CS 577 # Assignment 4 – More Greedy Spring 2022

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: Yongsang Park Wisc id: 908 022 3069

More Greedy Algorithms

1. *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 Q 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.

Solution:

Let the set $S := \{p_1, p_2, \cdots, p_k\}$ which follows the greedy algorithm.
Let the optimal set $O := \{q_1, q_2, \cdots q_m\}$
The element in the set shows the number of stuff in the i^{th} truck.
Lemma 1. For $\forall r \mid p_r \le q_r$ in $1 \le r \le m$
proof.
when $r=1$, $p_1 \le q_1$ is true because only one truck and same stuffs are in the truck.
suppose $p_{r-1} \le q_{r-1}$ is true.
The optimal set O is consist of compatible intervals, so $q_{r-1} \le q_r$.
And we supposed $p_{r-1} \le q_{r-1}$, so $p_{r-1} \le q_{r-1} \le q_r$. It means r^{th} truck starts to be filled after $(r-1)^{th}$ truck is full. That tells after we choose $(r-1)^{th}$ truck, we still have a choice to choose r^{th} truck. It shows $p_r \le q_r$.
Theorem. This algorithm returns the optimal solution.
The optimal set returns $k+1$ when our algorithm returns k.
As I showed in Lemma 1. if something left after q_{k+1}, that must be compatible with p_k. Our algorithm runs while there is no leftover. When the algorithm finishes even if there are some leftovers, it does not make sense. So, k is the optimal result.

2. *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.

Solution:

Let $G_1 = (V, E)$ be the original graph.

Suppose there are two distinct MSTs $T_1 = (V, E_1)$ and $T_2 = (V, E_2)$.

Since T_1 and T_2 are distinct, the set $E_1 - E_2$ and $E_2 - E_1$ are not empty.

Let $\exists e \in E_1 - E_2$. Since $e \notin E_2$, adding it to T_2 creates a cycle.

By cycle property, the most expensive edge of this cycle (e') does not belong to any MST.

But, if $e' = e$, then $e' \in E_1$

if $e' \neq e$, then $e' \in E_2$

Both cases contradict with the fact that the e' is not in ay MST.

So G_1 has a unique MST.

3. *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 $O(|E|)$.

Solution:

Make a copied tree from T. Add the new edge (e') to the copied tree (T').
Find the cycle by using BFS, and prove if the cost of e' is the most expensive edge in the cycle.

Find the way from V to W from T by using BFS
put the costs of edges into an Array.
if e' is the most expensive
| return true
else
| return false
end

costs
C_1 $(V-1)$
C_2
C_3

The time complexity is $C_1V - C_1 + C_2 + C_3$, and I can tell $V \leq E$, so its runtime is $O(|E|)$.

   (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 $O(|E|)$.

Solution:

If T is no longer the minimum-cost spanning tree, there is a cycle from V to W due to the new edge.

Q = queue { V ∈ T }
initialize Cycle graph A
while x ∈ Q
| put the adjacent node from x into Q.
| mark the previous path into the adjacent node
| if adjacent node is W
| | A = path from V to W
| | quit the loop
| end
end
delete the most expensive edge from A
Add the new edge into A
update T

costs
O(1)
O(1)
O(|V|)
O(1)
O(1)
O(1)
O(|V|)
O(1)
O(1)

Runtime = $O(1)+O(1)+O(|V|)(O(1)+O(1)+O(1))+O(|V|)+O(1)+O(1)$
$= O(|V|)$
$V \leq E+1$, so $O(|V|) = O(|E|+1) = O(|E|)$.

4. 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.

Solution:

Counter example

Cache : 2 pages

Request sequence: ABCAB

sequence	FWF	Fault?	FF	Fault?
A	A	Y	A	Y
B	A B	Y	A B	Y
C	C	Y	A C	Y
A	C A	Y	A C	N
B	B	Y	B C	Y

Faults FWF: 5
FF: 4

So FWF is not the optimal offline strategy.

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

Solution:

Counter example

Cache : 2 pages

Request sequence: ABCAB

sequence	LRU	Fault?	FF	Fault?
A	A	Y	A	Y
B	A B	Y	A B	Y
C	C B	Y	A C	Y
A	C A	Y	A C	N
B	B A	Y	B C	Y

Faults LRU : 5
FF: 4

So LRU is not the optimal offline strategy.

[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.

5. Coding problem

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

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.

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
15
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
9
```