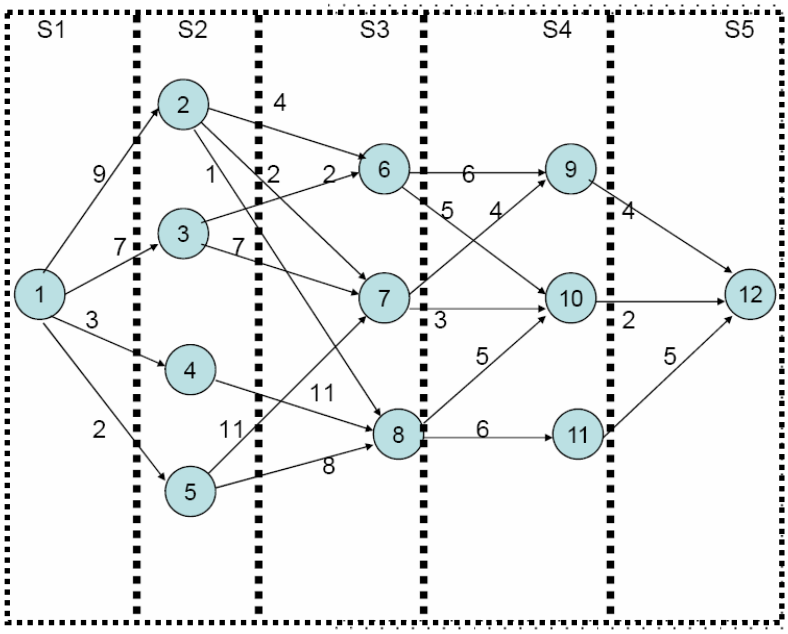
2008

MAKALAH

STRUKTUR DATA

PENGANTAR GRAPH



NAMA

IMAM CIPTARJO

NRP

6307130

KELAS

1-TI-1

POLITEKNIK KOMPUTER NIAGA

LPKIA BANDUNG

Dalam istilah ilmu komputer, sebuah struktur data adalah cara penyimpanan, pengorganisasian dan pengaturan data di dalam media penyimpanan komputer sehingga data tersebut dapat digunakan secara efisien. Dalam tehnik pemrograman, struktur data berarti tata letak data yang berisi kolom-kolom data, baik itu kolom yang tampak oleh pengguna (user) ataupun kolom yang hanya digunakan untuk keperluan pemrograman yang tiadak tampak oleh pengguna.

Struktur Data Graph

Struktur data yang berbentuk network/jaringan, hubungan antar elemen adalah many-to-many

Keterhubungan dan jarak tidak langsung antara dua kota = data keterhubungan langsung dari kota-kota lainnya yang memperantarainya.

Penerapan struktur data linear atau hirarkis pada masalah graph dapat dilakukan tetapi kurang efisien. Struktur data graph secara eksplisit menyatakan keterhubungan ini sehingga pencariannya langsung (straight forward) dilakukan pada strukturnya sendiri.

1. Struktur Data Linear = keterhubungan sekuensial antara entitas data
2. Struktur Data Tree = keterhubungan hirarkis
3. Struktur Data Graph = keterhubungan tak terbatas antara entitas data.

   Contoh graph : *Informasi topologi jaringan dan keterhubungan antar kota-kota*

Graph terdiri dari himpunan verteks (node) dan himpunan sisi (edge, arc). Verteks menyatakan entitas-entitas data dan sisi menyatakan keterhubungan antara verteks.

Notasi matematis graph G adalah :

$$G = (V, E)$$

Sebuah sisi antara verteks x dan y ditulis {x, y}.

Subgraph : graph yang merupakan suatu subset (bagian) graph yang connected

Graph H = (V1, E1) disebut subgraph dari graph G jika V1 adalah himpunan bagian dari V dan E1 himpunan bagian dari E.

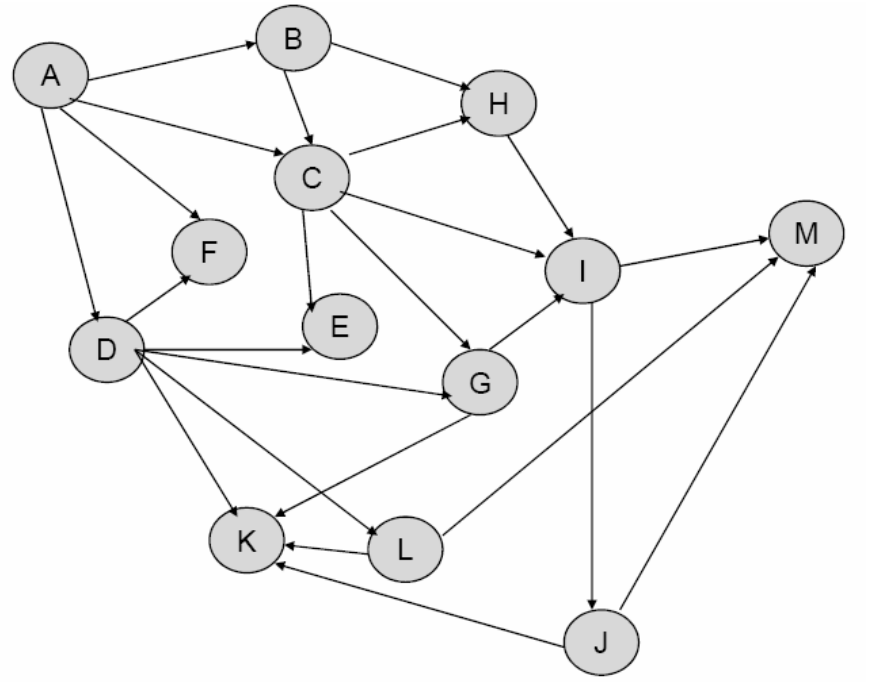
Jenis Graph

a. Directed Graph (Digraph)

Jika sisi-sisi graph hanya berlaku satu arah. Misalnya : {x,y} yaitu arah x ke y, bukan dari y ke x; x disebut origin dan y disebut terminus. Secara notasi sisi digraph ditulis sebagai vektor (x, y).

