Greedy algorithms
MST

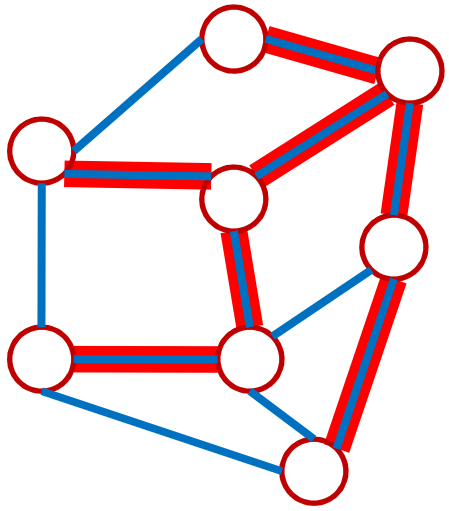
wijanarto

Materi

- Graph
- MST
- Kruskal
- Prim
- Dijkstra

(Minimum Spanning Tree)
MST

- G=(V,E) adalah Undirected graph
- Spanning Tree G adalah tree yang meliputi seluruh vertek dalam G
  - Spanning : terdiri dari seluruh vertek dalam g
  - Tree :subgraph terhubung tanpa cycle

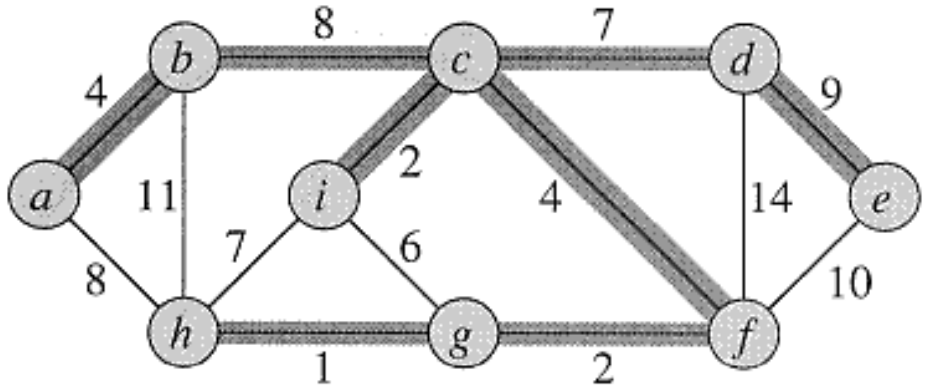


ST mempunyai |v|-1 edge

(Minimum Spanning Tree)
MST

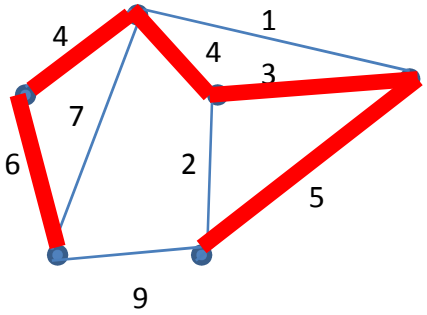
- MST adalah Spanning Tree dengan bobot/panjang minimum(panjang/bobot)
- G=(V,E), dengan ℓ:E→R^+ adalah panjang/bobot
- Panjang ST adalah Jumlahan panjang dari edge dalam Tree

MST TIDAK UNIK



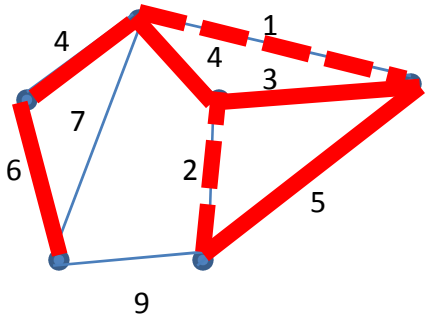
- Total bobot di atas =37, tetapi dengan menghilangkan edge(b,c) dan mengganti dengan edge (a,h) akan di dapatkan bobot 37 juga

(Minimum Spanning Tree)
MST



Length: 6+4+4+3+5=22

Spanning Tree



22-(4-1) -(5-2)=16

MST

Algoritma MST

- Mulai dari suatu tree
- Menyertakan edge dalam tree tsb
- Cari apakah ada cycle yang terbentuk dari langkah sebelumnya
- Buang edge dengan bobot atau pangjang yang lebih besar dari cycle yang terbentuk (jika ada)

