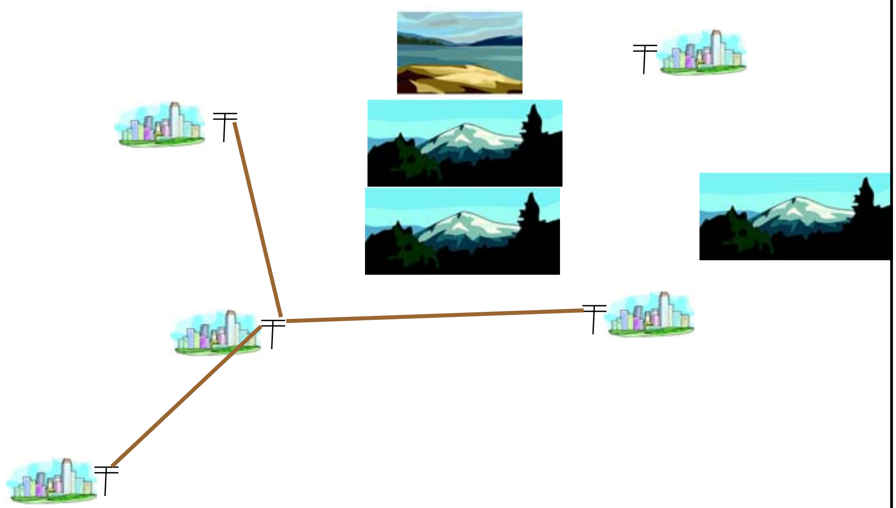
Minimum Spanning Trees

- In this section of notes you will learn two algorithms for creating a connected graph at minimum cost as well as a method for ordering a graph

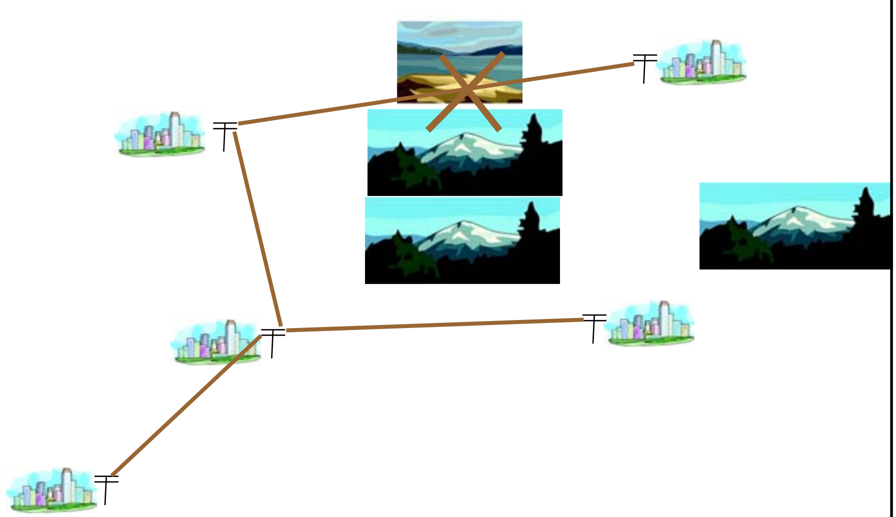
Minimum Spanning Trees

- Applies to weighted, undirected and connected graph
- Create the minimum number of edges/arcs so that all nodes/vertices are connected

