CS 577 | # Assignment 2: Greedy | Summer 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: Gurusharan Kunusoth   Wisc id: Kunusoth

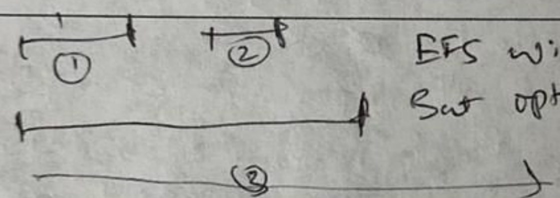
Greedy Algorithms

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.

Greedy algorithm is a ~~sort~~ short-signed algorithm trying to maximize local gain (progress) at each step. It searches for the current optimal solution in each step and ignore the future trend.

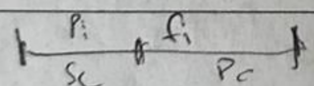
2. There are many different problems all described as "scheduling" problems. In the following questions, pay attention to the details of the problem setup, as they will change each time!

   (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.



EFS will get value 2.
But optimal should be 100.

   (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.



Use the longest finished time first algorithm, consider job

$G = \{j_1, \dots, j_n\}$

Set initial S as empty set.

while $G \neq \emptyset$ do:
  choose j_i with smallest f_i with in G
  (breaking ties arbitrarily).
  add j_i to S, remove j_i from G.
end
return S.

This requires sorting jobs based on f_i became $O(n \log n)$.

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

we define schedule A has inversion if i befor j $f_i < f_j$.
lemma: All schedules with no inversions and no idle time have the same lateness.
proof: we only focus on the job with same f_i, they must be sequential. Rearrange the order of them wont change lateness.
Theorem: There is an optimal schedule has no inversion and no idle time.
proof: we are exchange argument techique.
Consider we have an optional schedule S^*.
If S^* has inversion, we know there is at least one pair of jobs i, j with i after j. $f_i > f_j$.
we exchange i, j, we have i', j', j' after i'.
we have new schedule S'.
Define P is the time all super computer jobs finished. i.e. $P = \sum_{i=1}^{n} P_i$, b_i is the time i finished in S^*. b'_i is the time i' finished in S'. Similarly, for j, j', we know $b_i^* = \sum_{k=1}^{i} P_k + f_i$, $b'_i = \sum_{k=1}^{n+1} P_k + f_i$
Since we take j ahead, we have $b'_i < b^*_i$
we have b'_j, $\sum_{k=1}^{j'} P_k + f_j$, $= \sum_{k=1}^{i} P_k + f_j \leq \sum_{k=1}^{i} P_k + f_j < f_i$
Since $f_i > f_j$. then we have S' is as optimal as S^*.
We repeat these steps until no inversions.

3. *Kleinberg, Jon. Algorithm Design (p. 190, q. 5)*

Consider a long straight road with houses scattered along it. We want to place cell phone towers

(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.

consider the road as straight lines with some points known as ~~homes~~ houses
we start from left walk towards right with encountering first house, we set one tower 4 miles right away from this house. We start from the right boundary of range of previous tower when we encounter another house, we repeat ② we rep ① and ② until all the houses are covered.

(b) Prove the correctness of your algorithm.

we use $S=\langle i_1, i_2, \ldots, i_k\rangle$ denote the set of towers from left to right. $S^* = \langle j_1, j_2 \ldots j_m\rangle$ denote optimal solution.
Let $r_L, L \in k, m$ denote the right boundry of range of tower i_L, also the range of $\langle i_1, \ldots i_L\rangle$. Similar for r^*_i in S^*. we use always stays ~~ahead~~ ahead technique.
Lemma1: for all r_i, r^*_i we have $r_L > r^*_L$.
proof: by induction, $b=1$ it holds.
Suppose $r=n$ it holds. Since we choose $(n+1)$th tower as right as possible while covering the next home, we have $r_{i+1} \geq r^*_{i+1}$.
Theorem: Our algorithm produce optimal arrangement.
proof: By contradiction, assume $k > m$. Since by lemma1, $i_1 \ldots i_n$ can cover $j_1 \ldots j_m$ range. and $S^* : \langle j_1, \ldots j_m\rangle$ cover ~~all~~ the houses we dont need $i_{m+1} \ldots i_k$. Then it contradicts we have S is as optimal as S^*.