Generik MST

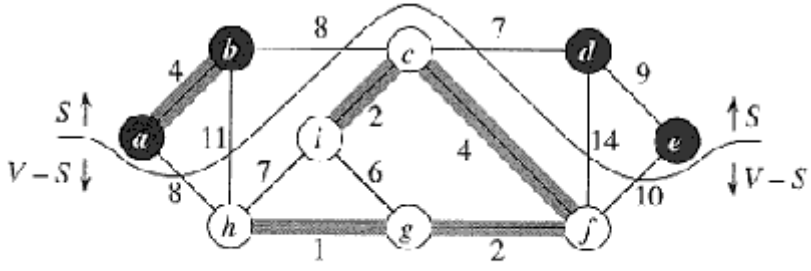
```
GEN-MST(G,w)
1. A←∅
2. while A bukan Spanning Tree do
   3. Cari edge(u,v) yang "aman"
      untuk A
   4. A←A∪{(u,v)}
5. return A
```

Generik MST

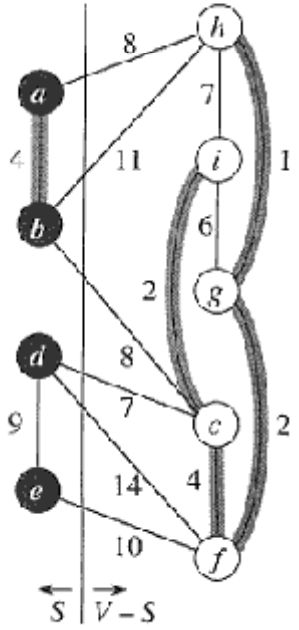
- Dari psuedo code sebelumnya, yang paling penting terletak pada baris 3
- Pada baris 3, Harus terdapat spanning tree T sehingga $A \subseteq T$, dan jika ada edge $(u,v) \in T$ sedemikian rupa sehingga $(u,v) \in A$, MAKA (u,v) “aman” untuk A

Bukti dari Generik MST

Potongan (S,V-S) dari G=(V,E) adalah partisi dari V. Kita katakan edge $(u,v) \in E$ memotong (S,V-S) jika terfapat satu titik akhir dalam S dan lainnya pada (V-S)



(a)

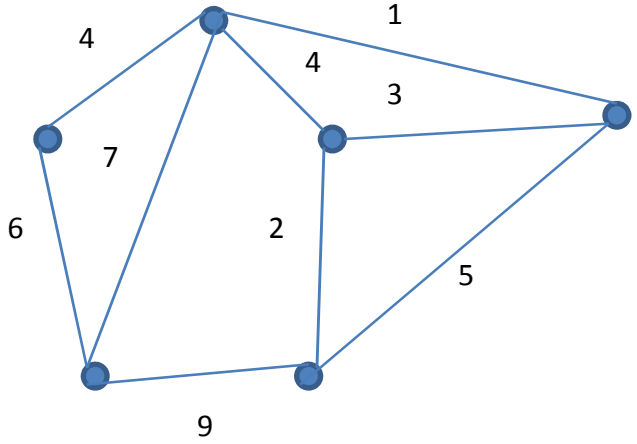


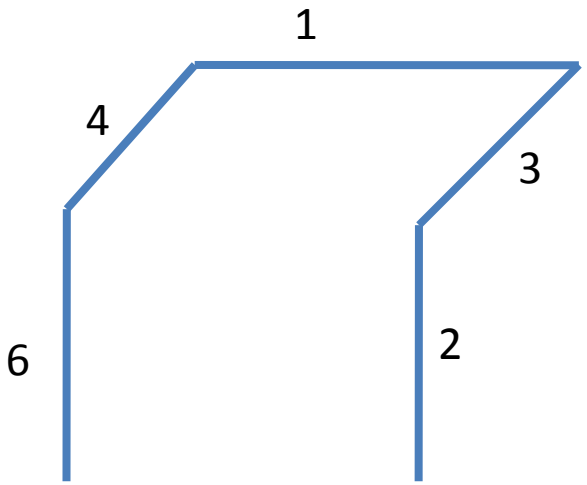
(b)

