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: Ethan Yan Wisc id: 9084649137

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.

First note that boxes need to be shipped in the order that they arrive or a customer may be upset. (1)

Let S be the set of packages sorted from earliest to latest arrival time.

We will use induction to show that the company's greedy algorithm g is optimal.

Base case: $|S|=1$, there is only 1 package, p, if $w_p \leq W$ then it can be shipped on truck, else if $w_p > W$, will have to wait for next truck, no other packages can be shipped considering (1).

IH: Assume true for $k-1$ packages, ie both greedy algo & optimal soln uses same # of trucks.

greedy algo will prioritize next k^{th} package, whereby if w_k causes sum of current packages in truck $+ w_k > W$, k^{th} package will be on the next truck, else in will be on ~~latest~~ current truck.

This is on par will optimal ~~at~~ algo as optimal algo would not allow any package $>k$ to be ahead of k considering (1) and that packages arrive one by one at New York.

This thus shows greedy algo is at least on par with optimal algo and thus stays ahead, minimizing number of trucks needed.

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.

Given a graph G with ~~not~~ distinct edge costs, we can use Kruskal's algorithm to derive a unique minimum spanning tree.

Kruskal's algorithm work in such a way:

1. Sort edges by cost from lowest to highest
2. Insert edges in order unless insertion causes a cycle.

~~we use exchange argument to show optimality of greedy algo in producing a unique minimum spanning~~

we will use a proof by ~~contradiction~~ induction.

~~we know Kruskal's produces a MST G'. Consider 2 vertices in G' u, v.~~

Base case: $|V|=1, |E|=0$, trivial, unique

$|V|=2, |E|=1$, only 1 edge to pick, definitely unique.

IH: $|V|=k$, there is a minimum spanning that is unique.

assume it holds that there is unique MST for $|V|=k$.

Then for $|V|=k+1$, we simply pick lowest cost edge that connects MST to $(k+1)^{st}$ vertex. It will still be unique as ~~no~~ edge costs are all distinct

Proof by contradiction graph $|V|=k+1$ ~~is~~ has more than 1 MST.

This is saying there is more than one edge to $(k+1)^{st}$ vertex of the same cost, as by IH, we know that there is a unique MST. Therefore, for any graph G with $|V|=k+1$, since removing $(k+1)^{st}$ vertex gives $|V|=k$ which by IH has a unique MST, we know that all edges to the $(k+1)^{st}$ has to be distinct and selecting lowest cost edge produces unique MST for graphs G of $|V|=k+1$.

Therefore we proved by induction for all connected graph G with distinct edge costs, there is 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 $O(|E|)$.

Let e=(v,w) be the new edge being added. We represent T with an adjacency list & we find the v-w path P in time linear in the number of vertices and edges of T, which is O(|V|). If every edge on this path is less than c, then edge e is not in the MST, since it is the most expensive edge in the cycle containing v-w.

If there is some edge e' on the path with cost > c, then e is a cheaper edge and e' would be cut instead so T is no longer the MST.

Given |V|<|E| then O(|V|) & 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 $O(|E|)$.

Replace the heaviest edge on the v-w path P in T with new edge e, obtaining new spanning tree T'. To prove T' is a MST, we consider any edge e' not in T' and show we can cut that edge of our cycle to conclude that e' is not in any MST. Adding e' to T' gives cycle C' containing v'-w' path p' in T' plus e'. We just need to show e' is most expensive on C'.
Consider another cycle K by adding e' to original T. We know e' is most expensive edge on K. Edge f is the most expensive edge on K, so if new edge e is not in C' then C'=K and e' is most expensive edge in C'. Otherwise cycle K includes f and C' uses portion of C and portion of K. So, e' is more expensive than f as f lies on K, and hence is more expensive than everything on C. It is also more expensive than everything else on K, so it is the most expensive edge on C' as desired. As we are looping through all edges, the runtime is at worst 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.

Consider page requests in order: 1, 2, 3, 2, with cache size of 2.

	1	2	3	2
FF:	x	x	x	✓
FWF:	x	x	x	x

FF would have hit rate of 0.25, while FWF would have hit rate of 0.

When 3 is been processed, FF evicts 1 in cache to have [2,3] whereas, FWF would clear the cache to have [3].

Leading to a hit for FF on the 2nd '2' and fault for FWF.

This shows how FF outperforms FWF as it has a higher hit rate as shown and FWF is not optimal.

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

Consider page requests in order: 1, 2, 3, 1, with cache size of 2.

		fault/hit	cache
1	FF	x	[1]
	LRU	x	[1]
2	FF	x	[1,2]
	LRU	x	[1,2]
3	FF	x	[1,3]
	LRU	x	[2,3]
1	FF	✓	[1,3]
	LRU	x	[3,1].

FF: 0.25, LRU: 0.

LRU is essentially a PQ. It would perform badly on n-cycle of numbers with cache size < n as shown on the left. Given that we know future sequence and with uncontrollable cache size, LRU would not be optimal, especially in cases where we know it is a cycle.

[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.