CS341 ASSIGNMENT 3 – SOLUTIONS

(1) (a) Suppose that (P, N, D, Q, L, T) is an optimal solution where P = number of pennies, N = number of nickels, D = number of dimes, Q = number of quarters, L = number of loonies, and T = number of toonies. The greedy solution is characterized by the value of each coin being greater than the total value in the solution contributed by lower value coins. To verify that the optimal solution coincides with the greedy solution, we will check this criteria.

Notice that $P \leq 4$ since otherwise we could replace 5 pennies with a nickel. $5 > 4 * 1$, thus the value of a nickel is greater than the total value contributed by lower value coins.

$N \leq 1$ since otherwise we could replace 2 nickels with a dime. $10 > 1*5+4*1$, thus the value of a dime is greater than the total value contributed by lower value coins.

Since $N \leq 1$, we have $N + D \leq 2$ because if we had a nickel and 2 dimes then we could replace these coins with a quarter. $25 > 2 * 10 + 4 * 1$, thus the value of a quarter is greater than the total value contributed by lower value coins.

$Q \leq 3$ since otherwise we could replace 4 quarters with a loonie. $100 > 3 * 25 + 2 * 10 + 4 * 1$, thus the value of a loonie is greater than the total value contributed by lower value coins.

$L \leq 1$ since otherwise we could replace 2 loonies with a toonie. $200 > 1*100 + 3 * 25 + 2 * 10 + 4 * 1$, thus the value of a toonie is greater than the total value contributed by lower value coins.

Thus the optimal solution coincides with the greedy solution. So the greedy solution is optimal.

(b) Consider the coin system $(13, 6, 1)$. To make 18 the greedy solution is $18 = 13 + 1 + 1 + 1 + 1 + 1$ but the optimal solution is $18 = 6 + 6 + 6$.

(c) Let $(n_0, n_1, \ldots, n_k)$ be an optimal solution where n_i is the number of coins with value c^i. As in part a, we will show that the value of each coin is greater than the total value contributed by lower value coins.

$n_0 \leq c - 1$ since otherwise we could replace c c^0 coins with a single c^1 coin. $c^1 > (c - 1) * c^0$, thus the value of a c^1 coin is greater than the total value contributed by lower value coins.

Now suppose that $c_i > \sum_{j=0}^{i-1} n_j c_j$. We know that $n_i \leq c-1$ since otherwise we could replace c c^i coins with a single c^{i+1} coin. Thus

$$\begin{aligned} & \sum_{j=0}^{i} n_j c^j \\ = \; & n_i c^i + \sum_{j=0}^{i-1} n_j c^j \\ < \; & (c-1)c^i + c^i \\ = \; & c^{i+1} \end{aligned}$$

Thus c^{i+1} is greater than the total value contributed by lower value coins. Applying this step from $0 \leq i < k$ we see that the optimal solution coincides with the greedy solution and thus the greedy solution is optimal.

(2) (a) Suppose that we have 4 matrices of dimensions $(3 \times 2)(2 \times 1)(1 \times 2)(2 \times 3)$. Doing the minimum cost multiplication first would lead to:

$$\begin{aligned} & & (3 \times 2)(2 \times 1)(1 \times 2)(2 \times 3) \\ (\text{cost: } 2*1*2 = 4) \mapsto \; & & (3 \times 2)(2 \times 2)(2 \times 3) \\ (\text{cost: } 3*2*2 = 12) \mapsto \; & & (3 \times 2)(2 \times 3) \\ (\text{cost: } 3*2*3 = 18) \mapsto \; & & (3 \times 3) \\ \text{total cost: } 34 & & \end{aligned}$$

However, a better order is:

$$\begin{aligned} & & (3 \times 2)(2 \times 1)(1 \times 2)(2 \times 3) \\ (\text{cost: } 3*2*1 = 6) \mapsto \; & & (3 \times 1)(1 \times 2)(2 \times 3) \\ (\text{cost: } 1*2*3 = 6) \mapsto \; & & (3 \times 1)(1 \times 3) \\ (\text{cost: } 3*1*3 = 9) \mapsto \; & & (3 \times 3) \\ \text{total cost: } 21 & & \end{aligned}$$

