[CENG 315 All Sections] Algorithms

Dashboard / My courses / 571 - Computer Engineering / CENG 315 All Sections / November 14 - November 20 / THE4

Description Submission view

THE4

Available from: Friday, November 18, 2022, 11:59 AM
Due date: Saturday, November 19, 2022, 11:59 PM
Requested files: the4.cpp, test.cpp, the4_solution.cpp (Download)
Type of work: Individual work

Problem:

In this exam, you are given two 2D boxes consisting of full and empty cells. The goal is to convert the first box into the second with the minimum cost of operations (the operations are defined below). The boxes are represented as 2D char arrays. In the arguments, both boxes will be defined to include the same number of rows, yet the number of their columns may be different. For instance, Box-1 can be an array of the size of 10x12 whereas Box-2 can be an array of the size of 10x15. In order to represent empty cells, '-' character is used and for the full cells a letter is used. In the figure below, a few input box illustrations are given:

(a)

BOX-1

M
R
V
-

A box defined as an array of 4x1

BOX-2

-	M	V
M	-	M
R	-	R
V	R	-

A box defined as an array of 4x3

Resulting mem

0	3	5	8
3	4	6	8

(b)

BOX-1

-	B	C
B	-	C

A box defined as an array of 2x3

BOX-2

A	-	C	A	B
-	C	C	A	-

A box defined as an array of 2x5

Resulting mem

0	1	2	4	6	7
1	2	2	4	6	7
2	2	3	5	7	6
4	4	4	3	5	6

(c)

BOX-1

A	A	C	B	-
B	C	-	-	A
A	C	-	A	-
-	C	-	-	C

A box defined as an array of 4x5

BOX-2

A	C	B	-	A	B	A
B	A	B	-	C	-	C
B	-	-	A	C	A	B
C	-	B	-	B	A	B

A box defined as an array of 4x7

Resulting mem

0	4	6	9	10	14	17	21
3	3	5	8	9	13	16	20
7	5	7	10	11	10	13	17
8	6	7	10	11	11	14	18
10	8	9	12	12	13	13	17
12	10	10	13	14	15	15	19

(d)

BOX-1

-	Y	W	Y	-	-
-	Z	Y	Y	-	W
W	Y	Z	Y	-	-
W	W	-	Y	-	Y
-	Z	-	-	Z	Z

A box defined as an array of 5x6

BOX-2

Z	W	-
W	Y	-
Y	Z	Y
-	Z	-
-	-	-

A box defined as an array of 5x3

Resulting mem

0	3	7	8
2	5	9	10
7	8	11	12
10	10	10	11
14	14	13	14
15	15	14	15
18	18	17	18

(e)

BOX-1

A	B	C	A	B	C
B	C	C	A	-	A

A box defined as an array of 2x6

BOX-2

A	B	B	A
B	A	-	C

A box defined as an array of 2x4

Resulting mem

0	2	4	5	7
2	0	2	3	5
4	2	1	2	4
6	4	3	4	3
8	6	5	6	5
9	7	6	5	6
11	9	8	7	7

(f)

BOX-1

-	M	V	R
M	-	R	-
R	M	M	-
V	M	-	R

A box defined as an array of 4x4

BOX-2

M	-	-	V
V	R	R	M
-	-	-	-
-	M	-	R

A box defined as an array of 4x4

Resulting mem

0	2	4	5	8
3	5	6	7	8
6	8	9	10	11
9	10	12	13	13
11	12	14	15	15

Your task is to convert the first box into the second box by using some operations resulting in the minimum cost. The conversion rules and operations are defined as follows:

- You should compare the boxes column by column. Each conversion operation is column-wise.
- A column could be deleted completely. The **deletion operation** costs as much as the number of full cells in the column. For instance; if the column consists of 5 cells where 3 of them full and 2 of them are empty, then deleting that column costs 3 units.
- For a column of Box-2, a new corresponding column could be inserted into Box-1 at any location (between two columns or as the initial column or as the final column). The **insertion operation** costs as much as the number of full cells inside the new column. For instance; if the newly inserted column consists of 5 cells where 3 of them full and 2 of them are empty, then inserting that column costs 3 units.
- A column could be converted into a new column by reordering its cells. For intance, if a column consists of 5 cells including ['X', 'A', '-', 'B', '-'], it can be reordered as ['A', '-', '-', 'B', 'X']. The **reordering operation** costs as much as the number of cells whose locations are changed. For the example given, since the locations of the cells including 'A', 'X' and '-' changed only, it costs 3 units.
- A column could be converted into a new column by replacing its cells with some other cells. For the **replacement operation**, if a full cell is replaced with some other full cell, then it costs 1 unit. However, if an empty cell is replaced with a full cell, or vice versa, then it costs 2 units. For instance, if a column consists of 5 cells including ['X', 'A', '-', 'B', '-'], its cells can be replaced as ['X', 'C', 'D', '-', '-'], it costs <change from 'A' to 'C'> + <change from '-' to 'D'> + <change from 'B' to '-'> = 1 + 2 + 2 = 5 units.
- Each operation is independent from each other. At each transition, apply only one of them.
- **HINT:** You should implement the dynamic programming column-wise. That is, for each column of Box-2, consider a corresponding column inside Box-1 which has been obtained by the operations above. The way of how to apply memoization is explained in the following parts.

Example IO:

1) Given boxes in (a) of the above Figure:

- return value (i.e. min cost) is 8.
- Since this is the first example, let's explain all the cells of mem array:

mem[0][0] : *Conversion of no columns of box1 to no columns of box2*
No operation
=> costs 0

mem[0][1] : *Conversion of no columns of box1 to first column of box2*
Apply insertion operation to obtain the first column of box2
=> costs 3

mem[0][2] : *Conversion of no columns of box1 to first 2 columns of box2*
Apply insertion operation for both of the first two columns of box2
=> costs 3 +2 = 5

mem[0][3] : *Conversion of no columns of box1 to first 3 columns of box2*
Apply insertion operation for each of the 3 columns of box2
=> costs 3 + 2 + 3 = 8

mem[1][0] : *Conversion of first column of box1 to no columns of box2*
Apply deletion operation on the initial column of box1
=> costs 3

mem[1][1] : *Conversion of first column of box1 to first column of box2*
Apply reordering operation to change the first column of box1 to the first column of box2
=> costs 4

mem[1][2] : *Conversion of first column of box1 to first 2 columns of box2*
Apply reordering operation to change the first column of box1 to the first column of box2 and
Apply insertion operation to obtain the second column of box2
=> costs 4 + 2 = 6

mem[1][3] : *Conversion of first column of box1 to first 3 columns of box2*
Apply insertion operation to obtain the first column of box2 and
Apply insertion operation to obtain the second column of box2 and
Apply reordering operation to change the first column of box1 to the third column of box2
=> costs 3 +2 + 3 = 8

2) **Given boxes in (b) of the above Figure:**

- return value (i.e. min cost) is 6.
- at dynamic programming, final mem array is given its right side.

3) Given boxes in (c) of the above Figure:

- return value (i.e. min cost) is 19.
- at dynamic programming, final mem array is given its right side.

4) Given boxes in (d) of the above Figure:

- return value (i.e. min cost) is 18.
- at dynamic programming, final mem array is given its right side.

5) Given boxes in (e) of the above Figure:

- return value (i.e. min cost) is 7.
- at dynamic programming, final mem array is given its right side.

6) Given boxes in (f) of the above Figure:

- return value (i.e. min cost) is 15.
- at dynamic programming, final mem array is given its right side.

Implementation:

You will implement only one function for solution of that problem:

- Dynamic programming in ***dp_sln()***

The function is expected to **return** the answer to the given problem which is **the minimum cost of operations.** Return **only** the min cost value and nothing more.

The ***char**& box1*** and ***char**& box2*** variables are the parameters which pass the input 2D array of boxes to your functions. **Do not modify those arrays!** The format of boxes will be as stated in the problem definition above.

The ***int nrow***, ***int ncol1*** and ***int ncol2*** variables are the parameters which passes the number of rows of both boxes, number of columns of *box1* and number of columns of *box2,* repectively, to your function.

