# 04-630
# Data Structures and Algorithms for Engineers

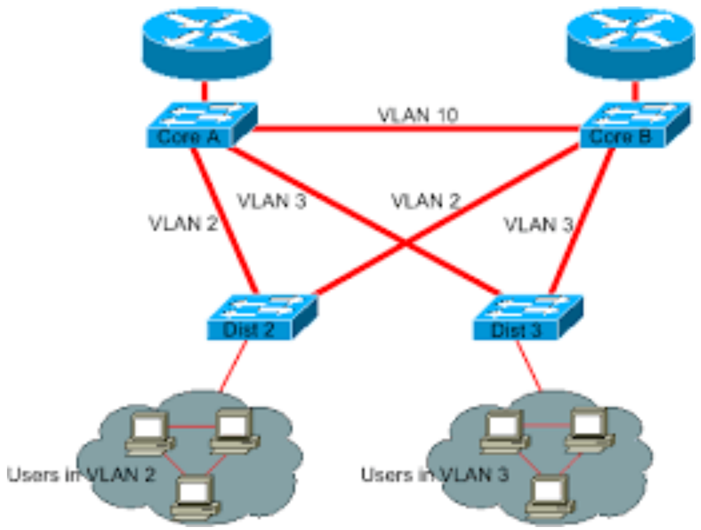
Lecture 18: Graph Algorithms

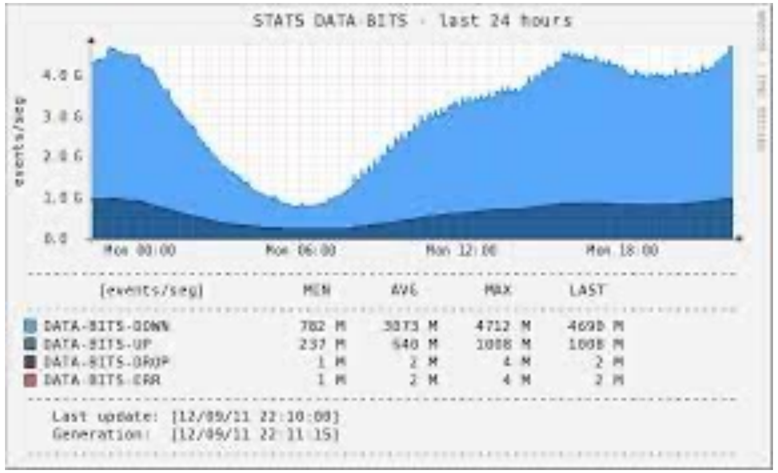
# Previous

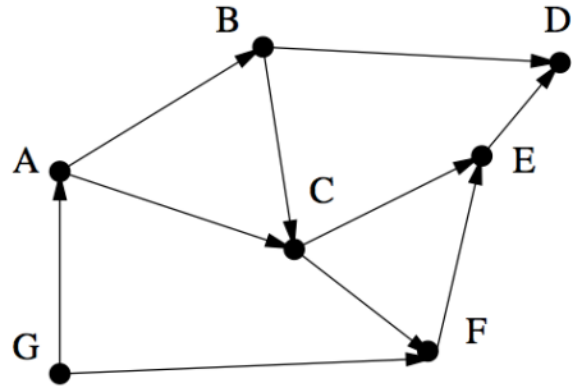
- Graphs basics
- Applications
- Traversal
  - BFS
  - DFS

# Outline

- DAGs and Topological sorting
- Minimum spanning tree
  - Prims
  - Kruskall
- Shortest path algorithms
  - Dijkstras,
  - Floyds






# Topological sorting

# Topological sorting: applications

- In applications where precedence ordering is needed, e.g.:
  - Dressing up
  - Preparing a recipe
  - Choosing courses (based on prerequisites).
  - Scheduling jobs or tasks where there are dependencies among jobs or tasks.
- See more examples.

