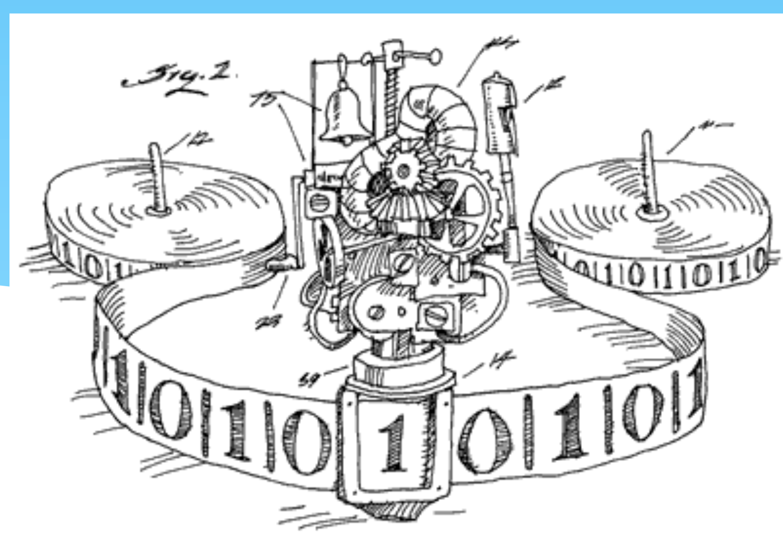
EECS 376: Foundations of Computer Science

Seth Pettie

Lecture 6



Template

* Solve the problem in a "greedy", "myopic" way
  * Rarely gives you an ***exactly*** optimum solution, but makes for some very elegant algorithms when it does.
  * (Often works well for ***approximation algorithms*** — more on this in November.)
* **Main difficulty:** arguing correctness
  * Exchange arguments

Activity scheduling

- An activity i has start time s_i and end time f_i
- **Goal:** Given a set A of n activities (classes), select a subset $S \subseteq A$ that are ***mutually disjoint*** *that maximizes* $|S|$, i.e. a maximum schedule.
- Activities i and j are disjoint if their intervals $[s_i, f_i)$ and $[s_j, f_j)$ don't overlap
  - $s_i \geq f_j$ or $s_j \geq f_i$

No preeminent "Greedy" algorithm

* Possible greedy heuristics:
  * Pick a set of activities one at a time, ***shortest activities first*** (minimizing $|f_i - s_i|$).
  * Pick a set of activities one at a time, ***earliest starting time*** first.
  * Pick a set of activities one at a time, ***earliest finishing time*** first.

