COMP 6651
Algorithm Design Techniques
Week 9

Minimum Spanning Trees .Shortest Paths. Dijkstra.

Ack: Some slides from Fredo Durand MIT, Daniel Heilper from Haifa University and the web

Spanning Trees

- Tree that contains all vertices in the graph
- Examples:
  - BFS
  - DFS
  - Minimum Spanning Trees
- Why?
- Provides some vertex relationship and ordering
- Reveals some properties of the graph
- Help solve several important problems

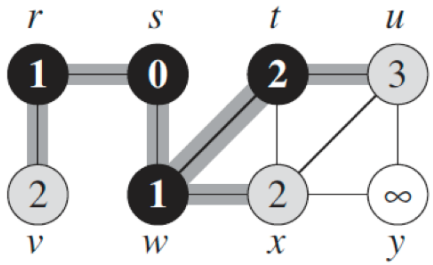
Breadth-first Search Exploration of a Graph

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

BFS assume the graph is connected

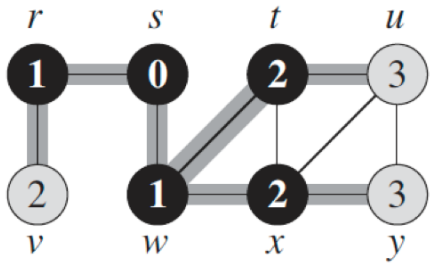
Breadth-first Search Exploration of a Graph

(e)



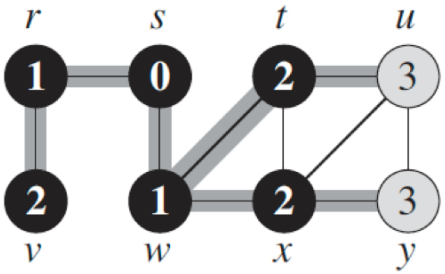
Q | x | v | u |
2 2 3

(f)



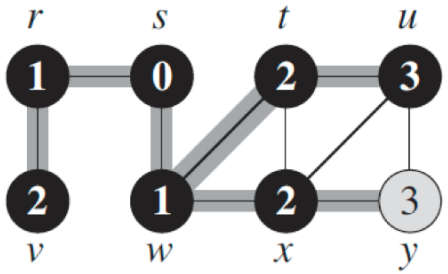
Q | v | u | y |
2 3 3

(g)



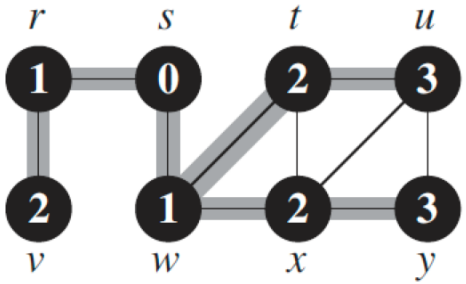
Q | u | y |
3 3

(h)



Q | y |
3

(i)



Q $\emptyset$

Breadth-first Search Exploration of a Graph

Analysis:

O(V+E) –

What is E – worse case: E = V^2

O(v) – for planar graphs

Spanning Trees

- Trees that contain all nodes
- Edges in the tree are subset of edges in graph
- Applications
  - How many connected components (true for all spanning trees)
  - Helps find cycles (true for all spanning trees)
  - Eulerian and Hamiltonian cycles