Example Of A Problem Dealing With Minimum Spanning Trees

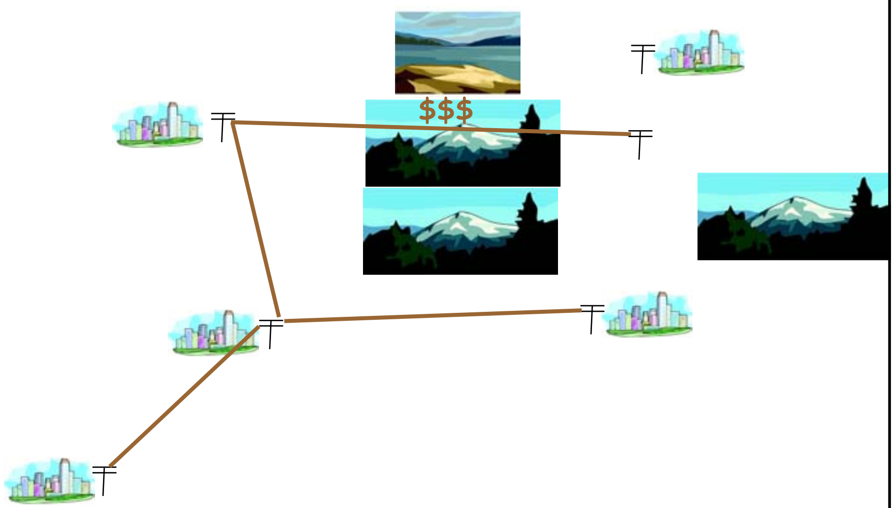


Example Of A Problem Dealing With Minimum Spanning Trees

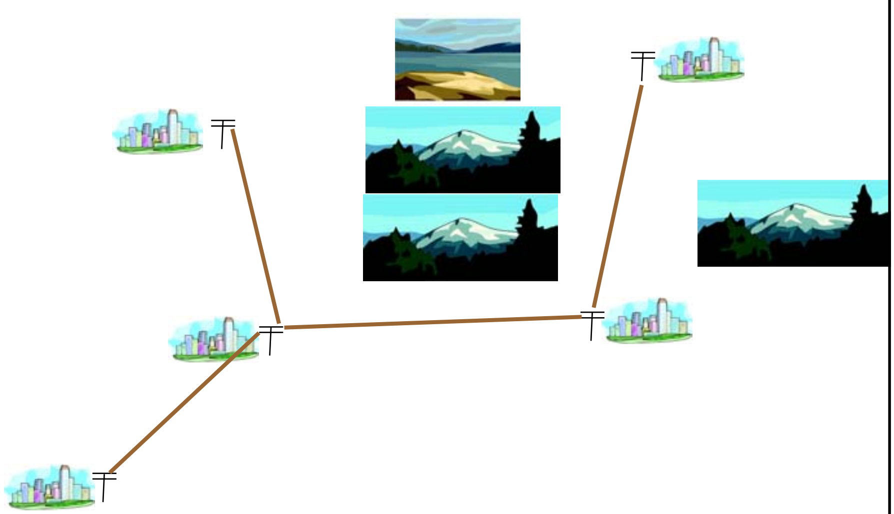


James Tam

Example Of A Problem Dealing With Minimum Spanning Trees



Example Of A Problem Dealing With Minimum Spanning Trees



Algorithms For Determining The Minimum Spanning Tree

- Prim's Algorithm
- Kruskal's Algorithm

Prim's Algorithm

```
primsAlgorithm (Graph g, Tree t, Node start)
{
    PriorityQueue nodesLeft = new PriorityQueue();
    Node temp;
    Node neighbor;
    for (int i = 1; i <= g.noNodes (); i++)
    {
        g[i].setWeight = ∞;
        g[i].setParent (null);
        nodesLeft.add (g[i]);
    }
    start.setWeight (0);
```

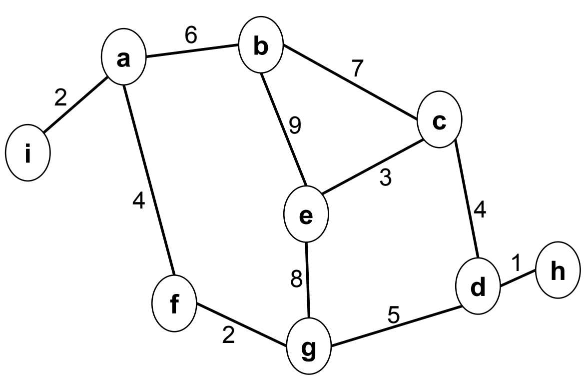
Pseudo code is based roughly on the algorithm provided by Matthew A. Becker
http://www.andrew.cmu.edu/user/mbecker/

Prim's Algorithm (2)

```
while (nodesLeft.isEmpty() == false)
{
    temp = nodesLeft.dequeueMinWeightNode ();
    t.add (temp);
    for each node "neighbor" which is adjacent to temp and is in nodesLeft
    {
        if (weight (temp, neighbor) < neighbor.getWeight())
        {
            neighbor.setParent (temp);
            neighbor.setWeight (weight (temp, neighbor));
        }
    }
}
}
```

Pseudo code based roughly on the algorithm provided by Matthew A. Becker
http://www.andrew.cmu.edu/user/mbecker/

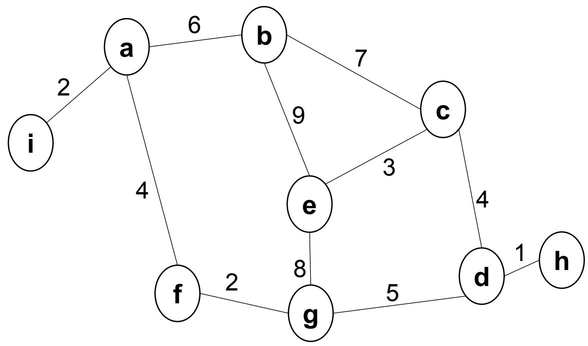
Example: Finding The Minimum Spanning Tree In A Graph (Prim's Algorithm)



Initialized Values For The Graph And Queue

Graph

Tree



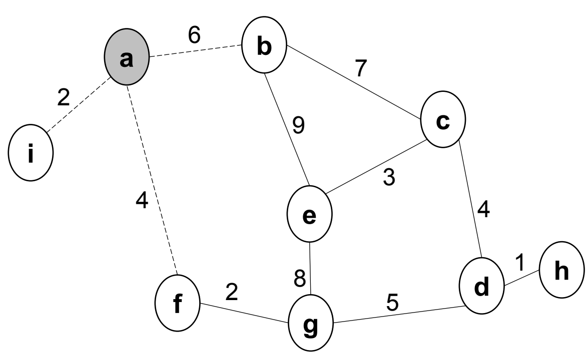
Queue

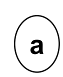
Weight	**0**	∞	∞	∞	∞	∞	∞	∞	∞
Node	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
Predecessor	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Mark “A” As Visited, Update Neighbors

Graph

Tree



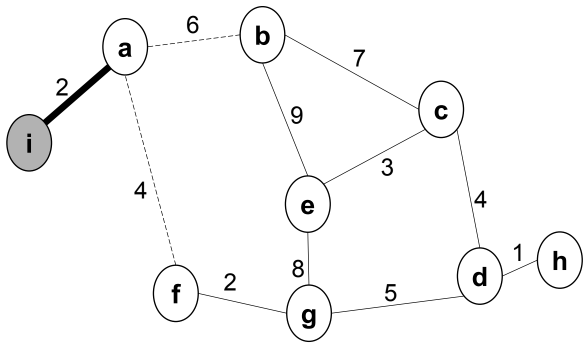


Queue

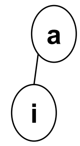
Weight	0	**6**	∞	∞	∞	**4**	∞	∞	**2**
Node	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
Predecessor	0	**A**	**0**	**0**	**0**	**A**	**0**	**0**	**A**

Mark Node "I" As Visited And Include The Edge (A,I)

Graph



Tree

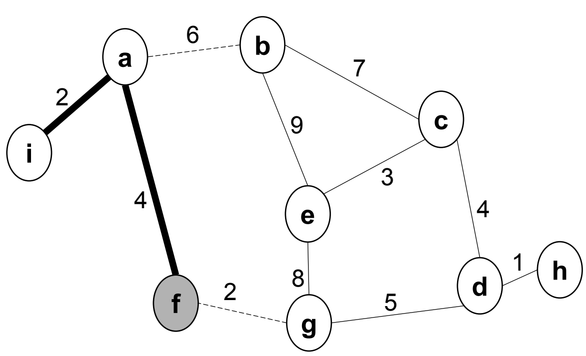


Queue

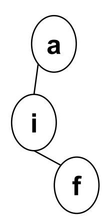
Weight	0	**6**	∞	∞	∞	**4**	∞	∞	2
Node	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	I
Predecessor	0	**A**	**0**	**0**	**0**	**A**	**0**	**0**	A

Mark Node "F" As Visisted And Include The Edge (A,F)

