

Dear ODTÜClass Users,  
You can access/login ODTUClass 2022-2023 Spring semester at <https://odtuclass2022s.metu.edu.tr/>  
If you want to access the page directly by typing **odtuclass.metu.edu.tr**, please clear the cache in your browser(s).  
Best regards,  
ODTÜClass Support Team

## [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:

<div>(a)</div> <div><div>BOX-1</div><div><div>M</div><div>R</div><div>V</div><div>-</div></div><div>A box defined as an array of 4x1</div></div> <div><div>BOX-2</div><div><div><div>-</div><div>M</div><div>V</div></div><div><div>M</div><div>-</div><div>M</div></div><div><div>R</div><div>-</div><div>R</div></div><div><div>V</div><div>R</div><div>-</div></div></div><div>A box defined as an array of 4x3</div></div> <div><div>Resulting mem</div><div><div>0</div><div>3</div><div>5</div><div>8</div></div><div><div>3</div><div>4</div><div>6</div><div>8</div></div></div>
<div>(b)</div> <div><div>BOX-1</div><div><div><div>-</div><div>B</div><div>C</div></div><div><div>B</div><div>-</div><div>C</div></div></div><div>A box defined as an array of 2x3</div></div> <div><div>BOX-2</div><div><div><div>A</div><div>-</div><div>C</div><div>A</div><div>B</div></div><div><div>-</div><div>C</div><div>C</div><div>A</div><div>-</div></div></div><div>A box defined as an array of 2x5</div></div> <div><div>Resulting mem</div><div><div>0</div><div>1</div><div>2</div><div>4</div><div>6</div><div>7</div></div><div><div>1</div><div>2</div><div>2</div><div>4</div><div>6</div><div>7</div></div><div><div>2</div><div>2</div><div>3</div><div>5</div><div>7</div><div>6</div></div><div><div>4</div><div>4</div><div>4</div><div>3</div><div>5</div><div>6</div></div></div>
<div>(c)</div> <div><div>BOX-1</div><div><div><div>A</div><div>A</div><div>C</div><div>B</div><div>-</div></div><div><div>B</div><div>C</div><div>-</div><div>-</div><div>A</div></div><div><div>A</div><div>C</div><div>-</div><div>A</div><div>-</div></div><div><div>-</div><div>C</div><div>-</div><div>-</div><div>C</div></div></div><div>A box defined as an array of 4x5</div></div> <div><div>BOX-2</div><div><div><div>A</div><div>C</div><div>B</div><div>-</div><div>A</div><div>B</div><div>A</div></div><div><div>B</div><div>A</div><div>B</div><div>-</div><div>C</div><div>-</div><div>C</div></div><div><div>B</div><div>-</div><div>-</div><div>A</div><div>C</div><div>A</div><div>B</div></div><div><div>C</div><div>-</div><div>B</div><div>-</div><div>B</div><div>A</div><div>B</div></div></div><div>A box defined as an array of 4x7</div></div> <div><div>Resulting mem</div><div><div>0</div><div>4</div><div>6</div><div>9</div><div>10</div><div>14</div><div>17</div><div>21</div></div><div><div>3</div><div>3</div><div>5</div><div>8</div><div>9</div><div>13</div><div>16</div><div>20</div></div><div><div>7</div><div>5</div><div>7</div><div>10</div><div>11</div><div>10</div><div>13</div><div>17</div></div><div><div>8</div><div>6</div><div>7</div><div>10</div><div>11</div><div>11</div><div>14</div><div>18</div></div><div><div>10</div><div>8</div><div>9</div><div>12</div><div>12</div><div>13</div><div>13</div><div>17</div></div><div><div>12</div><div>10</div><div>10</div><div>13</div><div>14</div><div>15</div><div>15</div><div>19</div></div></div>