Contoh Digraph G = {V, E} :
V = {A, B, C, D, E, F, G, H, I,J, K, L, M}

E = {(A,B),(A,C), (A,D), (A,F), (B,C), (B,H), (C,E), (C,G), (C,H), (C,I), (D,E), (D,F), (D,G), (D,K), (D,L), (E,F), (G,I), (G,K), (H,I), (I,J), (I,M), (J,K), (J,M), (L,K), (L,M)}.



b. Graph Tak Berarah (Undirected Graph atau Undigraph)
Setiap sisi {x, y} berlaku pada kedua arah: baik x ke y maupun y ke x. Secara grafis sisi pada undigraph tidak memiliki mata panah dan secara notasional menggunakan kurung kurawal.

Contoh Undigraph G = {V, E}
V = {A, B, C, D, E, F, G, H, I,J, K, L, M}

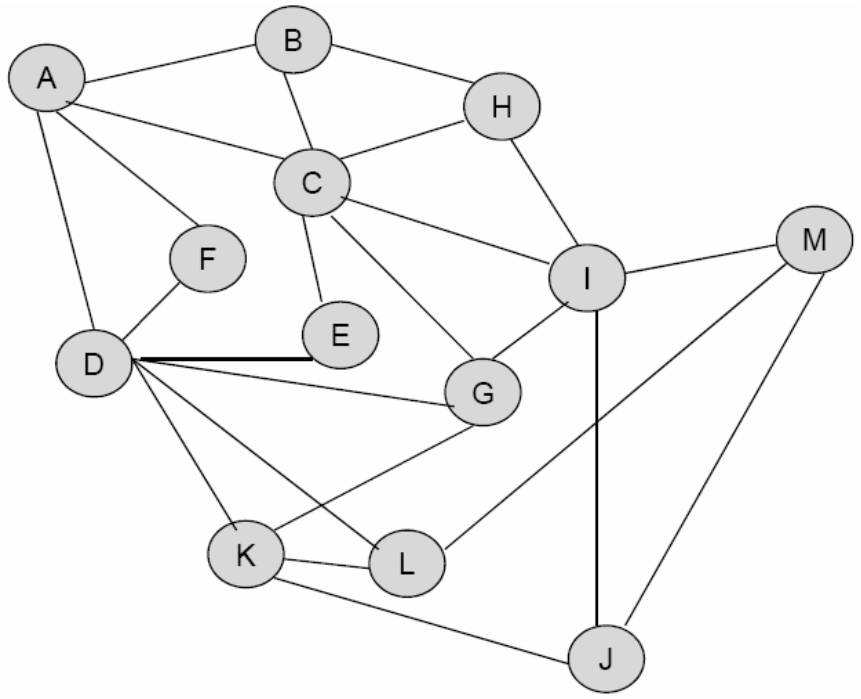
E = { {A,B},{A,C}, {A,D}, {A,F}, {B,C}, {B,H}, {C,E}, {C,G}, {C,H}, {C,I}, {D,E}, {D,F}, {D,G}, {D,K}, {D,L}, {E,F}, {G,I}, {G,K}, {H,I}, {I,J}, {I,M}, {J,K}, {J,M}, {L,K}, {L,M}}.

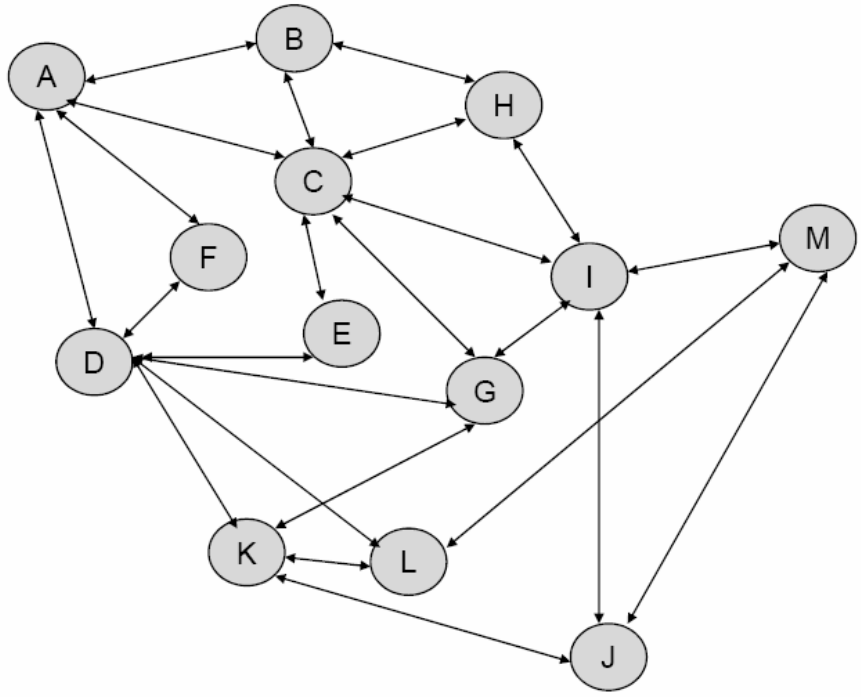
Khusus graph, undigraph bisa sebagai digraph (panah di kedua ujung edge berlawanan)

Struktur data linear maupun hirarkis adalah juga graph. Node-node pada struktur linear ataupun hirarkis adalah verteks-verteks dalam

pengertian graph dengan sisi-sisinya menyusun node-node tersebut secara linear atau hirarkis.

