17—Minimum Spanning Trees and Kruskal’s Algorithm

May 19, 2002

Subgraphs

- Cities and edges form a graph
- Cities and red edges form a subgraph
- Weighted graphs have a value: $\text{value}(G) = \sum_{e \in E} \text{weight}(e)$

A Graph Problem

$G$: a weighted graph

What is the cheapest, connected subgraph of $G$?
How to Find Spanning Trees

Two Greedy Algorithms

Prim's Algorithm

Almost the same as Dijkstra's

Kruskal's Algorithm

Totally different!

Kruskal's Algorithm (Almost)

Key idea: We like short edges

Kruskal's Algorithm (Almost)

Kruskal(Graph G)
{
    Graph MST;
    PQ pq.Build(G.edges());
    while (num components > 1) {
        Edge e = pq.DeleteMin();
        if (e does not make a cycle)
            MST.Add(e);
    }
    return MST;
}
Can we always extend our current tree to some MST after choosing any valid minimum-cost edge?

- $G$ is our current forest
- $e$ is the valid minimum-cost edge we're going to add
- Assuming there is an extension from $G$ to a MST, how do we know $e$ is in that extension?

- $G$ is our current forest
- $e = (u, v)$ is the valid minimum-cost edge we're going to add
- $F$ is a minimum-cost extension of $G$
More Slides = Better

- \( \exists u \rightarrow v \) path \( p \) in \( F \)
  - Why?
- \( p \) has an edge \( e' \) between components of \( G \)
  - Why?
- \( \text{weight}(e) \leq \text{weight}(e') \)
  - Why?

Yes, More Slides = Better

- \( \text{value}(F - e' + e) \leq \text{value}(F) \)
  - Why?
- \( F - e' + e \) is a tree
  - Why?
- Hence \( G + e \) can be extended to an MST
  - Yay!

Implementing Kruskal's

```java
Kruskal(Graph G) {
    Graph MST;
    PQ pq.Build(G.edges());
    while (num components > 1) {
        Edge e = pq.DeleteMin();
        if (e does not make a cycle) MST.Add(e);
    }
    return MST;
}
```

Which part is hard?
Disjoint Sets

- Components are sets
- \( e = (u, v) \) won't cause cycle if sets containing \( u \) and \( v \) are disjoint
- If we add \( e \), union the sets containing \( u \) and \( v \)

Just like mazes

Kruskal Implementation

```cpp
Kruskal(Graph G) {
    Graph MST;
    PQ pq.Build(G.edges());
    DS ds.MakeSets(G.num_vertices());
    int components = G.num_vertices();
    while ( ) {
        Edge e = pq.DeleteMin();
        DS::Set *u = ds.Find(e->u->Number());
        DS::Set *v = ds.Find(e->v->Number());
        if ( ) {
            MST.Add(e);
        }
    }
    return MST;
}
```

Disjoint Set Implementation

```cpp
class DS {
    class Set {
        ...
    };

    public:
    class Set;
    void Union(Set *, Set *);
    Set *Find(Set *);
    void MakeSets(int n) {
    }
    Set *Find(int) {
    }
};
```
Running Time?

Kruskal(Graph G)
{
    Graph MST;
PQ pq.Build(G.edges());
    DS ds.MakeSets(G.num_vertices());
    int components = G.num_vertices();
    while (components > 1) {
        Edge e = pq.DeleteMin();
        DS::Set *u = ds.Find(e->u->Number());
        DS::Set *v = ds.Find(e->v->Number());
        if (u != v) {
            MST.Add(e);
            ds.Union(u,v);
            components--;
        }
    }
    return MST;
}