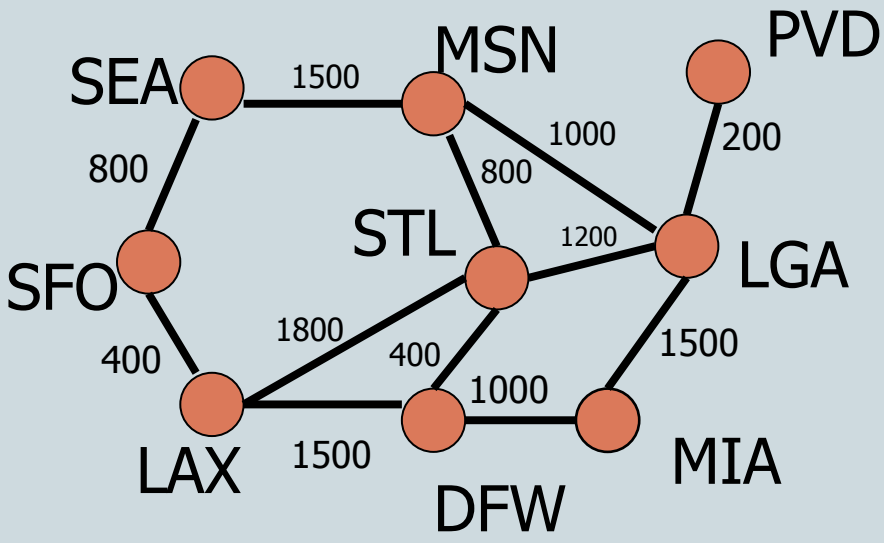
CS 521 Lecture V

DREXEL UNIVERSITY
DEPT. OF COMPUTER SCIENCE

FALL 2011

Introduction to Graph Theory

- A graph $G=(V,E)$ is pair of sets:
  - V: vertex set.
  - E: edge set.
- A graph may be weighted and its edges might be directed.

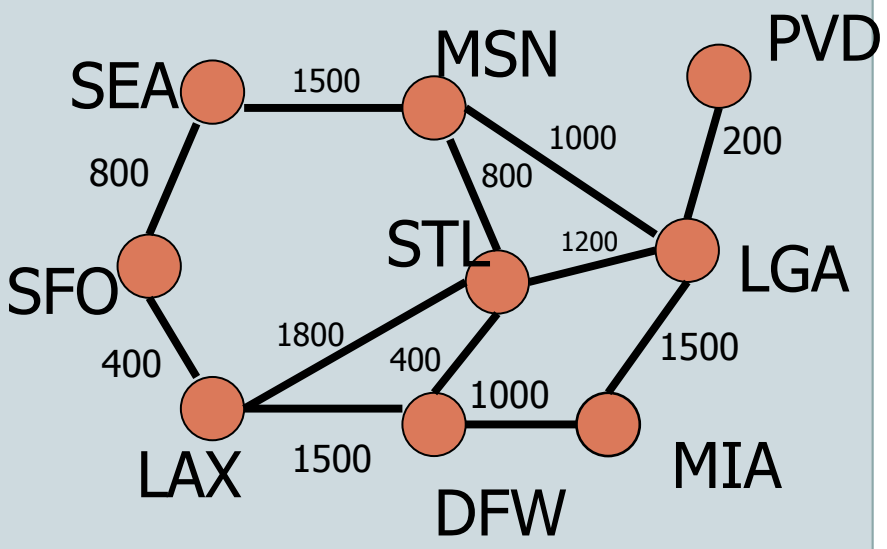


$V=$ {Sea, Sfo, Lax, Msn, Stl, Dfw, Mia, Lga, Pvd}

$E=$ {(Sea, Sfo), (Sfo,Lax), (Sea,Msn),…, (Lga, Pvd)}

Preliminaries

- Things to know:
  - Path
  - Cycle
  - Sub-graph
  - Degree of a Node
  - Maximum and Minimum Degree
  - Maximum Number of Edges in an Undirected Graph
  - Connected Components of a Graph
  - Shortest Path in a Weighted Graph
  - Tree (rooted tree)
  - Spanning Tree of a Graph:
  - Acyclic Graph
  - Bipartite Graph



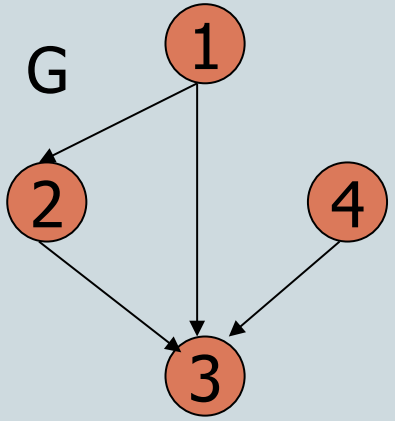
Notations:

- Given A graph $G=(V,E)$, where
  - V is its vertex set, $|V|=n$,
  - E is its edge set, with $|E|=m=O(n^2)$.
- If G is connected then for every pair of vertices u,v in G there is path connecting them.
- In an undirected graph an edge $(u,v)=(v,u)$.
- In directed graph (u,v) is different from (v,u).
- In a weighted graph there are weights associated with edges and/or vertices.
- Running time of graph algorithms are usually expressed in terms of n or m.

Graph Representation in terms of Adjacency Matrix

- The adjacency matrix of a graph G, denoted by A_G is an n by n defined as follows:

$$A_G[i,j] = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{if } (i,j) \notin E \end{cases}$$



- If G is undirected then A_G is symmetric.

