**Given 3 strings of all having length < 100, write a program to find the longest common subsequence in all three given sequences.**

**Algorithm: LCS**

**Input:** Three sequences(array), a[], b[], c[].

**Output:** Length of LCS.

1. Set l1🡨 array\_size(a)-1, l2🡨 array\_size(b) -1, l3🡨 array\_size(c) -1
2. Initialize 3-D array arr[l1][l2][l3]
3. for i =0 to l1

3.1. for j = 0 to l2

3.1.1. for k = 0 to l3

3.1.1.1. if i==0 or j==0 or k==0 (any of the sequence is of 0-length, ie. empty)

3.1.1.1.1. arr[i][j][k] = 0;

3.1.1.2. else if (a[i-1] == b[j-1] && a[i-1]==c[k-1]) (if all characters matches)

3.1.1.2.1 arr[i][j][k] = arr[i-1][j-1][k-1] + 1

3.1.1.3. else

3.1.1.3.1. arr[i][j][k] = max(arr[i-1][j][k], arr[i][j-1][k]), arr[i][j][k-1]);

3.1.2 end for

3.2 end for

4. end for

5. return arr[l1][l2][l3]

5. End Algorithm.

#include<stdio.h>

int max(int a, int b){

if(a>b) return a;

else if(b>a) return b;

}

int function( char a[], char b[], char c[], int l1, int l2, int l3){

int arr[l1+1][l2+1][l3+1];

int i,j,k;

for (i=0;i<=l1;i++){

for (j=0;j<=l2;j++){

for (k=0;k<=l3;k++){

if (i == 0 || j == 0||k==0) arr[i][j][k] = 0;

else if (a[i-1] == b[j-1] && a[i-1]==c[k-1]) arr[i][j][k] = arr[i-1][j-1][k-1] + 1;

else

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

}

}

} return arr[l1][l2][l3];

}

int main(){

char a[] = "As123"; char b[] = "123bd"; char c[] = "12123X";

int l1 = sizeof(a)-1; int l2 = sizeof(b)-1; int l3 = sizeof(c)-1;

int res = function(a, b, c, l1, l2, l3);

printf("Length = %d",res);

return 0;

}

T(n) = c1 + c2 + l1[ c3 + l2[ c4 + l3( c5 + c6)]] +c7

= c1 + l1(c2) + l1l2(c3) + l1l2l3(c4)

T(n) = O(l1.l2.3)

**Given a sequence of matrices, find the most efficient way to multiply these matrices together.**

**The problem is not actually to perform the multiplications, but merely to decide in which order**

**to perform the multiplications.**

**Algorithm: MCM**

**Input:** Array p[] denoting the chain of matrices.

**Output:** Least number of multiplications.

1. Initialize arr[p\_size][p\_size]
2. for i = 1 to arr\_size-1

2.1. arr[i][i] = 0

3. for l =2 to n-1

3.1. for i = 1 to n-l

3.1.1. Set j🡨i+l-1

3.1.2. arr[i][j] = INT\_MAX.

3.1.3. for k = i to j-1

3.1.3.1. set q🡨 arr[i][k] + arr[k+1][j] + p[i-1]\*p[k]\*p[j]

3.1.3.2. arr[i][j] 🡨 q.

3.1.4. endfor

3.2 end for

4. endfor

6. return arr[1][n-1].

5.End Algorithm.

