



# ANALISIS ALGORITMA

Ir. Rismayani, S.Kom., M.T

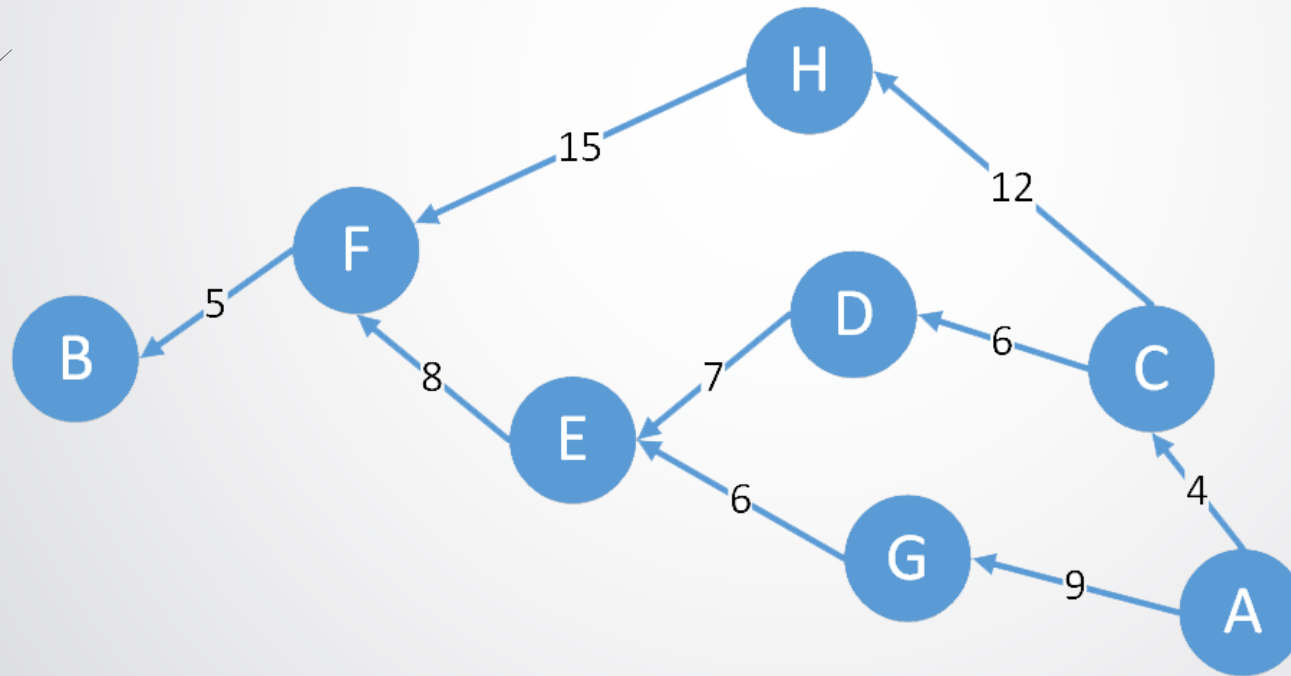


# Algoritma Greedy

- Algoritma greedy merupakan jenis algoritma yang menggunakan pendekatan penyelesaian masalah dengan mencari nilai maksimum sementara pada setiap langkahnya.
- Nilai maksimum sementara ini dikenal dengan istilah *local maximum*.
- Pada kebanyakan kasus, algoritma greedy tidak akan menghasilkan solusi paling optimal, begitupun algoritma greedy biasanya memberikan solusi yang mendekati nilai optimum dalam waktu yang cukup cepat.

# CONTOH

- Misalkan kita ingin bergerak dari titik A ke titik B, dan kita telah menemukan beberapa jalur dari peta:





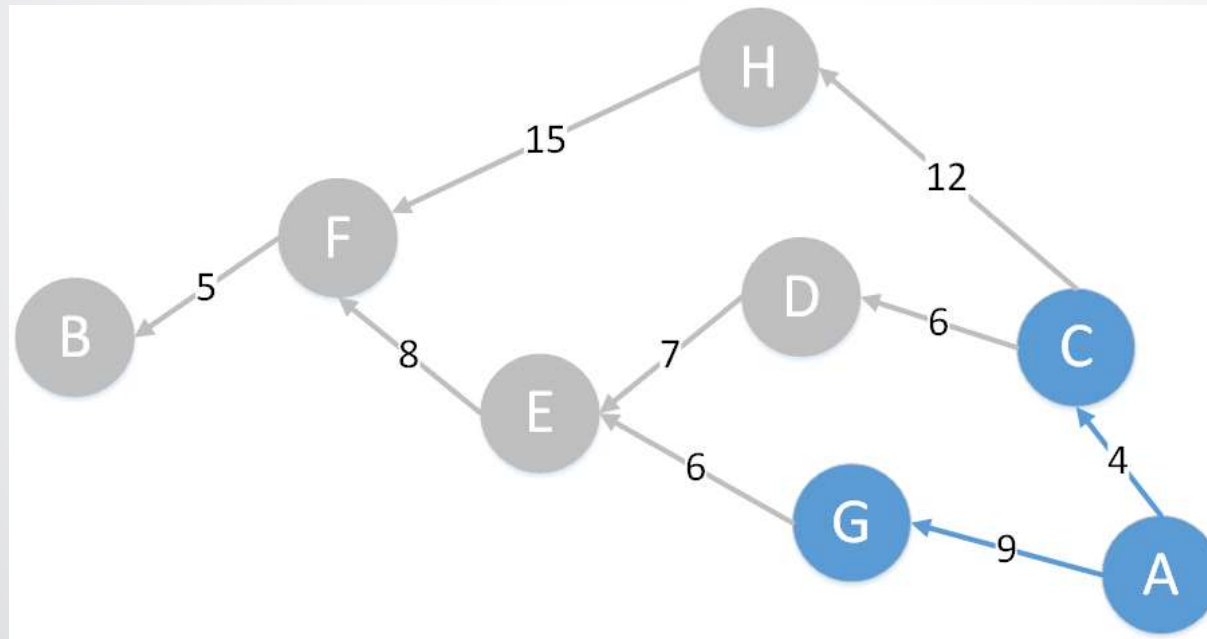
# LANJUTAN



- Langkah pertama yang harus kita lakukan tentunya adalah memilih struktur data yang tepat untuk digunakan dalam merepresentasikan peta.
- Untuk mencari jarak terpendek dari A ke B, sebuah algoritma greedy akan menjalankan langkah-langkah seperti berikut:
  - Kunjungi satu titik pada graph, dan ambil seluruh titik yang dapat dikunjungi dari titik sekarang.
  - Cari *local maximum* ke titik selanjutnya.
  - Tandai graph sekarang sebagai graph yang telah dikunjungi, dan pindah ke *local maximum* yang telah ditentukan.
  - Kembali ke langkah 1 sampai titik tujuan didapatkan.

