritma ini dinamakan algoritma *greedy* karena sifatnya yang rakus atau tamak. Prinsip lain yang ada pada algoritma *greedy* adalah prinsip bahwa sesuatu yang sudah terpilih tidak dapat dikembalikan sehingga algoritma tersebut akan tetap memilih sebuah pilihan yang optimal tanpa memedulikan efek yang diberikan oleh pilihan tersebut. Algoritma tersebut juga tidak akan membuang pilihan yang sudah dipilihnya walaupun pada iterasi-iterasi selanjutnya ditemukan bahwa pilihan tersebut memiliki efek negatif pada himpunan solusi.

Algoritma *greedy* cocok untuk digunakan pada persoalan-persoalan yang membutuhkan optimasi solusi yakni memaksimalkan ataupun meminimalkan sesuatu. Jika persoalan merupakan persoalan memaksimalkan, maka strategi algoritma *greedy* adalah memilih yang paling maksimum pada setiap iterasi. Sedangkan jika persoalan

merupakan persoalan meminimalkan, maka strategi algoritma *greedy* yang diterapkan adalah memilih yang paling minimum pada setiap iterasi.

Dalam menyelesaikan suatu masalah dengan algoritma *greedy*, diperlukan pemetaan masalah tersebut menjadi elemen-elemen algoritma *greedy*. Terdapat 6 elemen dalam algoritma *greedy* yang perlu dipetakan dari sebuah persoalan yakni himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi objektif. Berikut merupakan penjelasan dari masing-masing elemen tersebut.

1. Himpunan kandidat (C): berisi langkah-langkah yang mungkin untuk dipilih pada setiap iterasi.
2. Himpunan solusi (S): berisi langkah-langkah yang terpilih dari himpunan kandidat.
3. Fungsi solusi: fungsi untuk menentukan apakah langkah yang sudah terpilih dan dimasukkan ke himpunan solusi sudah memberikan hasil yang diinginkan.
4. Fungsi seleksi: fungsi yang menentukan langkah mana yang dipilih berdasarkan strategi *greedy* yakni memaksimalkan atau meminimalkan.
5. Fungsi kelayakan: Memeriksa apakah langkah yang terpilih oleh fungsi seleksi tidak melanggar *constraint* apa pun dan bisa dimasukkan ke himpunan solusi.
6. Fungsi objektif: memaksimalkan atau meminimalkan langkah yang dipilih.

Berikut merupakan skema umum algoritma *greedy* dalam *pseudocode*:

Skema umum algoritma *greedy*:

```
function greedy(C : himpunan_kandidat) → himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }
Deklarasi
x : kandidat
S : himpunan_solusi

Algoritma:
S ← {} { inisialisasi S dengan kosong }
while (not SOLUSI(S) and (C ≠ {})) do
x ← SELEKSI(C) { pilih sebuah kandidat dari C }
C ← C - {x} { buang x dari C karena sudah dipilih }
if LAYAK(S ∪ {x}) then { x memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi }
S ← S ∪ {x} { masukkan x ke dalam himpunan solusi }
endif
endwhile
{ SOLUSI(S) or C = {} }

if SOLUSI(S) then { solusi sudah lengkap }
return S
else
write("tidak ada solusi")
endif
```

Gambar 2.1 Skema Umum Algoritma Greedy dalam Pseudocode

Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)

Algoritma *greedy* berusaha untuk mencari optimum lokal pada setiap iterasi untuk mencapai optimum global. Namun algoritma *greedy* tidak dapat memastikan bahwa solusi yang didapat merupakan optimum global. Solusi yang diperoleh dapat berupa pseudo-optimum atau sub-optimum yakni optimum semu. Berikut merupakan alasan algoritma *greedy* tidak selalu menghasilkan solusi yang optimum.

1. Algoritma *greedy* tidak melakukan iterasi terhadap semua kemungkinan seperti yang dilakukan oleh algoritma *brute force*.
2. Algoritma *greedy* menggunakan fungsi seleksi yang berbeda-beda untuk setiap persoalan yang berbeda. Oleh karena itu perlu untuk menemukan fungsi seleksi yang tepat agar solusi persoalan tersebut menjadi optimal.

