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: Zinnia Nie Wisc id: 908 319 4044

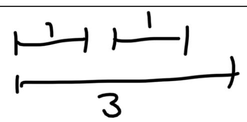
Greedy Algorithms

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 picks the best thing to do at each step, treating the input as if it is the last one.

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.



With this sequence of jobs, the two first finish jobs each with value 1 are scheduled for total of value 2. However, scheduling the last finish job with value 3 is optimal.

   (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.

Longest finish time first

Sort jobs by decreasing finish time f_i

return the array

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

Preprocessing time does not affect overall job time because there is only one super computer, so the jobs are essentially just arriving to PCs one by one. We want the longest finish time to start running first so that they can be completing while the supercomputer is preprocessing other jobs. This will minimize completion time.

Exchange Argument

- Let S be a schedule given by the greedy algorithm without any inversions.
- Any Schedule with no inversions has the same completion time.
  - No inversions means schedules only differ between jobs with same finish time. The end time of the two jobs does not depend on the order because they will be processed and run for the same time no matter their order.
- There is an optimal schedule with no inversions.
  - Let S' be an optimal schedule. Let there be an inversion in S' where two jobs J_i and J_j that are scheduled such that J_j is before J_i but $f_i > f_j$. We want to show that completion time T' after swapping the inversion is $T' \leq T$.
  - The preprocessing time doesn't change as they are processed one by one.
  - Let the completion time before the swap be t. The last job running is the last processed job, which is J_i. Thus after the swap, J_i is processed earlier and will end earlier than t. Since preprocessing time doesn't change, J_j ends at the same time. Then because $f_j < f_i$, the completion time $T' < T$.
  - Therefore, the completion time of the schedule with no inversions is ≤ optimal time with inversion

Since schedules with no inversions have same completion time, the greedy schedule is ≤ optimal time.

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.

Sort the distances from closest to farthest
Set a tower at the farthest reach for the first distance (distance[1] + 4)
Num of towers = 1
For the remaining distances:
  If distance [i] is out of range (>4 miles away from tower):
    Set tower at distance [i] + 4
    Num of towers = Num of towers + 1
end

(b) Prove the correctness of your algorithm.

Since we are placing the towers are the farthest edge from the first uncovered house, the distance between each is maximized so the number used is minimized.

Stays Ahead

- Let M be the solution created by the Greedy algorithm
- Let O be an optimal solution
- let d_m be distance where a cell tower is placed in greedy and d_o be where it is placed in optimal.
- Then for m_i, o_i, $d_{m_i} \geq d_{o_i}$.

Induction: Base Case: One tower, all houses are within 8 mi of the first house, so optimal also has one tower

Inductive: $d_{m_K} \geq d_{o_K}$
When choosing the next tower location, the only way for $d_{m_{K+1}} < d_{o_K}$ is for $d_{m_{K+1}}$ to not be the farthest away which contradicts the algorithm or for optimal to not cover a house which would not be optimal. Therefore $d_{m_i} \geq d_{o_i}$.

Since distance between towers is maximized while maintaining coverage, the fewest towers are placed in a stretch of road.

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.

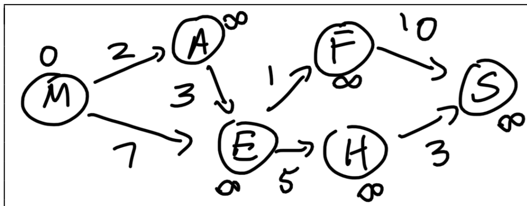
   (a) Design an algorithm your friends can use to plot the quickest route. You may assume that they start at time $t = 0$, and that the predictions made by the weather forecasting site are accurate.

<u>Dijkstra's Algorithm</u>

```
Let S be a set of unvisited nodes
Let V be an empty set of visited nodes
Assign start node a distance value d(v) = 0
Assign all other nodes distance d(v) = ∞
while destination node ∉ V :
    If current ∈ visited:
        continue
    For each unvisited neighbor:
        If current node d(v) + edge weight < neighbor d(v)
            update neighbor d(v) to the lower value
            update neighbor parent to current node
            put neighbor in min heap
    end
    Put current in visited
    set current to node returned by pop heap
end
Starting from destination node, trace back through
parent pointers to get the shortest path
```

(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



Intial

0. unvisited: {A, E, F, H, M, S}
visited: { }
Min-heap: { }
parents: { }
distances: {A - ∞, E - ∞, F - ∞, H - ∞, M - 0, S - ∞}

1. unvisited: {A, E, F, H, S}
visited: {M}
Min-heap: {A - 2, E - 7}
parents: {A → M, E → M}
distances: {A - 2, E - 7, F - ∞, H - ∞, M - 0, S - ∞}

2. unvisited: {E, F, H, S}
visited: {M, A}
Min-heap: {E - 5, E - 7}
parents: {A → M, E → A}
distances: {A - 2, E - 5, F - ∞, H - ∞, M - 0, S - ∞}

3. unvisited: {F, H, S}
visited: {M, A, E}
Min-heap: {F - 6, E - 7, H - 10}
parents: {A → M, E → A, F → E, H → E}
distances: {A - 2, E - 5, F - 6, H - 10, M - 0, S - ∞}

4+5. unvisited: {H, S}
visited: {M, A, E, F}
Min-heap: {H - 10, S - 16}
parents: {A → M, E → A, F → E, H → E, S → F}
distances: {A - 2, E - 5, F - 6, H - 10, M - 0, S - 16}

6. unvisited: {S}
visited: {M, A, E, F, H}
Min-heap: {S - 13, S - 16}
parents: {A → M, E → A, F → E, H → E, S → H}
distances: {A - 2, E - 5, F - 6, H - 10, M - 0, S - 13}

From destination nodes trace back parents to get:

M → A → E → H → S

which is the shortest route

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
```