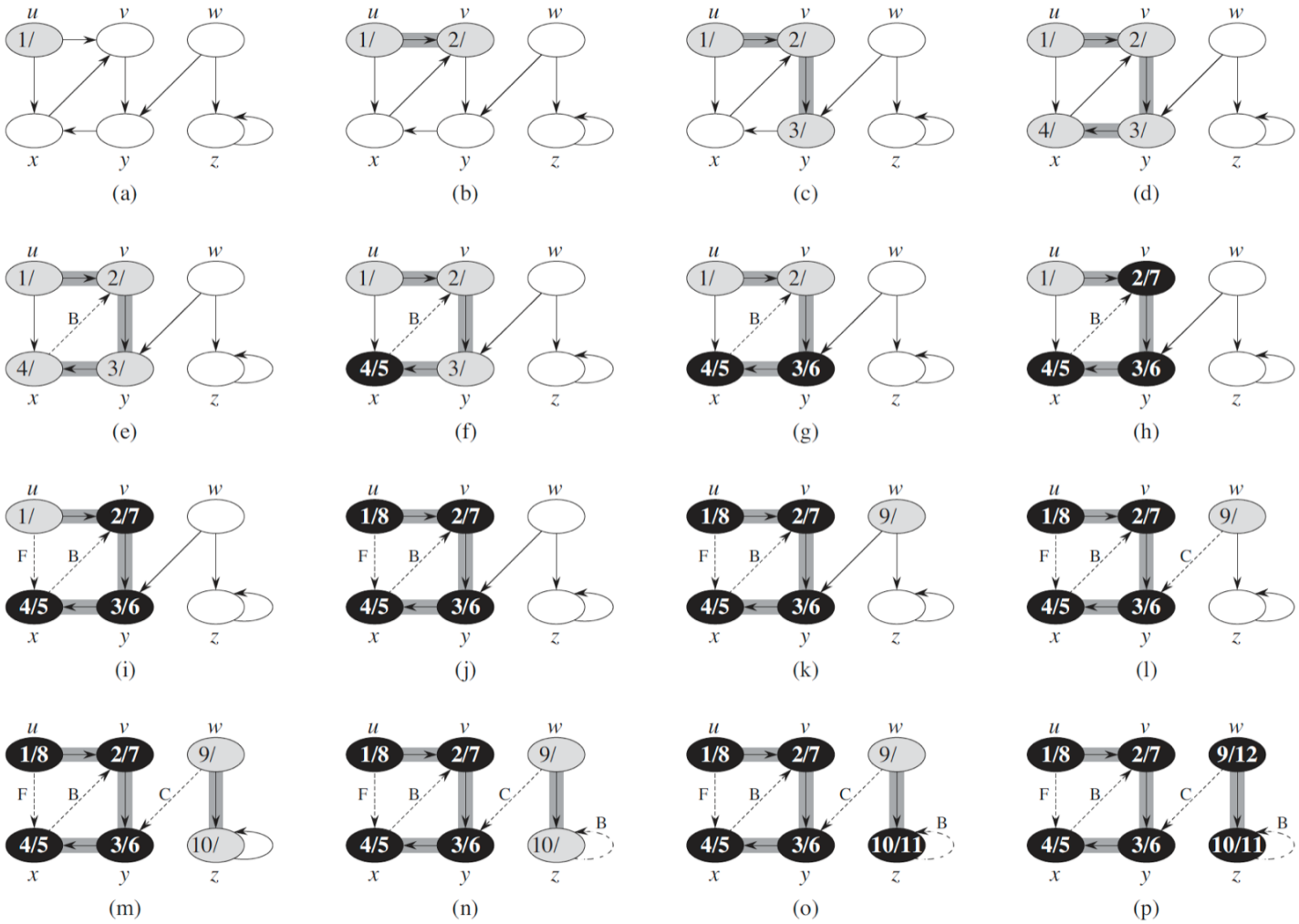
Spanning Trees

- BFS Trees
  - Shortest paths (in terms of nodes)
  - Testing bi-partitiness (on board)
    - Bipartite graph → no odd cycles (proof)