(b) Suppose that we have 3 matrices of dimensions $(3 \times 1)(1 \times 3)(3 \times 2)$. Doing the maximum cost multiplication first would lead to:

$$\begin{aligned} & & (3 \times 1)(1 \times 3)(3 \times 2) \\ (\text{cost: } 3*1*3 = 9) \mapsto \; & & (3 \times 3)(3 \times 2) \\ (\text{cost: } 3*3*2 = 18) \mapsto \; & & (3 \times 2) \\ \text{total cost: } 27 & & \end{aligned}$$

However, a better order is:

$$\begin{aligned} & (3 \times 1)(1 \times 3)(3 \times 2) \\ (\text{cost: } 1 * 3 * 2 = 6) \mapsto \quad & (3 \times 1)(1 \times 2) \\ (\text{cost: } 3 * 1 * 2 = 6) \mapsto \quad & (3 \times 2) \\ \text{total cost: } 12 & \end{aligned}$$

(c) Suppose that we have 3 matrices of dimensions $(2 \times 3)(3 \times 2)(2 \times 1)$. Doing the multiplication with maximum k_{i+1} first would lead to:

$$\begin{aligned} & (2 \times 3)(3 \times 2)(2 \times 1) \\ (\text{cost: } 2 * 3 * 2 = 12) \mapsto \quad & (2 \times 2)(2 \times 1) \\ (\text{cost: } 2 * 2 * 1 = 4) \mapsto \quad & (2 \times 1) \\ \text{total cost: } 16 & \end{aligned}$$

However, a better order is:

$$\begin{aligned} & (2 \times 3)(3 \times 2)(2 \times 1) \\ (\text{cost: } 3 * 2 * 1 = 6) \mapsto \quad & (2 \times 3)(3 \times 1) \\ (\text{cost: } 2 * 3 * 1 = 6) \mapsto \quad & (2 \times 1) \\ \text{total cost: } 12 & \end{aligned}$$

(3) To find the largest subset S of edges in T such that no two edges in S are incident on the same vertex, we apply the following algorithm. Starting at the leaf level, greedily add each edge to S if it is not incident to the same vertex as any edge already in S. Move up to the next level of the tree and repeat this until the root level is reached.

To see that this algorithm finds a largest subset S, we will proceed by induction on the height of the tree.

Base case: The tree is of height 1.

This tree consists of only 1 vertex and no edges. So the algorithm will trivially construct $S = \emptyset$ which is the largest possible.

Inductive case: Suppose the algorithm finds the largest possible S for trees of height $\leq k$.

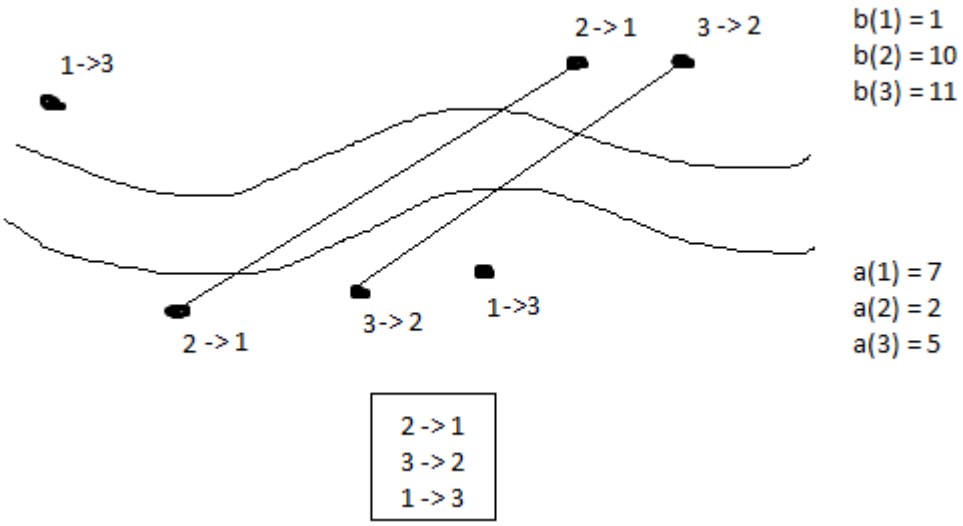
Let T be a tree of height $k + 1$. The algorithm greedily takes as many edges as it can in the leaf level into S. Since the algorithm will never pick any edges that share vertices with these edges, it is safe to now delete the subtrees rooted at vertices above the chosen edges. Call the tree resulting from these deletions T'. T' is a tree of height at most k so by the inductive hypothesis, running the algorithm on T'