<div>(d)</div> <div><div>BOX-1</div><div><div><div>-</div><div>Y</div><div>W</div><div>Y</div><div>-</div><div>-</div></div><div><div>-</div><div>Z</div><div>Y</div><div>Y</div><div>-</div><div>W</div></div><div><div>W</div><div>Y</div><div>Z</div><div>Y</div><div>-</div><div>-</div></div><div><div>W</div><div>W</div><div>-</div><div>Y</div><div>-</div><div>Y</div></div><div><div>-</div><div>Z</div><div>-</div><div>-</div><div>Z</div><div>Z</div></div></div><div>A box defined as an array of 5x6</div></div> <div><div>BOX-2</div><div><div><div>Z</div><div>W</div><div>-</div></div><div><div>W</div><div>Y</div><div>-</div></div><div><div>Y</div><div>Z</div><div>Y</div></div><div><div>-</div><div>Z</div><div>-</div></div><div><div>-</div><div>-</div><div>-</div></div></div><div>A box defined as an array of 5x3</div></div> <div><div>Resulting mem</div><div><div>0</div><div>3</div><div>7</div><div>8</div></div><div><div>2</div><div>5</div><div>9</div><div>10</div></div><div><div>7</div><div>8</div><div>11</div><div>12</div></div><div><div>10</div><div>10</div><div>10</div><div>11</div></div><div><div>14</div><div>14</div><div>13</div><div>14</div></div><div><div>15</div><div>15</div><div>14</div><div>15</div></div><div><div>18</div><div>18</div><div>17</div><div>18</div></div></div>
<div>(e)</div> <div><div>BOX-1</div><div><div><div>A</div><div>B</div><div>C</div><div>A</div><div>B</div><div>C</div></div><div><div>B</div><div>C</div><div>C</div><div>A</div><div>-</div><div>A</div></div></div><div>A box defined as an array of 2x6</div></div> <div><div>BOX-2</div><div><div><div>A</div><div>B</div><div>B</div><div>A</div></div><div><div>B</div><div>A</div><div>-</div><div>C</div></div></div><div>A box defined as an array of 2x4</div></div> <div><div>Resulting mem</div><div><div>0</div><div>2</div><div>4</div><div>5</div><div>7</div></div><div><div>2</div><div>0</div><div>2</div><div>3</div><div>5</div></div><div><div>4</div><div>2</div><div>1</div><div>2</div><div>4</div></div><div><div>6</div><div>4</div><div>3</div><div>4</div><div>3</div></div><div><div>8</div><div>6</div><div>5</div><div>6</div><div>5</div></div><div><div>9</div><div>7</div><div>6</div><div>5</div><div>6</div></div><div><div>11</div><div>9</div><div>8</div><div>7</div><div>7</div></div></div>
<div>(f)</div> <div><div>BOX-1</div><div><div><div>-</div><div>M</div><div>V</div><div>R</div></div><div><div>M</div><div>-</div><div>R</div><div>-</div></div><div><div>R</div><div>M</div><div>M</div><div>-</div></div><div><div>V</div><div>M</div><div>-</div><div>R</div></div></div><div>A box defined as an array of 4x4</div></div> <div><div>BOX-2</div><div><div><div>M</div><div>-</div><div>-</div><div>V</div></div><div><div>V</div><div>R</div><div>R</div><div>M</div></div><div><div>-</div><div>-</div><div>-</div><div>-</div></div><div><div>-</div><div>M</div><div>-</div><div>R</div></div></div><div>A box defined as an array of 4x4</div></div> <div><div>Resulting mem</div><div><div>0</div><div>2</div><div>4</div><div>5</div><div>8</div></div><div><div>3</div><div>5</div><div>6</div><div>7</div><div>8</div></div><div><div>6</div><div>8</div><div>9</div><div>10</div><div>11</div></div><div><div>9</div><div>10</div><div>12</div><div>13</div><div>13</div></div><div><div>11</div><div>12</div><div>14</div><div>15</div><div>15</div></div></div>