You should use ***int**& mem*** variable (i.e. array), which is the last parameter at definition of the function, as **the array of memoized values**. For *dp_sln()* function, final values in the *mem* variable will be considered for grading. Note that it is a 2D array. It is defined as **the size of (ncol1+1) x (ncol2+1**) such that its rows correspond to columns of *box1* and its columns correspond to columns of *box2*. **That is, the *mem[i][j]* will be used to indicate the TOTAL COST of matching of THE FIRST i columns of box1 with THE FIRST j columns of box2**. Thus *mem[0][0]* indicates there is no matching columns in box1 and box2! While testing and grading, all the cells of *mem* array will be initialized to -1's. So, while implementing your function, **you can assume that *mem* is an array of array of -1's. Do not return that variable/array.**

The *dp_sln()* function should be implemented with bottom-up (iterative) approach.

Implement the function in most efficient way.

Constraints:

- Maximum number of rows and columns of boxes will be **100**.

Evaluation:

- After your exam, black box evaluation will be carried out. You will get full points if

  1. your functionsreturn the correct min cost
  2. and you fill the ***mem*** array correctly, as stated.
  3. you did not change the input arrays (the array of boxes).

Specifications:

- There is **1 task** to be solved in **12 hours** in this take home exam.
- You will implement your solution in **the4.cpp** file.
- Do **not** change the first line of **the4.cpp**, which is **#include "the4.h"**
- *<iostream>, <climits>* , *<cmath>* , *<cstdlib>* are included in "the4.h" for your convenience.
- Do **not** change the arguments and return **type** of the function **dp_sln()** in the file **the4.cpp.** (You should change return **value**, on the other hand.)
- Do **not** include any other library or write include anywhere in your **the4.cpp** file (not even in comments).
- Do **not** write any helper method.

Compilation:

- You are given **test.cpp** file to **test** your work on **ODTÜClass** or your **locale**. You can and you are encouraged to modify this file to add different test cases.
- If you want to **test** your work and see your outputs you can **compile and run** your work on your locale as:

```
>g++ test.cpp the4.cpp -Wall -std=c++11 -o test

> ./test
```

- You can test your **the4.cpp** on virtual lab environment. If you click **run**, your function will be compiled and executed with **test.cpp**. If you click **evaluate**, you will get a feedback for your current work and your work will be **temporarily** graded for **limited** number of inputs.
- The grade you see in lab is **not** your final grade, your code will be re-evaluated with **completely different** inputs after the exam.

The system has the following limits:

- a maximum execution time of 32 seconds
- a 192 MB maximum memory limit
- an execution file size of 1M.
- Solutions with longer running times will not be graded.
- If you are sure that your solution works in the expected complexity constrains but your evaluation fails due to limits in the lab environment, the constant factors may be the problem.

```
int dp_sln(char**& arr1, char**& arr2, int nrow, int ncol1, int ncol2, int**& mem);
```

Requested files

the4.cpp

```
#include "the4.h"


int dp_sln(char**& arr1, char**& arr2, int nrow, int ncol1, int ncol2, int**& mem){ //dynamic programming

    //your code here

    return 0; // this is a dummy return value. YOU SHOULD CHANGE THIS!
}


