طراحي و تحليل الگوريتم ها

امیر لکی زاده استادیار گروه مهندسی کامپیوتر دانشگاه قم

Topological Sort Minimum Spanning Trees

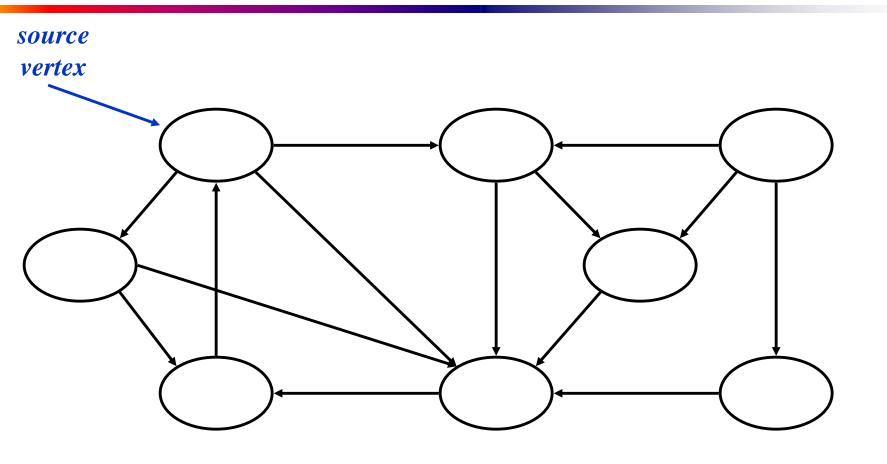
Review: Breadth-First Search

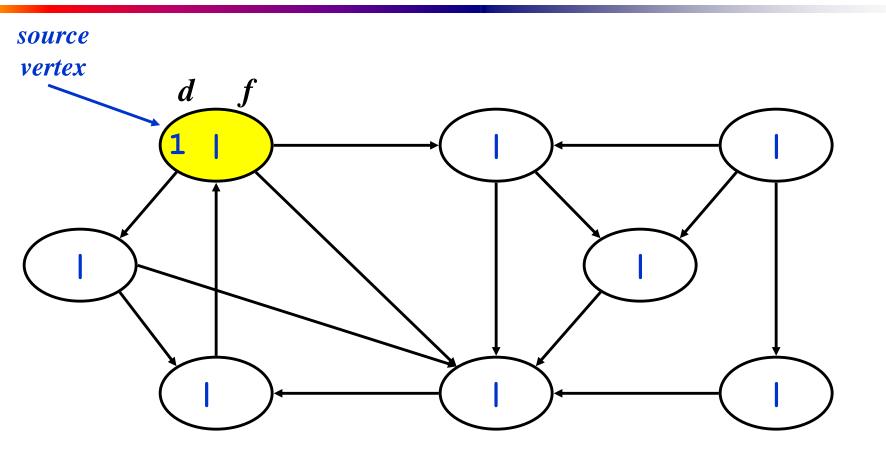
```
BFS(G, s) {
    initialize vertices:
    Q = \{s\}; // Q is a queue (duh); initialize to s
    while (Q not empty) {
        u = RemoveTop(Q);
        for each v \in u->adj {
             if (v->color == WHITE)
                 v->color = GREY;
                 v->d = u->d + 1; v->d represents depth in tree
                 v->p = u;
                                      v->p represents parent in tree
                 Enqueue(Q, v);
        u \rightarrow color = BLACK;
```