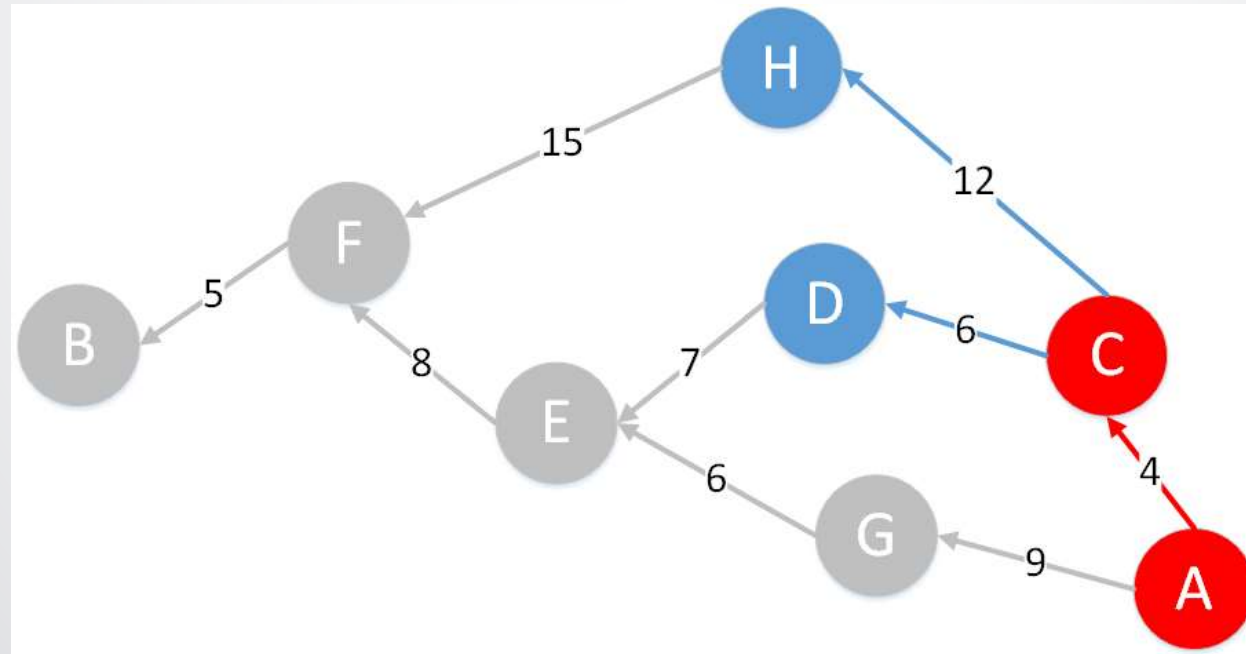
# LANJUTAN

- ▶ Jika mengaplikasikan langkah-langkah di atas pada graph A ke B sebelumnya maka kita akan mendapatkan pergerakan seperti berikut:
  - ▶ Mulai dari titik awal (A). Ambil seluruh titik yang dapat dikunjungi.



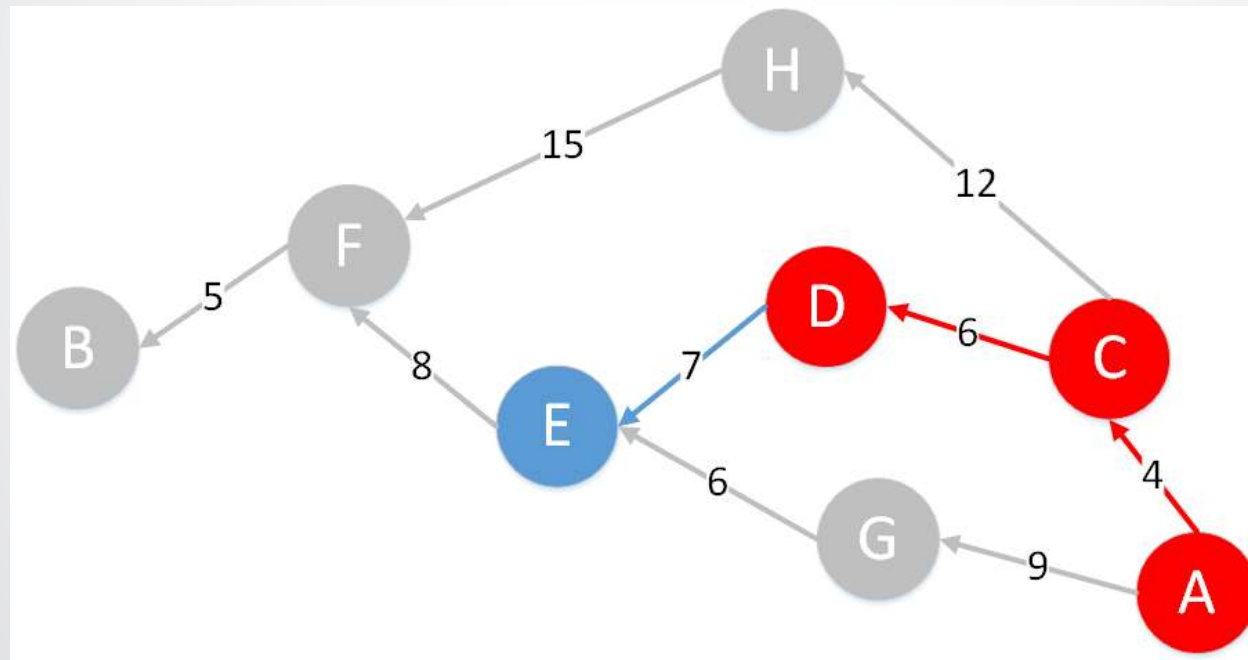
# LANJUTAN

- ▶ Local maximum adalah ke C, karena jarak ke C adalah yang paling dekat.
- ▶ Tandai A sebagai titik yang telah dikunjungi, dan pindah ke C.
- ▶ Ambil seluruh titik yang dapat dikunjungi dari C.