```

test.cpp

```cpp
// this file is for you for testing purposes, it won't be included in evaluation.

#include <iostream>
#include <random>
#include <ctime>
#include <cstdlib>
#include <algorithm>
#include <vector>
#include "the4.h"

char getRandomChar(){
    char r = rand() % 5 + 65;
    return r;
}


void randomArray(char**& box1, char**& box2, int nrow, int ncol1, int ncol2)
{
    box1 = new char* [nrow];
    box2 = new char* [nrow];
    std::vector<char> column;

    for (int i = 0; i < nrow; i++) {
       box1[i] = new char [ncol1];
       box2[i] = new char [ncol2];
    }

    for (int i = 0; i < ncol1; i++)
    {
        int nfull = rand() % nrow + 1;
        for (int j = 0; j < nfull; j++) {
            char r = getRandomChar();
            column.push_back(r);
        }
        for (int j = nfull; j < nrow; j++) {
            column.push_back('-');
        }
        std::random_shuffle(column.begin(), column.end());
        for (int j = 0; j < nrow; j++)
            box1[j][i] = column[j];
        column.clear();
    }

    for (int i = 0; i < ncol2; i++)
    {
        int nfull = rand() % nrow + 1;
        for (int j = 0; j < nfull; j++) {
            char r = getRandomChar();
            column.push_back(r);
        }
        for (int j = nfull; j < nrow; j++) {
            column.push_back('-');
        }
        std::random_shuffle(column.begin(), column.end());
        for (int j = 0; j < nrow; j++)
            box2[j][i] = column[j];
    }
}


void printArrayInLine(char** arr, int nrow, int ncol){
    std::cout << "[ ";
    for(int i = 0; i < nrow; i++){
        std::cout << "[";
        for (int j = 0; j < ncol; j++) {
            std::cout << arr[i][j];
            if (j == ncol - 1)
                std::cout << "]";
            else
                std::cout << ", ";
        }
        if (i == nrow - 1)
            std::cout << " ]" << std::endl;
        else
            std::cout << ",\n";
    }
}


void printMemInLine(int** arr, int nrow, int ncol){
    std::cout << "[ ";
    for(int i = 0; i < nrow; i++){
        std::cout << "[";
        for (int j = 0; j < ncol; j++) {
            std::cout << arr[i][j];
            if (j == ncol - 1)
                std::cout << "]";
            else
                std::cout << ", ";
        }
        if (i == nrow - 1)
            std::cout << " ]" << std::endl;
        else
            std::cout << ",\n";
    }
}


void fillArray(char**& box1, char**& box2, int nrow, int ncol1, int ncol2)
{

    box1 = new char* [nrow];
    box2 = new char* [nrow];

    for (int i = 0; i < nrow; i++) {
       box1[i] = new char [ncol1];
       box2[i] = new char [ncol2];
    }

    // "DO NOT FORGET TO CHANGE THE nrow, ncol1, ncol2 VALUES AT THE BEGINNING OF test() METHOD!!!!!!"
    // EXAMPLE (a)

