Minimal Spanning Trees

Reading. Weiss, sec. 24.2.2

Real quotes from a Dilbert-quotes contest:

"As of tomorrow, employees will be able to access the building only using individual security cards. Pictures will be taken next Wednesday and employees will receive their cards in two weeks." (Microsoft Corp. in Redmond, WA)

"What I need is an exact list of specific unknown problems we might encounter." (Lykes Lines Shipping)

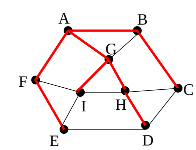
"E-mail is not to be used to pass on information or data. It should be used only for company business." (Electric Boat Company)

"This project is so important, we can't let things that are more important interfere with it." (United Parcel Service)

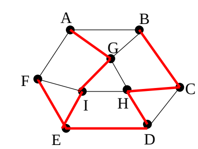
Spanning Tree

Assume you have a directed or undirected graph.

Spanning tree of a graph is tree such that
- Tree has same set of nodes
- Set of tree edges is a subset of graph edges



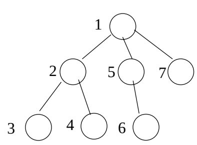
Tree



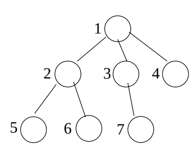
Another tree

BFS and DFS Walk

Pre-order traversal gives a **depth-first search** (DFS) of a tree. **Breadth-first search (FBS)** as in second diagram



DFS

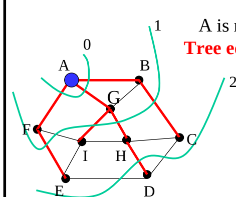


BFS

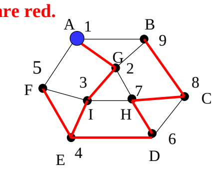
Nodes numbered in the order visited

Two spanning trees of the same graph

A is root of tree.
Tree edges are red.



Breadth-first Spanning Tree



Depth-first Spanning Tree

To build a spanning tree: (1) start with one node, A, as root. (2) At each step, add to tree one edge from a node in tree to a node that is not yet in the tree.

Build a DFS spanning tree (V, E). Use a stack

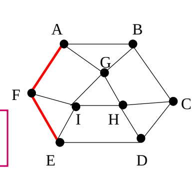
V = {A}; E = {}; s = (A,F), (A,G), (A,B) // s: **stack** of edges

Step 1: Take (A,F) off stack and add to E;
push onto s edges leaving F.

V = {A,F}; E = {(A,F)}; s = (F,E), (F,I), (F,A) (A,G), (A,B)

Step 2: Take (F,E) off s and add to E;
push onto s edges leaving E.

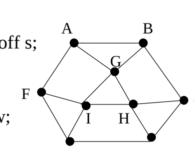
V = {A,F,E}; E = {(A,F),(F,E)};
s = (E,F), (E,I), (E,D),
(F,I), (F,A) (A,G), (A,B)



After taking an edge (v,w) off stack: if w already in V, don't do anything

Build a DFS spanning tree (V, E)

```
V = {A};  E = {};                //start off with one-node tree
s = stack of edges to neighbors of A;  // s is a stack of edges
/** invariant:(V,E) is a tree.
    For all edges (v,w) in s: v is in V and (v,w) not in E.
    Any node in graph that is not in V is reachable from the
       end node of some edge in s. **/
while (s is not empty do) {
   (v, w) = pop(s); //  Take top edge (v,w) off s;
   if (w is not in V) {
     Add w to V; add (v,w) to E;
     Push onto s all edges with start node w;
   }
}
```



Build a BFS spanning tree (V, E). Use a queue.

V = {A}; E= {}; s = (A,F), (A,G), (A,B) // s: **queue** of edges

Step 1: Take (A,F) off s and add to E;

push onto s edges leaving F.

V = {A,F}; E = {(A,F)}; s = (A,G), (A,B), (F,E), (F,I), (F,A)

Step 2: Take (A,G) off s and add to E;

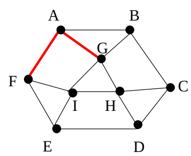
push onto s edges leaving G.

V = {A,F,G}; E = {(A,F),(A,G)};

s = (A,B), (F,E), (F,I), (F,A),

(G,I), (G,H), (G,B), (G,A)

After taking an edge (v,w) off queue:
if w already in V, don't do anything



Build a BFS spanning tree (V, E)

V = {A}; E = {}; //start off with one-node tree

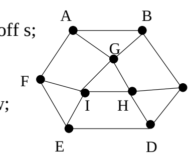
s = **queue** of edges to neighbors of A; // s is a **queue** of edges

/** invariant:(V,E) is a tree.

For all edges (v,w) in s: v is in V and (v,w) not in E.

Any node in graph that is not in V is reachable from the end node of some edge is s. **/

```
while (s is not empty do) {
   (v, w) = pop(s); //  Take top edge (v,w) off s;
   if (w is not in V) {
      Add w to V; add (v,w) to E;
      Push onto s all edges with start node w;
   }
}
```

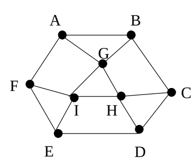


Build a DFS or BFS spanning tree (V, E)

Building a DFS spanning tree and building a BFS spanning tree are essentially the same algorithm.