DFS Trees

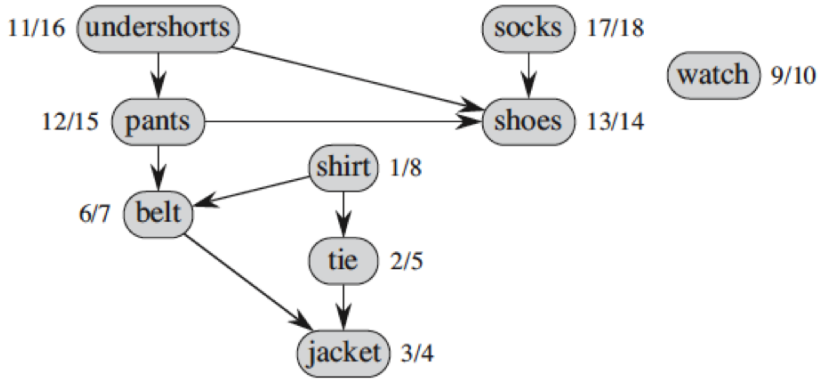
```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1          // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]    // explore edge (u, v)
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK          // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

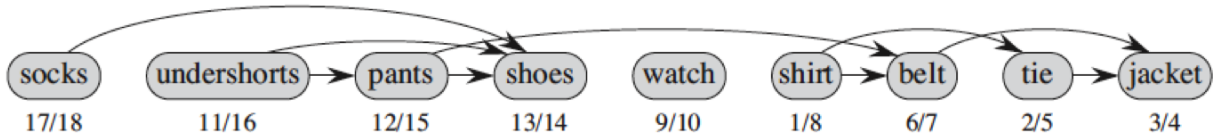
DFS Trees


u v w
1/ x
1/ 2/ x
1/ 2/ 3/ x
1/ 2/ 4/ 3/ x
B
4/5 3/ x
1/ 2/ 4/5 3/6 x
F
1/ 2/7 4/5 3/6 x
1/8 2/7 9/ 4/5 3/6 x
C
1/8 2/7 9/ 4/5 3/6 10/ z
10/11 z
9/12 z
y
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(j)
(k)
(l)
(m)
(n)
(o)
(p)

DFS

- Topological Sorting
- Directed Acyclic Graphs (DAGs)
- Linear sorting of a vertex such that:
  - If u→ v is an edge, u comes before v
- **Applications: task scheduling**





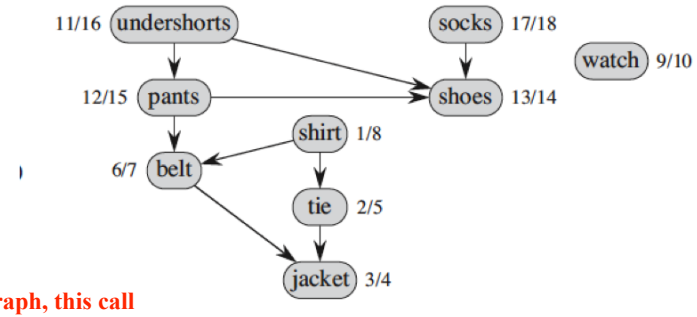
DFS

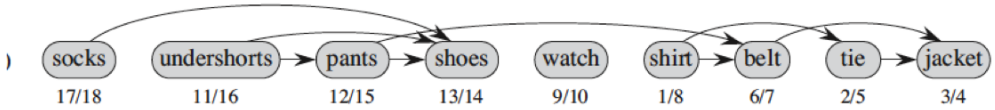
– Topological Sorting

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)
```