MST Kruskal

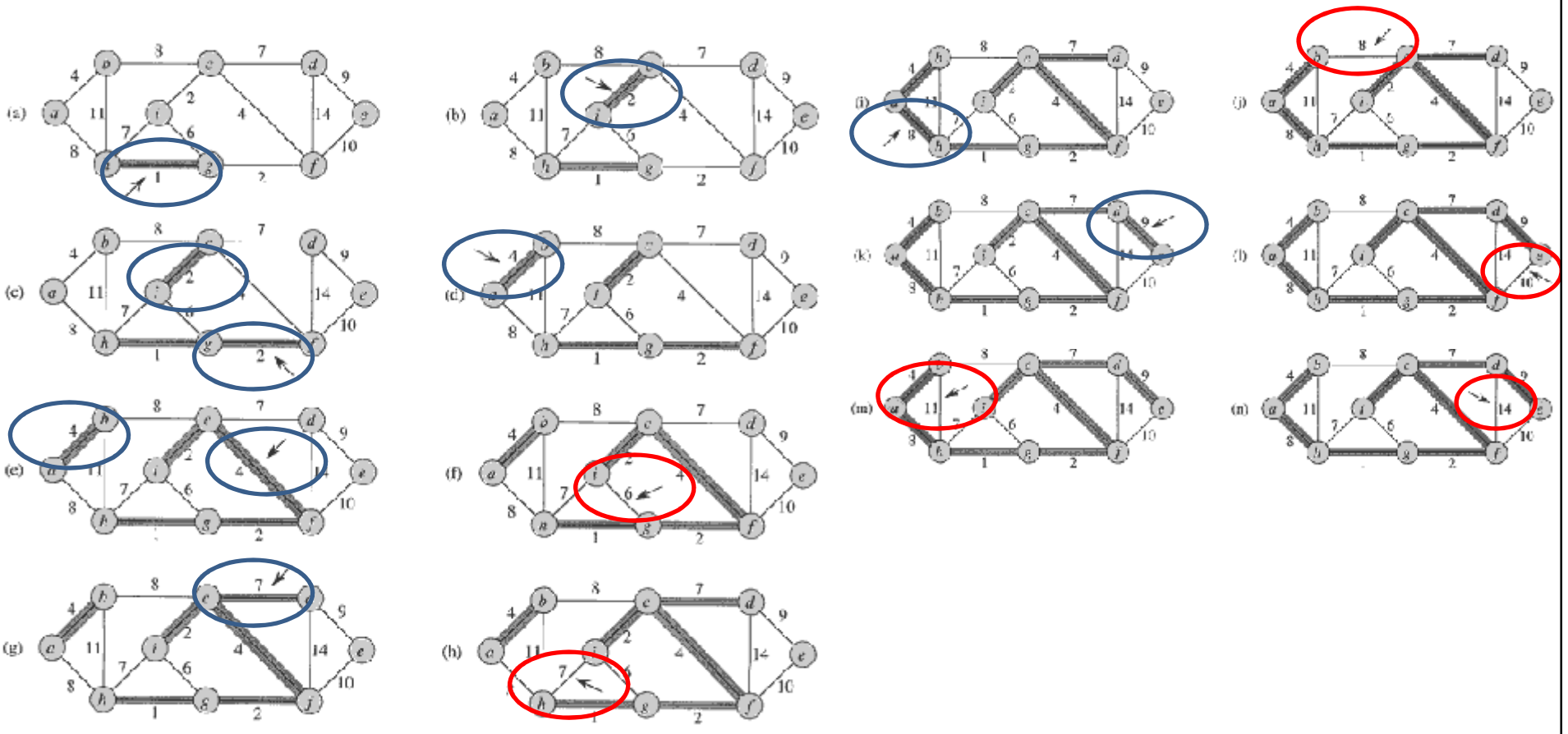
- Algo ini mendevelop MST berdasarkan edge yang diambil dg bobot minimum
- Jika edge yang diambil menyebabkan cycle maka tidak kita ambil
- Algo berhenti jika sudah membentuk Spanning Tree (semua vertek ada)