# LANJUTAN

- ▶ Local maximum adaah ke D, dengan jarak 6.
- ▶ Tandai C sebagai titik yang telah dikunjungi, dan pindah ke D







# LANJUTAN

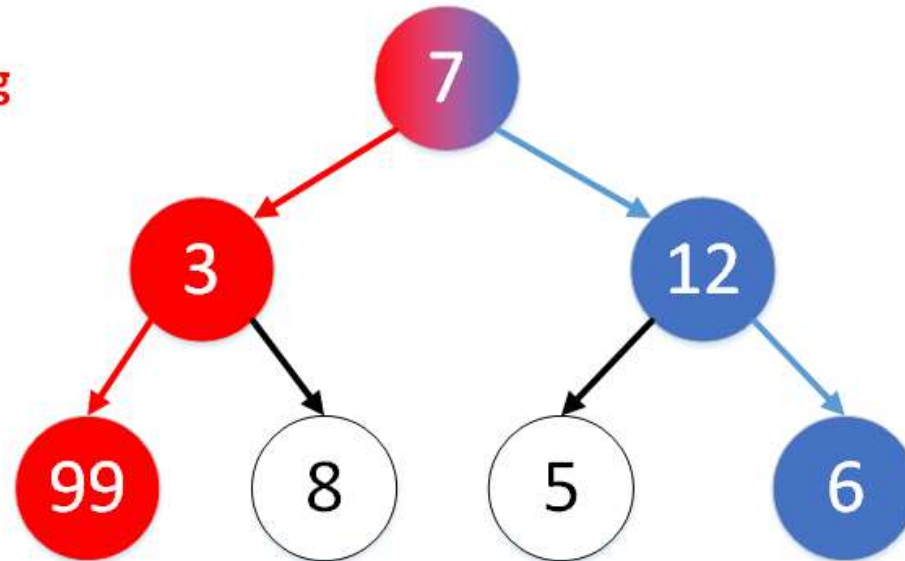
- ▶ Dengan menggunakan algoritma greedy pada graph di atas, hasil akhir yang akan didapatkan sebagai jarak terpendek adalah A-C-D-E-F-B.
- ▶ Hasil jarak terpendek yang didapatkan ini tidak tepat dengan jarak terpendek yang sebenarnya (A-G-E-F-B).
- ▶ Algoritma greedy memang tidak selamanya memberikan solusi yang optimal, dikarenakan pencarian *local maximum* pada setiap langkahnya, tanpa memperhatikan solusi secara keseluruhan.



# CONTOH

## Pencarian Nilai Terbesar

Langkah yang benar



Langkah dengan Algoritma Greedy



# LANJUTAN

- ▶ Tetapi ingat bahwa untuk kasus umum, kerap kali algoritma greedy memberikan hasil yang *cukup baik* dengan kompleksitas waktu yang cepat.
- ▶ Hal ini mengakibatkan algoritma greedy sering digunakan untuk menyelesaikan permasalahan kompleks yang memerlukan kecepatan jawaban, bukan solusi optimal, misalnya pada game.

# Implementasi Algoritma Greedy

- Pada implementasi yang kita lakukan, graph direpresentasikan dengan menggunakan dictionary di dalam dictionary, seperti berikut:

```
DAG = {'A': {'C': 4, 'G': 9},  
      'G': {'E': 6},  
      'C': {'D': 6, 'H': 12},  
      'D': {'E': 7},  
      'H': {'F': 15},  
      'E': {'F': 8},  
      'F': {'B': 5}}
```

*# Hasil Representasi:*

```
{'A': {'C': 4, 'G': 9},  
 'C': {'D': 6, 'H': 12},  
 'D': {'E': 7},  
 'E': {'F': 8},  
 'F': {'B': 5},  
 'G': {'E': 6},  
 'H': {'F': 15}}
```

# LANJUTAN

- Selanjutnya kita akan membuat fungsi yang mencari jarak terpendek dari graph yang dibangun, dengan menggunakan algoritma greedy.
- Definisi dari fungsi tersebut sangat sederhana, hanya sebuah fungsi yang mengambil graph, titik awal, dan titik akhir sebagai argumennya.

```
def shortest_path(graph, source, dest):
```

- Jarak terpendek yang didapatkan akan dibangun langkah demi langkah, seperti pada algoritma greedy yang mengambil nilai local maximum pada setiap langkahnya.
- Untuk hal ini, tentunya kita akan perlu menyimpan jarak terpendek ke dalam sebuah variabel, dengan "Source" sebagai isi awal variabel tersebut.
- Jarak terpendek kita simpan ke dalam sebuah list, untuk menyederhanakan proses penambahan nilai.

# LANJUTAN

```
result = []  
result.append(source)
```

- Penelusuran graph sendiri akan kita lakukan melalui "result" karena variabel ini merepresentasikan seluruh node yang telah kita kunjungi dari keseluruhan graph.
- Variabel result pada dasarnya merupakan hasil implementasi dari langkah 3 algoritma ("Tandai graph sekarang sebagai graph yang telah dikunjungi").
- Titik awal dari rute tentunya secara otomatis ditandai sebagai node yang telah dikunjungi.
- Selanjutnya, kita akan menelusuri graph sampai titik tujuan ditemukan, dengan menggunakan iterasi:

# LANJUTAN

```
while dest not in result:  
    current_node = result[-1]
```

- dengan mengambil node yang sekarang sedang dicari *local maximum*-nya dari isi terakhir “result” Pencarian *local maximum* sendiri lebih memerlukan pengetahuan python daripada algoritma:

```
# Cari local maximum  
local_max = min(graph[current_node].values())  
  
# Ambil node dari local maximum,  
# dan tambahkan ke result  
# agar iterasi selanjutnya dimulai  
# dari node sekarang.  
for node, weight in graph[current_node].items():  
    if weight == local_max:  
        result.append(node)
```



# LANJUTAN

- setelah seluruh graph ditelusuri sampai mendapatkan hasil, kita dapat mengembalikan "result" ke pemanggil fungsi.

```
return result
```

- Keseluruhan fungsi yang dibangun adalah sebagai berikut:

```
def shortest_path(graph, source, dest):  
    result = []  
    result.append(source)  
  
    while dest not in result:  
        current_node = result[-1]  
  
        local_max = min(graph[current_node].values())  
        for node, weight in graph[current_node].items():  
            if weight == local_max:  
                result.append(node)  
  
    return result
```





# LANJUTAN

- ▶ Perlu diingat bahwa fungsi ini masih banyak memiliki kekurangan, misalnya tidak adanya penanganan kasus jika titik tujuan tidak ditemukan, atau jika terdapat node yang memiliki nilai negatif (bergerak balik).
- ▶ Penanganan hal-hal tersebut tidak dibuat karena fungsi hanya bertujuan untuk mengilustrasikan cara kerja algoritma greedy, bukan untuk digunakan pada aplikasi nyata.