Graph



Tree

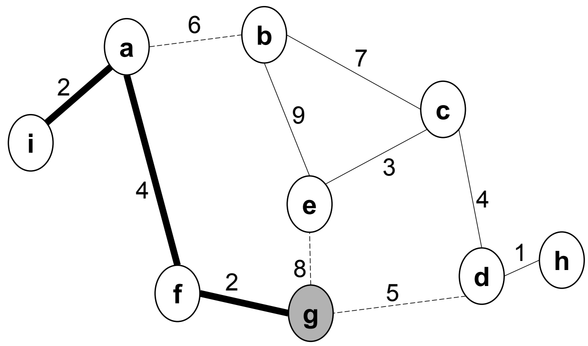


Queue

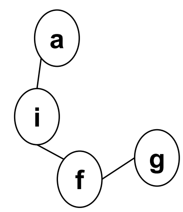
Weight	0	**6**	∞	∞	∞	4	**2**	∞	2
Node	A	**B**	**C**	**D**	**E**	F	**G**	**H**	I
Predecessor	0	**A**	**0**	**0**	**0**	A	**F**	**0**	A

Mark Node "G" As Visisted And Include The Edge (F,G)

Graph



Tree

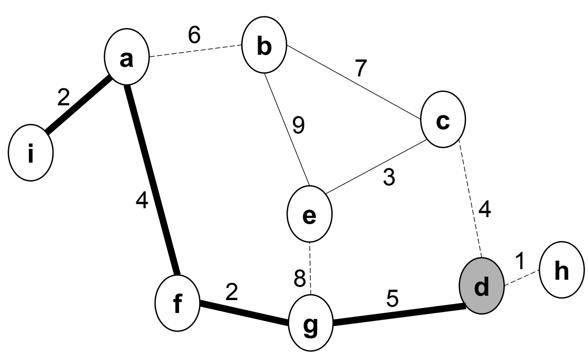


Queue

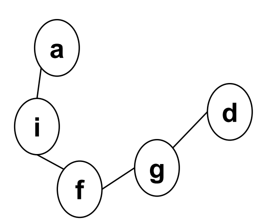
Weight	0	**6**	∞	**5**	**8**	4	2	∞	2
Node	A	**B**	**C**	**D**	**E**	F	G	**H**	I
Predecessor	0	**A**	**0**	**G**	**G**	A	F	**0**	A

Mark Node "D" As Visisted And Include The Edge (D,G)

Graph



Tree

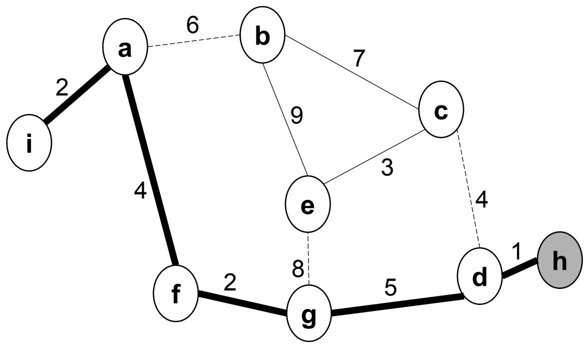


Queue

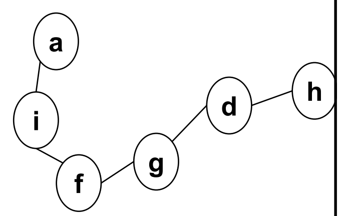
Weight	0	**6**	**4**	5	**8**	4	2	**1**	2
Node	A	**B**	**C**	D	**E**	F	G	**H**	I
Predecessor	0	**A**	**D**	G	**G**	A	F	**D**	A

Mark Node "H" As Visisted And Include The Edge (D,H)

Graph



Tree

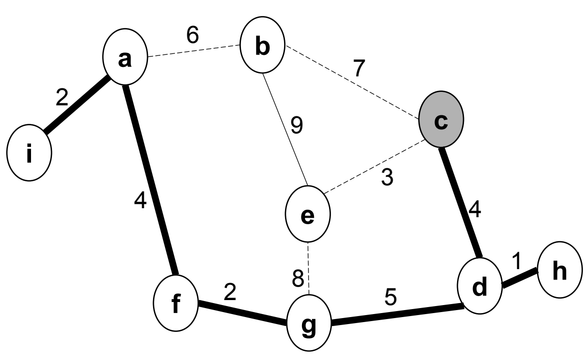


Queue

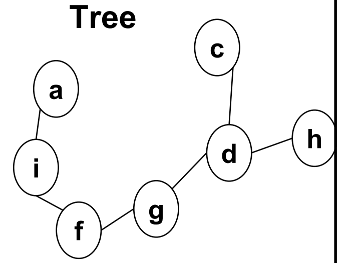
Weight	0	**6**	**4**	5	**8**	4	2	1	2
Node	A	**B**	**C**	D	**E**	F	G	H	I
Predecessor	0	**A**	**D**	G	**G**	A	F	D	A

Mark Node "C" As Visisted And Include The Edge (C,D)

Graph



Tree

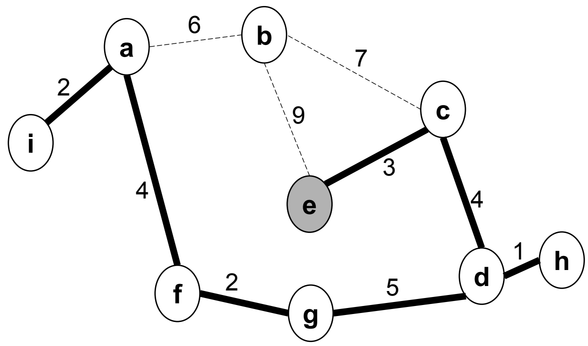


Queue

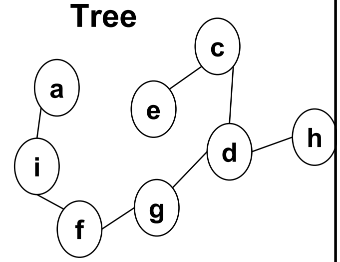
Weight	0	**6**	4	5	**3**	4	2	1	2
Node	A	**B**	C	D	**E**	F	G	H	I
Predecessor	0	**A**	D	G	**C**	A	F	D	A

