

# CS 344: Design and Analysis of Computer Algorithms

Instructor: Kangning Wang

## Homework 4: Due on **April 7, 2025**

**Acknowledgment:** Our team members are Alexander Lin (al1655), Pranav Tikkawar (pt422), and Ivan Zheng (iz72) .

### Problem 1

We covered the offline caching problem in class and stated that the farthest-in-future algorithm is optimal. Formally prove its correctness using the exchange argument. You can refer to our textbook [KT, Section 4.3] to see how the proof is done.

**Solution.** Let  $S$  be an optimal strategy for the given request sequence. Let FiF be the strategy followed by the Farthest-in-Future algorithm. Assume that  $S$  and FiF behave identically up to a certain point where they differ in their eviction choices. We'll show that we can modify  $S$  to align with FiF at that point without increasing the number of misses.

- Consider the first time  $S$  and FiF differ in their eviction decisions. Let this be at request  $r_i$  where item  $x$  is requested, and it's not in the cache.
- At this point, both  $S$  and FiF must evict an item to bring in  $x$  (since the cache is full and  $x$  is not present).
- Suppose FiF evicts item  $a$  (the one farthest in the future or never used again) and  $S$  evicts item  $b$  ( $b \neq a$ ).
- Since FiF chose  $a$ , this means  $a$  is used farther in the future than  $b$ , or  $a$  is not used again while  $b$  is used sooner.
- Define  $S'$  to be the same as  $S$  up to request  $r_i$ .
- At  $r_i$ ,  $S'$  evicts  $a$  (like FiF) instead of  $b$ .
- After  $r_i$ ,  $S'$  behaves optimally but ensures that it doesn't incur more misses than  $S$ .
- Case 1:  $a$  is never requested again. Evicting  $a$  is optimal since keeping  $b$  might lead to an earlier eviction of  $b$  later.  $S'$  doesn't incur extra misses compared to  $S$ .
- Case 2:  $a$  is requested later than  $b$ . By evicting  $a$  instead of  $b$ ,  $b$  will be in the cache when it's next needed before  $a$ 's next request. Thus,  $S'$  might avoid a miss on  $b$  that  $S$  would have had, or at least not do worse.

- In both cases,  $S'$  does not have more misses than  $S$ .
- Repeat this process for all points where  $S$  and FiF differ. Each exchange either keeps the number of misses the same or reduces them. Eventually,  $S$  is transformed into FiF without increasing the number of misses.
- Since any optimal strategy  $S$  can be transformed into FiF without increasing the number of misses, FiF must itself be optimal.

## Problem 2

We have  $n$  boxes. The box  $i$  has weight  $w_i$  and weight limit  $\ell_i$ . A box will break if the sum of weight above it exceeds its weight limit. Each box has the same size of  $1 \times 1 \times 1$ .

What is the tallest tower we can build by stacking a subset of boxes on top of each other? We can freely choose the order. Give a greedy algorithm and formally prove its correctness using the exchange argument.

**Solution.** Here is my solution.

## Problem 3

Given a weighted undirected graph  $G = (V, E)$ , decide whether its minimum spanning tree is unique. Give an algorithm that runs in  $O(m \log m)$  time and prove its correctness. [Hint: You can modify Kruskal's algorithm.]

**Solution.** Here is my solution.

## Problem 4

In this problem, we will design an algorithm to compute the minimum spanning tree in a given graph in  $O(m \log \log n)$  time, where  $m$  is the number of edges and  $n$  is the number of vertices in the given graph. For example, if  $m = \Theta(n)$ , then the running time of this algorithm ( $\Theta(n \log \log n)$ ) is better than that ( $\Theta(n \log n)$ ) of Prim's algorithm, Kruskal's algorithm, and Borůvka's algorithm. Here are the ideas.

- In the original Borůvka's algorithm, we might need  $\Theta(\log n)$  rounds to merge all vertices into one group, and each round takes  $O(m)$  time. The twist is that we now only run Borůvka's algorithm for  $r$  rounds.
- After running Borůvka's algorithm for  $r$  rounds, we obtain a new graph with at most  $\frac{n}{2^r}$  vertices and at most  $m$  edges. We run Prim's algorithm (with a Fibonacci heap) on it to find its MST.

How should we choose the parameter  $r$  so that the total running time becomes  $O(m \log \log n)$ ?

**Solution.** Here is my solution.

## Problem 5

We have an undirected graph  $G = (V, E)$ . Each edge has a positive length, and there is at most one edge between each pair of vertices. Find the length of the shortest simple cycle in  $G$ . Your algorithm should run in  $O(n^3)$  time, where  $n = |V|$ . [Hint: You can modify the Floyd–Warshall algorithm. Recall that  $f(k, u, v)$  is the length of the shortest path from  $u$  to  $v$  using vertices with indices of at most  $k$ . Consider a cycle where  $k$  is the largest index among its vertices and  $u$  and  $v$  are the two neighbors of  $k$  in the cycle.]

**Solution.** Here is my solution.