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

Graph Representation in terms of Adjacency List

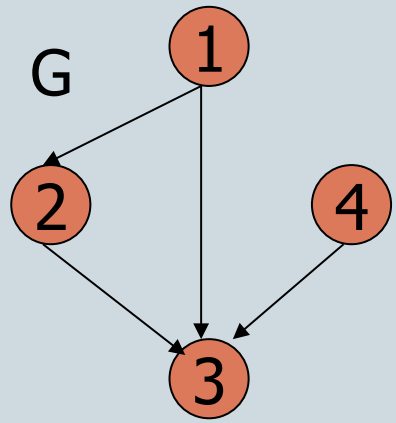
- In this method for each vertex v in V, a list $Adj[v]$ will represent those vertices adjacent to v. The size of this list is the degree of v.

$$Adj[1] = \{2,3\}$$

$$Adj[2] = \{3\}$$

$$Adj[3] = \{\}$$

$$Adj[4] = \{3\}$$



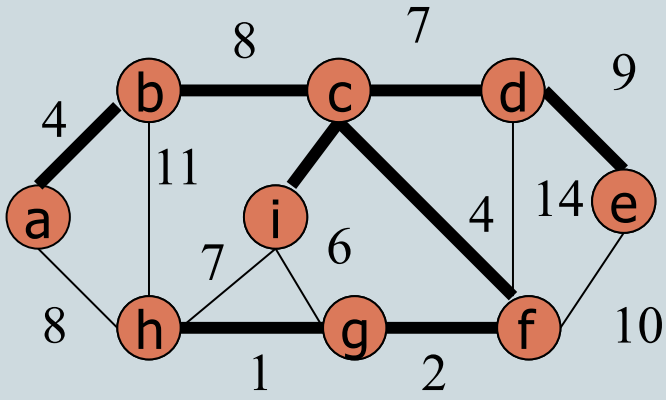
Note that:

- Number of 1's in A_G is m if G is directed; if its undirected, then number of 1's is $2m$.
- Degree of a vertex is the sum of entries in corresponding row of A_G
- If G is undirected then sum of all degree is $2m$.
- In a directed graph sum of the out degrees is equal to m.

Minimum Spanning Tree (MST) in a Weighted Graph

- Let $G=(V,E)$ be a graph on n vertices and m edges, and a weight function w on edges in E.
- A sub-graph T of G through all vertices which avoids any cycle is a spanning tree.
- The weight of T is defined as sum of the weights of all edges in T:

$$w(T) = \sum_{(u,v)\in T} w(u,v)$$



Greedy MST

- The greedy algorithm tries to solve the MST problem by making locally optimal choices:
  - Sort the edges by weight.
  - For each edge on sorted list, include that in the tree if it does not form a cycle with the edges already taken; Otherwise discard it
- The algorithm can be halted as soon as $n-1$ edges have been kept.
- Step 1. takes $O(m \log m)=O(m \log n)$.
- Today, we will see that Step 2 can be done in $O(n \log n)$ time, later we will present a linear time implementation from this step.

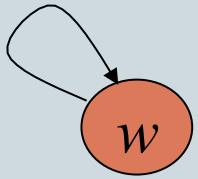
Drexel UNIVERSITY

Set Operation

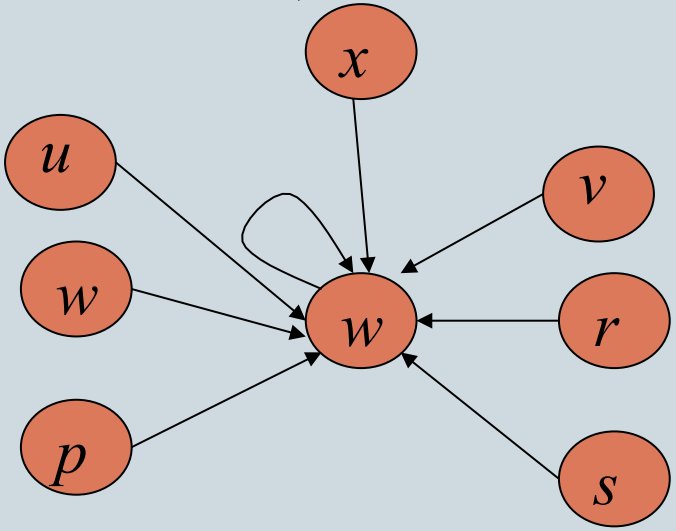
- In the proof of running time for our MST algorithm we will use the following set operations:
  - **Make-Set**(v): creates a set containing element v, $\{v\}$.
  - **Find-Set**(u): returns the set to which v belongs to.
  - **Union**(u,v): creates a set which is the union of the two sets, one containing v and one containing u.
- As an example, we can use a pointer to implement a set system:
  - **Make-Set(v)** will create a single node containing element ***v***.
  - **Find-set(u)** will return the name of the first element in the set that contains ***u***,
  - and finally the **union(u,v)** will concatenate the sets containing ***u*** and ***v***.

