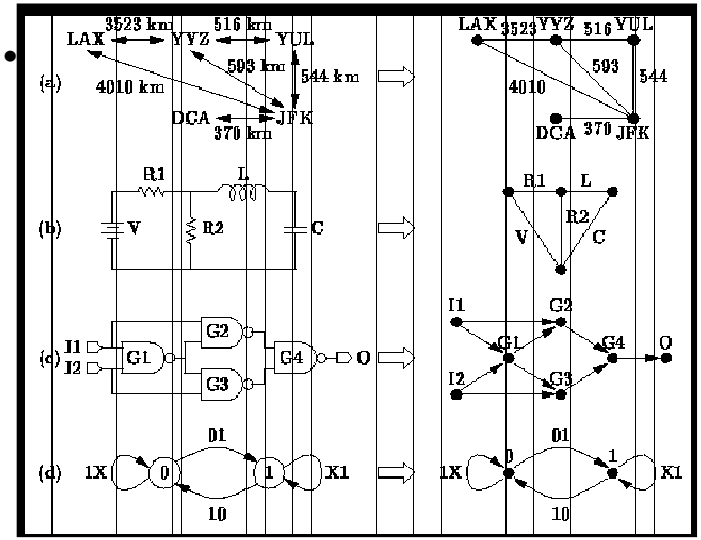
# Content

- Motivation
- Graph
  - Directed Graph
  - Undirected Graph
  - Representation
  - Implementation
- Algorithms
  - Graph Traversal
  - Shortest Path
  - Minimum Cost Spanning Trees
  - Critical Path Analysis



Motivation ...

# Graphs are everywhere






# Directed Graph

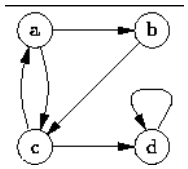
**Definition**

A directed graph, or digraph , is an ordered pair G = (V, E) with the following properties:

1. The first component, V, is a finite, non-empty set. The elements of V are called the vertices of G.
2. The second component, E, is a finite set of ordered pairs of vertices. That is, E ⊆ V x V . The elements of are called the edges of G.

**Example**

G = (V, E)
V = {a, b, c, d}
E = { (a,b), (a,c), (b,c), (c,a), (c,d), (d,d) }




**Remark**

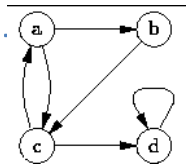
E cannot contain more than one instance of a given edge.
different then (b,a).

Eg. Consider a downwhill (a,b) cities. Although the distance is the same, the time it takes to travel a2b and b2a maybe different….



Directed Graph ...

# Head, Tail, Adjacent



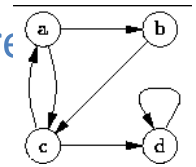

**Terminology**

Consider a directed graph G = (V, E).

- Each element of V is called a vertex or a node of G. Hence, V is the set of vertices (or nodes) of G.
- Each element of E is called an edge or an arc of G. Hence, E is the set of edges (or arcs) of G.
- An edge (v,w) ∈ E can be represented as v→w . An arrow that points from v to w is known as a directed arc . Vertex w is called the head of the arc because it is found at the arrow head. Conversely, v is called the tail of the arc. Finally, vertex w is said to be adjacent to vertex v.

Directed Graph ...

# Out-degree, In-degre




**Terminology**

Consider a directed graph G = (V, E).

- An edge e=(v,w) is said to emanate from vertex v. We use notation to denote A(v) the set of edges emanating from vertex v. That is, A(v) = {(a,b) ∈ E : a=v}
- The out-degree of a node is the number of edges emanating from that node. Therefore, the out-degree of v is |A(v)|
- An edge e=(v,w) is said to be incident on vertex w. We use notation I(w) to denote the set of edges incident on vertex w. That is, I(w) = {(a,b) ∈ E : b=w}
- The in-degree of a node is the number of edges incident on that node. Therefore, the in-degree of w is |I(v)|



Directed Graph ...

