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 \( u \rightarrow v \rightarrow l \rightarrow u \) can be concatenated to form the desired cycle.

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)
Useful lemma

- In any tree \( T = (V,E) \), \( |E| = |V| -1 \)
  - Proof: (by induction on \( |V| \))
    - If \( |V| = 1 \), we have the trivial tree containing a single node, and the result is obviously true.
    - Assume result is true for all trees for which \( 1 \leq |V| < n \) and consider a tree \( S=(E_S,V_S) \) with \( |V_S| = n \). Such a tree must have at least one leaf node; removing the leaf node and edge incident on that node gives a smaller tree \( T \) with less than \( n \) nodes. By inductive assumption, \( |E_S| = |V_S|+1 \). Since \( |E_S| = |E_T|+1 \) and \( |V_S| = |V_T|+1 \), the required result follow.

- An undirected graph \( G = (V,E) \) which is (1) a connected component, and (2) \( |E| = |V|-1 \) is a tree.

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. Inserting edge \( c \) into \( T \) and deleting any edge in \( Y \) gives another spanning tree \( T' \).

![Graph with edge (H,C) and cycle (H,C,B,A,G,H)]

Proof of Property 2

- \( T' \) is a connected component.
  - 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 \( a \).

- In \( T' \), number of edges = number of nodes – 1
  - Proof: by construction of \( T' \) and fact that \( T \) is a tree

  Therefore, from lemma, \( T' \) 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
  
  \[
  [(\text{dummy,A})]
  [(A,B),(A,G),(A,F)]
  [(A,G),(A,F),(B,G),(B,C)]
  \]
**Weighted Spanning Trees**

- Assume you have an undirected graph $G = (V,E)$ with weights on each edge.
- Spanning tree of graph $G$ is a 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.

**Example**

The graph and SSSP tree are shown with weights on the edges. The minimal spanning tree is also demonstrated.

**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 $T$
  - There is a simple cycle $Y$ containing only edge $c$ and edges in spanning tree. Moreover, weight(u→v) must be greater than or equal to weight of any edge in this cycle.
  - Proof: Otherwise, modifying $T$ by adding $(u\rightarrow v)$ and dropping heavier edge on cycle gives spanning tree (Property 2) 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 $\text{MST} = \text{empty tree}$;
Heap $h = \text{new Heap}();$
//any node can be the root of the MST
$h.\text{put}((\text{dummyRoot} \rightarrow \text{startNode}), 0);$
while ($h$ is not empty) {
    get minimum priority bridge ($t \rightarrow f$);
    if ($f$ is not lifted) {
        add ($t \rightarrow f$) to $\text{MST}$; //grow $\text{MST}$
        make $f$ a lifted node;
        for each edge ($f \rightarrow n$)
            if ($n$ is not lifted)
                $h.\text{put}((f \rightarrow n), \text{length}(f, n));$
    }
}

Steps of Prim’s algorithm

<table>
<thead>
<tr>
<th>$((\text{dummy} \rightarrow A), 0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[] add ($\text{dummy} \rightarrow A$) to $\text{MST}$</td>
</tr>
<tr>
<td>$((A \rightarrow B), 2),((A \rightarrow G), 5),((A \rightarrow F), 9)$</td>
</tr>
<tr>
<td>$((A \rightarrow G), 5),((A \rightarrow F), 9),((B \rightarrow G), 6),((B \rightarrow C), 4)$</td>
</tr>
<tr>
<td>$((A \rightarrow G), 5),((A \rightarrow F), 9),((B \rightarrow G), 6),((C \rightarrow H), 5),((C \rightarrow D), 2))$</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→v) to P, but resulting tree cannot be extended to an MST.

**Proof (contd.)**

- Consider simple cycle formed by adding (u→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→q) is a bridge edge at step (k+1) as is (u→v). Since our algorithm chose (u→v) at step (k+1), weight(u→v) is less than or equal to weight(p→q).
- From Property (3), weight of (u→v) must be greater than or equal to weight(p→q).

- Therefore, weight(p→q) = weight(u→v).
- This means that the tree obtained by taking M, deleting edge (p→q) and adding edge (u→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 examined once and inserted into PQ when one of its two end points is first lifted.
- Every edge is examined again when it is removed from the PQ.
- Number of insertions and deletions into PQ is |E| + 1
- Complexity = O(|E|log(|E|))
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
  - (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