Fakultas Informatika
School of Computing
Telkom University
Think Big! Go Forward!

Telkom University

ANALISIS ALGORITMA

Week 07: Problem Komputasi Graf

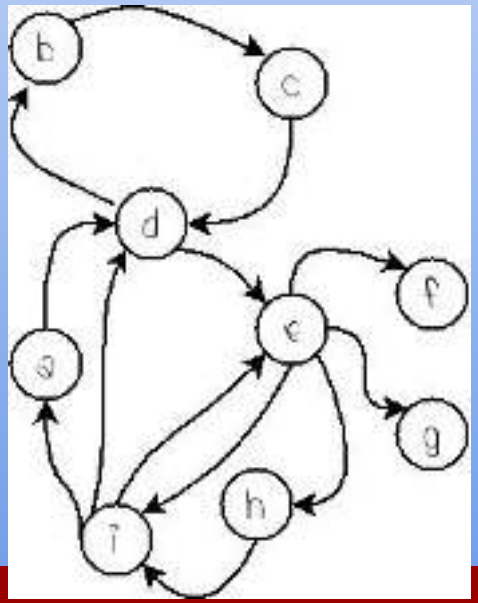
PROGRAM PASCA SARJANA INFORMATIKA
FAKULTAS TEKNIK INFORMATIKA
UNIVERSITAS TELKOM

2022/2023

Problem Graf

Metoda Breadth First Search (BFS)

- Tentukan atau diberikan verteks awal **s ∈ V** (mis s=a pada contoh)
- Kunjungi setiap verteks lain dimulai dari verteks **s**, dengan cara:
- Kunjungi setiap tetangga dari suatu verteks sebelum lanjut ke verteks lain
- Beri label setiap verteks yang dikunjungi dengan:
  - penomoran sekuensial dimulai dari s=1, dan
  - selanjutnya +1 dari nomor bapaknya/verteks sebelumnya
- Hubungan bapak-anak tersimpan
  - pohon spanning (spanning tree) dapat direkonstruksi



Algoritma BFS

```
proc Initialize( G, s )
     forall v in G.V do
          v.status := UNPROCESSED
          v.value  := +∞
          v.parent := NIL
     endfor
     G.V[s].status := INPROCESS
     G.V[s].value  := 0
endproc
```

Apa?

- Kompleksitas BFS
- Nilai tersimpan dalam field value
  - Perhatikan penggunaan queue

```
proc BFS( G, s )
     Initialize(G, s)
     Clear(Q) // Q adalah struktur queue
     Enqueue(Q, s)
     while not Empty(Q) do
          u := Dequeue(Q)
          forall v in Neighbor(G,u) do
               if v.status == UNPROCESSED then
                    v.status := INPROCESS
                    v.value  := u.value + 1
                    v.parent := u
                    Enqueue(Q, v)
               endif
          endfor
          u.status := PROCESSED
     endwhile
endproc
```

Penggunaan Queue pada BFS

- Operasi awal pada queue
  - Clear(Q)
  - Enqueue(Q, s)
- Operasi dalam loop utama
  - u := Dequeue(Q); sekali dalam setiap iterasi
  - Enqueue(Q, v); untuk semua verteks u tetangga dari u yang belum diproses
- Observasi:
  - Saudara (mempunyai nilai yang sama) akan diproses lebih dulu
  - Verteks baru datang kedalam queue adalah keponakan (dan sepupu) dari verteks yang sedang menunggu dalam queue
  - Nilai verteks2 yang berdatangan akan selalu monotonik membesar
  - **Karenanya, setiap saat dalam queue nilai verteks yang ada tidak lebih dari 2 macam saja**

Label BFS: Jalur Terpendek dari Sumber s

- Dengan Kontradiksi:
  - Misalkan ada verteks **v** dimana dimana valuei ***l*** bukan nilai terpendek, atau
  - Ada jalur lain ke **v** melalui **u'** yang lebih pendek , ***p < l***
  - Tetapi u', salah satu tetangganya, mempunyai nilai terkecil pada valuenya
  - Artinya, sepertinya jalur yang tepat adalah melalui u' untuk mencapai v
  - Dan jalur yang sekarang, melalui u menuju v agak lebih panjang
  - Artinya u' < u dan seharusnya u' masuk lebih dulu dari u kedalam Q
  - Tetapi ketika u' diproses, semua tetangganya masuk ke Q
  - Jika v adalah tetangga u', maka seharusnya masuk ke Q saat itu
  - Karena ternyata tidak ada u' yang datang lebih dulu sebelum u, terjadi kontradiksi dengan pernyataan awal diatas.
  - Kesimpulannya v sudah mendapatkan nilai terkecil/terpendek, dan jalur melalui u

