ASSIGNMENT #5 – Comparing LCS and LPS

C Code of Longest Common Subsequence

#include <iostream>

#include <vector>

#include <string>

using namespace std;

void LCS() {

string x = " ABCBDAB";

string y = " BDCABA";

vector < vector <int> > C(x.size(), vector <int>(y.size())), movement(x.size(), vector <int>(y.size()));

int m = x.size();

int n = y.size();

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

C[i][0] = 0;

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

C[0][j] = 0;

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

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

if (x[i] == y[j]) {

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

movement[i][j] = 2; //Recording the movement to backtrack later on

}

else {

if (C[i - 1][j] >= C[i][j - 1]) {

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

movement[i][j] = 3; //Recording the movement to backtrack later on

}

else {

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

movement[i][j] = 1; //Recording the movement to backtrack later on

}

}

}

}

for (int i = 0; i < C.size(); i++) { //displaying the matrix

for (int j = 0; j < C[i].size(); j++) {

cout << C[i][j] << " \t";

}

cout << endl;

}

cout << "\n\nLCS result : " << C[m - 1][n - 1] << endl; //Displaying the bottom right corner value

vector <char> answer;

int i = m - 1;

int j = n - 1;

bool found = false;

while (!(i <= 0 && j <=0)) { //Finding out the letters for LCS

if (found == true)

break;

switch (movement[i][j]) {

case 2: //2 : left-top corner

if (x[i] == y[j]) {

answer.push\_back(x[i]);

}

i = i - 1;

j = j - 1;

break;

case 3: //3 : up

if (x[i] == y[j]) {

answer.push\_back(x[i]);

}

i = i - 1;

break;

case 1: // 1 : left

if (x[i] == y[j]) {

answer.push\_back(x[i]);

}

j = j - 1;

break;

default:

found = true;

break;

}

}

cout << answer.size()<< " - LCS output : ";

for (auto it = answer.rbegin(); it != answer.rend(); it++) { // Outputting the vector from reverse to get the correct answer

cout << \*it;

}

}

int main()

{

LCS();

}

C Code of Longest Palindromic Subsequence

#include <iostream>

#include <vector>

#include <string>

using namespace std;

void display(std::vector < std::vector<int> > v) {

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

for (int j = 0; j < v[i].size(); j++) { //Printing the row of the d table

std::cout << v[i][j] << "\t";

}

std::cout << "\n";

}

}

int max(int a, int b) {

if (a < b)

return b;

return a;

}

void polindromic(std::string text, std::vector < std::vector <int> > L) {

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

for (int j = 0; j < L[i].size(); j++) {

if (i == j)

L[i][j] = 1;

}

}

int i = 0, j = 1, counter = 2;

while (true) {

if (i == L.size())

break;

if (j == L[i].size())

{

if (counter == L[i].size())

break;

j = counter;

counter++;

i = 0;

}

if (text[i] != text[j])

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

else if ((text[i] == text[j]) && (j == i + 1))

L[i][j] = 2;

else

L[i][j] = L[i + 1][j - 1] + 2;

i++;

j++;

}

display(L);

std::cout << "\t\n\nResult = " << L[0][L[0].size() - 1] << "\n";

i = 0;

j = L[0].size() - 1;

std::cout << "Polindromic = ";

while (true) {

if (L[i + 1][j] == L[i][j - 1] && L[i + 1][j] > L[i][j - 1]) {

i++;

}

else if (L[i + 1][j] < L[i][j - 1]){

j--;

}

else {

i++;

j--;

}

std::cout << text[i];

if (L[i][j] == 0)

break;

}

}

int main()

