Assignment 3: Greedy Algorithms

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Ramapriya Ranganath Wisc id: rranganath

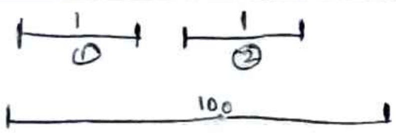
Greedy Algorithms

1. In one or two sentences, describe what a greedy algorithm is. Your definition should be informal, something you could share with a non computer scientist.

Greedy algorithm is a short-sighted algorithm, here the intention is to maximize the profit at each step. Searches for the current optimal solution in each step and ignore the future trends

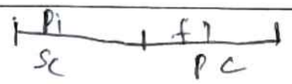
2. There are many different problems all described as "scheduling" problems. In the following questions, pay attention to the details of the problem setup, as they will change each time!

   (a) Let each job have a start time, an end time, and a value. We want to schedule as much value of non-conflicting jobs as possible. Use a counterexample to show that Earliest Finish First (the greedy algorithm we used for jobs with all equal value) does NOT work in this case.



EFS will get value = 2
Optimal should be 100

   (b) *Kleinberg, Jon. Algorithm Design (p. 191, q. 7)* Now let each job consist of two durations. A job i must be preprocessed for p_i time on a supercomputer, and then finished for f_i time on a standard PC. There are enough PCs available to run all jobs at the same time, but there is only one supercomputer (which can only run a single job at a time). The completion time of a schedule is defined as the earliest time when all jobs are done running on both the supercomputer and the PCs. Give a polynomial time algorithm that finds a schedule with the earliest completion time possible.



$\sigma = \{j_1, j_2 \ldots j_n\}$

We use the longest finished time first algorithm, consider job set.

S as empty set, while $\sigma \neq \phi$ do:

~~store~~ σ have j_i with smallest f_i within σ

add j_i to S, remove j_i from σ

end

return S. As we use sorting it takes O(nlogn)

(c) Prove the correctness and efficiency of your algorithm from part (c).

We define schedule A has inversion if i before j but $f_i < f_j$ lemma. All schedules with no inversions and no idle time have the same lateness.

Proof:- We only focus on the job with same f_i, they must be sequential rearrange the order of them won't change lateness

Theorem:- There is an optimal schedule has no inversion and no idle time

We use exchange argument method. Consider we have a optimal ~~solut~~ schedule S^*, if S^* has inversion, we know there is at least one pair of jobs i, j, with i after j $f_i > f_j$

We exchange i, j we have i', j', we have new schedule S'

P is the time all supercomputer jobs finished i.e $P = \sum_{i=1}^{n} P_i$, T_i is the time i finished in S'.

$t_i = \sum_{k=1}^{i'} P_k + f_i$.

Since we take i ahead we have $t_i < t_i^*$

We have $t_j = \sum_{k=1}^{i'2} P_k + f_j \le \sum_{k=1}^{i} P_k + f_i < t_i^*$

Since $f_i > f_j$, thus we have S' is as optimal as S^* until no inversion

3. *Kleinberg, Jon. Algorithm Design (p. 190, q. 5)*

(a) Consider a long road with houses scattered along it. We want to place cell phone towers along the road so that every house is within four miles of at least one tower. Give an efficient algorithm that achieves this goal using the minimum possible number of towers.

Assumptions :- Roads are straight

1) We start from left walk towards right until encountering first house we set one tower 4 miles right away from first house.

2) We again do the same after next four miles

3) Like a Arithmetic progression $a_n = (a_0 \times n) + d$, $d = 4$ miles here

(b) Prove the correctness of your algorithm.

We have Solution $S = \langle i_1, i_2, \cdots i_k \rangle$ denote the set of towers from left to right.

$S^* = \langle j_1, j_2, \cdots j_m \rangle$ denote optimal solutions

Let, r_i, $L < k, m$ denote the right boundary of range of tower i, also the range of $\langle i_1, \dots i_L \rangle$

Similarly for r_i^* in S^*, we use always stay ahead technique

Lemma 1 :- For all r_i, r_i^* we have $r_L \geq r^*_2$

Proof :- By induction $L = 1$ it holds.

Suppose $r = n$ it holds since we chose $(n+1)$-th tower as right as possible while covering the next house.

We have $r_{L+1} \geq r^*_{L+1}$.

Then, Our algorithm produce optimal arrangement.

Proof :- By contradiction assume $k > m$. Since by lemma 1 $r_1 \dots r_m$ can cover $j_1 \dots j_m$ range and $S^* = \langle j_1 \cdots j_m \rangle$ covers all houses we do not need $i_{m+1} \dots i_k$. Thus contradicts

We have S is as optimal as S^*

4. *Kleinberg, Jon. Algorithm Design (p. 197, q. 18)* Your friends are planning to drive north from Madison to the town of Superior, Wisconsin over winter break. They have drawn a directed graph with nodes representing potential stops and edges representing the roads between them.

They have also found a weather forecasting site that can accurately predict how long it will take to traverse one of the edges on their graph, given the starting time t. This is important because some of the roads on their graph are affected strongly by the seasons and by extreme weather. It's guaranteed that it never takes negative time to traverse an edge, and that you can never arrive earlier by starting later.

(a) Design an algorithm your friends can use to plot the quickest route. You may assume that they start at time $t = 0$, and that the predictions made by the weather forecasting site are accurate.

Let S be the set of explored nodes. For each $u \in S$ we store a distance $d(u)$

Let l_e ($e = (u,v)$) denote the set of all nodes (locations)

while $S \neq V$

select node v s.t $v \notin S$, v is one edge from S

$d'(v) = \min_{e=(u,v)} [d(u) + l_e]$

add v to S and define $d(v) = d'(u)$

if v = sup (superior)

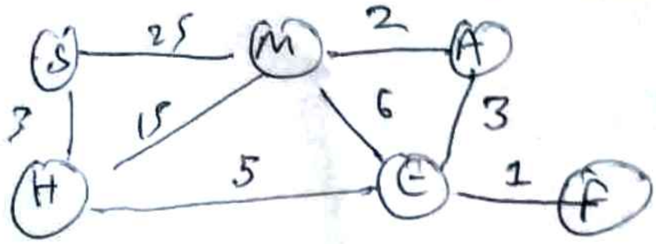
break

End

- Finding shortest path, we start from S_p and find the edge (u, S_p) the last step ($u \in S$) then we locate node u and find the edge (y, u) at step where u added to S ($y \in S$). We do this recursively until we reach Madison [superior]
- Then we have all edge together in path from ~~Sp~~ to Madison
- We reverse the path and get results

(b) Demonstrate how your algorithm works using a small example with 6 nodes. Your demonstration should include any data structures you maintain during the execution of your algorithm and any queries you make to the weather forecasting site. For example, if your algorithm maintains a "current path" that grows from (M)adison to (S)uperior, you might show something like the following table:

Path	Total time
M	0
M,A	2
M,A,E	5
M,A,E,F	6
M,A,E	5
M,A,E,H	10
M,A,E,H,S	13



~~Strat~~ ~~Start fr~~

Start from M, explore the nodes near M, chose the nearest one, add it to linked list, at each step we have a linked list showing the shortest path from M to the new added nodes. We also have one list noted the numerical value for each step, the value represents the time during the shortest path. We terminate until we reach S, we have the linked list showing the path and the value showing time consumption.