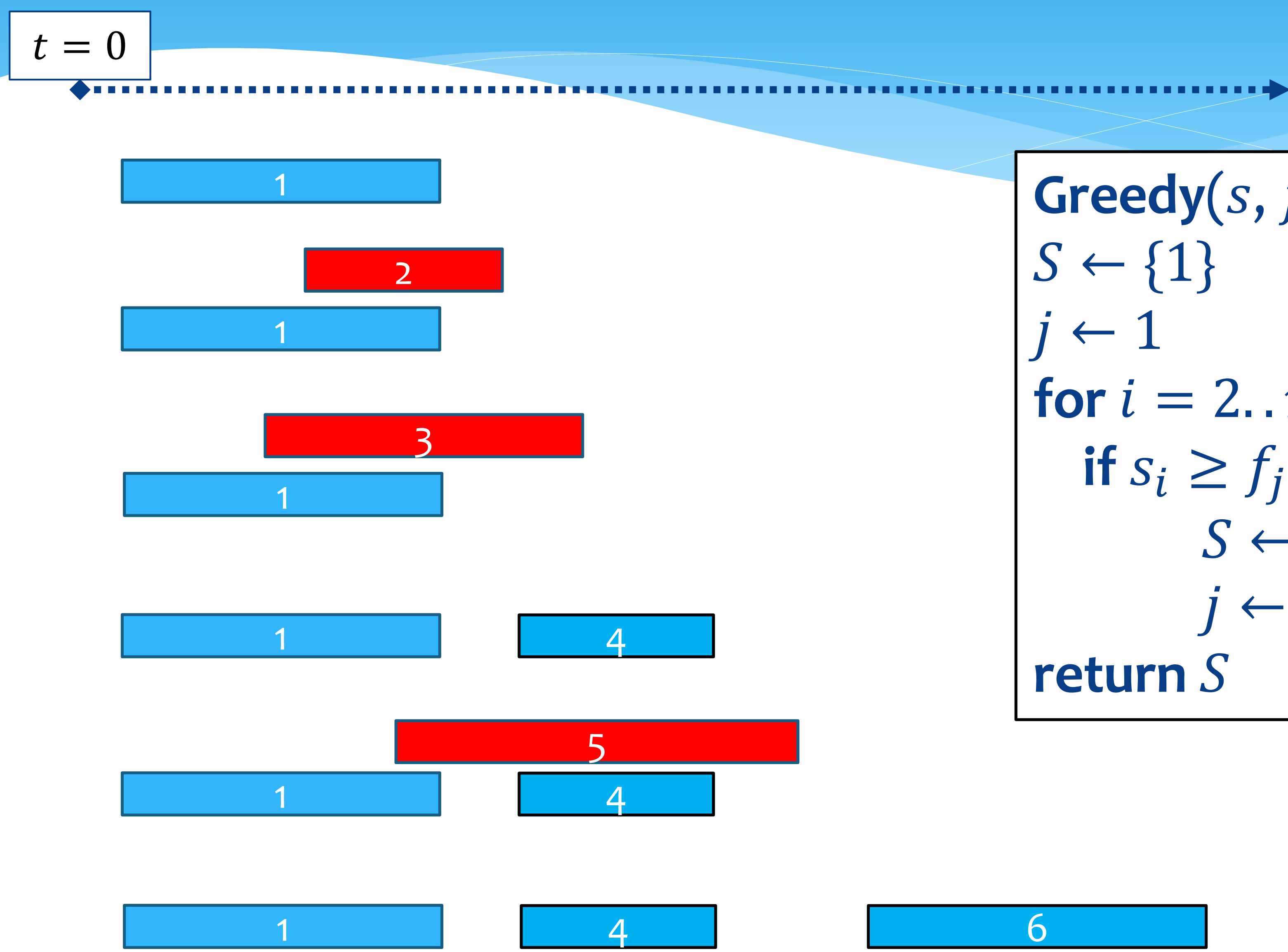
A greedy algorithm

Assume they're sorted in increasing order by finishing time: $f_1 \leq f_2 \leq \cdots \leq f_n$

Greedy(s, f)**:**
$S \leftarrow \{1\}$ \\ chosen activities
$j \leftarrow 1$ \\ activity chosen with the largest f_j
for $i = 2..n$**:**
  if $s_i \geq f_j$ **:**
    $S \leftarrow S \cup \{i\}$
    $j \leftarrow i$
return S

Runtime: $O(n)$

A greedy algorithm



Greedy(s, f):
$S \leftarrow \{1\}$
$j \leftarrow 1$
for $i = 2..n$:
  if $s_i \geq f_j$:
    $S \leftarrow S \cup \{i\}$
    $j \leftarrow i$
return S

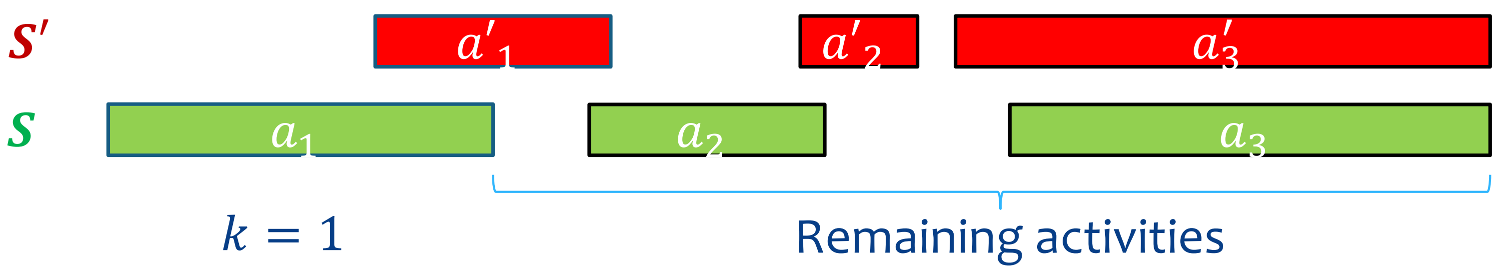
Greed is good

- **Theorem: Greedy**(s, f) returns a *maximum size* schedule $\boldsymbol{S}$
- Suppose $\boldsymbol{S'}$ is an *arbitrary* maximum size schedule.
- **Goal:** show size $\boldsymbol{S}$ = size of $\boldsymbol{S'}$
- **Idea:** show that we can *transform* $\boldsymbol{S'}$ to $\boldsymbol{S}$ *while maintaining the size* (by induction, "swapping in" an activity of $\boldsymbol{S}$ for an activity of $\boldsymbol{S'}$, one at a time)