# Successor, Predecessor

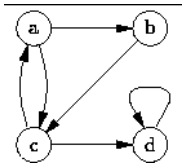
**Terminology**

Consider the path P = {$v_1$, $v_2$, ..., $v_k$} in directed graph G = (V, E).

- Vertex $v_{i+1}$ is the successor of vertex $v_i$ for $1 \le i < k$.
  Each element $v_i$ of path P (except the last) has a successor.
- Vertex $v_{i-1}$ is the predecessor of vertex $v_i$ for $1 < i \le k$.
  Each element $v_i$ of path P (except the first) has a predecessor.

**Example**

{a, b, c, d}






Directed Graph ...

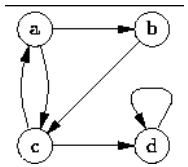
# Path

**Definition**

A path in a directed graph G = (V, E) is a non-empty sequence of vertices
P = {$v_1$, $v_2$, ..., $v_k$}
where $v_i \in V$ for $1 \le i \le k$ such that $(v_i, v_{i+1}) \in E$ for $1 \le i \le k$.
The length of path *P* is *k*-1.

**Example**

$P_1$ = (a)
$P_2$ = (b, c)
$P_3$ = (a, b, c)
$P_4$ = (a, c, a, c, a, c, a, c, a, c, a, c, a)




**Question**

What is the maximum path length?



Directed Graph ...

# Simple path, Cycle, Loop

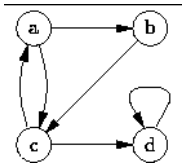
**Terminology**

Consider the path P = {$v_1$, $v_2$, ..., $v_k$} in directed graph G = (V, E).

- A path P is called a simple path if and only if $v_i \ne v_j$ for all i and j such that $1 \le i < j \le k$.
  However, it is permissible for $v_1 = v_k$.
- A cycle is a path P of non-zero length in which $v_1 = v_k$.
  The length of a cycle is just the length of the path P.
- A loop is a cycle of length one.
  That is, it is a path of the form {v, v}.
- A simple cycle is a path that is both a cycle and simple.

**Example**

{a, b, c, d}
{c, a, c, d}
{a, b, c, a}
{a, c, a, c, a}



Directed Graph ...

# Directed Acyclic Graph

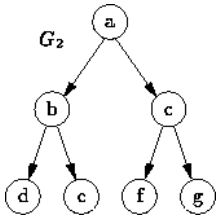
**Definition**

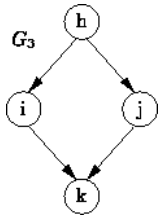
A directed, acyclic graph (DAG) is a directed graph that contains no cycles.

**Remark**

Trees $\Rightarrow$ DAG.

DAG $\Rightarrow$ Tree






# Undirected Graph

**Definition**

An undirected graph is an ordered pair G = (V, E) with the following properties:

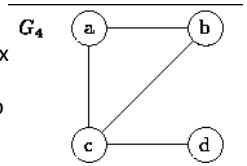
1. The first component, V, is a finite, non-empty set. The elements of V are called the vertices of G.
2. The second component, E, is a finite set of sets. Each element of E is a set that is comprised of exactly two (distinct) vertices. The elements of E are called the edges of G.

**Example**

G = ( V, E)
V = {a, b, c, d}
E = { {a,b}, {a,c}, {b,c}, {c,d} }

**Remark**

- {a,b} = {b,a} ≡ undirected *eg. full duplex between nodes
- There cannot be an edge from a node to graph






Undirected Graph ...

# Incident

**Terminology**

Consider a undirected graph G = (V, E).

- An edge e={v,w} ∈ E is said to emanate from and incident on both vertices v and w.
- The set of edges emanating from a vertex v is the set A(v) = {$(v_0,v_1) \in E : v_0 = v \vee v_1 = v$}
- The set of edges incident on a vertex w is I(w) = A(w)



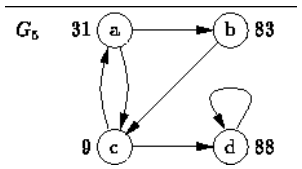
# Labeled Graph

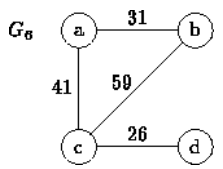
**Definition**

A graph which has been annotated in some way is called a labeled graph.