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 instance, 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`**, respectively, 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  $(ncol1+1) \times (ncol2+1)$**  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 functions return 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>`**, **`<limits>`**, **`<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 constraints 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

```
1  #include "the4.h"
2
3
4  int dp_sln(char**& arr1, char**& arr2, int nrow, int ncol1, int ncol2, int**& mem){ //dynamic programming
5
6      //your code here
7
8      return 0; // this is a dummy return value. YOU SHOULD CHANGE THIS!
9  }
10
11
```

test.cpp

```

1 // this file is for you for testing purposes, it won't be included in evaluation.
2
3 #include <iostream>
4 #include <random>
5 #include <ctime>
6 #include <cstdlib>
7 #include <algorithm>
8 #include <vector>
9 #include "the4.h"
10
11 char getRandomChar(){
12     char r = rand() % 5 + 65;
13     return r;
14 }
15
16
17 void randomArray(char**& box1, char**& box2, int nrow, int ncol1, int ncol2)
18 {
19     box1 = new char* [nrow];
20     box2 = new char* [nrow];
21     std::vector<char> column;
22
23     for (int i = 0; i < nrow; i++) {
24         box1[i] = new char [ncol1];
25         box2[i] = new char [ncol2];
26     }
27
28     for (int i = 0; i < ncol1; i++)
29     {
30         int nfull = rand() % nrow + 1;
31         for (int j = 0; j < nfull; j++) {
32             char r = getRandomChar();
33             column.push_back(r);
34         }
35         for (int j = nfull; j < nrow; j++) {
36             column.push_back('-');
37         }
38         std::random_shuffle(column.begin(), column.end());
39         for (int j = 0; j < nrow; j++)
40             box1[j][i] = column[j];
41         column.clear();
42     }
43
44     for (int i = 0; i < ncol2; i++)
45     {
46         int nfull = rand() % nrow + 1;
47         for (int j = 0; j < nfull; j++) {
48             char r = getRandomChar();
49             column.push_back(r);
50         }
51         for (int j = nfull; j < nrow; j++) {
52             column.push_back('-');
53         }
54         std::random_shuffle(column.begin(), column.end());
55         for (int j = 0; j < nrow; j++)
56             box2[j][i] = column[j];
57     }
58 }
59
60
61 void printArrayInline(char** arr, int nrow, int ncol){
62     std::cout << "[ ";
63     for(int i = 0; i < nrow; i++){
64         std::cout << "[";
65         for (int j = 0; j < ncol; j++) {
66             std::cout << arr[i][j];
67             if (j == ncol - 1)
68                 std::cout << "]";
69             else
70                 std::cout << ", ";
71         }
72         if (i == nrow - 1)
73             std::cout << " ]" << std::endl;
74         else
75             std::cout << ",\n";
76     }
77 }
78
79
80 void printMemInline(int** arr, int nrow, int ncol){
81     std::cout << "[ ";
82     for(int i = 0; i < nrow; i++){
83         std::cout << "[";
84         for (int j = 0; j < ncol; j++) {
85             std::cout << arr[i][j];
86             if (j == ncol - 1)
87                 std::cout << "]";
88             else
89                 std::cout << ", ";
90         }
91         if (i == nrow - 1)
92             std::cout << " ]" << std::endl;
93         else
94             std::cout << ",\n";
95     }
96 }
97
98
99 void fillArray(char**& box1, char**& box2, int nrow, int ncol1, int ncol2)
100 {
101
102     box1 = new char* [nrow];
103     box2 = new char* [nrow];
104
105     for (int i = 0; i < nrow; i++) {
106         box1[i] = new char [ncol1];
107         box2[i] = new char [ncol2];
108     }
109
110     // DO NOT FORGET TO CHANGE THE 31 - 33 VALUES AT THE BEGINNING OF EACH METHOD CALL