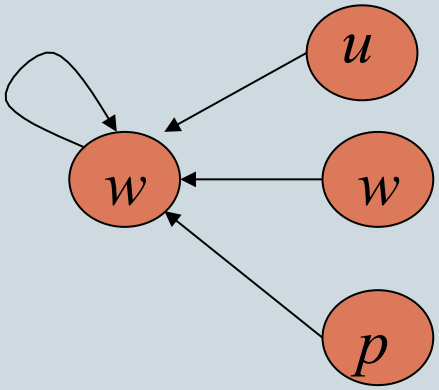
Example of a set Operations

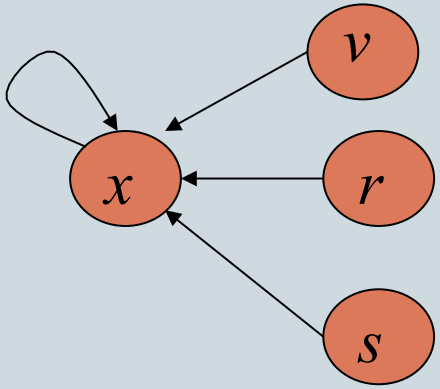
- Use linked list to show a set
- Make-Set(w):



- Find-Set(u): (will return w)
- Union(u,v):







Running Time of Set Operations

- The Make-Set and Find-Set will run in $O(1)$-time.
- How fast can we compute the union.
- Let us ask a different question.
  - Let $N=\{1,\ldots,n\}$ be a set of n integers, and let P=$\{(u,v)|$ u and v in N$\}$ be a subset of pairs from nxn.
- For $u=1$ to n Make-Set(u);

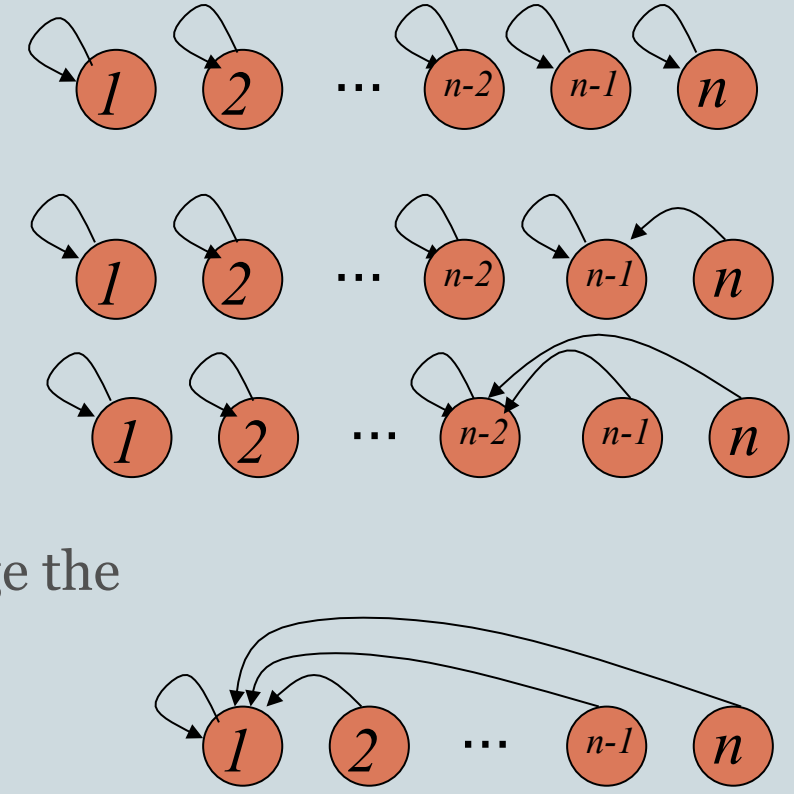
  For every pair (u,v) in P

  If Find-Set(u) ≠ Find-Set(v)

  Union(u,v)
- Question: How many times does the pointer for an element get redirected?

Union Operation

- Each merge of two sets might take linear number of pointer changes.
- We might have up $O(n^2)$ pointer changes.
- Let us keep a number associated with each set in its root, Rank(u), which tell how many elements a set has.
- When merging two lists, always change the pointers in the list with smaller rank.



Union Operation

- Now each time a pointer changes, its corresponding set doubles in the size.
- During the whole process the maximum set can become of size at most n.
- For a specific pointer this happen at most log ***n*** times:

$$2^0, 2^1, 2^2, \ldots, 2^k = m, \text{ which means } k = \log n$$

- Over all n elements, this will result in an O($\boldsymbol{n} \log \boldsymbol{n}$) number of pointer updates.

Drexel UNIVERSITY

Kruskal's MST Algorithm

- It is directly based on Generic MST.
- At each iteration, it finds a light edge, which is also safe, and adds it to an ever growing set, A, which will eventually become the MST.
- During the course of algorithm, the structure generated by algorithm is a forest.

1. $A \leftarrow \varnothing$
2. for each $v \in V_G$ do
3. Make-Set(v)
4. Sort Edges in E_G
5. for each $(u,v) \in E_G$

   (In order of increasing weights)
6. if Find-Set(u) $\neq$ Find-Set(v)
7. $A \leftarrow A \cup \{(u,v)\}$
8. Union(u,v)
9. Return A

