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: _____ Wisc id: _____

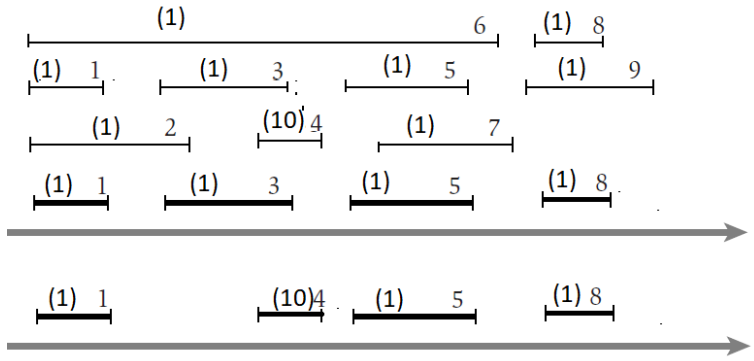
## Greedy Algorithms

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:**
The greedy algorithm is an optimal solution (minimum, maximum) problem seeking model that involves making the locally optimal choice at each stage of the algorithm, with the expectation that the globally optimal solution will eventually be reached. The greedy algorithm selects the best available option at each step without considering the future consequences. Greedy algorithms are useful in situations where making the best decision at each step leads to a globally optimal solution. However, there are situations where greedy algorithms may not always provide an 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.

**Solution:**
All the activitiees have the same value - 1, except activity 4 with value 10. Optimal using Earlyest Finish First (the greedy algorithm we used for jobs with all wqual values) gives solution 1, 3, 5, 8 while optimal solutions seeking for as much value is 1, 4, 5, 8 (but also 2, 4, 7, 9 and others)




   (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:**
We can have 2 different situations

All the jobs have to be finished as one job on the supercomputer and after last finished job they are released to start in parallel on standard PC. In this case order of the execution is irrelevant. The completion time of execution will be $\sum_{i=1}^{n} p_i + max_{i=1}^{n}(f_i)$, which is O(n). Any swap of jobs with durations $p_i$, $p_j$ will not change the sum $\sum_{i=1}^{n} p_i$, that is commutative and associative operation, nor swaps influence the second term $max_{i=1}^{n}(f_i)$

If each job can be released from execution on the supercomputer after finished its turn in sequence, than optimal schedule with minimum execution time will be :
$T$ : Run jobs in the order of decreasing finishing time $f_i$

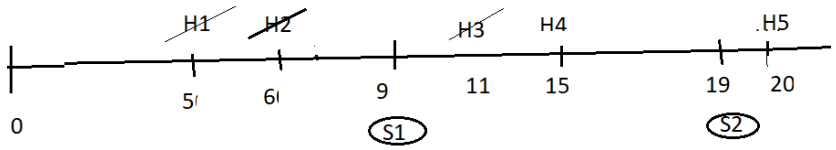
(c) Prove the correctness and efficiency of your algorithm from part (c).

**Solution:**
In the second case if we ordered execution on the supercomputer to go in the descending order of $f_i$ for $i = 1$ to n, we will still have $\sum_{i=1}^{n} p_i$ for completion time on the supercomputer, that do not depend on swapping the order of the jobs. But the second term now will be less than $max_{i=1}^{n}(f_i)$. because now execution on the regular PCs of the part of the jobs can go in parallel with execution on the supercomputer. While with ordering the execution of jobs on supercomputer will shorten the overall time of execution. On the other hand sorting the $f_i$, $i = 1..n$ need nlogn operations. So the complexity is now O(nlogn).

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:**




Narrative description: Starting from the west end of the road, going east, place in the list all destinations of the houses regarding the beginning of the street, in ascending order. Place the first base station S1, 4 miles east of the first house (in our case house H1 placed at 5th mile from beginning of the road so station will be placed on 9th mile). Remove from the list all the houses in the range 4 miles east and west of that station (in our case H1, H2, H3). Repeat the procedure for the next house from the list (in our case H4), and place S2, 4 miles east of H4 (that is at 19th mile). It will cover H4 and H5.

(b) Prove the correctness of your algorithm.