```

```

110 // "DO NOT FORGET TO CHANGE THE nrow, ncol1, ncol2 VALUES AT THE BEGINNING OF test() METHOD!!!!!!"
111 // EXAMPLE (a)
112
113 box1[0][0] = 'M';
114 box1[1][0] = 'R';
115 box1[2][0] = 'V';
116 box1[3][0] = '-';
117
118 box2[0][0] = '-'; box2[0][1] = 'M'; box2[0][2] = 'V';
119 box2[1][0] = 'M'; box2[1][1] = '-'; box2[1][2] = 'M';
120 box2[2][0] = 'R'; box2[2][1] = '-'; box2[2][2] = 'R';
121 box2[3][0] = 'V'; box2[3][1] = 'R'; box2[3][2] = '-';
122
123
124 // "DO NOT FORGET TO CHANGE THE nrow, ncol1, ncol2 VALUES AT THE BEGINNING OF test() METHOD!!!!!!"
125 // EXAMPLE (b)
126 /*
127 box1[0][0] = '-'; box1[0][1] = 'B'; box1[0][2] = 'C';
128 box1[1][0] = 'B'; box1[1][1] = '-'; box1[1][2] = 'C';
129
130 box2[0][0] = 'A'; box2[0][1] = '-'; box2[0][2] = 'C'; box2[0][3] = 'A'; box2[0][4] = 'B';
131 box2[1][0] = '-'; box2[1][1] = 'C'; box2[1][2] = 'C'; box2[1][3] = 'A'; box2[1][4] = '-';
132 */
133
134 // "DO NOT FORGET TO CHANGE THE nrow, ncol1, ncol2 VALUES AT THE BEGINNING OF test() METHOD!!!!!!"
135 // EXAMPLE (c)
136 /*
137 box1[0][0] = 'A'; box1[0][1] = 'A'; box1[0][2] = 'C'; box1[0][3] = 'B'; box1[0][4] = '-';
138 box1[1][0] = 'B'; box1[1][1] = 'C'; box1[1][2] = '-'; box1[1][3] = '-'; box1[1][4] = 'A';
139 box1[2][0] = 'A'; box1[2][1] = 'C'; box1[2][2] = '-'; box1[2][3] = 'A'; box1[2][4] = '-';
140 box1[3][0] = '-'; box1[3][1] = 'C'; box1[3][2] = '-'; box1[3][3] = '-'; box1[3][4] = 'C';
141
142 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] = 'A';
143 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] = 'C';
144 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] = 'B';
145 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] = 'B';
146 */
147
148 // "DO NOT FORGET TO CHANGE THE nrow, ncol1, ncol2 VALUES AT THE BEGINNING OF test() METHOD!!!!!!"
149 // EXAMPLE (d)
150 /*
151 box1[0][0] = '-'; box1[0][1] = 'Y'; box1[0][2] = 'W'; box1[0][3] = 'Y'; box1[0][4] = '-'; box1[0][5] = '-';
152 box1[1][0] = '-'; box1[1][1] = 'Z'; box1[1][2] = 'Y'; box1[1][3] = 'Y'; box1[1][4] = '-'; box1[1][5] = 'W';
153 box1[2][0] = 'W'; box1[2][1] = 'Y'; box1[2][2] = 'Z'; box1[2][3] = 'Y'; box1[2][4] = '-'; box1[2][5] = '-';
154 box1[3][0] = 'W'; box1[3][1] = 'W'; box1[3][2] = '-'; box1[3][3] = 'Y'; box1[3][4] = '-'; box1[3][5] = 'Y';
155 box1[4][0] = '-'; box1[4][1] = 'Z'; box1[4][2] = '-'; box1[4][3] = '-'; box1[4][4] = 'Z'; box1[4][5] = 'Z';
156