**Remark**

Both edges and vertices can be labeled



# Representation

Consider a directed graph G = (V, E).

- Since $E \subseteq V \times V$, graph G contains at most $|V|^2$ edges.
- There are $2^{|V|^2}$ possible sets of edges for a given set of vertices .





# Adjacency Matrix for DAG

Consider a directed graph G = (V, E) with V = $\{v_1, v_2, ..., v_n\}$.

- The simplest graph representation scheme uses an n x n matrix A of zeroes and ones given by

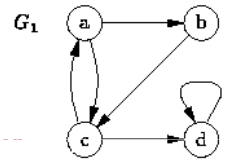
$$A_{i,j} = \begin{cases} 1 & (v_i, v_j) \in \mathcal{E}, \\ 0 & \text{otherwise.} \end{cases}$$

- The matrix A is called an adjacency matrix.





**Example**




$$A_1 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

**Rema**

- The number of ones in the adjacency matrix is equal to the number of edges in the graph
- Each one in the $i^{th}$ row corresponds to an edge that emanates from vertex $v_i$.
- Each one in the $i^{th}$ column corresponds to an edge incident on vertex $v_i$.



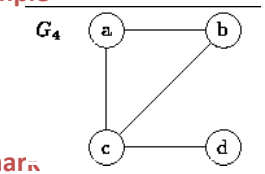


# Adjacency Matrix for undirected

Represent an undirected graph G = (V, E) with V = $\{v_1, v_2, ..., v_n\}$, using an n x n matrix *A* of zeroes and ones given by

$$A_{i,j} = \begin{cases} 1 & \{v_i, v_j\} \in \mathcal{E}, \\ 0 & \text{otherwise.} \end{cases}$$

**Example**




$$A_4 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

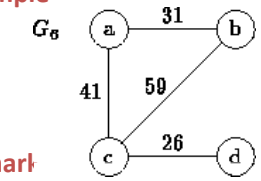
**Remark**

- Since $\{v_i, v_j\} = \{v_j, v_i\}$, matrix *A* is symmetric about the diagonal.
  That is, $a_{ij} = a_{ji}$
- All of the entries on the diagonal are zero.
  That is, $a_{ii} = 0$ for $1 \le i \le n$





**Example**




$$A_5 = \begin{bmatrix} \infty & 31 & 41 & \infty \\ 31 & \infty & 59 & \infty \\ 41 & 59 & \infty & 26 \\ \infty & \infty & 26 & \infty \end{bmatrix}$$

**Remark**

- Since $\{v_i, v_j\} = \{v_j, v_i\}$, matrix *A* is symmetric about the diagonal.
  That is, $a_{ij} = a_{ji}$
- All of the entries on the diagonal are zero.
  That is, $a_{ii} = 0$ for $1 \le i \le n$





## Complexity

- Since the adjacency matrix has $|V|^2$ entries, the amount of spaced needed to represent the edges of a graph is
  $O(|V|^2)$
  *regardless of the actual number of edges* in the graph.
- If $|E| << |V|^2$ then most of the entries are 0
- Wasteful representation !





