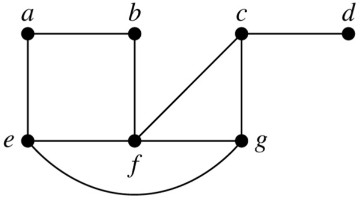
Discrete Mathematics: Lecture 15

- Last time:
  - Chap 11.2: Applications of trees
- Today:
  - Chap 11.4 Spanning trees
  - Chap 11.5 Minimum spanning trees

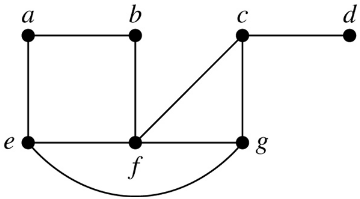
11.4: Motivating problem

- Consider a system of roads.
- In winter, to keep a road open, we have to frequently plow it.
- We want to plow the fewest roads so that there are cleared roads between any two towns.
- How can this be done?
- Solution: find a connected subgraph with minimum number of edges containing all vertices of the original graph.



Spanning trees

- Definition: Let G be a simple graph. A spanning tree of G is a subgraph of G that is a tree containing every vertex of G.
- Example: Find a spanning tree for the following simple graph:



- Th'm 1: A simple graph is connected iff it has a spanning tree.

Constructing spanning trees

- Proof of Theorem 1 gives an algorithm for finding spanning trees by removing edges from simple circuits. But the algorithm is inefficient.
- Alternatively, spanning trees can be constructed by successively adding edges.
- We will discuss two such algorithms: depth-first search and breadth-first search.

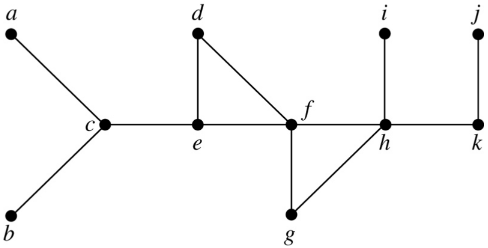
Depth-first search

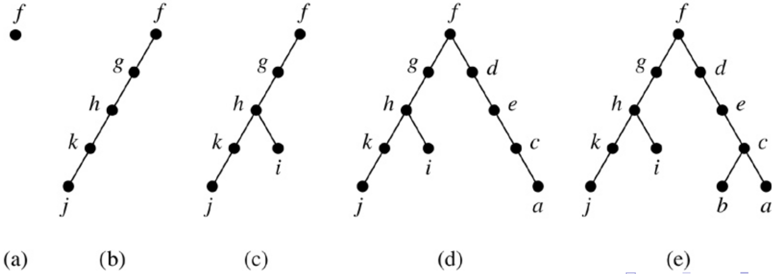
We will form a rooted tree, its underlying undirected graph will be the spanning tree.

1. Arbitrarily select a vertex v as the root.
2. Form a path starting at v by successively adding vertices not already in the path. Continue this process as long as possible.
3. If all vertices have been included, we get a spanning tree.
4. Otherwise, move back to the next to last vertex v in the path, go to Step 2.

Depth-first search is also called backtracking.

An example





A recursive algorithm for depth-first search

procedure DFS(G: connected graph with vertices $v_1, v_2, \ldots, v_n$)
T := tree consisting only of v_1
visit(v_1)

procedure visit(v : vertex of G)
for each vertex w adjacent to v and not yet in T
    add vertex w and edge $\{v, w\}$ to T
    visit w

The computational complexity of depth-first search

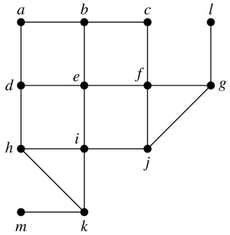
- For each vertex v, $visit(v)$ is called when v is first encountered in the search and it is not called again.
- For each edge $\{v, w\}$, we consider it when we $visit(v)$ and when we $visit(w)$.
- Thus we examine each edge at most twice.
- Thus $O(e)$, or $O(n^2)$, where e is the number of edges, and n is the number of vertices. Here we make use of $e \leq n(n-1)/2$.

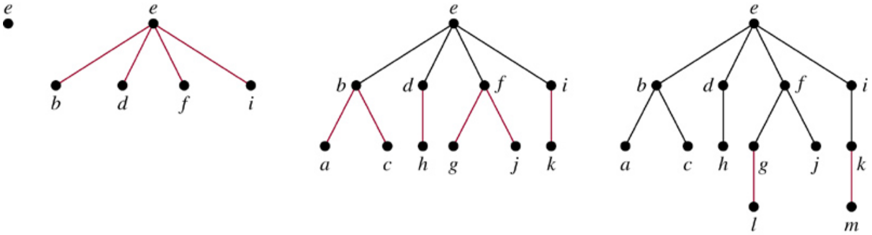
Breadth-first search

Again, we will form a rooted tree, its underlying undirected graph will be the spanning tree.