CS 577 **Assignment 2: Greedy** Summer 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: Gurusharan Kunusoth Wisc id: Kunusoth

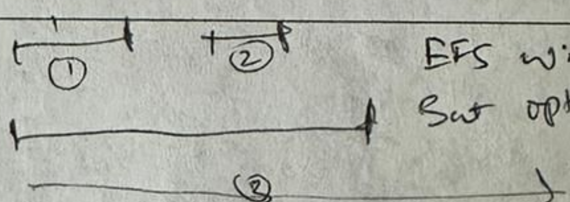
Greedy Algorithms

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.

Greedy algorithm is a ~~sort~~ short-signed algorithm trying to maximize local gain (progress) at each step. It searches for the current optional solution in each step and ignore the future trend.

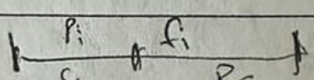
2. There are many different problems all described as "scheduling" problems. In the following questions, pay attention to the details of the problem setup, as they will change each time!

   (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.



EFS will get value 2.
But optional should be 100.

   (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.



Use the longest finished time first algorithm, consider job

$G = \{j_1, \ldots, j_n\}$

set initial S as empty set.

while $G \neq \emptyset$ do:
choose j_i with smallest f_i with in G.
(breaking ties arbitrarily).
add j_i to S, remove j_i from G.
end
return S.

This requires sorting jobs based on f_i because $O(n \log n)$.

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

we define schedule A has inversion if i befor j $f_i < f_j$.
lemma: All schedules with no inversions and no idle time have the same lateness.
proof: we only focus on the job with same f_i, they must be sequential. Rearrange the order of them wont charge lateness.
Theorem: There is an optimal schedule has no inversion and no idle time.
proof: we use exchange argument techique. Consider we have an optimal schedule S^*.
If S^* has inversion, we know there is at least one pair of jobs i, j with i after j. $f_i > f_j$.
we exchange i, j, we have i', j', j' after i'.
we have new schedule S'.
Define P is the time all super computer jobs finished. i.e. $P = \sum_{i=1}^{n} P_i$, l_i^* is the time i finished in S^*. l_i' is the time i' finished in S'. Similarly, for j, j', we know $l_i^* = \sum_{k=1}^{i} P_k + f_i$, $l_i' = \sum_{k=1}^{i+1} P_k + f_i$.
Since we take j ahead, we have $l_i' < l_i^*$
we have $l_j' = \sum_{k=1}^{j'} P_k + f_j = \sum_{k=1}^{i} P_k + f_j \leq \sum_{k=1}^{i} P_k + f_j < f_i^*$
Since $f_i > f_j$. then we have S' is as optimal as S^*.
We repeat these steps until no inversions.

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.

consider the road as straight lines with some points known as ~~...~~ houses
we start from left walk towards right with encountering first house, we set one tower 4 miles right away from this house. We start from the right boundary of range of previous tower when we encounter another house, we repeat ②. we rep ① and ② until all the houses are covered.

(b) Prove the correctness of your algorithm.

We use $S<i_1, i_2, \ldots, i_k>$ denote the set of towers from left to right. $S^* = <j_1, j_2 \ldots j_m>$ denote optimal solution.
Let ~~...~~ $r_L, L \leq k, m$ denote the right boundry of range of tower i_L, also the range of $<i_1, \ldots i_L>$. Similar for r^*_i in S^*. We use always stays ~~ahead~~ ahead technique.
Lemma1: for all r_1, r^*_i we have $r_L > r^*_2$.
proof: by induction, $b=1$ it holds.
Suppose $r=n$ it holds. Since we choose $(n+1)$th power as right as possible while covering the next house, we have $r_{i+1} \geq r^*_{i+1}$.
Theorem: Our algorithm produce optimal arrangement.
proof: By induction, assume $k > m$. Since by Lemma 1, $i_1 \ldots i_n$ can cover $j_1 \ldots j_m$ range. and $S^* = <j_1, \ldots j_m>$ cover ~~all~~ the houses we dont need $i_{m+1} \ldots i_k$. Then it contradicts
we have S is as optimal as S^*.