Mark Node “E” As Visisted And Include The Edge (C,E)

Graph



Tree

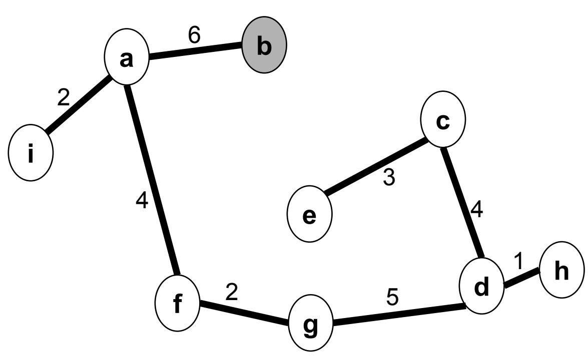


Queue

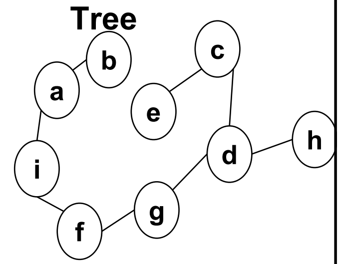
Weight	0	**6**	4	5	3	4	2	1	2
Node	A	**B**	C	D	E	F	G	H	I
Predecessor	0	**A**	D	G	C	A	F	D	A

Mark Node “B” As Visisted And Include The Edge (A,B)

Graph



Tree

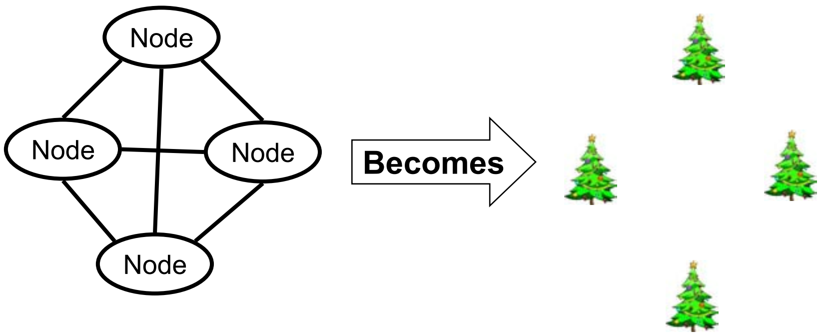


Queue

Weight	0	6	4	5	3	4	2	1	2
Node	A	B	C	D	E	F	G	H	I
Predecessor	0	A	D	G	C	A	F	D	A

Kruskal's Algorithm: Description

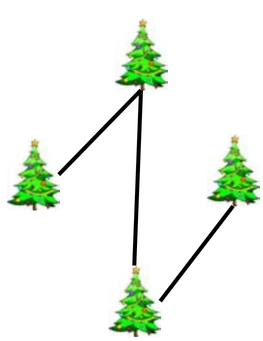
•Start out with a "forest" of unconnected trees ("extract" the nodes from the graph)



Kruskal's Algorithm: Description

•Make the connections between the separate trees (add edges) which have the lowest cost.

•Connecting two separate nodes (two separate trees) merges them into one tree



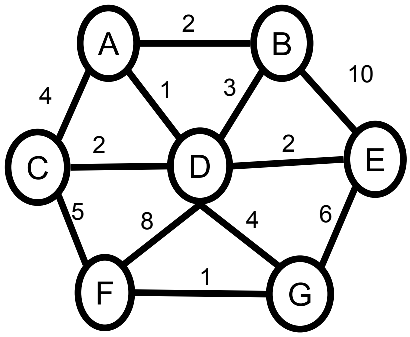
Kruskal's Algorithm

```
public Set kruskal (Graph g)
{
    Set combinedSet = new Set ();
    int edgesAccepted = 0;
    PriorityQueue edgesLeft = g.sortEdges ();
    Edge usedEdge;
    Node sourceNode;
    Node destinationNode;
```

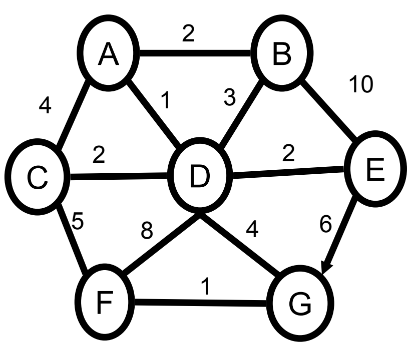

Kruskal's Algorithm (2)

```
    while (edgesAccepted < (g.getNumberNodes() - 1))
    {
        usedEdge = edgesLeft.dequeueMin ();
        sourceNode = usedEdge.getSourceNode ();
        destinationNode = usedEdge.getDestinationNode ();
        if ((sourceNode != destinationNode) &&
            (notCycle(sourceNode,destinationNode,combinedSet) == true)
        {
            edgesAccepted++;
            combinedSet.union (sourceNode, destinationNode);
        }
    }
    return combinedSet;
}
```

