CS 577 (Epic) # Assignment 4 – More Greedy Spring 2020

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: Zhuoyan Xu Wisc id: zxu444

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 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,

① consider we only have 1 truck. then the greedy algorithm is the same as other methods due to the first ship policy.

② 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 algorithm. then we prove lemma □.

Now Consider we have an 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.

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.

lemma 13: 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 13. 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\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\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 this cycle is not in any MST.
- This contradicts with T and T' are all MST.

we prove G has a unique minimum spanning tree.

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 $O(|E|)$.

O(|V|)

Bidirectional BFS. ?

Consider the new added edge e', we have $v', w' \in V$ at the ends of e'.

Initialize $S = \{w'\}$ and . array 1, array 2. Consider MST. T

While $v' \notin S$:

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.

then we have the path from w' to v', add e' to the T.

we have a cycle C containg e' on T. check whether e' is the most expensive edge on C. If it is, T remains a 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 algorith, 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 run time is $O(|V|)$.

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.

Consider cache: □□ U = {a, b, c, d}

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: □□ U = {a, b, c, d}

request: a b c 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.

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 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
```