## Sparse and Dense Graphs

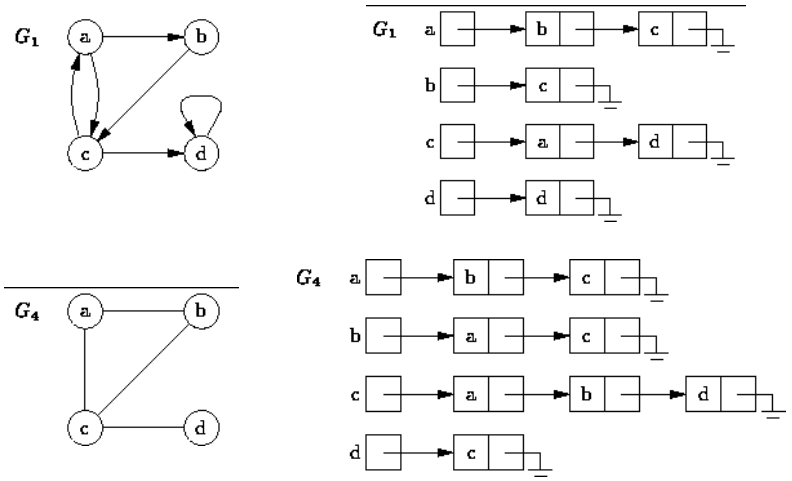
**Definition**

A sparse graph is a graph G = (V, E) in which $|E| = O(|V|)$

**Definition**

A dense graph is a graph G = (V, E) in which $|E| = \Theta(|V|^2)$

# Adjacency Lists








# Comparison of Representations

| operation | adjacency matrix | adjacency list |
|---|---|---|
| find edge (v,w) | O(1) | O( \|A(v)\| ) |
| enumerate all edges | O( $\|V\|^2$ ) | O( \|V\| + \|E\| ) |
| enumerate edges emanating from v | O( \|V\| ) | O( \|A(v)\| ) |
| enumerate edges incident on w | O( \|V\| ) | O( \|V\| + \|E\| ) |





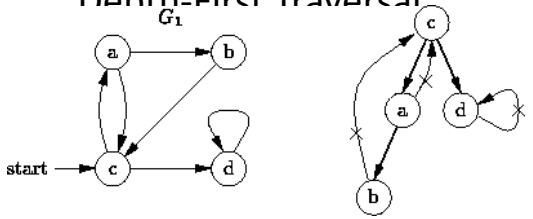
# Depth-First Traversal

- similar to depth-first tree traversal
- because there is not root,
  we must specify a starting node
- because a graph may be cyclic,
  we must prevent infinite recursion



# Graph Traversal

## Depth-First Traversal




The depth-first traversal visits the nodes in the order
c, a, b, d

Remark

- A depth-first traversal only follows edges that lead to unvisited vertices.
- if we omit the edges that are not followed, the remaining edges form a tree.
- The depth-first traversal of this tree is equivalent to the depth-first traversal of the graph

Graph Traversal

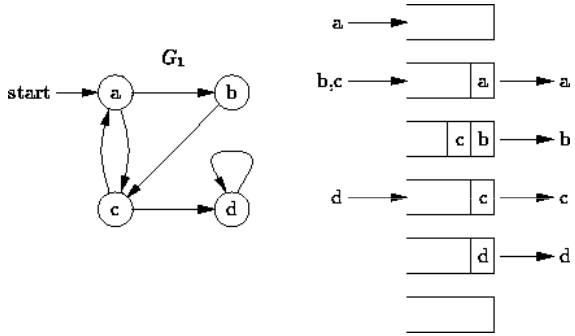
# Breadth-First Traversal

- similar to breadth-first tree traversal
- uses a queue of vertices
- because there is not root,
  we must specify a starting node
- because a graph may be cyclic,
  we must ensure that a vertex is enqueued
  only once



Graph Traversal

# Breadth-First Traversal ...




The breadth-first traversal visits the nodes in the order
a, b, c, d



Graph Traversal

# Topological Sort

- Definition
  Consider a DAG with G = (V, E).

  A topological sort of the vertices of G is a sequence $S = \{ v_1, v_2, ..., v_{|V|} \}$ in which each element of V appears exactly once.

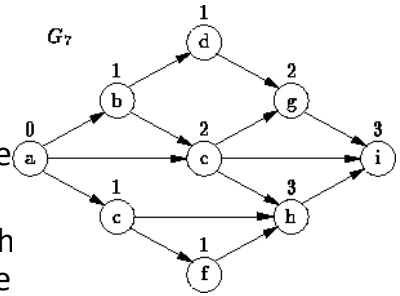
  For every pair of distinct vertices $v_i$ and $v_j$ in the sequence S, if $v_i \rightarrow v_j$ is an edge in G, i.e., $(v_i, v_j) \in E$, then i < j.