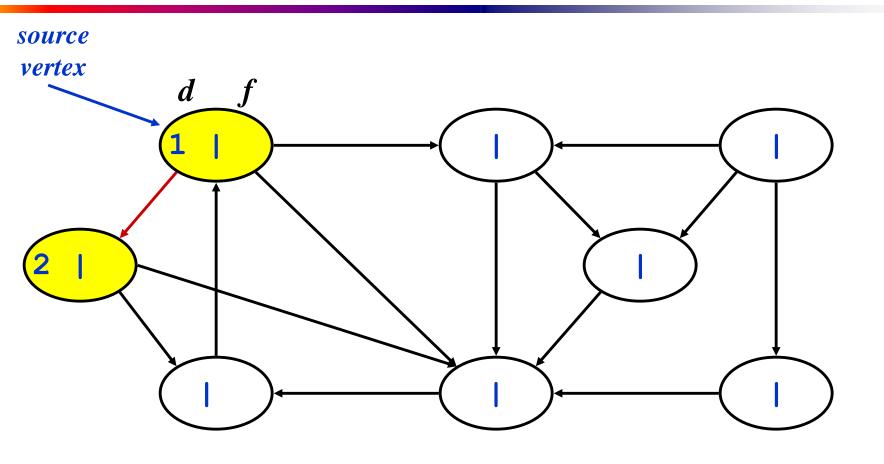
Review: DFS Code

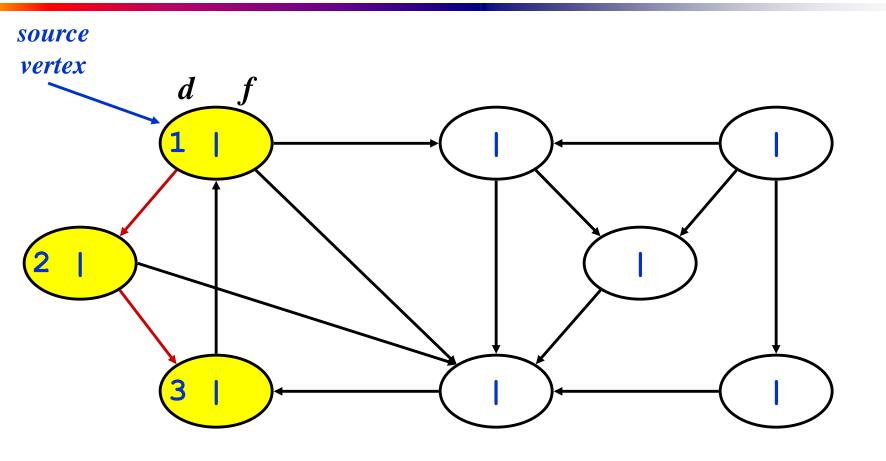
```
DFS(G)
   for each vertex u \in G->V
      u->color = WHITE;
   time = 0;
   for each vertex u \in G->V
      if (u->color == WHITE)
         DFS Visit(u);
```

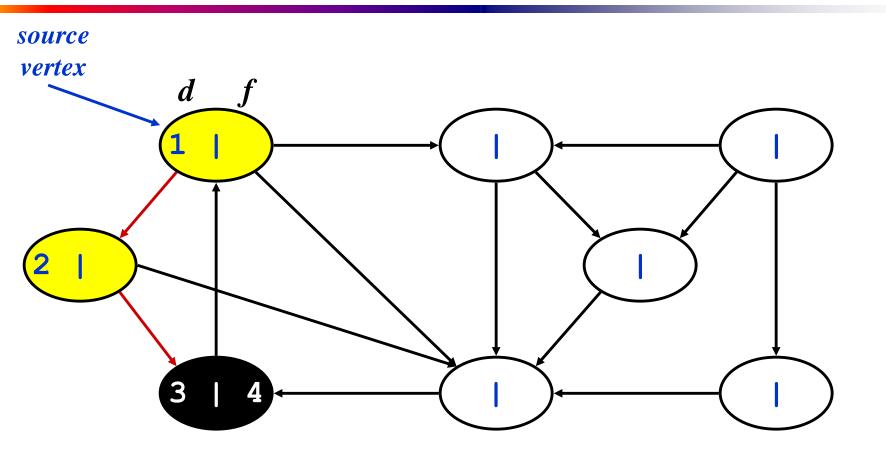
```
DFS Visit(u)
   u->color = YELLOW;
   time = time+1;
   u->d = time;
   for each v \in u-\lambda j[]
      if (v->color == WHITE)
         DFS Visit(v);
   u->color = BLACK;
   time = time+1;
   u->f = time;
```