Struktur data linear adalah juga tree dengan pencabangan pada setiap node hanya satu atau tidak ada. Linear 1-way linked list (digraph), linear 2-way linked list (undigraph).





Masalah-Masalah Graph

Masalah path minimum (Shortest path problem)

Mencari route dengan jarak terpendek dalam suatu jaringan transportasi.

Masalah aliran maksimum (maximum flow problem)

Menghitung volume aliran BBM dari suatu reservoir ke suatu titik tujuan melalui jaringan pipa.

Masalah pencarian dalam graph (graph searching problem)

Mencari langkah-langkah terbaik dalam program permainan catur komputer.

Masalah pengurutan topologis (topological ordering problem)

Menentukan urutan pengambilan mata-mata kuliah yang saling berkaitan dalam hubungan prasyarat (prerequisite).

Masalah jaringan tugas (Task Network Problem)

Membuat penjadwalan pengerjaan suatu proyek yang memungkinkan waktu penyelesaian tersingkat.

Masalah pencarian pohon rentang minimum (Minimum Spanning Tree Problem)

Mencari rentangan kabel listrik yang totalnya adalah minimal untuk menghubungkan sejumlah kota.

Travelling Salesperson Problem

Tukang pos mencari lintasan terpendek melalui semua alamat penerima pos tanpa harus mendatangi suatu tempat lebih dari satu kali.

Four-color problem

Dalam menggambar peta, memberikan warna yang berbeda pada setiap propinsi yang saling bersebelahan.

Konektivitas pada Digraph dan Undigraph

Konektivitas pada Digraph

1. Terminologi pada Undigrap berlaku, kecuali dalam digraph harus dikaitkan dengan arah tertentu karena pada arah yang sebaliknya belum tentu terdefinisi.
2. **Adjacency ke/dari :** Jika ada sisi (x,y) maka pada digraph dikatakan bahwa x "adjacent ke" y atau y "adjacent dari" x. Demikian pula jika terdapat path dari x ke y maka belum tentu ada path dari y ke x. Jadi dalam digraph keterkoneksian didefinisikan lebih lanjut lagi sebagai berikut.

   Terkoneksi Kuat : Bila setiap pasangan verteks berbeda x dan y dalam S, x berkoneksi dengan y dan y berkoneksi dengan x

Terkoneksi Lemah : Bila setiap pasangan verteks berbeda x dan y dalam S, salah satu: x berkoneksi dengan y (atau y berkoneksi dengan x) dan tidak kebalikan arahnya (hanya terdefinisi satu path: dari x ke y atau sebaliknya dari y ke x).

Konektivitas pada Undigraph

1. **Adjacency :** Dua verteks x dan y yang berlainan disebut terhubung langsung (adjacent) jika ada sisi {x, y} dalam E.
2. **Path :** Urutan dari kumpulan node-node, dimana tiap node dengan node berikutnya dihubungkan dengan Edge
3. **Simple Path :** Jika node dalam path tsb hanya muncul 1 kali
4. **Panjang dari path :** jumlah sisi yang dilalui path.
5. **Siklus (cycle) :** suatu path dengan panjang lebih dari satu, dimana vertex awal dan akhir sama.
6. Siklus sederhana: dalam undigraph, siklus yang terbentuk dari tiga atau lebih verteks yang berlainan, dimana tidak ada vertex muncul lebih satu kali kecuali verteks awal/akhir.
7. Dua verteks x dan y yang berbeda dalam suatu undigraph disebut berkoneksi (connected) apabila ada path penghubung.
8. Suatu komponen terkoneksi (connected components) adalah subgraph (bagian dari graph) yang berisikan satu himpunan bagian verteks yang berkoneksi.

Graph berbobot (Weighted Graph)

Graph dengan sisi mempunyai Bobot/ Biaya. “Biaya" ini bisa mewakili banyak aspek: biaya ekonomi suatu aktifitas, jarak geografis/tempuh, waktu tempuh, tingkat kesulitan, dan lain sebagainya.

Contoh :

Graph ini merupakan Undirected Weighted Graph. Order dari verteks A = 4, verteks B = 3, dst. Adjacentcy list dari D adalah = {A, E, F, G, K, L}.

