CS 577 # Assignment 3: Greedy Algorithms Fall 2023

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: Seyeong Oh Wisc id: 9084744136

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.

The greedy algorithm makes the locally optional choice at each step with the hope of finding a golbal optimum. It chooses the best option without reconsidering previous choices, which may or may not lead to the most optimal solution.

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.

JobA : {startTime = 1, endTime = 4, value = 5},
JobB: {startTime = 2; endTime = 3, value = 4},
JobC: {startTime = 3, endTime = 6, value = 10}
If we apply the EFF algorithm, it would first select JobB first, but the value of JobB is less than JobA and JobC.
Therefore, it fails to maximize the total value of non-conflicting jobs in this case.

(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.

To find a schedule with the earliest completion time possible, we can use a greedy algorithm:

1. sort the jobs in decreasing order of their processing times (pi + fi) -> O(nlogn) using quicksort or mergesort
2. Initialize currentTime and completionTime = 0
3. Schedule jobs on the supercomputer while updating currentTime/completionTime
4. add its finishing time to currentTime
5. Return completionTime(will hold the earliest completionTime)

The time complexity of this algorithm is dominated by the sort step, so it will be O(nlogn) in the worst case

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

Corectness:
The greedy algorithm says that a global optimum can be arrived at by making a locally optimal choice. In part(b), the greedy choice is scheduling jobs with longer preprocessing times pi first on the supercomputer and then available standard PC. This local optimum ensures that shorter tasks don't get stuck behind longer ones for processing in the queue for the supercomputer.

The algorithm schedules jobs sequentially, and this means that the sub-problem of scheduling jobs up to a certain point is solved optimally.

Therefore, if we schedule jobs according to their decreasing order of pi, we will get an earliest completion time possible because no PC stays idle while there are tasks ready for processing after their preprocessing stage

Efficiency:
The efficiency of this greedy algorithm depends mainly on sorting jobs based on their preprocessing times pi. Sorting takes O(nlogn) time compexity can be used such as mergesort.

after sorting, we need the schedule each job one by one onto supercomputer -> O(n) and start each job immediately after finishing preprocessing onto PC -> O(n)

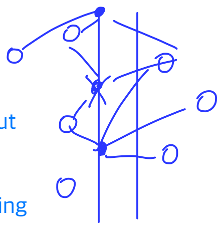
The total complexity would be O(nlogn)+O(n)+O(n) = O(nlogn)
Thus, this algorithm is efficient.

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.

we can solve it using greedy algorithm.
First, we strart from one end of the road with no towers place yet.
Second, if the house is already within four miles of previously place tower, we move on to consider the next house. If this house isn't within range of any existing tower, we need to place a new tower. But instead of placing it right at this house's position, we put it 4 miles further down the road. (ensures any subsequent houses will also be covered)

This solution works because placing each tower as far forward as possible without leaving any previous house uncovered, minimizing total number of towers



(b) Prove the correctness of your algorithm.

To prove the correctness of the algorithm, we can use induction.

Base Case: When there is only one house, we place a tower at the position of the house + 4 miles.
Then, think there are two houses within 4 miles, placing a tower 4 miles from the first house will cover both.
If they are more than 4 miles apart, each will need one tower - also optimal

Inductive Step: Assume this algorithm works for n houses, and consider adding an additional n+1th house.
- CASE1 : n+1th house is within 4 miles from previously placed tower
  No new tower is needed to cover
- CASE2 : n+1th house is not within 4 miles from previously placed tower
  Place a new one as far forward as possible without leaving this house uncovered. (any future houses (n+2)th, (n+3)th··· within those next 4 miles would be covered by this same tower.

Therefore, if this algorithm works optimally for n houses, then it will also work optimally for n+1 houses.

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.

In this case, we can use Dijkstra's algorithm to accomodate time-dependent edge weights.

First, we need to initialize the starting node and earliest arrival time as 0. Initialize all other nodes' earliest arrival times to infinity, because we haven't found a route yet.

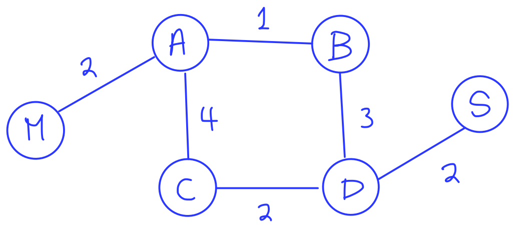
Second, for each node adjacent to the active node,
- calculate the new arrival time at this adjacent node if you took the edge from active node to it.
  (use weather forecasting site's prediction based on current earliest arrival time)
- If the new potential arrival time is earlier than current earliest arrival time, update it with this new one

Third, Once you've done all edges from your current active node, choose a new active node, and repeat step two and three until you've dteremined shortest paths to all nodes (or found destination superior)

This algorithm will find the quickest route by considering weather predictions.

(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



Node	Distance	Predecessor	Priority Queue
M	0	None	W(0,A):2
A	2	M	W(2, C):4 W(2, B):1
B	3	A	W(2, C):4 W(1, D):3
C	6	A	W(2, C):4 W(3, C):2 W(3, S):2
D	6	B	W(2, C):4 W(3, C):2
S	8	D	

The Shortest Path: M A B D S, which total time of 8