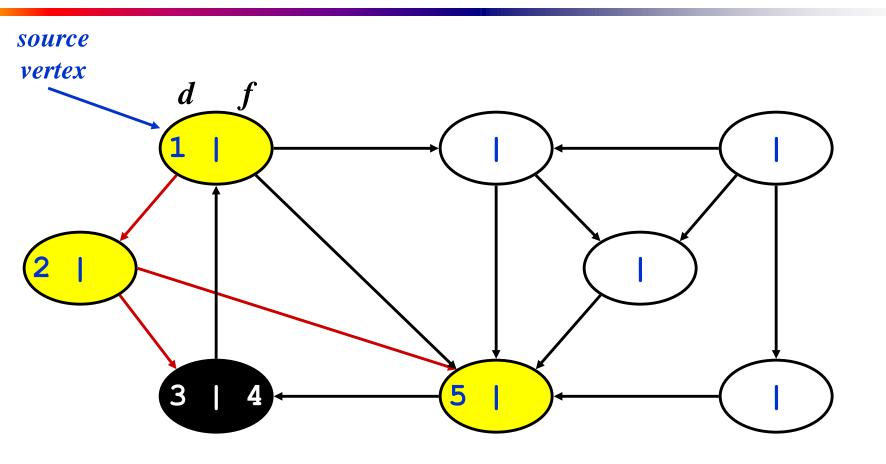


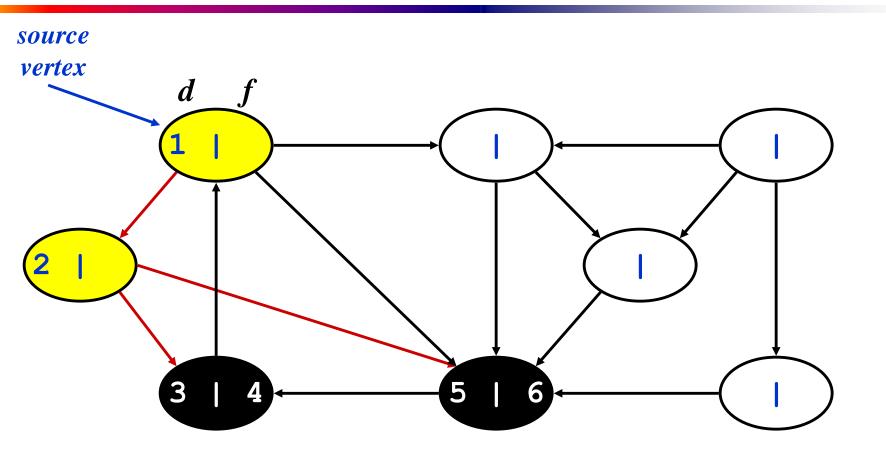


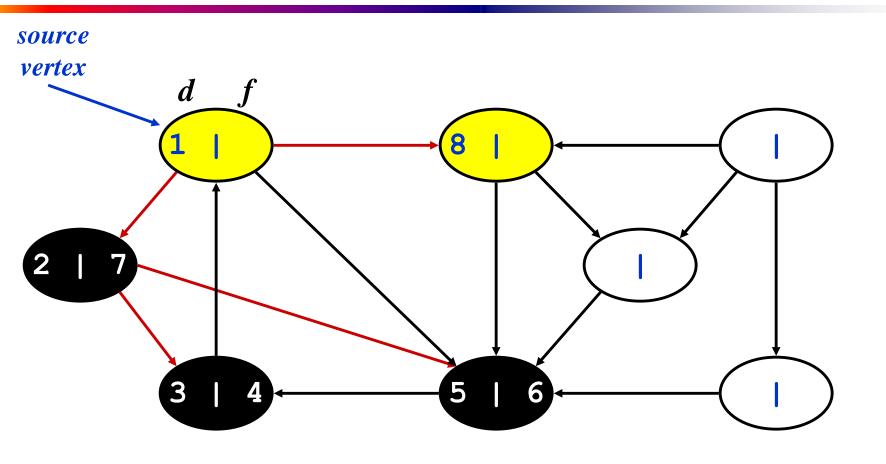


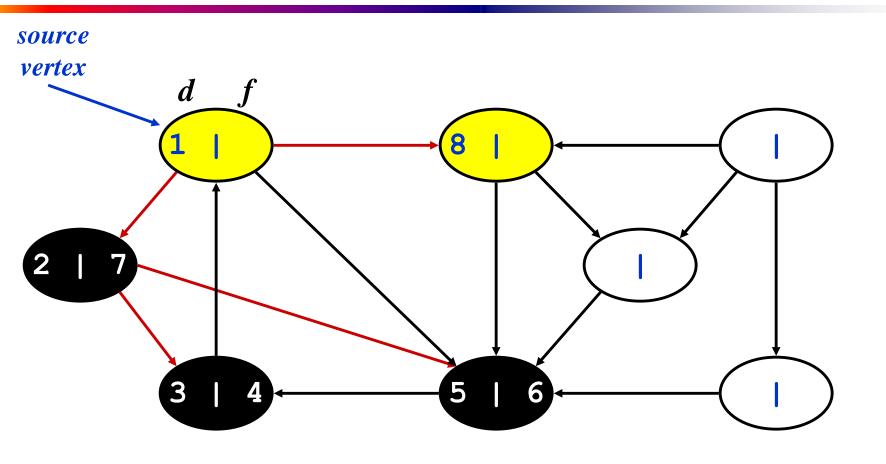


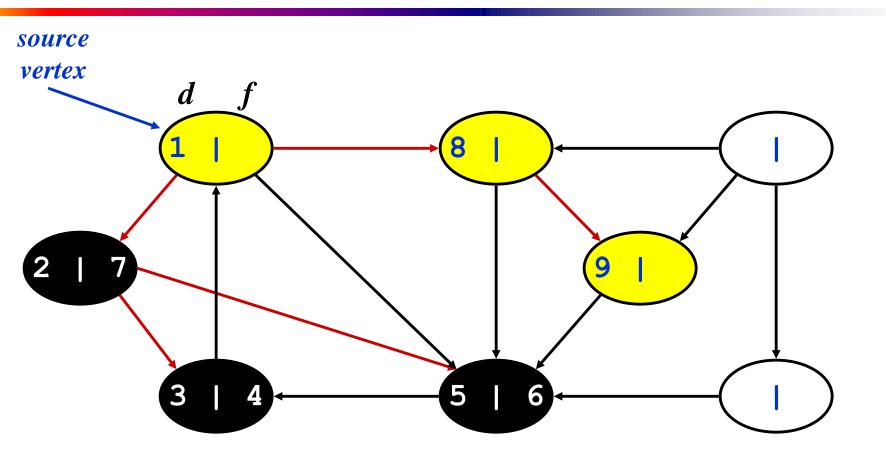


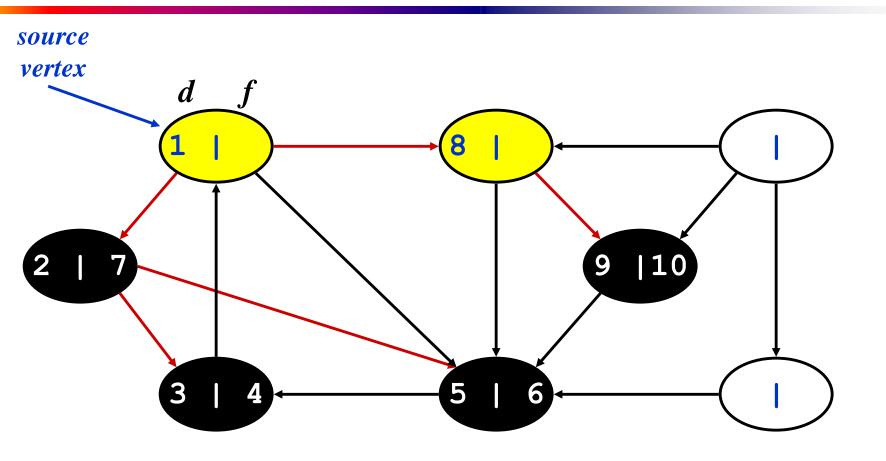


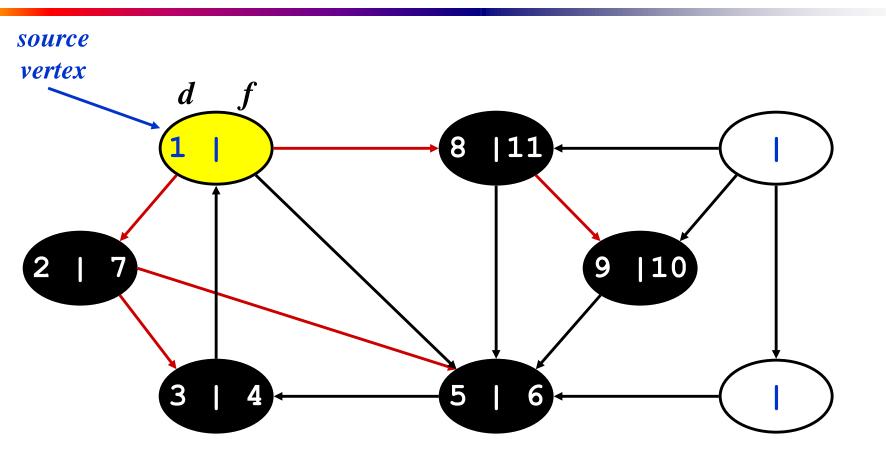


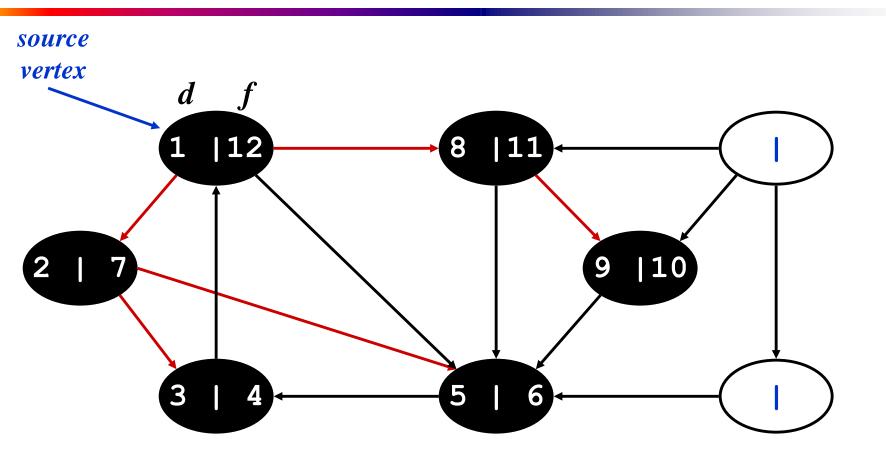


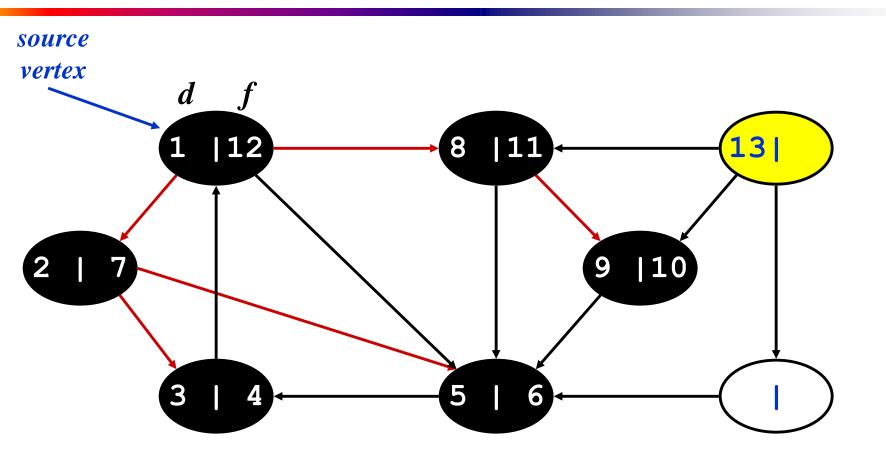


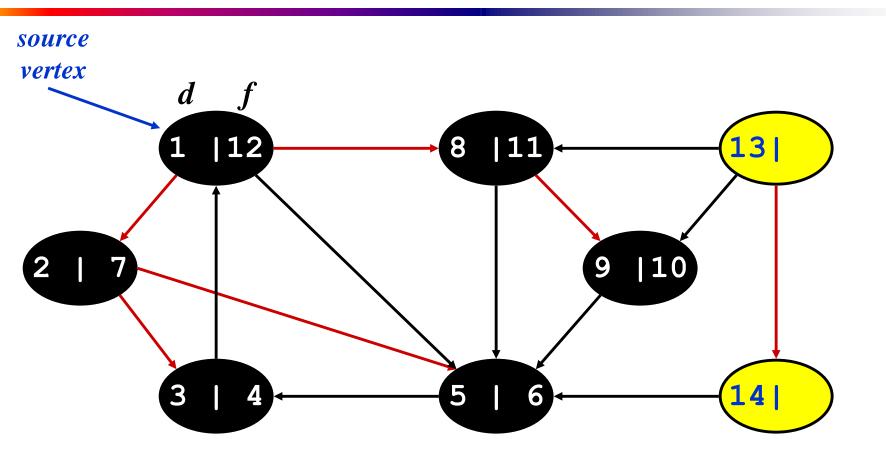


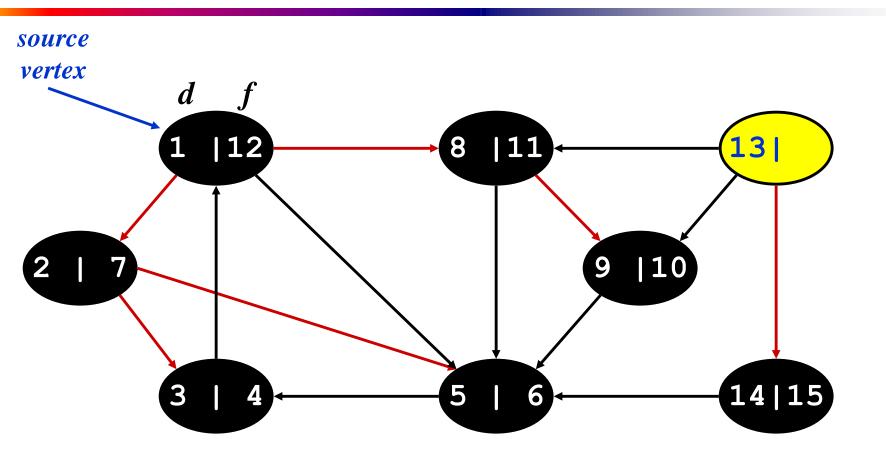


