Degree: Artificial Intelligence
Subject: Fundamentals of Programming 2
Practical project 2

Anna Sikora
Manuel Montoto
Nischey Verma Kumar
Computer Architecture and Operating Systems
Universitat Autònoma de Barcelona
Cerdanyola, Spain
{anna.sikora, manuel.montoto}@uab.cat

***Abstract*—As you already know, during this second semester of the 2022-2023 academic year you will have 2 practical projects in the Fundamentals of Programming 2 subject. You've developed and delivered the first one. Now, the second one comes, so this document presents the second practical project related to the concepts we've seen up to now. With the double aim of putting into practice the knowledge obtained in this subject and of acquiring independent work habits, the practical project 2 will focus on the development of a program in C which includes different aspects, such as static and dynamic memory, dynamic data structures, in this case trees and graphs, and recursion.**

Similarly to the development of the first project, you will have 3 sessions in order to deliver a functional version of the program, but, it will require more hours of dedication, not only the 3 in-present sessions. During the last sessions you must deliver the work done.

In this document we introduce a description of the problem you will have to solve. It will be related to discovering your ancestors' tree traveling through different cities.

***Index Terms*—C programming, trees, graphs, data structures' creation, searching algorithms, DFS, BFS.**

I. Introduction

With the double objective of putting into practice the knowledge obtained in this subject and of acquiring independent work habits, the laboratory practices focus on the development of C programs that cover different aspects seen up to now. This project will allow you to design and develop the code in a more independent way. We won't provide you with any template of the code. It will be your task to design the program flow (functions, separate files, structures, etc.). In this project we propose a set of possible milestones (implementation of different functionalities of the program) and a deadline (second deliverable) in which you will have to deliver the work done.

The planning necessary to achieve these milestones, and even deciding which milestones you want to achieve, is your responsibility.

In Section II we present the description of the proposed program. Section III describes how to execute the program and proposes different possible configurations. In Section IV we describe the milestones that the project may have (although they depend on you and your organization) and in Section V we give some recommendations and general orientations. Finally, in Section VI you will find the delivery description and how the project will be assessed.

II. Problem description

Imagine that you want to find your ancestors. You will have to prepare a journey through all over the world visiting different places (Figure 1) to find out your ancestors (Figure 2). You live in a given city with your parents where they were born. The information of your ancestors (previous generation) are available in civil registries of different cities. Therefore, you have to go to the next civil registry to a given city, find out the names of your ancestors together with the next city where you will have a civil registry and then information about the previous generation of your ancestors.

To simplify the search phase, we assume that both (mather and father) are from the same city. So, the grandparents from the mother's side are from one city (the same city for both), and the grandparents from the father's side are from another city (but also, from the same city for both).

Fig. 1. Visiting different places all over the world.

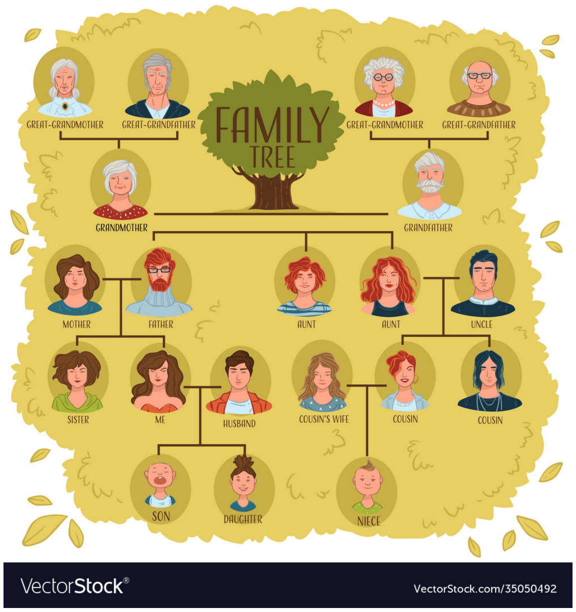

Fig. 2. Example of a family tree.

There are two main parts of the problem. First, journey all over the world where you will have to plan a trip, step by step, going to the next city and applying different routing searches. Second, building an ancestors' tree adding next generations.

A. World journey

In order to build your ancestors' tree, you will have to go from one civil registry (that will be in one city) to other one (that will be in other city). You will be travelling from one city (source) to another city (destination) passing through other different cities.

In this case you will be working with graphs. This graph will contain cities (nodes) and cost between them (edges). We will provide you with a corresponding adjacency table where you find connections and cost between cities.

