CS 577 (Epic) # Assignment 3 – Greedy Algorithms Spring 2021

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: Zhuoyan Xu Wisc id: ZXU444

Greedy Algorithms

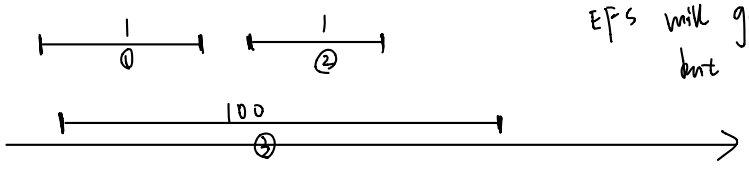
1. In one or two sentences, describe what a greedy algorithm is. Your definition should be informal, something you could share with a non computer scientist.

Solution: Greedy algorithm is a short-sighted algorithm trying to maximize local gain (profit) at each step. It search for the current optimal solution in each step and ignore the future trend.

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.

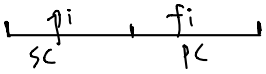
Solution:



EFS will get value 2. but 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.

Solution:



$\sigma = \{j_1, \cdots j_n\}$

we use the longest finished time first algorithm, consider job set

Initial S as empty set.

While $\sigma \neq \phi$ do:

choose j_i with smallest f_i within σ, (break ties arbitrarily)

add j_i to S, remove j_i from σ

end

return S.

this requires sorting jobs based on f_i. cause $O(n \log n)$

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

Solution: 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.

pf: we only focus on the job with same f_i, they must be sequential, rearrange the order of them won't change lateness.

Thm: There is an optimal schedule has no inversion and no idle time.

pf: we use exchange argument technique. Consider we have a optimal schedule S^*.

- If S^* has inversion, we know there is at least one pair of jobs i, j, with i after j. $j \cdots i$ $i, \cdots j'$ $f_i > f_j$.
- we exchange i, j, we have i', j' j' after i', we have new schedule S'
- define P is the time all supercomputer jobs finished. ie. $P = \sum_{i=1}^{n} p_i$, l^*_i is the time i finished in S^* l'_i is the time i' finished in S'. similar for j, j'. we know: $l^*_i = \sum_{k=1}^{i} p_k + f_i$, $l'_i = \sum_{k=1}^{i'} p_k + f_{i'} = \sum_{k=1}^{j'} p_k + f_i$
- Since we take i ahead, we have $l'_i < l^*_i$
- we have $l'_j = \sum_{k=1}^{j'} p_k + f_{j'} = \sum_{k=1}^{i} p_k + f_j \leq \sum_{k=1}^{i} p_k + f_i < l^*_i$ since $f_i > f_j$. then we have S' is as optimal as S^* we repeat these steps until no inversions. ∎

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

(a) Consider a long, straight 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.

Solution: Consider the road as straight lines with some points known as houses

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

② we start from the right boundry of range of previous tower. when we encounter another house, we repeat ①.

③ we repeat ①. ② until all the houses are covered.

(b) Prove the correctness of your algorithm.

Solution: we use $S = \langle i_1, i_2, \dots i_k \rangle$ denote the set of towers from left to right
$S^* = \langle j_1, j_2, \dots j_m \rangle$ denote the optimal solution.

let r_l, $l < k, m$ denote the right boundary of range of tower i_l, also the range of $\langle i_1, \dots i_l \rangle$. similar for r^*_l in S^*. we use always stays ahead technique

Lemma 1: for all r_l, r^*_l. we have $r_l \geq r^*_l$

pf: by induction. $l=1$ it holds.
suppose $r=n$ it holds. since we choose $(n+1)$-th tower as right as possible while covering the next house. we have $r_{l+1} \geq r^*_{l+1}$ ∎

Thm: Our algorithm produce optimal arrangement.

pf: By contradiction, assume $k > m$. since by lemma 1, $i_1, \dots i_m$ can cover $j_1, \dots j_m$ range, and $S^* = \langle j_1, \dots j_m \rangle$ cover all the houses. we do not need $i_{m+1}, \dots i_k$. Then it 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.

Solution: 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 time take from u to V
Initialize $S = \{M\}$. $d(M) = 0$. let 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 $d(v) = d'(v)$
  if $v = Sup(Superior)$:
    break

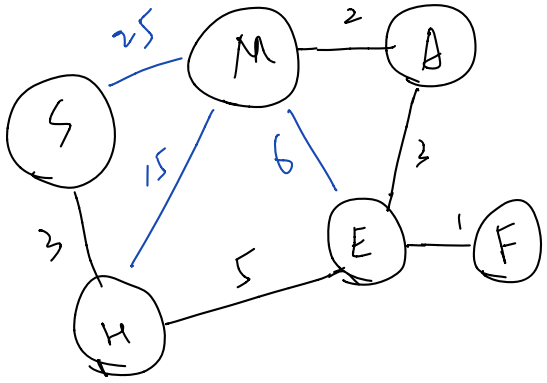
End

- to find shortest path, we start from Sup and find the edge (u, Sup) 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 M(adison)
- Then we have all edge together is path from Sup to M
- we reverse the path, get result.

(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

Solution:



Start from M, explore the nodes near M, choose 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 (shown in the table). We also have one list stored 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 the time consumption.

Coding Question

5. Implement the optimal algorithm for interval scheduling (for a definition of the problem, see the Greedy slides on Canvas) in either C, C++, C#, Java, or Python. Be efficient and implement it in $O(n \log n)$ time, where n is the number of jobs.

   The input will start with an positive integer, giving the number of instances that follow. For each instance, there will be a positive integer, giving the number of jobs. For each job, there will be a pair of positive integers i and j, where $i < j$, and i is the start time, and j is the end time.

   A sample input is the following:

   ```
   2
   1
   1 4
   3
   1 2
   3 4
   2 6
   ```

   The sample input has two instances. The first instance has one job to schedule with a start time of 1 and an end time of 4. The second instance has 3 jobs.

   For each instance, your program should output the number of intervals scheduled on a separate line. Each output line should be terminated by a newline. The correct output to the sample input would be:

   ```
   1
   2
   ```