157 box2[0][0] = 'Z'; box2[0][1] = 'W'; box2[0][2] = '-';
158 box2[1][0] = 'W'; box2[1][1] = 'Y'; box2[1][2] = '-';
159 box2[2][0] = 'Y'; box2[2][1] = 'Z'; box2[2][2] = 'Y';
160 box2[3][0] = '-'; box2[3][1] = 'Z'; box2[3][2] = '-';
161 box2[4][0] = '-'; box2[4][1] = '-'; box2[4][2] = '-';
162 */
163
164 // "DO NOT FORGET TO CHANGE THE nrow, ncol1, ncol2 VALUES AT THE BEGINNING OF test() METHOD!!!!!!"
165 // EXAMPLE (e)
166 /*
167 box1[0][0] = 'A'; box1[0][1] = 'B'; box1[0][2] = 'C'; box1[0][3] = 'A'; box1[0][4] = 'B'; box1[0][5] = 'C';
168 box1[1][0] = 'B'; box1[1][1] = 'C'; box1[1][2] = 'C'; box1[1][3] = 'A'; box1[1][4] = '-'; box1[1][5] = 'A';
169
170 box2[0][0] = 'A'; box2[0][1] = 'B'; box2[0][2] = 'B'; box2[0][3] = 'A';
171 box2[1][0] = 'B'; box2[1][1] = 'A'; box2[1][2] = '-'; box2[1][3] = 'C';
172 */
173
174 // "DO NOT FORGET TO CHANGE THE nrow, ncol1, ncol2 VALUES AT THE BEGINNING OF test() METHOD!!!!!!"
175 // EXAMPLE (f)
176 /*
177 box1[0][0] = '-'; box1[0][1] = 'M'; box1[0][2] = 'V'; box1[0][3] = 'R';
178 box1[1][0] = 'M'; box1[1][1] = '-'; box1[1][2] = 'R'; box1[1][3] = '-';
179 box1[2][0] = 'R'; box1[2][1] = 'M'; box1[2][2] = 'M'; box1[2][3] = '-';
180 box1[3][0] = 'V'; box1[3][1] = 'M'; box1[3][2] = '-'; box1[3][3] = 'R';
181
182 box2[0][0] = 'M'; box2[0][1] = '-'; box2[0][2] = '-'; box2[0][3] = 'V';
183 box2[1][0] = 'V'; box2[1][1] = 'R'; box2[1][2] = 'R'; box2[1][3] = 'M';
184 box2[2][0] = '-'; box2[2][1] = '-'; box2[2][2] = '-'; box2[2][3] = '-';
185 box2[3][0] = '-'; box2[3][1] = 'M'; box2[3][2] = '-'; box2[3][3] = 'R';
186 */
187
188 }
189
190 void test(){
191     clock_t begin, end;
192     double duration;
193     int min_cost_dp;
194
195
196     int nrow = 4; // max 100
197     int ncol1 = 1; // max 100
198     int ncol2 = 3; // max 100
199     char** box1, ** box2;
200     //randomArray(box1, box2, nrow, ncol1, ncol2);
201     fillArray(box1, box2, nrow, ncol1, ncol2);
202     std::cout << "BOX-1:" << std::endl;
203     printArrayInLine(box1, nrow, ncol1);
204     std::cout << "\nBOX-2:" << std::endl;
205     printArrayInLine(box2, nrow, ncol2);
206     std::cout << "\n\n";
207
208     int** mem = new int*[ncol1+1];
209
210     for(int i = 0; i <= ncol1; i++){
211         mem[i] = new int [ncol2+1];
212         for (int j = 0; j <= ncol2; j++)
213             mem[i][j] = -1;
214     }
215
216
217
218
219     std::cout << " DYNAMIC PROGRAMMING: " << std::endl;

