CMPT 409 Assignment 3

Heather Li, Ekjot Singh Billing, Manshant Singh Kohli

February 23, 2018

1 Unidirectional TSP

(C++11) For this question, we used a DP approach where we used two arrays *dp* and *grid* to store information about the minimal path starting from the left side of the matrix and terminating at a point (i, j).

For each index (i, j), $dp[i][j]$ stored the location of the penultimate step of the minimal path ending at location (i, j) in the array. If there were multiple minimum paths ending at the same point, $dp[i][j]$ would store the smallest index.

Our array *grid* stored the weight of the minimum path ending on (i, j). In $O(n^2)$ time, we iterated through the entire array (checking every row in every column from left to right).

$$grid[i][j] = \min(dp[i-1][j-1], dp[i][j-1], dp[i+1][i-1]) + cost[i][j]$$

(where $cost[i][j]$ is the value stored in the matrix at entry (i, j) and the row indices are taken modulo m).

Scanning our *grid* array gives us the weight of the minimum cost path and the *dp* array gives us the indices along its path.

2 Chopsticks

(C++ 11) For this question, we noted that to minimize badness in a set of three chopsticks A,B,C the chopsticks A and B must be consecutive elements in the sequence of lengths given. We stored an array *badness* of the badness of consecutive elements.

We created a dynamic programming array *dp*.

$$dp[i][j] = min(dp[i][j-1], dp[i-1][j-2] + badness[n-(j-2)]$$

$dp[i][j]$ stores the minimum badness that must be created if there must be i sets of chopsticks, using the last j lengths given to us in our input length. It considers $dp[i][j-1]$, the value if the $j-1$th last chopsticks were not included in a set and $dp[i-1][j-2] + badness[n-(j-2)]$, the badness if the $j-1$ and $j-2$ chopsticks were to be included in a set.

Using such a DP function, we could recurse through the values in the array. If $3i < j$, then there wouldn't be enough space to fully create all the chopstick sets. If $i = 0$, then no chopsticks sets needed to be created. Calling $dp[k+8][n]$ once our array is fully populated would return our solution.

3 Chopsticks

(C++ 11) For this question, we noted that if assigned directed edges between two countries which had a domination relationship, we could create a forest. Furthermore, by assigning a dummy root node, we could create a tree. We used a map to map the country names to indices and stored the domination/adjacency relationships in an adjacency list.

We established a dynamic programming array dp, where $dp[i][j]$ stored the minimum number of diamonds to bribe at least j of country i and its subjugates.

We used a recursive depth-first search to help us load our dp array. Our recursive function, when called on a country "$root$", returned the number of countries under its (in)direct domination, including itself. We then iterated through each dominated country of $root$ and found the minimum amount of diamonds we could use if we wanted to gain k of the dominated country's votes and $j-k$ of the dominating country's other votes.

Calling $dp[0][m]$ would give us the best way to get m of the root country's (the dummy node which dominates all countries) votes.

4 Distinct Subsequences

(Java) First, we used a hash table to store the lists of the indices where every character in our subsequence Z is stored in the main string X.

We then created a dp array $total$, where $total[i]$ contains the number of distinct subsequences of the first $i+1$ characters found in X. We populated this dynamic programming array by adding $total[i-1]$ to $total[i]$ for every occurence of a character of Z at index i.

Once our dp array $total$ is populated, the answer is found at its last entry.