May not be connected graph, this call is for each tree, if it's connected graph, this function is only called once

```
DFS-VISIT(G, u)
1  time = time + 1
2  u.d = time
3  u.color = GRAY
4  for each v ∈ G.Adj[u]
5      if v.color == WHITE
6          v.π = u
7          DFS-VISIT(G, v)
8  u.color = BLACK
9  time = time + 1
10 u.f = time
```






```
TOPOLOGICAL-SORT(G)
1  call DFS(G) to compute finishing times v.f for each vertex v
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

Spanning Trees

- Minimum Spanning Trees
  - Electronic board problem
  - N points on the board
  - I can connect any node with any node
    - N-regular graph
  - Weight is the distance between points
  - Need to connect all points (no cycles needed) → tree would suffice
  - Tree of minimum total weight!!!

Spanning Trees

- Minimum Spanning Trees
  - Kruskal Algorithm
  - Prim's Algorithm
  - Same idea:
    - Iterative construction
    - At each step current subtree is a minimum spanning tree of the subgraph

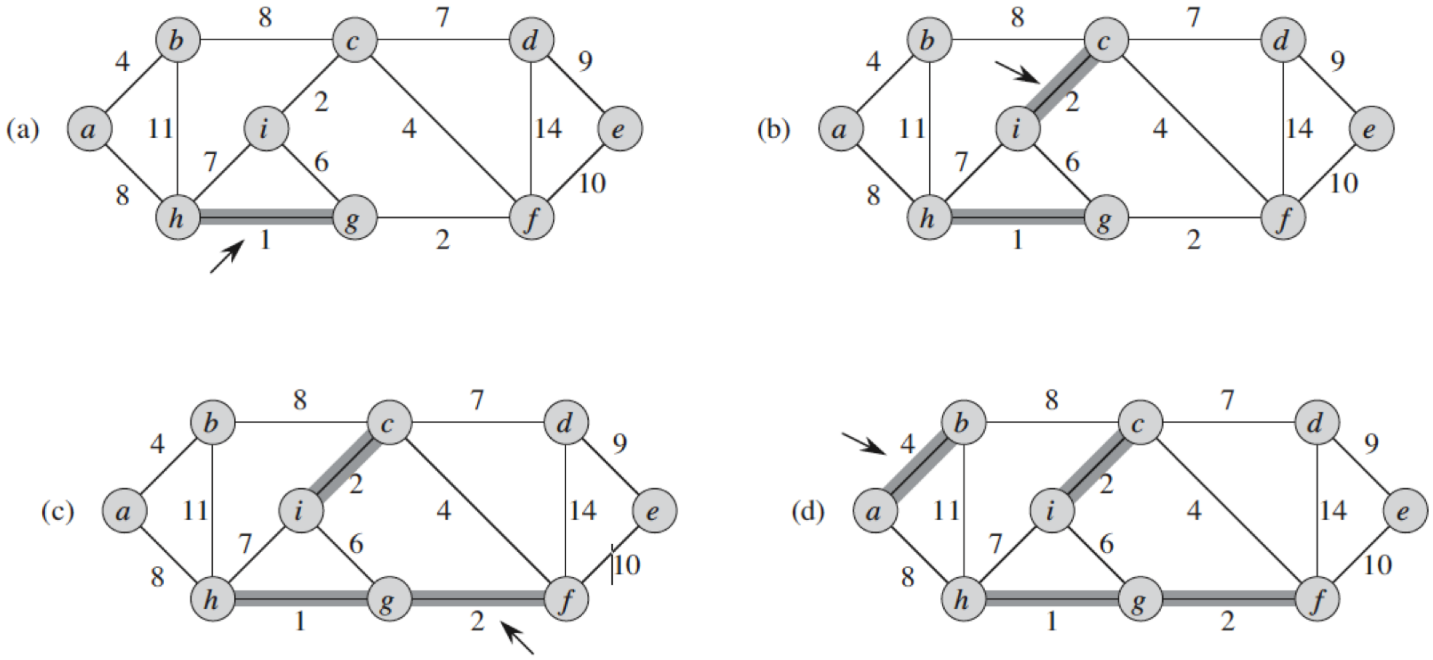
Spanning Trees

- Minimum Spanning Trees
  - At each step current subtree is a minimum spanning tree of the subgraph
  - If this property holds, will the final spanning tree be a minimum spanning tree?
  - Yes/Induction/Greedy