commonly employed strategy to show that a greedy algorithm is optimal

Greed is good

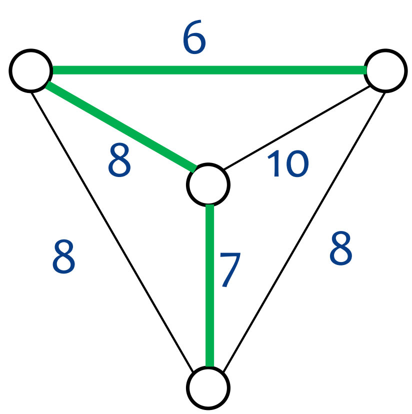
* Order the activities $a_1, a_2, \dots$ in $\boldsymbol{S}$ by their finish time
* Order the activities $a_1', a'_2, \dots$ in $\boldsymbol{S'}$ by their finish time
* **Idea:** Show by induction, every time we "swap" an activity, still have a max size schedule
* **Base case:** $k = 0$ swaps; current schedule is $\boldsymbol{S'}$
* **Ind. step:** Suppose we've swapped in first k activities from $\boldsymbol{S}$
  * Last activity swapped in was a_k
  * Of the activities remaining, $\{i \in A \mid s_i \geq f_{k-1}\}$, a_{k+1} has the earliest f_k
  * Remove a_{k+1}' and add a_{k+1}: can only be moving the finish time earlier
  * Still a (disjoint) schedule *that's the same size*
* $\boldsymbol{S'}$ a max size schedule, so $\boldsymbol{S}$ is too!



The Routing Problem

$d(i, j) = d(j, i)$

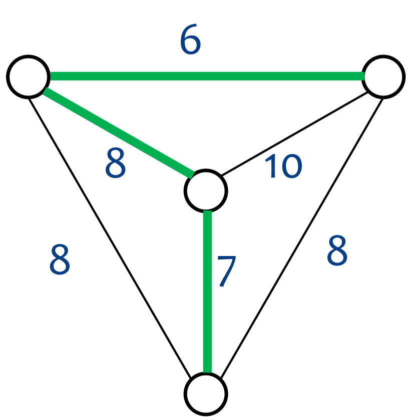
* n cities with *symmetric*, positive distances $d(i, j)$ between city i and j
* **Goal:** Find the *minimum* length of highway needed to ***connect*** the cities, i.e., it is possible to drive from any city to another using the highway



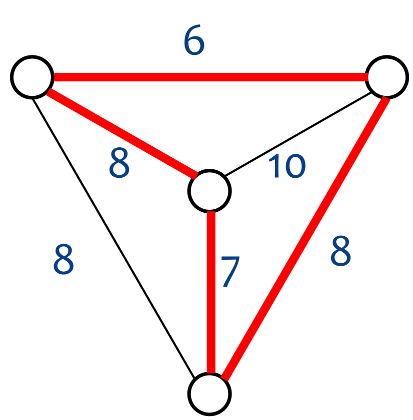
Q: What's the minimum length of highway needed here?

Review: Graph theory

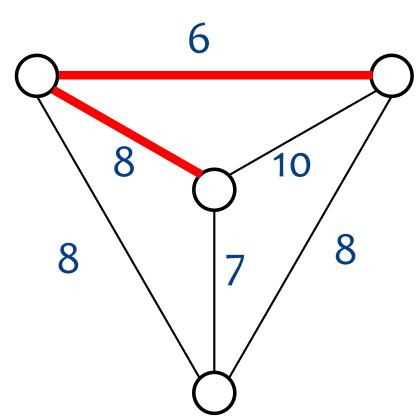
- An ***undirected graph*** has bidirectional edges
- **Fact:** Every ***connected*** undirected graph G has a ***spanning tree***, i.e., a <u>*connected*</u> graph that (i) contains every vertex of G and (ii) does not contain any ***cycles***
- **Goal:** Find a <u>*minimum-weight*</u> spanning tree (***MST***)



(i) and (ii)



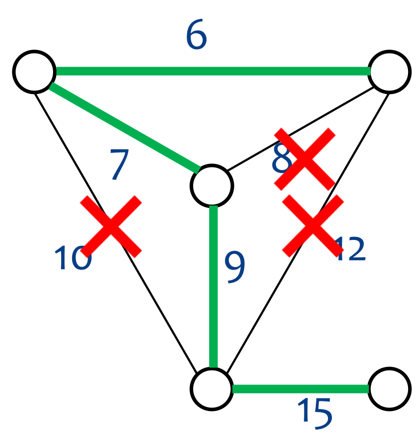
(i) but not (ii)