**Solution:**
Let the first base station $s_1$ covers all the houses in the range $(0, s_1)$, within 4 miles of its positions, that is placed at the most east position of the start of the road. Let $s_{i+1}$ be the next station, placed the most east of position of previous $\{s_1, s_2, \ldots, s_i\}$ stations, sorted in increasing order, that together cover within 4 miles, all houses between $s_i$ and $s_{i+1}$. Now let $S = \{s_1, \ldots . s_n\}$ denote some solution of our algorithm, and suppose that $Q = \{q_1, \ldots . q_m\}$ is an optimal solution. To prove correctness of our algorithm we have to prove that n=m. Suppose that our S stays ahead than Q, and $s_i \geq q_i$ for each i. By induction: For i=1 we chose $s_1$ to be as far as possible to the east so $s_1 \geq q_1$. Assume that this is true for some $i \geq 1$. This means that ordered stations $\{s_1, \ldots . s_i\}$ cover all the houses also covered by stations $q_1, \ldots q_i$. We now add $q_{i+1}$ to the stations $\{s_1, \ldots s_i\}$, so all the houses between $s_i$ and $q_{i+1}$ are also covered. By our algorithm $s_{i+1}$ is placed the most east of $s_i$, with 4 miles distance feature so it follows that $s_{i+1} \geq q_{i+1}$. Suppose now that $n > m$. This means that $\{s_1, s_2, \ldots, s_m\}$ do not cover all the houses. But by induction we prove that $s_{i+1} \geq q_{i+1}$ for all i. This means that also $\{q_1, \ldots . q_m\}$ is not optimal solution that contradict to our assumption

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.

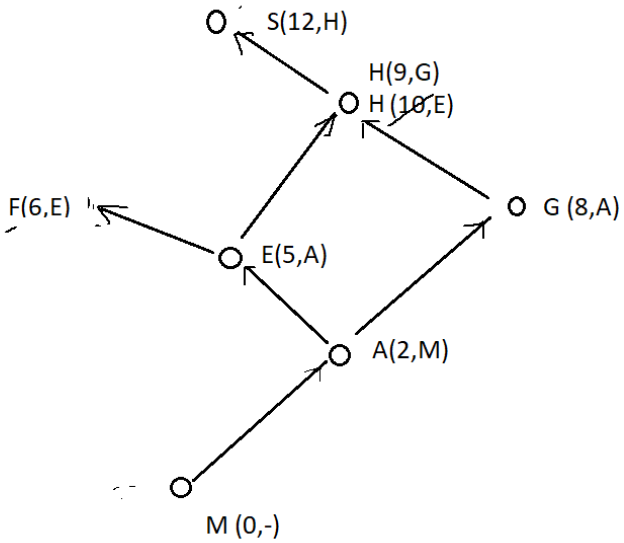
**Solution:**
The problem we have is to find the shortest travel time in a directed graph between two nodes, as also the fastest way of travel. The particularity is that the weight of each edge in the graph is not a fixed value, but depends on the summed travel times of the previous edges. For example if we reach the node u at the time d(u), and there is an edge e=(u,v), than by weather forecast we will reach the node v at a predicted time fe (u). We will also track the shortest travel path by keeping at each node fe(u) the value of the previous node from which we reach the minimum travel time to this node r(v)=u. (i.e. is a positive and monotone function).

Let S be the set of explored nodes.
For each $u \in S$, we store the earliest time d(u) when we can arrive at u
and the last r(u) before u, from which we come on the fastest path to u
Initially $S = \{s\}$ and d(s)=0.
While $S \neq V$
Select a node $\nu \notin S$ with at least one edge from S for which
$d'(\nu) = \min_{e=(u,\nu):u \in S} f_e(d(u))$ is as small as possible.
Add $\nu$ to S and define $d(\nu) = d'(\nu)$ and $r(\nu) = u$
Endwhile

(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:**




| Path | Total time |
|---|---|
| M | 0 |
| M,A | 2 |
| M,A,E | 5 |
| M.A.G | 8 |
| M,A,E,F | 6 |
| M,A,E | 5 |
| M,A,E,H | 10 |
| M.A.G | 8 |
| M,A,G,H | 9 |
| M,A,G,H,S | 12 |

The algorithm can be implemented using a priority queue, as in the basic version of Dijkstra's algorithm: when node u is added, we look all edges (u, v), and change their keys according to the travel time from u. This takes time O(log(n)) per edge, for a total of O(mlog(n)).

## 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
   ```