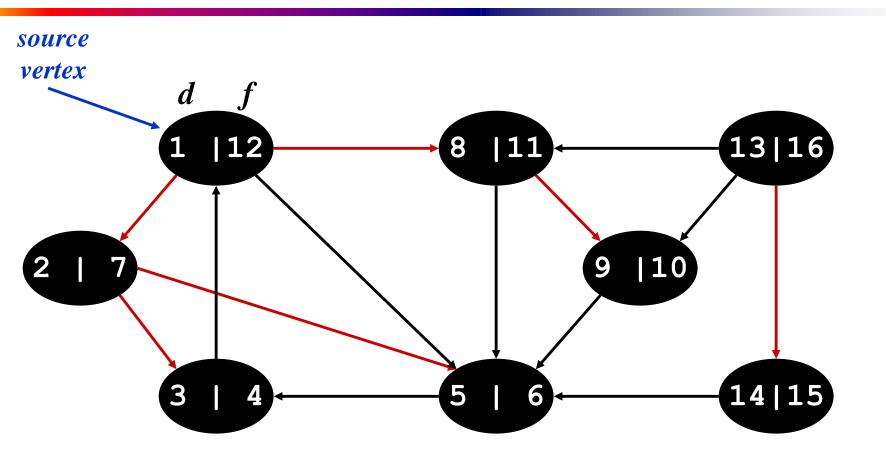








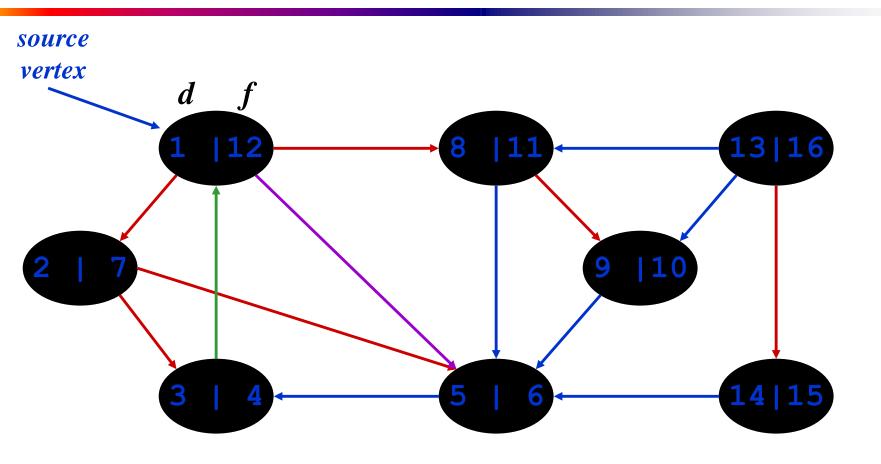




Review: Kinds Of Edges

- Thm: If G is undirected, a DFS produces only tree and back edges
- Thm: An undirected graph is *acyclic* iff a DFS yields no back edges
- Thus, can run DFS to find cycles

Review: Kinds of Edges



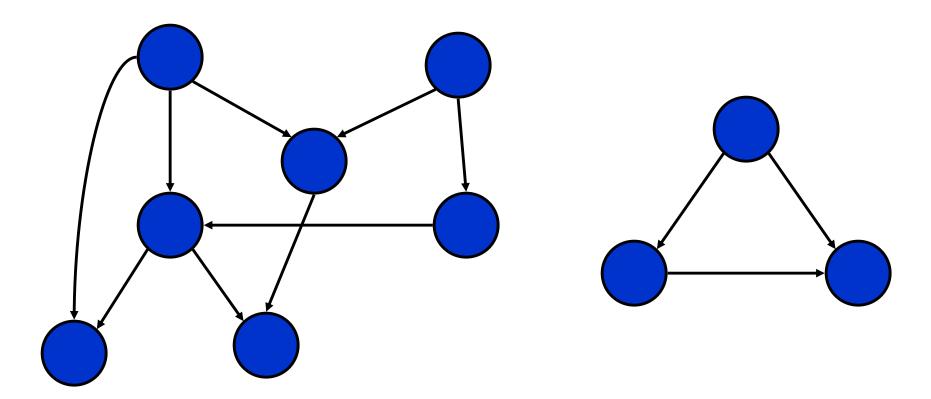
Tree edges Back edges Forward edges Cross edges

DFS And Cycles

- Running time: O(V+E)
- We can actually determine if cycles exist in O(V) time:
 - In an undirected acyclic forest, $|E| \le |V| 1$
 - So count the edges: if ever see |V| distinct edges, must have seen a back edge along the way
 - Why not just test if |E| < |V| and answer the question in constant time?

Directed Acyclic Graphs

• A directed acyclic graph or DAG is a directed graph with no directed cycles:



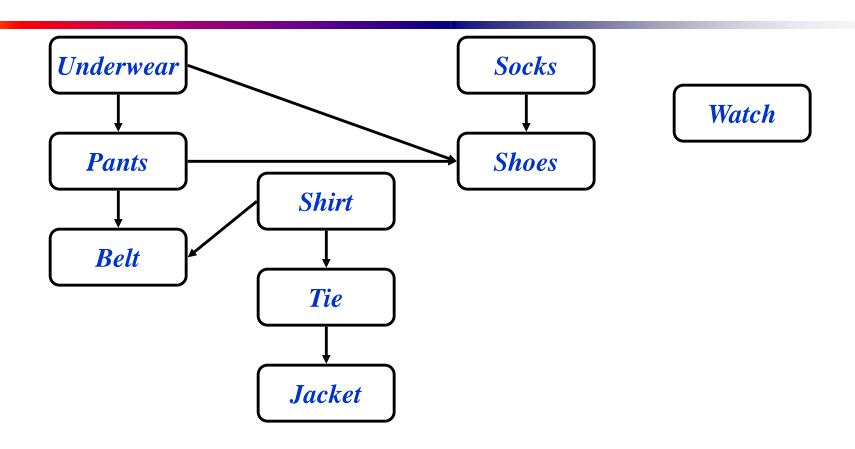
DFS and **DAGs**

- Argue that a directed graph G is acyclic iff a DFS of G yields no back edges:
 - Forward: if G is acyclic, will be no back edges
 - o Trivial: a back edge implies a cycle
 - Backward: if no back edges, G is acyclic
 - o Argue contrapositive: G has a cycle $\Rightarrow \exists$ a back edge
 - lack Let v be the vertex on the cycle first discovered, and u be the predecessor of v on the cycle
 - ◆ When *v* discovered, whole cycle is white
 - Must visit everything reachable from v before returning from DFS-Visit()
 - ♦ So path from $u \rightarrow v$ is yellow $\rightarrow yellow$, thus (u, v) is a back edge