```
GENERIC-MST(G, w)
1  A = ∅
2  while A does not form a spanning tree
3      find an edge (u, v) that is safe for A
4      A = A ∪ {(u, v)}
5  return A
```

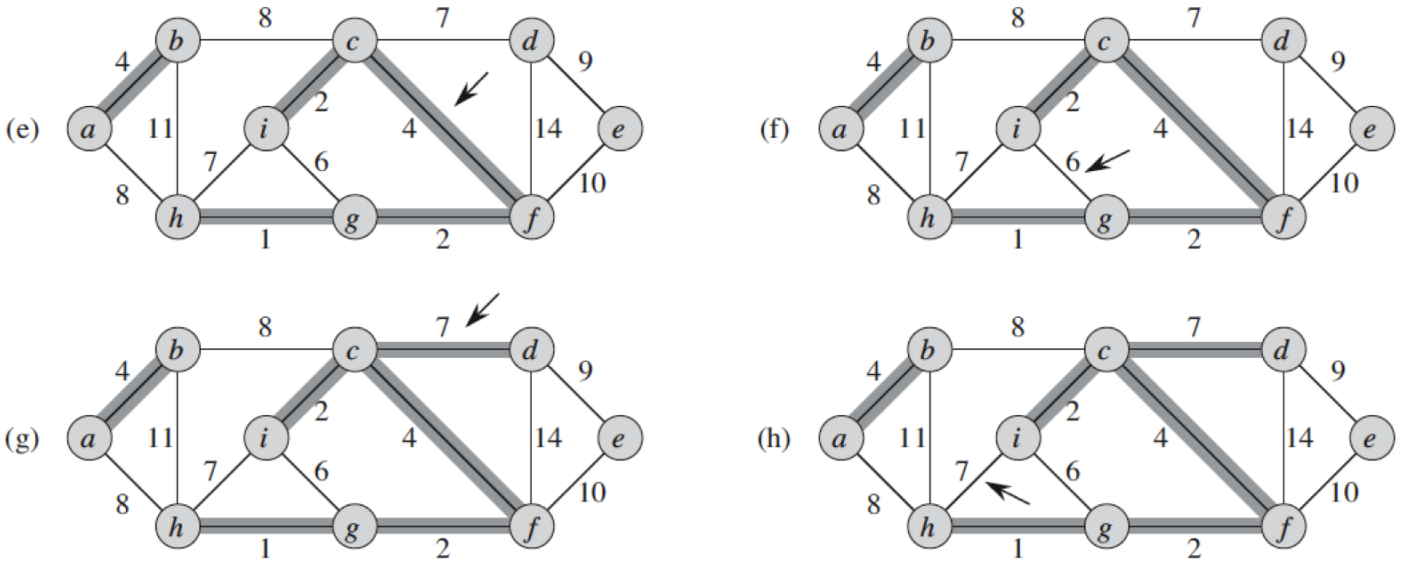
Spanning Trees

- Minimum Spanning Trees
  - Kruskal Algorithm



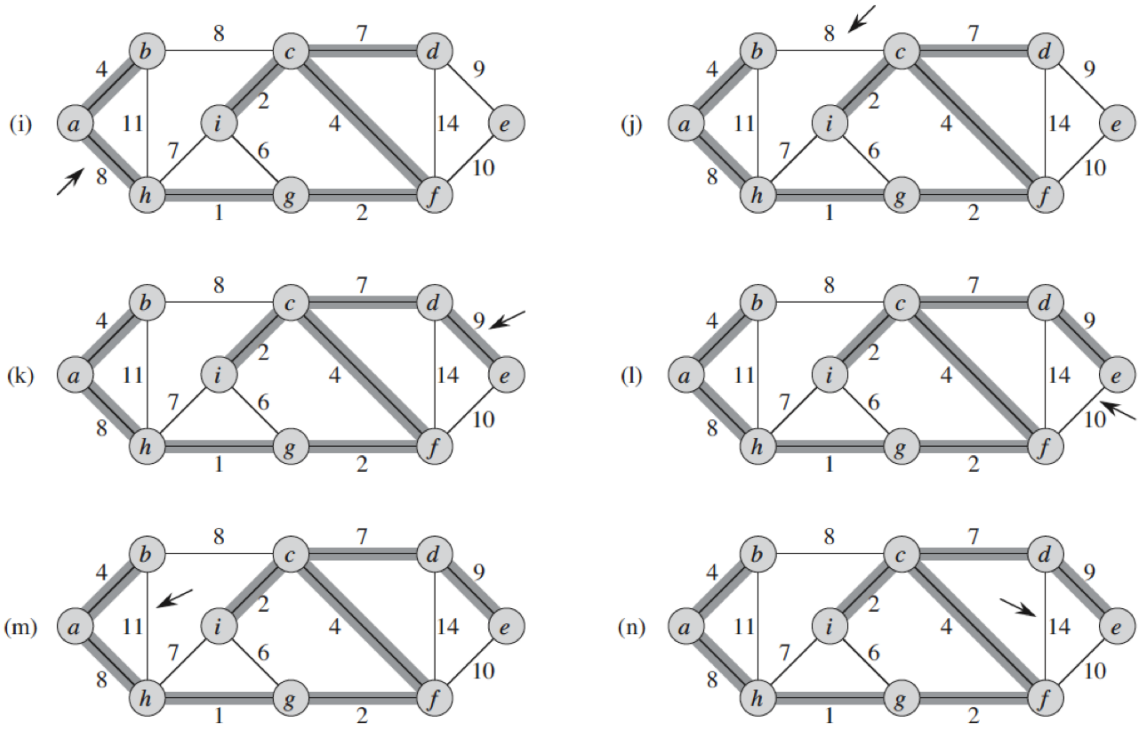
Spanning Trees

- Minimum Spanning Trees
  - Kruskal Algorithm



Spanning Trees