(ii) but not (i)

Kruskal's Algorithm

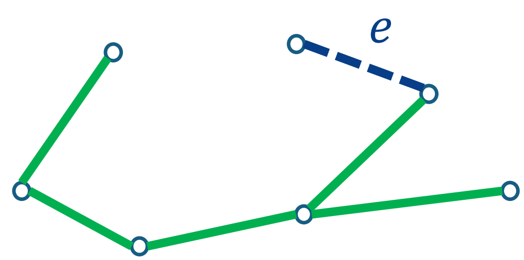
Kruskal(G)**:** // G is weighted, undirected graph
$T \leftarrow \emptyset$ // invariant: T is a *forest* (set of trees) of G
for each edge e in *increasing order of weight*:
  if $T + e$ is acyclic: $T \leftarrow T + e$
return T



Correctness

Kruskal(G): // G is weighted, undirected graph
$T \leftarrow \emptyset$ // invariant: T is a spanning *forest* (set of trees) of G
for each edge e in *increasing order of weight*:
  if $T + e$ is acyclic: $T \leftarrow T + e$
return T

- **Theorem: Kruskal**(G) returns a *minimum* spanning tree T of G
- **Q:** Why does Kruskal return a spanning tree?
  - Suppose you could add some edge e to the output T that doesn't introduce a cycle. Then what happens?



- **Q:** Why does Kruskal return a *minimum* spanning tree?
  - Suppose T' is an arbitrary MST of G.
  - **Goal:** show weight of T = weight of T'.
- **Idea:** show that we can *transform* T' to T *while maintaining the weight* (by induction, "swapping in" an edge of T for an edge of T', one at a time).

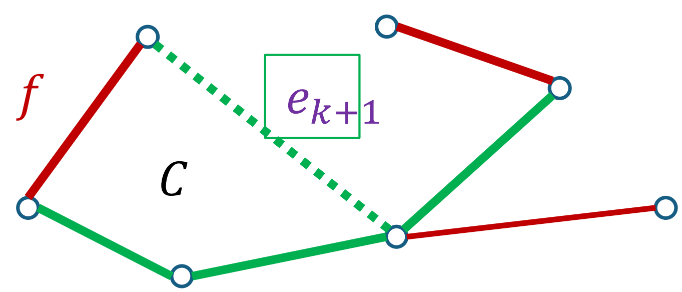
commonly employed strategy to show that a greedy algorithm is optimal

Correctness

Kruskal(G): // G is weighted, undirected graph
$T \leftarrow \emptyset$ // invariant: T is a spanning *forest* (set of trees) of G
for each edge e in *increasing order of weight*:
  if $T + e$ is acyclic: $T \leftarrow T + e$
return T

* Let $e_1, e_2, \ldots$ be the edges of T, *in order of addition to* T
* **Idea:** Show by induction, every time we "swap" an edge, still have an MST
* **Base case:** $k = 0$ swaps; still have T', an MST
* **Ind. step:** Suppose we've swapped in first k edges and it's still an MST
  * Consider the next edge e_{k+1} added to T.
  * If $e_{k+1} \in T'$, MST doesn't change
  * If $e_{k+1} \notin T'$, then adding it creates a cycle C (adding any edge to MST makes a cycle)
  * Since T is acyclic, there is an edge $f \in T'$ on the cycle C.
  * "Swap in e_{k+1}": Remove f and add e_{k+1}. It's still an MST!

Claim: e_{k+1}'s weight $\leq f$'s weight.
* edges added *in increasing order of weight*
* f + first k edges do not form a cycle
  * Kruskal would have considered adding it, but added e_{k+1} instead



Practice

- **Fractional knapsack:** A delivery person wants to load divisible goods, like flour or oil, onto a truck, but the truck can only carry weight W, and they want to maximize the value of the goods. There are v_i dollars worth of each good to be loaded, and weighs w_i pounds in total.
  - **Goal:** How much of each good should be taken to maximize the value of one truckload?
  - **Q:** What is a greedy algorithm to solve this problem? How should we order the goods to be selected?

Goodbye Algorithms…

* Congratulations! You've just finished a crash course on algorithms. (Next: practice, practice, practice.)
* More advanced algorithms and techniques in EECS 477 (*Introduction to Algorithms*).

* On Wednesday we begin ***Computability***.