#include<stdio.h>

#include<limits.h>

int Matrix(int [], int);

int main(){

int arr[] = {1, 2, 3, 4};

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

printf("Least operations are: %d ", Matrix(arr, size));

}

int Matrix(int p[], int n)

{

int arr[n][n];

int i, j, k, L, q;

for (i=1; i<n; i++)

arr[i][i] = 0;

for (L=2; L<n; L++){

for (i=1; i<n-L+1; i++){

j = i+L-1;

arr[i][j] = INT\_MAX;

for (k=i; k<=j-1; k++){

q = arr[i][k] + arr[k+1][j] + p[i-1]\*p[k]\*p[j];

arr[i][j] = q;

}

}

}

return arr[1][n-1];

}

**Largest Sum Contiguous Subarray**

**Algorithm: LSCS**

**Input:** Array a[] of positive or negative numbers.

**Output:** Largest sum and corresponding subarray.

1. Initilize max🡨 INT\_MIN, last🡨 0, start 🡨 0, end 🡨 0.
2. For i = 0 to array\_size(a) -1 do

2.1. last 🡨 last+a[i]

2.2. if (max<last)

2.2.1. max 🡨 last

2.2.2. end 🡨 i

2.3. if (last<0)

2.3.1. last 🡨 0

2.3.2. start 🡨 i+1

3. endfor.

4. for i = start to end do

4.1. print a[i]

5. endfor

6. print(max)

7. End Algorithm.

#include<stdio.h>

#include<limits.h>

int function(int a[], int n){

int max = INT\_MIN, last = 0 ,i,start=0,end=0;

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

last=last+a[i];

if (max<last){

max = last;

end=i;

}

if (last<0){

last = 0;

start=i+1;

}

}

for(i=start;i<=end;i++)

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

return max;

}

int main(){

int a[] = {5, 15, -30, 10, -5, 40, 10};

int n = sizeof(a)/sizeof(a[0]);

int res = function(a, n);

printf("Maximum sum is %d", res);

return 0;

}

**Devise an algorithm that takes a sequence x[1 : : : n] and returns the (length of the) longest palindromic subsequence.**

**Algorithm: LPS**

**Input:** String (say s).

**Output:** Length of longest palindromic subsequence.

1. Initialize n 🡨 strlen(string)
2. Initialize an array L[n][n].
3. For i = 0 to n-1

3.1. L[i][i] 🡨 1. (For string of length 1).

4. For cl = 2 to n do

4.1. For i = 0 to n-cl do

4.1.1. j 🡨 i+cl-1.

4.1.2. if (str[i] == str[j] && cl == 2)

4.1.2.1. L[i][j] 🡨 2. (For Strings of type aa of length 2.)

4.1.3. else if (str[i] == str[j])

4.1.3.1. L[i][j] 🡨 L[i+1][j-1] + 2.

4.1.4. else

4.1.4.1. L[i][j] 🡨 max(L[i][j-1], L[i+1][j]).

4.2. endfor

5. endfor.

6. return L[0][n-1]

7. End Algorithm.

#include<stdio.h>

#include<string.h>

int max (int x, int y) { if(x>y) return x; else return y; }

int max\_palindrome(char \*str){

int n = strlen(str);

int i, j, cl;

int L[n][n];

for (i = 0; i < n; i++) L[i][i] = 1;

for (cl=2; cl<=n; cl++){

for (i=0; i<n-cl+1; i++){

j = i+cl-1;

if (str[i] == str[j] && cl == 2) L[i][j] = 2;

else if (str[i] == str[j]) L[i][j] = L[i+1][j-1] + 2;

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

}

}

return L[0][n-1];

}

int main(){

char string[] = "agbdba";

int n = strlen(string);

printf ("The max palindrome length is %d", max\_palindrome(string));

getchar();

return 0;

}

**A list of n positive integers a1; a2; : : : ; an; and a positive integer t is given. Write a program to find subset of the ai's add up to t? (You can use each ai at most once.)**

**Algorithm: Subset Sum**

**Input:** Array of numbers s[], target sum.

**Output:** Subsequences of total sum equal to target sum.

1. Initialize an array t[] of size array\_size(s).
2. Initialize s\_size 🡨 array\_size(s), t\_size 🡨 0, sum 🡨 0, ite 🡨 0.
3. Call to Subset\_function(s[], t[], s\_size, t\_size, sum, ite, target\_sum).
4. Return
5. End Algorithm.

