1 Average case analysis for the Stable Matching algorithm

In the first lecture, I presented the Stable Matching algorithm, where I showed that there are \( O(n^2) \) rounds in the worst case. In this problem, we will show that the average case is much better: the expected number of proposals is much smaller.

We assume the men’s choices are picked uniformly at random while the woman’s choices can be arbitrary but are fixed before the algorithm starts. Since the men’s choices are random, we can view the algorithm as this: when a man proposes, he will propose to a woman chosen uniformly at random. Note that the original Stable Matching algorithm does not allow a man to propose to the same woman twice. Here, for the sake of analysis, we do allow a man to propose to the same woman multiple times (note that this does not change the outcome of the algorithm: the duplicate proposals will just be wasted; but note that this changed algorithm only needs no fewer proposals than then the original algorithm). That is, at each iteration of the Stable Matching algorithm, a man will propose to one of the \( n \) women uniformly at random.

Now, here are your tasks.

1. First, let us consider yet another balls and bins question. We still consider the settings of the Problem 5 of HW3. Imagine that we have infinite supply of balls and we keep throwing the balls. Your task is computing how many balls we expect to throw for \( n \) bins until each bin receives at least one ball.
2. Argue that the Stable Matching algorithm will terminate once each woman receives at least one proposal.
3. Then compute the expected number of proposals made during the execution of the algorithm.

2 Heap

Do parts (a), (b) and (c) of Problem 6.2 (p.167) in textbook. For part (a), tell me where the \( d \) children (if exists) of a node \( i \) will be located in the array, and where the parent of a node \( i \) will be in the array.

3 Lower bound

Consider the task of searching a sorted array \( A[1 \ldots n] \) for a given element \( x \): a task we usually perform by binary search in time \( O(\log n) \). Show that any algorithm that accesses the array only via comparisons (that is, by asking questions of the form “is \( A[i] \leq z \)?”), must take \( \Omega(\log n) \) steps.

4 Coin Change

We now prove the simple greedy algorithm for the coin change problem with quarters, dimes, nickels and pennies are optimal (i.e. the number of coins in the given change is minimized), when the supply of each coin type is unlimited.

Let \( q_o, d_o, k_o, p_o \) be the number of quarters, dimes, nickels and pennies used for changing \( n \) cents in an optimal solution.
1. First, show that \( d_o \leq 2, k_o \leq 1, p_o \leq 4 \). Also, show that if \( k_o = 1 \), then \( d_o \leq 1 \).

2. Now show it is always optimal to choose as many quarters as possible.

3. Finally, show that the greedy algorithm is optimal in its choice for dimes, nickels and pennies.

5 Activity Selection

Let us consider an alternative greedy strategy for the activity selection problem. Let \( \mathcal{R} = \{R_1, \ldots, R_n\} \) be the set of activities. Recall that each activity \( R_i = [s_i, f_i] \).

1: while \( \mathcal{R} \) is not empty do
2: Select the activity \( R_i \in \mathcal{R} \) that overlaps with the fewest activities in \( \mathcal{R} \) and add this activity to solution set \( A \).
3: Remove \( R_i \) and any activity \( R_j \) that overlaps with \( R_i \) from \( \mathcal{R} \).
4: end while

Now prove or disprove that the greedy strategy solves the activity selection problem optimally.