1. Arbitrarily select a vertex as the root.
2. Add all edges incident to the root. The new vertices become vertices at level 1.
3. For each vertex at level 1, add each edge incident to it as long as this does not produce a simple circuit. The new vertices become vertices at level 2.
4. Repeat this process until all vertices have been included.

An example





A recursive algorithm for breadth-first search

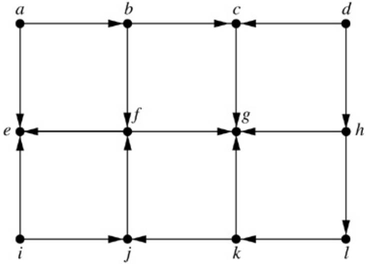
```
procedure BFS(G: connected graph with vertices v_1, v_2, ..., v_n)
T := tree consisting only of v_1
L := empty list // L represents the list of unprocessed vertices
put v_1 in L
while L is not empty
    remove the first vertex v from L
    for each neighbor w of v
        if w is not in L and not in T then
            add w to the end of L
            add w and edge {v, w} to T
```

The computational complexity of breadth-first search

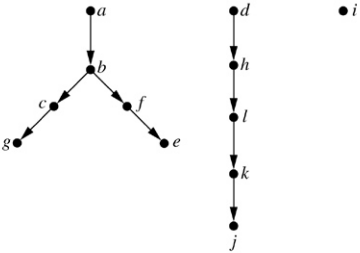
- For each vertex v, we examine all vertices adjacent to v.
- Thus we examine each edge at most twice.
- Thus $O(e)$, or $O(n^2)$.

Search in directed graphs

- We can easily adapt DFS and BFS for directed graphs.
- We can add an edge only when it is directed away from the vertex that is being visited and to a vertex not yet added.
- If there does not exist an edge between a vertex already added and a vertex not yet added, add a vertex not yet added, which becomes the root of a new tree.
- The output is a spanning forest.



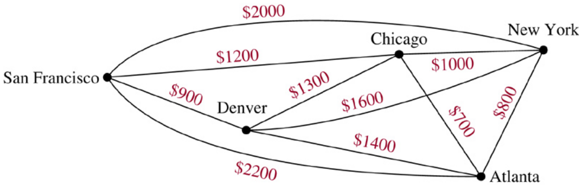
(a)



(b)

11.5: Motivating problem

- A company plans to build a communication network connecting its 5 computer centers
- Any pair of these centers can be linked with a leased telephone line
- Which links should be made to ensure that there is a path between any two centers so that the total cost of the network is minimized?
- We can solve this problem by finding a spanning tree so that the sum of the weights of the edges of the tree is minimized



Minimum spanning trees (最小生成树)

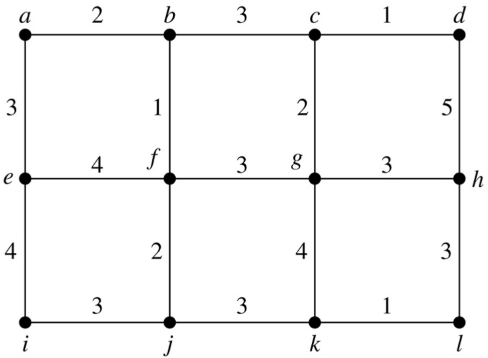
- Definition： A minimum spanning tree in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.
- Two algorithms for constructing minimum spanning trees: Prim's and Kruskal's algorithms

ALGORITHM 1 Prim's Algorithm.

procedure *Prim*(G: weighted connected undirected graph with n vertices)
T := a minimum-weight edge
for $i := 1$ **to** $n - 2$
    e := an edge of minimum weight incident to a vertex in T and not forming a simple circuit in T if added to T
    $T := T$ with e added
return T {T is a minimum spanning tree of G}

Example: communication network

Another example



ALGORITHM 2 Kruskal's Algorithm.

procedure *Kruskal*(G: weighted connected undirected graph with n vertices)
T := empty graph
for $i := 1$ **to** $n - 1$
    e := any edge in G with smallest weight that does not form a simple circuit
        when added to T
    $T := T$ with e added
return T {T is a minimum spanning tree of G}

Example 2

The computational complexity

- For a graph with v vertices and e edges
  - Prim's algorithm can be carried out using $O(e \log v)$ operations
  - Kruskal's algorithm can be carried out using $O(e \log e)$ operations
- Thus it is preferable to use Kruskal's algorithm when the graph is sparse