

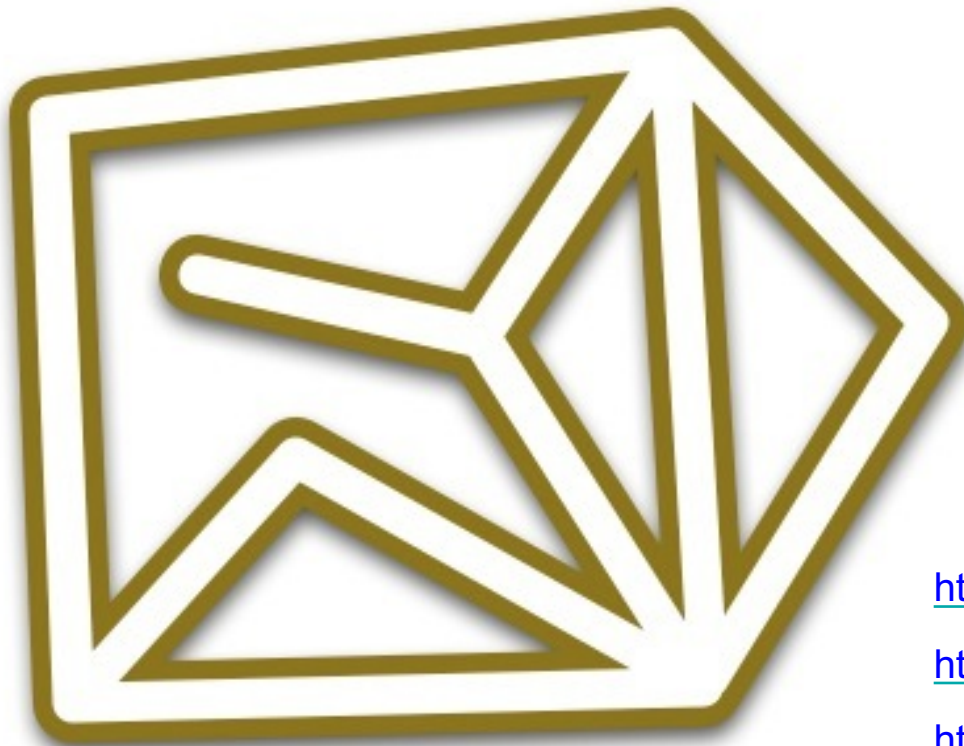
Today's announcements:

MP7 available. Due 5/2, 11:59p.

Graphs: Traversal - DFS



Ariadne, Theseus, and the
Minotaur



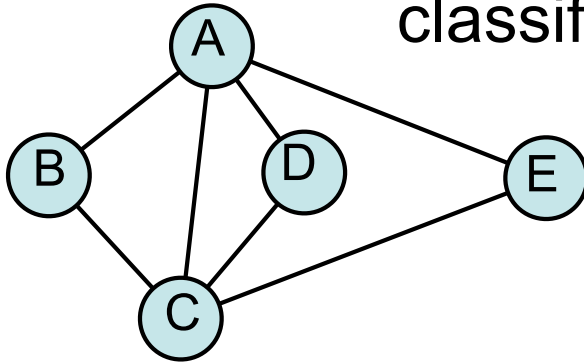
<http://www.cs.duke.edu/csed/jawaa2/examples/DFS.html>

<http://www.student.seas.gwu.edu/~idsv/idsv.html>

<http://www.youtube.com/watch?v=8qrZ1clEp-Y>

DFS: “visits” each vertex

classifies each edge as either “discovery” or “back”



Algorithm DFS(G)

Input: graph G

Output: labeling of the edges of G
as discovery edges and back edges

For all u in G.vertices()

 setLabel(u, UNVISITED)

For all e in G.edges()

 setLabel(e, UNEXPLORED)

For all v in G.vertices()

 if getLabel(v) = UNVISITED

 DFS(G,v)

Algorithm DFS(G,v)

Input: graph G and start vertex v

Output: labeling of the edges of G in the
connected component of v as discovery edges and
back edges

setLabel(v, VISITED)