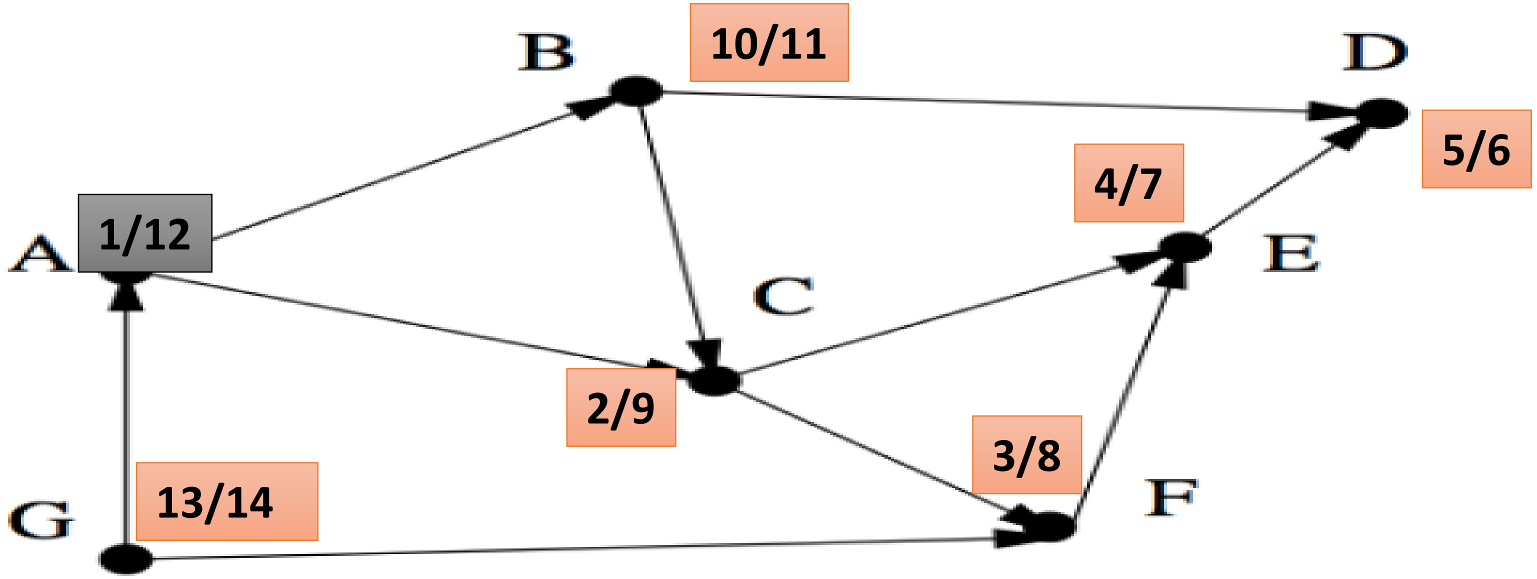
# DAG & Topological sorting

- Directed acyclic graph (DAG): directed graph with no cycles.
- Can denote precedence among nodes.
- Using ***topological sorting***, we can obtain a ***total order***.
- Topological sorting:
  - involves sorting a DAG
  - Label the vertices in the ***reverse order*** in which they are ***processed*** (completed) to find the topological sort of a DAG
- ***Definition***: A topological sort of a DAG is a linear ordering of all its vertices such that for any edge (u,v) in the DAG, u appears before v in the ordering

# Topological sorting: algorithm

- TopologicalSort(G)
  - Execute DFS(G) to compute v.endtime for each vertex v
  - As each vertex is finished, insert it at the beginning of a linked list (*or insert it on the stack*)
  - Return the linked list *(or stack)* of vertices

