Kruskal's Algorithm

Minimum Spanning Trees

Spanning Trees

- a (free) tree is an acyclic undirected connected graph
  - free because it is not implicitly directed, the way a rooted tree is
- simple facts about trees:
  - a connected graph with n vertices and $n-1$ edges is a tree
  - adding one edge to a tree creates a unique cycle, deleting any edge from the cycle creates a tree
- a spanning tree T of G is a tree which contains every vertex in V and has a subset of G's edges so that it is a tree - $T=(V,E')$, $E' \subseteq E$
- the weight of a graph $G=(V,E)$ is $w(G)=\sum_{e\in E} w(e)$

Minimum Spanning Tree problem

Given an undirected connected graph $G=(V,E)$ and a weight function $w: E \to \mathbb{R}$ (do not require weights to be non-negative), find a minimum weight spanning tree of G.

- a complete graph with n vertices has n^{n-2} spanning trees, so calculating the weight of each spanning tree and choosing the minimum weight would be ridiculously slow
- problem was originally solved by engineers deciding how to connect a power grid

Kruskal's Algorithm

- Kruskal worked at Bell Labs, published this algorithm in 1956
- is a greedy algorithm

```
def Kruskal(G):
    H = heap containing (e, w(e)) for each e in E
    S = disjoint set forest containing each v in V
    F = empty set  # trivial partial solution
    while |F| < n - 1:
        # greedily extend partial solution
        (u, v) = ExtractMin(H)  # find least weight edge
        # insert if u and v are not already connected
        if Find(u) != Find(v):
            insert(F, (u, v))
            Union(u, v)  # mark u, v connected
    return F
```

Runtime

- creating heap takes $\mathcal{O}(m)$ time
- creating initial disjoint set forest takes $\mathcal{O}(n)$ time
- `ExtractMin` is performed at most m times as it starts with m items, and each one takes at most $\log(m)$ time, however $m \leq n^2$ and $\log(n^2) = 2\log(n)$, so all `ExtractMin` operations take $\mathcal{O}(m\log(n))$
- unions and finds together take $\mathcal{O}(m\log^*(n))$ time in total

So in total, runtime complexity is $\mathcal{O}(m\log(n))$

Correctness

Cut of a graph

a cut of $G = (V, E)$ is a partition of V into $S \subseteq V$ and $\overline{S} = V \setminus S$

- (u, v) *crosses* the cut if $u \in S$ and $v \in \overline{S}$

Cut property

Suppose $F \subseteq E$ is contained in some MST of G (F is a partial solution for the MST), $(S, \overline{S})$ is a cut of G so that no edge in F crosses the cut, and e is a minimum weight edge which crosses $(S, \overline{S})$. Then $F \cup \{e\}$ is contained in some MST of G.

Proof.

Let T be an MST of G.

If $e \in T$, then we are done.

If $e \notin T$, then adding e to T creates a unique cycle in T. This cycle contains $e' \neq e$ that crosses the cut. This is because if $e = (u, v)$ and e is part of a cycle, then there must be another path from u to v, which must cross the cut some other way.

Let $T' = (T \cup \{e\}) \setminus \{e'\}$, then T must still be a spanning tree.

$w(T') = w(T) + w(e) - w(e')$

e was a minimum weight edge crossing the cut and e' was another edge crossing the cut, so $w(e) \leq w(e')$, so $w(T') \leq w(T)$

T is an MST of G and $w(T') \leq w(T)$, so T' must also be an MST of G

Loop Invariant

For the i^{th} iteration,

1. F_i has no cycles
2. F_i is contained in some MST

Base Case

F_0 is empty, so it has no cycles and is contained in every MST.

Inductive Step

Suppose F_k has no cycles and is contained in some MST.

$F_k = F_{i+1} \cup \{e\}$ and e was chosen so that F_{k+1} does not have any cycles, so F_{k+1} has no cycles.

Let S be the set of nodes that are connected by F_k and $\overline{S} = V \setminus S$, then $e \in \overline{S}$. e crosses this cut, and it was chosen to be a minimum-weight edge[citation needed], so F_{k+1} is also contained in an MST.

Conclusion

Since the loop invariant is true for every iteration, it is true when $|F| = n - 1$ aka when the loop terminates. Since $|F| = n - 1$, the resulting graph must be a spanning tree, so F details a minimum spanning tree of G.