Example Trace Of Kruskal's Algorithm: Original Graph

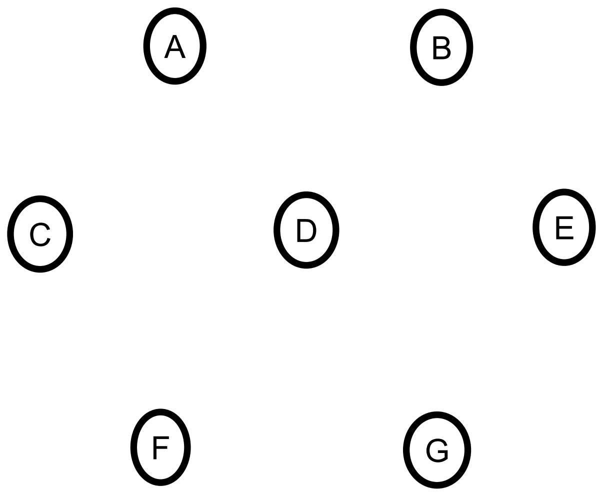


Example Trace Of Kruskal's Algorithm: Sorted Edges In Priority Queue



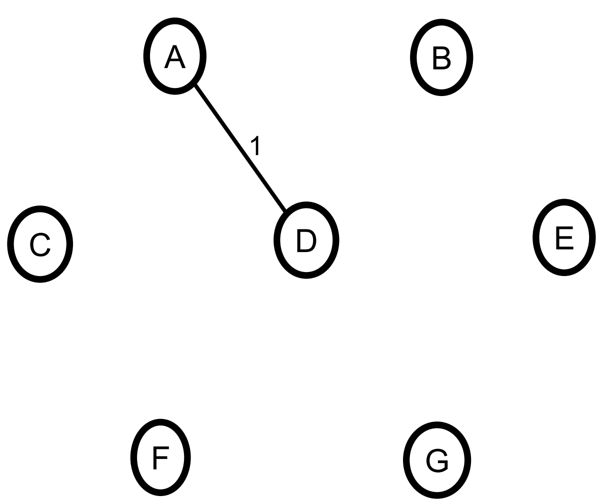
Priority queue
A - D: Weight = 1
F - G: Weight = 1
A - B: Weight = 2
C - D: Weight = 2
D - E: Weight = 2
B - D: Weight = 3
A - C: Weight = 4
D - G: Weight = 4
C – F: Weight = 5
E – G: Weight = 6
D – F: Weight = 8
B – E: Weight = 10

Example Trace Of Kruskal's Algorithm: Forest Of Nodes



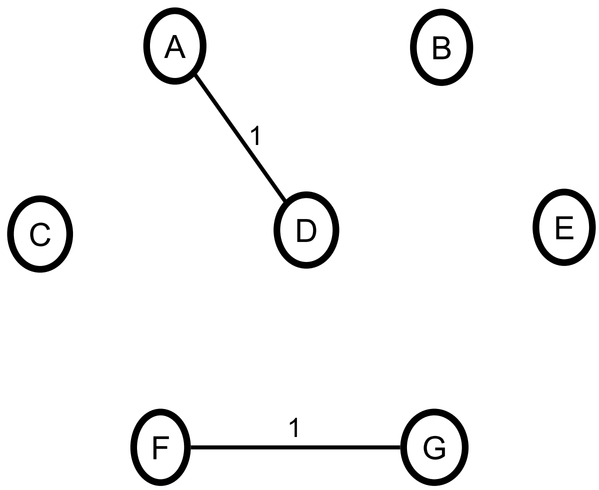
Priority queue
A - D: Weight = 1
F - G: Weight = 1
A - B: Weight = 2
C - D: Weight = 2
D - E: Weight = 2
B - D: Weight = 3
A - C: Weight = 4
D - G: Weight = 4
C – F: Weight = 5
E – G: Weight = 6
D – F: Weight = 8
B – E: Weight = 10

Example Trace Of Kruskal's Algorithm: First Edge Added (A-D)



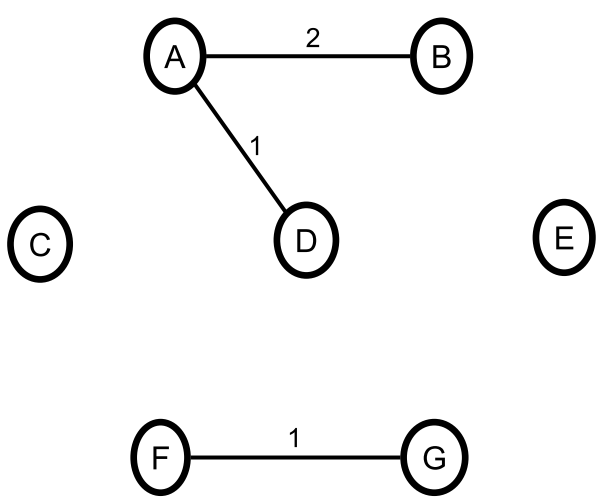
Priority queue
A - D: Weight = 1
F - G: Weight = 1
A - B: Weight = 2
C - D: Weight = 2
D - E: Weight = 2
B - D: Weight = 3
A - C: Weight = 4
D - G: Weight = 4
C – F: Weight = 5
E – G: Weight = 6
D – F: Weight = 8
B – E: Weight = 10

Example Trace Of Kruskal's Algorithm: Second Edge Added (F-G)



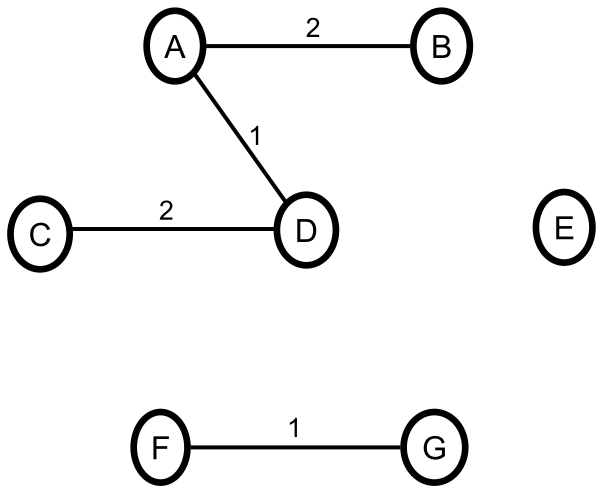
Priority queue
A - D: Weight = 1
F - G: Weight = 1
A - B: Weight = 2
C - D: Weight = 2
D - E: Weight = 2
B - D: Weight = 3
A - C: Weight = 4
D - G: Weight = 4
C – F: Weight = 5
E – G: Weight = 6
D – F: Weight = 8
B – E: Weight = 10

Example Trace Of Kruskal's Algorithm: Third Edge Added (A-B)