# Topological sorting: worked example (14/14)

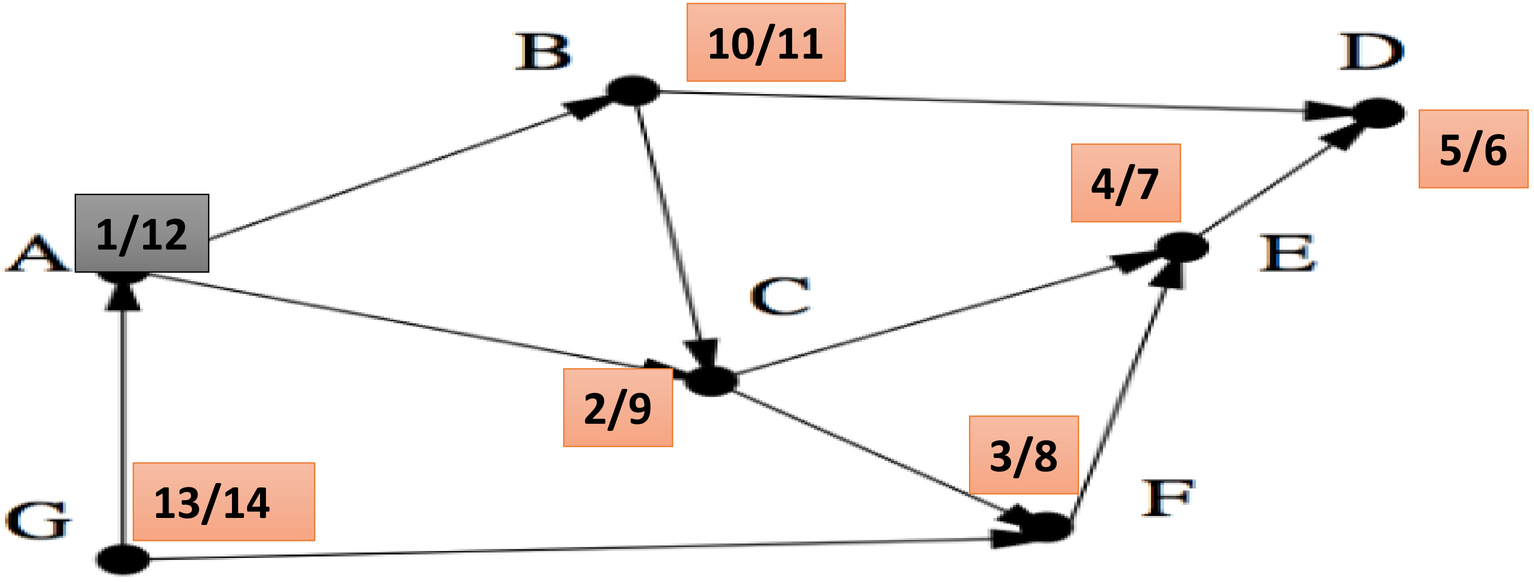


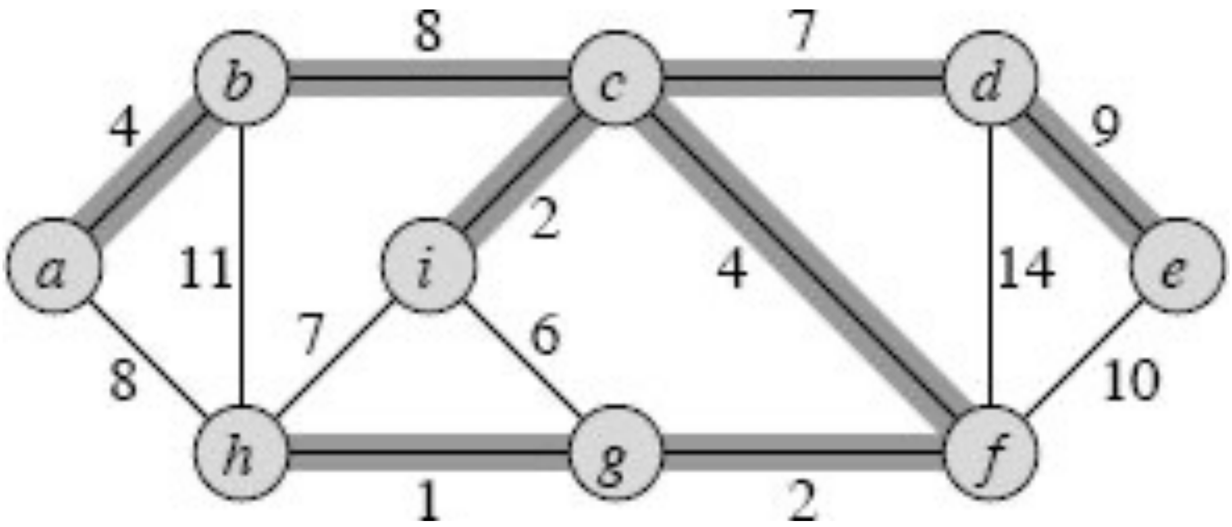

| Stack |
| --- |
| G |
| A |
| B |
| C |
| F |
| E |
| D |

**Topological order**: G, A, B, C, F, E, D

# Quiz

- Comment on the performance of topological sorting algorithm.