    box1[0][0] = 'M';
    box1[1][0] = 'R';
    box1[2][0] = 'V';
    box1[3][0] = '-';

    box2[0][0] = '-'; box2[0][1] = 'M'; box2[0][2] = 'V';
    box2[1][0] = 'M'; box2[1][1] = '-'; box2[1][2] = 'M';
    box2[2][0] = 'R'; box2[2][1] = '-'; box2[2][2] = 'R';
    box2[3][0] = 'V'; box2[3][1] = 'R'; box2[3][2] = '-';


    // "DO NOT FORGET TO CHANGE THE nrow, ncol1, ncol2 VALUES AT THE BEGINNING OF test() METHOD!!!!!!"
    // EXAMPLE (b)
    /*
    box1[0][0] = '-'; box1[0][1] = 'B'; box1[0][2] = 'C';
    box1[1][0] = 'B'; box1[1][1] = '-'; box1[1][2] = 'C';

    box2[0][0] = 'A'; box2[0][1] = '-'; box2[0][2] = 'C'; box2[0][3] = 'A'; box2[0][4] = 'B';
    box2[1][0] = '-'; box2[1][1] = 'C'; box2[1][2] = 'C'; box2[1][3] = 'A'; box2[1][4] = '-';
    */

    // "DO NOT FORGET TO CHANGE THE nrow, ncol1, ncol2 VALUES AT THE BEGINNING OF test() METHOD!!!!!!"
    // EXAMPLE (c)
    /*
    box1[0][0] = 'A'; box1[0][1] = 'A'; box1[0][2] = 'C'; box1[0][3] = 'B'; box1[0][4] = '-';
    box1[1][0] = 'B'; box1[1][1] = 'C'; box1[1][2] = '-'; box1[1][3] = '-'; box1[1][4] = 'A';
    box1[2][0] = 'A'; box1[2][1] = 'C'; box1[2][2] = '-'; box1[2][3] = 'A'; box1[2][4] = '-';
    box1[3][0] = '-'; box1[3][1] = 'C'; box1[3][2] = '-'; box1[3][3] = '-'; box1[3][4] = 'C';

    box2[0][0] = 'A'; box2[0][1] = 'C'; box2[0][2] = 'B'; box2[0][3] = '-'; box2[0][4] = 'A'; box2[0][5] = 'B'; box2[0][6] =
    box2[1][0] = 'B'; box2[1][1] = 'A'; box2[1][2] = 'B'; box2[1][3] = '-'; box2[1][4] = 'C'; box2[1][5] = '-'; box2[1][6] =
    box2[2][0] = 'B'; box2[2][1] = '-'; box2[2][2] = '-'; box2[2][3] = 'A'; box2[2][4] = 'C'; box2[2][5] = 'A'; box2[2][6] =
    box2[3][0] = 'C'; box2[3][1] = '-'; box2[3][2] = 'B'; box2[3][3] = '-'; box2[3][4] = 'B'; box2[3][5] = 'A'; box2[3][6] =
    */

    // "DO NOT FORGET TO CHANGE THE nrow, ncol1, ncol2 VALUES AT THE BEGINNING OF test() METHOD!!!!!!"
    // EXAMPLE (d)
    /*
    box1[0][0] = '-'; box1[0][1] = 'Y'; box1[0][2] = 'W'; box1[0][3] = 'Y'; box1[0][4] = '-'; box1[0][5] = '-';
    box1[1][0] = '-'; box1[1][1] = 'Z'; box1[1][2] = 'Y'; box1[1][3] = 'Y'; box1[1][4] = '-'; box1[1][5] = 'W';
    box1[2][0] = 'W'; box1[2][1] = 'Y'; box1[2][2] = 'Z'; box1[2][3] = 'Y'; box1[2][4] = '-'; box1[2][5] = '-';
    box1[3][0] = 'W'; box1[3][1] = 'W'; box1[3][2] = '-'; box1[3][3] = 'Y'; box1[3][4] = '-'; box1[3][5] = 'Y';
    box1[4][0] = '-'; box1[4][1] = 'Z'; box1[4][2] = '-'; box1[4][3] = '-'; box1[4][4] = 'Z'; box1[4][5] = 'Z';

    box2[0][0] = 'Z'; box2[0][1] = 'W'; box2[0][2] = '-';
    box2[1][0] = 'W'; box2[1][1] = 'Y'; box2[1][2] = '-';
    box2[2][0] = 'Y'; box2[2][1] = 'Z'; box2[2][2] = 'Y';
    box2[3][0] = '-'; box2[3][1] = 'Z'; box2[3][2] = '-';
    box2[4][0] = '-'; box2[4][1] = '-'; box2[4][2] = '-';
    */