Karena kedua alasan tersebut, tidak semua solusi algoritma *greedy* dapat menghasilkan solusi yang optimal. Namun bukan berarti algoritma *greedy* tidak dapat menghasilkan solusi yang optimal. Untuk beberapa persoalan, hasil dari solusi algoritma *greedy* dapat menghasilkan solusi yang optimal. Berikut beberapa contoh persoalan yang dapat diselesaikan menggunakan algoritma *greedy* beserta keoptimalan solusinya.

Tidak selalu Optimal:

1. Persoalan penukaran uang
2. Persoalan *knapsack* 1/0
3. Persoalan lintasan terpendek
4. Persoalan pecahan mesir
5. Persoalan TSP

Selalu Optimal:

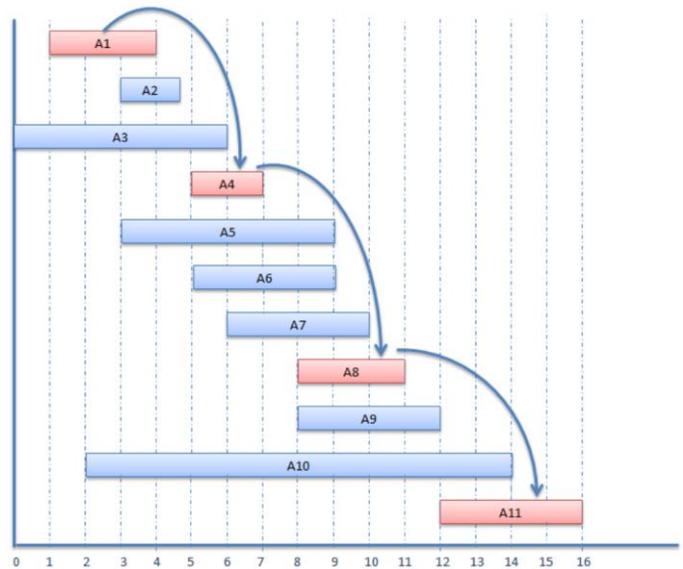
1. Persoalan *activity selection*
2. Persoalan meminimalkan waktu di sistem
3. Persoalan *knapsack* pecahan
4. Persoalan *job* dengan tenggat waktu
5. Persoalan pohon merentang minimum
6. Persoalan kode Huffman

Walaupun algoritma *greedy* tidak selalu menghasilkan solusi optimal, namun ada beberapa kondisi ketika algoritma ini baik untuk digunakan. Kondisi pertama tentunya ketika kita tahu bahwa solusi *greedy* tersebut merupakan solusi optimal. Kondisi kedua adalah ketika hasil eksak tidak diperlukan. Pada kondisi kedua, solusi dari algoritma *greedy* dapat dijadikan patokan atau hampiran dari solusi optimal.

### B. Persoalan Memilih Aktivitas

Persoalan memilih aktivitas adalah persoalan ketika kita diberikan beberapa pilihan aktivitas pada waktu yang mungkin sama maupun beda dan kita ingin menghadiri sebanyak mungkin aktivitas. Aktivitas yang mungkin untuk dihadiri adalah aktivitas yang waktunya tidak *overlap* dengan aktivitas yang kita pilih. Setiap aktivitas memiliki waktu mulai, durasi dan waktu berakhir. Dengan data tersebut, kita harus dapat menjadwalkan sebanyak mungkin aktivitas yang dapat kita hadiri tanpa menabrak satu sama lain (Dengan anggapan waktu perpindahan tempat dari suatu aktivitas ke aktivitas lainnya adalah nol). Berikut merupakan contoh dari persoalan memilih aktivitas:

Start time (si)	finish time (fi)	Activity name
1	4	A1
3	5	A2
0	6	A3
5	7	A4
3	9	A5
5	9	A6
6	10	A7
8	11	A8
8	12	A9
2	14	A10
12	16	A11



Gambar 2.2: Contoh Persoalan Memilih Aktivitas

Sumber: [https://scanfree.com/Data\\_Structure/activity-selection-problem](https://scanfree.com/Data_Structure/activity-selection-problem)

Pada contoh tersebut, untuk memaksimalkan jumlah aktivitas yang dipilih, maka kita harus memilih aktivitas 1,4,8, dan 11. Untuk menyelesaikan persoalan tersebut, dapat diterapkan beberapa strategi seperti strategi *brute force* dan strategi *greedy*. Jika menggunakan *brute force*, maka cara pencarian solusi adalah dengan mengiterasi semua kemungkinan dan mencari kemungkinan dengan jumlah aktivitas terbanyak. Jumlah kemungkinan yang adalah  $2^n$  dengan  $n$  banyaknya aktivitas. Sehingga untuk mencari solusi pada persoalan sebelumnya dibutuhkan iterasi sebanyak  $2^{11}$  kemungkinan yakni 2048 kemungkinan. Strategi tersebut memang memastikan solusi optimal, namun kurang cocok untuk dipakai pada persoalan dengan jumlah aktivitas yang banyak.