Representasi Graph

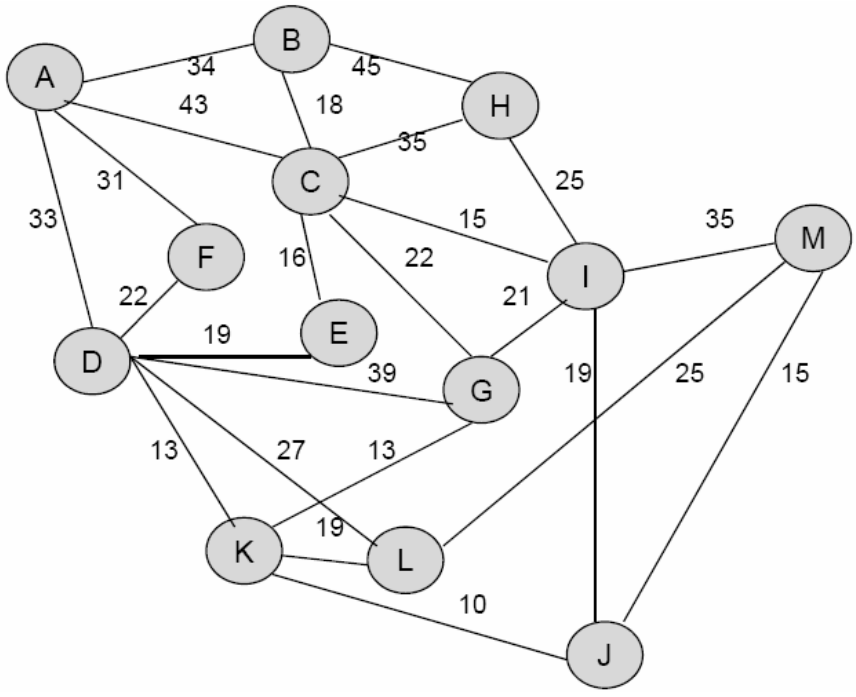
Representasi Matriks Keterhubungan Langsung (Adjacency Matrix)

Matriks digunakan untuk himpunan adjacency setiap verteks. Baris berisi vertex asal adjacency, sedangkan kolom berisi vertex tujuan adjacency. Bila sisi graph tidak mempunyai bobot, maka [x, y] adjacency disimbolkan dengan 1 dan 0 bila tidak adjacency.

Bila sisi graph mempunyai bobot, maka [x, y] adjacency disimbolkan dengan bobot sisi tersebut, dan bila tidak disimbolkan ∞

Direpresentasikan dalam matriks sebagai berikut :

Dari\Ke	A	B	C	D	E	F	G	H	I	J	K	L	M
A	-	1	1	1	0	1	0	0	0	0	0	0	0
B	1	-	1	0	0	0	0	1	0	0	0	0	0
C	1	1	-	0	1	0	1	1	1	0	0	0	0
D	1	0	0	-	1	1	1	0	0	0	1	1	0
E	0	0	1	1	-	1	0	0	0	0	0	0	0
F	1	0	0	1	1	-	0	0	0	0	0	0	0
G	0	0	1	1	0	0	-	0	1	0	1	0	0
H	0	1	1	0	0	0	0	-	1	0	0	0	0
I	0	0	1	0	0	0	1	1	-	1	0	0	1
J	0	0	0	0	0	0	0	0	1	-	1	0	1
K	0	0	0	1	0	0	1	0	0	1	-	1	0
L	0	0	0	1	0	0	0	0	0	0	1	-	1
M	0	0	0	0	0	0	0	0	1	1	0	1	-



Dari\Ke	A	B	C	D	E	F	G	H	I	J	K	L	M
A	-	24	43	33	∞	31	∞	∞	∞	∞	∞	∞	∞
B	24	-	18	∞	∞	∞	∞	45	∞	∞	∞	∞	∞
C	43	18	-	∞	16	∞	22	35	15	∞	∞	∞	∞
D	33	∞	∞	-	19	22	39	∞	∞	∞	13	27	∞
E	∞	∞	16	19	-	15	∞	∞	∞	∞	∞	∞	∞
F	31	∞	∞	22	15	-	∞	∞	∞	∞	∞	∞	∞
G	∞	∞	22	39	∞	∞	-	∞	21	∞	13	∞	∞
H	∞	45	35	∞	∞	∞	∞	-	25	∞	∞	∞	∞
I	∞	∞	15	∞	∞	∞	21	25	-	19	∞	∞	35
J	∞	∞	∞	∞	∞	∞	∞	∞	19	-	10	∞	15
K	∞	∞	∞	13	∞	∞	13	∞	∞	10	-	19	∞
L	∞	∞	∞	27	∞	∞	∞	∞	∞	∞	19	-	25
M	∞	∞	∞	∞	∞	∞	∞	∞	35	15	∞	25	-

Adjacency List : ???

Perhatikan deklarasi berikut :

```
Type
      Pointer = ^Simpul;
      Simpul = Record
            Elemen : TipeData; (* tipe data menurut pilihan kita *)
            Next : Pointer
      End;
Var
      X : Array [1..N] Of Pointer;
```

Algoritma-Algoritma Pencarian

Pencarian vertex adalah proses umum dalam graph. Terdapat 2 metoda pencarian:

Depth First Search (DFS) :

Pada setiap pencabangan, penelusuran verteks-verteks yang belum dikunjungi dilakukan secara lengkap pada pencabangan pertama, kemudian selengkapnya pada pencabangan kedua, dan seterusnya secara rekursif.

