***NAME : Himanshu Dixit***

***ENROLL NO. : 21103262***

***BATCH : B11***

***Algorithms and Problem Solving [15B17CI471]***

***Lab - Week 13***

**Solution 1,2,3,4,5,7)**

#include <bits/stdc++.h>

using namespace std;

#define MAX 128

// Function to check whether a character is

// vowel or not

bool isVowel(char x)

{

return (x == 'a' || x == 'e' || x == 'i' ||

x == 'o' || x == 'u' || x == 'A' ||

x == 'E' || x == 'I' || x == 'O' ||

x == 'U');

}

int solution1(char s[], int k)

{

int n = strlen(s);

int c[MAX];

memset(c, 0, sizeof(c));

int result = -1;

for (int i = 0, j = -1; i < n; ++i) {

int x = s[i];

if (isVowel(x)) {

if (++c[x] == 1) {

--k;

}

}

while (k < 0) {

x = s[++j];

if (isVowel(x)) {

if (--c[x] == 0) {

++k;

}

}

}

if (k == 0)

result = max(result, i - j);

}

return result;

}

int solution2(vector<int> arr){

vector<int> dp(arr.size(),0);

if(arr[1]>=dp[1]){

dp[1] = arr[1];

}

int maxi=dp[1];

for(int i=1;i<dp.size();i++){

if(dp[i-1]+arr[i]>dp[i]){

dp[i] = dp[i-1] + arr[i];

}

maxi = fmax(maxi,dp[i]);

}

return maxi;

}

int solution3(string s1,string s2,int index1,int index2,vector<vector<int>> &dp){

if(index1==0 || index2==0){

return 0;

}

if(s1[index1-1] == s2[index2-1]){

return dp[index1][index2] = 1 + solution3(s1,s2,index1-1,index2-1,dp);

}

if(dp[index1][index2]!=-1){

return dp[index1][index2];

}

int op1 = solution3(s1,s2,index1-1,index2,dp);

int op2 = solution3(s1,s2,index1,index2-1,dp);

return dp[index1][index2] = fmax(op1,op2);

}

int solution4(int p[], int i, int j)

{

if (i == j)

return 0;

int k;

int mini = INT\_MAX;

int count;

for (k = i; k < j; k++)

{

count = solution4(p, i, k)

+ solution4(p, k + 1, j)

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

mini = min(count, mini);

}

return mini;

}

int solution5(string s1,string s2,int index1,int index2,vector<vector<int>> &dp){

if(index1==0 || index2==0){

return 0;

}

if(s1[index1-1] == s2[index2-1]){

return dp[index1][index2] = 1 + solution3(s1,s2,index1-1,index2-1,dp);

}

if(dp[index1][index2]!=-1){

return dp[index1][index2];

}

int op1 = solution3(s1,s2,index1-1,index2,dp);

int op2 = solution3(s1,s2,index1,index2-1,dp);

return dp[index1][index2] = fmax(op1,op2);

}

int solution7(vector<int>& nums) {

int n = nums.size();

vector<vector<int>> dp(n, vector<int>(n, 0));

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

dp[i][i] = nums[i];

}

for(int len = 2; len <= n; len++) {

for(int i = 0; i <= n-len; i++) {

int j = i+len-1;

for(int k = i; k < j; k++) {

if(nums[k+1] == '+') {

dp[i][j] = max(dp[i][j], dp[i][k] + dp[k+1][j]);

} else if(nums[k+1] == '-') {

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

}

}

}

}

return dp[0][n-1];

}

int main(){

//question1

char s[] = "artyebui";

int k = 2;

int len = solution1(s, k);

cout<<"length of longest substring "<<len<<endl;

//question2

vector<int> input2{5,15,-30,10,-5,40,10};

cout<<"maximum sum "<<solution2(input2)<<endl;

//question3

string s1 = "vshkl";

string s2 = "vishal";

vector<vector<int>> dp(s1.size()+1,vector<int>(s2.size()+1,-1));

int ans = solution3(s1,s2,s1.length(),s2.length(),dp);

cout<<"number of operations "<<fmax(s1.size()-ans,s2.size()-ans)<<endl;

//question4

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

int N = 5;

cout << "Minimum number of multiplications is "<< solution4(arr, 1, N - 1)<<endl;

//question5

string s3 = "sitting";

string s4 = "kitten";

vector<vector<int>> dp1(s3.size()+1,vector<int>(s4.size()+1,-1));

cout<<"longest common subsequence "<<solution5(s3,s4,s3.size(),s4.size(),dp1)<<endl;

//question7

vector<int> nums = {1, '+', 3, '-', 2, '-', 5, '+', 1, '-', 6, '+', 7};

int maxVal = solution7(nums);

cout<< "Maximum value of expression: " << maxVal << endl;

return 0;

}

**Solution 6)**

#include<bits/stdc++.h>

using namespace std;

class NAryTree

{

int N;

list<int> \*adj;

void getMinIterUtil(int v, int minItr[]);

public:

NAryTree(int N);

void addChild(int v, int w);

int getMinIter();

static int compare(const void \* a, const void \* b);

};

NAryTree::NAryTree(int N)

{

this->N = N;

adj = new list<int>[N];

}

void NAryTree::addChild(int v, int w)

{

adj[v].push\_back(w);