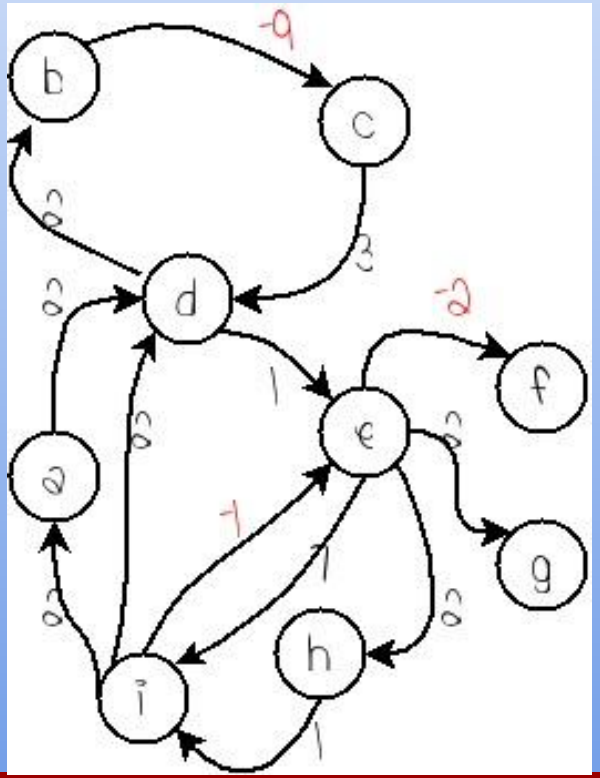
Problem Jalur Terpendek (Shortest Path)

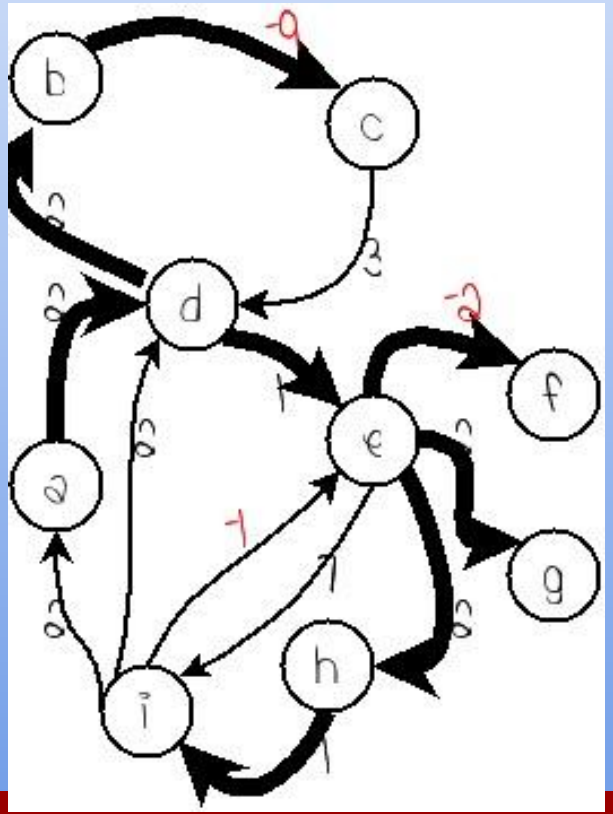
- Diberikan graf berbobot (dan berarah) G=(V,E)
  - Cari jalur terpendek dari **s** ke **t**
  - **Cari jalur-jalur terpendek dari s ke semua verteks lain**
  - Cari jalur-jalur terpendek dari setiap verteks ke semua verteks lain
- Setiap verteks hanya boleh dikunjungi maksimum 1x
  - Kunjungan lebih dari 1x akan membentuk loop
  - Jika total bobot loop adalah positif, kunjungan lebih dari 1x sangat tidak diperlukan
  - Jika total bobot loop adalah negatif, kunjungan secara infinit akan tidak adil