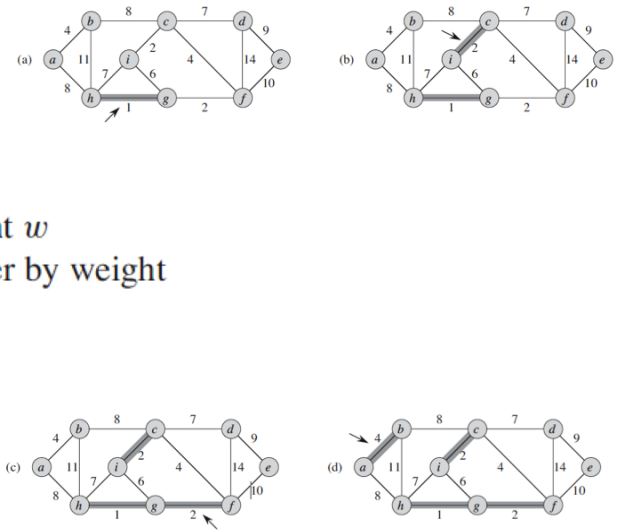
- Minimum Spanning Trees
  - Kruskal Algorithm



Spanning Trees

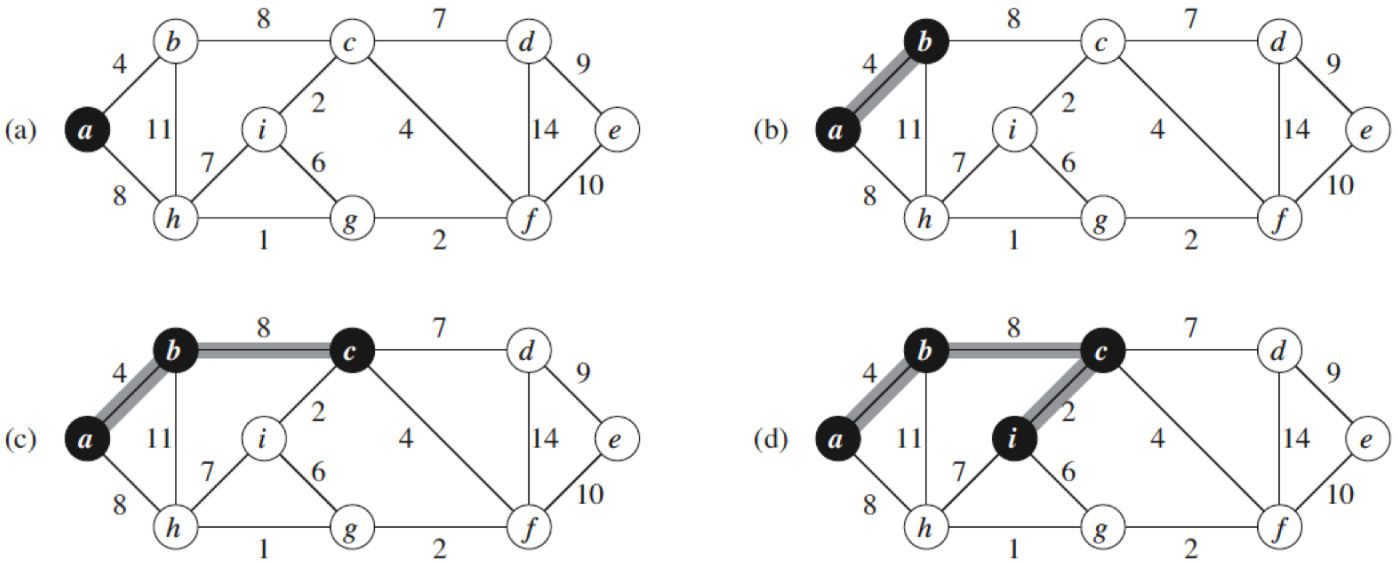
- Minimum Spanning Trees
  - Kruskal Algorithm

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3       MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6       if FIND-SET(u) ≠ FIND-SET(v)
7            A = A ∪ {(u, v)}
8            UNION(u, v)
9  return A
```