# KESIMPULAN

- ▶ Algoritma greedy merupakan algoritma yang bersifat heuristik, mencari nilai maksimal sementara dengan harapan akan mendapatkan solusi yang cukup baik.
- ▶ Meskipun tidak selalu mendapatkan solusi terbaik (optimum), algoritma greedy umumnya memiliki kompleksitas waktu yang cukup baik, sehingga algoritma ini sering digunakan untuk kasus yang memerlukan solusi cepat meskipun tidak optimal seperti sistem real-time atau game.



# LANJUTAN



- Dari implementasi yang kita lakukan, dapat dilihat bagaimana algoritma greedy memiliki beberapa fungsionalitas dasar, yaitu:
  - Fungsi untuk melakukan penelusuran masalah.
  - Fungsi untuk memilih *local maximum* dari pilihan-pilihan yang ada tiap langkahnya.
  - Fungsi untuk mengisi nilai *local maximum* ke solusi keseluruhan.
  - Fungsi yang menentukan apakah solusi telah didapatkan.
- Tentunya fungsi-fungsi di atas juga dapat digabungkan atau dipecah lebih lanjut lagi, menyesuaikan dengan strategi greedy yang dikembangkan.

# TAMBAHAN

## Pengertian Algoritma Greedy

Pengertian Algoritma Greedy adalah jenis algoritma yang membentuk solusi langkah per langkah dengan mencari nilai maksimum sementara pada setiap langkahnya. Nilai maksimum sementara ini dikenal dengan istilah *local maximum*. Pada kebanyakan kasus, algoritma greedy tidak akan menghasilkan solusi paling optimal, begitupun algoritma greedy biasanya memberikan solusi yang mendekati nilai optimum dalam waktu yang cukup cepat.

## Prinsip Utama Algoritma Greedy

Prinsip utama algoritma greedy adalah “take what you can get now!”. Maksud dari prinsip tersebut adalah sebagai berikut: Pada setiap langkah dalam algoritma greedy, kita ambil keputusan yang paling optimal untuk langkah tersebut tanpa memperhatikan konsekuensi pada langkah selanjutnya. Sebagai contoh, jika kita menggunakan algoritma Greedy untuk menempatkan komponen diatas papan sirkuit, sekali komponen telah diletakkan dan dipasang maka tidak dapat dipindahkan lagi. Kita namakan solusi tersebut dengan optimum lokal. Kemudian saat pengambilan nilai optimum lokal pada setiap langkah, diharapkan tercapai optimum global, yaitu tercapainya solusi optimum yang melibatkan keseluruhan langkah dari awal sampai akhir.





# LANJUTAN



## Skema umum Algoritma Greedy

Algoritma greedy disusun oleh elemen, dan elemen-elemen yang digunakan dalam penerapan algoritma greedy antara lain :

### 1. Himpunan Kandidat

Himpunan yang berisi elemen pembentuk solusi.

### 2. Himpunan Solusi

Himpunan yang terpilih sebagai solusi persoalan.

### 3. Fungsi Seleksi

Fungsi yang memilih kandidat yang paling mungkin untuk mencapai solusi optimal.

### 4. Fungsi Kelayakan

Fungsi yang memeriksa apakah suatu kandidat yang dipilih dapat memberikan solusi yang layak. Maksudnya yaitu apakah kandidat tersebut bersama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala yang ada.



# LANJUTAN

## 5. Fungsi Solusi

Fungsi yang mengembalikan nilai boolean. True jika himpunan solusi yang sudah terbentuk merupakan solusi yang lengkap; False jika himpunan solusi belum lengkap.

## 6. Fungsi Objektif

Fungsi yang mengoptimalkan solusi.

Di dalam mencari sebuah solusi (optimasi) algoritma greedy hanya memakai 2 buah macam persoalan Optimasi, yaitu:

- *Maksimasi (maximization)*
- *Minimasi (minimization)*

# LANJUTAN

- ▶ Algoritma greedy membentuk solusi langkah per langkah (step by step).
  - ▶ Terdapat banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi.
  - ▶ Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan.
  - ▶ Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya.
  - ▶ **Persoalan optimasi** (optimization problems): persoalan yang menuntut pencarian solusi optimum.
  - ▶ Persoalan optimasi ada dua macam: Maksimasi (maximization) dan Minimasi (minimization)
- Solusi optimum (terbaik) adalah solusi yang bernilai minimum atau maksimum dari sekumpulan alternatif solusi yang mungkin.
- Elemen persoalan optimasi: kendala (constraints) dan fungsi objektif (atau fungsi optimasi)





# CONTOH

- Masalah penukaran uang
- Diberikan uang senilai A. Tukar A dengan koin-koin uang yang ada.
- Berapa jumlah minimum koin yang diperlukan untuk penukaran tersebut?
- Persoalan Minimasi
- Contoh tersedia banyak koin 1, 5, 10, 25

# LANJUTAN

- ▶ Uang senilai  $A = 32$  dapat ditukar dengan banyak cara berikut :
  - ▶  $32 = 1 + 1 + 1 + \dots + 1$  (32 koin)
  - ▶  $32 = 5 + 5 + 5 + 5 + 10 + 1 + 1$  (7 koin)
  - ▶  $32 = 10 + 10 + 10 + 1 + 1$  (5 koin)
  - ▶ ...dst
- ▶ Minimum :
  - ▶  $32 = 25 + 5 + 1 + 1$  (4 koin)
- ▶ Tinjau masalah penukaran uang
- ▶ Strategi greedy : pada setiap langkah, pilihlah koin dengan nilai terbesar dari himpunan koin yang tersisa.