Deskripsi Grafis





Running MST-K



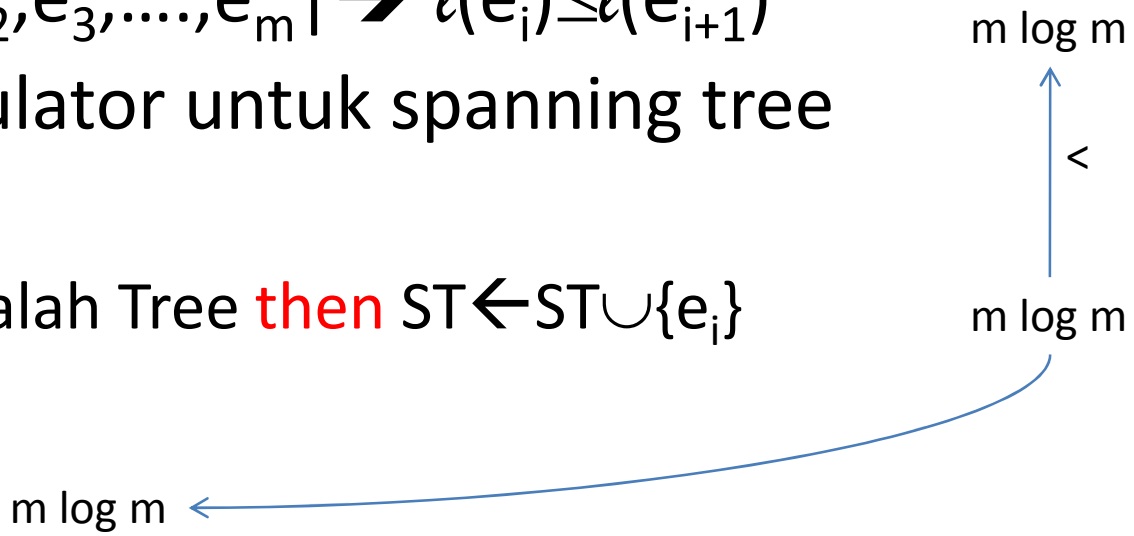
Generik MST-KRUSKAL

```
MST-KRUSKAL(G, w)
1  A ← ∅
2  for each vertex v ∈ V[G]
3       do MAKE-SET(v)
4  sort the edges of E into nondecreasing order by weight w
5  for each edge (u, v) ∈ E, taken in nondecreasing order by weight
6       do if FIND-SET(u) ≠ FIND-SET(v)
7            then A ← A ∪ {(u, v)}
8                 UNION(u, v)
9  return A
```

Analisia MST-Kruskal

1. Urutkan edge secara menaik berdasarkan bobotnya $|e_1,e_2,e_3,\ldots,e_m| \rightarrow \ell(e_i) \leq \ell(e_{i+1})$
2. ST$\leftarrow\varnothing$ //akumulator untuk spanning tree
3. For i=1 to m do

   If $\{e_i\} \cup$ ST adalah Tree then ST$\leftarrow$ST$\cup\{e_i\}$
4. Return ST



BAGAIMANA MEMERIKSA APAKAH EDGE YANG DISISIPKAN DALAM SPANNING TREE MEMBENTUK CYCLE ATAU TIDAK???

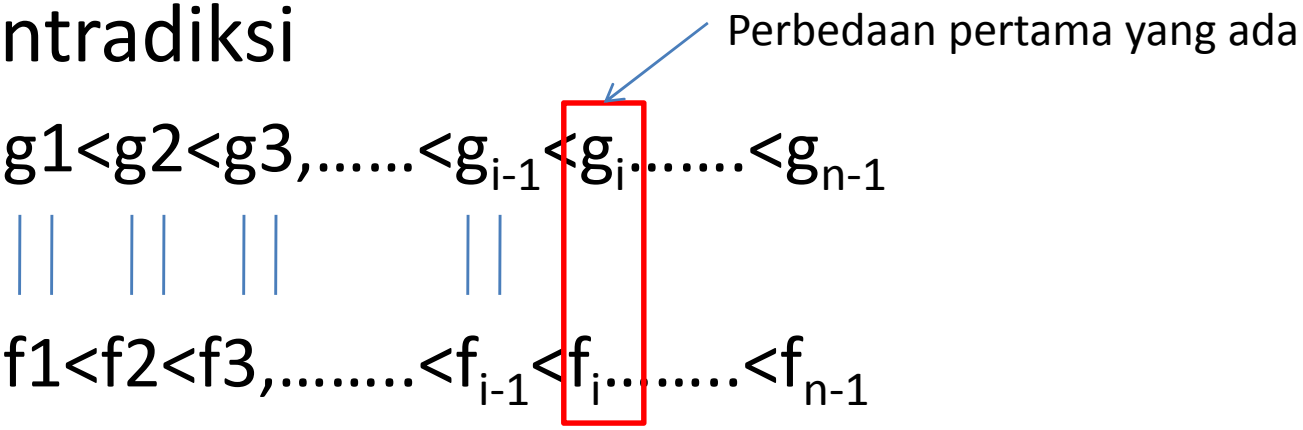
simulasi

Bukti Intuisi MST-K

- Asumsikan tiap edge berbobot beda
- Sortir tiap edge dalam graph, misal hasilnya
  - $g1<g2<g3,\ldots\ldots<g_{n-1}$
- Ambil Tiap Edge shg terbentuk optimum tree
  - $f1<f2<f3,\ldots\ldots\ldots<f_{n-1}$
- Dua himpunan edge diatas tidak harus sama (egual bobotnya)