```



```

220
221     for(int i = 0; i <= ncol1; i++){
222         for (int j = 0; j <= ncol2; j++){
223             mem[i][j] = -1;
224         }
225     }
226
227     if ((begin = clock() ) == -1)
228         std::cerr << "clock error" << std::endl;
229
230     min_cost_dp = dp_sln(box1, box2, nrow, ncol1, ncol2, mem);
231
232     if ((end = clock() ) == -1)
233         std::cerr << "clock error" << std::endl;
234
235     duration = ((double) end - begin) / CLOCKS_PER_SEC;
236     std::cout << "Duration: " << duration << " seconds." << std::endl;
237
238     std::cout << "Min cost: " << min_cost_dp << std::endl;
239     std::cout << "Final mem: " << std::endl;
240     printMemInLine(mem, ncol1+1, ncol2+1);
241
242     std::cout << "-----";
243     std::cout << "\n" << std::endl;
244
245 }
246
247 int main()
248 {
249     srand(time(0));
250     test();
251     return 0;
252 }
253

```

the4\_solution.cpp

```

1 #include "sol4.h"
2
3
4
5 int recursive_sln(int i, int j, char**& arr1, char**& arr2, int nrow, int ncol1, int ncol2, int &number_of_calls){ //direct recursive
6     number_of_calls+=1;
7
8     if (i == 0) {
9         if (j == 0)
10             return 0;    // never goes here
11         else {
12             int num_of_insertions = 0;
13             // count the number of cells in arr2
14             for (int n = 0; n < j; n++) {
15                 for (int m = 0; m < nrow; m++) {
16                     if (arr2[m][n] == '-')
17                         continue;
18                     else
19                         num_of_insertions ++; // num of insertions
20                 }
21             }
22             return num_of_insertions;
23         }
24     }
25     else if (j == 0) {
26         int num_of_deletions = 0;
27         // count the number of cells in arr1
28         for (int n = 0; n < i; n++) {
29             for (int m = 0; m < nrow; m++) {
30                 if (arr1[m][n] == '-')
31                     continue;
32                 else
33                     num_of_deletions ++; // num of deletions
34             }
35         }
36         return num_of_deletions;
37     }
38     else {
39         // DELETION
40         int num_of_deletions = 0;
41         // count the number of cells in arr1
42         for (int m = 0; m < nrow; m++) {
43             if (arr1[m][i-1] == '-')
44                 continue;
45             else
46                 num_of_deletions ++;
47         }
48
49         int x = recursive_sln(i-1, j, arr1, arr2, nrow, ncol1, ncol2, number_of_calls);
50         int cost = x + num_of_deletions; // cost of deletions
51
52         // INSERTION
53         int num_of_insertions = 0;
54         // count the number of cells in arr2
55         for (int m = 0; m < nrow; m++) {
56             if (arr2[m][j-1] == '-')
57                 continue;
58             else
59                 num_of_insertions ++;
60         }
61
62         int y = recursive_sln(i, j-1, arr1, arr2, nrow, ncol1, ncol2, number_of_calls);
63         if (cost > y + num_of_insertions)
64             cost = y + num_of_insertions; // cost of insertions
65
66         // REPLACEMENT
67         int cost_of_replacements = 0;
68         // check for the same items in arr1 and arr2
69         for (int m = 0; m < nrow; m++) {
70             if (arr1[m][i-1] == arr2[m][j-1])
71                 continue;
72             else if (arr1[m][i-1] == '-' || arr2[m][j-1] == '-')
73                 cost_of_replacements += 2;
74             else
75                 cost_of_replacements ++;
76         }
77         int z = recursive_sln(i-1, j-1, arr1, arr2, nrow, ncol1, ncol2, number_of_calls);
78         if (cost > z + cost_of_replacements)
79             cost = z + cost_of_replacements;
80
81         // REORDERING
82         int cost_of_reordering = 0;
83         std::string order1 = "";
84         std::string order2 = "";
85         for (int m = 0; m < nrow; m++) {
86             order1 += arr1[m][i-1];
87             order2 += arr2[m][j-1];
88         }
89         bool equivalent = true;
90         for (int m = 0; m < nrow; m++) {
91             std::size_t found = order2.find(order1[m]);
92             if (found!=std::string::npos)
93                 order2 = order2.substr(0, found) + order2.substr(found+1);
94             else {
95                 equivalent = false;
96                 break;
97             }
98         }
99
100         if (equivalent) {
101             for (int m = 0; m < nrow; m++) {
102                 if (arr1[m][i-1] == arr2[m][j-1])
103                     continue;
104                 else
105                     cost_of_reordering += 1;
106             }
107             if (cost > z + cost_of_reordering)
108                 cost = z + cost_of_reordering;
109         }
110     }