Strategi lain yang dapat digunakan adalah strategi *greedy*. Strategi tersebut dapat diterapkan dengan cara mengurutkan aktivitas berdasarkan waktu selesai terkecil lalu melakukan iterasi dan pada setiap iterasi, pilihlah aktivitas yang waktu mulainya lebih besar atau sama dengan waktu selesai aktivitas yang dipilih sebelumnya. Berikut merupakan *pseudocode* strategi tersebut

```

function Greedy-Activity-Selector( $s_1, s_2, \dots, s_n$  : integer,  $f_1, f_2, \dots, f_n$  : integer) → set of integer
{ Asumsi: aktivitas sudah diurut terlebih dahulu berdasarkan waktu selesai:  $f_1 \leq f_2 \leq \dots \leq f_n$  }
Deklarasi
   $i, j, n$  : integer
  A : set of integer
Algoritma:
   $n \leftarrow \text{length}(s)$ 
   $A \leftarrow \{1\}$  { aktivitas nomor 1 selalu terpilih }
   $j \leftarrow 1$ 
  for  $i \leftarrow 2$  to  $n$  do
    if  $s_i \geq f_j$  then
       $A \leftarrow A \cup \{i\}$ 
       $j \leftarrow i$ 
    endif
  endif

```

Gambar 2.3 Notasi Algoritma Persoalan pemilihan Aktivitas

Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)

Dengan menerapkan strategi *greedy* tersebut, maka dapat dijamin bahwa solusi yang diperoleh optimal. Bukan hanya itu saja, jumlah iterasi yang dilakukan hanyalah  $n-1$  dengan  $n$  jumlah aktivitas. Hal tersebut jauh lebih baik dibandingkan dengan strategi *greedy* yang ordenya adalah eksponensial.

### C. Penjadwalan di Dunia Medis

Penjadwalan adalah proses perencanaan untuk menentukan jadwal suatu hal diproses atau suatu tugas dikerjakan. Penjadwalan merupakan suatu hal yang sangat penting dalam suatu bisnis produksi barang maupun jasa. Dalam bisnis produksi barang, penjadwalan berkaitan dengan alokasi *resources* yang terbatas untuk mengerjakan sebanyak-banyaknya barang. Dalam bisnis jasa, penjadwalan berkaitan dengan alokasi waktu karena dalam bisnis jasa, yang dijual adalah waktu dari pemberi jasa.

Dalam dunia medis, penjadwalan juga sangat penting karena dunia medis biasanya berkaitan dengan bisnis jasa sehingga alokasi waktu sangat diperlukan agar hasil yang diperoleh lebih maksimum. Pada dunia medis biasanya terdapat dua penjadwalan yang umum diterapkan yakni penjadwalan berdasarkan waktu kedatangan dan penjadwalan berdasarkan perjanjian. Penjadwalan berdasarkan waktu kedatangan biasanya diterapkan pada pasien yang baru datang untuk pertama kali atau pada dokter yang tidak memiliki bagian pendaftaran sehingga penjadwalan dilakukan secara *simple* yakni berdasarkan urutan kedatangan. Penjadwalan berdasarkan perjanjian biasanya diterapkan kepada pasien langganan atau diterapkan oleh dokter yang memiliki bagian resepsionis atau bagian pendaftaran sehingga penjadwalan tidak dilakukan oleh dokter tersebut melainkan dilakukan oleh bagian pendaftaran.