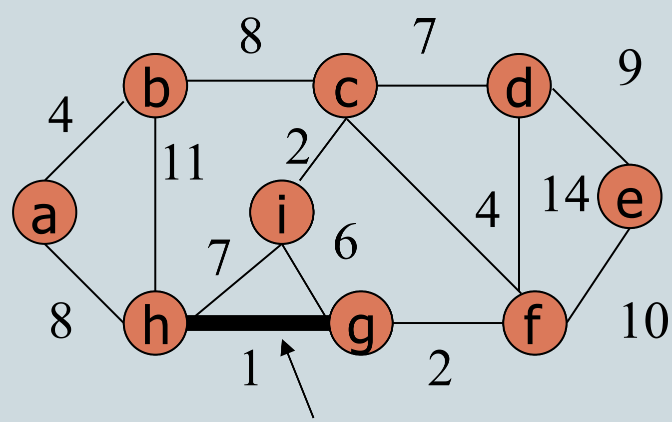
Running time of Kruskal's Algorithm

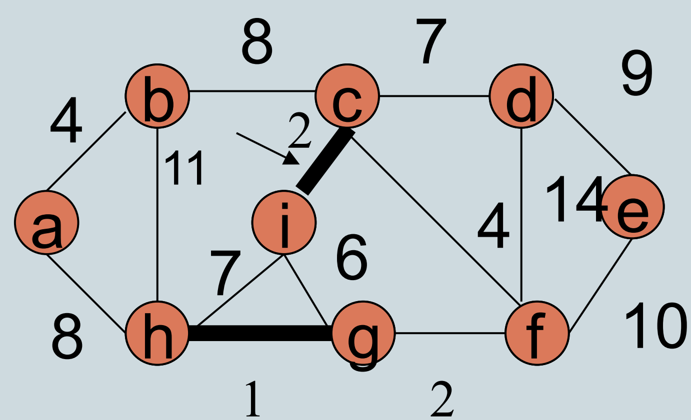
- Step 1: $O(1)$
- Steps 2,3: $O(n)$
- Step 4: $O(m \log m)$
- Steps 5-8: $O(n \log n)$

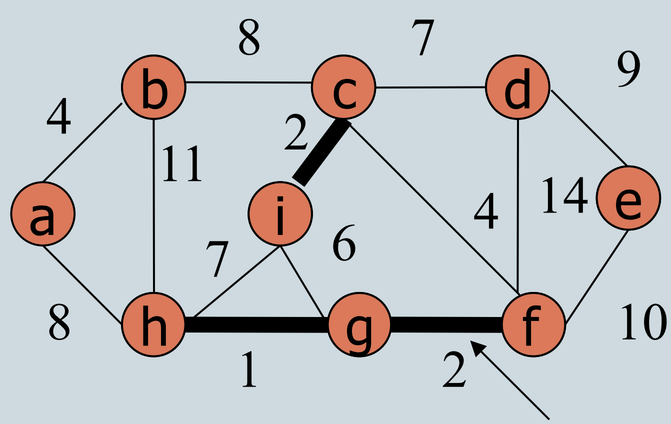
1. $A \leftarrow \emptyset$
2. for each $v \in V_G$ do
3. Make-Set(v)
4. Sort Edges in E_G
5. for each $(u,v) \in E_G$

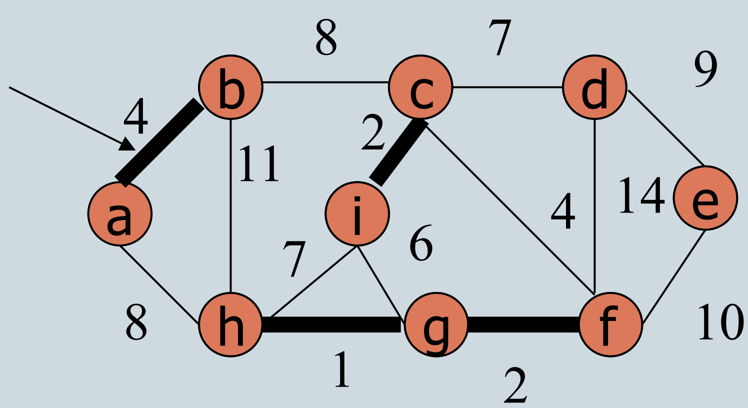
   (In order of increasing weights)
6. if Find-Set(u) $\neq$ Find-Set(v)
7. $A \leftarrow A \cup \{(u,v)\}$
8. Union(u,v)
9. Return A

