2019 Nov. 15 - Midterm 2 Prep.

• a more developed solution now

, ?

m 2 next Thursday. See Dieter email for logistics.

day: Advice for structuring solutions.

Greed: 2 ~~paradigms: 2 Greedy Stays ahead & Exchange arguments~~

- Describe your algorithm at a high-level (ignoring implementation details) [1-3 sentences]
- State a claim that implies your algorithm is correct
  ↳ Typically a statement that, for every other solution, greedy does at least as well as it.
- * Prove the claim
- Give a refined implementation including any implementation details (if needed)
- Argue running time. [1-2 sents]

GSA proofs

- Clearly state what is to be proven by induction.
  (Usually your claim should look like "at every step, greedy stays ahead ..." & induction is on # steps.)
- Give an inductive proof (Base case, inductive step.)

↳ The hard part is finding the right way that greedy "stays ahead" & formulating it precisely.

Exchange Arguments

- Locate an exchange in the other solution that makes it closer to greedy.
- Prove that exchange makes the other solution no worse.

Greed with GSA example : HW5 #4 (b).

The problem: Input: n intervals $(s_i, f_i] \in \mathbb{R}$
Output: A set S of points s.t. $\forall i$ $(s_i, f_i] \cap S \neq \emptyset$
and $|S|$ is as small as possible.

Algo: Find ~~largest earliest~~ j s.t. f_j is smallest. Add f_j to S.
Discard intervals ~~with~~ that intersect with S & repeat until no more intervals.

Correctness: Let $g_1 \leq g_2 \leq \cdots < g_k$ be the points in greedy solution
Let $t_1 < t_2 < \cdots < t_\ell$ be the points in an arbitrary solution S.

~~Obsn~~ Claim For all $i = 1, 2, \cdots, n$, $g_i \geq t_i$, where
$g_i = +\infty$ if $i > k$ and $t_i = +\infty$ if $i > \ell$.

Implies my algo correct since ~~...~~, if $k > \ell$, then we would have
g_k is finite ~~but~~ and $t_k = +\infty$, but claim says $g_k \geq t_k$.

Proof of claim: By induction on i.

Base case: $i = 1$. Let f_j be earliest end of an interval. We have $g_1 = f_j$.
Some point in S is in $(s_j, f_j]$.
In particular, $t_1 \leq f_j = g_1$.

Induct. Step: We know $g_{i-1} \geq t_{i-1}$, and we can assume WLOG both are finite.
(if either is infinite, g_{i-1} is infinite, & so g_i is infinite & $g_i \geq t_i$ follows)
Likewise assume g_i is finite.
Let $(s_j, f_j]$ s.t. $f_j = g_i$.
By construction, $s_j > g_{i-1} \geq t_{i-1} \geq t_{i-2} \geq \cdots \geq t_1$.
So there is another point in S that is in $(s_j, f_j]$.
This implies t_i is finite and $t_i \leq f_j = g_i$. $\square$

Implementation:
Sort intervals ~~..~~ to have increasing order of f_j.
Let $e = -\infty$, $S = \emptyset$.
For each interval $(s_j, f_j]$:
  if $e < s_j$:
    $S \leftarrow S \cup \{f_j\}$
    $e \leftarrow f_j$
Return S

Running Time: Sort in $O(n \log n)$ time, then $O(n)$ work.
$\Rightarrow$ Overall $O(n \log n)$.

Greedy with Exchange example : HW6 #1.

Problem: Input: n jobs taking time $t_1 \dots t_n$ with weights $w_1 - w_n$

Output: An order of the jobs so that

$$\sum_{i=1}^{n} C_i w_i$$

is minimized, where C_i is total time of jobs completed before (& including) the i-th.

Algo: Sort jobs in decreasing ~~ratio~~ order of $\frac{w_i}{t_i}$.

Correctness: Claim: This order minimizes $\sum_{i=1}^{n} C_i w_i$.

Proof: Exchange argument. Fix an arbitrary schedule.

Suppose jobs l and r are <u>adjacent</u> in that schedule, (l before r) and out of order w.r.t. our solution. $\left(\frac{w_l}{t_l} \leq \frac{w_r}{t_r}\right)$

Consider swapping their order.

Let C_i be completion times pre-swap, C_i' post-swap.

We have $C_i = C_i'$ for all $i \neq l, r$.

So $\sum_i C_i w_i - \sum_i C_i' w_i = (C_l - C_l')w_l + (C_r - C_r')w_r$.

$C_l - C_l'$ is $-t_r$. $C_r - C_r'$ is t_l.

So $(C_l - C_l')w_l + (C_r - C_r')w_r = -t_r w_l + t_l w_r$.

Since $\frac{w_l}{t_l} \leq \frac{w_r}{t_r}$, this is positive.

ie $\sum_i C_i w_i \geq \sum_i C_i' w_i$. So the new order is no worse.

Since any solution with no adjacent out-of-order pairs <u>is</u> the greedy solution, this proves the greedy solution is optimal. □

Running Time: Sorting takes $O(n \log n)$ time.

Network Flow: (reductions to problems solved using network flow)

- State the problem you reduce to. (Max flow, min cut, proj. selection, etc.) [Est. 1 sentence]

* Describe the reduction. Given input to original problem, describe how to build input to problem you reduce to. Be declarative (the vtxs are ___, the edges are ___, ...)

  ↳ Be careful with pictures. Don't rely on them exclusively. [Est. 3-6 sentences]

- Say how to solve original problem given a solution of problem reduced-to. [Est. 1-2 sentences]

- Prove correctness. Relate solutions of orig. problem to solutions of reduced-to problem

  - May need to make "WLOG" assumptions, such as integrality of a maximum flow. [Est. 4-6 sentences]

- Running Time Analysis:

  Compute size of output instance in terms of size of input instance. [Est. 1-2 sentences]

  * Plug into running time of reduced-to problem's algorithm.

577 Discussion 2019 Nov 15

Network Flow reduction example · HW 8 #3

Problem: Input: n components to be bought from Alpha or Omega.

Alpha cost: $\alpha_1 \cdots \alpha_n$

Omega cost: $\omega_1 \cdots \omega_n$

Incompat cost: $c(i,j)$ if i from Alpha, j from Omega

Output: where to buy each item to minimize total cost.

Reduce to min cut.

Make flow network with vertices $s, t, v_1, \ldots, v_n$;

edges $s \to v_i$ for each $i = 1 \ldots n$ with cap ω_i

$v_i \to t$ " with cap α_i

$v_i \to v_j$ for each i, j with cap $c(i,j)$.

Find min-cap. st-cut: (S, T).

For each i, if v_i in S, buy item i from Alpha

if v_i in T, buy item i from Omega

Correctness: Given a way to buy items from Alpha & Omega (sets A, Ω that partition $[n]$), make a cut $S = \{s\} \cup A$, $T = \{t\} \cup \Omega$.

Its capacity is $\sum_{i \in A} \alpha_i + \sum_{i \in \Omega} \omega_i + \sum_{i \in A,\, j \in \Omega} c(i,j) =$ cost of the buying strategy.

Each cut likewise produces a buying strategy whose cost is the capacity of the cut.

$\Rightarrow$ Finding a <u>min</u> cut gives a buying strategy with minimum cost. $\square$

Running Time: $O(n)$ vtxs, $O(n^2)$ edges.

Using min-cut algo from class, running time is $O(n) \cdot O(n^2) = O(n^3)$.