CS 577

Assignment 4 – More Greedy

Spring 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: Matej Popovski

Wisc id: popovski@wisc.edu | 9083541632

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:
Denote by $a_1, a_2 \ldots, .a_n$, ordered boxes that arrived to the station, where each box a_i has $w_i \geq 0$ weight. They are packed into the $x_1, \ldots, x_N$ trucks each having limited capacity W. It is asked that boxes are packed into the tracks in the order they have arrived, no track to be overloaded, and number of used tracks to be minimal. The order of arrived boxes is preserved if the box a_i is sent before box a_j, where a_i is packed to the track x_p and box a_j is packed to the track x_q where $p < q$, and arrivals of the boxes to the company is in the order $a_i < a_j$, $(i < j)$. We prove that the greedy algorithm uses the fewest possible trucks by showing that it "stays ahead" of any other solution. Greedy algorithm fits boxes $a_1, a_2 \ldots, a_j$ into the first k track, and other solution fits $a_1, a_2 \ldots, a_i$ into the first k tracks, then $i \leq j$. This implies the optimality of the greedy algorithm, by setting k to be the number of tracks used by the greedy algorithm. By induction the case $k = 1$ is clear, the greedy algorithm fits as many boxes in the first truck. Assuming this holds for $k-1$ trucks. The greedy algorithm fits $j^{'}$ boxes into the first $k-1$, and the other algorithm fits $i^{'} < j^{'}$. For the k-th truck, the other algorithm fits $a_{i^{'}+1}, \ldots, a_i$. Since $j^{'} > i^{'}$ and greedy algorithm can fit $a_{j^{'}+1}, \ldots, a_i$, and potentially can fit more. This proves the claim of the optimality of the greedy algorithm.

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:
Prove by contradiction. Suppose that T and $T^{'}$ are two distinct minimum spanning trees of the graph G. Being minimal, both trees, T and $T^{'}$ must have the same number of edges. Being not equal, there is at least one edge $e^{'}$ in the tree $T^{'}$ that is not in T. We add this edge $e^{'}$ to T, forming the cycle C in the tree T (edge $e^{'}$ being element of C). Let e be the most expensive edge in this cycle. By the Cycle property e does not belong to any minimum spanning tree, contradicting the fact that it is in at least one T or $T^{'}$.

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:
We assume that all edge costs are distinct. (If not, as suggested in chapter 4.5 of the book, we can perturb all the costs by very small amounts and conduct proofs under these assumptions). Let we add the edge e = (v,w) with the cost c to the graph G. The minimum spanning tree T is represented using an adjacency list, and determine the v-w path P in T (by the linear time algorithm in the number of nodes and edges). If every node on this path has cost less than c, (at most O(|E|) checks), then the Cycle Property implies that the new edge e=(v,w) is not in the minimum spanning tree, since it is the most expensive edge on the cycle C formed from the edge e and the path P. In this case the minimum spanning tree do not change. On the other hand, if some edge on this path has cost greater than c, then the Cycle Property implies that the most expensive such edge f cannot be in the minimum spanning tree, so T is no longer the minimum spanning tree.

   (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:
By replacing the most expensive edge f on the path v-w and P by new edge e, we obtain new spanning tree T'. We claim that T' is a minimum spanning tree. To prove this, we consider any edge e' not in T', and show that by applying the Cycle Property we conclude that e' is not in any minimum spanning tree. Let e' = (v', w'). Adding e' to T' establish a cycle C' consisting of the v'-w' path P' in T' including e'. If we can show that e' is the most expensive edge in cycle C', we conclude that it is not in any spanning tree. Consider one further cycle K formed by adding e' to T. By the Cycle Property, e' is the most expensive edge on K. Now there are three cycles in graph: C, C', and K. Edge f is the most expensive edge on C, so the edge e' is the most expensive edge on K. If the added edge e does not belong to C', then C'=K and so e' is the most expensive edge on C'. Otherwise, the cycle K includes f (since C' needed to use e instead), and C' uses a portion of C (including e) and a portion of K. In this case, e' is more expensive than f (since f lies on K), and hence it is more expensive than every edge in C (since f is the most expensive edge on C). It is also more expensive than every other edge in K, and so it is the most expensive edge on C', as we want to show.

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:
Contra example
Let Cash be 2 pages and requested sequence: ABCAB

Sequence	SFW	fault	FF	fault
A	A_	Y	A_	Y
B	AB	Y	AB	Y
C	C_	Y	AC	Y
A	CA	Y	AC	N
B	A_	Y	BC	Y

For the FWF strategy we have 5 faults while for FF strategy there are 4 faults. So FWF is not an optimal 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:
Let Cash be 2 pages and requested sequence: ABCAB

Sequence	LRU	fault	FF	fault
A	A_	Y	A_	Y
B	AB	Y	AB	Y
C	CB	Y	AC	Y
A	CA	Y	AC	N
B	BA	Y	BC	Y

For the LRU strategy we have 5 faults while for FF strategy there are 4 faults. So LRU is not an optimal 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
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
```