You will have to create a route, step by step, searching for a closer connection between two cities. When you visit one city, looking at the civil registry, you will obtain information about the previous generation of your ancestors.

For example, to start the journey, you can imagine that you and your parents were born in the city X (source city). It means that the civil registry from the city has information about your grandparents, i.e. in which city they were born. So, you will have to go to this city taking into account the best route from source to destination (the cost is the important aspect here). When you visit the city and obtain the information about the next generation, you will continue with your journey. Therefore, you will be creating a road map (see Figure 3).

After the journey, you will have to display the route (road map) and the cost of it (the total cost of passing through all the cities) on the screen in order to check the correctness and present it to the viewer. Please, see Annex to see a sample output.

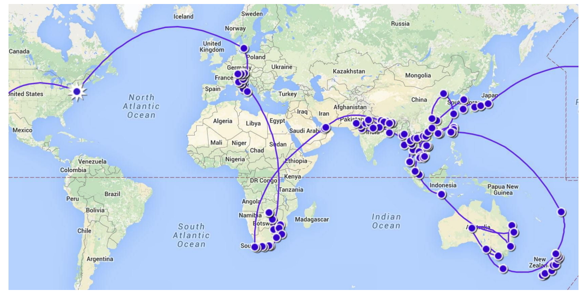

Fig. 3. Example of a road map.

Path searching As you know, the complexity of searching within graphs is very high, which makes it infeasible to calculate the optimal route when the number of cities is moderately high. For this reason, instead of brute force, heuristic strategies are proposed that allow for finding good solutions, even if they are not optimal.

In order to implement a sample heuristic, you have to provide a corresponding function (for example, `RouteSearch()`) that will receive as parameters both, the identifier of the source city and the identifier of the destination city. Analyzing the adjacent matrix, the function must find out if there is a direct connection with the destination city. If yes, this connection will be chosen. If not, the function must choose the connection with the city that has the lowest cost to the current location (what allows you to chose the best cost at a moment and that will approximate you on your route to the destination city). The function will also receive as a parameter the road map (the list of cities of the journey that will be created). Finally, the function will return the total cost of the calculated route.

Data structure We have a certain number of cities N that are connected to each other forming a bidirected graph where all nodes are connected by two edges, one in each direction).

We will provide you with an adjacent matrix, where you can find cities and cost between them. We could interpret it as a graph of cities where each element `i`, `j` specifies the cost of going from city `i` to city `j`. In the case `i == j` (diagonal) we assume an infinite cost (in our case 0).

Thus, we can use this structure to build a route from the source city to the destination city and calculate what its cost is. An example adjacency matrix is presented here:

$$\begin{pmatrix} 0 & 1 & 5 & 9 & 0 & 0 \\ 1 & 0 & 3 & 0 & 1 & 0 \\ 5 & 3 & 0 & 1 & 0 & 0 \\ 9 & 0 & 1 & 0 & 8 & 1 \\ 0 & 1 & 0 & 8 & 0 & 5 \\ 0 & 0 & 0 & 1 & 5 & 0 \end{pmatrix}$$

For creating the road map of the journey, you can use a list structure where you will be adding (for example, to the end of this list) all the cities that you pass during the journey together with the accumulating total cost:

```
struct RoadMap
{
    int city_id;
    int total_cost;
    RoadMap * next;
}
```

You will have to implement three functions: `addToRoadMap()` function that will add a new city to the list, `printRoadMap()` function that will print the current status of the list, as well as `deleteAllRoadMap()` that will delete only once all the cities from the list (before the program finishes its execution).

B. Ancestors' tree

You want to build a tree of your ancestors (Figure 4). To simplify the problem, we can assume the following things:

- You will create a binary tree. The ancestors' tree will be a binary tree so each node can have maximum 2 elements.
- Your parents are the root of the tree.
- You will have to define a data structure for representing this tree: each element of the tree will contain two persons (mother and father), as well two cities: the city where the parents of the mother were born and the city where the parents of the father were born.