Shortest Path Problem, Properti

- Jarak terpendek dari **s** ke **v** tidak akan lebih dari jarak terpendek dari **s** ke **u** + bobot edge (**u,v**), dimana **(u,v) ∈ E**
  - Ketidaksamaan segitiga: **δ(s,v) ≤ δ(s,u) + d(u,v)** dimana **(u,v) ∈ E**
  - Jika suatu **v.value = δ(s,v)** maka **v.value** tidak akan berubah lagi
  - Jika semua **v.value** sudah tidak berubah lagi, maka jalur terpendek dari s ke semua node sudah ditemukan
- **δ(s,t) = δ(s,v) + δ(v,t)** adalah jalur terpendek dari **s** ke **t**, jika:
  - **δ(s,v)** adalah jalur terpendek dari **s** ke **v** dan
  - **δ(v,t)** adalah jalur terpendek dari **v** ke **t**
- Beberapa ide algoritma:
  - Bellman-Ford: meng-update informasi jalur terpendek untuk setiap verteks sampai tidak berubah
  - Dijkstra: menentukan jalur terpendek node berikutnya berdasarkan jalut terpendek bapaknya

Contoh Shortest Path (s=a)





Algoritma BFS (Revisited)

```
proc Initialize( G, s )
     forall v in G.V do
          v.status := UNPROCESSED
          v.value  := +∞
          v.parent := NIL
     endfor
     G.V[s].status := INPROCESS
     G.V[s].value  := 0
endproc
```

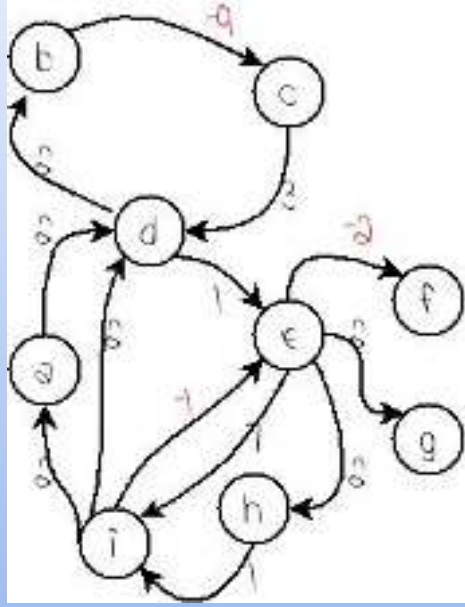
```
proc BFS( G, s )
     Initialize(G, s)
     Clear(Q) // Q is a priority queue
     Enqueue(Q, s)
     while not Empty(Q) do
          u := Dequeue(Q)
          forall v in Neighbor(G,u) do
               if v.status != PROCESSED then
                    if v.value == +∞ then
                         v.status := INPROCESS
                         v.value := u.value + 1
                         v.parent := u
                         Enqueue(Q, v)
                    endif
               endif
          endfor
          u.status := PROCESSED
     endwhile
endproc
```

Algoritma Shortest Path Dijkstra

```
proc Initialize( G, s )
     forall v in G.V do
          v.status := UNPROCESSED
          v.value  := +∞
          v.parent := NIL
     endfor
     G.V[s].status := INPROCESS
     G.V[s].value  := 0
endproc
```

```
proc Dijkstra( G, s )
     Initialize(G, s)
     Clear(Q) // Q is a priority queue on value
     Enqueue(Q, s)
     while not Empty(Q) do
          u := Dequeue(Q)
          forall v in Neighbor(G,u) do
               if v.status != PROCESSED then
                    d := u.value + weight(u,v)
                    if v.value > d then
                         v.value := d
                         v.parent := u
                         Requeue(Q, v)
                    endif
               endif
          endfor
          u.status := PROCESSED
     endwhile
endproc
```

Problem Minimum Spanning Tree



- Ingat, Pohon adalah …
- Pohon spanning (dari graf G)
  - berisi semua verteks dari G
  - semua verteks terhubung
  - $|E_T| = |V| - 1$
  - Bobot pohon adalah jumlah bobot dari semua edge dalam E_T
- MST mempunyai total bobot terkecil diantara semua pohon spanning
- Beberapa ide algoritma:
  - Kruskal: Gabungan dari dua sub-MST menghasilkan sebuah sub-MST baru
  - Prim: Memilih satu verteks dengan bobot terkecil untuk memperluas sub-MST

