HW3

Wednesday, February 15, 2023 3:58 PM

CS 577

Assignment 3: Greedy Algorithms

Spring 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: Carson Miller Wisc id: 9081473028

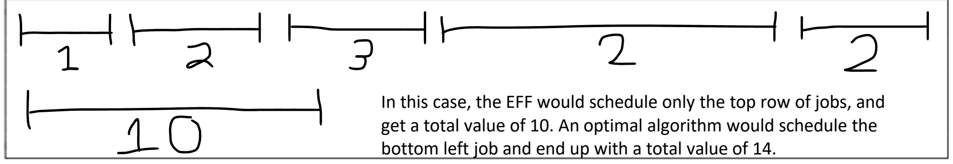
Greedy Algorithms

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.

A greedy algorithm is an algorithm that makes the locally optimal decision. It makes the best possible decision at each step based on the available information, with no regard for future steps or data

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.



In this case, the EFF would schedule only the top row of jobs, and get a total value of 10. An optimal algorithm would schedule the bottom left job and end up with a total value of 14.

(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.

Earliest Completion First

- Sort jobs by $c_i = p_i + f_i$ from lowest to highest
- Schedule the job with the lowest total time c_i ($p_i + f_i$) at time $t = 0$
- Schedule subsequent jobs to start at time $t = p_{i-1}$
- After a job finishes on the supercomputer move it a PC to finish

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

By definition, ECF is correct because it schedules all jobs. I will use the Stays Ahead Analysis to prove its efficiency and optimality.

Let S denote my ECF algorithm and S* denote an optimal solution.
S = <i_1,…,i_k> such that $c_{iu} < c_{iv}$ for u < v
S* = <j_i,…,j_m> such that $c_{ju} < c_{jv}$ for u < v

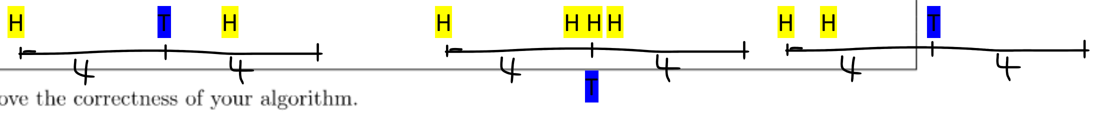
I will prove that for all i_r, j_r with r <= k, we have $c_{ir} < c_{jr}$.
The proof is by induction

- For r = 1, the claim is true because ECF selects the job with smallest completion time ($p_i + f_i$).
- Assume true for some r.
  - By the induction hypothesis, we have $c_{ir} < c_{jr}$
  - The only way for S to fall behind S* would be for ECF to choose a job *t* where $f_t > f_{j(r+1)}$, but this is a contradiction because ECF chooses the job with the smallest total completion time at every step.

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.

1. Start at the house farthest in one direction along the road
2. Drive 4 miles towards the other end of the road, place a tower there, and drive another 4 miles.
3. From the next house along the road, drive 4 miles, place a tower, and drive another 4 miles.
4. Repeat step 3 until you reach the other end of the road.
5. If you reach the end of the road while driving 4 miles before placing a tower, place a tower at the end of the road



(b) Prove the correctness of your algorithm.

There are essentially 3 states that we will be in when executing this algorithm: Search for house, drive to tower, and drive from tower. Search for house is exactly how it sounds, we are driving along the road searching for a house. Drive to tower is a state following "Search for house," where we are watching our odometer (which we reset when we finally found a house in the previous state) to tell us when we have driven 4 miles. Drive from tower is a state following "Drive to tower," where we are watching our odometer (which we reset when we placed the tower at the end of the previous state) to tell us when we have driven 4 miles. Following those 4 miles, we re-enter "Search for House."

Let h be some house along this road. We start in state "search for house" from one end of the road. If h is the first house we see, then we will drive 4 miles from it, place a tower, and h will be within 4 miles of a tower. If we see h while in state "drive to tower," then we finish driving our 4 miles, place a tower, and h is <4 miles from a tower. If we see h while in state "drive from tower," then we are passing the house <4 miles from the tower we just placed. If we pass h while in the "Drive to tower" state and then reach the other end of the road before racking up our 4 miles, place a tower at the end of the road and h will be <4 miles from a tower.

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.

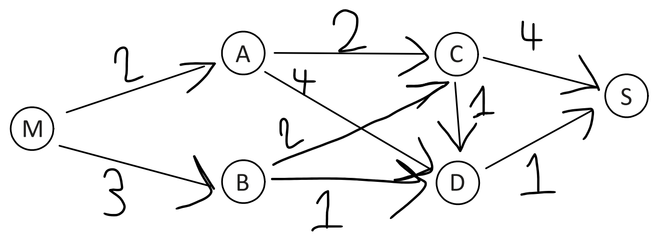
   (a) Design an algorithm your friends can use to plot the quickest route. You may assume that they start at time $t = 0$, and that the predictions made by the weather forecasting site are accurate.

Let *P* be the set of explored potential stops. For each stop in *P*, we store a *value t* that represents the total time to that stop from Madison. We start with only Madison in P.

1. Of potential stops that are only 1 outgoing edge away from all nodes in *P*, choose one we have not yet explored that is the smallest time *t* away from Madison, breaking ties by shortest outgoing (from P) edge time.
2. Add that node to *P* and store the total time *t*.
3. Repeat 1-2 until we have explored every potential stop, including Superior.

(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



Iteration	Node added to *P*	Time *t*
1	A	2
2	B	3
3	D	4
4	C	4
5	S	5

Node	Shortest Path from M
M	M
A	M, A
B	M, B
C	M, A, C
D	M, B, D
S	M, B, D, S

Starting at M, we see A and B and choose A because 2 < 3. Now that *P* = {M, A}, we see B, C, and D. We choose B because it is only 3 from M, while C is 4 from M and D is 6 from M. Now *P* = {M, A, B} and we see C and D. We choose D because it is 4 away (C is also 4 away but it's time outgoing from *P* is 2 which is greater than D's of 1). Now *P* = {M, A, B, D} and we see C and S. We choose C because it is only 4 away from Madison, while S is 5 away. Now we have *P* = {M, A, B, D, C} and see S. We choose S because it is all we see, but we pick the path from D because it's time is lower.

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
```