MAYANK SRIVASTAVA

9922103143

WEEK 7

1-)

#include <bits/stdc++.h>

using namespace std;

#define MAX 128

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 KDistinctVowel(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 main(void) {

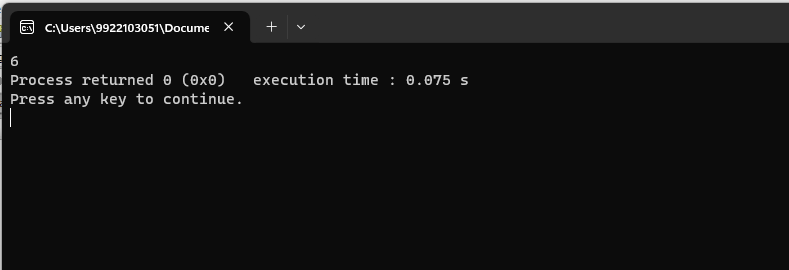
char s[] = "artyebui";

int k = 2;

cout << KDistinctVowel(s, k);

return 0;

}



2-)

#include <bits/stdc++.h>

using namespace std;

int maxSubArraySum(int a[], int size) {

int max\_so\_far = INT\_MIN;

int max\_ending\_here = 0;

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

max\_ending\_here += a[i];

if (max\_so\_far < max\_ending\_here)

max\_so\_far = max\_ending\_here;

if (max\_ending\_here < 0)

max\_ending\_here = 0;

}

return max\_so\_far;

}

int main() {

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

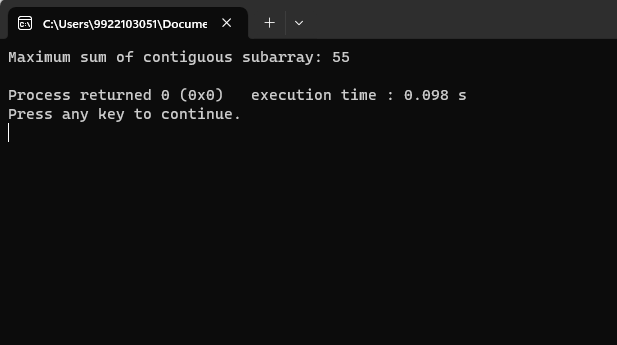
int n = sizeof(arr) / sizeof(arr[0]);

int result = maxSubArraySum(arr, n);

cout << "Maximum sum of contiguous subarray: " << result << endl;

return 0;

}



3-)

#include <iostream>

#include <string>

using namespace std;

int min(int x, int y, int z) {

return min(min(x, y), z);

}

int minEditDistance(string src, string dest) {

int len1 = src.length();

int len2 = dest.length();

int dp[len1 + 1][len2 + 1];

for (int i = 0; i <= len1; ++i)

dp[i][0] = i;

for (int j = 0; j <= len2; ++j)

dp[0][j] = j;

for (int i = 1; i <= len1; ++i) {

for (int j = 1; j <= len2; ++j) {

if (src[i - 1] == dest[j - 1])

dp[i][j] = dp[i - 1][j - 1];

else

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

}

}

return dp[len1][len2];

}

int main() {

string src, dest;

cout << "Enter source string: ";

cin >> src;

cout << "Enter destination string: ";

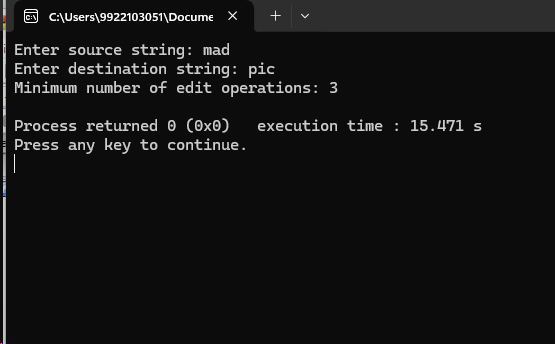
cin >> dest;

int result = minEditDistance(src, dest);

cout << "Minimum number of edit operations: " << result << endl;

return 0;

}



4-)

#include <bits/stdc++.h>

using namespace std;

int min(int a, int b) {

return (a < b) ? a : b;

}

int MatrixChainOrder(int p[], int n) {

int dp[n][n];

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

dp[i][i] = 0;

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

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

int j = i + len - 1;

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

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

int cost = dp[i][k] + dp[k + 1][j] + p[i - 1] \* p[k] \* p[j];

dp[i][j] = min(dp[i][j], cost);

}

}

}

return dp[1][n - 1];

}

int main() {

int arr[] = {40, 20, 30, 10, 30};

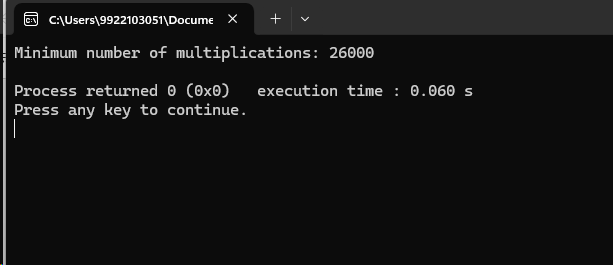
int n = sizeof(arr) / sizeof(arr[0]);

int result = MatrixChainOrder(arr, n);

cout << "Minimum number of multiplications: " << result << endl;

return 0;

