Minimal Spanning Trees

**Spanning Tree**

- Assume you have an undirected graph \( G = (V,E) \)
- Spanning tree of graph \( G \) is tree \( T = (V,E_T \subseteq E, R) \)
  - Tree has same set of nodes
  - All tree edges are graph edges
  - Root of tree is \( R \)

**Spanning trees**

- Breadth-first Spanning Tree
- Depth-first spanning tree

**Property 1 of spanning trees**

- Graph: \( G = (V,E) \)
- Spanning tree: \( T = (V,E_T,R) \)
- 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: if \( u \) is ancestor of \( v \), result is easy. Otherwise, let \( l \) be the first node in common to paths from \( u \) to root of tree, and from \( v \) to root of tree. The paths \( u \rightarrow v, v \rightarrow l, l \rightarrow u \) can be concatenated to form the desired cycle.

**Example:**

- Edge (I,H):
  - \( l \) is node G
  - simple cycle is (I,H,G,I)
- Edge (H,C):
  - \( l \) is node A
  - simple cycle is (H,C,B,A,G,H)
Property 2 of spanning trees

- Graph: $G = (V,E)$
- Spanning tree: $T = (V,E_T,R)$
- 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 $(s \rightarrow t)$ in $Y$ gives another spanning tree $T'$.

Proof of Property 2

- In $T'$, every node is reachable from every other node.
  - Otherwise, assume node $a$ is not reachable from node $b$ in $T'$. In $T$, there must be a path from $b$ to $a$ that contains edge $(s \rightarrow t)$. In this path, replace edge $(s \rightarrow t)$ by the path in $T'$ obtained by deleting $(s \rightarrow t)$ from cycle $Y$, which gives a path from $b$ to $a$.

Proof of Property 2 (contd.)

- In $T'$, there is a unique simple path between any two nodes.
  - Otherwise, suppose $p_1 = (x \rightarrow y)$ and $p_2 = (x \rightarrow y)$ are two distinct, edge-disjoint simple paths in $T'$.
  - The edge $(u \rightarrow v)$ occurs on at least one of these paths. Concatenating $p_1$ and $p_2$, and deleting edge $(u \rightarrow v)$ gives a path $p_1$ in $T'$ from $u$ to $v$. This path does not contain $(s \rightarrow t)$, so it must be present in $T$ as well. But $T$ also contains another simple path $p_2$ from $u$ to $v$ obtained by taking cycle $Y$ and deleting $(u \rightarrow v)$ from it; this is distinct from $p_1$ because it does contain $(s \rightarrow t)$.
  - This is a contradiction.
  - Therefore, $T'$ is a tree.

Building BFS/DFS spanning trees

- Use sequence structure as before, but put/get edges, not nodes
  - Get edge $(s,d)$ from structure
  - If $d$ is not in done set, add $d$ to done set
  - $(s,d)$ is in spanning tree
  - add out-edges $(d,t)$ to seq structure if $t$ is not in done set

- Example: BFS Queue
  - $[(\text{dummy},A)]$
  - $[(B),(A,G),(A,F)]$
  - $[(A,G),(A,F),(B,G),(B,C)]$.....
Weighted Spanning Trees

- Assume you have 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.
  - In general, a graph has several MST’s.
  - Applications: circuit-board routing etc.

Caution: in general, SSSP tree is not MST

- 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.

Property 3 of spanning trees

- Graph: $G = (V, E)$
- Spanning tree: $T = (V, E_T, R)$
- Edge: $c = (u, v)$ in $G$ but not in $T$
- Cycle edges: $(G \rightarrow H)$, $(I \rightarrow E)$, $(E \rightarrow D), (H \rightarrow D)$ all have weights less than $(G \rightarrow H)$.
- Proof: Otherwise, modifying $T$ by adding $(u \rightarrow v)$ and dropping heavier edge on cycle gives spanning tree of less weight.
Building Minimal Spanning Trees

- Prim’s algorithm: simple variation of Dijkstra’s SSSP algorithm
- Change Dijkstra’s algorithm so the priority of bridge \((f \rightarrow n)\) is \(\text{length}(f,n)\) rather than \(\text{minDistance}(f) + \text{length}(f,n)\)
- Algorithm produces minimal spanning tree!