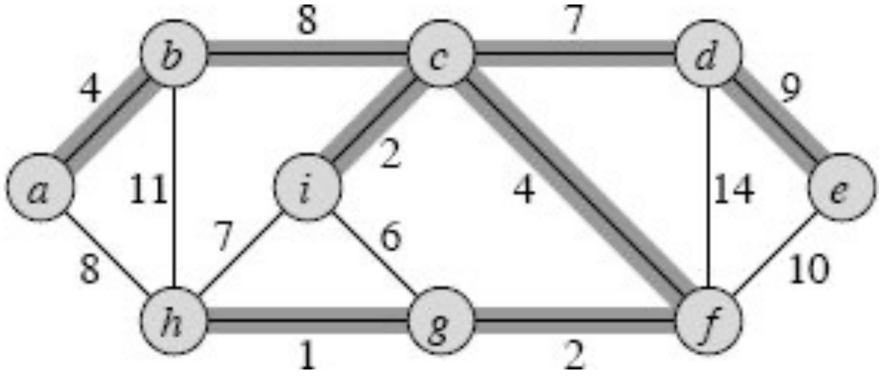



# Minimum Spanning Tree

Prims, Kruskall

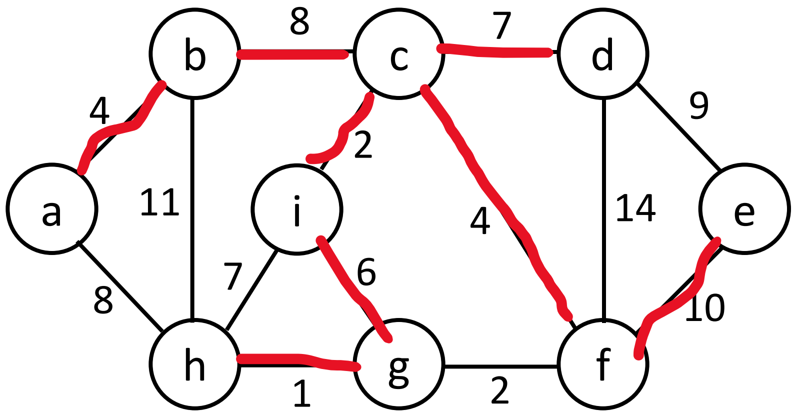
# MST

- Spanning forest
  - If a graph is not connected, then there is a spanning tree for each connected component of the graph



# Spanning Tree

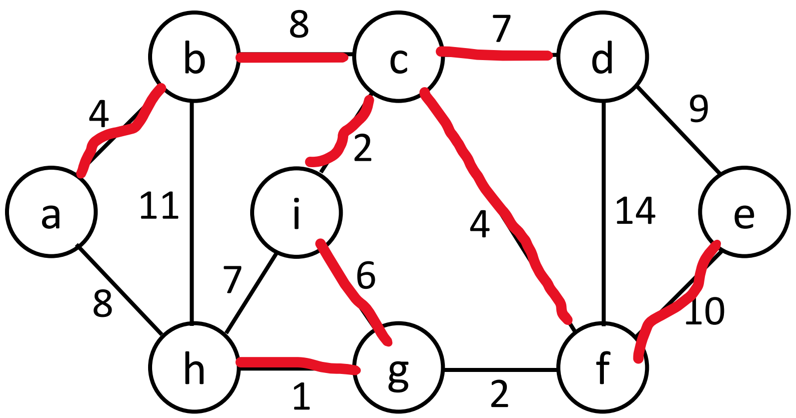
- A tree which contains all the vertices of the graph
- Given (a connected) graph G(V,E), a spanning tree T(V',E'):
  - Is a subgraph of G; such that, $V' \subseteq V$, $E' \subseteq E$, and $V' = V$
  - T forms a tree (i.e., no cycle); and
  - $|E'|=|V| -1$ edges




This is a **spanning tree** **<u>not</u>** a minimum spanning tree

# Minimum Spanning Tree

- Minimum Spanning Tree
  - Spanning tree with the **minimum sum of weights.**
  - There may be more than one MST for a graph.
- Given weighted edges:
  - find the minimum cost spanning tree
- Process:
  - Add an edge of minimum cost that does not create a cycle (greedy algorithm)
  - Repeat **|V| -1** times