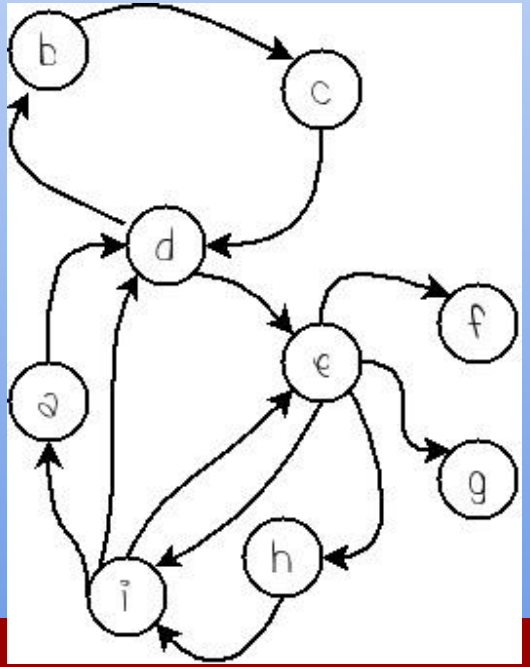
Prim MST Algorithm

```
proc Initialize( G, s )
     forall v in G.V do
          v.status := UNPROCESSED
          v.value  := +∞
          v.parent := NIL
     endfor
     G.V[s].status := INPROCESS
     G.V[s].value  := 0
endproc
```

```
proc MST( G )
     s := 1   // let any vertex as the starting point
     Initialize(G, s)
     Clear(Q) // Q is a priority queue on weight
     Enqueue(Q, s)
     while not Empty(Q) do
          u := Dequeue(Q)
          forall v in Neighbor(G,u) do
               if v.status != PROCESSED then
                    if v.value > weight(u,v) then
                         v.value := weight(u,v)
                         v.parent := u
                         Requeue(Q, v)
                    endif
               endif
          endfor
          u.status := PROCESSED
     endwhile
endproc
```

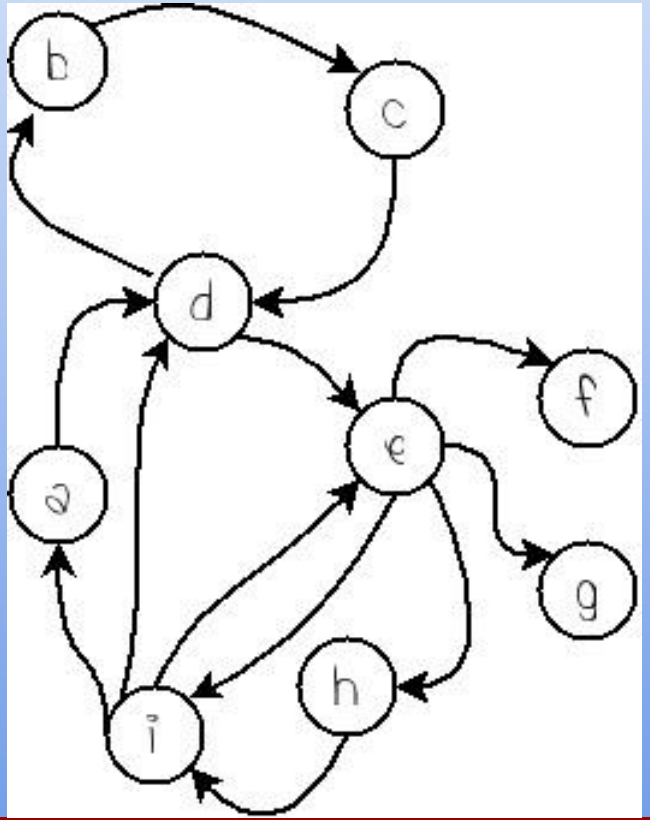
Metoda Depth First Search (DFS)

- Kunjungi setiap verteks mulai dari verteks $s \in V$
- Dilanjut mengunjungi anak pertama dari verteks yang sedang dikunjungi, atau saudaranya jika tidak ada anak lagi
- Label setiap verteks yang dikunjungi:
  - penomoran sekuensial +1 saat mulai dikunjungi
  - dan +1 lagi saat kunjungan verteks tersebut berakhir
- Relasi bapak-anak disimpan
  - Pohon spanning juga dapat direkonstruksi



DFS Process Example

- start with s=a



Algoritma DFS

```
proc Initialize( G, s )
     forall v in G.V do
          v.status := UNPROCESSED
          v.stime  := -1
          v.etime  := -1
          v.parent := NIL
     endfor
     G.V[s].status := INPROCESS
endproc
```

- Complexity of DFS

```
proc DFS( G, s )
     Initialize(G, s)
     time := 0
     Clear(Q) // Q is a stack structure
     Push(Q, s)
     while not Empty(Q) do
          u := Pop(Q)
          if u.status ≠ PROCESSED then
               forall v in Neighbor(G, s) do
                    if v.status == UNPROCESSED then
                         v.status := INPROCESS
                         v.stime  := time
                         Push(Q, v)
                    endif
               endfor
               u.status := PROCESSED
          endif
     endwhile
endproc
```