IE170: Algorithms in Systems Engineering: Lecture 18

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

March 12, 2007

Taking Stock

Last Time

- Topological Sort: Making the Perfect Martini
- Strongly Connected Components

This Time: Uses of DFS

- Turn in Homework Now, please!
- Minimum Spanning Trees

A Canonical Problem

- A town as a set of houses and a set of potential roads
- Each each connects two and only two houses
- Constructing road from house u to house w costs w_{uv}

The Objective: Construct roads such that

1. Everyone is Connected
2. The total repair cost is minimum

Spanning Tree

- We model the problem as a graph problem.
- $G = (V, E)$ is an undirected graph
- Weights $w : E \to \mathbb{R}^{|E|}$
  - $w_{uv} \; \forall (u, v) \in E$
- Find $T \subset E$ such that
  1. T connects all vertices
  2. The weight

$$w(T) \stackrel{\text{def}}{=} \sum_{(u,v) \in T} w_{uv}$$

  is minimized

Spanning TREE

- The notation T is not a coincidence.
- The set of edges T will form a tree. (Why?)
- This subset is known as a minimum spanning tree (MST) of G

How to Build It!?

- Let A be a set of edges (initially empty)
- Add to A keeping the following loop invariant:
- A is always a subset of some MST
- Call edge $(u, v) \in E$ safe for A if $A \cup \{(u, v)\}$ is also a subset of a MST.
- The goal for the algorithms is to quickly detect and add safe edges.

Generic-MST(V, E, w)
$A \leftarrow \emptyset$
while A is not a spanning tree
do find $(u, v) \in E$ that is safe for A
$\quad A \leftarrow A \cup \{(u, v)\}$
return A

Finding Safe Edges

How do I know if (u, v) is safe? (Intuition)

- Let $S \subset V$ be any set of vertices that includes u but not v
- In any MST there must be at least one edge that connects S to $V \setminus S$, so let's make the greedy choice of choosing the one with the minimum weight

Some Definitions for graph $G = (V, E)$

- A cut $(S, V \setminus S)$ is a partition of the vertices into disjoint sets S and $V \setminus S$
- An edge $(u, v) \in E$ crosses cut $(S, V \setminus S)$ is one endpoint is in S and the other is in $V \setminus S$
- A cut respects a set of edges $A \subseteq E$ if and only if no edge in A crosses the cut

---

MST Theorem

Let A be a subset of some MST, let $(S, V \setminus S)$ be a cut that respects A, and let (u, v) be the minimum weight edge crossing $(S, V \setminus S)$. Then (u, v) is safe for A

Proof?

Kruskal's Algorithm

1. Start with each vertex being its own component
2. Merge two components into one by choosing the light edge that connects them
3. Scans the set of edges in increasing order of weight
4. It uses an abstract "disjoint sets" data structure to determine if an edge connects different vertices in different sets.
5. We will use Java Collections Classes
   - Less efficient
   - Easier to Code!

Kruskal's Algorithm

KRUSKAL(V, E, w)

```
A ← ∅
for each v in V
do MAKE-SET(v)
SORT(E, w)
for each (u, v) in (sorted) E
do if FIND-SET(u) ≠ FIND-SET(v)
      then A ← A ∪ {(u, v)}
            UNION(u, v)return A
```


Analysis

- Let $\mathcal{T}(\mathcal{X})$ be the running time of the method $\mathcal{X}$

Task	**Running Time**
Initialize A	$O(1)$
First for loop	$\|V\|\mathcal{T}(\text{MAKE-SET})$
Sort E	$O(E \lg E)$
Second for loop	$O(E)(\mathcal{T}(\text{FIND-SET} + \text{UNION}))$

We Skipped That Chapter!

- If we use a clever data structure for FIND-SET and UNION, the running time can go to $\alpha(m, n)$, where m is the total number of operations, and n is the number of unions.
- $\alpha(m, n)$ is the inverse of the Ackerman function, which is a slowly growing function.
- $\alpha(m, n) \leq 4$ for all practical purposes
- In this case, we have that the operations take $\alpha(|E|, |V|)$

Kruskal Analysis

- Also, you should know that $\alpha(|E|, |V|) = O(\lg |V|)$
- Finally, not that $|E| \leq |V|^2 \Rightarrow \lg |E| = O(2 \lg |V|) = O(\lg |V|)$
- Therefore the running time for Kruskal's Algorithm is $O(|E| \lg |V|)$.
- If the edges are already sorted, it runs in $O(|E|\alpha(|E|, |V|))$, which is essentially linear

Prim's Algorithm

- Builds one tree, so A is always a tree
- Let V_A be the set of vertices on which A is incident
- Start from an arbitrary root r
- At each step find a light edge crossing the cut $(V_A, V \setminus V_A)$

Main Question for Prim..

How do we find a light edge crossing the cut quickly?

Prim's Algorithm

The Answer!

- Use a priority queue!
- We built a priority queue in Lab 4. heaps are priority queues

- Each object in the queue is a vertex in $V \setminus V_A$ (A vertex that might be linked to our MST)
- The key of v is the minimum weight of any edge (u, v) such that $u \in V_A$.
- The key of v is ∞ if v is not adjacent to any vertices in V_A

Prim's Algorithm

- Prim's Algorithm starts from an (arbitrary) vertex (the root r)
- It keeps track of the parent $\pi[v]$ of every vertex v. ($\pi[r] =$ NIL).
- As the algorithm processes $A = \{(v, \pi[v]) \mid v \in V \setminus \{r\} \setminus Q\}$
- At termination, $V_A = V \Rightarrow Q = \emptyset$, so MST is

$$A = \{(v, \pi[v]) \mid v \in V \setminus \{r\}\}.$$

Pseudocode for Prim

$\text{PRIM}(V, E, w, r)$
$Q \leftarrow \emptyset$
for each $u \in V$
do $key[u] \leftarrow \infty$
    $\pi[u] \leftarrow \text{NILINSERT}(Q, u)$
    $key[r] = 0$
while $Q \neq \emptyset$
do $u \leftarrow \text{EXTRACT-MIN}(Q)$
    for each $v \in Adj[u]$
    do if $v \in Q$ **and** $w_{uv} < key[v]$
        then $\pi[v] \leftarrow u$
            $key[v] = w_{uv}$

DFS Review
Topological Sort
Strongly Connected Components

Next Time: Happy Dats

- I have to go to Pittsburgh
- Substitute Lecturer on Wednesday
- Also no Office Hours on Wednesday

Jeff Linderoth IE170:Lecture 18