Graph Traversal

# Topological Order Traversal

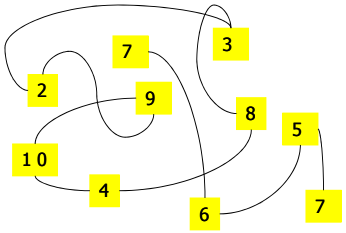
- a traversal that visits the vertices of a DAG in the order given by the topological sort
- compute the in-degrees of the vertices
- repeatedly select a vertex with zero in-degree, visit it, and the "remove it from the graph"




$S_1 = \{ a, b, c, d, e, f, g, h, i \}$
$S_2 = \{ a, c, b, f, e, d, h, g, i \}$
$S_3 = \{ a, b, d, e, g, c, f, h, i \}$
...
There are many

# Applications

- Is it connected
- Is there any cycles





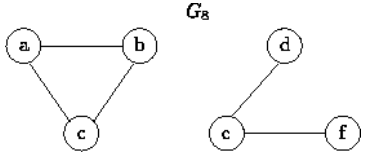



# Connectedness

Definition

An undirected graph G = (V, E) is connected if there is a path in G between every pair of vertices in V.

The connected sub-graphs of a graph are called connected components.







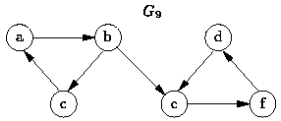

# Connectedness

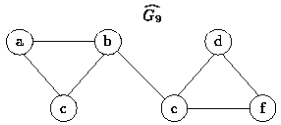
Definition

A directed graph G = (V, E) is strongly connected if there is a path in G between every pair of vertices in V.

Definition

A directed graph G = (V, E) is weakly connected if the underlying undirected graph $\breve{G}$ is connected.








# Connectedness

Algorithm

For all vertices

Start from each vertex

Traverse the graph

Mark every visited vertex

If there is an unvisited vertex, unconnected

# Cycle

## Algorithm

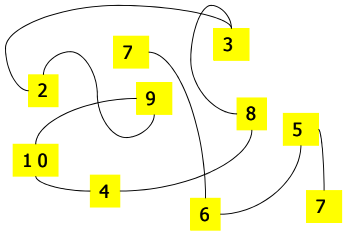
Apply topological-order traversal





# Distance

- Distance between two vertices
  - 2-9
  - 3-9
  - 7-4








# Weighted Path Length

## Definition

Consider an edge-weighted graph G = (V, E) . Let $C(v_i,v_j)$ be the weight on the edge connecting $v_i$ to $v_j$. A path in G is a non-empty sequence of vertices $P = \{v_1, v_2, ..., v_k\}$ . The weighted path length of path P is given by

$$\sum_{i=1}^{k-1} c(v_i, v_{i+1})$$

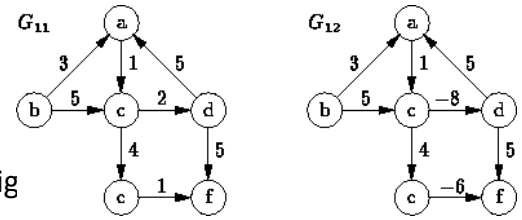




# Single-source shortest path

## Definition

Single-source shortest path problem
Given an edge-weighted graph G = (V, E) and a vertex $v_s \in V$, find the shortest weighted path from $v_s$ to every other vertex in V.




## Remark

If all weig

Shortest Path

## Special Cases non-negative weights

- if the graph is acyclic, the problem is easy
  - do a topological order traversal
- as long as all the edge weights are non-negative the shortest-path problem is well defined
  - a greedy algorithm works (Dijkstra's algorithm)



Shortest Path

## Special Cases negative weights

- if the graph has negative weights (but not negative cost cycles) a solution exists but the greedy algorithm does not work
- if the graph has a negative cost cycle, *no solution exists*



Shortest Path

## Dijkstra's Algorithm

$k_v$ indicates that the shortest path to vertex v is known. Initially false

$d_v$ the length of the shortest known path from $v_s$ to v. Initially ∞.

