Middle East Technical University | Department of Computer Engineering

CENG 315

Algorithms

Fall 2024-2025

Take-Home Exam 3

Due date: 10 November 2024, Sunday, 23.59

1 Problem Definition

Following your friend Isabelle, you decided to move to a deserted island and make it your new home. But there are many things to do to make the place feel like a home. One of those things is covering the perimeter of the island with flowers, bushes and trees. Isabelle has been there longer than you, so she tells you some constraints to decide what to plant around the perimeter of the island:

- You can plant a flower, a bush or a tree to each 1 meter of the perimeter. On one corner of the island, there is a statue covering a 1 meter spot. Hence, if the perimeter of the island is N meters, you have $N - 1$ spots to cover with plants.
- You cannot plant two trees next to each other.
- You cannot plant two bushes next to each other.
- You can plant two flowers next to each other, but they have to be the same species.
- The cost of planting a flower, a bush, or a tree is different for each spot because of the diverse soil structure around the island.
- There are two species of each plant, i.e., there are two types of flowers, two types of bushes and two types of trees to choose from.

In this exam, you are expected to calculate the minimum possible cost of covering the perimeter with plants. The island can be seen in Figure 1, where the statue is shown as S, and the first and last planting spots are shown as spots A and B. When unravelled, spots A and B reside in the starting and ending points of the straight line.

island layout with a perimeter of N meters, statue S, and spots A and B next to the statue

perimeter marked from A to B as a straight line of length N-1 meters, with statue spot S omitted

Figure 1: Island and the perimeter

In this exam, you are expected to solve this task using the bottom-up approach (DP table creation) by completing the method `find_min_cost` shown below. Your method should return the calculated `min_cost` for the task. You should use a dynamic programming table to calculate the value.

```
int find_min_cost(const std::vector<std::vector<int>>& costs, int N);
```

- N is the length of the perimeter in meters. Hence, there are $N-1$ plantable spots. Following holds for N: $5 <= N <= 10^6$
- `costs` is a 2-D array of size $N-1$, where `costs[i]` holds the costs of planting on i^{th} spot.
- For any given spot i, the costs of plants given in `costs[i]` are in the following order: `[flower_type_1, flower_type_2, bush_type_1, bush_type_2, tree_type_1, tree_type_2]`.
- Following holds for the cost value c in `costs` for each plant type: $1 <= c <= 10$
- `min_cost` is the calculated minimum cost of the planting task.

1.1 Example I/O

```
Input (N, followed by N-1 rows of costs for N-1 spots)
---
5
3 8 2 10 1 6
3 1 10 4 9 3
1 9 6 2 4 1
10 2 6 6 10 7

Output (min_cost)
---
5
```

One possible setting with minimum cost: `tree_type_1, flower_type_2, tree_type_2, flower_type_2`.

2 Limits and Specifications

In this exam, the complexity of your implementations will be checked by obedience to VPL limits. Hence the system limitations are as follows:

- a maximum execution time of 16 seconds
- a 160 MB memory limit for the given 10 inputs (worst case input should take around 24 MB)
- a stack size of 512 KB for function calls

If you think that your implementation has the expected time complexity $O(N)$ and space complexity $O(N)$ but your solution fails, constant factors and non-global variables might be the problem.

- You will implement your solutions in the `the3.cpp` file.
- Do not change the first line of `the3.cpp`, which is `#include "the3.h"`.
- Do not change the arguments and the return value of the given functions in the file `the3.cpp`, but you are free to add other functions to `the3.cpp`.
- Do not include any other library or write include anywhere in your `the3.cpp` file (not even in comments).
- You are given a `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.
- You can test your `the3.cpp` on the virtual lab environment. If you click run, your function will be compiled and executed with `test.cpp`. If you click evaluate, you will get feedback for your current work and your work will be temporarily graded with a limited number of inputs.
- The grade you see in VPL is not your final grade, your code will be reevaluated with more inputs after the exam.
- If you want to test your work and see your outputs on your locale you can use the following commands:

```
> g++ test.cpp the3.cpp -Wall -std=c++11 -o test
> ./test
```

3 Regulations

- **Implementation and Submission:** The template files are available in the Virtual Programming Lab (VPL) activity called "THE3" on ODTUCLASS. At this point, you have two options:
  - You can download the template files, complete the implementation, and test it with the given sample I/O on your local machine. Then submit the same file through this activity.
  - You can directly use the editor of the VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submitting a file.

  Please make sure that your code runs on ODTUCLASS. There is no limitation in running your code online. The last save/submission will determine your final grade.

- **Programming Language:** You must code your program in C++. Your submission will be tested on the VPL environment in ODTUCLASS, hence you are expected to make sure your code runs successfully there.

- **Cheating: This assignment is designed to be worked on individually.** Additionally, the use of any LLMs (chatgpt, copilot, the other one that you are thinking about...) and copying code directly from the internet for implementations is strictly forbidden. Your work will be evaluated for cheating, and disciplinary action may be taken if necessary.

- **Evaluation:** Your program will be evaluated automatically using "black-box" testing, so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to worry about invalid cases. **Important Note:** The given sample I/O's are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official test cases to determine your final grade after the deadline.