CS 577

Assignment 3: Greedy Algorithms

Spring 2024

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: Ki Min Kang Wisc id: 908 404 5062

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 chooses the best option at each step, but that might not lead to the best result overall in the end.

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.

Let Job A runs from time 0 to 4 and has a value of 2
Job B runs from time 1 to 3 and has a value of 5
With Earliest Finish First, we will choose Job A,
but Job 2 is optimum.
This counter example shows that "Earliest Finish First" algorithm does not always result in optimum value.

(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.

The "Longest Finish Time First" algorithm sorts jobs by finishing times in descending order and it takes O(NlogN) time, where N is the number of jobs. As soon as a job finishes preprocessing on the super-computer, it is immediately assigned to a stanadard PC to complete the finishing process. This strategy aims to the earliest overall completion time for all jobs.

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

The "Longest Finish Time First" (LFTF) scheduling algorithm sorts jobs by their finishing times on a PC, not considering preprocessing time on a supercomputer.

If an optimal algorithm chooses any job earlier than LFTF does, but that job finishes at the time or sooner, we can swap the job with the one chosen by LFTF without affecting completion times.

This shows that swapping jobs will not delay the overall schedule and all jobs can be rearranged into the LFTF order.

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 start from the one end of the road, and follow it until the end.
When we reach a house not within the range of a tower, continue four miles further and then place a new tower there.

(b) Prove the correctness of your algorithm.

We will use induction to show the placement of each cell phone tower by the greedy algorithm is at least as good as the placement by any optimal algorithm.

Base Case:
The first tower placement by the greedy algorithm (g_1) must be at least as far along the road as the first tower placed by an optimal algorithm (o_1), because both must cover the first house.
The greedy algorithm chooses the position as far along the road as possible within it cover the first house.
$g_1 \geq o_1$, therefore the base case holds.

Inductive Step:
Assume that up to the $k-1$-th tower, the greedy algorithm's towers are at least as far along the road as the optimal algorithm's towers ($g_{k-1} \geq o_{k-1}$)
If the k-th house is already covered by the $k-1$th tower placed by the greedy algorithm, there is no need for an additional tower at that house. If it was not covered, the greedy algorithm will place the next tower at the farthest point.
$g_k = x_h$ (the position of hth house) $+ 4$. Since $x_h - g_{k-1} > 4$, house h is not in the range of g_{k-1}, and thus $o_k \leq x_h + 4 = g_k$ to cover house h.
This maintains the greedy algorithm's "stay ahead" of the optimal algorithm.

Therefore, the greedy algorithm never falls behind the optimal algorithm in tower placement.

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.

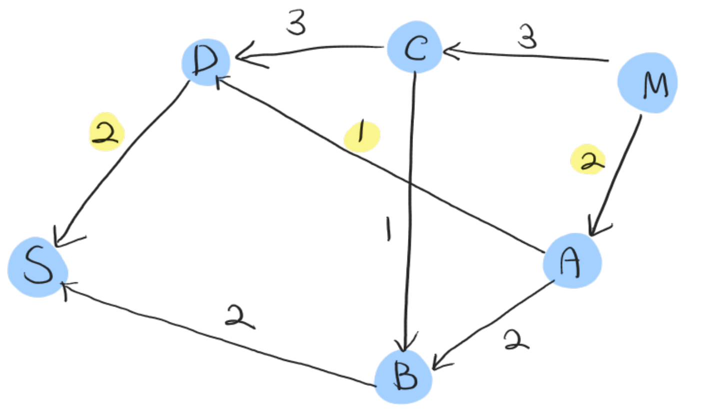
   They have also found a weather forecasting site that can accurately predict how long it will take to traverse one of the edges on their graph, given the starting time t. This is important because some of the roads on their graph are affected strongly by the seasons and by extreme weather. It's guaranteed 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.

Using Dijksetra's shortest path algorithm, we can find the quickest route from Madison. As each node is added to the shortest path tree, the forecasting site finds the time it will take to traverse outgoing edges from the current node at current time. The algorithm account for road 'lengths' that change due to weather, updating the shortest path in real-time. To easily track the route, the shortest path tree includes not just the shortest distance to each node but also the last edge used to reach that node. Therefore, this algorithm examines all possible paths from the current location, selects the quickest one at that moment, continuously updating the shortest path with the weather forecast information.

(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



Current Node	Distance	Predecessor	Possible next path
M	0	None	M→A (2), M→C (3)
A	2	M	A→D(1), A→B(2)
C	3	M	C→D (3), C→B(1)
D	3	A	D→S(2)
B	4	A	B→S(2)
S	5	D	

Starting at node M(Madison) to S(Superior), we explore the shortest Path using Dijkstra's algorithm.

The reconstructed path from M to S is: M→A→D→S

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, k, giving the number of instances that follow. For each instance, there will be a positive integer, n, giving the number of jobs. For each job, there will be a pair of positive integers i and j, where $i \leq j$, and i is the start time, and j is the end time.

   Input constraints:

   - $1 \leq k \leq 1000$
   - $1 \leq n \leq 100000$
   - $\forall i, j : 1 \leq i \leq 100000 \land 1 \leq j \leq 100000$

   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
```