# LANJUTAN

- ▶ Misal  $A = 32$ , koin yang tersedia : 1, 5, 10, dan 25
  - ▶ Langkah 1 : pilih satu buah koin bernilai 25 (total = 25)
  - ▶ Langkah 2 : pilih satu buah koin bernilai 5 (total =  $25 + 5 = 30$ )
  - ▶ Langkah 3 : pilih dua buah koin bernilai 1 (total =  $25 + 5 + 1 = 31$ )
- ▶ Solusi : jumlah koin minimum = 4 (Solusi optimal)
- ▶ Elemen-elemen algoritma greedy :
  - ▶ Himpunan kandidat,  $C$ .
  - ▶ Himpunan solusi,  $S$ .
  - ▶ Fungsi seleksi (Selection function).
  - ▶ Fungsi kelayakan (feasible).
  - ▶ Fungsi obyektif.



# LANJUTAN

- Dengan kata lain :
- Algoritma greedy melibatkan pencarian sebuah himpunan bagian  $S$  dari himpunan bagian  $C$ , yang dalam hal ini,  $S$  harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan  $S$  dioptimasi oleh fungsi obyektif.



# LANJUTAN



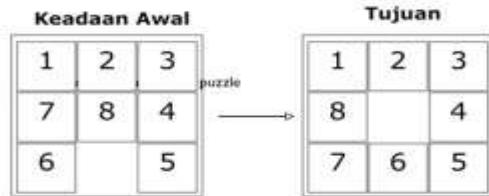
- ▶ Pada masalah penukaran mata uang :
  - ▶ Himpunan kandidat : himpunan koin yang merepresentasikan nilai 1, 5, 10, dan 25, paling sedikit mengandung satu koin untuk setiap nilai.
  - ▶ Himpunan solusi : total nilai koin yang dipilih tepat, sama jumlahnya dengan nilai uang yang ditukarkan.
  - ▶ Fungsi seleksi : pilihlah nilai koin yang bernilai paling tinggi dari himpunan kandidat yang tersisa.
  - ▶ Fungsi layak : memeriksa apakah nilai total dari himpunan koin yang dipilih tidak melebihi jumlah uang akan di tukarkan.
  - ▶ Fungsi obyektif : jumlah koin yang digunakan minimum.



# PENCARIAN HEURISTIK

- Pencarian Heuristik
- Pencarian buta tidak selalu dapat diterapkan dengan baik
- Waktu aksesnya yang cukup lama
- Besarnya memori yang diperlukan
- Metode heuristic search diharapkan bisa menyelesaikan permasalahan yang lebih besar.
- Metode heuristic search menggunakan suatu fungsi yang menghitung biaya perkiraan (estimasi) dari suatu simpul tertentu menuju ke simpul tujuan → disebut fungsi heuristic
- Aplikasi yang menggunakan fungsi heuristic : Google, Deep Blue, Chess Machine

# CONTOH

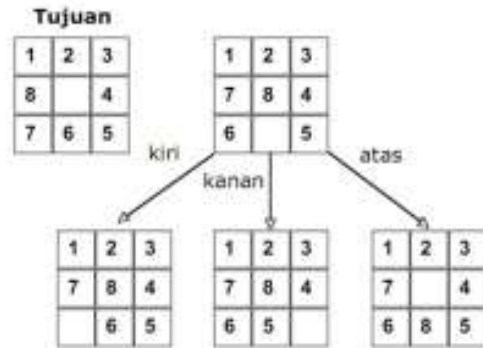


- Operator
- Ubin kosong geser ke kanan
- Ubin kosong geser ke kiri
- Ubin kosong geser ke atas
- Ubin kosong geser ke bawah



# LANJUTAN

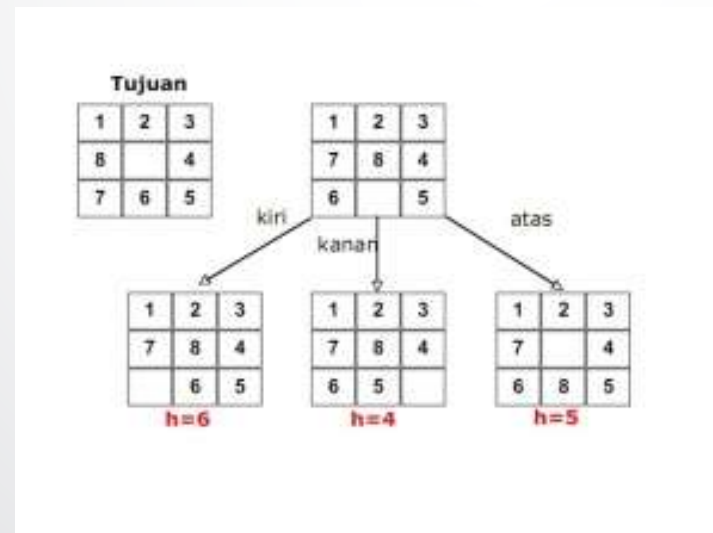
## Langkah Awal



- ▶ Langkah Awal hanya 3 operator yang bisa digunakan
- ▶ Ubin kosong digeser ke kiri, ke kanan dan ke atas.
- ▶ Jika menggunakan pencarian buta, tidak perlu mengetahui operasi apa yang akan dikerjakan (sembarang)

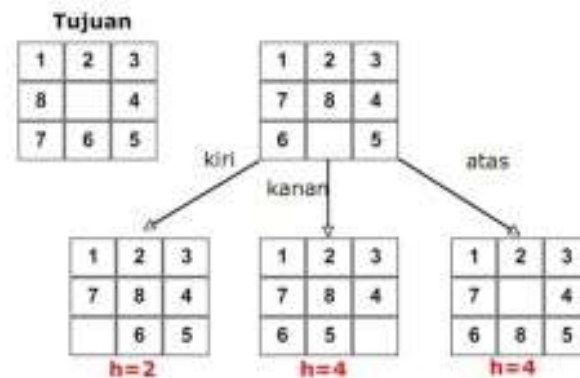
# LANJUTAN

- Pada pencarian heuristik perlu diberikan informasi khusus dalam domain tersebut
- Untuk jumlah ubin yang menempati posisi yang benar jumlah yang lebih tinggi adalah yang lebih diharapkan (lebih baik)



# LANJUTAN

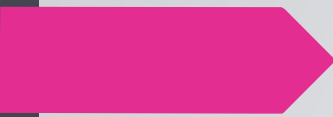
- Menghitung total gerakan yang diperlukan untuk mencapai tujuan jumlah yang lebih kecil adalah yang diharapkan (lebih baik).