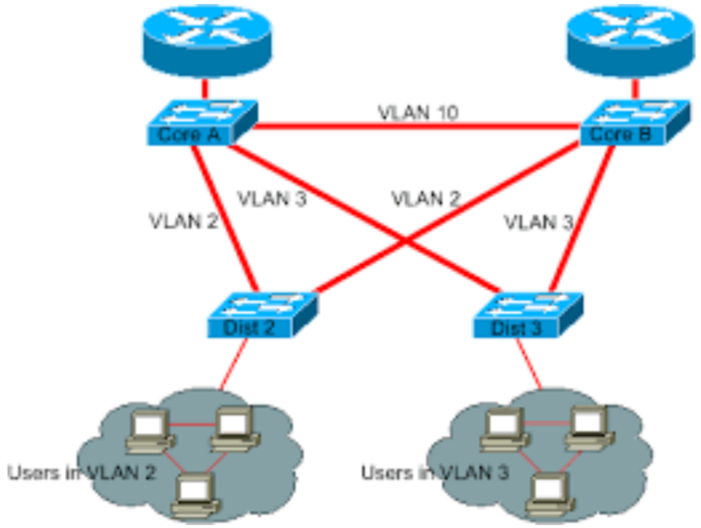


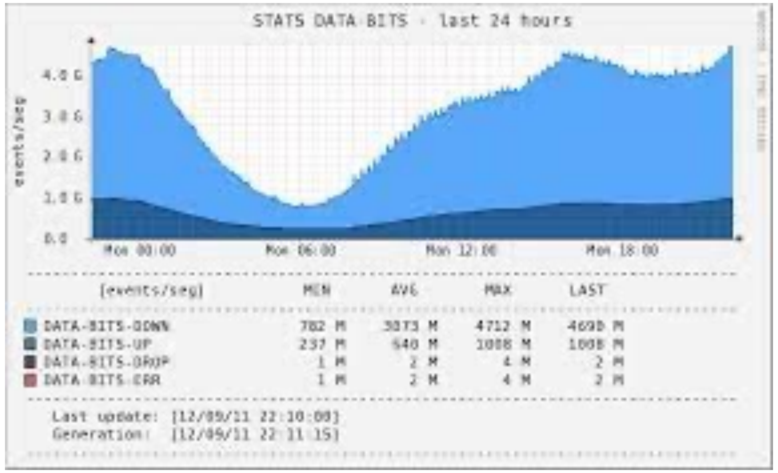

This is a **spanning tree <u>not</u>** a minimum spanning tree

# Applications of MST

- Find the cheapest connections for cities, computers, networks, etc.

- Plan road repairs in city or between towns such that traffic continues to flow.

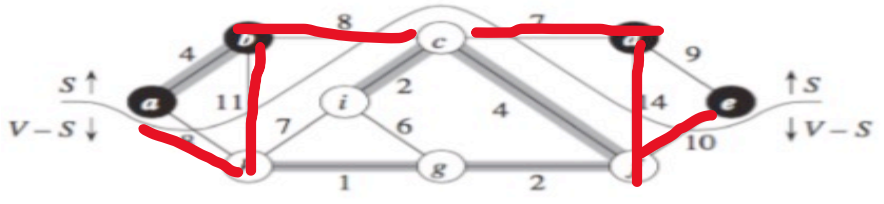


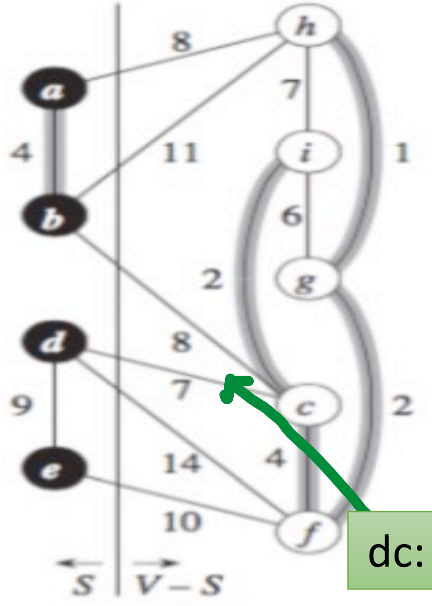
# MST: definitions

Edges crossing the cut (all that are marked red)

- ***Definition***: A **cut** (S, V –S) of an undirected graph is a partition of the set of vertices into the sets S and V – S.
- ***Definition***: A cut **respects** a set of edges A if no edge in A crosses the cut. That is, none of the edges have one vertex in S and the other vertex in V – S.
- ***Definition***: An edge is a **light edge** satisfying a property if it has the smallest weight out of all edges that satisfy that property
  - Specifically, an edge is a ***light edge*** crossing a cut if it has the smallest weight out of all edges that cross the cut.



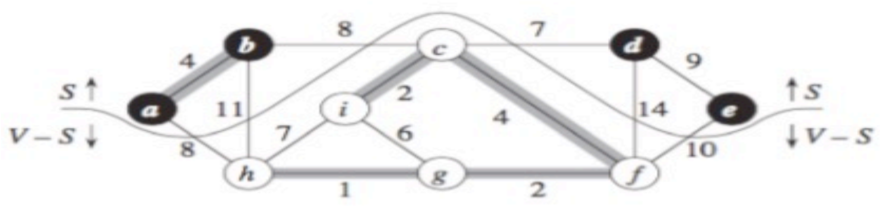
(a)



(b)

dc: light edge

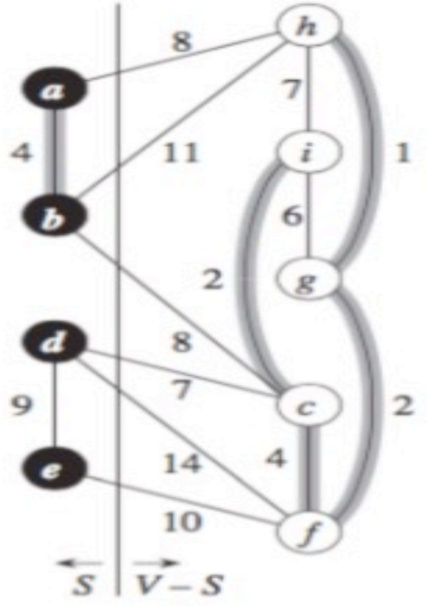
# MST: definitions



(a)

- ***Definition***: A **cut** (S, V −S) of an undirected graph is a partition of the set of vertices into the sets S and V − S.
- ***Definition***: A cut **respects** a set of edges A if no edge in A crosses the cut. That is, none of the edges have one vertex in S and the other vertex in V − S.
- ***Definition***: An edge is a **light edge** satisfying a property if it has the smallest weight out of all edges that satisfy that property
  - Specifically, an edge is a ***light edge*** crossing a cut if it has the smallest weight out of all edges that cross the cut.