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.

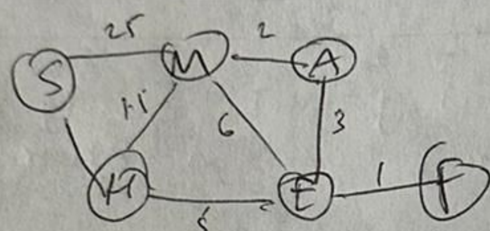
They have also found a weather forecasting site that can accurately predict how long it will take to traverse one of the edges on their graph, given the starting time t. This is important because some of the roads on their graph are affected strongly by the seasons and by extreme weather. It's guaranteed 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 we we store a distance $d(u)$. Let be ($e \in E = (u,v)$) denote the time take from u to v,
Initialize, $S = \{M\}$. $d(M) = 0$. Let V denote the set of all nodes (locations).
while $S \neq V$,
select nodes v s.t $v \notin S$. v is one edge from S. $d'(v) = \min_{e=(u,v)} [d(u) + l_e]$.
add v to S and define $d(v) = d'(v)$
if v = Sup (Superior)
break.
end
to find shortest path, we start from Sup and find the edge (u, Sup) the last step ($u \in S$) then we locate node u and find the edge (y, u) at step where u added to S ($y \in S$) we do this recursively until we reach Madison
then we have all edge together is path from Sup to M.
we reverse path, get result.

(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



start from M, explore the nodes near M, choose the nearest one, add it to linked list, at each step, we have a linked list showing the shortest path from M to the new added nodes (shown in the table). We also have one list stored the numerical value for each step, the value represents the time during the shortest path.

we terminate until we reach ~~\$~~ S, we have the linked list showing the path, and the value showing the time consumpt-ion.

5. *Kleinberg, Jon. Algorithm Design (p. 189, q. 3).*

You are consulting for a trucking company that does a large amount of business shipping packages between New York and Boston. The volume is high enough that they have to send a number of trucks each day between the two locations. Trucks have a fixed limit W on the maximum amount of weight they are allowed to carry. Boxes arrive at the New York station one by one, and each package i has a weight w_i. The trucking station is quite small, so at most one truck can be at the station at any time. Company policy requires that boxes are shipped in the order they arrive; otherwise, a customer might get upset upon seeing a box that arrived after his make it to Boston faster. At the moment, the company is using a simple greedy algorithm for packing: they pack boxes in the order they arrive, and whenever the next box does not fit, they send the truck on its way.

Prove that, for a given set of boxes with specified weights, the greedy algorithm currently in use actually minimizes the number of trucks that are needed. Hint: Use the stay ahead method.

Claim:
For the same number of trucks, greedy algorithm will ship as many boxes as other optimal algorithm.
we prove by induction,
- consider we only have 1 truck, then the greedy algorithm is the same as other methods due to the first ship policy.
- consider the conclusion holds for k trucks. consider we have $k+1$ trucks, the greedy algorithm will pack boxes in order in the $(k+1)$th truck. The item in $(k+1)$th is as many as that in optimal algorithms then we prove lemma.

Now consider we have a greedy algorithm require k trucks and an optimal algorithm requires m trucks for the same amount of boxes. Assume $k > m$, then the greedy algorithm will pack all the boxes in $1 \cdots m$ trucks. which contradicts Then $k \leq m$. we have greedy algorithm is the optimal solution.

6. *Kleinberg, Jon. Algorithm Design (p. 192, q. 8).* Suppose you are given a connected graph G with edge costs that are all distinct. Prove that G has a unique minimum spanning tree.