```

```

110     }
111     return cost;
112 }
113 }
114 }
115 }
116 }
117 }
118
119 int memoization_sln(int i, int j, char**& arr1, char**& arr2, int nrow, int ncol1, int ncol2, int**& mem){ //memoization
120     // mem is (ncol1+1) x (ncol2+1)
121     // initialize trivial parts of mem
122     if (i == 0) {
123         if (j == 0)
124             mem[i][j] = 0; // never goes here
125         else {
126             mem[0][j] = 0;
127             // count the number of cells in arr2
128             for (int n = 0; n < j; n++) {
129                 for (int m = 0; m < nrow; m++) {
130                     if (arr2[m][n] == '-')
131                         continue;
132                     else
133                         mem[0][j] ++; // num of insertions
134                 }
135             }
136             mem[0][0] = 0;
137         }
138     }
139     else if (j == 0) {
140         mem[i][0] = 0;
141         // count the number of cells in arr1
142         for (int n = 0; n < i; n++) {
143             for (int m = 0; m < nrow; m++) {
144                 if (arr1[m][n] == '-')
145                     continue;
146                 else
147                     mem[i][0] ++; // num of deletions
148             }
149         }
150     }
151 }
152
153 // for the nontrivial parts of mem
154 else {
155     // DELETION
156     int num_of_deletions = 0;
157     // count the number of cells in arr1
158     for (int m = 0; m < nrow; m++) {
159         if (arr1[m][i-1] == '-')
160             continue;
161         else
162             num_of_deletions ++;
163     }
164
165     if (mem[i-1][j] == -1)
166         memoization_sln(i-1, j, arr1, arr2, nrow, ncol1, ncol2, mem);
167     int cost = mem[i-1][j] + num_of_deletions; // cost of deletions
168
169     // INSERTION
170     int num_of_insertions = 0;
171     // count the number of cells in arr2
172     for (int m = 0; m < nrow; m++) {
173         if (arr2[m][j-1] == '-')
174             continue;
175         else
176             num_of_insertions ++;
177     }
178
179     if (mem[i][j-1] == -1)
180         memoization_sln(i, j-1, arr1, arr2, nrow, ncol1, ncol2, mem);
181     if (cost > mem[i][j-1] + num_of_insertions)
182         cost = mem[i][j-1] + num_of_insertions; // cost of insertions
183
184     // REPLACEMENT
185     int cost_of_replacements = 0;
186     // check for the same items in arr1 and arr2
187     for (int m = 0; m < nrow; m++) {
188         if (arr1[m][i-1] == arr2[m][j-1])
189             continue;
190         else if (arr1[m][i-1] == '-' || arr2[m][j-1] == '-')
191             cost_of_replacements += 2;
192         else
193             cost_of_replacements ++;
194     }
195
196     if (mem[i-1][j-1] == -1)
197         memoization_sln(i-1, j-1, arr1, arr2, nrow, ncol1, ncol2, mem);
198     if (cost > mem[i-1][j-1] + cost_of_replacements)
199         cost = mem[i-1][j-1] + cost_of_replacements;
200
201     // REORDERING
202     int cost_of_reordering = 0;
203     std::string order1 = "";
204     std::string order2 = "";
205     for (int m = 0; m < nrow; m++) {
206         order1 += arr1[m][i-1];
207         order2 += arr2[m][j-1];
208     }
209     bool equivalent = true;
210     for (int m = 0; m < nrow; m++) {
211         std::size_t found = order2.find(order1[m]);
212         if (found != std::string::npos)
213             order2 = order2.substr(0, found) + order2.substr(found+1);
214         else {
215             equivalent = false;
216             break;
217         }
218     }
219 }