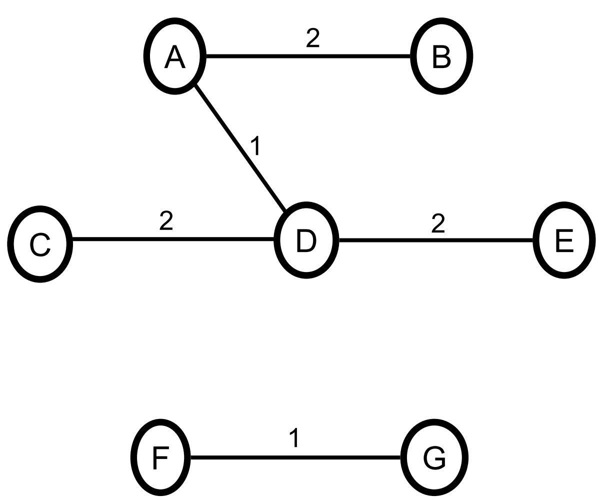
Priority queue
A - D: Weight = 1
F - G: Weight = 1
A - B: Weight = 2
C - D: Weight = 2
D - E: Weight = 2
B - D: Weight = 3
A - C: Weight = 4
D - G: Weight = 4
C – F: Weight = 5
E – G: Weight = 6
D – F: Weight = 8
B – E: Weight = 10

Example Trace Of Kruskal's Algorithm: Fourth Edge Added (C-D)



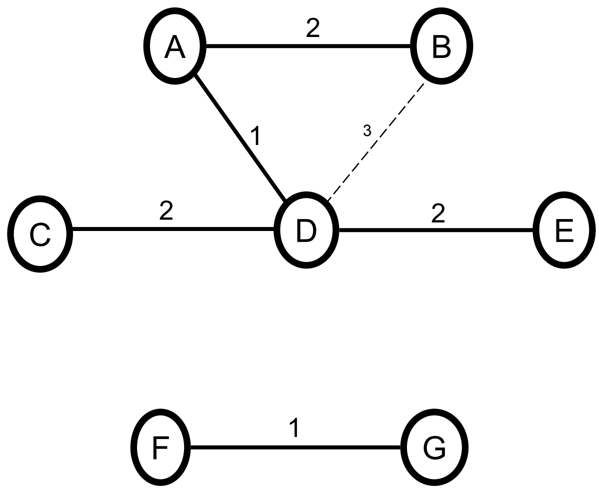
Priority queue
A - D: Weight = 1
F - G: Weight = 1
A - B: Weight = 2
C - D: Weight = 2
D - E: Weight = 2
B - D: Weight = 3
A - C: Weight = 4
D - G: Weight = 4
C – F: Weight = 5
E – G: Weight = 6
D – F: Weight = 8
B – E: Weight = 10

Example Trace Of Kruskal's Algorithm: Add Fifth Edge (D-E)



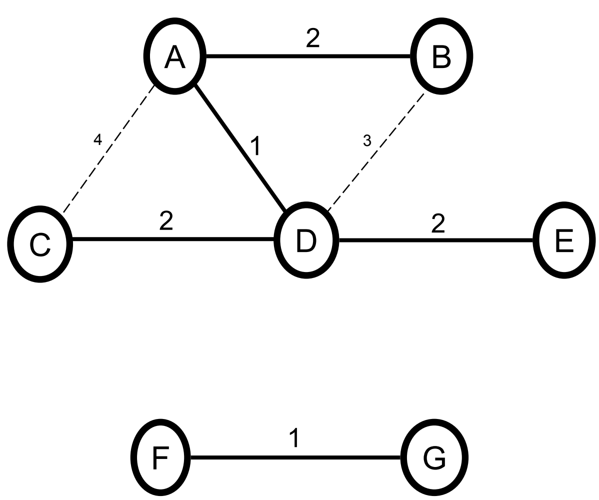
Priority queue
A - D: Weight = 1
F - G: Weight = 1
A - B: Weight = 2
C - D: Weight = 2
D - E: Weight = 2
B - D: Weight = 3
A - C: Weight = 4
D - G: Weight = 4
C – F: Weight = 5
E – G: Weight = 6
D – F: Weight = 8
B – E: Weight = 10

Example Trace Of Kruskal’s Algorithm: Don’t Use Edge (B-D)



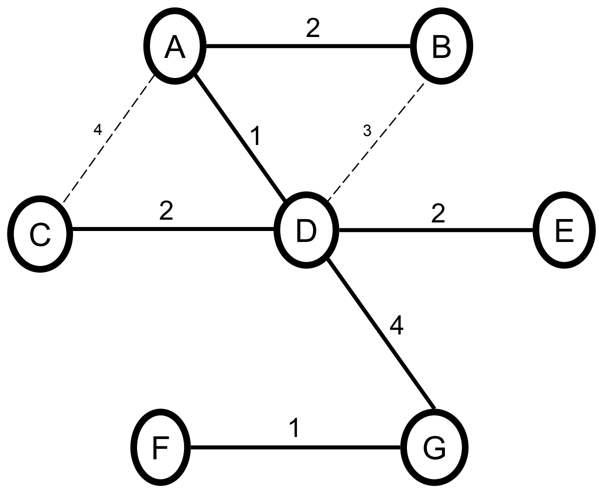
	Priority queue
	A - D: Weight = 1
	F - G: Weight = 1
	A - B: Weight = 2
	C - D: Weight = 2
	D - E: Weight = 2
Skip	B - D: Weight = 3
	A - C: Weight = 4
	D - G: Weight = 4
	C – F: Weight = 5
	E – G: Weight = 6
	D – F: Weight = 8
	B – E: Weight = 10

Example Trace Of Kruskal’s Algorithm: Don’t Use Edge (A-C)



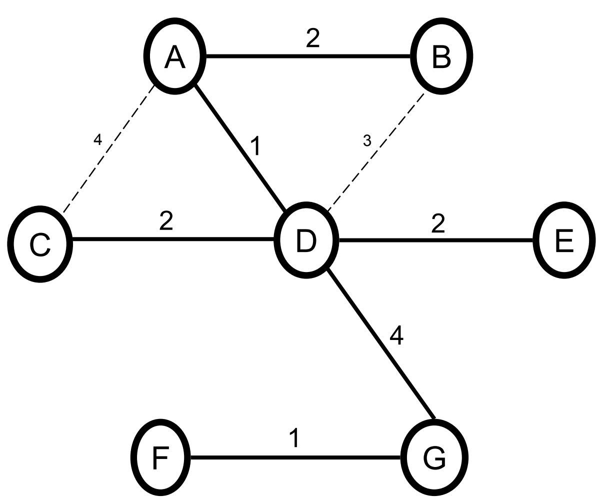
	Priority queue
	A - D: Weight = 1
	F - G: Weight = 1
	A - B: Weight = 2
	C - D: Weight = 2
	D - E: Weight = 2
Skip	B - D: Weight = 3
Skip	A - C: Weight = 4
	D - G: Weight = 4
	C – F: Weight = 5
	E – G: Weight = 6
	D – F: Weight = 8
	B – E: Weight = 10