Example

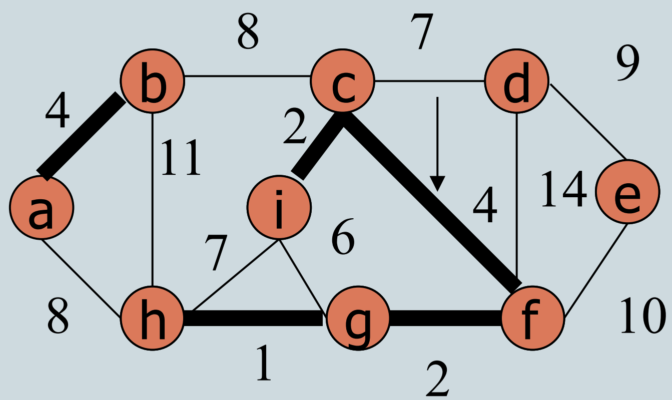

a b c d e f g h i
4 8 7 9 11 2 14 4 7 6 8 10 1 2

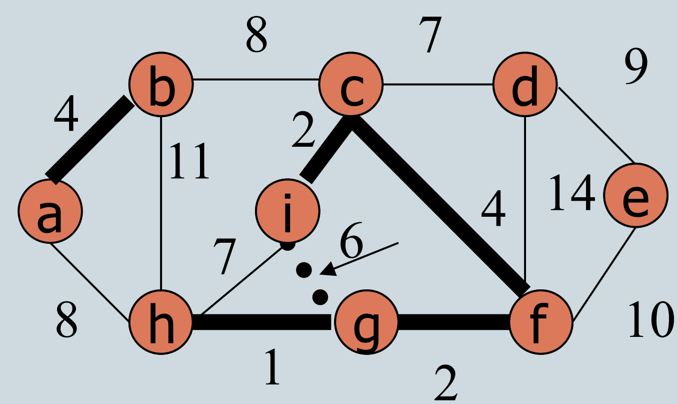

a b c d e f g h i
4 8 7 9 11 2 14 4 7 6 8 10 1 2

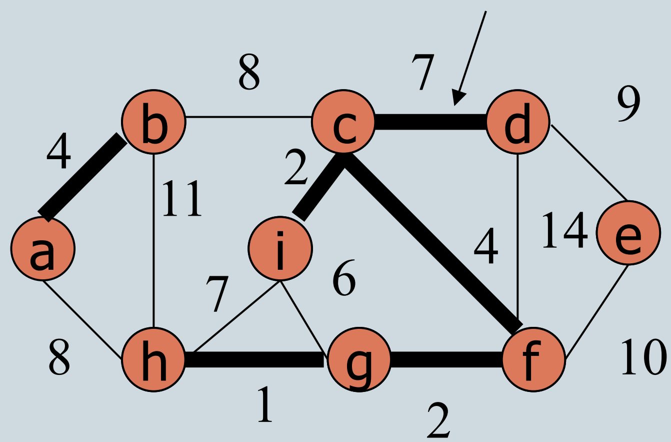

a b c d e f g h i
4 8 7 9 11 2 14 4 7 6 8 10 1 2

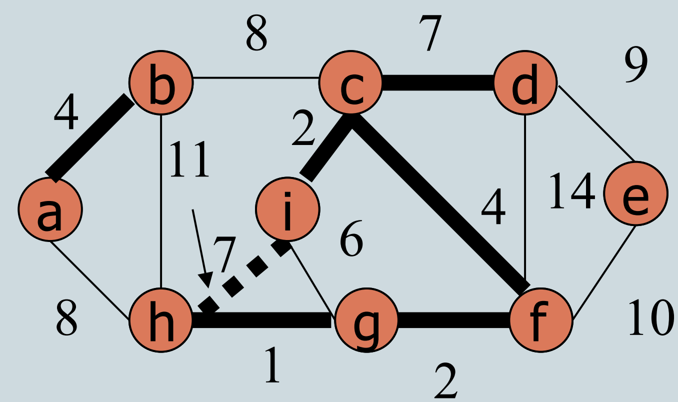

a b c d e f g h i
4 8 7 9 11 2 14 4 7 6 8 10 1 2

Example

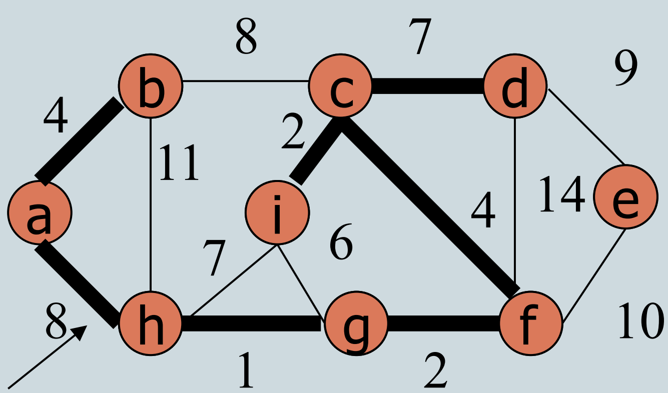

b c d
a i e
h g f
8 7 9
4 11 2 4 14
7 6
8 1 2 10

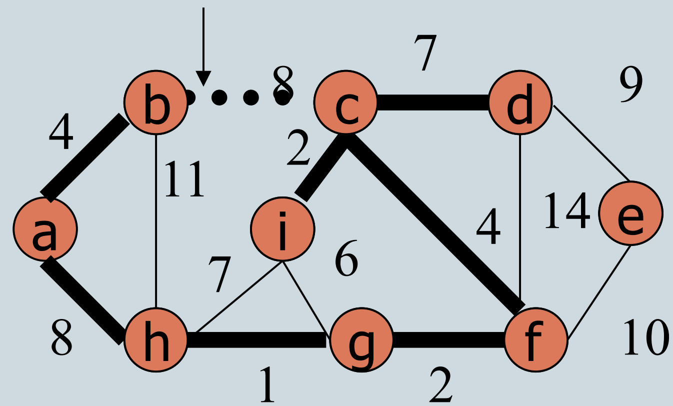

b c d
a i e
h g f
8 7 9
4 11 2 4 14
7 6
8 1 2 10

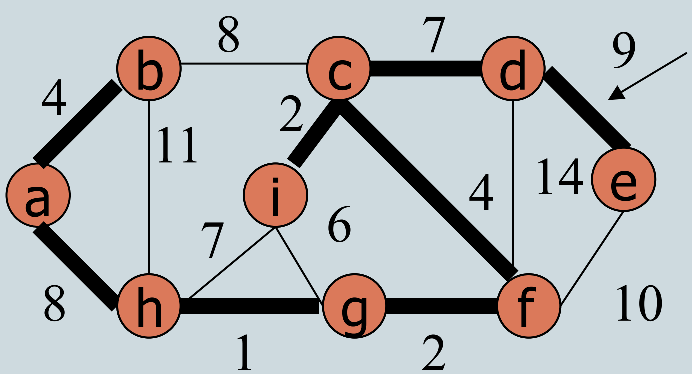

b c d
a i e
h g f
8 7 9
4 11 2 4 14
7 6
8 1 2 10

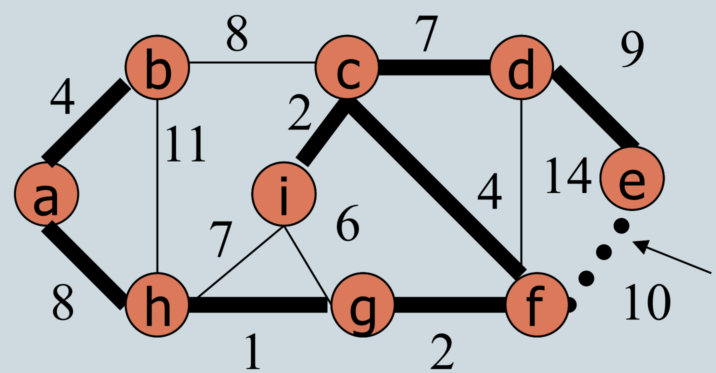

b c d
a i e
h g f
8 7 9
4 11 2 4 14
7 6
8 1 2 10

Example

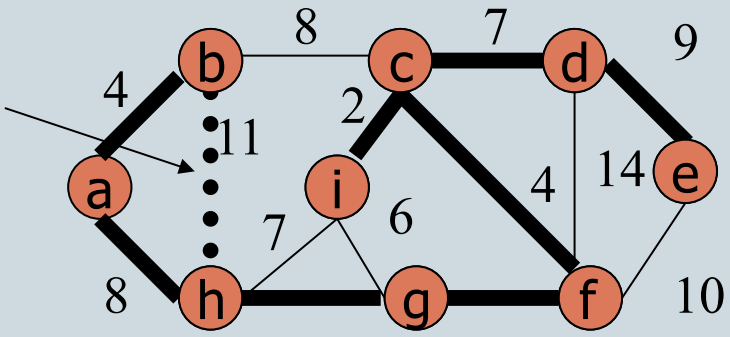

a
b
c
d
e
f
g
h
i
4
8