For all w in G.adjacentVertices(v)

 if getLabel(w) = UNVISITED

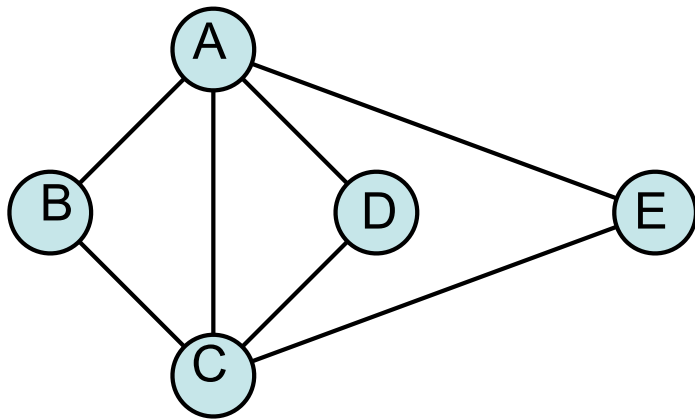
 setLabel((v,w),DISCOVERY)

 DFS(G,w)

 else if getLabel((v,w)) = UNEXPLORED

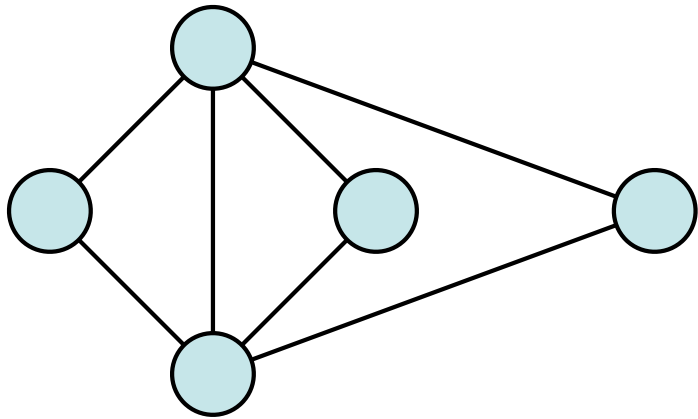
 setLabel(e,BACK)