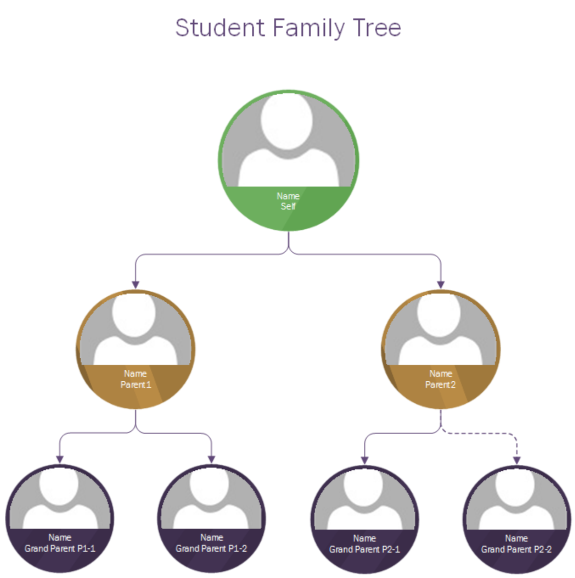

Fig. 4. Binary tree of ancestors where one single node is for both, mother and father.

Tree creation strategies As you know, there are two basic exhaustive strategies that are used for searching: depth-first search (DFS) and breadth-first search (BFS). We will adapt these strategies to create the ancestors' tree:

- In the case of DFS: you will create a tree in a depth way, i.e., first you will create all nodes for mothers' side, and then, if there is no more possibility to go deeper, for fathers' side.
- In the case of BFS: you will create a tree in a breadth way, i.e., first you will create all nodes for the same level (for mother's side, then for father's side) and then you will continue with the next level of the tree.

After the creation of the whole ancestors' tree, you will have to display it on the screen in order to check the correctness and present it to the viewer (the order of creation when using each of the strategies - DFS and BFS, and the content - names for each ancestors). Please, see Annex to see a sample output.

Data structure

In order to build your ancestors' tree, each node of this tree will contain the following fields:

```
struct FamilyTreeNode
{
    char mother_name[10];
    char father_name[10];
    int city_id;
    FamilyTreeNode * mother_parents;
    FamilyTreeNode * father_parents;
}
```

Structure data that represents a civil registry of a given city will be composed of the following fields:

```
struct CivilRegistry
{
    int city_id;
    char city_name[20];
    char mother_name[10];
    char father_name[10];
    int mother_parents_city_id;
    int father_parents_city_id;
}
```

We will also define one additional data structure as it will be necessary to hold the information related to the civil registry - different cities of your ancestors. We will create a static array that will contain CivilRegistry structures and each civil registry will be in a position of the array that is related to the parents' city.

```
CivilRegistry citiesInfo[NUMBER_CITIES];
```

III. Program Execution

Remember that, instead of choosing the "Run Code" option from Visual Studio Code, you can compile and execute the program from the terminal. Just input the following command to compile:

```
gcc -o executable source_file.c
```

and then execute the executable file (with the corresponding arguments, if necessary), for example:

```
program 1
```

In this project you may opt for the basic program execution, but you can also try to find and develop a more advanced level of compilation and execution.

We provide you with 3 different files where we pre-defined certain structures and data to be processed:

- `small.h` - a small number of cities to visit
- `medium.h` - a medium number of cities to visit
- `large.h` - a large number of cities to visit

The basic program execution works for the small.h case where you can find certain data structures already defined, as well as initialized (in particular, the adjacency matrix and information about cities). So, you just simply have to include this header file into the main program.

However, if you want to provide the possibility to include different cases (i.e., small, medium or large), you will have to include a specific header file depending on the chosen case. As all these 3 header files contains exactly the same declaration of structures (with different initial data), you cannot simply include all of them. You have to do the inclusion of these header files and compilation in a special way. Therefore, you have to find out how you can do it in C.

Our tip for you: search for a special directive in C

```
#ifdef
```

The example output and statistics that we can obtain after execution of the program for the small case are presented in Appendix.

IV. Milestones

The goal of the project is to provide all the functionality of the program. You can set the following milestones:

1) Functionalities of graph
   a) searching for the route using a proposed heuristic
   b) printing road map and the total cost
2) Functionalities of ancestors' tree
   a) DFS tree creation
   b) BFS tree creation
   c) printing the final ancestors' tree
3) Specific inclusion of header files and compilation of the program for small, medium and large cases.

V. Recommendations

Here are some recommendations on how to develop the project:

- Properly organize the project with the classmate of your group.
- Comment your code with helpful comments (mostly for yourself). To provide a useful comment, it is very important that it is accompanied by the date and the author.
- Working with strings in C you can use arrays of characters, for example: `char array[20];` or using dynamic arrays and allocating/releasing memory with `malloc()/free()`, respectively. Moreover, while doing operations on strings in C, you will have to use specific functions, for example:
  - `strlen()` - calculates the length of a string
  - `strcpy()` - copies a string to another
  - `strcmp()` - compares two strings
  - `strcat()` - concatenates two strings
- You can use version control tools (there are a few with free options):
  - **Bitbucket** https://bitbucket.org
  - **GitHub** https://github.com/
  - **GitLab** https://about.gitlab.com/

  In this way we will avoid panic attacks and problems due to loss of information, catastrophic decisions, etc.