11
7
2
6
1
2
4
14
7
9
10
8

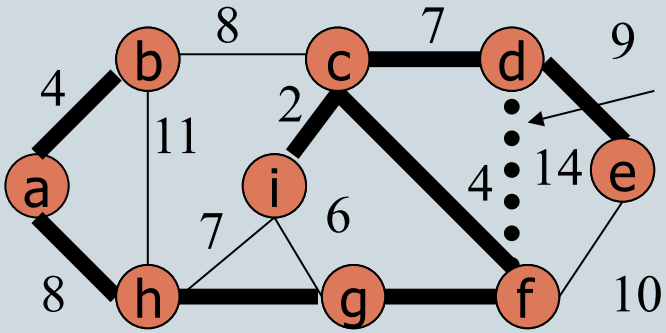

a
b
c
d
e
f
g
h
i
4
8
11
7
2
6
1
2
4
14
7
9
10
8


a
b
c
d
e
f
g
h
i
4
8
11
7
2
6
1
2
4
14
7
9
10
8


a
b
c
d
e
f
g
h
i
4
8
11
7
2
6
1
2
4
14
7
9
10
8

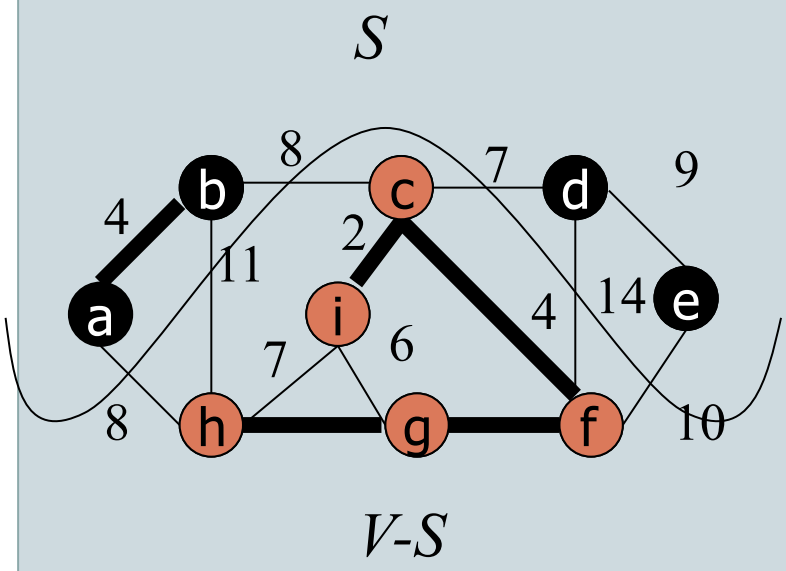
Example


a
b
c
d
e
f
g
h
i
4
8
11
8
7
9
2
7
6
4
14
8
10


a
b
c
d
e
f
g
h
i
4
8
11
8
7
9
2
7
6
4
14
8
10

Why Does the MST Work?

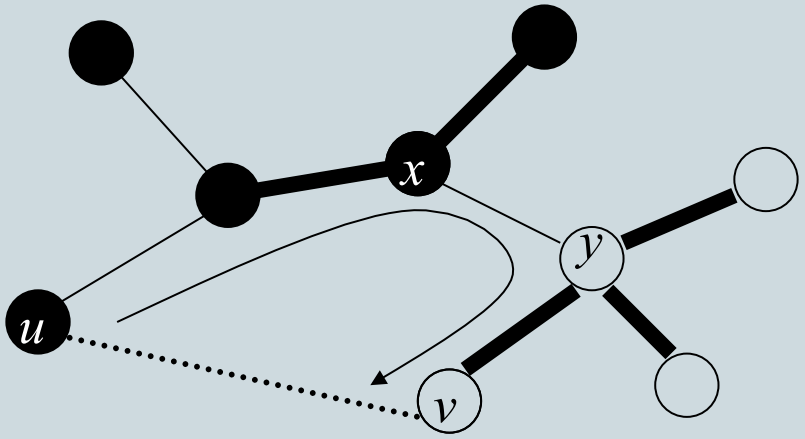
- Cuts in graphs: A cut $(S,V\text{-}S)$ of an undirected graph $G=(V,E)$ is a partition of V:



- A ***Cross edge*** is an edge with one endpoint in S and the other in $V\text{-}S$.
- We say a cut $(S,V\text{-}S)$ ***respects*** the set A if no edge of A is cross edge.
- An edge is ***a light edge*** crossing the cut $(S,V\text{-}S)$ if it has the minimum weight among all crossing edges.