Kedua tipe penjadwalan tersebut memiliki kelebihan dan kekurangannya masing-masing serta kecocokan untuknya masing-masing. Penjadwalan berdasarkan urutan kedatangan memiliki kelebihan pada bagian *simple* dan tidak perlu menambah biaya dengan mempekerjakan bagian pendaftaran. Namun kekurangan dari penjadwalan tipe tersebut adalah penjadwalan tersebut berpotensi membuat orang malas untuk datang karena malas menunggu antrian apa lagi jika dokter tersebut sudah terkenal sehingga antreannya selalu panjang. Penjadwalan tipe tersebut cocok untuk digunakan oleh dokter yang masih baru membuka praktik dan belum memiliki langganan. Tujuan menerapkan penjadwalan tipe ini adalah

untuk mencari langganan. Penjadwalan tipe perjanjian memiliki kelebihan yakni pasien tidak perlu menunggu lama. Pasien cukup datang tepat waktu sesuai waktu perjanjian dan pasien bisa langsung dilayani. Namun kelemahan dari penjadwalan tipe ini adalah biaya karena harus mempekerjakan bagian penjadwalan dan jika dokter tersebut belum memiliki langganan, maka penjadwalan tipe ini cukup percuma karena ujung-ujungnya tidak akan ada orang yang menjadwal. Tipe penjadwalan ini cocok untuk digunakan oleh dokter yang sudah memiliki langganan dan sudah cukup dikenal. Tujuan dari penerapan penjadwalan ini adalah untuk mengurangi waktu tunggu sehingga orang tidak malas untuk datang kembali.

### III. OPTIMALISASI PENJADWALAN PASIEN DENGAN ALGORITMA GREEDY

Untuk melakukan optimasi pasien, maka perlu sedikit penyesuaian terhadap metode penerimaan pasien. Penyesuaian yang harus dilakukan adalah seluruh pasien yang mendaftar pada hari tersebut tidak akan langsung diberi kepastian apakah dapat dikerjakan atau tidak. Lalu pasien tidak langsung memilih sebuah *slot* waktu melainkan pasien akan diminta waktu *available* pada hari tersebut beserta keluhannya. Dengan keluhan tersebut kita dapat memprediksi perkiraan lama pengerjaan. Setelah itu kita akan menunggu sampai waktu tertentu sebelum hari H di mana kita akan melakukan optimasi terhadap data yang dimiliki pada saat tersebut. Pasien yang terpilih akan dilayani sedangkan yang tidak terpilih akan dikabari untuk jadwal ulang lain kali.

Persoalan penjadwalan pasien untuk mendapatkan jumlah pasien terbanyak mirip dengan persoalan *activity selection* problem. Perbedaannya terletak pada waktu yang *fixed* untuk *activity selection* problem seperti waktu mulai 7, durasi 2, maka waktu akhir 9. Sedangkan untuk penjadwalan pasien, waktu tersebut lebih tentatif seperti waktu *available* 7 sampai dengan 11 dengan perkiraan lama pengerjaan 2 jam, maka pasien tersebut dapat dikerjakan pada waktu 7-9. Oleh karena itu, perlu ada penyesuaian pada algoritma *activity selection problems* untuk memperoleh hasil yang diinginkan

#### A. Penyesuaian Algoritma Greedy

Penyesuaian pertama yang dilakukan adalah menambah kolom *perkiraan\_selesai* yang merupakan hasil dari kolom *available* + kolom *perkiraan lama pengerjaan*. Kolom tersebut berpadanan dengan kolom aktivitas selesai pada *activity selection problem*. Namun berbeda dengan persoalan memilih aktivitas, pada persoalan penjadwalan, data tidak perlu dan tidak boleh diurut berdasarkan *perkiraan\_selesai* karena kolom *perkiraan\_selesai* akan terus menerus *diupdate* selama pencarian solusi sehingga tidak perlu melakukan pengurutan berdasarkan kolom tersebut.

Penyesuaian kedua yang dilakukan pemilihan pasien tidak dilakukan secara sekuensial seperti pada persoalan pemilihan aktivitas karena berbeda dengan persoalan pemilihan aktivitas, data pada persoalan penjadwalan tidak terurut. Pemilihan pasien dilakukan dengan melakukan iterasi terhadap seluruh data dengan mencari nilai terkecil pada kolom *perkiraan\_selesai*. Pencarian tersebut dilakukan bersama-sama dengan pembaharuan kolom *perkiraan\_selesai*. Kolom

perkiraan selesai akan diperbaharui jika kolom *available* suatu data < kolom *perkiraan* selesai data yang dimasukkan ke himpunan solusi (). Berikut adalah contoh pencarian beserta *update* yang dimaksud.