# LANJUTAN

- **Ada 4 metode pencarian heuristik**
- Pembangkit & Pengujian (Generate and Test)
- Pendakian Bukit (Hill Climbing)
- Pencarian Terbaik Pertama (Best First Search)
- Simulated Annealing



# Pembangkit & Pengujian (Generate and Test)

- ▶ Pada prinsipnya metode ini merupakan penggabungan antara depth-first search dengan pelacakan mundur (backtracking), yaitu bergerak ke belakang menuju pada suatu keadaan awal.
- ▶ Algoritma:
- ▶ Bangkitkan suatu kemungkinan solusi (membangkitkan suatu titik tertentu atau lintasan tertentu dari keadaan awal).
- ▶ Uji untuk melihat apakah node tersebut benar-benar merupakan solusinya dengan cara membandingkan node tersebut atau node akhir dari suatu lintasan yang dipilih dengan kumpulan tujuan yang diharapkan.
- ▶ Jika solusi ditemukan, keluar. Jika tidak, ulangi kembali langkah yang pertama



# LANJUTAN

- **Kelemahan Pembangkit & Pengujian (Generate and Test)**
- Perlu membangkitkan semua kemungkinan sebelum dilakukan pengujian
- Membutuhkan waktu yang cukup lama dalam pencariannya



# Pendakian Bukit(Hill Climbing)

- Metode ini hampir sama dengan metode pembangkitan & pengujian, hanya saja proses pengujian dilakukan dengan menggunakan fungsi heuristik.
- Pembangkitan keadaan berikutnya sangat tergantung pada feedback dari prosedur pengetesan.
- Tes yang berupa fungsi heuristic ini akan menunjukkan seberapa baiknya nilai terkaan yang diambil terhadap keadaan-keadaan lainnya yang mungkin.





# LANJUTAN

- ▶ Algoritma
- ▶ Mulai dari keadaan awal, lakukan pengujian: jika merupakan tujuan, maka berhenti; dan jika tidak, lanjutkan dengan keadaan sekarang sebagai keadaan awal.
- ▶ Kerjakan langkah-langkah berikut sampai solusinya ditemukan, atau sampai tidak ada operator baru yang akan diaplikasikan pada keadaan sekarang:
- ▶ Cari operator yang belum pernah digunakan; gunakan operator ini untuk mendapatkan keadaan yang baru.
- ▶ Evaluasi keadaan baru tersebut.
- ▶ Jika keadaan baru merupakan tujuan, keluar.
- ▶ Jika bukan tujuan, namun nilainya lebih baik daripada keadaan sekarang, maka jadikan keadaan baru tersebut menjadi keadaan sekarang.
- ▶ Jika keadaan baru tidak lebih baik daripada keadaan sekarang, maka lanjutkan iterasi.



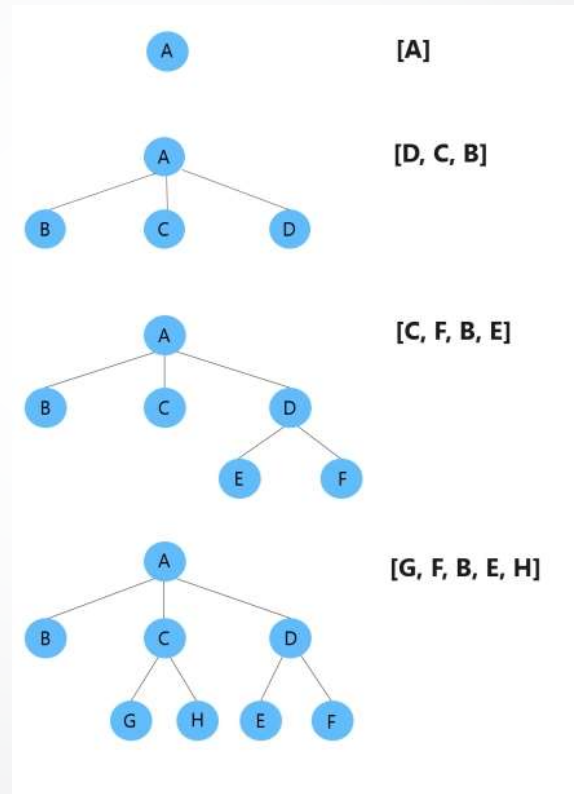
# Pencarian – Best First Search

- Pencarian Terbaik Pertama ( Best First Search )
- Kombinasi dari depth-first search dan breadth-first search dengan mengambil kelebihan dari kedua metode tersebut.
- Pencarian diperbolehkan mengunjungi node yang lebih rendah.
- Penentuan node berikutnya adalah node yang terbaik yang pernah dibangkitkan.
- Menggunakan informasi :
  - 1. Biaya perkiraan
  - 2. Biaya sebenarnya

# LANJUTAN

- ▶ Dua jenis Best First Search
  - ▶ 1. Greedy Best First Search
    - ▶ Biaya perkiraan
    - ▶  $f(n) = h(n)$
  - ▶ 2.  $A^*$ 
    - ▶ Biaya perkiraan + biaya sebenarnya
    - ▶  $f(n) = g(n) + h(n)$
- ▶ Implementasi memakai Graph keadaan, dibutuhkan dua antrian yang berisi node – node, yaitu :
  - ▶ 1. OPEN
    - ▶ Berisi node – node yang sudah dibangkitkan, namun belum diuji.
  - ▶ 2. CLOSED
    - ▶ Berisi node – node yang sudah diuji.

