CS 577

Assignment 4 – More Greedy

Fall 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: ______________________ Wisc id: ______________________

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.

Solution: By induction on the number of trucks, we will show that the described greedy strategy, FF, is always ahead of any other strategy. That is, FF has shipped at least as many boxes.

Base case: 1 truck. With one truck, only what fits can be packed.

Induction step: Assume the claim to be true for k trucks. Consider the case of $k+1$ trucks. By the induction hypothesis, we know that FF has shipped at least as many boxes as any strategy S in the first k trucks. For S to overtake FF with the $(k+1)$-th truck, S would have to pack, at the very least, all the items packed by FF in its $(k+1)$-th truck plus the next item j. Since FF did not pack j in the $(k+1)$-th truck, all those items cannot fit on a single truck.

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: Assume that G has at least 2 MSTs, T_1 and T_2, that differ. Let $e \in T_1 \setminus T_2 \cup T_2 \setminus T_1$ be the minimum cost edge not shared by T_1 and T_2. Without loss of generality, assume that e comes from T_1. We can create another graph $T_2 \cup e$ which now has a cycle C containing e. Let f be the most expensive edge in C.

If $f = e$, then T_1 is not MST as per Lemma 13 in lecture slides which is a contradiction.

If $f \neq e$, then, let $T_3 := T_2 \cup e \setminus f$. T_3 is a tree and, moreover, T_3 has must have a lower cost than T_2 since $c_e < c_f$ and they cannot be equal under the assumption of distinct edge weights. This is a contradiction as the overall cost of T_3 is strictly less than T_2.

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:

Data Structures: G will be represented by an adjacency list. For each adjacent node, the cost of the edge and a bit is associated. The associate bit being 1 means that edge is in the MST.

Algorithm: Beginning from v, do a DFS on T until w is found, storing the simple path from v to w path. Consider the cycle C formed by adding the new edge. If the cost of the new edge is not the maximum cost edge of C, then T is no longer an MST.

Run Time:

- DFS: $O(|V| + |E|) = O(|E|)$.
- Checking the max cost of the v to w path: $O(|E|)$.

   (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:

Algorithm: Beginning from v, do a DFS on T until w is found, storing the simple path from v to w path. Consider the cycle C formed by adding the new edge. Let f be the most expensive edge. Update the MST bit for f to 0, and update the MST bit for the new edge to 1.

Run Time:

- DFS: $O(|V| + |E|) = O(|E|)$.
- Finding the max cost edge of the v to w path: $O(|E|)$.

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: Request sequence: $\sigma = \langle a, b, c, a \rangle$
   Cache size: $k = 2$

   FWF: After the first two requests, the cache is full and FWF will evict the entire cache causing 2 more page faults. Overall 4 page faults.

   FF: After the first two requests, the cache is full and FF will evict b to bring in c with no page fault on the last request. Overall 3 page faults.

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

   Solution: Request sequence: $\sigma = \langle a, b, c, a \rangle$
   Cache size: $k = 2$

   LRU: After the first two requests, the cache is full and LRU will evict a to bring in c. Then, evict b to bring in a. Overall 4 page faults.

   FF: After the first two requests, the cache is full and FF will evict b to bring in c with no page fault on the last request. Overall 3 page 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 Problem

5. 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.

   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 20 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
```