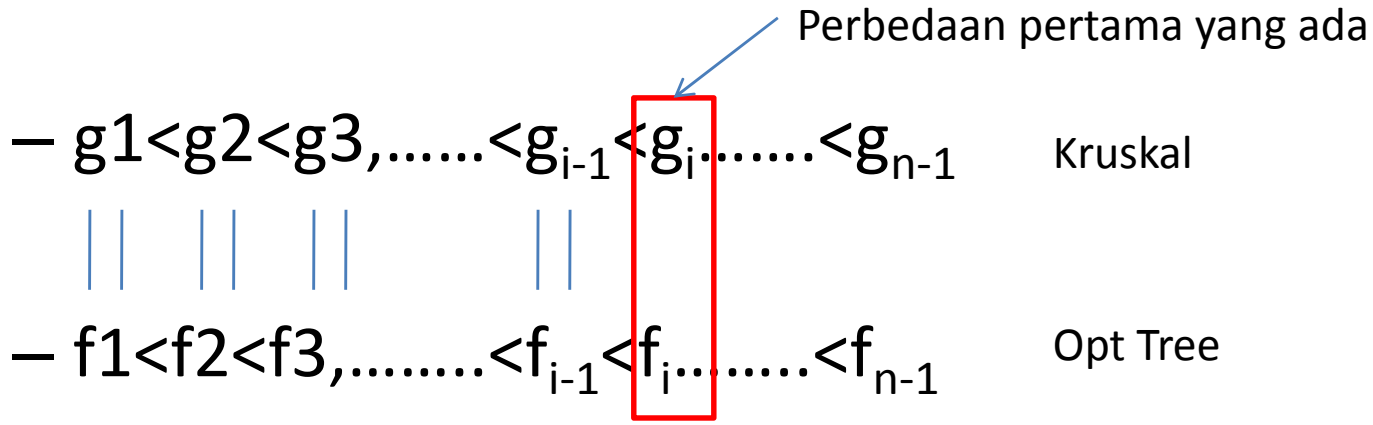
Bukti Intuisi MST-K

- Karena kedua himpunan Edge diatas tidak harus sama, maka kita buktikan dengan kontradiksi
  - $g1<g2<g3,\ldots\ldots<g_{i-1}<g_i\ldots\ldots<g_{n-1}$
  - $f1<f2<f3,\ldots\ldots\ldots<f_{i-1}<f_i\ldots\ldots\ldots<f_{n-1}$



- Dua himpunan edge diatas tidak harus sama (egual bobotnya)

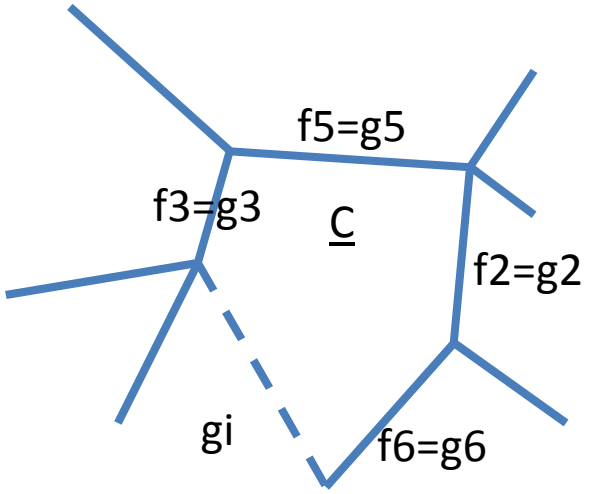
Bukti Intuisi MST-K



- Misal perbedaan yang pertama muncul di *i*
- **<u>Kasus 1</u>**: $g_i<f_i$, artinya $\ell(g_i)<\ell(f_i)$ jika demikian maka g_i tidak mungkin muncul di antara f_i dan f_{n-1}. Maka tambahkan g_i ke <u>*opt tree*</u>