# LANJUTAN





# Algoritma SA (Simulated Annealing)

- ▶ Algoritma SA (Simulated Annealing) adalah salah satu algoritma yang digunakan untuk penjadwalan (scheduling).
- ▶ Tetapi bisa juga digunakan untuk pencarian jalur.
- ▶ Contoh yang dibahas kali ini adalah mengenai penjadwalan sistem kerja karyawan terhadap pekerjaan dengan waktu kerja paling minimal



# LANJUTAN

- Diasumsikan ada 5 karyawan dan 6 pekerjaan yang harus dilakukan Pekerjaan tertentu hanya dapat dilakukan oleh karyawan tertentu dan membutuhkan waktu yang berbeda-beda Jika seorang karyawan melakukan lebih dari 1 pekerjaan, maka untuk berpindah ke pekerjaan berikutnya akan terkena tambahan waktu untuk mempersiapkan diri Diasumsikan data awal mengenai karyawan, pekerjaan, dan waktu adalah sebagai berikut Angka 0 berarti karyawan tersebut tidak dapat melakukan pekerjaan



# LANJUTAN

(menit)	menggoreng	merebus	mencetak	menyaring	mencuci	memotong
karyawan A	15	7	5	0	0	0
karyawan B	0	3	9	7	0	0
karyawan C	0	0	7	11	7	0
karyawan D	0	0	0	13	3	9
karyawan E	5	0	0	0	5	5

# LANJUTAN

- Langkah pertama adalah memasukkan data-data yang digunakan. Contoh data awal adalah sebagai berikut:

```
Dim data(jumlahKaryawan - 1)() As Double  
data(0) = New Double() {15, 7, 5, 0, 0, 0}  
data(1) = New Double() {0, 3, 9, 7, 0, 0}  
data(2) = New Double() {0, 0, 7, 11, 7, 0}  
data(3) = New Double() {0, 0, 0, 13, 3, 9}  
data(4) = New Double() {5, 0, 0, 0, 5, 5}
```

# LANJUTAN

- Sebelum masuk kedalam langkah-langkah pembahasan algoritma, ada beberapa konstanta atau parameter yang harus diketahui, yaitu:
  - \* Tentukan jumlah karyawan yang digunakanDiasumsikan dalam kasus ini, jumlah karyawan ada 5 orang, yaitu karyawan A,B,C,D,E

```
Const jumlahKaryawan As Integer = 5
```

- \* Tentukan jumlah pekerjaan yang ada
- Diasumsikan dalam kasus ini, jumlah pekerjaan ada 6 jenis, yaitu memanaskan, mengepress, mendinginkan, membongkar, mencuci, merakit

# LANJUTAN

```
Const jumlahPekerjaan As Integer = 6
```

- ▶ \* Tentukan banyak perulangan yang dilakukan oleh proses algoritma ini.  
Diasumsikan dalam kasus ini, jumlah iterasi yang digunakan adalah 100.000

```
Dim iterasi As Integer = 0  
Dim maksIterasi As Integer = 100000
```

# LANJUTAN

- ▶ \* Tentukan nilai suhu awal yang digunakan dalam proses. Diasumsikan dalam kasus ini, nilai suhu awal adalah 10.000

```
Dim suhuSekarang As Double = 10000.0
```

- ▶ \* Tentukan faktor penurun suhu pada saat digunakan untuk proses berikutnya. Faktor penurun suhu berguna dalam proses perhitungan kemungkinan solusi berikutnya. Pada saat suhu tinggi (perhitungan awal), variabel solusi akan lebih mudah menerima solusi-solusi baru, agar tidak terjebak pada solusi awal yang mungkin tidak optimal. Pada saat suhu rendah (perhitungan akhir), variabel solusi tidak lagi menerima solusi baru, agar jawaban solusi tidak berubah-ubah. Diasumsikan dalam kasus ini, nilai faktor penurun suhu adalah 0.995, artinya: untuk setiap perulangan berikutnya suhu akan berkurang sebanyak  $0.005 * 10.000 = 50$  derajat.

# LANJUTAN

```
Dim alpha As Double = 0.995
```

- **Langkah-langkah penggunaan algoritma ini adalah**
- 1. Tentukan solusi acak untuk digunakan sebagai solusi awal

```
Dim solusi(jumlahPekerjaan - 1) As Integer
For t = 0 To jumlahPekerjaan - 1
    'Ambil nilai w secara acak
    Dim w As Integer = rnd.Next(0, jumlahKaryawan)
    ' pastikan karyawan ke w dapat melakukan pekerjaan ke t
    Do While data(w)(t) = 0.0
        w += 1
        If w > jumlahKaryawan - 1 Then
            w = 0
        End If
    Loop
    solusi(t) = w
Next t
```



# LANJUTAN

- Tentukan nilai energi dari solusi tersebut, yaitu jumlah waktu yang digunakan masing-masing karyawan untuk melakukan pekerjaan yang ada

```
Dim energi As Double = HitungEnergi(solusi, data)
```

- Lakukan proses perhitungan sebanyak jumlah perulangan dan selama suhu belum mendekati 0 derajat (poin 3 – 8)
- 3. Tentukan kemungkinan solusi lainnya dan hitung nilai energi untuk solusi ini

```
solusiBerikutnya = KemungkinanSolusiBerikutnya(solusi, data)  
energiSolusiBerikutnya = HitungEnergi(solusiBerikutnya, data)
```



# LANJUTAN

- ▶ \* Gunakan fungsi ini untuk menghitung kemungkinan solusi berikutnya. Tentukan pekerjaan secara acak, kemudian tentukan karyawan secara acak. Pastikan karyawan acak tersebut mampu melakukan pekerjaan acak yang didefinisikan sebelumnya. Perlu diperhatikan bahwa fungsi ini bisa mengembalikan nilai solusi yang sama persis dengan solusi yang sebelumnya. Dengan kata lain, tidak menghasilkan solusi baru.



# LANJUTAN



```
Private Function KemungkinanSolusiBerikutnya(ByVal solusiSekarang() As Integer, ByVal data()() As Double) As Integer()  
    Dim jumlahKaryawan As Integer = data.Length  
    Dim jumlahPekerjaan As Integer = data(0).Length  
    Dim solusi(jumlahPekerjaan - 1) As Integer  
  
    ' Tentukan pekerjaan secara acak  
    Dim pekerjaan As Integer = rnd.Next(0, jumlahPekerjaan)  
    ' Tentukan karyawan secara acak  
    Dim karyawan As Integer = rnd.Next(0, jumlahKaryawan)  
    ' Pastikan karyawan acak tersebut mampu melakukan pekerjaan acak yang didefinisikan sebelumnya  
    Do While data(karyawan)(pekerjaan) = 0.0  
        karyawan += 1  
        If karyawan > jumlahKaryawan - 1 Then  
            karyawan = 0  
        End If  
    Loop  
  
    solusiSekarang.CopyTo(solusi, 0)  
    solusi(pekerjaan) = karyawan  
    Return solusi  
End Function
```