}

void NAryTree::getMinIterUtil(int u, int minItr[])

{

minItr[u] = adj[u].size();

int \*minItrTemp = new int[minItr[u]];

int k = 0, tmp = 0;

list<int>::iterator i;

for (i = adj[u].begin(); i!= adj[u].end(); ++i)

{

getMinIterUtil(\*i, minItr);

minItrTemp[k++] = minItr[\*i];

}

qsort(minItrTemp, minItr[u], sizeof (int), compare);

for (k = 0; k < adj[u].size(); k++)

{

tmp = minItrTemp[k] + k + 1;

minItr[u] = max(minItr[u], tmp);

}

delete[] minItrTemp;

}

int NAryTree::getMinIter()

{

int \*minItr = new int[N];

int res = -1;

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

minItr[i] = 0;

getMinIterUtil(0, minItr);

res = minItr[0];

delete[] minItr;

return res;

}

int NAryTree::compare(const void \* a, const void \* b)

{

return ( (int)b - (int)a );

}

int main()

{

// TestCase 1

NAryTree tree1(17);

tree1.addChild(0, 1);

tree1.addChild(0, 2);

tree1.addChild(0, 3);

tree1.addChild(0, 4);

tree1.addChild(0, 5);

tree1.addChild(0, 6);

tree1.addChild(1, 7);

tree1.addChild(1, 8);

tree1.addChild(1, 9);

tree1.addChild(4, 10);

tree1.addChild(4, 11);

tree1.addChild(6, 12);

tree1.addChild(7, 13);

tree1.addChild(7, 14);

tree1.addChild(10, 15);

tree1.addChild(11, 16);

cout << "TestCase 1 - Minimum Rounds: "

<< tree1.getMinIter() << endl;

// TestCase 2

NAryTree tree2(3);

tree2.addChild(0, 1);

tree2.addChild(0, 2);

cout << "TestCase 2 - Minimum Rounds: "

<< tree2.getMinIter() << endl;

// TestCase 3

NAryTree tree3(1);

cout << "TestCase 3 - Minimum Rounds: "

<< tree3.getMinIter() << endl;

// TestCase 4

NAryTree tree4(6);

tree4.addChild(0, 1);

tree4.addChild(1, 2);

tree4.addChild(2, 3);

tree4.addChild(3, 4);

tree4.addChild(4, 5);

cout << "TestCase 4 - Minimum Rounds: "

<< tree4.getMinIter() << endl;

// TestCase 5

NAryTree tree5(6);

tree5.addChild(0, 1);

tree5.addChild(0, 2);

tree5.addChild(2, 3);

tree5.addChild(2, 4);

tree5.addChild(2, 5);

cout << "TestCase 5 - Minimum Rounds: "

<< tree5.getMinIter() << endl;

// TestCase 6

NAryTree tree6(6);

tree6.addChild(0, 1);

tree6.addChild(0, 2);

tree6.addChild(2, 3);

tree6.addChild(2, 4);

tree6.addChild(3, 5);

cout << "TestCase 6 - Minimum Rounds: "

<< tree6.getMinIter() << endl;

// TestCase 7

NAryTree tree7(14);

tree7.addChild(0, 1);

tree7.addChild(0, 2);

tree7.addChild(0, 3);

tree7.addChild(1, 4);

tree7.addChild(2, 5);

tree7.addChild(2, 6);

tree7.addChild(4, 7);

tree7.addChild(5, 8);

tree7.addChild(5, 9);

tree7.addChild(7, 10);

tree7.addChild(8, 11);

tree7.addChild(8, 12);

tree7.addChild(10, 13);

cout << "TestCase 7 - Minimum Rounds: "

<< tree7.getMinIter() << endl;

// TestCase 8

NAryTree tree8(14);

tree8.addChild(0, 1);

tree8.addChild(0, 2);

tree8.addChild(0, 3);

tree8.addChild(0, 4);

tree8.addChild(0, 5);

tree8.addChild(1, 6);

tree8.addChild(2, 7);

tree8.addChild(3, 8);

tree8.addChild(4, 9);

tree8.addChild(6, 10);

tree8.addChild(7, 11);

tree8.addChild(8, 12);

tree8.addChild(9, 13);

cout << "TestCase 8 - Minimum Rounds: "

<< tree8.getMinIter() << endl;

// TestCase 9

NAryTree tree9(25);

tree9.addChild(0, 1);

tree9.addChild(0, 2);

tree9.addChild(0, 3);

tree9.addChild(0, 4);

tree9.addChild(0, 5);

tree9.addChild(0, 6);

tree9.addChild(1, 7);

tree9.addChild(2, 8);

tree9.addChild(3, 9);

tree9.addChild(4, 10);

tree9.addChild(5, 11);

tree9.addChild(6, 12);

tree9.addChild(7, 13);

tree9.addChild(8, 14);

tree9.addChild(9, 15);

tree9.addChild(10, 16);

tree9.addChild(11, 17);

tree9.addChild(12, 18);

tree9.addChild(13, 19);

tree9.addChild(14, 20);

tree9.addChild(15, 21);

tree9.addChild(16, 22);

tree9.addChild(17, 23);

tree9.addChild(19, 24);

cout << "TestCase 9 - Minimum Rounds: "

<< tree9.getMinIter() << endl;

}