# Matrix Chain Multiplication

# Partition problem

# Coin change problem

Given a value N, if we want to make change for N cents, and we have infinite supply of each of S = { S1, S2, .. , Sm} valued coins,

how many ways can we make the change? The order of coins doesn’t matter.

Example:

For N = 10 and S = {2, 5, 3, 6}, there are five solutions:

{2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} and {5,5}.

So the output should be 5.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int count( int S[], int m, int n ) {

// table[i] will be storing the number of solutions for value i.

// We need n+1 rows as the table is constructed in

//bottom up manner using the base case (n = 0)

int table[n+1];

// Initialize all table values as 0

memset(table, 0, sizeof(table));

// Base case (If given value is 0)

table[0] = 1;

// Pick all coins one by one and update the table[] values

// after the index greater than or equal to the value of the

// picked coin

for(int i=0; i<m; i++)

for(int j=S[i]; j<=n; j++)

table[j] += table[j-S[i]];

return table[n];

}

// Driver program to test above function

int main() {

int i, j;

int arr[] = {1, 2, 3};

int m = sizeof(arr)/sizeof(arr[0]);

printf("%d ", count(arr, m, 4));

return 0;

}

Output:

4

# Longest Increasing Subsequence

The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order.

Example, the length of LIS for {10, 22, 9, 33, 21, 50, 41, 60, 80} is 6 and LIS is {10, 22, 33, 50, 60, 80}.

Optimal Substructure:

Let arr[0..n-1] be the input array and L(i) be the length of the LIS ending at index i such that arr[i] is the last element of the LIS.

Then, L(i) can be recursively written as:

L(i) = 1 + max( L(j) ) where 0 < j < i and arr[j] < arr[i]; or

L(i) = 1, if no such j exists.

To find the LIS for a given array, we need to return max(L(i)) where 0 < i < n.

Thus, we see the LIS problem satisfies the optimal substructure property as the main problem can be solved using solutions to subproblems.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define N 100000

int find\_len\_lis(int \* arg\_arr, int arg\_n);

int main(void) {

int arr[N] = {0};

int no\_elem = 0;

int i = 0;

int len\_lis = 0;

scanf("%d", &no\_elem);

for(i = 0; i < no\_elem; i++) {

scanf("%d", &arr[i]);

}

/\*

for(i = 0; i < no\_elem; i++) {

printf("%d ", arr[i]);

}

printf("\n");

\*/

len\_lis = find\_len\_lis(arr, no\_elem);

printf("%d", len\_lis);

return 0;

}

int find\_len\_lis(int \* arg\_arr, int arg\_n) {

int lenLis = 0;

int lis\_arr[N] = {1};

int i = 0;

int j = 0;

for(i = 0; i < arg\_n; i++) {

lis\_arr[i] = 1;

for(j = 0; j < i; j++) {

if( (arg\_arr[i] > arg\_arr[j]) && (lis\_arr[i] < lis\_arr[j] + 1) ) {

lis\_arr[i] = lis\_arr[j] + 1;

}

}

}

for(i = 0; i < arg\_n; i++) {

//printf("%d ", lis\_arr[i]);

if(lenLis < lis\_arr[i]) {

lenLis = lis\_arr[i];

}

}

//printf("\n");

return lenLis;

}

Input:

5

3 10 2 1 20

Output:

3

# Longest Common Subsequence

LCS Problem Statement: Given two sequences, find the length of longest subsequence present in both of them.

A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. For example, “abc”, “abg”, “bdf”, “aeg”, ‘”acefg”, .. etc are subsequences of “abcdefg”. So a string of length n has 2^n different possible subsequences.

It is a classic computer science problem, the basis of diff (a file comparison program that outputs the differences between two files), and has applications in bioinformatics.

Examples:

LCS for input Sequences “ABCDGH” and “AEDFHR” is “ADH” of length 3.

LCS for input Sequences “AGGTAB” and “GXTXAYB” is “GTAB” of length 4.

**Optimal Substructure:**

Let the input sequences be X[0..m-1] and Y[0..n-1] of lengths m and n respectively. And let L(X[0..m-1], Y[0..n-1]) be the length of LCS of the two sequences X and Y. Following is the recursive definition of L(X[0..m-1], Y[0..n-1]).

If last characters of both sequences match (or X[m-1] == Y[n-1]) then

L(X[0..m-1], Y[0..n-1]) = 1 + L(X[0..m-2], Y[0..n-2])

If last characters of both sequences do not match (or X[m-1] != Y[n-1]) then

L(X[0..m-1], Y[0..n-1]) = MAX ( L(X[0..m-2], Y[0..n-1]), L(X[0..m-1], Y[0..n-2])

Examples:

Consider the input strings “AGGTAB” and “GXTXAYB”. Last characters match for the strings. So length of LCS can be written as:

L(“AGGTAB”, “GXTXAYB”) = 1 + L(“AGGTA”, “GXTXAY”)

Consider the input strings “ABCDGH” and “AEDFHR. Last characters do not match for the strings. So length of LCS can be written as:

L(“ABCDGH”, “AEDFHR”) = MAX ( L(“ABCDG”, “AEDFHR”), L(“ABCDGH”, “AEDFH”) )

So the LCS problem has optimal substructure property as the main problem can be solved using solutions to subproblems.

/\* Dynamic Programming C/C++ implementation of LCS problem \*/

#include<bits/stdc++.h>

int max(int a, int b);

/\* Returns length of LCS for X[0..m-1], Y[0..n-1] \*/

int lcs( char \*X, char \*Y, int m, int n ) {

int L[m+1][n+1];

int i, j;

/\* Following steps build L[m+1][n+1] in bottom up fashion. Note

that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] \*/

for (i=0; i<=m; i++)

{

for (j=0; j<=n; j++)

{

if (i == 0 || j == 0)

L[i][j] = 0;

else if (X[i-1] == Y[j-1])

L[i][j] = L[i-1][j-1] + 1;

else

L[i][j] = max(L[i-1][j], L[i][j-1]);

}

}

/\* L[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1] \*/

return L[m][n];

}

/\* Utility function to get max of 2 integers \*/

int max(int a, int b) {

return (a > b)? a : b;

}

/\* Driver program to test above function \*/

int main() {

char X[] = "AGGTAB";

char Y[] = "GXTXAYB";

int m = strlen(X);

int n = strlen(Y);

printf("Length of LCS is %d", lcs( X, Y, m, n ) );

return 0;

}

Output:

Length of LCS is 4

# References

<https://www.geeksforgeeks.org/dynamic-programming/>