1. Solusi: []

Nama	Available	Unavailable	Lama	Perkiraan
A	7	9	2	9
B	8	11	1	9
C	8	11	3	11
D	10	11	1	11

Karena A (dan B) memiliki perkiraan terkecil, maka A dimasukkan ke dalam solusi.

2. Solusi: [A], 9

Nama	Available	Unavailable	Lama	Perkiraan
B	8	11	1	9
C	8	11	3	11
D	10	11	1	11

Perhatikan bahwa waktu *available* B (8) < waktu perkiraan solusi (9). Dalam *activity selection* problem, B akan dicoret, namun jika kita perhatikan, B memiliki rentang waktu *available* sebesar 3 sehingga jika B baru dikerjakan pada waktu 9, B masih layak dijadikan solusi. Oleh karena itu, setiap ada penambahan solusi, lakukan pencarian perkiraan terkecil sambil melakukan *update* seperti contoh di bawah ini.

3. Solusi: [A], 9

Nama	Available	Unavailable	Lama	Perkiraan
B	9	11	1	10
C	9	11	3	11
D	10	11	1	11

Dapat dilihat pada tabel tersebut bahwa dilakukan *update* pada B dan C. Dapat dilihat juga bahwa perkiraan C > *unavailable* C sehingga data C dihapus dari data. Karena B merupakan data dengan kolom perkiraan terkecil, maka B akan dimasukkan ke solusi.

4. Solusi: [A,B], 10

Nama	Available	Unavailable	Lama	Perkiraan
D	10	11	1	11

Karena D merupakan data dengan kolom perkiraan terkecil, maka D dimasukkan ke solusi.

5. Solusi: [A,B,D], 11

Solusi yang didapat untuk contoh tersebut adalah 3 pasien yakni A, B, dan D dengan waktu perkiraan selesai 11.

### B. Pseudocodee Algoritma Penjadwalan

Karena algoritma penyelesaian tersebut berubah, maka perlu ada penyesuaian algoritma terhadap algoritma *activity selection problem* pada dasar teori. Berikut merupakan perbaikan algoritma dalam notasi *pseudocode*:

**Struct** pasien:

Nama : string

Available : integer

Unavailable : integer

Lama : integer

Perkiraan : integer

**Function** penjadwalan(pasiens : list of struct pasien) → list of string

**Deklarasi**

i,n : integer

min\_idx : integer

max\_selesai : integer

result : list of string

p : struct pasien

removed : boolean

**Algoritma:**

max\_selesai ← 0

while (not empty(pasiens)) do

n ← lenght(pasiens)

for i ← 1 to n do

p ← pasien[i]

min\_idx ← 0

if( p.available < max\_selesai) then

p.available ← max\_selesai

p.perkiraan ← p.available + p.lama

if(p.perkiraan > p.unavailable) then

remove(p)

else

if(pasiens[min\_idx].prediksi>p.prediksi) then

min\_idx ← i

max\_selesai ← pasien[min\_idx].prediksi

result ← result + pasien[min\_idx].nama

remove(pasiens[min\_idx])

Performa algoritma perbaikan tersebut memang tidak sebaik performa algoritma *greedy* pada *activity selection problem*. Jika dilihat, kompleksitas algoritma tersebut adalah  $O(n^2)$  karena mengandung *for loop* di dalam *while loop*. Hal tersebut lebih buruk jika dibandingkan dengan *activity selection problem* yang memiliki kompleksitas  $O(N)$  saja. Namun jika dilihat lagi dengan lebih seksama, maka sebenarnya kompleksitas kedua algoritma tidak berbeda jauh karena algoritma *activity selection problem* memiliki kompleksitas  $O(N)$  karena waktu pengurutan data tidak dihitung. Jika waktu pengurutan data dihitung, maka kompleksitasnya akan bertambah. Berbeda dengan algoritma penjadwalan yang tidak membutuhkan pengurutan sehingga kompleksitasnya tetap  $O(N^2)$ . Walaupun algoritma penjadwalan memiliki kompleksitas yang lebih buruk, namun kelebihan dari algoritma tersebut adalah tidak terpatok oleh waktu sehingga untuk persoalan yang mirip dengan *activity selection problem*, namun memiliki waktu yang tidak strict, maka algoritma penjadwalan ini dapat digunakan.