# LANJUTAN

- \* Gunakan fungsi ini untuk menghitung nilai energi dari sebuah solusi. Jangan lupa menghitung tambahan waktu jika seorang karyawan melakukan lebih dari 1 pekerjaan.

```
Private Function HitungEnergi(ByVal solusi() As Integer, ByVal data()() As Double) As Double
    Dim hasil As Double = 0.0
    For t = 0 To solusi.Length - 1
        Dim karyawan As Integer = solusi(t)
        Dim waktu As Double = data(karyawan)(t)
        hasil += waktu
    Next t

    'Jangan lupa menghitung tambahan waktu jika seorang karyawan melakukan lebih dari 1 pekerjaan
    Dim jumlahKaryawan As Integer = data.Length
    Dim jumlahPekerjaan(jumlahKaryawan - 1) As Integer
    For t = 0 To solusi.Length - 1
        Dim karyawan As Integer = solusi(t)
        jumlahPekerjaan(karyawan) += 1
        If jumlahPekerjaan(karyawan) > 1 Then
            hasil += 5
        End If
    Next t
    Return hasil
End Function
```

# LANJUTAN

- 4. Jika solusi yang ditemukan ternyata lebih baik dari solusi umum, maka ambil solusi ini sebagai solusi terbaik

```
If energiSolusiBerikutnya < energiTerbaik Then
    solusiTerbaik = solusiBerikutnya
    energiTerbaik = energiSolusiBerikutnya

    Console.WriteLine("Iterasi ke " & iterasi & ", pada suhu " & suhuSekarang.ToString("F2") & " derajat")
    Console.Write("Solusi terbaik yang baru: ")
    For i = 0 To solusiTerbaik.Length - 1
        Console.Write(solusiTerbaik(i) & " ")
    Next i
    Console.Write(", dengan Nilai Energi = " & energiTerbaik.ToString("F2") & vbCrLf)
End If
```

# LANJUTAN

```
Dim p As Double = rnd.NextDouble()
```

- ▶ 6. Hitung probabilitas penerimaan solusi baru  
Jika nilai energi solusi baru lebih dari nilai energi sekarang, maka nilai kemungkinan penerimaan solusi baru adalah 1 (pasti diterima)  
Jika nilai energi solusi baru tidak lebih dari nilai energi sekarang, maka nilai kemungkinan penerimaan solusi baru akan dihitung dengan rumus  $\exp((e - e') / T)$   
Jika nilai suhu tinggi, maka rumus tersebut akan mengembalikan nilai mendekati angka 1, dan sebaliknya pada suhu rendah, maka rumus tersebut akan mengembalikan nilai mendekati angka 0



# LANJUTAN

```
Dim kemungkinanPenerimaanSolusiBaru As Double = 0
If energiSolusiBerikutnya < energi Then
    kemungkinanPenerimaanSolusiBaru = 1.0
Else
    kemungkinanPenerimaanSolusiBaru = Math.Exp((energi - energiSolusiBerikutnya) / suhuSekarang)
End If
```

- 7. Jika nilai kemungkinan penerimaan solusi baru lebih dari nilai acak p, maka ambil solusi ini sebagai solusi umum terbaik

```
If kemungkinanPenerimaanSolusiBaru > p Then
    solusi = solusiBerikutnya
    energi = energiSolusiBerikutnya
End If
```

# LANJUTAN

- ▶ 8. Turunkan suhu pada saat memasuki perulangan berikutnya, yaitu dengan rumus:  $\text{suhu} * \alpha$

```
suhuSekarang = suhuSekarang * alpha  
iterasi += 1
```

# LANJUTAN

```
file:///G:/Documents/Pip/Work/DOTNET/VB/Tutorial Algoritma/SA/ConsoleApplication1/bin/Debug/ConsoleApplication1.EXE
Algoritma SA (Simulated Annealing)
Contoh: Penjadwalan sistem kerja karyawan terhadap pekerjaan dengan waktu kerja paling minimal
Diasumsikan ada 5 karyawan dan 6 pekerjaan yang harus dilakukan
Pekerjaan tertentu hanya dapat dilakukan oleh karyawan tertentu dan membutuhkan waktu yang berbeda-beda
Jika seorang karyawan melakukan lebih dari 1 pekerjaan, maka untuk berpindah ke pekerjaan berikutnya akan terkena tambahan waktu untuk mempersiapkan diri
Diasumsikan data awal mengenai karyawan, pekerjaan, dan waktu adalah sebagai berikut
Angka 0 berarti karyawan tersebut tidak dapat melakukan pekerjaan
(menit), menggoreng, merebus, mencetak, menyaring, mencuci, memotong
karyawan A, 15, 7, 5, 0, 0, 0
karyawan B, 0, 3, 9, 7, 0, 0
karyawan C, 0, 0, 7, 11, 7, 0
karyawan D, 0, 0, 0, 13, 3, 9
karyawan E, 5, 0, 0, 0, 5, 5

Solusi awal: 4 0 0 2 2 3 , Nilai energi awal: 54.00

Memasuki proses utama algoritma ini
Iterasi ke 1, pada suhu 9950.00 derajat
Solusi terbaik yang baru: 4 1 0 2 2 3 , dengan Nilai Energi = 45.00
Iterasi ke 29, pada suhu 8647.08 derajat
Solusi terbaik yang baru: 4 0 2 1 3 3 , dengan Nilai Energi = 43.00
Iterasi ke 71, pada suhu 7005.49 derajat
Solusi terbaik yang baru: 4 1 0 1 2 3 , dengan Nilai Energi = 41.00
Iterasi ke 795, pada suhu 185.93 derajat
Solusi terbaik yang baru: 4 0 2 1 3 4 , dengan Nilai Energi = 39.00
Iterasi ke 860, pada suhu 134.23 derajat
Solusi terbaik yang baru: 4 1 0 1 3 4 , dengan Nilai Energi = 38.00
Suhu sudah turun mendekati 0 derajat pada iterasi ke 3675

Solusi terbaik yang ditemukan: 4 1 0 1 3 4 , dengan Nilai Energi = 38.00

Pekerjaan menggoreng diberikan kepada karyawan E dengan waktu 5.00 menit
Pekerjaan merebus diberikan kepada karyawan B dengan waktu 3.00 menit
Pekerjaan mencetak diberikan kepada karyawan A dengan waktu 5.00 menit
Pekerjaan menyaring diberikan kepada karyawan B dengan waktu 7.00 menit
Pekerjaan mencuci diberikan kepada karyawan D dengan waktu 3.00 menit
Pekerjaan memotong diberikan kepada karyawan E dengan waktu 5.00 menit
```