(b)

# MST Algorithms

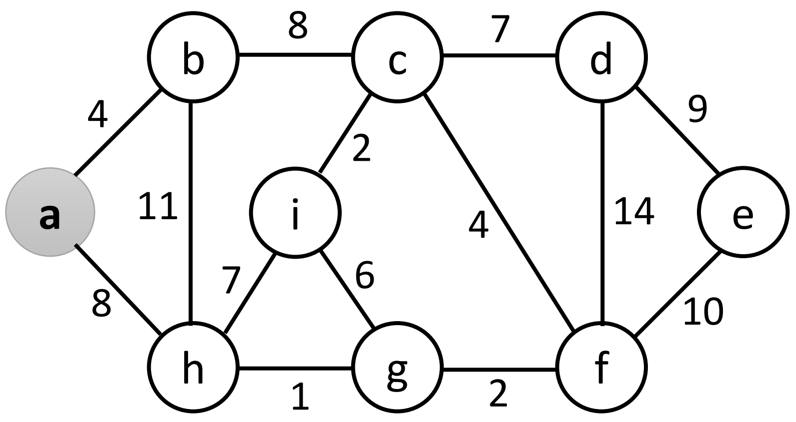
- Prim’s algorithm:
  - build tree incrementally
- Kruskal’s algorithm:
  - build forest that will finish as a tree.

# MST: Prim's Algorithm

- Repeatedly select the smallest weight edge that increases the number of vertices in the tree.
    1. Start from any vertex
    2. Grow the rest of the tree, one edge at a time
    3. Until all vertices are included.

Not Pym

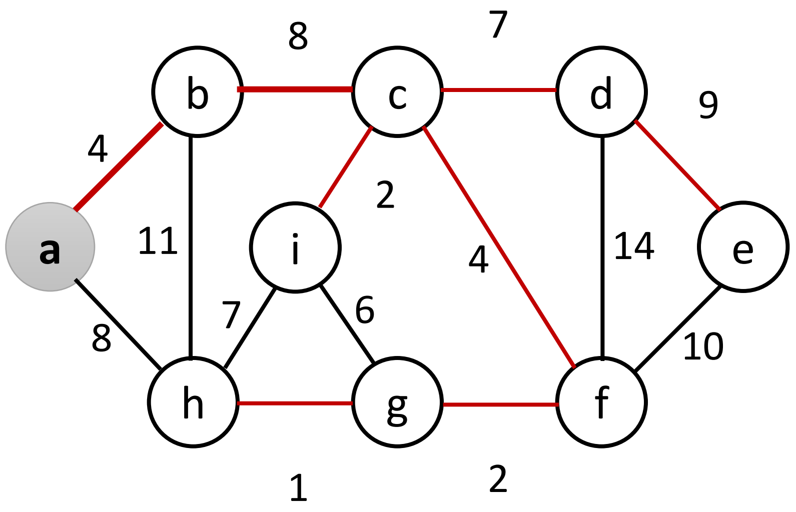
# Prim’s Algorithm example




Choose a vertex at random and initialize

e.g. Select a.  Initialize: V={a}, E’={}

# Prim’s Algorithm example




Repeat  until all vertices have been chosen

Choose the vertex u not in V’ such that
edge weight from u to a vertex in V’ is minimal}

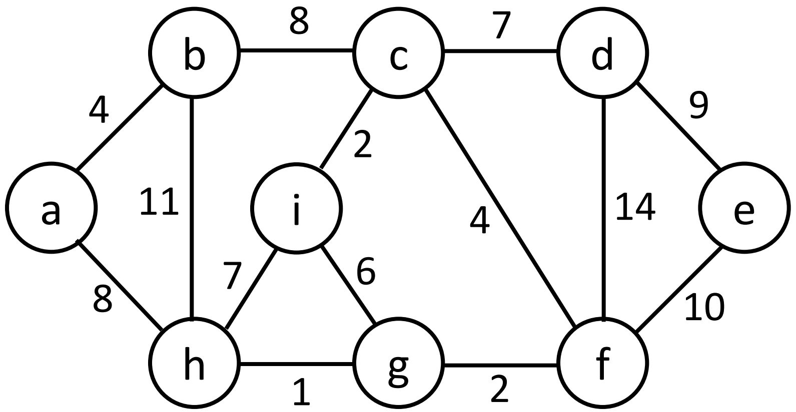
Choose e.

