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: Pei Yuan Li Wisc id: pli233 / 9082321614

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 a method that focuses on making a locally optimal choice at each small step when solving a problem. It is easy to implement, straightforward, and fast. However, it does not guarantee a optimal solution in global schema.

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.

start	end	value	
0	1	1	1)
1	3	2	2)
0	3	100	3)

For the example on the left, we will select interval 1 and 2 and give up 3 based on Earliest Finish First. However, we in fact should take interval 3 solely to maximize our value. Thus, finish first not wor

   (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.

Since the finish time of p_i is completely determined by the supercomputer, and it only run one job at a time, p_i is not affected by order.

Thus, put the largest finish time of f_i at first can maximize performance overall by reducing idle of super computer.

Algorithm: sort job by f_i decreasingly, do job with this order.

Time complexity: $O(n \log n)$, since there is sorting included.

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

Show by exchange:

We have an Optimal Schedule G, does jobs follow decreasing t_i. Consider a schdule S which does not completely follow decreasing order as G. S has two jobs J_i and J_k which what J_i is right completed before J_k. ($J_i \to J_k$) And $f_i < f_k$.

We can optimize S by exchange the schedule order J_i and J_k, this new Schedule is S'.

Time on Super Computer: $P_i + P_k = P_k + P_i$, no increasing cost

Time on PCs: S: J_k finish after J_i since J_k has to wait for P_k and have larger f_k.

S': J_k begin earlier on supercomputer, thus finish eailier than in S. J_i ends the same time as J_k did in S. $\because f_i < f_k$, J_i finishes earlier than J_k in S.

Overall, this swap does not create latency and we can change any S to G by this swap. Thus, G is optimal

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.

While there is still house in the set H;
From left side of the Road r, move a phone tower to right, until it is at 4 miles right compare to the first house it meet. Do following:
1) Put a station there.
2) Delete every house this station can cover from H.
continue loop

(b) Prove the correctness of your algorithm.

Show by stay ahead:
Let O be the set of phone towers denoted by an optimal algorithm.
S is the set of phone towers we produced by our algorithm.
To prove that S is in fact optimal, we show $S_i \geq O_i$ by induction.
Base Case: $S_1 \geq O_1$, since our algorithm has reached the right range limit of the first house, if S_1 is move more right, the first house is out of range. Then you either has S_2 for it or your algorithm fails $\therefore S_1 \geq O_1$
I.H: For any $i \geq 1$, if $S_i \geq O_i$, $S_{i+1} \geq O_{i+1}$.
$\because S_i \geq O_i$, S_i covers every house O_i covered.
Now we add one more station to S, based on the algorithm we have, the new station S_{i+1} will move right until it has the right range limit of the newest uncovered house. Thus, $S_{i+1} \geq O_{i+1}$ is $S_i \geq O_i$, since S_{i+1} has matched the largest number of house O_{i+1} have.
$\therefore$ Base case holds and I.H. holds, we proved algorithm is optimal

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 S be the set of explored nodes, For each $u \in S$, we store the earliest time day we can arrive at u, and last site stay before u on the fastest path to u.

Initialized each nodes with the path direct to it = None, and time = 0

While $S \neq V$:

choose $v \notin S$ with at least one edge from a node in set S, get the varing cost to v, and the true cost to this stop is the minimum of every possible way to it.

Add v to set S, update fastest and time to it.

continue loop

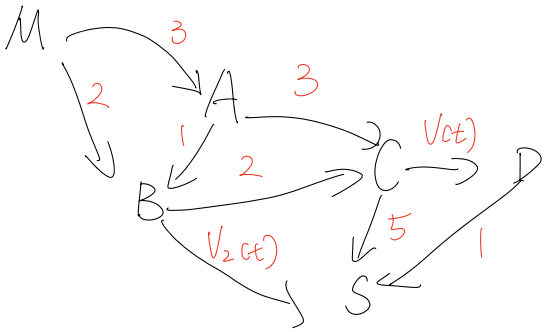
End While

Basically, this is a Dijkstra algorithm with varying time edge and should store path to a node since we need a route

(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

Here is a sample:



$V(t) = \lceil \frac{1}{2}t \rceil + 1$

$V_2(t) = 3t + 1$

Path	Total time	
M	0	
M B	2	
M B C	4	→ CD = V(4) = 2+1 = 3.
M B C D	7	
M B C D S	8	✓
M B S	9	×

= 3×2+1 = 7

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
```