Algoritma Depth First Search :

1. Algoritma diawali pada vertex S dalam G
2. Kemudian algoritma menelusuri graph dengan suatu insiden edge (u, v) ke current vertex u.
3. Jika edge (u, v) menunjuk ke suatu vertex v yang siap untuk dikunjungi, maka kita ikuti jalur mundur ke current vertex u. Jika pada sisi lain, edge (u, v) menunjuk ke vertex v yang tidak dikunjungi, maka kita pergi ke v dan v menjadi current vertex.
4. Kita proses ini hingga kita mencapai sasaran akhir.
5. Pada titik ini, kita mulai jalur mundur. Proses ini berakhir ketika jalur mundur menunjuk balik ke awal vertex.

```
void graph()
{     for semua node u do
      {      colour[u]=white;
             p[u]=NULL;
      }
      time = 0;
      for semua node u do
             if colour[u] == white then
             DFS(u);
}
void DFS(u)
{     visit(u); time = time + 1;
      d[u] = time;colour[u] = grey;
      for semua node v adjacent ke u do
      {      if colour[v] == white then
             { p[v] = u; DFS(u); }
      }
      time = time + 1; f[u] = time;
      colour[u] = black;
}
```

Breadth First Search (BFS) :

Pada setiap pencabangan penelusuran verteks-verteks yang belum dikunjungi dilakukan pada verteks-verteks adjacent, kemudian berturut-turut selengkapnya pada masing-masing pencabangan dari setiap verteks adjacent tersebut secara rekursif.

Algoritma Breadth First Search :

BFS diawali dengan vertex yang diberikan, yang mana di level 0. Dalam stage pertama, kita kunjungi semua vertex di level 1. Stage kedua, kita kunjungi semua vertex di level 2. Disini vertex baru, yang mana adjacent ke vertex level 1, dan seterusnya. Penelusuran BFS berakhir ketika setiap vertex selesai ditemui.

Implementasi algoritma BFS

Algoritma BFS menjadi kurang straightforward jika dinyatakan secara rekursif. Jadi sebaiknya diimplementasikan secara nonrekursif dengan memanfaatkan queue sebagai struktur data pendukung

```
void BFS()
{     Queue q = new Queue();
      Vertex v, w, t;
      for (setiap vertex v dalam G)
             v.Status = false;
      for (setiap vertex v dalam G)
      {      if (v.Status == false)
             {      v.Status = true; q.Enqueue(v);
                    while (EmptyQueue(Q) == false)
                    {      w = q.Dequeue();
                           visit(w);
                           for (setiap vertex T adjacent dari w)
                           {      if (t.Status == false)
                                  {      t.Status = true;
```

```
                                        q.Enqueue(t);
                                    }
                                }
                            }
                        }
                    }
                }
```

Masalah Pencarian Lintasan Terpendek

Pencarian shortest path (lintasan terpendek) adalah masalah umum dalam suatu weighted, connected graph. Misal : Pencarian jaringan jalan raya yang menghubungkan kota-kota disuatu wilayah.

1. Lintasan terpendek yag menghubungkan antara dua kota berlainan tertentu (Single-source Single-destination Shortest Path Problems)
2. Semua lintasan terpendek masing-masing dari suatu kota ke setiap kota lainnya (Single-source Shortest Path problems)
3. Semua lintasan terpendek masing-masing antara tiap kemungkinan pasang kota yang berbeda (All-pairs Shortest Path Problems)

Untuk memecahkan masing-masing dari masalah-masalah tersebut terdapat sejumlah solusi.

- `

Dalam beberapa masalah graph lain, suatu graph dapat memiliki bobot negatif dan kasus ini dipecahkan oleh algoritma Bellman-Ford. Yang akan dibahas di sini adalah algoritma Dijkstra yaitu mencari lintasan terpendek dari suatu verteks asal tertentu vs ke setiap verteks lainnya.

Algoritma Dijkstra

Algoritma Dijkstra's :

1. Menyelesaikan problem single-source shortest-path ketika semua edge memiliki bobot tidak negatif.
2. Algoritma greedy mirip ke algoritma Prim's.
3. Algoritma di awali pada vertex sumber s, kemudian berkembang membentuk sebuah tree T, pada akhirnya periode semua vertex dijangkau dari S. Vertex di tambah ke T sesuai urutan.
   Misalnya :
   Pertama S, kemudian vertex yang tepat ke S, kemudian yang tepat berikutnya dan seterusnya.

```
DIJKSTRA (G, w, s)
{     INITIALIZE SINGLE-SOURCE (G, s)
      S ← { }
      Initialize priority queue Q
      Q ← V[G]
      while priority queue Q is not empty do
            u ← EXTRACT_MIN(Q)
```

```
            S ← S È {u}
            // Lakukan relaxation untuk setiap vertex v adjacent ke u
            for setiap vertex v dalam Adj[u] do
                  Relax (u, v, w)
}
Relax (u, v, w)
{     If d[u] + w(u, v) < d[v] then
      d[v] = d[u] + w[u, v]
}
```

Dynamic Programming :