    // "DO NOT FORGET TO CHANGE THE nrow, ncol1, ncol2 VALUES AT THE BEGINNING OF test() METHOD!!!!!!"
    // EXAMPLE (e)
    /*
    box1[0][0] = 'A'; box1[0][1] = 'B'; box1[0][2] = 'C'; box1[0][3] = 'A'; box1[0][4] = 'B'; box1[0][5] = 'C';
    box1[1][0] = 'B'; box1[1][1] = 'C'; box1[1][2] = 'C'; box1[1][3] = 'A'; box1[1][4] = '-'; box1[1][5] = 'A';

    box2[0][0] = 'A'; box2[0][1] = 'B'; box2[0][2] = 'B'; box2[0][3] = 'A';
    box2[1][0] = 'B'; box2[1][1] = 'A'; box2[1][2] = '-'; box2[1][3] = 'C';
    */

    // "DO NOT FORGET TO CHANGE THE nrow, ncol1, ncol2 VALUES AT THE BEGINNING OF test() METHOD!!!!!!"
    // EXAMPLE (f)
    /*
    box1[0][0] = '-'; box1[0][1] = 'M'; box1[0][2] = 'V'; box1[0][3] = 'R';
    box1[1][0] = 'M'; box1[1][1] = '-'; box1[1][2] = 'R'; box1[1][3] = '-';
    box1[2][0] = 'R'; box1[2][1] = 'M'; box1[2][2] = 'M'; box1[2][3] = '-';
    box1[3][0] = 'V'; box1[3][1] = 'M'; box1[3][2] = '-'; box1[3][3] = 'R';
    box2[0][0] = 'M'; box2[0][1] = '-'; box2[0][2] = '-'; box2[0][3] = 'V';
    box2[1][0] = 'V'; box2[1][1] = 'R'; box2[1][2] = 'R'; box2[1][3] = 'M';
    box2[2][0] = '-'; box2[2][1] = '-'; box2[2][2] = '-'; box2[2][3] = '-';
    box2[3][0] = '-'; box2[3][1] = 'M'; box2[3][2] = '-'; box2[3][3] = 'R';
    */

}


void test(){
    clock_t begin, end;
    double duration;
    int min_cost_dp;


    int nrow = 4;      // max 100
    int ncol1 = 1;     // max 100
    int ncol2 = 3;     // max 100
    char** box1, ** box2;
    //randomArray(box1, box2, nrow, ncol1, ncol2);
    fillArray(box1, box2, nrow, ncol1, ncol2);
    std::cout << "BOX-1:" << std::endl;
    printArrayInLine(box1, nrow, ncol1);
    std::cout << "\nBOX-2:" << std::endl;
    printArrayInLine(box2, nrow, ncol2);
    std::cout << "\n\n";

    int** mem = new int*[ncol1+1];

    for(int i = 0; i <= ncol1; i++){
        mem[i] = new int [ncol2+1];
        for (int j = 0; j <= ncol2; j++)
            mem[i][j] = -1;
    }



    std::cout << "__________________DYNAMIC PROGRAMMING:______________" << std::endl;

    for(int i = 0; i <= ncol1; i++){
        for (int j = 0; j <= ncol2; j++)
            mem[i][j] = -1;
    }


    if ((begin = clock() ) ==-1)
        std::cerr << "clock error" << std::endl;

    min_cost_dp = dp_sln(box1, box2, nrow, ncol1, ncol2, mem);

    if ((end = clock() ) ==-1)
        std::cerr << "clock error" << std::endl;

    duration = ((double) end - begin) / CLOCKS_PER_SEC;
    std::cout << "Duration: " << duration << " seconds." << std::endl;

    std::cout << "Min cost: " << min_cost_dp << std::endl;
    std::cout << "Final mem: " << std::endl;
    printMemInLine(mem, ncol1+1, ncol2+1);

    std::cout << "----------------------------------------------------";
    std::cout << "\n" << std::endl;

}

int main()
{
    srandom(time(0));
    test();
    return 0;
}