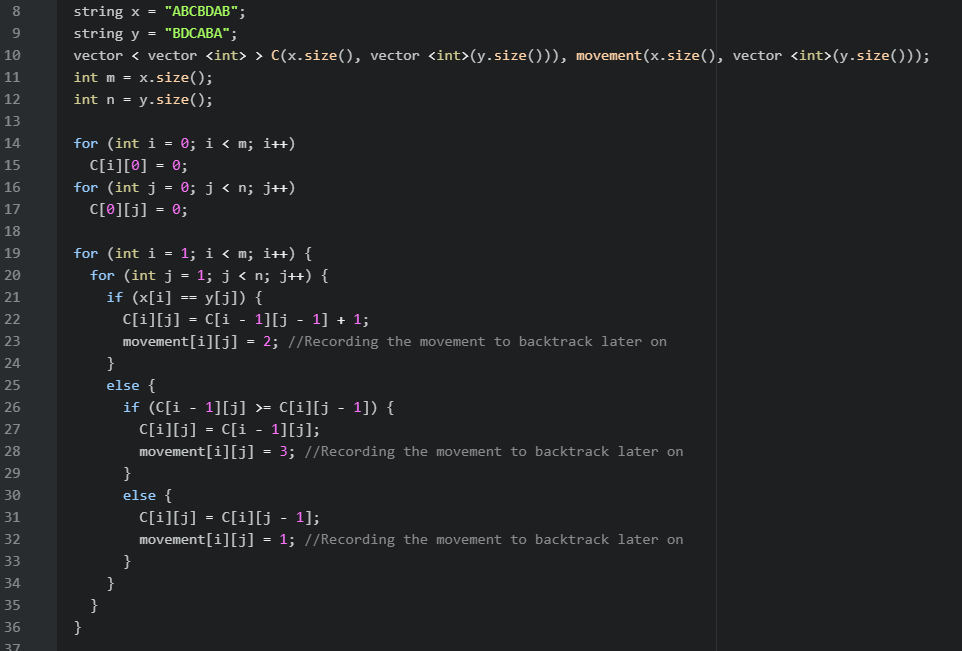
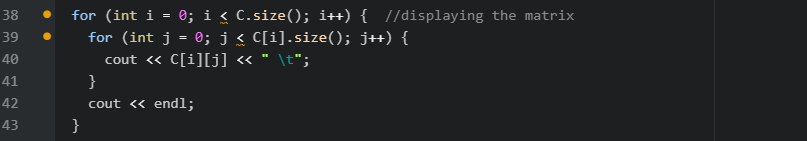
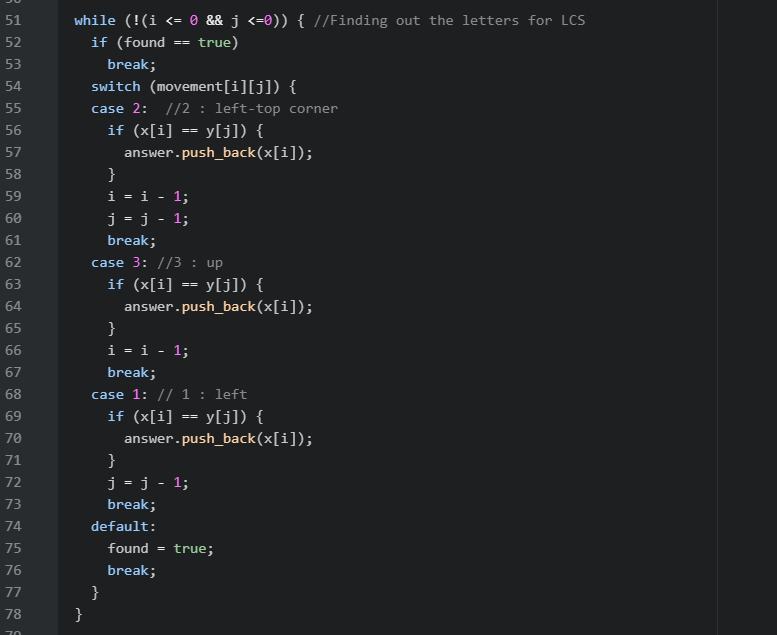
{

string text = "ABCBDAB";

vector < std::vector <int> > L (text.size(), std::vector <int> (text.size()));

polindromic(text, L);

}

Big O notation LCS

**Finally, Big O Notation result for LCS = O(n2)**

O(n2)