**Sub-Algorithm:** Subset\_function.

Input: s[], t[], s\_size, t\_size, sum, ite, target\_sum.

Output: Call to print\_subset function with parameters as t[] and t\_size.

1. Initialize s\_size 🡨 array\_size(s), t\_size 🡨 0, sum 🡨 0, ite 🡨 0.
2. If ( target\_sum = sum )
   1. Call to print\_subset( t[], t\_size).
   2. Subset\_function(s, t, s\_size, t\_size-1, sum - s[ite], ite + 1, target\_sum)
   3. Return
   4. End SubAlgorithm.
3. Else
   1. For i = ite to s\_size-1 do
      1. t[t\_size] = s[i].
      2. Subset\_function(s, t, s\_size, t\_size + 1, sum + s[i], i + 1, target\_sum).
   2. Endfor

**Sub-Algorithm:** Print\_subset.

Input: t[], t\_size.

Output: Print results (ie., subsets whose sum is equal to target\_sum).

1. For i = 0 to t\_size-1 do
   1. Print t[i].

2. Endfor.

3. End SubAlgorithm.

#include <stdio.h>

#include <stdlib.h>

#define ARRAYSIZE(a) (sizeof(a))/(sizeof(a[0]))

void printSubset(int A[], int size){

int i;

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

printf("%d\t", A[i]);

printf("\n");

}

void subset\_sum(int s[], int t[], int s\_size, int t\_size, int sum, int ite, int const target\_sum){

if( target\_sum == sum ){

printSubset(t, t\_size);

subset\_sum(s, t, s\_size, t\_size-1, sum - s[ite], ite + 1, target\_sum);

return;

}

else{

int i;

for( i = ite; i < s\_size; i++ ){

t[t\_size] = s[i];

subset\_sum(s, t, s\_size, t\_size + 1, sum + s[i], i + 1, target\_sum);

}

}

}

int main(){

int array[] = {5, 10, 15};

int target\_sum = 15;

int size = ARRAYSIZE(array);

int tuplet\_vector[size];

subset\_sum(array, tuplet\_vector, size, 0, 0, 0, target\_sum);

return 0;

}

**Write a program that calculates the highest sum of numbers passed on a route that starts at the top and ends somewhere on the base.**

**7**

**3 8**

**8 1 0**

**2 7 4 4**

**4 5 2 6 5**

**Algorithm: HighestSum**

**Input:** 2-D Array of numbers a[].

**Output:** Highest Sum.

1. Input number of rows of 2-D array. (say n)
2. Initialize a[n+1][n+1] 🡨-1, and dp[n+1][n+1] 🡨 -1.
3. Input Array in required triangle format.
4. Set max 🡨 findMaxSum( a[n+1][n+1], dp[n+1][n+1], 1,1,n)
5. Print max.
6. End Algorithm.

**SubAlgorithm:** findMaxSum.

Input: a[][], dp[][], i, j, n(number of rows).

Output: Max\_Sum

1. if(i>end || j>end)
   1. Return 0.

2. if(dp[i][j]!=-1)

2.1. return dp[i][j].

3. else if(i<=end)

3.1. return dp[i][j]=a[i][j]+ max[findMaxSum(i+1,j,end),findMaxSum(i+1,j+1,end)].

4. else

4.1. return 0.

import java.util.Arrays;

import java.util.Scanner;

public class Ass5\_38{

static int a[][], int dp[][];

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the number of rows"); int n=sc.nextInt();

a=new int[n+1][n+1];

dp=new int[n+1][n+1];

for(int[] t:dp) Arrays.fill(t,-1);

System.out.println("Enter elements");

for(int i=1;i<=n;++i){

for(int j=1;j<=i;++j){ a[i][j]=sc.nextInt(); }

}

int sum=findMaxSum(1,1,n); System.out.println("Answer is "+sum);

