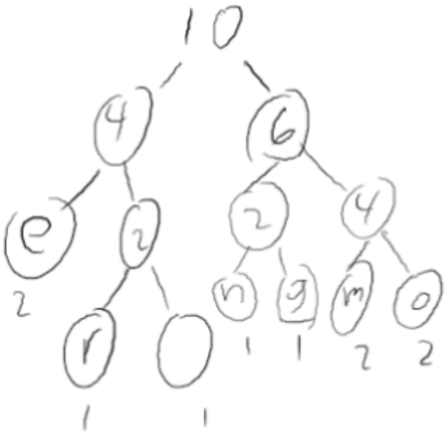
Lab 8:
Tries and Dynamic Programming

Michael Morikawa

April 30, 2020

Lab Questions

Question 1 Construct a Huffman coding tree for the following input string “more money”. Show the code for each character and include the total number of bits for the input string.



e:	000
r:	010
" ":	011
n:	100
y:	101
m:	110
o:	111

Input bits = 7*10 = 70

Question 2 Lab question 2: Explain the main difference between standard tries and compressed tries. How much space is saved from standard tries to compressed tries?
The difference between the two is that compressed tries make each internal node have at least 2 children. Not much space is saved if you are simply storing the characters/substrings and the nodes. You lessen the ammount of total nodes but the amount of data stored within the nodes is the same. You can modify it to work with a collection of strings where you instead store a 3-tuple. The first element is the index of the string, and the other 2 are the begin and end points of the string. This will reduce the size from O(n), n = total length of strings, to O(s), s = number of strings.