### C. Pemetaan Persoalan

Setelah memperbaiki algoritma, maka langkah selanjutnya adalah memperbaiki pemetaan persoalan menjadi elemen *greedy*. Berikut merupakan perbaikan dari pemetaan persoalan menjadi elemen-elemen *greedy*.

1. Himpunan kandidat (C): pasien-pasien yang mendaftar pada hari optimasi tersebut
2. Himpunan solusi (S): pasien-pasien yang terpilih untuk dilayani pada hari tersebut
3. Fungsi solusi: Mengecek apakah jumlah pasien tersebut sudah maksimal.
4. Fungsi seleksi: Memilih pasien yang waktu prediksi\_selesai-nya paling kecil
5. Fungsi kelayakan: Periksa apakah dengan menambah pasien tersebut membuat ada waktu yang *overlapping*.
6. Fungsi objektif: memaksimalkan jumlah pasien.

### D. Penerapan Menggunakan Bahasa Python

Berikut merupakan contoh persoalan penjadwalan 20 orang dengan waktu *available* antara 7 – 12 (orang yang bangunnya pagi dan bangunnya siang) dan waktu *unavailable* di antara 18 – 24 (orang yang tidurnya cepat dan yang tidurnya lama). Waktu pengerjaan adalah di antara 2-6 jam. Berikut merupakan contoh data yang digunakan.

Nama	Available	Unavailable	Lama	Prediksi
Pasien1	8	19	2	10
Pasien2	9	18	5	14
Pasien3	11	23	4	15
Pasien4	9	19	4	13
Pasien5	11	20	3	14
Pasien6	12	20	2	14

Pasien7	11	24	3	14
Pasien8	9	22	2	11
Pasien9	12	23	5	17
Pasien10	11	19	4	15
Pasien11	10	18	5	15
Pasien12	12	23	2	14
Pasien13	10	21	5	15
Pasien14	11	19	3	14
Pasien15	9	23	2	11
Pasien16	7	20	4	11
Pasien17	7	18	4	11
Pasien18	9	20	2	11
Pasien19	11	20	5	16
Pasien20	8	22	3	11

Jika seseorang harus memikirkan pendaftaran sekian banyak orang secara manual agar dapat melayani paling banyak pasien, maka kemungkinan orang tersebut akan pusing karena jumlah data yang cukup banyak. Jika pendaftaran dilakukan secara *bruteforce*, maka akan ada  $2^{20}$  kemungkinan yakni 1048576 kemungkinan. Jika 1 kemungkinan di cek dalam 1 detik, maka akan dibutuhkan setidaknya 12 hari untuk mengecek semua kemungkinan tersebut. Oleh karena itu, dibutuhkan algoritma *greedy* untuk menyelesaikan permasalahan tersebut. Berikut hasil dari penerapan algoritma *greedy* tersebut.

```
python -u "c:\Users\Igen\Downloads\test.py"
pasien1 8 19 2 10
pasien2 9 18 5 14
pasien3 11 23 4 15
pasien4 9 19 4 13
pasien5 11 20 3 14
pasien6 12 20 2 14
pasien7 11 24 3 14
pasien8 9 22 2 11
pasien9 12 23 5 17
pasien10 11 19 4 15
pasien11 10 18 5 15
pasien12 12 23 2 14
pasien13 10 21 5 15
pasien14 11 19 3 14
pasien15 9 23 2 11
pasien16 7 20 4 11
pasien17 7 18 4 11
pasien18 9 20 2 11
pasien19 11 20 5 16
pasien20 8 22 3 11

pasien1 dikerjakan pada jam 8 selesai pada jam 10
pasien8 dikerjakan pada jam 10 selesai pada jam 12
pasien6 dikerjakan pada jam 12 selesai pada jam 14
pasien12 dikerjakan pada jam 14 selesai pada jam 16
pasien15 dikerjakan pada jam 16 selesai pada jam 18
pasien18 dikerjakan pada jam 18 selesai pada jam 20
pasien7 dikerjakan pada jam 20 selesai pada jam 23
```

Gambar 3.1 Hasil Implementasi Menggunakan Bahasa Python