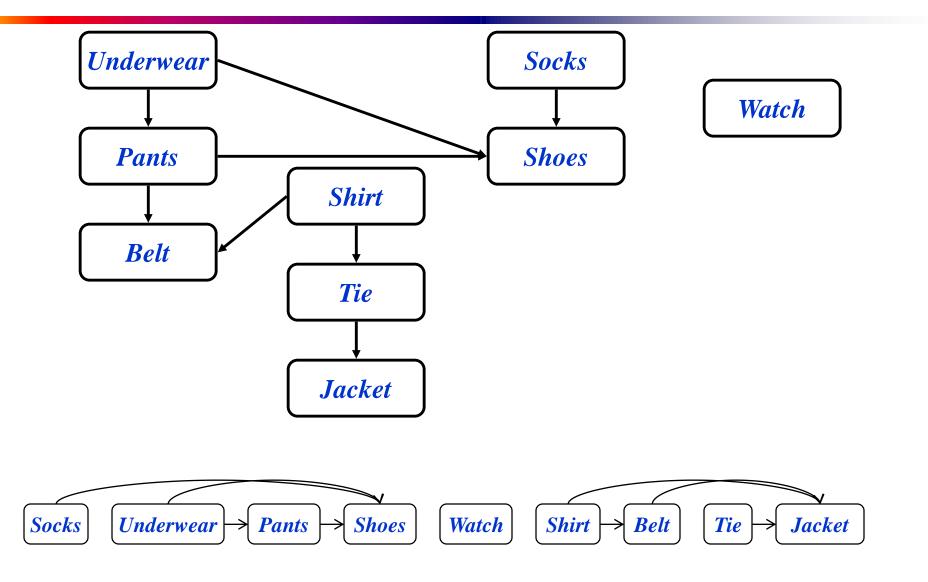
Topological Sort

- *Topological sort* of a DAG:
 - Linear ordering of all vertices in graph G such that vertex u comes before vertex v if edge $(u, v) \in G$
- Real-world example: getting dressed

Getting Dressed



Getting Dressed



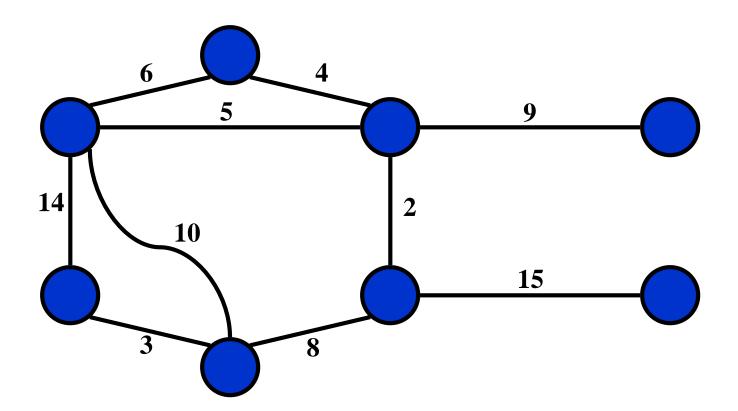
Topological Sort Algorithm

```
Topological-Sort()
   Run DFS
   When a vertex is finished, output it
   Vertices are output in reverse
     topological order
• Time: O(V+E)
• Correctness: Want to prove that
      (u,v) \in G \Rightarrow u \rightarrow f > v \rightarrow f
```

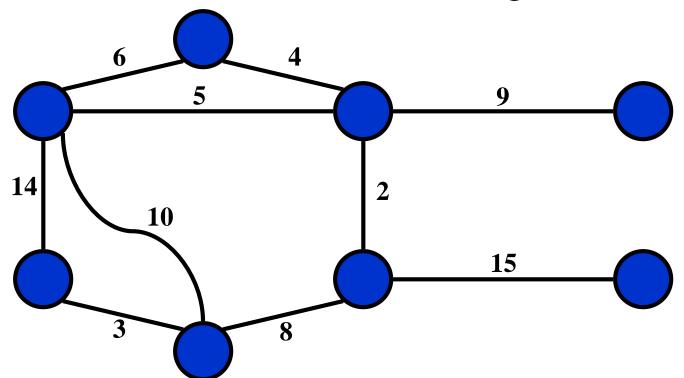
Correctness of Topological Sort

- Claim: $(u,v) \in G \Rightarrow u \rightarrow f > v \rightarrow f$
 - When (u,v) is explored, u is yellow
 - $\circ v = \text{yellow} \Rightarrow (u, v) \text{ is back edge. Contradiction } (Why?)$
 - o $v = \text{white} \Rightarrow v \text{ becomes descendent of } u \Rightarrow v \rightarrow f < u \rightarrow f$ (since must finish v before backtracking and finishing u)
 - $\circ v = \text{black} \Rightarrow v \text{ already finished} \Rightarrow v \rightarrow f < u \rightarrow f$

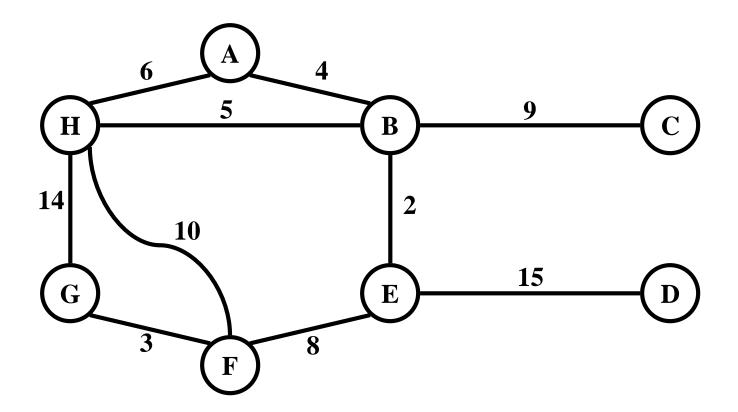
• Problem: given a connected, undirected, weighted graph:



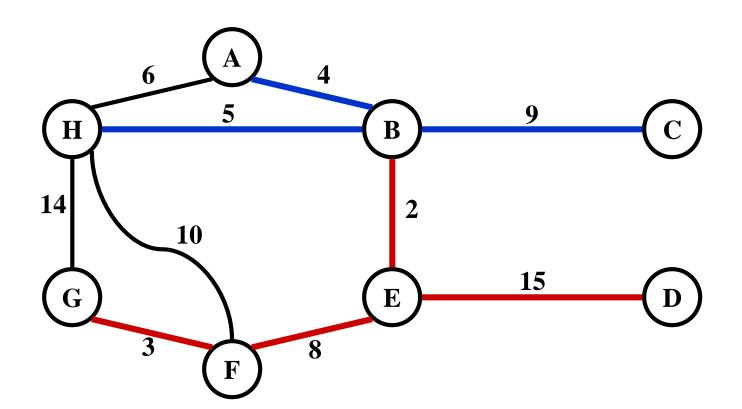
• Problem: given a connected, undirected, weighted graph, find a *spanning tree* using edges that minimize the total weight



• Which edges form the minimum spanning tree (MST) of the below graph?



• Answer:



- MSTs satisfy the *optimal substructure* property: an optimal tree is composed of optimal subtrees
 - Let T be an MST of G with an edge (u,v) in the middle
 - Removing (u,v) partitions T into two trees T_1 and T_2
 - Claim: T_1 is an MST of $G_1 = (V_1, E_1)$, and T_2 is an MST of $G_2 = (V_2, E_2)$ (Do V_1 and V_2 share vertices? Why?)
 - Proof: $w(T) = w(u,v) + w(T_1) + w(T_2)$ (There can't be a better tree than T_1 or T_2 , or T would be suboptimal)

- Thm:
 - Let T be MST of G, and let $A \subseteq T$ be subtree of T
 - Let (u,v) be min-weight edge connecting A to V-A
 - Then $(u,v) \in T$

Minimum Spanning Tree

- Thm:
 - Let T be MST of G, and let $A \subseteq T$ be subtree of T
 - Let (u,v) be min-weight edge connecting A to V-A
 - Then $(u,v) \in T$
- Proof: in book (see Thm 24.1)

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
         key[u] = \infty;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                   10
                                                     15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q); Run on example graph
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                  p[v] = u;
                  key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                    10
                                                      15
    key[r] = 0;
    p[r] = NULL;
                                      \infty
    while (Q not empty)
         u = ExtractMin(Q); Run on example graph
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
     for each u \in Q
                            14
         key[u] = \infty;
                                    10
                                                       15
    key[r] = 0;
    p[r] = NULL;
                                      \infty
    while (Q not empty)
                                Pick a start vertex r
         u = ExtractMin(Q);
          for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
     for each u \in Q
                            14
         key[u] = \infty;
                                     10
                                                       15
    key[r] = 0;
    p[r] = NULL;
                                       \infty
    while (Q not empty)
         u = ExtractMin(Q); Red vertices have been removed from Q
          for each v \in Adj[u]
               if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
     for each u \in Q
                            14
         key[u] = \infty;
                                    10
                                                       15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q); Red arrows indicate parent pointers
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                    10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                    10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                    10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                   10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                   10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                   10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                   10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                   10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                   10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                   10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                    10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                    10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                    10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                    10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
                           14
         key[u] = \infty;
                                   10
                                                      15
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                   p[v] = u;
                   key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
         key[u] = \infty;
    key[r] = 0;
                       What is the hidden cost in this code?
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                  p[v] = u;
                  key[v] = w(u,v);
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
         key[u] = \infty;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
              if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                  p[v] = u;
                  DecreaseKey(v, w(u,v));
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u \in Q
         key[u] = \infty; How often is ExtractMin() called?
    key[r] = 0;
                       How often is DecreaseKey() called?
    p[r] = NULL;
    while (Q not empty)
         u = ExtractMin(Q);
         for each v \in Adj[u]
             if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                  p[v] = u;
                  DecreaseKey(v, w(u,v));
```

```
MST-Prim(G, w, r)
    Q = V[G];
                         What will be the running time?
    for each u \in Q
        key[u] = \infty; A: Depends on queue
    key[r] = 0;
                           binary heap: O(E lg V)
    p[r] = NULL;
                           Fibonacci heap: O(V \lg V + E)
    while (Q not empty)
        u = ExtractMin(Q);
        for each v \in Adj[u]
             if (v \in Q \text{ and } w(u,v) < \text{key}[v])
                 p[v] = u;
                 key[v] = w(u,v);
```