Correctness:

- Let $G=(V,E)$ be a connected undirected graph with real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S,V\text{-}S)$ be any cut that respects A, and let (u,v) be a light edge crossing $(S,V\text{-}S)$. Then edge (u,v) is a safe edge for A.



Prim's MST Algorithm

- At each step the set A is one connected component.
- Start at an arbitrary vertex r (root of the tree).
- At each step adds a new vertex which is connected to A through a minimum weight edge.
- The growth starts at r and continue till all vertices are covered, each vertex $\boldsymbol{u}$ has a parent $\boldsymbol{p(u)}$, which represents its parent in the tree.
- Also, each vertex $\boldsymbol{u}$ has a $\boldsymbol{key(u)}$, which represents the cost of adding $\boldsymbol{u}$ to $\boldsymbol{A}$ at each point of algorithm.

Prim's MST

- To implement the priority queue Q, we can use a binary heap.
- The steps 1-5 can be done in $O(n)$-time.
- Step 7 take $O(\log n)$ –time.
- Step 11 can be implemented with decrease key which takes $O(\log n)$-time.
- Since there are at most $m=|E|$ elements in all $Adj[]$ list for all elements in Q, the the algorithm take

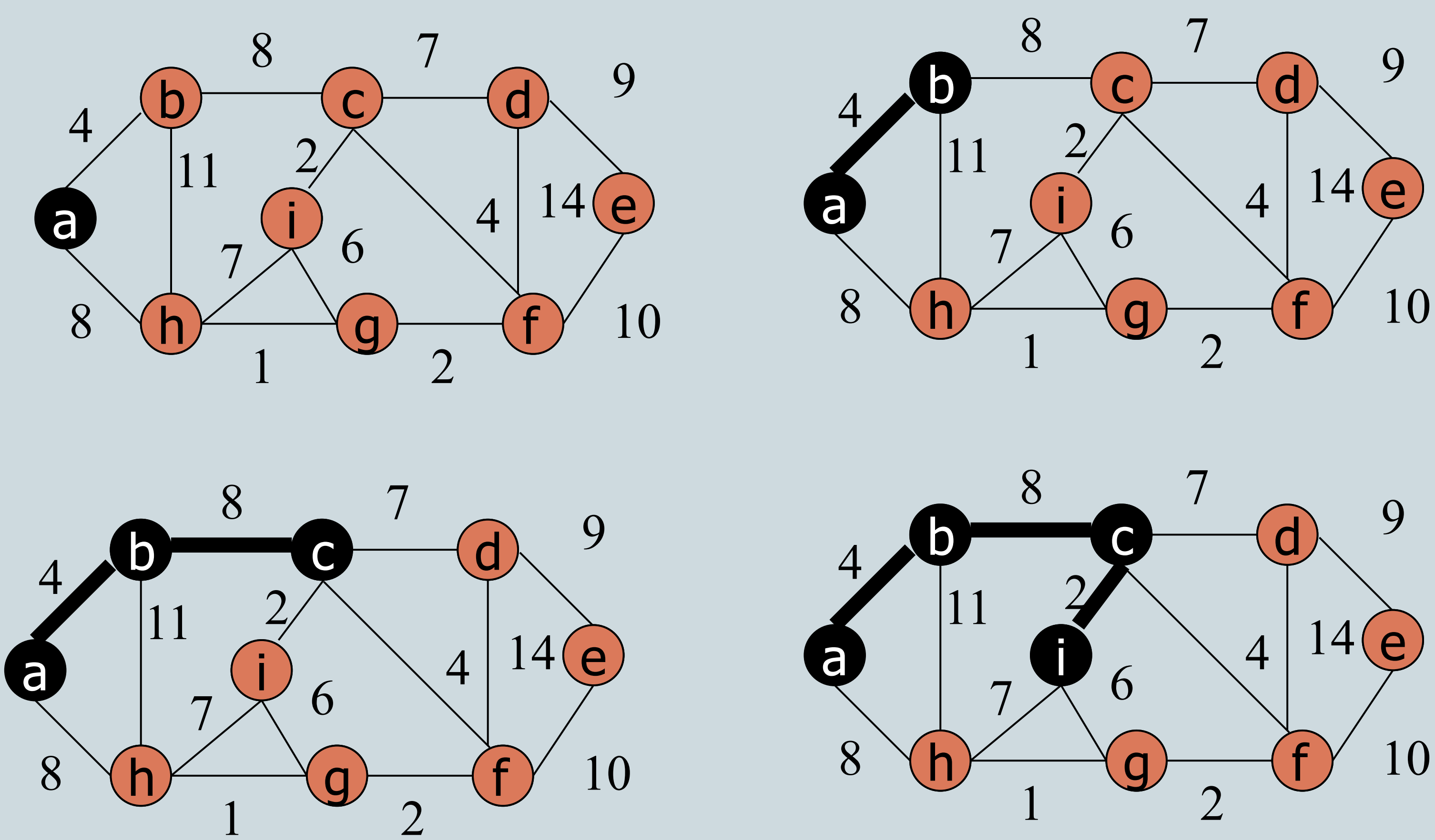
  $O(n \log n + m \log n) = O(m \log n)$