```

the4_solution.cpp

```
#include "sol4.h"



int recursive_sln(int i, int j, char**& arr1, char**& arr2, int nrow, int ncol1, int ncol2, int &number_of_calls){ //direct r
    number_of_calls+=1;

    if (i == 0) {
        if (j == 0)
            return 0;   // never goes here
        else {
            int num_of_insertions = 0;
            // count the number of cells in arr2
            for (int n = 0; n < j; n++) {
                for (int m = 0; m < nrow; m++) {
                    if (arr2[m][n] == '-')
                        continue;
                    else
                        num_of_insertions ++;  // num of insertions
                }
            }
            return num_of_insertions;
        }
    }
    else if (j == 0) {
        int num_of_deletions = 0;
        // count the number of cells in arr1
        for (int n = 0; n < i; n++) {
            for (int m = 0; m < nrow; m++) {
                if (arr1[m][n] == '-')
                    continue;
                else
                    num_of_deletions ++;  // num of deletions
            }
        }
        return num_of_deletions;
    }
    else {

        // DELETION
         int num_of_deletions = 0;
        // count the number of cells in arr1
        for (int m = 0; m < nrow; m++) {
            if (arr1[m][i-1] == '-')
                continue;
            else
                num_of_deletions ++;
        }

        int x = recursive_sln(i-1, j, arr1, arr2, nrow, ncol1, ncol2, number_of_calls);
        int cost = x + num_of_deletions;  // cost of deletions

        // INSERTION
        int num_of_insertions = 0;
        // count the number of cells in arr2
        for (int m = 0; m < nrow; m++) {
            if (arr2[m][j-1] == '-')
                    continue;
            else
                num_of_insertions ++;
        }

        int y = recursive_sln(i, j-1, arr1, arr2, nrow, ncol1, ncol2, number_of_calls);
        if (cost > y + num_of_insertions)
            cost = y + num_of_insertions; // cost of insertions

        // REPLACEMENT
        int cost_of_replacements = 0;
        // check for the same items in arr1 and arr2
        for (int m = 0; m < nrow; m++) {
            if (arr1[m][i-1] == arr2[m][j-1])
                continue;
            else if (arr1[m][i-1] == '-' || arr2[m][j-1] == '-')
                cost_of_replacements += 2;
            else
                cost_of_replacements ++;
        }
        int z = recursive_sln(i-1, j-1, arr1, arr2, nrow, ncol1, ncol2, number_of_calls);
        if (cost > z + cost_of_replacements)
            cost = z + cost_of_replacements;

        // REORDERING
        int cost_of_reordering = 0;
        std::string order1 = "";
        std::string order2 = "";
        for (int m = 0; m < nrow; m++) {
            order1 += arr1[m][i-1];
            order2 += arr2[m][j-1];
        }
        bool equivalent = true;
        for (int m = 0; m < nrow; m++) {
            std::size_t found = order2.find(order1[m]);
            if (found!=std::string::npos)
                order2 = order2.substr(0, found) + order2.substr(found+1);
            else {
                equivalent = false;
                break;
            }
        }

        if (equivalent) {
            for (int m = 0; m < nrow; m++) {
                if (arr1[m][i-1] == arr2[m][j-1])
                    continue;
                else
                    cost_of_reordering += 1;
            }
            if (cost > z + cost_of_reordering)
                cost = z + cost_of_reordering;
        }

        return cost;
    }

}



int memoization_sln(int i, int j, char**& arr1, char**& arr2, int nrow, int ncol1, int ncol2, int**& mem){ //memoization

    // mem is (ncol1+1) x (ncol2+1)

    // initialize trivial parts of mem
    if (i == 0) {
        if (j == 0)
            mem[i][j] = 0;  // never goes here
        else {
            mem[0][j] = 0;
            // count the number of cells in arr2
            for (int n = 0; n < j; n++) {
                for (int m = 0; m < nrow; m++) {
                    if (arr2[m][n] == '-')
                        continue;
                    else
                        mem[0][j] ++;  // num of insertions
                }
            }
            mem[0][0] = 0;
        }
    }
    else if (j == 0) {
        mem[i][0] = 0;
        // count the number of cells in arr1
        for (int n = 0; n < i; n++) {
            for (int m = 0; m < nrow; m++) {
                if (arr1[m][n] == '-')
                    continue;
                else
                    mem[i][0] ++;  // num of deletions
            }
        }
    }

    // for the nontrivial parts of mem
    else {
        // DELETION
        int num_of_deletions = 0;
        // count the number of cells in arr1
        for (int m = 0; m < nrow; m++) {
            if (arr1[m][i-1] == '-')
                continue;
            else
                num_of_deletions ++;
        }

        if (mem[i-1][j] == -1)
            memoization_sln(i-1, j, arr1, arr2, nrow, ncol1, ncol2, mem);
        int cost = mem[i-1][j] + num_of_deletions;  // cost of deletions

        // INSERTION
        int num_of_insertions = 0;
        // count the number of cells in arr2
        for (int m = 0; m < nrow; m++) {
            if (arr2[m][j-1] == '-')
                continue;
            else
                num_of_insertions ++;
        }

        if (mem[i][j-1] == -1)
            memoization_sln(i, j-1, arr1, arr2, nrow, ncol1, ncol2, mem);
            memoization_sln(i, j-1, arr1, arr2, nrow, ncol1, ncol2, mem);