gives a largest possible S'. Each of the deleted subtrees can only contribute 1 edge each (because all edges in the subtree share a common vertex) so by adding S' to the S we have already collected, we get a maximal subset of edges such that no edges in S share a vertex.

By induction, the algorithm produces a maximal S for all heights of T.

(4) We begin by permuting the labels on the cities such that cities on the south bank are in ascending order. This can be accomplished by sorting based on their x-coordinates in $O(n \log n)$ time with a sort such as mergesort. The labels in the solution we find will be the same permutation of the labels in the solution to the original problem.

Here is an example showing the permutation that puts the cities on the southern bank in ascending order.



Notice that two bridges cross exactly when there is a bridge connecting cities i and a second bridge connecting cities j such that $i < j$ (i.e. i is to the left of j on the south bank) but i is to the right of j on the north bank. In other words, a sequence of cities on the north bank corresponds to a set of non-crossing bridges iff it is non-decreasing. Thus the problem reduces to finding the longest increasing subsequence of the cities on the north bank. We will use dynamic programming to solve the longest increasing subsequence problem.

Let A_i be the label of the i^{th} city on the north bank in order of x-coordinate. Let L_i be a longest increasing subsequence of $\{A_j\}_{j=0}^{i}$. Note that $L_0 = \{A_0\}$.

L_i can be calculated in the following way.

```
L_i = { A_i }
for j = 0..(i-1)
    if A_i > last( L_j ) then
        L_i = max( L_i, L_j + {A_i} )
    else
        L_i = max( L_i, L_j )
    end if
end for
```

Since L_i depends only on previous values (L_j such that $j < i$), we can calculate L_i for each $i < n$ in succession. A longest increasing subsequence of the cities on the north bank is L_{n-1}. By applying the inverse of the permuation to this subsequence, we get the labels of the cities for which we should build bridges.

(5) Let $V_{i,j}$ be the amount of money you can definitely win if presented with coins $v(i), \ldots, v(j)$. Note that $V_{i,i} = v(i)$ and let us say that if $i > j$ then $V_{i,j} = 0$ by convention. To calculate $V_{i,j}$, we see that we have two choices: to take coin $v(i)$ or $v(j)$. Our opponent is then left with two choices ($v(i+1), v(j)$ or $v(i), v(j-1)$ depending on our choice). Then it is our turn again and we are left with a subproblem with two fewer coins. We cannot make assumptions about our opponent's behavior, so we can only guarantee that we get the minimum value resulting from each of our opponent's choices. We can, however, make the choice on our turn that maximizes our value.

$$V_{i,j} = \max \left\{ \begin{array}{l} v(i) + \min(V_{i+2,j}, V_{i+1,j-1}) \\ v(j) + \min(V_{i+1,j-1}, V_{i,j-2}) \end{array} \right\}$$

We initialize by setting $V_{i,i} = v(i)$. Since each $V_{i,j}$ only depends on values $V_{i',j'}$ where $j' - i' < j - i$ we can now calculate $V_{i,j}$ where $j - i = 1$. We can then calculate $V_{i,j}$ where $j - i = 2$ and so on until we have calculated $V_{1,n}$ which is our final answer.