Graphs: DFS example



A	B	C	D	E
B	A	C		
C	B	A	D	E
D	A	C		
E	A	C		

Graphs: DFS Analysis



setting/getting labels

every vertex labeled twice

every edge is labeled twice

querying vertices

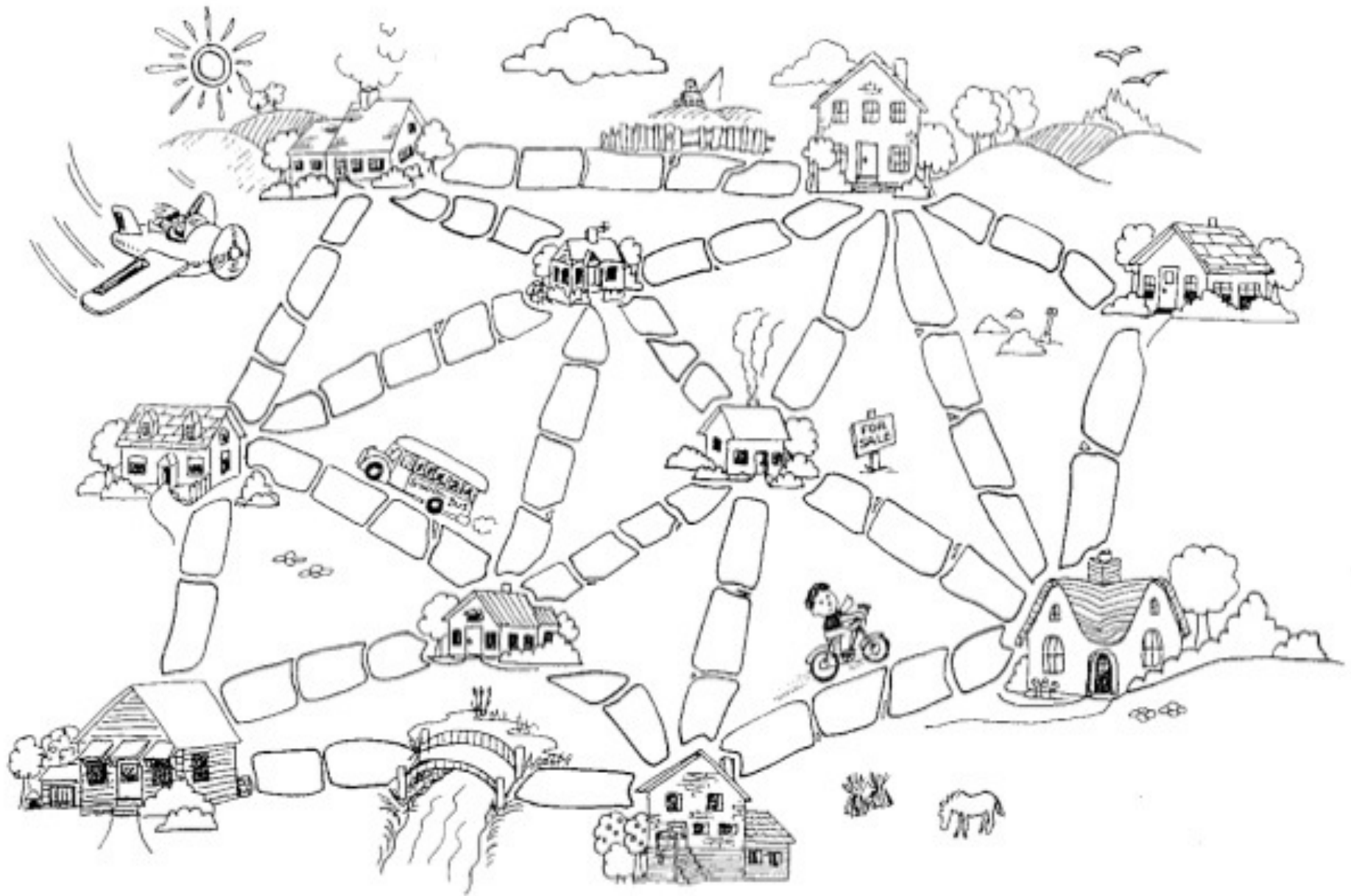
each vertex

total over algorithm

querying edges

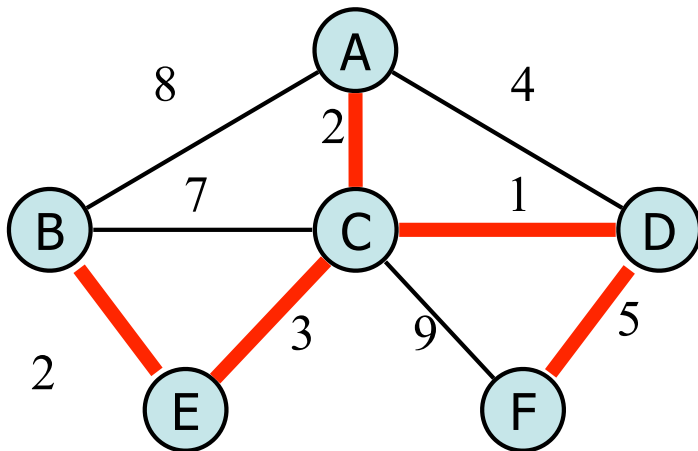
TOTAL RUNNING TIME:

Muddy City...