Example Trace Of Kruskal's Algorithm: Sixth Edge Added (D-G)



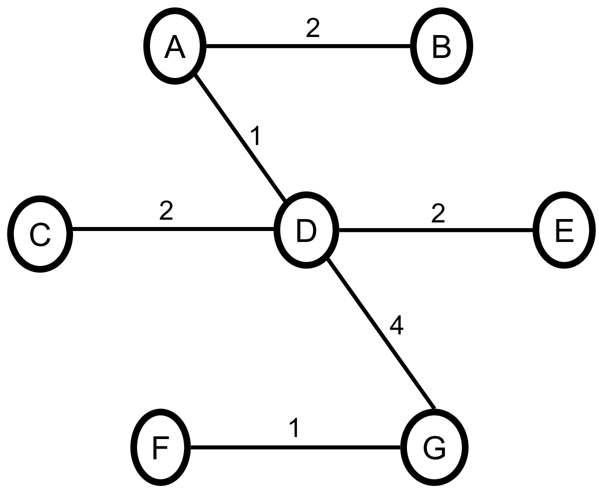
	Priority queue
	A - D: Weight = 1
	F - G: Weight = 1
	A - B: Weight = 2
	C - D: Weight = 2
	D - E: Weight = 2
Skip	B - D: Weight = 3
Skip	A - C: Weight = 4
	D - G: Weight = 4
	C – F: Weight = 5
	E – G: Weight = 6
	D – F: Weight = 8
	B – E: Weight = 10

Example Trace Of Kruskal's Algorithm: Stop, Edges Accepted = No Nodes - 1



	Priority queue
#1	A - D: Weight = 1
#2	F - G: Weight = 1
#3	A - B: Weight = 2
#4	C - D: Weight = 2
#5	D - E: Weight = 2
Skip	B - D: Weight = 3
Skip	A - C: Weight = 4
#6	D - G: Weight = 4
	C – F: Weight = 5
	E – G: Weight = 6
	D – F: Weight = 8
	B – E: Weight = 10

Example Trace Of Kruskal's Algorithm: The Final Tree



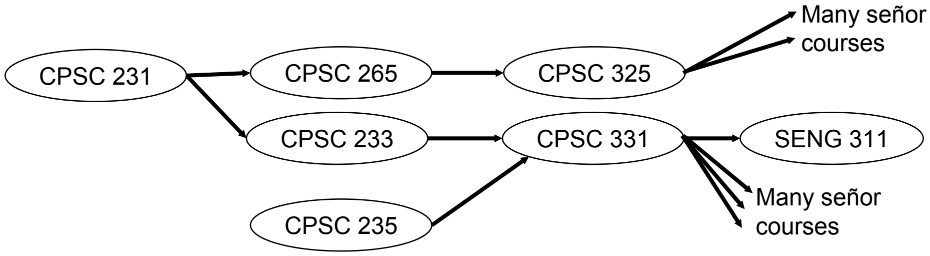
Topological Sorting Of Graphs: Real Life Example!

- Used to order information that could be represented as nodes in a directed acyclic graph.
- If there's a path from a node n1 to another node n2, then n2 appears after n1 in the ordering.

First year	
Fall	**Winter**
CPSC 231 or 235	CPSC 233
Math 221	CPSC 265
Math 249 or 251	Math 271
Phil 279	Math 253 or Stat 211
Second year	
CPSC 325	CPSC 313
CPSC 331	SENG 311 or CPSC 333
: **:**	**:** **:**

Topological Sorting Of Graphs: Real Life Example

- Used to order information that could be represented in the form of the nodes in a directed acyclic graph:

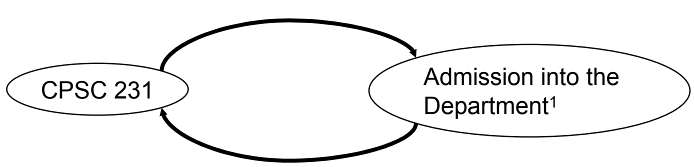


Note: Topological Sorting Cannot Be Done With Cyclical Graphs

- Sorry no vacancies today.[1]

Computer Science 231 H(3-2T-1)
Introduction to Computer Science I
Problem solving and programming in a structured language. Data representation, program control, basic file handling, the use of simple data structures and their implementation. Pointers. Recursion.
Prerequisite: Admission into the Department of Computer Science

Admission requirements into the Department of Computer Science
A grade of C- or higher in CPSC 231



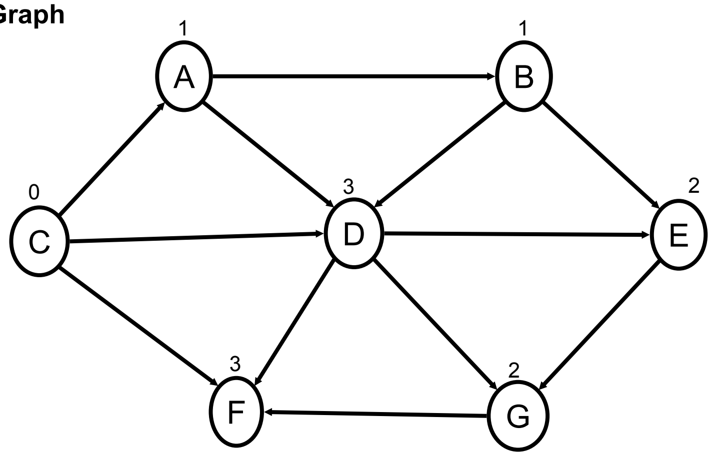
1 This example is purely fictional that was created to illustrate the principles of graph theory and should not be taken as an official description of perquisite requirements for the Department of Computer Science

Algorithm For A Topological Sort

```
public List topogicalSort (Graph g)
{
    int i;
    int noNodes = g.getNumberNodes ();
    List orderedNodes = new List ();
    Node temp;
    for (i = 0; i < noNodes; i++)
     {
        temp = g.getNextTopParent ();
        orderedNodes.add (temp);
        g.deleteNodeEdges (temp);
    }
}
```