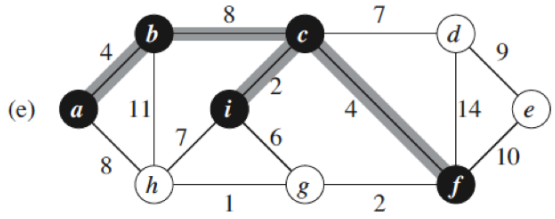
Spanning Trees

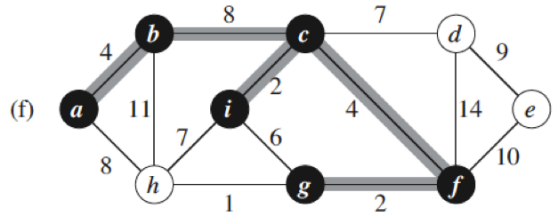
- Minimum Spanning Trees
  - Prim's Algorithm

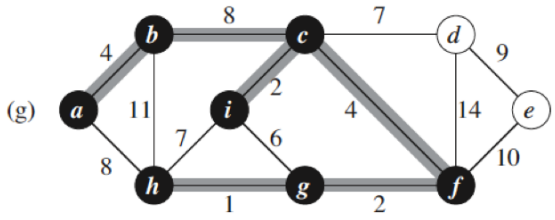


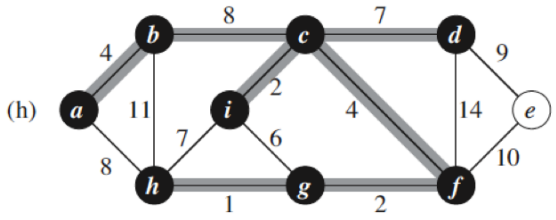
Spanning Trees

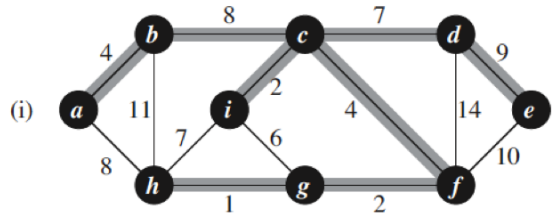
- Minimum Spanning Trees
  - Prim's Algorithm











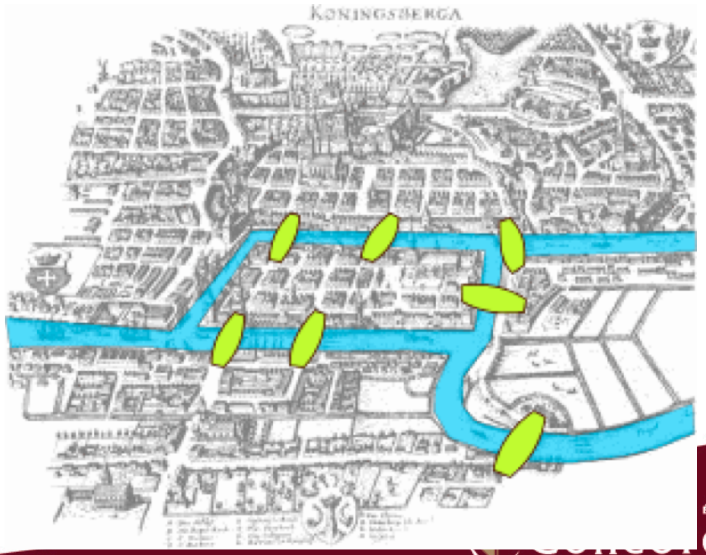
Spanning Trees

- Minimum Spanning Trees
  - Prim's Algorithm

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

Paths / Cycles

– Euler Paths
– Remember? The 7 bridges of **Königsberg**
– Topological sorting



Paths/Cycles

- Paths
- Simple Paths
- Cycles
- Simple Cycles
- Eulerian Paths/cycles
  - Path/cycle that visits all edges exactly once
- Hamiltonian Paths/Cycles
  - Path/cycle that visits all vertices exactly once (Other than first and last for cycles)

Path/cycle that visits all vertices

Edges at most once (may exist edge that has never been visited)