}



5-)

#include <iostream>

#include <string>

using namespace std;

int min(int a, int b) {

return (a < b) ? a : b;

}

string findLCS(string S1, string S2) {

int m = S1.length();

int n = S2.length();

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

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

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

dp[i][j] = 0;

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

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

if (S1[i - 1] == S2[j - 1])

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

else

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

}

}

string lcs;

int i = m, j = n;

while (i > 0 && j > 0) {

if (S1[i - 1] == S2[j - 1]) {

lcs = S1[i - 1] + lcs;

--i;

--j;

} else if (dp[i - 1][j] > dp[i][j - 1])

--i;

else

--j;

}

return lcs;

}

int main() {

string S1 = "BCDAACD";

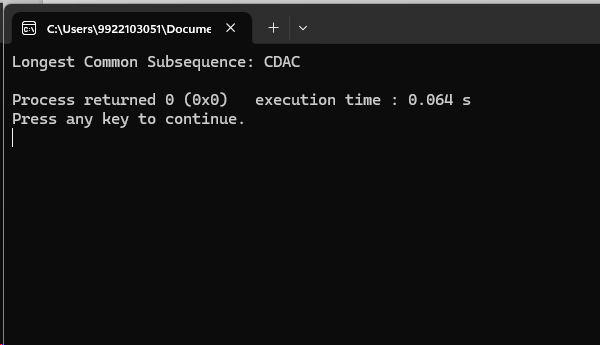
string S2 = "ACDBAC";

string longestCommonSubsequence = findLCS(S1, S2);

cout << "Longest Common Subsequence: " << longestCommonSubsequence << endl;

return 0;

}



6-)

#include <iostream>

#include <vector>

using namespace std;

struct TreeNode {

int val;

vector<TreeNode\*> children;

TreeNode(int x) : val(x) {}

};

int dfs(TreeNode\* root, int& maxDepth) {

if (!root)

return 0;

int rounds = 0;

for (TreeNode\* child : root->children) {

rounds = max(rounds, dfs(child, maxDepth) + 1);

}

maxDepth = max(maxDepth, rounds);

return rounds;

}

int minRounds(TreeNode\* root) {

int maxDepth = 0;

dfs(root, maxDepth);

return maxDepth;

}

int main() {

TreeNode\* root = new TreeNode(1);

root->children.push\_back(new TreeNode(2));

root->children.push\_back(new TreeNode(3));

root->children[0]->children.push\_back(new TreeNode(4));

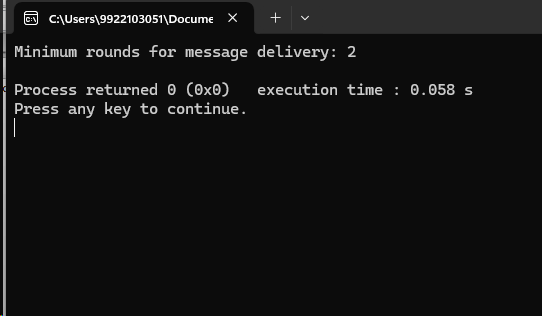
root->children[0]->children.push\_back(new TreeNode(5));

int result = minRounds(root);

cout << "Minimum rounds for message delivery: " << result << endl;

return 0;

}



7-)

#include <iostream>

#include <string>

#include <climits>

using namespace std;

int evaluateExpression(string expression) {

int result = 0;

char operation = '+';

int currentNumber = 0;

int index = 0;

while (index < expression.size()) {

if (expression[index] == '+' || expression[index] == '-') {

if (operation == '+') {

result += currentNumber;

} else {

result -= currentNumber;

}

operation = expression[index];

currentNumber = 0;

} else {

currentNumber = currentNumber \* 10 + (expression[index] - '0');

}

index++;

}

if (operation == '+') {

result += currentNumber;

} else {

result -= currentNumber;

}

return result;

}

int maximizeExpression(string expression) {

int maxVal = INT\_MIN;

for (int i = 0; i < expression.size(); i++) {

if (expression[i] == '+' || expression[i] == '-') {

string left = expression.substr(0, i);

string right = expression.substr(i + 1);

int leftMax = maximizeExpression(left);

int rightMax = maximizeExpression(right);

if (expression[i] == '+') {

maxVal = max(maxVal, leftMax + rightMax);

} else {

maxVal = max(maxVal, leftMax - rightMax);

}

}

}

if (maxVal == INT\_MIN) {

return evaluateExpression(expression);

}

return maxVal;

}

int main() {

string expression = "1+3-2-5+1-6+7";

int maxVal = maximizeExpression(expression);

cout << "Maximum possible value: " << maxVal << endl;

return 0;

}