O(n2)

O(n2) < O(n+n2) < O(n2 + n2)

O(2n) => O (n) +

O(n2) = O(n + n2)

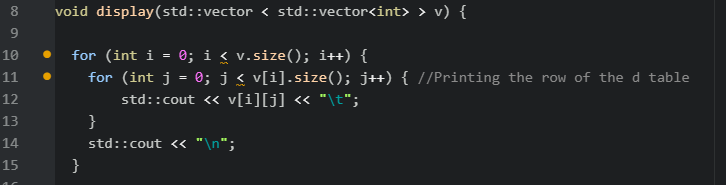
O(n + n2)🡺 O(n2)

O(n2)

O(2n) => O (n)

O(n)

O(n)

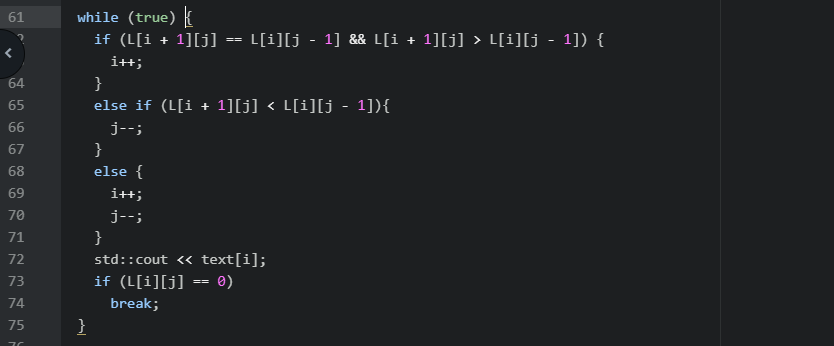
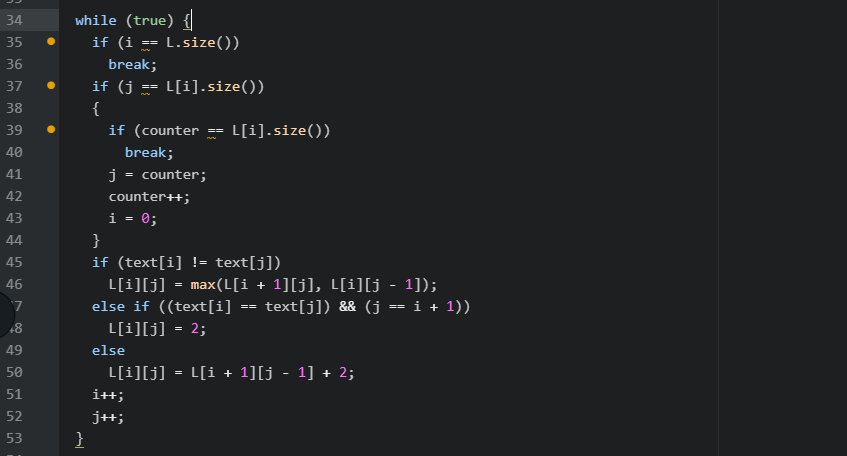
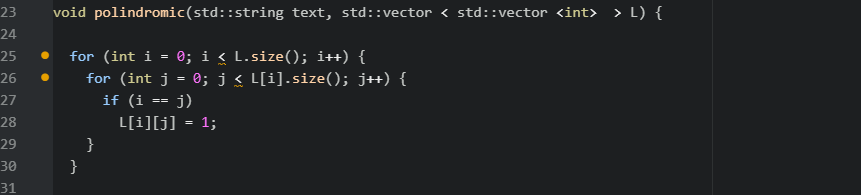
Big O notation LPS

O(n2)

O(n2)

O(n2)

O(n2)



O(4n2) => O (n2)

**Finally, Big O Notation result for LPS = O(n2)**

|  |  |
| --- | --- |
| Output of Program | |
|  | LCS |
|  | LPS |

**[Download the C Code File (until 3 January.)](http://fromsmash.com/ASSIGNMENT-5-Comparing-LCS-and-LPS)**