Algorithmics and complexity

TD 1/7 – Graph search

Training exercises

Note: The solution elements given here are not complete. Their purpose is only to guide you. We encourage you to write a proper answer, as you would do for the exam. If any question remains, feel free to ask your tutorial supervisor for help.

Exercice 1 : Find the exit!

This first exercise is relatively easy. You should be able to answer all questions alone, as in exam conditions.

In the first course, we saw depth-first search and breadth-first search algorithms to decide whether a node is accessible or not. However our algorithm cannot be used directly to find the exit path in a labyrinth. The goal of this exercise is to achieve this task.

Question 1

Modify the BFS algorithm given in the course in order to store in a python dictionary the preceding node (n) of any newly added neighbour (v). The algorithm must return this dictionary instead of a boolean value.

Solution elements :

Start with an empty `parent` dictionary and add an entry when a neighbor is added to the queue, if it is not already in the dictionary, to associate its ancestor.

Write the full algorithm in Python, don't content yourself with a "oh yes, I have understood"!

Question 2

Now, write a second algorithm that computes the path between s and t using the data structure given by the previous algorithm. The result is a list of vertices with s as first element and t as last.

Solution elements :

Starting from t, write a loop that stops when it reaches s and that looks in the dictionary for the current vertex' parent.

Question 3

What is the time complexity of your path-finding algorithm?

Solution elements :

You should still be in $\mathcal{O}(|E|)$. If you have doubts about your answer, discuss it with your tutorial supervisor.

Exercise 2 : Topological ordering

An industrial process, a project, a schooling, etc. are composed of tasks (modules, etc.). Some tasks can be performed only after performing some other tasks. We want to establish the order of execution of all the tasks while respecting these constraints.

Topological ordering of a directed graph $G = (V, E)$ is a linear ordering of its vertices V (a list) $v_1, v_2, \ldots, v_n$ such that for every directed edge (v_i, v_j) we have $i < j$.

Question 1

Propose an algorithm to compute a topological ordering for a DAG.

Solution elements :

There exists several famous algorithms to solve this problem. One was proposed by Kahn in 1962, another by Tarjan in 1979, and other algorithms in the 80s can even support parallel computing. You can find some details about these algorithms on Wikipedia:

`https://en.wikipedia.org/wiki/Topological_sorting`

The Tarjan algorithm is based on depth-first search, using a stack to order the nodes. Indeed, if G is a DAG, all you have to do is to depth-first search the graph and stack the vertex once all its successors have been visited (i.e. after the recursive calls on these successors). Unstacking the nodes simply gives you the topological order.

The Kahn algorithm is more direct: a vertex without ancestor (there is always one in a DAG) will be the first node (number 1). You remove this vertex and all its outgoing edges. Start again to find number 2, and continue...

We advise you to write these algorithms yourself, in Python, without looking at the solution.

Question 2

Prove the following theorem:

$$G \text{ admits a topological ordering} \iff G \text{ is a DAG.}$$

Solution elements :

In one direction, you can use ”Reductio ad absurdum” (consider the vertex with the lowest rank in the supposed cycle and its immediately preceding neighbor) and in the other direction yous can apply a mathematical induction similar to the approach used in Kahn algorithm (the hypothesis in the inductive step is that any DAG with n vertices has a topological order).

We recommend that you write these proofs properly as a preparation for the exam.