Minimum Spanning Tree

Greedy Algorithms for Minimum Spanning Tree
- Prim’s Algorithm: Extend a tree by including the cheapest out going edge
- Kruskal’s Algorithm: Add the cheapest edge that joins disjoint components
Why do the greedy algorithms work?

- For simplicity, assume all edge costs are distinct

Edge inclusion lemma

- Let \( S \) be a subset of \( V \), and suppose \( e = (u, v) \) is the minimum cost edge of \( E \), with \( u \) in \( S \) and \( v \) in \( V - S \)
- \( e \) is in every minimum spanning tree of \( G \)
  - Or equivalently, if \( e \) is not in \( T \), then \( T \) is not a minimum spanning tree

Proof

- Suppose \( T \) is a spanning tree that does not contain \( e \)
- Add \( e \) to \( T \), this creates a cycle
- The cycle must have some edge \( e_1 = (u_1, v_1) \) with \( u_1 \) in \( S \) and \( v_1 \) in \( V - S \)

\[ T_1 = T - \{e_1\} + \{e\} \text{ is a spanning tree with lower cost} \]
- Hence, \( T \) is not a minimum spanning tree

Optimality Proofs

- Prim’s Algorithm computes a MST
- Kruskal’s Algorithm computes a MST

- Show that when an edge is added to the MST by Prim or Kruskal, the edge is the minimum cost edge between \( S \) and \( V - S \) for some set \( S \).

Prim’s Algorithm

\[
S = \{ \text{a} \}; \quad T = \{ \}; \\
\text{while } S \neq V \\
\quad \text{choose the minimum cost edge } \\
\quad e = (u, v), \text{ with } u \text{ in } S, \text{ and } v \text{ in } V - S \\
\quad \text{add } e \text{ to } T \\
\quad \text{add } v \text{ to } S 
\]

Prove Prim’s algorithm computes an MST

- Show an edge \( e \) is in the MST when it is added to \( T \)
Kruskal’s Algorithm

Let \( C = \{\{v_1\}, \{v_2\}, \ldots, \{v_n\}\}; \ T = \{\} \)

while \(|C| > 1\)

Let \( e = (u, v) \) with \( u \) in \( C_i \) and \( v \) in \( C_j \) be the minimum cost edge joining distinct sets in \( C \)

Replace \( C_i \) and \( C_j \) by \( C_i \cup C_j \)

Add \( e \) to \( T \)

Prove Kruskal’s algorithm computes an MST

• Show an edge \( e \) is in the MST when it is added to \( T \)

MST Implementation and runtime

• Prim’s Algorithm
  – Implementation, runtime: just like Dijkstra’s algorithm
  – Use a heap, runtime \( O(m \log n) \)

• Kruskal’s Algorithm
  – Sorting edges by cost: \( O(m \log n) \)
  – Managing connected components uses the Union-Find data structure
    • Amazing, pointer based data structure
    • Very interesting mathematical result

Disjoint Set ADT

• Data: set of pairwise disjoint sets.

• Required operations
  – Union – merge two sets to create their union
  – Find – determine which set an item appears in

• Check \( \text{Find}(v) \neq \text{Find}(w) \) to determine if \((v,w)\) joins separate components

• Do \( \text{Union}(v,w) \) to merge sets

Up-Tree for DS Union/Find

Observation: we will only traverse these trees upward from any given node to find the root.

Idea: reverse the pointers (make them point up from child to parent). The result is an up-tree.

Initial state

```
1  2  3  4  5  6  7
```

Intermediate state

```
1  3  7
2  5  4
```

Roots are the names of each set.