CSCB039 Алгоритми и програмиране

Домашна работа #1

Владимир Стоянов F108982

IV семестър

Задача: Propose an algorithm that solve the Longest Common Subsequence (LCS) problem that reconstructs the solution from the completed table of LCS Lengths ***C*** and the two input sequences, without the usage of additional table ***d*** with directions of corresponding optimal subproblems used.

Моята имплементация на алгоритъма Longest Common Subsequence (LCS) е имплементиран с помощта на идеите на динамичното оптимиране. Използвал съм метода от долу нагоре (bottom up). Този метод работи с разделянето на големия проблем на по-малки като решенията се записват в таблица.

В случая на този алгоритъм, разделянето става като една по една и двете последователности се проверяват за съвпадения. Ако бъде открито такова, в таблицата със записи се минава по диагонал, а към стойността записана в клетката, отговаряща на това съвпадение, се добавя единица. В противен случай се продължава по реда, а стойността на клетката се определя като се вземе по-голямата измежду дясната и долната. Когато сравненията приключат, за да се открие дължината на LCS, се взима елемента на позиция [0, 0]. Сложността на алгоритъма е O(n\*m), тъй като алгоритъма извършва n пъти m проверки (n и m са дължините на първоначалните последователности).

За откриването на самата последователност е нужен алгоритъм, който да реконструира подпоследователността използвайки вече генерираната таблица и оригиналните последователности. Алгоритъмът работи като започва да прави сравнения на оригиналните последователности. Ако открие съвпадение, увеличава двата индекса с едно и продължава да прави сравнения. В противен случай, проверява в таблицата дали долната или дясната клетка, спрямо текущите стойности на индексите i и j, е с по-голяма стойност и увеличава само този индекс с едно. Тези проверки се повтарят докато и двата индекса (i и j) са по-малки от дължините на оригиналните последователности (n и m).

Програмен код (C++)

#include <iostream>

using std::cout;

using std::cin;

using std::endl;

using std::string;

using std::fill;

using std::max;

// functions declarations

string reconstruct(string text1, string text2, int \*\*dp, int n, int m);

void printMatrix(int \*\*dp, int n, int m);

int lcs(string text1, string text2);

int main()

{

string text1 = "ace";

string text2 = "abcde";

cout << lcs(text1, text2) << endl;

return 0;

}

/\*

Function that reconstructs the LCS.

Takes five arguments:

text1 -> first sequence

text2 -> second sequence

dp -> two dimensional array with the results of the dynamic programming algorithm

n -> first sequence length

m -> second sequence length

\*/

string reconstruct(string text1, string text2, int \*\*dp, int n, int m)

{

int i = 0, j = 0;

string result = "";

while (i < m && j < n)

{

if (text1[i] == text2[j])

{

// Continuing with the sequences

result += text1[i];

++i;

++j;

}

else if (dp[i + 1][j] > dp[i][j + 1])

{

// getting the the cell bellow the current one

++i;

}

else

{

// getting the the cell on the right

++j;

}

}

return result;

}

void printMatrix(int \*\*dp, int n, int m)

{

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

{

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

{

cout << dp[i][j] << " ";

}

cout << endl;

}

}

/\*

Function that finds the length of the LCS.

Takes two arguments:

text1 -> first sequence

text2 -> second sequence

\*/

int lcs(string text1, string text2)

{

int n = text2.size();

int m = text1.size();

int \*\*dp = new int\*[m + 1];

// initializing matrix

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

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

fill(dp[i], dp[i] + n + 1, 0);

}

for (int i = m - 1; i >= 0; --i)

{

for (int j = n - 1; j >= 0; --j)

{

// Choosing which value to increase

if (text1[i] == text2[j])

{

// Adding one to the bottom right cell

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

}

else

{

// Picking the bigger value between the bottom and the right cell

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

}

}

}

printMatrix(dp, n, m);

int result = dp[0][0];

cout << reconstruct(text1, text2, dp, n, m) << endl;

// Cleaning up memory

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

delete[]dp[i];

}

delete[]dp;

return result;

}