$MST - Prim(G, W, r)$

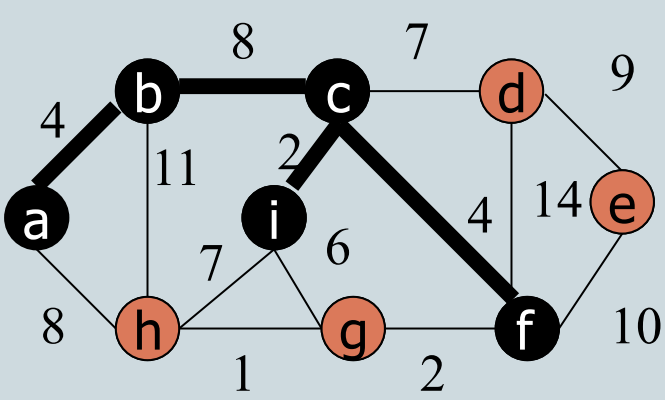
1. $Q \leftarrow V_G$
2. For each $u \in Q$
3. do $key[u] = \infty$
4. $key[r] \leftarrow 0$
5. $p(r) = \text{NIL}$
6. While $Q \neq \varnothing$ do
7. $u \leftarrow \text{Extract-Min}(Q)$
8. For each $v \in Adj[u]$ do
9. If $v \in Q$ and $w(u, v) < key[v]$
10. then $p(v) \leftarrow u$
11. $key[v] \leftarrow w(u, v)$

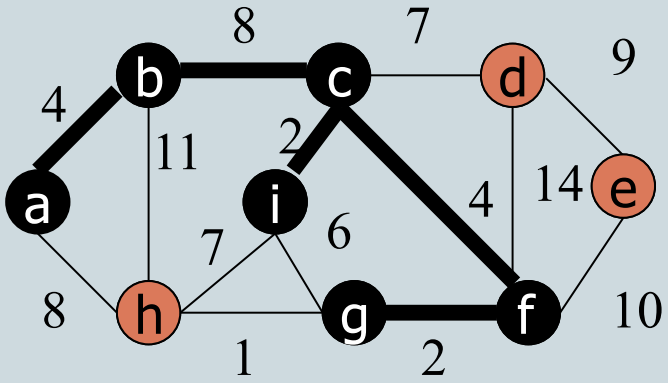
Drexel UNIVERSITY

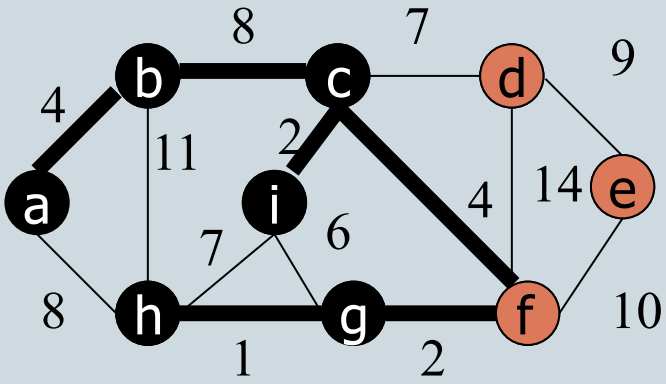
Example

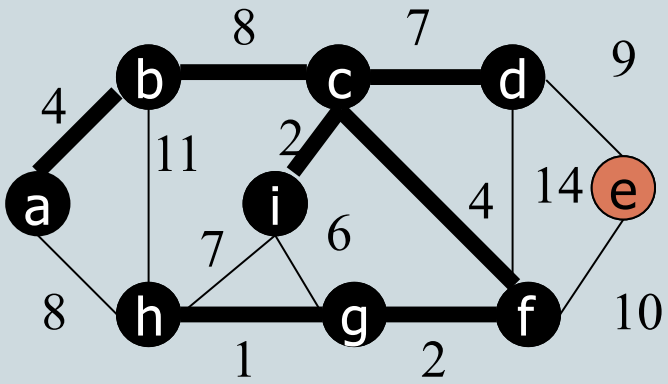

8 7 9
4 b c d
11 2 14 e
a i 4
7 6
8 h g f 10
1 2
8 7 9
4 b c d
11 2 14 e
a i 4
7 6
8 h g f 10
1 2
8 7 9
4 b c d
11 2 14 e
a i 4
7 6
8 h g f 10
1 2
8 7 9
4 b c d
11 2 14 e
a i 4
7 6
8 h g f 10
1 2

Example

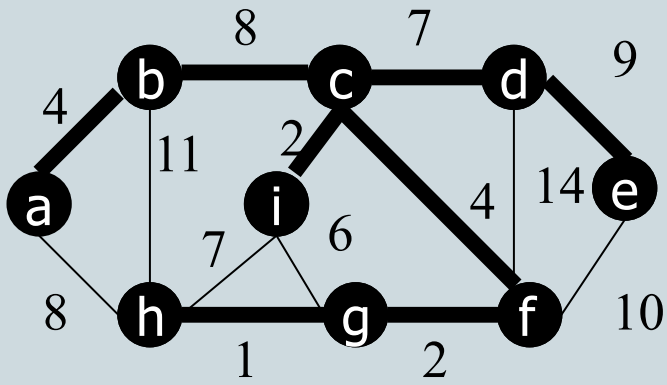

a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
7
6
4
14
9
10
8
1
2


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
7
6
4
14
9
10
1
2


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
7
6
4
14
9
10
1
2


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
7
6
4
14
9
10
1
2

Example



Correctness

- The theorem 23.1 holds here.
  - At each step the cut is $(Q, V-Q)$.
  - Extract-min will return the light edge of the current cut.
  - The $key[v]=w(u,v)$ will ensure that the cost of adding all new vertices in next iteration is up to date.
  - The algorithm stops after Q is empty, which happens after n iterations.