CS660: Algorithms - Lecture 8

Instructor: Hoa Vu

San Diego State University

Storing files on tape generalized

- $E(cost(\pi)) = \sum_{i=1}^{n} \sum_{k=1}^{i} Pr[\text{access file } i] L(\pi(k)) = \sum_{i=1}^{n} \sum_{k=1}^{i} F(i) L(\pi(k))$.
- Idea: order the files by increasing $L(i)/F(i)$.
- Proof of correctness: why does this algorithm give you the optimal cost?

Scheduling classes

- Each class i has a start time $S[i]$ and finish time $F[i]$ (where $0 \leq S[i] < F[i] \leq M$).
- Problem: schedule the most number of classes.
- Recursion solution?

Scheduling classes

- Idea: First class chosen to finish as soon as possible. Continue to pick the next class in the same manner.
- Let f be the class that finishes first, i.e., $F[i]$ is smallest. Add f to the schedule. Let A' be the class that starts after f finishes (no conflict). Continue picking class from A' in the same manner.
- Exercise: implement this in $O(n \log n)$ time.

Scheduling classes

- Why is this optimal?
- Theorem: the greedy algorithm is optimal.
- Proof: Suppose that the greedy algorithm is not optimal. Let the greedy schedule be $G = (g_1, g_2, \ldots, g_l)$. Let g_i be the first choice of the greedy algorithm that is different from *all* optimal schedule. That is $g_1, \ldots, g_{i-1}$ is the prefix of some optimal schedule. But no optimal schedule contains

$$g_1, \ldots, g_{i-1}, g_i$$

as a prefix.

Scheduling classes

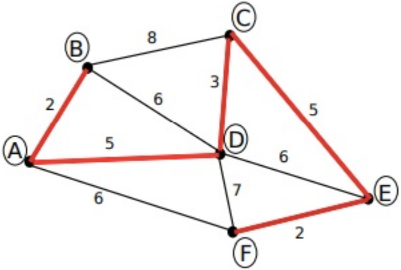
- Proof: Suppose that the greedy algorithm is not optimal. Let the greedy schedule be $G = (g_1, g_2, \ldots, g_l)$. Let g_i be the first choice of the greedy algorithm that is different from *all* optimal schedule. That is $g_1, \ldots, g_{i-1}$ is the prefix of some optimal schedule. But no optimal schedule contains

  $$g_1, \ldots, g_{i-1}, g_i$$

  as a prefix.
- Now, consider the optimal schedule $O = (o_1, o_2, \ldots, ..., o_k)$ where $o_i \neq g_i$ but $o_1 = g_1, o_2 = g_2, \ldots, o_{i-1} = g_{i-1}$. So we can replace o_i with g_i and get another optimal schedule since g_i finishes no later than o_i. But now since $(o_1, o_2, \ldots, o_{i-1}, g_i, o_{i+1}, \ldots, o_k)$ is also an optimal schedule. We assume that $(o_1, o_2, \ldots, o_{i-1}, g_i)$ is not the prefix of *all* optimal schedules. Hence, we have a contradiction.

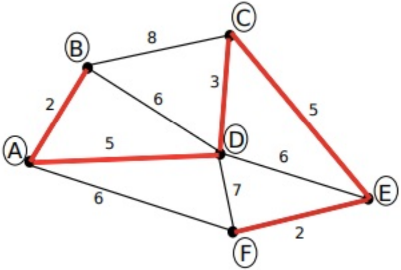
Minimum Spanning Tree

- Given a weighted connected graph. Find a tree that covers every vertex such that the total edge weights is minimized.



Minimum Spanning Tree

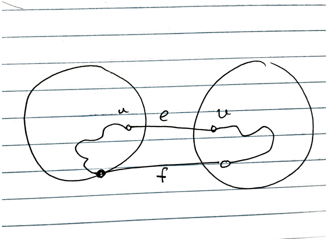
- Greedy algorithm: pick an arbitrary node as the starting tree. Add the minimum weight edge with one end point to the tree until the tree



covers all vertices.

Minimum Spanning Tree

- For simplicity, assume that the weights are distinct (note that the proof still goes through if they are not just a bit more wordy).
- Let $e = uv$ be the first edge added by the algorithm that is not in the minimum spanning tree (let the tree at that point be T and the remaining graph be $G \setminus T$).
- Since u and v must be connected in the MST, there must be a path from u to v. Hence, there must be an edge f between a vertex in T and a vertex in $G \setminus T$. If we remove f from the MST, and add e to the MST, we get a spanning tree with a smaller weight (since the algorithm picked e over f because $w(e) < w(f)$) which is a contradiction.



Minimum Spanning Tree

- Prim's algorithm can be implemented to run in $O(|E| + |V| \log |V|)$ time using Fibonacci heap (read 7.4).