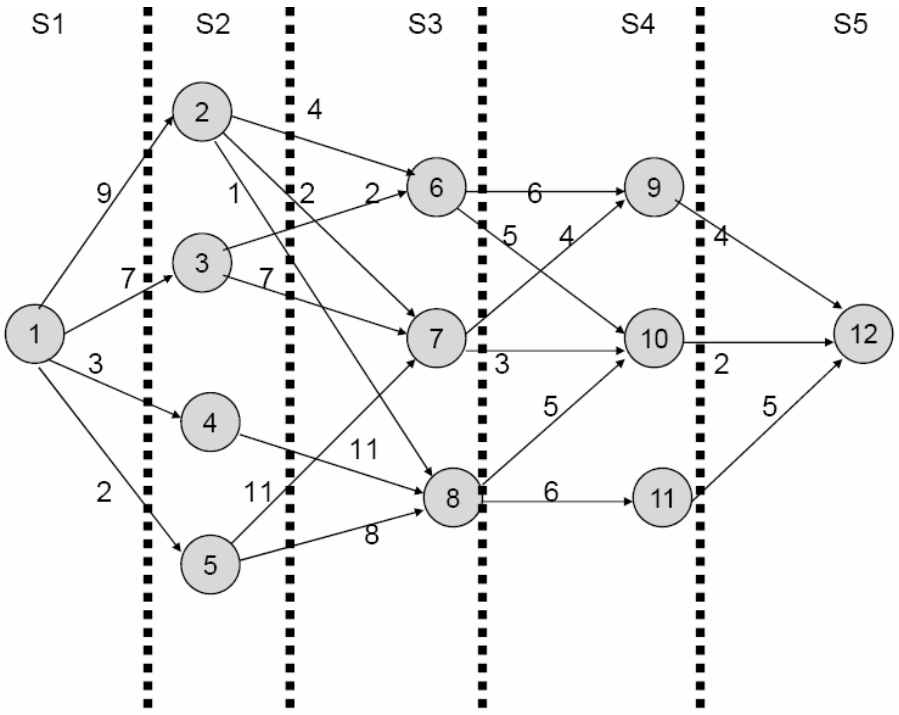
Terdiri dari sederetan tahapan keputusan. Pada setiap tahapan berlaku prinsip optimality (apapun keadaan awal dan keputusan yang diambil, keputusan berikutnya harus memberikan hasil yang optimal dengan melihat hasil keputusan sebelumnya.

Misalnya : Multistage Graph

Dimana : Cost (i,j) = Min(C(j,l) + Cost(i+1,l)}
Dengan :
C(j,l) = Bobot edge j dan l
l = Elemen Vi+1 Dan <j,l> eemen E
i=stage ke-I dan j = node dalam V

Proses dimulai dari k-2, dimana k adalah banyak stage
Perhatikan contoh untuk menentukan biaya termurah dari 1 hingga 12.

Diketahui graph dengan stage sebagai berikut :



Maka langkah-langkah yang dilakukan adalah :
K=5, sehingga dimulai dari S3
Cost(3,6) = Min{6+Cost(4,9); 5+Cost(4,10)} = Min{6+4;5+2} = 7
Cost(3,7) = Min{4+Cost(4,9); 3+Cost(4,10)} = Min{4+4;3+2} = 5
Cost(3,8) = Min{5+Cost(4,10); 6+Cost(4,11)} = Min(5+2;6+5} = 7

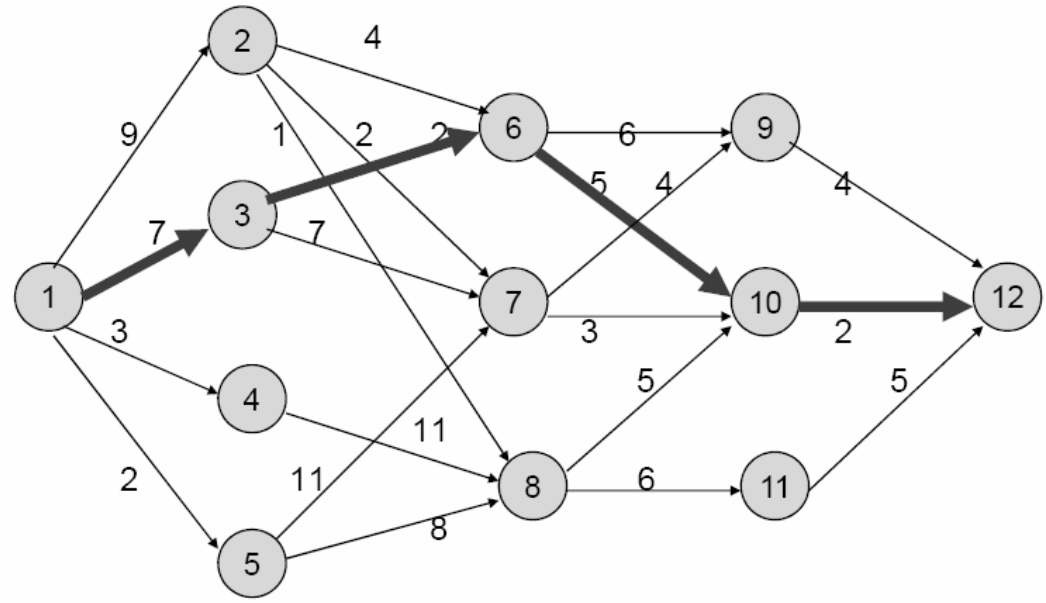
Cost(2,2) = Min{4+Cost(3,6);2+Cost(3,7);1+Cost(3,8)}
= Min{4+7;2+5;1+7} = 7
Cost(2,3) = Min{2+Cost(3,6); 7+Cost(3,7)} = Min(2+7; 7+5) = 9
Cost(2,4) = Min{11+Cost(3,8)} = 18
Cost(2,5) = Min{11+Cost(3,7); 8+Cost(3,8)} = Min(11+5;8+7} = 15
Cost(1,1) = Min{9+Cost(2,2);7+Cost(2,3);3+Cost(2,4),2+Cost(2,5)}
= Min{9+7;7+9;3+18;2+15} = 16

Jadi : Shorthest Path menjadi :