}

private static int findMaxSum(int i, int j, int end) {

if(i>end || j>end) return 0;

if(dp[i][j]!=-1) return dp[i][j];

else if(i<=end) return dp[i][j]=a[i][j]+Math.max(findMaxSum(i+1,j,end),findMaxSum(i+1,j+1,end));

else return 0;

}

}

**Consider a highway of M miles. The task is to place billboards on the highway such that revenue is maximized. The possible sites for billboards are given by number x1 < x2 < ….. < xn-1 < xn, specifying positions in miles measured from one end of the road. If we place a billboard at position xi, we receive a revenue of ri > 0. There is a restriction that no two billboards can be placed within t miles or less than it.**

**Note : All sites from x1 to xn are in range from 0 to M as need to place billboards on a highway of M miles.**

**Algorithm: BillBoard**

**Input:** Array x[] of billboard distances and corresponding Array revenue[], m (total miles),t (minimum distance).

**Output:** Maximum Revenue..

1. Initialize array maxRev[n+1] 🡨 0.
2. Initialize next = 0.
3. For i = 1 to m do
   1. if (next < n)

3.1.1. if (x[next] != i)

3.1.1.1. maxRev[i] = maxRev[i-1]

3.1.2. else

3.1.2.1. if (i <= t)

3.1.2.1.1. maxRev[i] = max(maxRev[i-1], revenue[next])

3.1.2.2. else

3.1.2.2.1. maxRev[i] = max(maxRev[i-t-1]+revenue[next],maxRev[i-1])

3.1.2.3. next++

3.1.3. end else

3.2. end if.

3.3. else

3.3.1. maxRev[i] = maxRev[i - 1]

4. end For.

5. Return maxRev[m].

6. End Algorithm.

#include<stdio.h>

int max(int a, int b) { if(a>b) return a; else return b; }

int function(int m, int x[], int revenue[], int n, int t){

int maxRev[m+1], 0,i;

memset(maxRev, 0, sizeof(maxRev));

for (i = 1; i <= m; i++){

if (next < n){

if (x[next] != i) maxRev[i] = maxRev[i-1];

else{

if (i <= t) maxRev[i] = max(maxRev[i-1], revenue[next]);

else maxRev[i] = max(maxRev[i-t-1]+revenue[next],maxRev[i-1]);

next++;

}

}

else maxRev[i] = maxRev[i - 1];

} return maxRev[m];

}

int main(){

int m = 20, t = 5, x[] = {6, 7, 12, 13, 14}, revenue[] = {5, 6, 5, 3, 1};

int n = sizeof(x)/sizeof(x[0]);

printf("Maximum revenue is: %d\n",function(m, x, revenue, n, t));

return 0;

}

**Given time taken by n tasks. Find the minimum time needed to finish the tasks such that skipping of tasks is allowed, but can not skip two consecutive tasks.**

**Algorithm: MinJob**

**Input:** Array arr[] of time required for jobs.

**Output:** Minimum time.

1. Initialize n 🡨 array\_size(arr).
2. if (n <= 0)
   1. return 0
3. Initialize incl 🡨 arr[0], excl 🡨 0
4. for i = 1 to n-1 do
   1. Set incl\_new 🡨 arr[i] + min(excl, incl)
   2. excl\_new 🡨 incl
   3. incl 🡨 incl\_new
   4. excl 🡨 excl\_new
5. return min(incl, excl)
6. End Algorithm.

#include<stdio.h>

int min(int a, int b) {

if(a<b) return a;

else return b;

}

int minTime(int arr[], int n){

if (n <= 0) return 0;

int incl = arr[0];

int excl = 0;

int i;

for (i=1; i<n; i++){

int incl\_new = arr[i] + min(excl, incl);

int excl\_new = incl;

incl = incl\_new;

excl = excl\_new;

}

return min(incl, excl);

}

int main(){

int arr1[] = {10, 5, 2, 7, 10};

int n1 = sizeof(arr1)/sizeof(arr1[0]);

printf("%d\n",minTime(arr1, n1));

return 0;

}