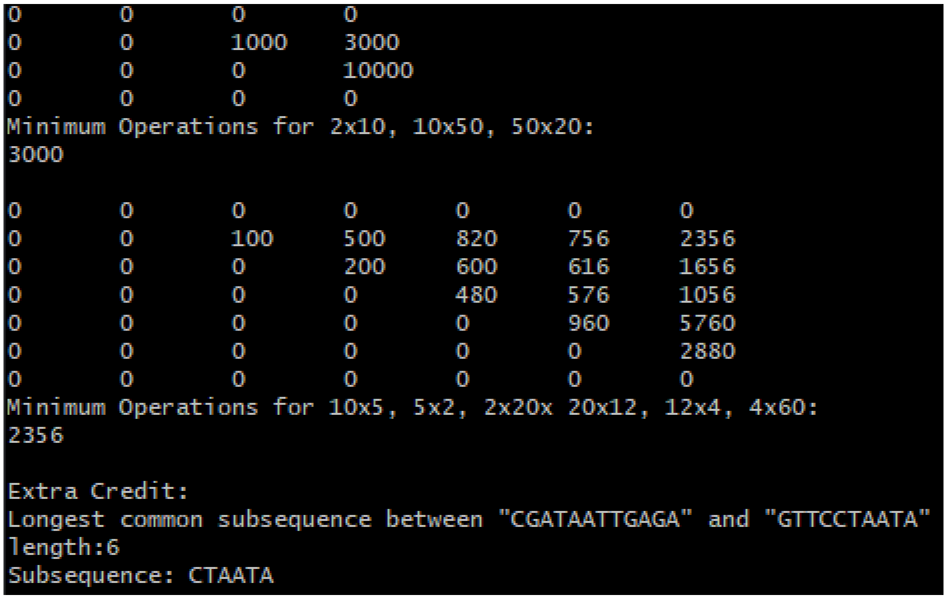
Source Code

main.cpp

```
#include <iostream>
#include <iomanip>
#include <cstring>
```

```
#include <vector>

int MatrixChainProduct(int dims[], int n);
void printTable(const std::vector<std::vector<int>> &arr);

std::vector<std::vector<int>> LongestCommonSubsequence(const char S1[], const char s2[]);

int main()
{
    int test1[] = {2, 10, 50, 20};
    int test2[] = {10, 5, 2, 20, 12, 4, 60};
    int size1 = sizeof(test1) / sizeof(int);
    int size2 = sizeof(test2) / sizeof(int);
    char Y[] = "CGATAATTGAGA";
    char X[] = "GTTCCTAATA";

    int minOps1 = MatrixChainProduct(test1, size1);
    std::cout << "Minimum Operations for 2x10, 10x50, 50x20: \n";
    std::cout << minOps1 << "\n\n";
    int minOps2 = MatrixChainProduct(test2, size2);
    std::cout << "Minimum Operations for 10x5, 5x2, 2x20x 20x12, 12x4, 4x60: \n";
    std::cout << minOps2 << "\n\n";

    auto LCSTable = LongestCommonSubsequence(X, Y);
    int length = LCSTable.back().back();

    std::cout << "Extra Credit:\n"
              << "Longest common subsequence between \"" << Y << "\" and \"" << X << "\"\n"
              << "length:" << length << "\n"
              << "Subsequence: ";
    int i = LCSTable.size() - 1;
    int j = LCSTable[0].size() - 1;
    std::vector<char> subsequence;
    //From the table generated push the subsequence to a vector
    while (i > 0 && j > 0)
    {
        //If they are equal then its part of the subsequence
        if (X[i - 1] == Y[j - 1])
        {
            subsequence.push_back(X[i - 1]);
            i--;
            j--;
        }
        else
        {
            if (LCSTable[i][j - 1] > LCSTable[i - 1][j])
            {
                j--;
            }
            else if (LCSTable[i - 1][j] > LCSTable[i][j - 1])
            {
                i--;
            }
            else
            {
                j--;
            }
        }
    }
    //Sequence is reversed so print from end
    for (int i = subsequence.size() - 1; i >= 0; i--)
    {
        std::cout << subsequence[i];
    }
    std::cout << '\n';
}

int MatrixChainProduct(int dims[], int n)
{
    std::vector<std::vector<int>> N(n, std::vector<int>(n));
    int j;
    for (int i = 1; i < n; i++)
    {
        N[i][i] = 0;
    }
    for (int b = 2; b < n; b++)
    {
        for (int i = 1; i < n - b + 1; i++)
        {
            j = i + b - 1;
            N[i][j] = INT_MAX;
            for (int k = i; k < j; k++)
            {
                N[i][j] = std::min(N[i][j], N[i][k] + N[k + 1][j] + dims[i - 1] * dims[k] * dims[j]);
            }
        }
    }
    printTable(N);
    return (int)N[1][n - 1];
}

std::vector<std::vector<int>> LongestCommonSubsequence(const char S1[], const char S2[])
{
    //Dimensions of 2D array
    //Add one to simulate a -1 index
    int n = strlen(S1) + 1;
    int m = strlen(S2) + 1;
    //initialize vector of size nxm
    std::vector<std::vector<int>> L(n, std::vector<int>(m));
    for (int i = 0; i < n; i++)
    {
        L[i][0] = 0;
    }
    for (int j = 1; j < m; j++)
    {
        L[0][j] = 0;
    }
    for (int i = 1; i < n; i++)
    {
        for (int j = 1; j < m; j++)
        {
            if (S1[i - 1] == S2[j - 1])
            {
                L[i][j] = L[i - 1][j - 1] + 1;
            }
            else
            {
                L[i][j] = std::max(L[i - 1][j], L[i][j - 1]);
            }
        }
    }
    return L;
}

void printTable(const std::vector<std::vector<int>> &arr)
{
    for (std::vector<int> vec : arr)
    {
        for (int n : vec)
        {
            std::cout << std::left << std::setw(8) << n;
        }
        std::cout << '\n';
    }
}
```

Output



```
0       0       0       0
0       0       1000    3000
0       0       0       10000
0       0       0       0
Minimum Operations for 2x10, 10x50, 50x20:
3000

0       0       0       0       0       0       0
0       0       100     500     820     756     2356
0       0       0       200     600     616     1656
0       0       0       0       480     576     1056
0       0       0       0       0       960     5760
0       0       0       0       0       0       2880
0       0       0       0       0       0       0
Minimum Operations for 10x5, 5x2, 2x20x 20x12, 12x4, 4x60:
2356

Extra Credit:
Longest common subsequence between "CGATAATTGAGA" and "GTTCCTAATA"
length:6
Subsequence: CTAATA
```