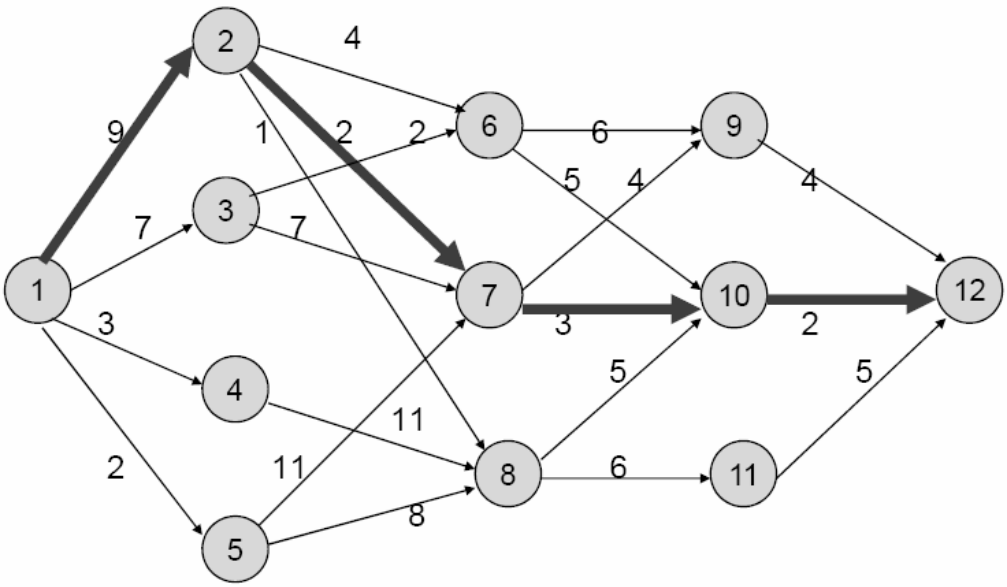
1 → 3→ 6 → 10 → 12 Atau : 1 → 2 → 7 → 10 → 12

Jika ada dua atau lebih shorthest path maka total biaya harus sama.

Shortest Path Pertama adalah :



Shortest Path Kedua adalah :

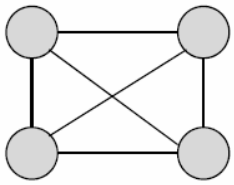


Pepresentasi & Algoritma-Algoritma

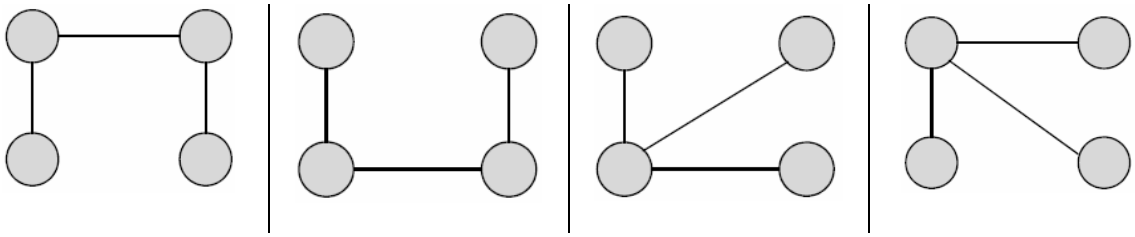
Masalah Pencarian Pohon Rentangan Minimum

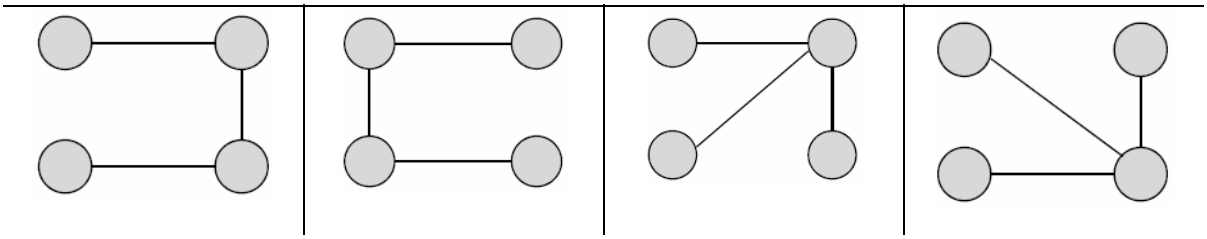
Definisi Pohon rentangan atau spanning tree dari suatu connected graph didefinisikan sebagai free-tree yang terbentuk dari subset sisi-sisi serta menghubungkan setiap verteks dalam graph tersebut. Pohon rentangan Minimum (MST) adalah pohon rentangan dengan total bobot dari sisisisinya adalah minimal. Dalam penelusuran vertex tidak diperkenankan terbentuk siklus (cycle).

Diketahui sebuah graph tak berarah dan tak berbobot sebagai berikut :



Kemungkinan Spanning Tree :





Bila jalur (edge) mempunyai biaya (cost) maka yang dicari adalah minimum cost spanning tree.

Dua algoritma populer untuk menentukan minimum spanning tree (MST) :

1. Kruskal Algorithm
2. Prim's Algorithm

Algoritma Kruskal

Algoritma ini lebih sederhana jika dilihat dari konsepnya namun lebih sulit dalam implementasinya. Idenya adalah mendapatkan satu demi satu sisi mulai dari yang berbobot terkecil untuk membentuk tree; suatu sisi walaupun berbobot kecil tidak akan diambil jika membentuk siklik dengan sisi yang sudah termasuk dalam tree.