Paths/Cycles

- Hamiltonian Paths
- Hamiltonian Cycles (i.e. Travel salesman problem)
- Bridge to next classes: NP (Complete/Hard)
- Easy to say of there is one
  - Several characterizations:
  - Best one done here: Bondy–Chvátal theorem

Paths/Cycles

- **Václav (Vašek) Chvátal**
- Used to teach COMP6651

Paths/Cycles

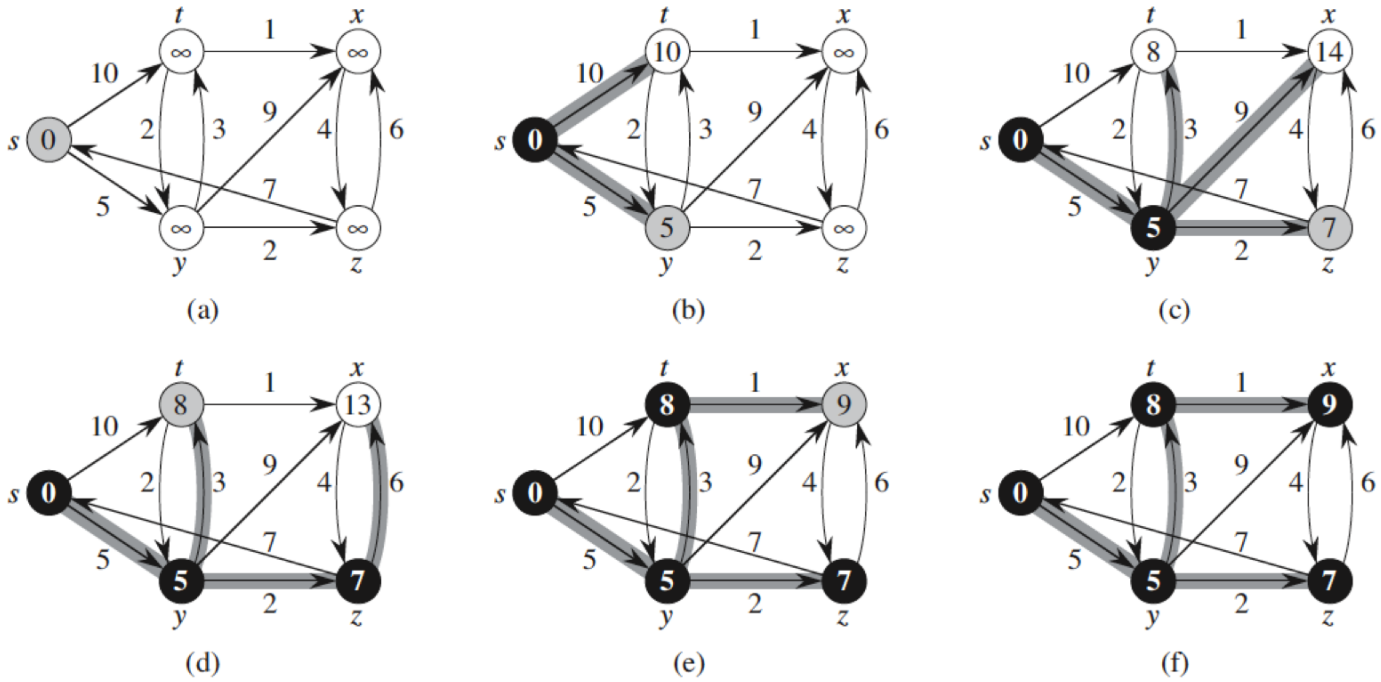
- Hamiltonian Paths
- Hamiltonian Cycles (i.e. Travel salesman problem)
- Very Difficult to find one
- In practice: regular graphs have a Hamiltonian cycle
- Have to minimize the travel cost
  - (i.e. Travel Salesman Problem)
- Implement for E4

Paths/Cycles

- Shortest Paths
- One node to all nodes
  - Dijkstra (positive weights)
  - Bellman-Ford
- Every node to every node
  - Floyd–Warshall

Paths/Cycles

– Dijkstra



Paths/Cycles

– Dijkstra

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

– O(V^2+E)

– Best: O(VlogV+E)