V’={a,b,c, i, f, g, h, d, e}

E’={(a,b),(b,c), (c,i), (c,f), (f,g), (g,h), (c,d), (d,e)}

# MST: Kruskal's algorithm

- Start with each vertex being its own component
- Repeatedly merge two components into one by choosing the **light edge** that connects them
- Which components to consider at each iteration?
  - Scan the set of edges by increasing order by weight

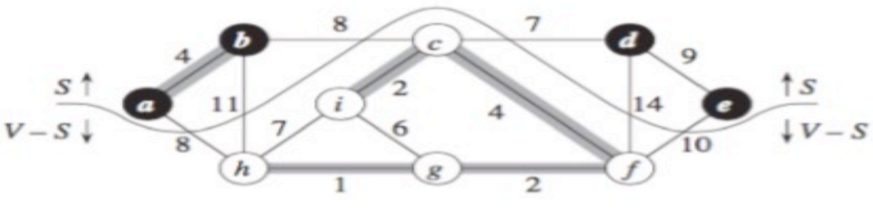



| Edge | Weight |
|---|---|
| hg | 1 |
| ci | 2 |
| gf | 2 |
| ab | 4 |
| cf | 4 |
| gi | 6 |
| hi | 7 |
| cd | 7 |
| bc | 8 |
| ah | 8 |
| de | 9 |
| ef | 10 |
| bh | 11 |
| df | 14 |

Which edges cross the cut?

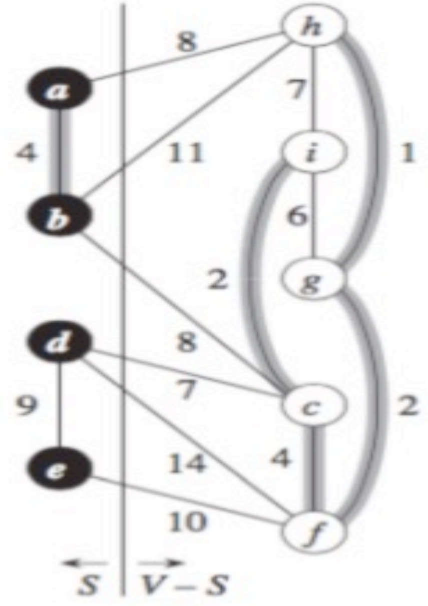
# MST: definitions

- ***Definition***: A **cut** (S, V −S) of an undirected graph is a partition of the set of vertices into the sets S and V − S.
- ***Definition***: A cut **respects** a set of edges A if no edge in A crosses the cut. That is, none of the edges have one vertex in S and the other vertex in V − S.
- ***Definition***: An edge is a **light edge** satisfying a property if it has the smallest weight out of all edges that satisfy that property
    - Specifically, an edge is a ***light edge*** crossing a cut if it has the smallest weight out of all edges that cross the cut.




(a)

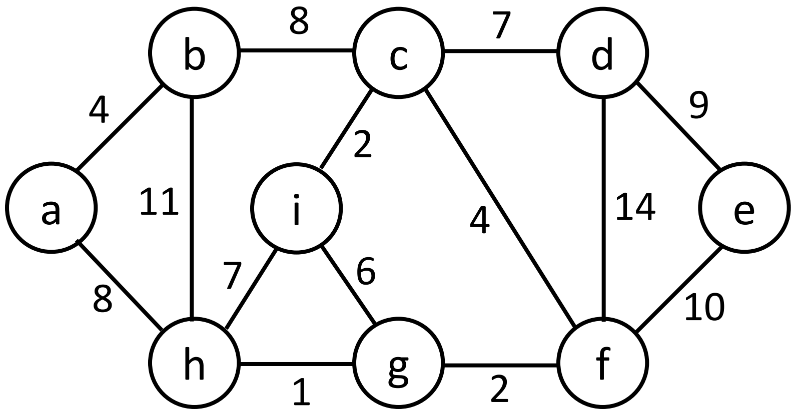
What is the light edge?




(b)

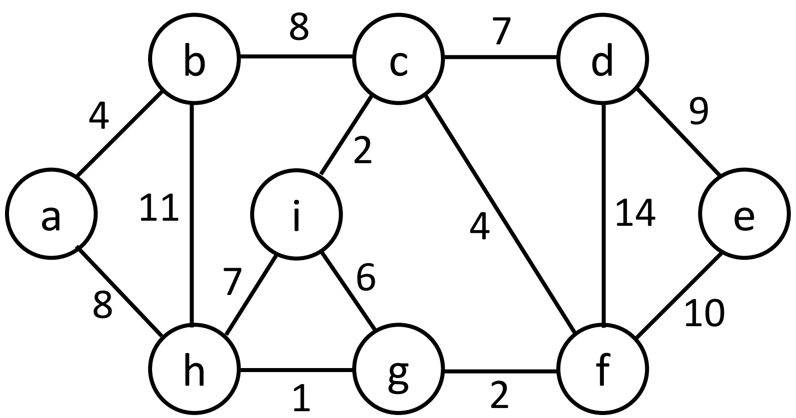
# MST: Kruskal’s algorithm

- Start with each vertex being its own component
- Repeatedly merge two components into one by choosing the **light edge** that connects them
- Which components to consider at each iteration?
  - Scan the set of edges by increasing order by weight