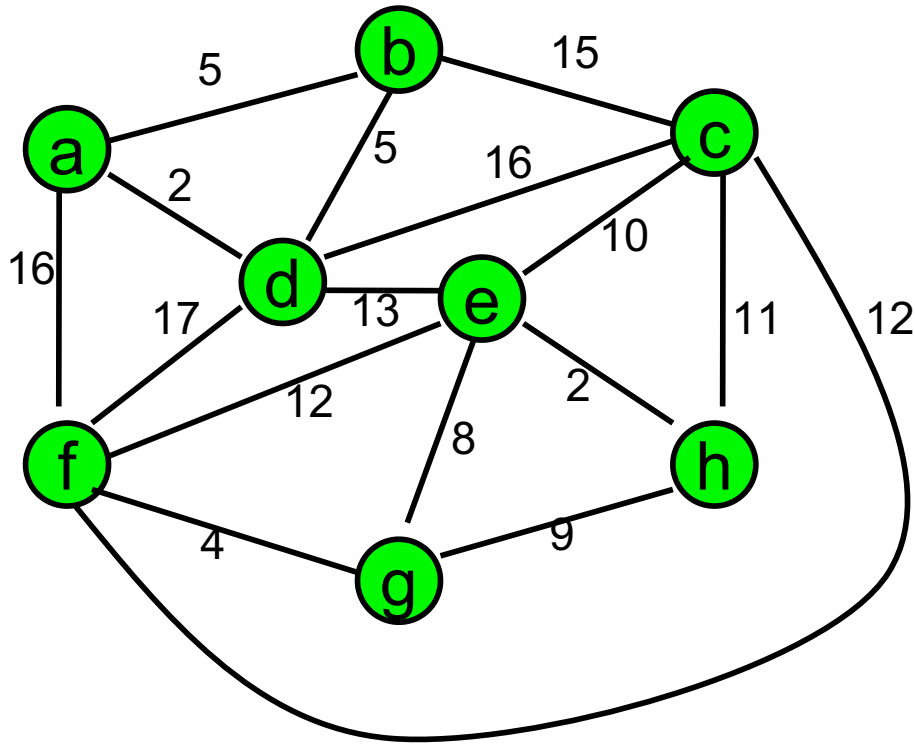


Minimum Spanning Tree Algorithms:

- Input: connected, undirected graph G with unconstrained edge weights
- Output: a graph G' with the following characteristics -
 - G' is a spanning subgraph of G
 - G' is connected and acyclic (a tree)
 - G' has minimal total weight among all such spanning trees -

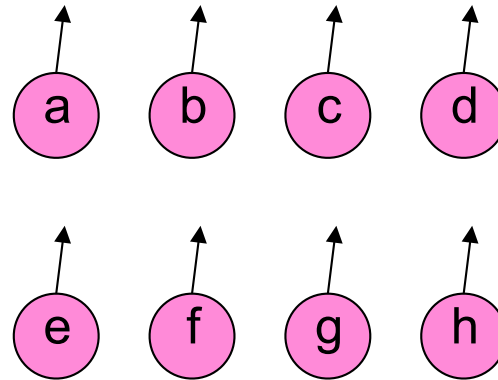
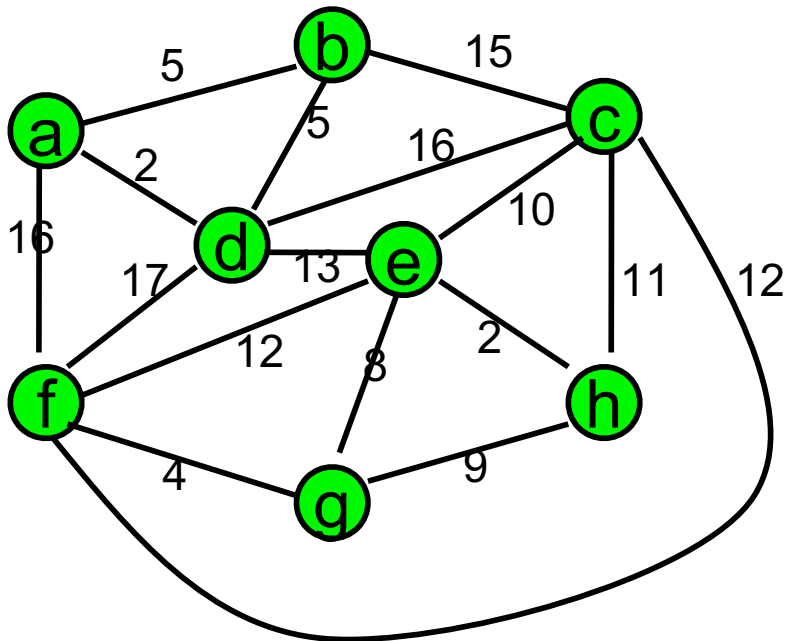


Kruskal's Algorithm



(a,d)
(e,h)
(f,g)
(a,b)
(b,d)
(g,e)
(g,h)
(e,c)
(c,h)
(e,f)
(f,c)
(d,e)
(b,c)
(c,d)
(a,f)
(d,f)

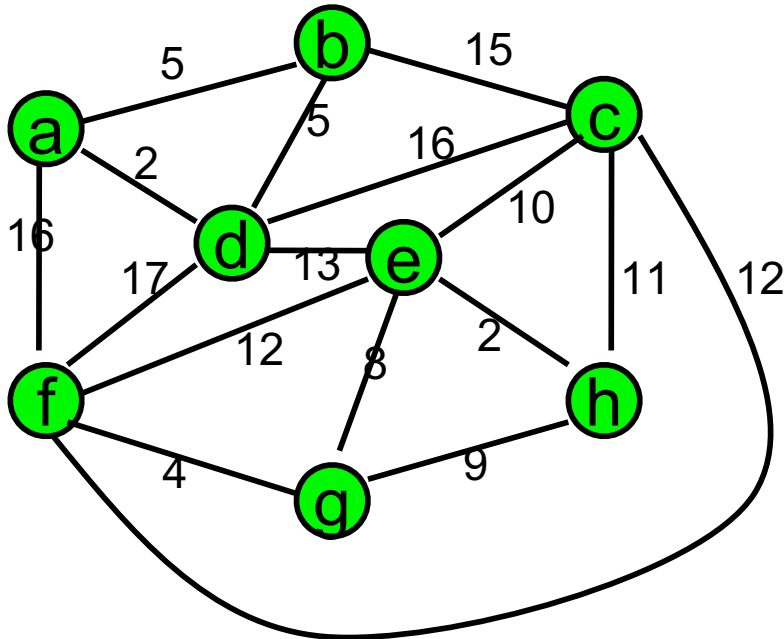
Kruskal's Algorithm (1956)