Prim’s MST algorithm

Tree MST = empty tree;
Heap \(h = \text{new Heap();}\
//any node can be the root of the MST
\(h.put((\text{dummyRoot} \rightarrow \text{startNode}), 0);\)
while (\(h\) is not empty) {
  get minimum priority bridge \((t \rightarrow f)\);
  if \((f)\) is not lifted) {
    add \((t \rightarrow f)\) to MST; //grow MST
    make \(f\) a lifted node;
    for each edge \((f \rightarrow n)\)
      if \((n)\) is not lifted
        \(h.put((f \rightarrow n), \text{length}(f,n));\)
  }
}

Steps of Prim’s algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>([(\text{dummy} \rightarrow A), 0])</td>
</tr>
<tr>
<td></td>
<td>add (dummy \rightarrow A) to MST</td>
</tr>
<tr>
<td></td>
<td>([(A \rightarrow B), 2], [(A \rightarrow G), 5], [(A \rightarrow F), 9])</td>
</tr>
<tr>
<td>2</td>
<td>([(A \rightarrow G), 5], [(A \rightarrow F), 9]) add (A \rightarrow B) to MST</td>
</tr>
<tr>
<td></td>
<td>([(A \rightarrow G), 5], [(A \rightarrow F), 9], [(B \rightarrow G), 6], [(B \rightarrow C), 4])</td>
</tr>
<tr>
<td>3</td>
<td>([(A \rightarrow G), 5], [(A \rightarrow F), 9], [(B \rightarrow G), 6]) add (B \rightarrow C) to MST</td>
</tr>
<tr>
<td></td>
<td>([(A \rightarrow G), 5], [(A \rightarrow F), 9], [(B \rightarrow G), 6], [(C, H), 5], [(C, D), 2])</td>
</tr>
<tr>
<td></td>
<td>............</td>
</tr>
</tbody>
</table>

Property of Prim’s algorithm

- At each step of the algorithm, we have a spanning tree for “lifted” nodes.
- This spanning tree grows by one new node and edge at each iteration.
Proof of correctness

- Suppose the algorithm does not produce MST.
- Each iteration adds one new node and edge to tree.
- First iteration adds the root to tree, and at least that step is "correct".
  - "Correct" means partial spanning tree built so far can be extended to an MST.
- Suppose first k steps were correct, and then algorithm made the wrong choice.
  - Partial spanning tree P built by first k steps can be extended to an MST M
  - Step (k+1) adds edge (u \rightarrow v) to P, but resulting tree cannot be extended to an MST.

Proof (contd.)

- Consider simple cycle formed by adding (u \rightarrow v) to M. Let p be the lowest ancestor of v in M that is also in P, and let q be p’s child in M that is also an ancestor of v. So (p \rightarrow q) is a bridge edge at step (k+1) as is (u \rightarrow v). Since our algorithm chose (u \rightarrow v) at step (k+1), weight(u \rightarrow v) is less than or equal to weight(p \rightarrow q).
- From Property (3), weight of (u \rightarrow v) must be greater than or equal to weight(p \rightarrow q).

Therefore, weight(p \rightarrow q) = weight(u \rightarrow v).

This means that the tree obtained by taking M, deleting edge (p \rightarrow q) and adding edge (u \rightarrow v) is a minimal spanning tree as well, contradicting the assumption that there was no MST that contained the partial spanning tree obtained after step (k+1).

Therefore, our algorithm is correct.

Complexity of algorithm

- Every edge is examines once and inserted into PQ when one of its two end points is first lifted.
- Every edge is examined again when its other end point is lifted.
- Number of insertions and deletions into PQ is |E| + 1
- Complexity = O(|E|log(|E|))
• 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
  – (eg) TSP
  – (eg) greedy algorithm for puzzle graph search: at each step, choose move that minimizes the number of tiles that are out of position
• Problem: we can get stuck in “local” minima and never find the global solution