Lemma: Let c be any cycle in G and let e be the most expensive edge of G, then e is not in any MST of G. we use Lemma B, and we prove by contradiction. consider 2 MST: T and T'. $\exists$ $e \in T$, $e \notin T'$. Denote all the nodes as V. we consider S and $V \setminus S$ as 2 connected component after removing e from T. Since $e \notin T'$, we know there must exist $e' \in T'$ s.t e' connects S and $V \setminus S$ in T'.

Then we have a cycle containing all the edge from T and T', since the edge are all distinct. we have the most expensive edge in the cycle is not in any MST.

this ~~contains~~ contradicts with T and T' are all MST. we prove G has a unique minimum spanning tree.

7. *Kleinberg, Jon. Algorithm Design (p. 193, q. 10).* Let $G = (V, E)$ be an (undirected) graph with costs $c_e \geq 0$ on the edges $e \in E$. Assume you are given a minimum-cost spanning tree T in G. Now assume that a new edge is added to G, connecting two nodes $v, w \in V$ with cost c.

   (a) Give an efficient ($O(|E|)$) algorithm to test if T remains the minimum-cost spanning tree with the new edge added to G (but not to the tree T). Please note any assumptions you make about what data structure is used to represent the tree T and the graph G, and prove that its runtime is $O(|E|)$.

consider the new added edge e', we have v', w' ∈ V. at the ends of e'.
Initialize S = {u'} and array 1, array 2. consider MST. T
while v' ∉ S:
explore adjacent node v with DFS
add v to S. add parent of v to array 1
add cost of edge connected to v to array 2.
end.
Now we have the path from u' to v', add e' to the T. we have a cycle C containing e' on T.
check whether e' is the most expensive edge on C. If it is T remains MST. The runtime is O(|V|

   (b) Suppose T is no longer the minimum-cost spanning tree. Give a linear-time algorithm (time $O(|E|)$) to update the tree T to the new minimum-cost spanning tree. Prove that its runtime is $O(|E|)$.

From the previous algorithm, we have a cycle containing e' on T. If e' is not the most expensive edge on cycle, we removing the most expensive edge on cycle. we got T' is the MST, the runtime is O(|V|).

8. In class, we saw that an optimal greedy strategy for the paging problem was to reject the page the furthest in the future (FF). The paging problem is a classic online problem, meaning that algorithms do not have access to future requests. Consider the following online eviction strategies for the paging problem, and provide counter-examples that show that they are not optimal offline strategies.[1]

(a) FWF is a strategy that, on a page fault, if the cache is full, it evicts all the pages.

consider cache: □□

request: a b c b a

FWF: 5 faults

FF: 4 faults.

(b) LRU is a strategy that, if the cache is full, evicts the least recently used page when there is a page fault.

consider cache: □□

request: a b c ~~[illegible]~~ a b

LRU: 5 faults

FF: 4 faults.

[1] An interesting note is that both of these strategies are k-competitive, meaning that they are equivalent under the standard theoretical measure of online algorithms. However, FWF really makes no sense in practice, whereas LRU is used in practice.

Coding Questions

9. **Interval Scheduling:**
   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, Python, or Rust. 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
```

10. **Paging:**
   For this question you will implement Furthest in the future paging in either C, C++, C#, Java, Python, or Rust.

   The input will start with an positive integer, giving the number of instances that follow. For each instance, the first line will be a positive integer, giving the number of pages in the cache. The second line of the instance will be a positive integer giving the number of page requests. The third and final line of each instance will be space delimited positive integers which will be the request sequence.

   Note: a naïve solution doing repeated linear searches will timeout.

   A sample input is the following:

```
3
2
7
1 2 3 2 3 1 2
4
12
12 3 33 14 12 20 12 3 14 33 12 20
3
20
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

   The sample input has three instances. The first has a cache which holds 2 pages. It then has a request sequence of 7 pages. The second has a cache which holds 4 pages and a request sequence of 12 pages. The third has a cache which holds 3 pages and a request sequence of 15 pages.

For each instance, your program should output the number of page faults achieved by furthest in the future paging assuming the cache is initially empty at the start of processing the page request sequence. One output should be given per line. The correct output for the sample input is

```
4
6
12
```