| Edge | Weight |
|---|---|
| hg | 1 |
| ci | 2 |
| gf | 2 |
| ab | 4 |
| cf | 4 |
| gi | 6 |
| hi | 7 |
| cd | 7 |
| bc | 8 |
| ah | 8 |
| de | 9 |
| ef | 10 |
| bh | 11 |
| df | 14 |

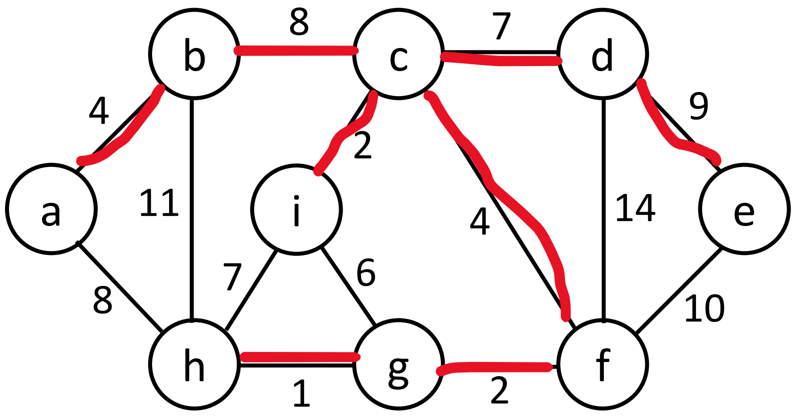
# Kruskal's algorithm example




Initial Forest: {a},{b},{c},{d},{e},{f},{g},{h},{i}

| Edge | Weight |
|---|---|
| hg | 1 |
| ci | 2 |
| gf | 2 |
| ab | 4 |
| cf | 4 |
| gi | 6 |
| hi | 7 |
| cd | 7 |
| bc | 8 |
| ah | 8 |
| de | 9 |
| ef | 10 |
| bh | 11 |
| df | 14 |

# Kruskal’s algorithm example




1. Add (h,g): **{g,h}**,{a},{b},{c},{d},{e},{f}, {i}
2. Add (c,i): **{g,h}** ,**{c,i}**,{a},{b}, {d},{e},{f}
3. Add (g,f): **{g,h,f}, {c,i}**, {a},{b}, {d},{e}
4. Add (a,b): **{g,h,f}, {c,i}**, **{a,b}**,{d},{e}
5. Add (c,f): **{g,h,f, c,i}, {a,b}**,{d},{e}
6. Ignore (g,i): why?
7. Ignore (h,i): why?
8. Add (c,d): **{g,h,f, c,i,d}, {a,b}**,{e}
9. Add (b,c): **{g,h,f, c,i,d, a,b},** {e}
10. Ignore (a,h): why?
11. Add (d,e):**{g,h,f, c,i,d, a,b,e}**
12. Ignore (e,f): **{g,h,f, c,i,d, a,b,e}**
13. Ignore (b,h): **{g,h,f, c,i,d, a,b,e}**
14. Ignore (d,f): **{g,h,f, c,i,d, a,b,e}**

| Edge | Weight |
|---|---|
| hg | 1 |
| ci | 2 |
| gf | 2 |
| ab | 4 |
| cf | 4 |
| gi | 6 |
| hi | 7 |
| cd | 7 |
| bc | 8 |
| ah | 8 |
| de | 9 |
| ef | 10 |
| bh | 11 |
| df | 14 |

# Implementing Kruskal's algorithm

- Use:
  - adjacency list to represent the graph
  - disjoint set to represent each tree in the forest
  - binary heap for edges

# MST: Kruskal’s Algorithm

- Difference with Prim’s algorithm:
  - Prim’s algorithm grows one tree all the time
  - Kruskal’s algorithm grows multiple trees (i.e., a forest) at the same time.
  - Since an MST has exactly |V| - 1 edges, after |V| - 1 merges, we would have only one component (one merged tree)

# Summary

- Topological sorting and its applications.
- Minimum spanning tree algorithms and applications.

# Acknowledgement

Adapted from material by Prof. David Vernon

Augmented by material from:

The Algorithm Design Manual 2nd Edition: by Steven Skiena

Introduction to Algorithms, 3rd Edition, Thomas H. Cormen et al. (2009)