```

```

220         if (equivalent) {
221             for (int m = 0; m < nrow; m++) {
222                 if (arr1[m][i-1] == arr2[m][j-1])
223                     continue;
224                 else
225                     cost_of_reordering += 1;
226             }
227             if (cost > mem[i-1][j-1] + cost_of_reordering)
228                 cost = mem[i-1][j-1] + cost_of_reordering;
229         }
230         mem[i][j] = cost;
231     }
232 }
233 return mem[i][j];
234 }
235 }
236
237
238
239 int dp_sln(char**& arr1, char**& arr2, int nrow, int ncol1, int ncol2, int**& mem){ //memoization
240
241     // mem is (ncol1+1) x (ncol2+1)
242
243     // initialize trivial parts of mem
244     mem[0][0] = 0;
245     for (int i = 1; i <= ncol1; i++) {
246         mem[i][0] = 0;
247         // count the number of cells in arr1
248         for (int n = 0; n < i; n++) {
249             for (int m = 0; m < nrow; m++) {
250                 if (arr1[m][n] == '-')
251                     continue;
252                 else
253                     mem[i][0]++; // num of deletions
254             }
255         }
256     }
257     for (int j = 1; j <= ncol2; j++) {
258         mem[0][j] = 0;
259         // count the number of cells in arr2
260         for (int n = 0; n < j; n++) {
261             for (int m = 0; m < nrow; m++) {
262                 if (arr2[m][n] == '-')
263                     continue;
264                 else
265                     mem[0][j]++; // num of insertions
266             }
267         }
268     }
269
270     // now start dynamic programming
271     for (int i = 1; i <= ncol1; i++)
272         for (int j = 1; j <= ncol2; j++) {
273
274             // DELETION
275             int num_of_deletions = 0;
276             // count the number of cells in arr1
277             for (int m = 0; m < nrow; m++) {
278                 if (arr1[m][i-1] == '-')
279                     continue;
280                 else
281                     num_of_deletions++;
282             }
283
284             int cost = mem[i-1][j] + num_of_deletions; // cost of deletions
285
286             // INSERTION
287             int num_of_insertions = 0;
288             // count the number of cells in arr2
289             for (int m = 0; m < nrow; m++) {
290                 if (arr2[m][j-1] == '-')
291                     continue;
292                 else
293                     num_of_insertions++;
294             }
295
296             if (cost > mem[i][j-1] + num_of_insertions)
297                 cost = mem[i][j-1] + num_of_insertions; // cost of insertions
298
299             // REPLACEMENT
300             int cost_of_replacements = 0;
301             // check for the same items in arr1 and arr2
302             for (int m = 0; m < nrow; m++) {
303                 if (arr1[m][i-1] == arr2[m][j-1])
304                     continue;
305                 else if (arr1[m][i-1] == '-' || arr2[m][j-1] == '-')
306                     cost_of_replacements += 2;
307                 else
308                     cost_of_replacements++;
309             }
310             if (cost > mem[i-1][j-1] + cost_of_replacements)
311                 cost = mem[i-1][j-1] + cost_of_replacements;
312
313             // REORDERING
314             int cost_of_reordering = 0;
315             std::string order1 = "";
316             std::string order2 = "";
317             for (int m = 0; m < nrow; m++) {
318                 order1 += arr1[m][i-1];
319                 order2 += arr2[m][j-1];
320             }
321             bool equivalent = true;
322             for (int m = 0; m < nrow; m++) {
323                 std::size_t found = order2.find(order1[m]);
324                 if (found != std::string::npos)
325                     order2 = order2.substr(0, found) + order2.substr(found+1);
326                 else {
327                     equivalent = false;
328                     break;

```

```

329     }
330 }
331
332 if (equivalent) {
333     for (int m = 0; m < nrow; m++) {
334         if (arr1[m][i-1] == arr2[m][j-1])
335             continue;
336         else
337             cost_of_reordering += 1;
338     }
339     if (cost > mem[i-1][j-1] + cost_of_reordering)
340         cost = mem[i-1][j-1] + cost_of_reordering;
341 }
342
343 mem[i][j] = cost;
344 }
345
346 return mem[ncol1][ncol2];
347 }
348
349
350
351

```

[VPL](#)

You are logged in as moustafa ismail hamed mohamed ismail (Log out)

CENG 315 All Sections

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