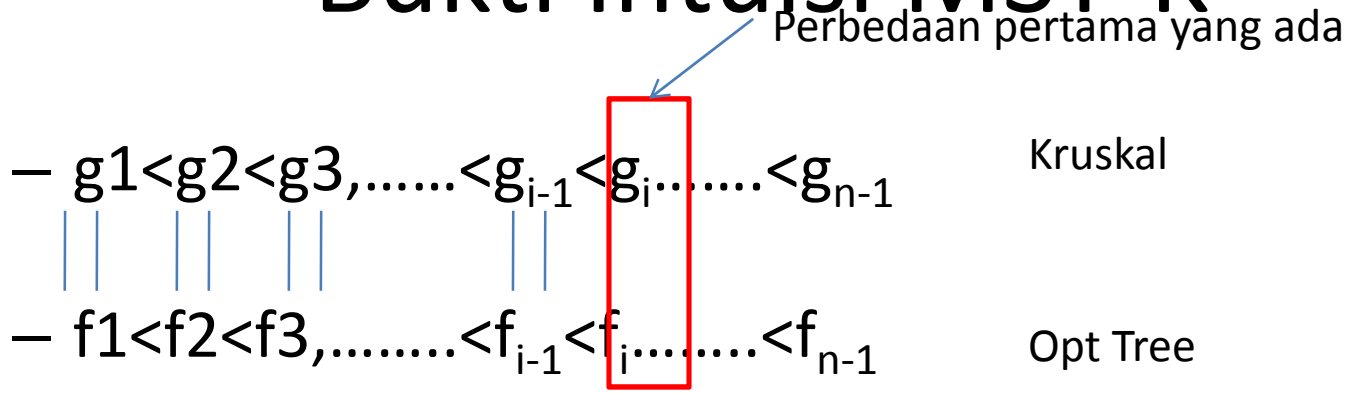
Bukti Intuisi MST-K

- Diketahuai opt tree
- Dapatkah g_i menjadi
- Edge terpanjang
- Dalam cycle C.



- Tiap edge dalam cycle C pasti lebih kecil dari g_i, sebab berada dalam $f_1 – f_{i-1}$ yang identik dengan g_1- g_{i-1}. (lihat gambar sebelumnya kruskal dan opt tree)

Bukti Intujsi MST-K



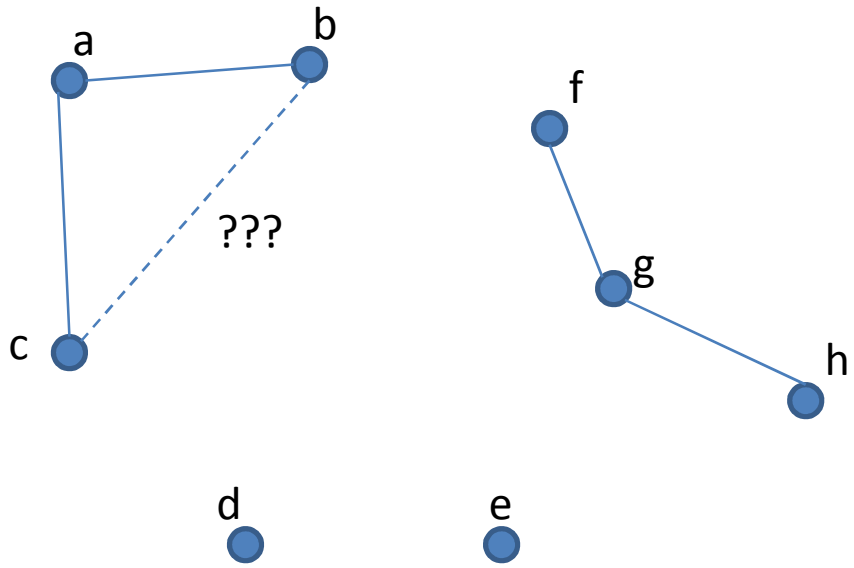
- Misal perbedaan yang pertama muncul di *i*
- **<u>Kasus 2</u>**: $g_i > f_i$, artinya $\ell(g_i) > \ell(f_i)$ jika demikian maka f_i berbeda dari g_i - g_{n-1}.
- *Kenapa Kruskal tidak mengambil edge f_i???*
- *Karena f_i U $\{g_1..g_{n-1}\}$ mengandung cycle, dengan demikian f_i U $\{f_1..f_{n-1}\}$ juga mengandung cycle. Sehingga Opt Tree pasti mengandung cycle*

Memeriksa Cycle

- Bagaimana kita memeriksa jika sebuah cycle terbentuk pada suatu edge e=(u,v) ?
- Cycle terbentuk jika dan hanya jika u dan v sudah terhubung , artinya u dan v berada dalam connected component yang sama.
- Jadi sebenarnya kita hanya melakukan pengelolaan koleksi komponen yang ada serta melakukan merge dari himpunan connected componen yang terbentuk .