Example Of A Topological Sort

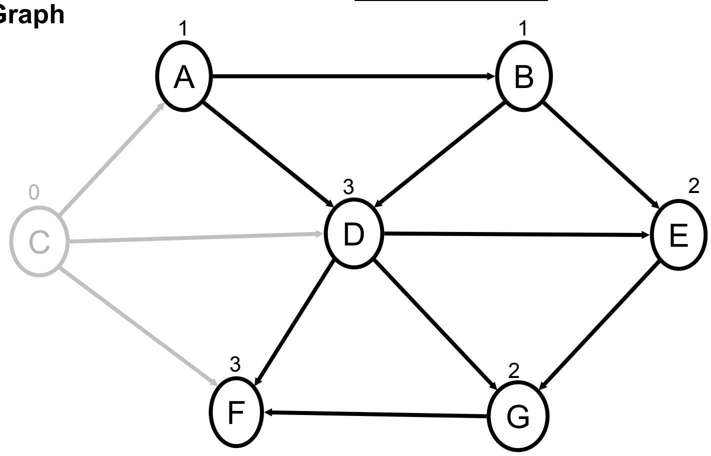
Graph



List

Remove “C” And Edges From Graph And Add To List

Graph

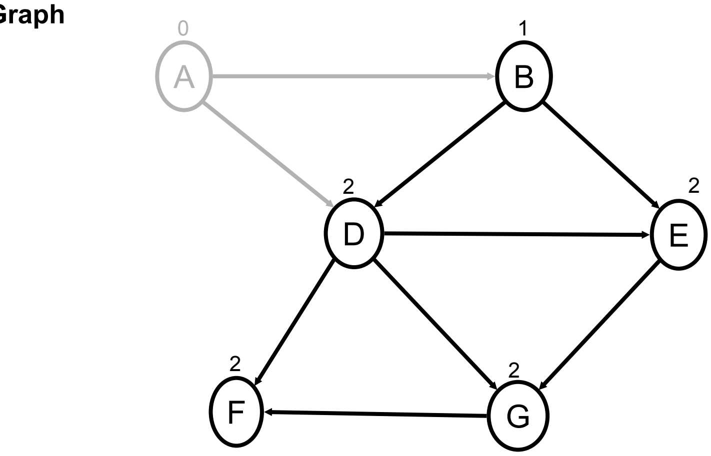


List

[0]	[1]	[2]	[3]	[4]	[5]	[6]
C						

Update The Numbers And Remove “A”

Graph

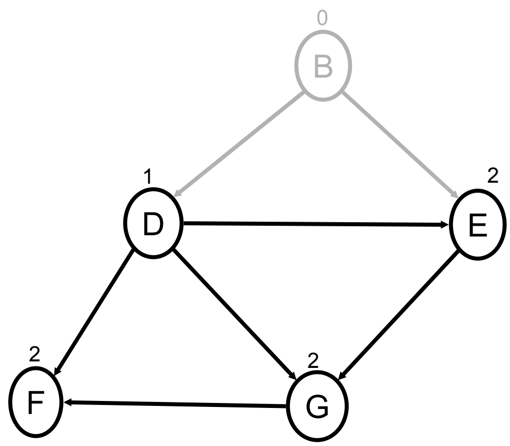


List

[0]	[1]	[2]	[3]	[4]	[5]	[6]
C	A					

Update The Numbers And Remove "B"

Graph

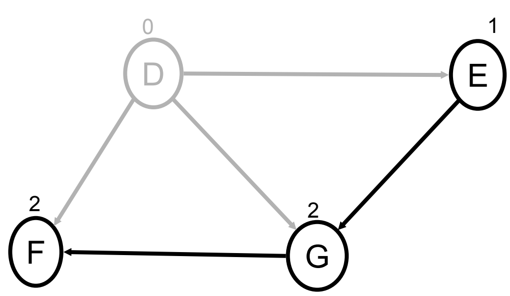


List

[0]	[1]	[2]	[3]	[4]	[5]	[6]
C	A	B				

Update The Numbers And Remove "D"

Graph

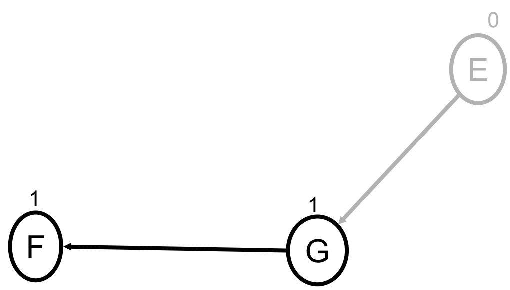


List

[0]	[1]	[2]	[3]	[4]	[5]	[6]
C	A	B	D			

Update The Numbers And Remove "E"

Graph

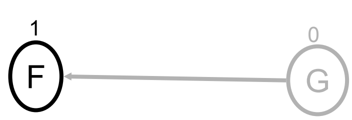


List

[0]	[1]	[2]	[3]	[4]	[5]	[6]
C	A	B	D	E		

Update The Numbers And Remove "G"

Graph



List

[0]	[1]	[2]	[3]	[4]	[5]	[6]
C	A	B	D	E	G	

Update The Numbers And Remove "F"

Graph

List

[0]	[1]	[2]	[3]	[4]	[5]	[6]
C	A	B	D	E	G	F

You Should Now Know

- What is a minimum spanning tree
- Two algorithms (Prim's and Kruskal's) for creating minimum spanning trees
- What is a topological sort and the algorithm for this sort.

Sources Of Lecture Material

- *"Data Abstraction and Problem Solving With Java: Walls and Mirrors"* updated edition by Frank M. Carrano and Janet J. Prichard
- "Data Structures and Problem Solving Using C++ (2nd edition)" by Mark Allan Weiss
- "Data Structures and Problem Solving Using Java (2nd edition)" by Mark Allan Weiss
- Lecture notes by Matthew A. Becker http://www.andrew.cmu.edu/user/mbecker/
- CPSC 331 course notes by Marina L. Gavrilova http://pages.cpsc.ucalgary.ca/~marina/331/