Yang menjadi masalah dalam implementasinya adalah keperluan adanya pemeriksaan kondisi siklik tersebut.Salah satu pemecahaannya adalah dengan subsetting yaitu pembentukan subset-subset yang disjoint dan secara bertahap dilakukan penggabungan atas tiap dua subset yang berhubungan dengan suatu sisi dengan bobot terpendek. Algoritma lengkapnya:

- Tahap pertama, jika dalam V terdapat n verteks maka diinisialisasi n buah subset yang disjoint, masing-masing berisi satu verteks, sebagai subset-subset awal.
- Tahap berikutnya, urutkan sisi-sisi dengan bobot yang terkecil hingga terbesar.
- Mulai dari sisi dengan bobot terkecil hingga terbesar lakukan dalam iterasi: jika sisi tsb. menghubungkan dua verteks dalam satu subset (berarti membentuk siklik) maka skip sisi tersebut dan periksa sisi berikutnya jika tidak (berarti membentuk siklik) maka kedua subset dari verteks-verteks yang bersangkutan digabungkan menjadi satu subset yang lebih besar. Iterasi akan berlangsung hingga semua sisi terproses.

```
MST_KRUSKAL (G)
{    For setiap vertex v dalam V[G] Do
{    set S(v) ← {v} }
```

```
    Inisialisasi priority queue Q yang berisi semua edge dari G,
gunakan bobot sebagai keys.
    A ← { } // A berisi edge dari MST
    While A lebih kecil dari pada n-1 edge Do
    {     set S(v) berisi v dan S(u) berisi u }
          IF S(v) != S(u) Then
    {     Tambahkan edge (u, v) ke A
          Merge S(v) dan S(u) menjadi satu set
    }
    Return A
}
```

Algoritma Prim

Algoritma dimulai dari suatu verteks awal tertentu: bisa ditentukan oleh pemanggil atau dipilih sebarang oleh algoritma. Misalnya verteks awal tersebut adalah v.

Pada setiap iterasi terdapat kondisi di mana himpunan verteks V terbagi dua dalam:

W yaitu himpunan verteks yang sudah dievaluasi sebagai node di dalam pohon, serta (V-W) yaitu himpunan verteks yang belum dievaluasi.

Di awal algoritma W diinisialisasi berisi verteks awal v. Selanjutnya, di dalam iterasinya:

Pada setiap adjacency dari tiap verteks dalam W dengan verteks dalam (V-W) dicari sisi dengan panjang minimal. setelah diperoleh, sisi tersebut ditandai sebagai sisi yang membentuk tree dan verteks adjacent sisi tersebut dalam (VW) dipindahkan ke W (menjadi anggota W).

Jika sisi tersebut tidak ada maka proses selesai.

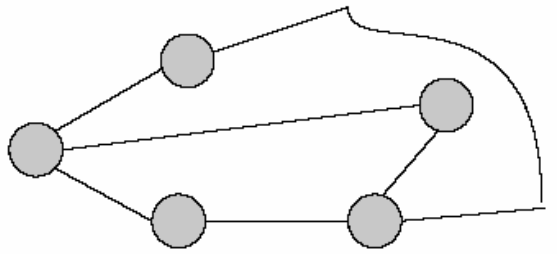
Dari contoh di atas misalnya dilakukan pencarian mulai dari verteks A (ingat tidak harus selalu dari verteks A!). Maka algoritma ini menghasilkan tahapan-tahapan iterasi pencarian sbb.:

```
MST_PRIM (G, w, v)
{    Q ← V[G]
     for setiap u dalam Q do key [u] ← ∞
     key [r] ← 0
     π[r] ← NIl
     while queue tidak kosong do
     {     u ← EXTRACT_MIN (Q)
           for setiap vertex v dalam Adj[u] do
       {   if v ada dalam Q dan w(u, v) < key [v] then
           {     π[v] ← w(u, v)
                 key [v] ← w(u, v)
           }
       }
   }
}
```

Rangkuman Struktur Data Graph

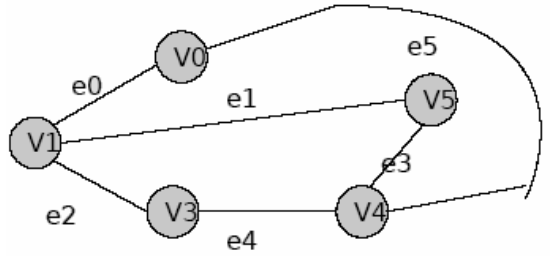
Mengenal Graph

- Terdiri dari node dan
- Terdiri dari link



Node disebut vertex

- Link disebut edge
- Informasi penting dalam graph adalah koneksi antar vertex



- Pada undirected graph, tidak terdapat directions (arah)
- Edge dari v0 ke v1 adalah sama dengan edge dari v1 ke v0
- Jika sebuah masalah dapat direpresentasikan ke dalam bentuk graph maka solusi dari masalah tersebut bisa dicari dengan bantuan graph
- Setiap vertex mewakili sebuah kondisi (state) dan edge mewakili transisi antar state

Analogi Graph dalam Kehidupan Sehari-Hari

Graph dalam kehidupan sehari-hari dapat dianalogikan sebagai suatu jaringan satu dengan jaringan lainnya yang saling terhubung. Misal seperti negara Indonesia yang memiliki banyak kota seperti: Jakarta, Bandung, Srabaya, Yogyakarta. Kota-kota itulah yang tergabung dalam negara Indonesia dan kota-kota itulah yang saling berhubungan.