(a,d)
(e,h)
(f,g)
(a,b)
(b,d)
(g,e)
(g,h)
(e,c)
(c,h)
(e,f)
(f,c)
(d,e)
(b,c)
(c,d)
(a,f)
(d,f)

1. Initialize graph T whose purpose is to be our output. Let it consist of all n vertices and no edges.
2. Initialize a disjoint sets structure where each vertex is represented by a set.
3. RemoveMin from PQ . If that edge connects 2 vertices from different sets, add the edge to T and take Union of the vertices' two sets, otherwise do nothing. Repeat until _____ edges are added to T .

Kruskal's Algorithm - preanalysis



Algorithm *KruskalMST(G)*

disjointSets forest;

for each vertex v in V do

forest.makeSet(v);

priorityQueue Q ;

Insert edges into Q , keyed by weights

graph $T = (V, E)$ with $E = \emptyset$;

while T has fewer than $n-1$ edges do

edge $e = Q.removeMin()$

Let u, v be the endpoints of e

if $forest.find(v) \neq forest.find(u)$ then

Add edge e to E

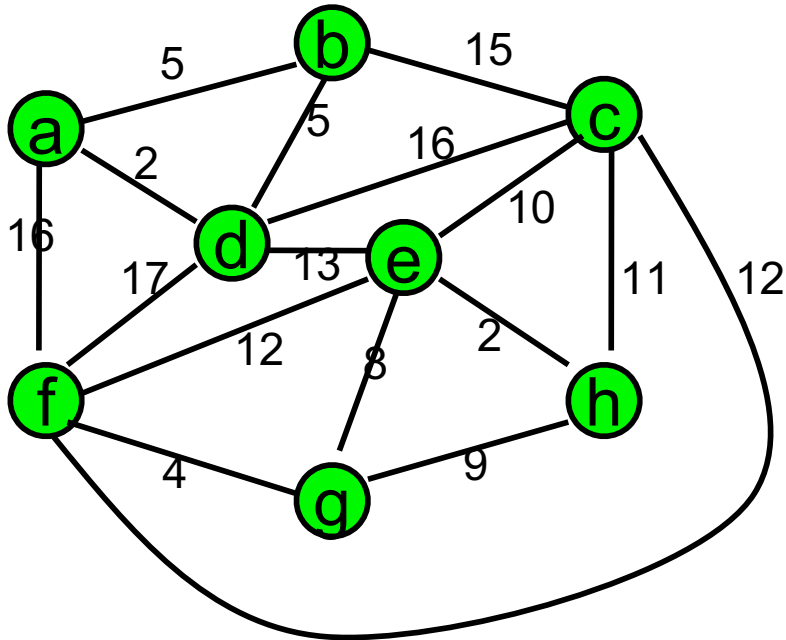
forest.smartUnion

(forest.find(v), forest.find(u))

return T

Priority Queue:	Heap	Sorted Array
To build		
Each removeMin		

Kruskal's Algorithm - analysis



Algorithm *KruskalMST*(G)

disjointSets forest;
for each vertex v in V **do**
 forest.makeSet(v);

priorityQueue Q;
 Insert edges into Q , keyed by weights

graph $T = (V, E)$ with $E = \emptyset$;

while T has fewer than $n-1$ edges **do**
 edge $e = Q.removeMin()$
 Let u, v be the endpoints of e
 if *forest.find*(v) $\neq$ *forest.find*(u) **then**
 Add edge e to E
 forest.smartUnion
 (*forest.find*(v), *forest.find*(u))

return T

Priority Queue:	Total Running time:
Heap	
Sorted Array	