Computational Thinking and Programming – A.Y. 2020/2021

Written examination – 14/06/2021

Given name: _____

Family name: _____

Matriculation number: _____

University e-mail: _____

Group name: _____

Is it your first try? Yes | No

The examination is organised in three different sections:

- Section 1: basic questions [max. score: 8]. It contains four simple questions about the topics of the whole course. Each question requires a short answer. Each question answered correctly will give you either 2 points (full answer) or 1 point (partial answer).
- Section 2: understanding [max. score 4]. It contains an algorithm in Python, and you have to report the particular results of some of its executions according to specific input values.
- Section 3: development [max. score 4] It describes a particular computational problem to solve, and you are asked to write an algorithm in Python for addressing it.

You have 1 hour and 30 minutes for completing the examination. By the final deadline, you should deliver only the original text (i.e. this document) with the definitive answers to the various exercises that must to be written with a pen – pencils are not permitted. You can keep all the draft papers that you may use during the examination for your convenience – blank sheets will be provided to you on request.

Section 1: basic questions

1 – Which of the following sentences are true statements?

- The three main kinds of programming languages are machine language, low-level programming languages, and high-level programming languages
- Noam Chomsky invented the Analytical Engine
- The Turing machine can simulate any algorithm
- Brute force algorithms use always a recursive approach
- A list is a countable collection of non-repeatable elements

2 – Consider the following Python function:

```
def g(x):
    r = set()
    idx = 0
    for it in x:
        if it not in r:
            r.add(idx)
        idx = idx + 1
    return r
```

What is the value returned by `g([5, 7, 7, 2, 5, 7])`?

3 – Write down a small function in Python that takes in input a string, an integer, and a boolean, and returns the character of the string at the index specified by the input integer if the input boolean is *True*, otherwise it returns the last character in the input string.

4 – Describe what is a *recursive function* and which are their basic components.

Section 2: understanding

Consider the following functions written in Python:

```
from collections import deque

def f(m_string):
    c = 0
    s = deque(m_string)
    while s:
        cur = int(s.pop())
        if cur % 2:
            c = c + 1

    m_len = len(m_string)
    m_col = list()
    while c < m_len - c:
        m_col.append(m_string[c])
        c = c + 1

    for i, d in enumerate(reversed(m_string)):
        if d not in m_col:
            if i < len(m_col):
                m_col.insert(i, d)
            else:
                m_col.append(d)

    return m_col
```

Consider the variable `my_mat_string` containing the string of your matriculation number. What is the value returned by calling the function `f` as shown as follows:

```
f(my_mat_string)
```

Section 3: development

A* is a search algorithm for weighted graphs. Starting from a specific `start` node of a graph, it aims to find a path to the given `goal` node having the smallest cost (i.e. the least distance travelled considered the sum of the weights of all the edges crossed from `start` to `goal`). Typical implementations of A* use a priority queue to perform the repeated selection of minimum cost nodes to expand, computed using the following formula:

```
f(n) = g(n) + h(n)
```

where `n` is the next node on the path, `g(n)` is the cost of the path from the `start` node to `n`, and `h(n)` is a heuristic function (`h` is provided as input of the algorithm) that estimates the cost of the cheapest path from `n` to the `goal`.

At the very beginning of the algorithm, the priority queue is initialised with the `start` node, and `f(start)` is set to `h(start)`. At each step of the algorithm, the node `n` with the lowest `f(n)` value is removed from the priority queue, the `f` and `g` values of its neighbours are updated accordingly, and these neighbours are added to the priority queue. The algorithm continues until a removed node (thus the node with the lowest `f` value out of the queue) is the `goal` node, and then `g(goal)` is returned.

Write an algorithm in Python – `def a_star(graph, start, goal, h)` – which takes in input a directed `graph` (defined according to the `networkx` library), a `start` node in the graph, a `goal` node in the graph, and the function `h` used to compute `f`, and returns the cost of the cheapest path from `start` to `goal`. All the edges in `graph` have specified an attribute `weight` (an integer) representing the cost to traverse that edge.