$p_v$ The predecessor of vertex v on the shortest path from $v_s$ to v. Initially unknown.



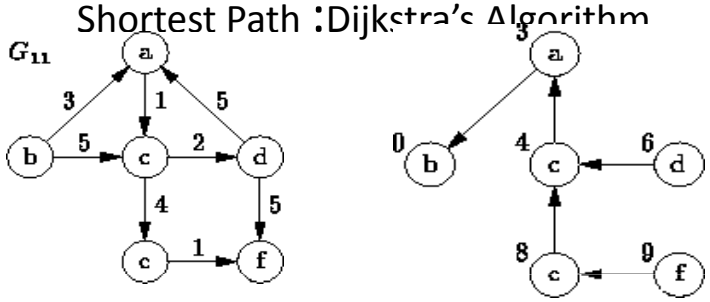
Shortest Path

## Dijkstra's Algorithm

1. From the set of vertices for with $k_v$=false, select the vertex v having the smallest tentative distance $d_v$.
2. Set $k_v$=true
3. For each vertex w adjacent to v for which $k_v \neq$true, test whether the tentative distance $d_v$ is greater than $d_v$+c(v,w). If it is, set $d_v \leftarrow d_v$+c(v,w) and set $p_v \leftarrow$ v.

In each pass exactly one vertex has its $k_v$ set to true. The algorithm terminates after |V|passes are completed at which time all the shortest paths are known.

## Shortest Path :Dijkstra's Algorithm




•http://www.lupinho.de/gishur/html/DijkstraApplet.html

| | | passes | | | | | |
|---|---|---|---|---|---|---|---|
| vertex | initially | 1 | 2 | 3 | 4 | 5 | 6 |
| $a$ | $\infty$ | 3 $b$ | √ 3 $b$ | √ 3 $b$ | √ 3 $b$ | √3 $b$ | √3 $b$ |
| $b$ | 0 — | √ 0 — | √ 0 — | √ 0 — | √ 0 — | √0 — | √0 — |
| $c$ | $\infty$ | 5 $b$ | 4 $a$ | √ 4 $a$ | √ 4 $a$ | √4 $a$ | √4 $a$ |
| $d$ | $\infty$ | $\infty$ | $\infty$ | 6 $c$ | √ 6 $c$ | √6 $c$ | √6 $c$ |
| $e$ | $\infty$ | $\infty$ | $\infty$ | 8 $c$ | 8 $c$ | √8 $c$ | √8 $c$ |
| $f$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 11 $d$ | 9 $e$ | √9 $e$ |



Shortest Path

## All-pairs source shortest path

Definition

All-pairs source shortest path problem

Given an edge-weighted graph G = (V, E), for each pair of vertices in V find the length of the shortest weighted path between the two vertices.



## Shortest Path with Floyd's Algorithm

*The idea: Determine whether a path going from Vi to Vj via Vk is shorter than the best-known path from Vi to Vk .*

$V_k$ the first k vertices in V

$P_k(v,w)$ the shortest path from vertex v to w that passes only through vertices in $V_k$

$D_k(v,w)$ the length of path $P_k(v;w)$

$$D_k(v,w) = \begin{cases} |P_k(v,w)| & P_k(v,w) \text{ exists,} \\ \infty & \text{otherwise.} \end{cases}$$



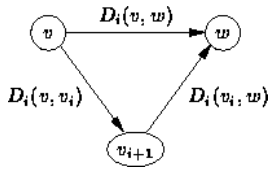
## Shortest Path: Floyd's Algorithm

$D_0(v,w)$ is the adjacency matrix

compute the sequence of matrices $D_0, D_1, \ldots, D_{|V|}$

obtain the distances in $D_{i+1}$ from those in $D_i$ by considering only the paths that pass through vertex $v_{i+1}$:

$D_{i+1}(v;w) = \min\{ D_i(v, v_{i+1})+D_i(v_{i+1}, w); D_i(v,w)\}$



# Minimum Cost Spanning Trees





## Spanning Tree

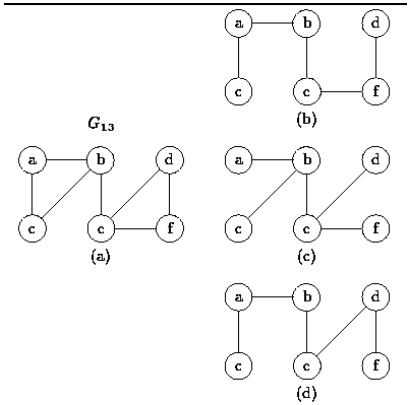
**Definition**

Consider a connected, undirected graph G=(V, E).
A spanning tree T = (V', E') of G is a subgraph of G with the following properties:

1. V' = V
2. T is connected
3. T is acyclic

**Remark**

Spanning tree is a tree.








## Total Cost

**Definition**

The total cost of an edge-weighted undirected graph is the sum of the weights on all the edges in that graph





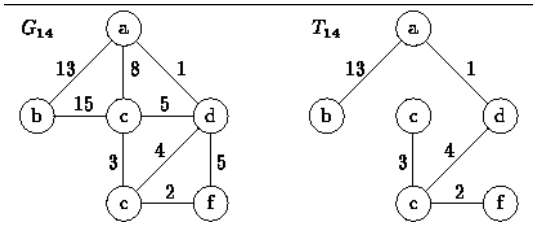
## Minimum Spanning Tree

**Definition**

Consider an edge-weighted, undirected, connected graph G=(V, E), where c(v, w) represents the weight on edge {v, w}∈ E. The minimum spanning tree of G is the spanning tree T=(V, E') that has the smallest total cost,

(a simple idea: remove the edges with large weights without loosing connection)

$$\sum_{\{v,w\}\in E'} C(v, w)$$



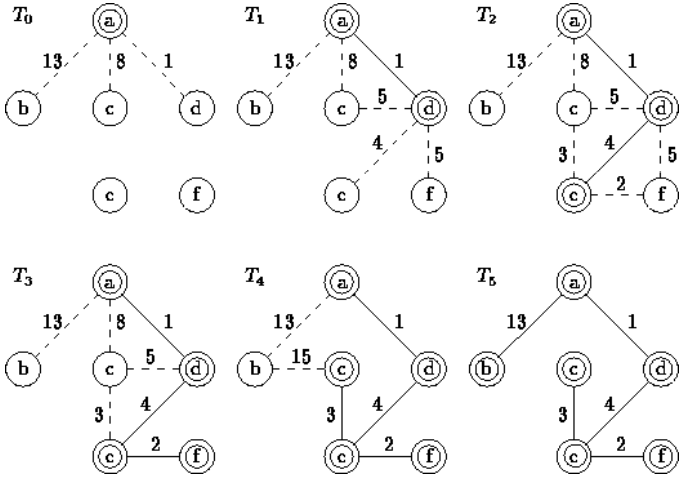
# Minimum Cost Spanning Trees: Prim

- is a minor variation of Dijkstra's algorithm for shortest path
- constructs the minimum-cost spanning tree of a graph by selecting edges from the graph one-by-one and adding those edges to the spanning tree
- at each step, select an edge with the smallest edge weight that connects the tree to a vertex not yet in the tree



## Minimum Cost Spanning Trees

# Prim's Algorithm






# Minimum Cost Spanning Tree: Kruskal

- is another greedy based approach that continually selects the edges in order of smallest weight without causing a cycle.
- Simple idea: Kruskal maintans a forest-a collection of trees.
  - Select the smallest weights first to establish forests (set of smaller trees)
  - Merge them at the end.
- Worst case running time: O(|E| Log|E|) which is dominated by heap operations. Note: since |E|=|V|^2, O(|E| Log|V|)
- Ex: Solve Figure 9.48 using Kruskal's algorithm