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: Mahir Khan

Wisc id: mhkhan 9

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 best possible decision in the current moment / situation
that it is in.

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.

-> Job =
# start
# end
# value

For the counter example, consider the following 3 jobs: Job 1 : start = 1, end = 2, val = 2
Job 2 : start = 2, end = 4, val = 2
Job 3 : start = 1, end = 5, val = 10

In this example, the earliest finish first
algorithm would choose jobs 1 and 2, and job 3
would cause a conflict. However, this only gives a total value of 4. In
reality, the optimal choice would be job 3 and 2 because that would not
a value of 12. Thus EFF is not 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.

The algorithm's input is a list that contains each p_i and f_i time
per element of the list. This list is then sorted in ascending
order of $(p_i - f_i)$. This means that there may be negative values,
signifying $p_i < f_i$. We want this because this means jobs with
large f_i time but short p_i time can start processing on the PCs as
soon as possible. Furthermore, jobs with small f_i time but large p_i
time can take more supercomputer time since f_i time is so small.
↳ Thus, the sorted list is then looped through, with the p_i times
being input to the super computer for processing, and after that, the
PCs can start on the finishing process, which is when the
next job's preprocessing starts on the supercomputer.
↳ Also ensures shortest jobs are near the end.

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

Correctness

We can show the correctness of my algorithm through the use of the stays ahead argument.

* Assume that there exists an optimal solution S^* that has a different order of jobs than my solution, S.

* Assume $|S^*| = |S|$, and that the earliest completion time for K jobs in S and S^* are the same.

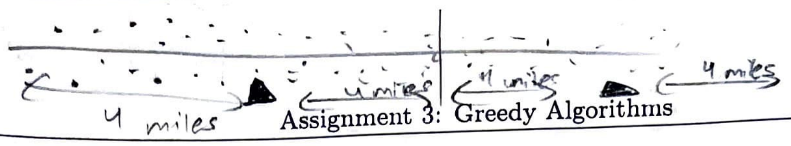
* If another $(K+1)^{th}$ job were added, then there would be two scenarios.
  - 1, S and S^* place the $(K+1)^{th}$ job in the same position, thus proving the optimality of our solution since S^* is optimal
  - 2, S^* and S place the $(K+1)^{th}$ job in different positions in the order. However, this is a contradiction, as the greedy ~~rec~~ algo S would've chosen the same position as S^*.

Thus, this shows the optimality and correctness of this algo.

Efficiency

Sorting the list of jobs based on $p_i - f_i$ would take $O(n \log n)$ time, and processing the sorted list sequentially would take linear time.

Thus overall, our algorithm runs in polynomial time or better.



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.

Input: will be a list of positions of all the houses on the road.
tower_positions ← empty list to store positions of cell towers
↳ The algorithm will start from the leftmost house, and then will continue going right until the house farthest (up to 4 miles) away from the leftmost house is reached. Then, a tower is placed there, and added to tower_positions. Another variable, current-position, should keep track of house position the algo is currently at. Once the current-position is 4 miles away from the previous tower, it chooses a new leftmost house in variable "leftmost", and restarts the process, until all houses are reached.

(b) Prove the correctness of your algorithm.

Correctness → stays ahead argument

Label S as a list of towers $\{i_1, i_2, \dots i_m\}$, and S^* as a list of towers $\{j_1, j_2, \dots, j_m\}$, both for a list of k houses.

Assume S^* is an optimal solution, for the min number of towers required for k houses.

For $k=1$ houses, only 1 tower is needed, which S has.

Assume for inductive hypothesis that $|S| \le |S^*|$.

Assume a new house is added, and that $|S| > |S^*|$.

However, this is a contradiction as in the IH, we know $|S| \le |S^*|$, and 1 more house would not cause $|S| > |S^*|$, i.e $|S| = |S| + 1$, as $|S^*|$ would have done the same.

Thus, our algorithm stays with or is ahead of S^*, thus S is correct

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.

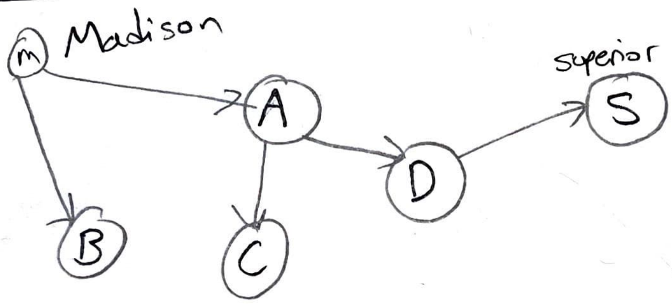
   (a) Design an algorithm your friends can use to plot the quickest route. You may assume that they start at time $t = 0$, and that the predictions made by the weather forecasting site are accurate.

Let S be the set of explored nodes, and
let G = (V, E) be a graph (G) with edges (E) between
the vertizes (V).
Create a Priority Queue (Q), and add all vertices from the
graph into Q. Initialize all vertices with a weight of infinity,
and the starting vertex Madison as 0.
The weights represent estimated arrival time.
While the priority queue is not empty, extract the vertex $\rightarrow$ ~~and~~ add to set S
with the minimum weight. Then, for all vertices connected
to the current vertex via edges, calculate their predicted
travel time from the current vertex to the neighbour
using the weather app. Update the weights of each
neighbouring vertex, and then ~~[illegible]~~ ~~[illegible]~~ add the
vertex with the minimum weight out of all the ~~set~~
neighbours into the set of explored nodes, S.
↳ continue this until ~~the~~ the lake superior node is
reached, at which point terminate the loop. This
because ~~b~~ when the lake superior node is reached,
then the destination is also reached.

(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

6 Node example



Beginning of Algo

Priority Queue (PQ) =

Node	Weight
M	0
A	∞
B	∞
C	∞
D	∞
S	∞

1)

Path	Total Time
M	0

→ make a query to the weather app for nodes A and B ↳ update PQ with new values → PQ →

A	3
B	5
C	∞
D	∞
S	∞

Choose node with min(weight) → $\bigcirc$A ←

2)

Path	Total Time
M	0
M,A	3

→ Query to weather app, for nodes C and D → update PQ →

C	2
D	3
S	∞

choose min(weight) → $\bigcirc$C ←

3)

Path	Total Time
M	0
M,A	3
M,A,C	5

→ Query for S, update PQ →

S	5

Path	
M	0
M,A	3
M,A,C	5
M,A,C,S	10
M,A,C,S	

→ Final path and total time!

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
```