[CENG 315 All Sections] Algorithms

Navigation

- Dashboard
- Site home
- Site pages
- My courses
- 571 - Computer Engineering
- [CENG 351 All Sections]
- CENG 300 All Sections
- CENG 300 Section 4
- CENG 315 All Sections
- Participants
- Badges
- Competencies
- Grades
- General
- October 13 - October 19
- October 20 - October 26
- October 27 - November 2
- November 3 - November 9
- November 10 - November 16
- November 17 - November 23
- November 24 - November 30
- December 1 - December 7
- December 8 - December 14
- December 15 - December 21
- THE5 Discussion Forum
- THE5
- Description
- Submission view
- THE5_IO
- THE5 Official IO
- THE5 Solution
- Some-graph-Alg-And-Shortest-path-Algs-2021-AY
- December 22 - December 28
- December 29 - January 4
- January 5 - January 11
- January 12 - January 18
- January 19 - January 25
- CENG 315 Section 3
- CENG 331 All Sections
- CENG 331 Section 2
- CENG 351 Section 3
- 651 - Music and Fine Arts
- 612 - Modern Languages (Persian)
- 642 - Turkish Language

Description | Submission view

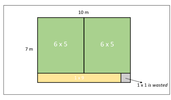
THE5

Available from: Friday, December 17, 2021, 11:59 AM
Due date: Friday, December 17, 2021, 11:59 PM
Requested files: test.cpp, the5.cpp (Download)
Type of work: Individual work

PROBLEM DEFINITION

In this exam, you will help farmers having a rectangular shaped field (Width x Length) who want to partition this field according to some fixed configuration considering the type of the plant. The aim here is to reduce the area which does not belong any partition. For example, let our farmers have a field of (7 x 10) and the smallest area needed for a group of plant might be (6 x 5) and (1 x 9) for another group. They can use the same group more than once. However, the width and the length is fixed for a group of plant considering the watering and optimal case for sunlight etc. In other words, an area for a plant is not open to rotation.

What is the minimum area wasted by not planting anything? An optimal solution can be seen as:



In general case, the solution can be found searching through possible cuts for partitions both vertically and horizontally. Here is the guide for computation:

- Initially,
  - $WastedArea(W, L) = 0$ if ($Wx L$) matches with some partition
  - $WastedArea(W, L) = W * L$ otherwise **(TO BE UPDATED BELOW)**
- Update phase,
  - $WastedArea(W, L) = min \{$

$$WastedArea(W, L),$$

$$min_{1 \le M \le (W/2)} WastedArea(W \cdot M, L) + WastedArea(M, L),$$

$$min_{1 \le N \le (L/2)} WastedArea(W, L \cdot N) + WastedArea(W, N),$$

$$\}$$

Notice that in the search of possible cuts, we go through the half of the edges since the results are symmetrical in our case. **This will not be applied in bottom-up solution. That is, we will construct the solution for all of the edges in bottom-up approach.**

You will solve this problem for 3 different methods as in the previous exam:

- Recursive method,
- Memoization method,
- Bottom-up method.

Specifications:

- There are 3 **tasks** to be solved in **12 hours** in this take home exam. All of the methods will return the wasted area according to the given field and partitions.
- Partitions will be given in a 2D Boolean array where if a partition of M x N exists then, partitions[M][N] is true, false otherwise. The size of the array is (W+1) x (L+1), *including the redundant initialization for 0th row and 0th column for the sake of simplicity*.
- Recursive and Memoization method will compute the number of calls during the computation where the Bottom-up method will count the number of iterations while searching possible cuts (separately for vertical and horizontal cases).
- You will implement your solutions in **the5.cpp** file.
- You are free to add other functions to **the5.cpp**
- Do **not** change the first line of **the5.cpp**, which is **#include "the5.h"**
- <iostream>, <climits> , <cmath>, <cstdlib> are included in "the5.h" for your convenience.
- Here is the full content of "the5.h" file:

```
#ifndef _THE_5_H_
#define _THE_5_H_

#include <iostream>
#include <climits>
#include <cmath>
#include <cstdlib>

int recursiveMethod(int W, int L, bool** partitions, int* numberOfCalls);
int memoizationMethod(int W, int L, bool** partitions, int* numberOfCalls);
int bottomUpMethod(int W, int L, bool** partitions, int* numberOfIterations);

#endif
```

- Do **not** change the arguments and return value of the functions **recursiveMethod, memoizationMethod, bottomUpMethod** in the file **the5.cpp**
- Do **not** include any other library or write include anywhere in your **the5.cpp** file (not even in comments).
- You are given **test.cpp** file to **test** your work on **Odtuclass** 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** your work on your locale as:

```
g++ test.cpp the5.cpp -Wall -std=c++11 -o test

./test
```

- You can test your **the5.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 reevaluated with **completely different** inputs after the exam.

The system has the following limits:

- a maximum execution time of 32 seconds (your functions should return in less than 1 seconds for the largest inputs)
- 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.

