[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
          - DP Extra Lecture-Problem: Maximum sum such that n...
          - CEng315-Yazici-Dynamic-Prog-2021
          - THE4 Discussion Forum
          - THE 4
            - Description
            - Submission view
        - December 8 - December 14
        - December 15 - December 21
        - 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 Fine Arts
    - 612 - Modern Languages (Persian)
    - 642 - Turkish Language

Description | Submission view

THE 4

Due date: Friday, December 3, 2021, 11:59 PM
Requested files: the4.cpp, test.cpp (Download)
Type of work: Individual work

Specifications:

- There are **3 tasks** to be solved in **12 hours** in this take home exam.
- You will implement your solutions in **the4.cpp** file.
- You are free to add other functions to **the4.cpp**
- 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 **types** of the functions **recursive_sln(), memoization_sln()** and **dp_sln()** in the file **the4.cpp.** (You should change return **values**, on the other hand.)
- Do **not** include any other library or write include anywhere in your **the4.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 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 reevaluated 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 recursive_sln(int i, int*& arr, int &number_of_calls);
int memoization_sln(int i, int*& arr, int*& mem);
int dp_sln(int size, int*& arr, int*& mem);
```

In this exam, given an array of positive numbers, you are asked to find a the maximum sum of a subsequence of the array with the constraint that any two numbers in the subsequence should have at least an index difference of 3 in the array (e.g. in arr{'0','1','2','3','4'}, index difference of '4' and '1' is 3). To illustrate, when arr = {50, 30, 100, 10, 80, 100} is given, your functions should return 200 (sum of 100 and 100) or when arr = {8, 9, 15} is given, they should return 15.
You will implement three different functions for three different solutions of that problem:

- Direct recursive implementation in ***recursive_sln()***
- Recursion with memoization in ***memoization_sln()***
- Dynamic programming in ***dp_sln()***

All three functions are expected to **return** the answer to the given problem which is **the maximum sum value** (such that index difference between elements is at least 3). Return **only** the max sum value and nothing more.

The number of recursive calls that your recursive function makes should be counted. That number should be counted and stored using the ***int &number_of_calls*** variable, which is the last parameter at the definition of the *recursive_sln()*. Basically, the value of that variable should be incremented by one at each execution of the *recursive_sln()* function. In order to accomplish that, the increment operation may be done at the first line of the function implementation, as already done in the function template given to you. So, **do not change the first line of the *recursive_sln()* function and do not manipulate the *number_of_calls* variable at anywhere else**. Do **not return** that variable. Since it is passed by reference, its final value will be available for testing/grading without returning it.

For memoization and dynamic programming, you should use ***int*& *mem*** variable (i.e. array), which is the last parameter at definitions of those functions, as **the array of memoized values**. For both *memoization_sln()* and *dp_sln()* functions, final values in the *mem* variable will be considered for grading. While testing and grading, the *mem* array will be initialized to all -1's. So, while implementing your functions, **you can assume that *mem* is an array of -1's. Do no return that variable/array.**

The ***int*& *arr*** variable is the parameter which passes the input array to your functions. **Do not modify that array!**

At *recursive_sln()* and *memoization_sln()*, ***int i*** is intended to represent and pass indices of *arr*. While testing and grading, it will be initialized to **sizeof(arr)-1** (i.e. the last index of the array) . At *dp_sln()*, instead of such a variable, directly the **size of the arr** is given via ***int size*** parameter.

Implement the functions in most efficient way.

Constraints:

- Maximum array size will be **1000**.
- Array elements will be positive integers in the closed interval **[0, 10000]**.

Evaluation:

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

  1. your all three functions return the correct max sum
  2. your recursive_sln() function makes the correct number of recursive calls
  3. and you fill the ***mem*** array correctly, as stated.

Example IO:

1) Given array arr = {8, 64, 55, 34, 46}:

- return value (i.e. max sum) is 110 for each of three functions.
- number of recursive calls is 5.
- at memoization and dynamic programming, final mem array is {8, 64, 64, 64, 110}

2) Given array arr = {32, 51, 51, 92, 54, 90, 13, 69, 20, 6}:

- return value (i.e. max sum) is 193 for each of three functions.
- number of recursive calls is 37.
- at memoization and dynamic programming, final mem array is {32, 51, 51, 124, 124, 141, 141, 193, 193, 193}

Requested files

the4.cpp

```
#include "the4.h"


int recursive_sln(int i, int*& arr, int &number_of_calls){ //direct recursive
    number_of_calls++;

    //your code here

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


int memoization_sln(int i, int*& arr, int*& mem){ //memoization

    //your code here

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


int dp_sln(int size, int*& arr, int*& mem){ //dynamic programming

    //your code here

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

test.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 "the4.h"

int getRandomInt(){
        int r = rand()%100;
        return r;
}


void randomArray(int*& array, int size){
    array = new int [size];
    for (int i = 0; i < size; i++){
        int r = getRandomInt();
        array[i] = r;
    }
}

void printArrayInLine(int arr[], int arraySize){
    std::cout << "[ ";
    for(int i = 0; i < arraySize; i++){
            std::cout << arr[i];
    if (i == arraySize - 1){
        continue;
    }else{
        std::cout << ", ";
    }
    }
    std::cout << " ]" << std::endl;
}

void test(){
    clock_t begin, end;
        double duration;
    int max_sum_rec;
        int *mx_sum_mem;
        int mx_sum_dp;


    int size = 10;      // max 10000
    int* arr;
    randomArray(arr, size);
    std::cout << "Array:" << std::endl;
    printArrayInLine(arr, size);


        std::cout << "____________RECURSIVE IMPLEMENTATION:_____________" << std::endl;

    int number_of_calls_rec = 0;

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

        max_sum_rec = recursive_sln(size-1, arr, number_of_calls_rec);

    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 << "Max sum: " << max_sum_rec << std::endl;
    std::cout << "Number of recursive calls: " << number_of_calls_rec << std::endl;

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


        int* mem = new int[size];


        std::cout << "_____________________MEMOIZATION:_____________________" << std::endl;

        for(int i = 0; i < size-1; i++) mem[i] = -1;


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

    max_sum_mem = memoization_sln(size-1, arr, 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 << "Max sum: " << max_sum_mem << std::endl;
        std::cout << "Final mem: " << std::endl;
        printArrayInLine(mem, size);

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


        std::cout << "_____________DYNAMIC PROGRAMMING:_____________" << std::endl;

        for(int i = 0; i < size; i++) mem[i] = -1;

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

        max_sum_dp = dp_sln(size, arr, 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 << "Max sum: " << max_sum_dp << std::endl;
        std::cout << "Final mem: " << std::endl;
        printArrayInLine(mem, size);

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

}

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

VPL