UNION Connected component

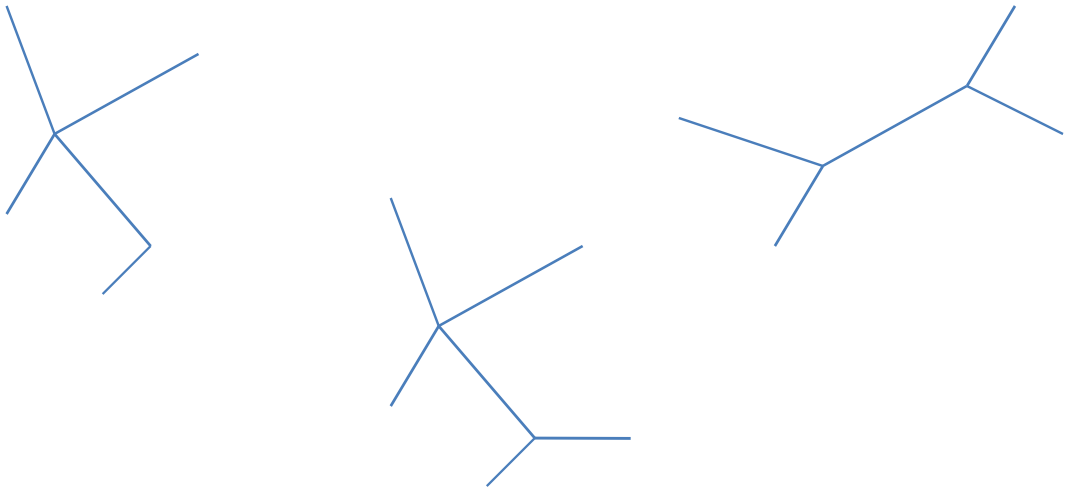
- {a,b,c}
- {f,g,h}
- {d}
- {e}



connected component adalah 4

Union-Find Data structure for MST-Kruskal

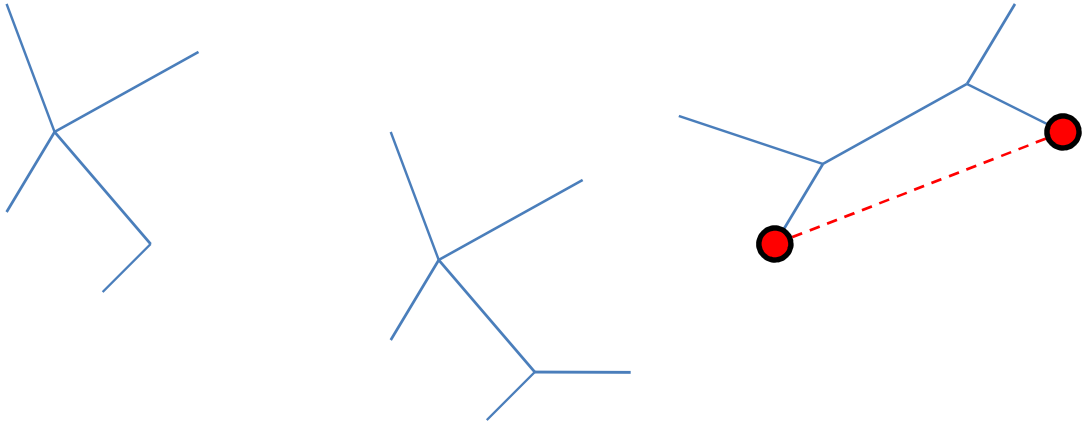
- Misalkan kita sudah mengambil himpunan data edge dalam suatu forest



- Lalu selanjutnya…

Union-Find Data structure for MST-Kruskal

- Kita akan check edge yang ditambahkan membentuk cycle atau tidak



- Cara memeriksanya adalah apakah 2 endpoint dari edge baru tadi berada dalam tree yang sama dalam forest.

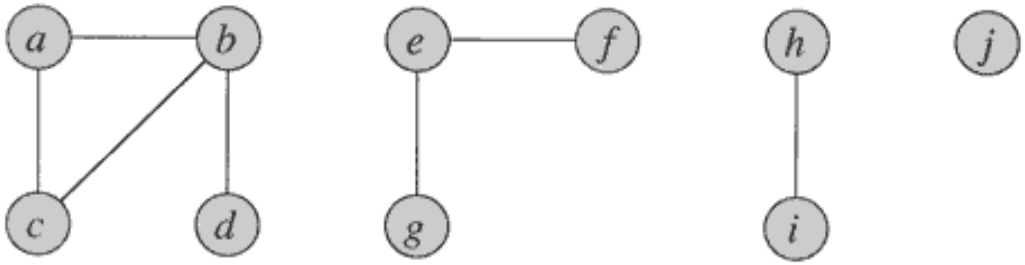
Union-Find Data structure for MST-Kruskal

- Jadi algoritma akan di mulai dengan n connected component
- Dan berakhir dengan satu connected component
- Misal kita definisikan Universe dari CC
  - U={e1,e2,e3,…,en}
- Inisialkan tiap CC dalam himpunan menjadi disjoint data struktur
  - {e1}{e2}{e3}…{en}
- Dimana e1…en adalah himpinan vertek