Constraints:

- $1 \le W \le 600$, $1 \le L \le 600$
- 1 ≤ Number of partitions ≤ 200
- 1 ≤ M (width of a partition) ≤ W
- 1 ≤ N (length of a partition) ≤ L
- Since recursive method generates huge number of calls, it will be tested small inputs only.

Evaluation:

- After your exam, black box evaluation will be carried out. You will get full points if you fill the return value as the wasted area and the number of calls/iterations are correct according to the given method.

Example IO:

Given field of 7 x 10 and partitions as (6 x 5) and (1 x 9):

1) Recursive Method:

- set number of calls = 7491377
- return 1

2) Memoization Method:

- set number of calls = 574
- return 1

3) Bottom-Up Method:

- set number of iterations = 644
- return 1

TEST EVALUATION:
Due to the limitation of our programming environment, larger inputs can not be stored. Therefore, we create them when needed. The test evaluation has 2 phases for each method. The first phase has the same inputs given here to check if your codes work fully correct on small inputs. If your code works perfectly on at least one of the first 2 tasks, it will also be tested on the second phase for the task(s) that works correct. The second phase on the other hand, creates and sorts larger arrays that are on boundaries. (Note that the tests give 50 pts for each phase. However, the real inputs will be like the ones on the second phase which means if your code works only on phase 1, it is possible for your real grade to be 0 afterwards).

Requested files

test.cpp

```
#include <iostream>
#include <random>
#include <ctime>
#include "the5.h"

int getRandomInt(int max){
    int r = rand() % max + 1;
    return r;
}

void createAndWritePartitions(int numberOfPartitions, bool** partitions, int W, int L){

    std::cout << "Field: " << W << " x " << L << std::endl;

    for(int p = 0; p < numberOfPartitions; p++){
        int w = getRandomInt(W);
        int l = getRandomInt(L);
        while (partitions[w][l]){      // make sure that no two partitions are same
            w = getRandomInt(W);
            l = getRandomInt(L);
        }
        partitions[w][l] = true;
        std::cout << "Partition " << (p+1) << ": " << w << " x " << l << std::endl;
    }
}

void testRecursive(int W, int L, bool** partitions){
    std::cout << "\n***Testing Recursive Method***\n";
    clock_t begin, end;
    double duration;
    int numberOfCalls = 0;

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

    int wastage = recursiveMethod(W, L, partitions, &numberOfCalls);

    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 << "Wasted Area: " << wastage << std::endl;
    std::cout << "Number of Calls: " << numberOfCalls << std::endl;
    std::cout << "**********************************\n";
}

void testMemoization(int W, int L, bool** partitions){
    std::cout << "\n***Testing Memoization Method***\n";
    clock_t begin, end;
    double duration;
    int numberOfCalls = 0;

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

    int wastage = memoizationMethod(W, L, partitions, &numberOfCalls);

    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 << "Wasted Area: " << wastage << std::endl;
    std::cout << "Number of Calls: " << numberOfCalls << std::endl;
    std::cout << "**********************************\n";
}

void testBottomUp(int W, int L, bool** partitions){
    std::cout << "\n***Testing Bottom-up Method***\n";
    clock_t begin, end;
    double duration;
    int numberOfIterations = 0;

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

    int wastage = bottomUpMethod(W, L, partitions, &numberOfIterations);

    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 << "Wasted Area: " << wastage << std::endl;
    std::cout << "Number of Iterations: " << numberOfIterations << std::endl;
    std::cout << "**********************************\n";
}
void test(){

    // Field
    int W = 5;
    int L = 3;

    // Initialization for partitions
    bool** partitions = new bool*[W+1];
    for(int i = 0; i < W+1; i++){
        partitions[i] = new bool[L+1];
        for (int j = 0; j < L+1; j++){
            partitions[i][j] = false;
        }
    }

    int numberOfPartitions = 2;
    createAndWritePartitions(numberOfPartitions, partitions, W, L);

    testRecursive(W, L, partitions);
    testMemoization(W, L, partitions);
    testBottomUp(W, L, partitions);
}
int main(){
    srand(unsigned(time(0)));
    test();
    return 0;
}
```

the5.cpp

```
#include "the5.h"
// do not add extra libraries here

/*
W: the width of the field
L: the length of the field
partitions: 2D Boolean array for partitions, if a partition of w x l exists then partitions[x][y] is true.
numberOfCalls/numberOfIterations: method specific control parameter
*/

int recursiveMethod(int W, int L, bool** partitions, int* numberOfCalls){
    return 0; // this is a dummy return value. YOU SHOULD CHANGE THIS!
}

int memoizationMethod(int W, int L, bool** partitions, int* numberOfCalls){
    return 0; // this is a dummy return value. YOU SHOULD CHANGE THIS!
}

int bottomUpMethod(int W, int L, bool** partitions, int* numberOfIterations){
    return 0; // this is a dummy return value. YOU SHOULD CHANGE THIS!
}
```

VPL