        if (cost > mem[i][j-1] + num_of_insertions)
            cost = mem[i][j-1] + num_of_insertions; // cost of insertions

        // REPLACEMENT
        int cost_of_replacements = 0;
        // check for the same items in arr1 and arr2
        for (int m = 0; m < nrow; m++) {
            if (arr1[m][i-1] == arr2[m][j-1])
                continue;
            else if (arr1[m][i-1] == '-' || arr2[m][j-1] == '-')
                cost_of_replacements += 2;
            else
                cost_of_replacements ++;
        }
        if (mem[i-1][j-1] == -1)
            memoization_sln(i-1, j-1, arr1, arr2, nrow, ncol1, ncol2, mem);
        if (cost > mem[i-1][j-1] + cost_of_replacements)
            cost = mem[i-1][j-1] + cost_of_replacements;

        // REORDERING
        int cost_of_reordering = 0;
        std::string order1 = "";
        std::string order2 = "";
        for (int m = 0; m < nrow; m++) {
            order1 += arr1[m][i-1];
            order2 += arr2[m][j-1];
        }
        bool equivalent = true;
        for (int m = 0; m < nrow; m++) {
            std::size_t found = order2.find(order1[m]);
            if (found!=std::string::npos)
                order2 = order2.substr(0, found) + order2.substr(found+1);
            else {
                equivalent = false;
                break;
            }
        }

        if (equivalent) {
            for (int m = 0; m < nrow; m++) {
                if (arr1[m][i-1] == arr2[m][j-1])
                    continue;
                else
                    cost_of_reordering += 1;
            }
            if (cost > mem[i-1][j-1] + cost_of_reordering)
                cost = mem[i-1][j-1] + cost_of_reordering;
        }

        mem[i][j] = cost;
    }

    return mem[i][j];
}



int dp_sln(char**& arr1, char**& arr2, int nrow, int ncol1, int ncol2, int**& mem){ //memoization

    // mem is (ncol1+1) x (ncol2+1)

    // initialize trivial parts of mem
    mem[0][0] = 0;
    for (int i = 1; i <= ncol1; i++) {
        mem[i][0] = 0;
        // count the number of cells in arr1
        for (int n = 0; n < i; n++) {
            for (int m = 0; m < nrow; m++) {
                if (arr1[m][n] == '-')
                    continue;
                else
                    mem[i][0] ++;  // num of deletions
            }
        }
    }
    for (int j = 1; j <= ncol2; j++) {
        mem[0][j] = 0;
        // count the number of cells in arr2
        for (int n = 0; n < j; n++) {
            for (int m = 0; m < nrow; m++) {
                if (arr2[m][n] == '-')
                    continue;
                else
                    mem[0][j] ++;  // num of insertions
            }
        }
    }

    // now start dynamic programming
    for (int i = 1; i <= ncol1; i++)
```

```
        for (int j = 1; j <= ncol2; j++) {

            // DELETION
            int num_of_deletions = 0;
            // count the number of cells in arr1
            for (int m = 0; m < nrow; m++) {
                if (arr1[m][i-1] == '-')
                    continue;
                else
                    num_of_deletions ++;
            }

            int cost = mem[i-1][j] + num_of_deletions;  // cost of deletions

            // INSERTION
            int num_of_insertions = 0;
            // count the number of cells in arr2
            for (int m = 0; m < nrow; m++) {
                if (arr2[m][j-1] == '-')
                    continue;
                else
                    num_of_insertions ++;
            }

            if (cost > mem[i][j-1] + num_of_insertions)
                cost = mem[i][j-1] + num_of_insertions; // cost of insertions

             // REPLACEMENT
            int cost_of_replacements = 0;
            // check for the same items in arr1 and arr2
            for (int m = 0; m < nrow; m++) {
                if (arr1[m][i-1] == arr2[m][j-1])
                    continue;
                else if (arr1[m][i-1] == '-' || arr2[m][j-1] == '-')
                    cost_of_replacements += 2;
                else
                    cost_of_replacements ++;
            }
            if (cost > mem[i-1][j-1] + cost_of_replacements)
                cost = mem[i-1][j-1] + cost_of_replacements;

            // REORDERING
            int cost_of_reordering = 0;
            std::string order1 = "";
            std::string order2 = "";
            for (int m = 0; m < nrow; m++) {
                order1 += arr1[m][i-1];
                order2 += arr2[m][j-1];
            }
            bool equivalent = true;
            for (int m = 0; m < nrow; m++) {
                std::size_t found = order2.find(order1[m]);
                if (found!=std::string::npos)
                    order2 = order2.substr(0, found) + order2.substr(found+1);
                else {
                    equivalent = false;
                    break;
                }
            }

            if (equivalent) {
                for (int m = 0; m < nrow; m++) {
                    if (arr1[m][i-1] == arr2[m][j-1])
                        continue;
                    else
                        cost_of_reordering += 1;
                }
                if (cost > mem[i-1][j-1] + cost_of_reordering)
                    cost = mem[i-1][j-1] + cost_of_reordering;
            }

            mem[i][j] = cost;
        }

    return mem[ncol1][ncol2];
}




```

VPL

ODTÜClass Archive
2021-2022 Summer
2021-2022 Spring
2021-2022 Fall
2020-2021 Summer
2020-2021 Spring
2020-2021 Fall
Class Archive
ODTÜClass 2021-2022 Summer School

Get the mobile app