TABLE I
WEIGHT OF EACH PRACTICAL PROJECT INTO THE SUBJECT'S FINAL GRADE.

Practical project 1	**20%**
Practical project 2	**20%**

VI. Delivery and assessment

The project will be done in pairs and you will have 3 'laboratory' sessions. Take into consideration, that this project will require more hours of dedication, not only the 3 face-to-face sessions. In the last session you will have to explain orally the solution and implementations carried out. After the last session you will have a short period (3 days) to hand in a report or fixing small errors. This means that by the last session you should already have the whole implementation and report almost finished and only check/fix/add any small details or observations you may have and comment them with the professors.

A. Delivery via Virtual Campus

The delivery will be done through the Virtual Campus until the indicated date. It's enough if only one of the members of the group deliver the project. You have to attach: the report in pdf format and the source code (`*.h, *.c`) compressed in one `.zip` file.

B. Assessment

The final grade of the practical projects represents 40% of the subject's final grade. It is an essential requirement to have a grade of 5 or higher in the practical projects to pass the subject. You have 2 deliveries (two practical projects) during the semester; this is the second one. The contribution of each project to the final subject's grade is shown in table I.

The following criteria will be used to calculate the grade of this second project:

- [6 points] The 'correct' working of the code. There are 2 functionalities:
  - [3 point] Map route generation
  - [3 points] Ancestors' tree generation
- [0,5 points] Possibility of execution for different cases (small, medium, large).
- [0,5 points] Code style and comments.
- [2 points] Report

- [1,5 points] Description of the strategy in an accurate and reasoned way.
  - [0,5 points] Criticism of the problems that arose and the solutions found during the development of the project.
- [1 point] Continuous assessment of attendance and participation in class [INDIVIDUAL PART].

VII. Conclusions

We have already presented you the second practical projects, the last that you will develop this semester. Please, contact us if you have any doubts or questions: `Manuel.Montoto@uab.cat` and/or `Anna.Sikora@uab.cat`. Do not hesitate to ask for meetings with us!

Appendix

Here we present you a sample output of the program execution for the small input data

```
This is a small case of the program

Ancestors' tree:

BFS -> Names:
Maria and Jordi (Barcelona)
-> Louise and Pol (Paris)
-> Eva and Albert (Zurich)
->-> Anna and Kazimierz (Varsovia)
->-> Agnese and Leonardo (Rome)
->-> Madalena and Lourenço (Lisboa)
->-> Amber and Fin (Amsterdam)

Partial road map:
Barcelona-Madrid-Paris 180
Paris-Berlin-Viena-Zurich 150
Zurich-Viena-Varsovia 140
Varsovia-Berlin-Viena-Zurich-Rome 230
Rome-Zurich-Viena-Berlin-Paris-Lisboa 340
Lisboa-Barcelona-Madrid-Rome-Amsterdam 340

Total Road Map:
Barcelona-Madrid-Paris-Berlin-Viena-Zurich-
Viena-Varsovia-Berlin-Viena-Zurich-Rome-
Zurich-Viena-Berlin-Paris-Lisboa-
Barcelona-Madrid-Rome-Amsterdam

Total cost: 1380

----------------------------------
DFS -> Names:
Maria and Jordi (Barcelona)
-> Louise and Pol (Paris)
->-> Anna and Kazimierz (Varsovia)
->-> Agnese and Leonardo (Rome)
-> Eva and Albert (Zurich)
->-> Madalena and Lourenço (Lisbon)
->-> Amber and Fin (Amsterdam)

Partial road map:
Barcelona-Madrid-Paris 180
Paris-Berlin-Varsovia 170
Varsovia-Berlin-Viena-Zurich-Rome 230
Rome-Zurich 70
Zurich-Viena-Berlin-Paris-Lisbon 270
Lisbon-Barcelona-Madrid-Rome-Amsterdam 340

Total Road Map:
Barcelona-Madrid-Paris-Berlin-Varsovia-
Berlin-Viena-Zurich-Rome-Zurich-Viena-
Berlin-Paris-Lisbon-Barcelona-Madrid-
Rome-Amsterdam

Total cost: 1260
```