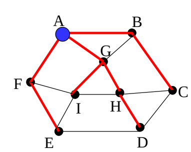
DFS algorithm uses a stack of edges to process

BFS algorithm uses a stack of edges to process



Property 1 of spanning trees

- Graph: G = (V,E)
- Spanning tree: T = (V,E_T,R)
- Choose any edge: c = (u,v) in G but not in T
- There is a simple cycle containing only edge c and edges in spanning tree.
- Proof: Let w be the first node in common to paths from u to root of tree and from v to root of tree. The paths u→v, v→w,w→u can be catenated to form the desired cycle.



edge (I,H):
w is node G
simple cycle is (I,H,G,I)
edge (H,C):
w is node A
simple cycle is (H,C,B,A,G,H)

Useful lemma

- **In any tree T = (V,E), |E|=|V| – 1**

For all n>0, P(n) holds, where

P(n) for a tree with n (>0) nodes: |E| = |V| – 1

Proof by induction on n

* n = 1. tree with node has 0 edges. 0 = 1 - 1.
* Assume P(n) for some n, 0 < n. Consider a tree S=(V_S, E_S) with n+1 nodes. S has a leaf. Remove 1 leaf (and the edge to it) to give a tree T with n nodes. By inductive assumption, P(n), $|E_T| = |V_T|$-1. Since $|E_S| = |E_T|$+1 and $|V_S|=|V_T|$+1, the result follows.

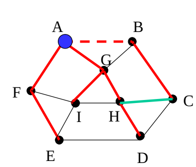
- **An undirected graph G = (V,E) is a tree** iff

(1) it is connected

(2) |E| = |V| – 1

Property 2 of spanning trees

- Graph: G = (V,E)
- Spanning tree: T = (V,E_T,R)
- Choose any edge: c = (u,v) in G but not in T
- There is a simple cycle Y containing only edge c and edges in spanning tree. Moreover, inserting edge c into T and deleting any edge (s→t) in Y gives another spanning tree T1.



edge (H,C):
simple cycle is (H,C,B,A,G,H)
adding (H,C) to T and deleting (A,B)
gives another spanning tree

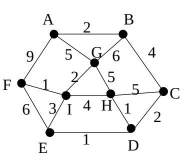
Proof of Property 2

- T1 is connected.
  - Otherwise, assume node a is not reachable from node b in T1. In T, there is a path from b to a that contains edge (s→t). In this path, replace edge (s→t) by the path in T1 obtained by deleting (s→t) from the cycle Y, which gives a path from b to a.
- In T1, numbers of edges = number of nodes –1
  - Proof: by construction of T1 and fact that T is a tree
- Therefore, from lemma, T1 is a tree.

Weighted Spanning Trees

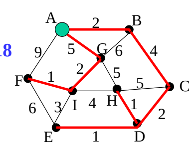
- Assume an undirected graph G = (V,E) with weights on each edge
- Spanning tree of graph G is tree $T = (V,E_T)$
  - Tree has same set of nodes
  - All tree edges are graph edges
  - Weight of spanning tree = sum of tree edge weights
- Minimal Spanning Tree (MST)
  - Any spanning tree whose weight is minimal
  - A graph can have several MST's
  - Applications: phone network design etc.

Example



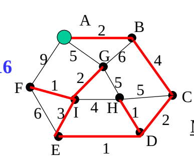
Graph

weight 18



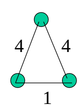
SSSP tree (nodes in order chosen by Dijkstra's shortest path algoritm

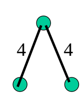
weight 16



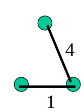
Minimal spanning tree

Caution: in general, SSSP tree is not MST





SSSP Tree



MSP

- Intuition:
  - SSSP: fixed start node
  - MST: at any point in construction, we have a bunch of nodes that we have reached, and we look at the shortest distance from any one of those nodes to a new node

Prims's algorithm

Use the DFS algorithm given earlier, but consider s to be a set rather than a stack.

At each iteration, choose the edge (v,w) from s that has minimum weight.

Hence, maintain s as a min-heap!

That's it!

Proof that this actually constructs a minimal spanning tree is not given here.

This is a **greedy algorithm**: At each step, it chooses the best.

Kruskal's algorithm

// Find a minimum-weight spanning tree for graph (V, E).
Sort the edges by weight;
V_T= V; E_T= empty // the vertices and edges of tree
for each edge (v,w) (in order of increasing weight):
  if (adding (v,w) does not create a cycle) {
    Add (v,w) to ET;
  }

Difficulty is testing whether a cycle is created.

Greedy algorithm: at each stage it picks the next best edge to add.

Editorial notes

- Dijkstra’s algorithm and Prim’s algorithm are examples of greedy algorithms:
  - making optimal choice at each step of the algorithm gives globally optimal solution
- In most problems, greedy algorithms do not yield globally optimal solutions
  - (e.g.) Traveling Salesman Problem