Union-Find Data structure for MST-Kruskal

- Tentukan operasi pada ADT disjoint
- MAKE-SET
- UNION
- LINK
- FINDSET

Union-Find Data structure for MST-Kruskal



(a)

UNION

Himpunan Ordered Tree

Edge processed	Collection of disjoint sets									
initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b,d)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e,g)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(a,c)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(h,i)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(a,b)	$\{a,b,c,d\}$				$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(e,f)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$
(b,c)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$

(b)

Union-Find Data structure for MST-Kruskal

$\textsc{Make-Set}(x)$

1 $p[x] \leftarrow x$

2 $rank[x] \leftarrow 0$

Union-Find Data structure for MST-Kruskal

```
UNION(x, y)
1   LINK(FIND-SET(x), FIND-SET(y))
```

Union-Find Data structure for MST-Kruskal

```
LINK(x, y)
1   if rank[x] > rank[y]
2       then p[y] ← x
3       else p[x] ← y
4            if rank[x] = rank[y]
5               then rank[y] ← rank[y] + 1
```

Union-Find Data structure for MST-Kruskal

```
FIND-SET(x)
1   if x ≠ p[x]
2       then p[x] ← FIND-SET(p[x])
3   return p[x]
```

SD Kruskal

```
#define MAXVERTICES 10
#define MAXEDGES    20
typedef enum {FALSE, TRUE} bool;
int edges[][3] =  {
                    {0,1,16},
                    {0,4,19},
                    {0,5,21},
                    {1,2,5},
                    {1,3,6},
                    {1,5,11},
                    {2,3,10},
                    {3,4,18},
                    {3,5,14},
                    {4,5,33}
                  };
```

Ambil Vertek Dari Graph

```
int getNVert(int edges[][3], int nedges) {
    int nvert = -1;
    int j;
    for( j=0; j<nedges; ++j ) {
        if( edges[j][0] > nvert )
            nvert = edges[j][0];
        if( edges[j][1] > nvert )
            nvert = edges[j][1];
    }
    return ++nvert;
}
```

Pemeriksaan Edge Pada Spanning Tree

```
bool FindSet(int edges[][3], int nedges, int v) {
    /*
     * periksa  apakah v sudah berada di spanning tree.
     * dg demikian kita dapat melihat bhw ada edge di v yang
     * punya cost negatif. cost negative menandakan bhw edge
     * termasuk dalam spanning tree.
     */
int j;
for(j=0; j<nedges; ++j)
   if(edges[j][2]<0 && (edges[j][0]==v || edges[j][1] == v))
     return TRUE;

return FALSE;
}
```

MST Kruskal

```
void spanning(int edges[][3], int nedges) {
 /* temukan spanning tree dr graph yg punya edges dengan kruskal asumsikan seluruh cost
    bernilai positive.    */
 int i, j, tv1, tv2, tcost; int nspanedges = 0;   int nvert = getNVert(edges, nedges);
 // urutkan cost pada edge.
 for(i=0; i<nedges-1; ++i)
    for(j=i; j<nedges; ++j)
       if(edges[i][2] > edges[j][2]) {
         tv1=edges[i][0]; tv2=edges[i][1]; tcost=edges[i][2];
         edges[i][0]=edges[j][0]; edges[i][1]=edges[j][1]; edges[i][2]=edges[j][2];
         edges[j][0]=tv1; edges[j][1]=tv2; edges[j][2]=tcost;
       }
       for(j=0; j<nedges-1; ++j) {       // apakah  edge j berhub dg vertek v1 dan v2.
           int v1 = edges[j][0];    int v2 = edges[j][1];
// cek apakah ada cycle yg terbentuk hingga spanning tree yg skarang. pemeriksaan
//dpt di selesaikan dg mudah dg memeriksa v1 dan v2 di spanning tree!
           if(FindSet(edges, nedges, v1) && FindSet(edges, nedges, v2)) // cycle.
             printf("ditolak: %d %d %d...\n", edges[j][0], edges[j][1], edges[j][2]);
           else {  edges[j][2] = -edges[j][2];
                   printf("%d %d %d.\n", edges[j][0], edges[j][1], - edges[j][2]);
                   if(++nspanedges == nvert-1)    return;
            }
        }
        printf("tidak ada spanning tree utk graph.\n");
    }
```

SAMPLE OUTPUT

GRAPH ADJ.MATRIK

0 1 16
0 4 19
0 5 21
1 2 5
1 3 6
1 5 11
2 3 10
3 4 18
3 5 14
4 5 33

1 2 5.
1 3 6.
ditolak: 2 3 10...
1 5 11.
ditolak: 3 5 14...
0 1 16.
3 4 18.