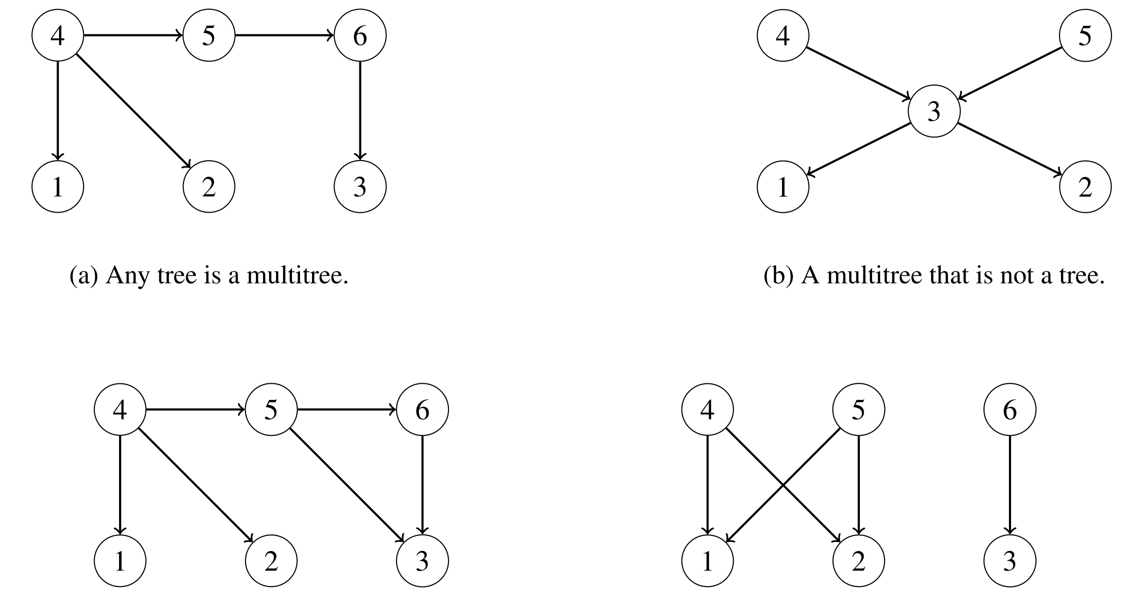
Enumeration lab

Enumeration in multitrees

This lab concerns the enumeration of reachable nodes, leaves or paths in graphs. Because the problem in general is more complicated, we will look at enumeration problems in *multitrees* which are a generalization of trees and a restriction of directed acyclic graphs. Formally multitrees are directed graphs that are acyclic and in which there is at most one path between every pair of nodes.

The notion of multitree may be confusing at first. We provide a few examples of directed graphs in Figure **??** below, and discuss whether they are multitrees or not.



(a) Any tree is a multitree.

(b) A multitree that is not a tree.

(c) A directed acyclic graph that is not a multitree.

(d) Multitrees are more general than forests.

FIGURE 1 – Illustrations of graphs, some of which are multitrees.

All the multitrees that we will consider in this lab will be **binary multitrees**, which means that the output degree of all nodes is at most 2.

A first enumeration task

Enumeration task 1 : node enumeration. In the first problem we study, we are given in the preprocessing phase a multitree $\mathcal{T}$ that we can preprocess and then, in a second phase, we are given a node n in $\mathcal{T}$ and we need to enumerate all the nodes reachable from n. For instance, on graph **??**, if we are given the node 4 we should output 3, 2, and 1 (the order is not important).

Question 1. *What are the characteristics of the following enumeration algorithm (delay, incremental and total complexities) ?*

```python
# neighbors is a list such that neighbors[v] is a python list
# of nodes w such that (v,w) is an edge

def enumerate_node_dfs(node):
  output(node)
  for nxt in neighbors[v]:
    enumerate_node_dfs(nxt)
```

The above algorithm can be seen as a Depth First Search algorithm. Such an algorithm can be implemented recursively (as above) or without recursion using a stack.

Question 2. *Propose a non-recursive version using a stack. What are the characteristics of your algorithm ? We can expect the stack to behave as seen in class, i.e. all operations are $O(1)$.*

Question 3. *Implement this algorithm in the function* `enumerate_node_stack(node)`.

A second enumeration task

Enumeration task 2 : leaf enumeration. In a multitree, a *leaf* is a node that has output degree 0. The second enumeration task that we consider is that we are given a multitree $\mathcal{T}$ that we can preprocess and then we are given a node n in $\mathcal{T}$ and we need to enumerate the leaves reachable from n. For instance, on graph **??**, if we are given the node 4 we should output 1 and 2 (once again, the order is not important).

Question 4. *Propose a recursive algorithm inspired by Question* **??**. *What are its characteristics ?*

Question 5. *Implement this algorithm in the function* `enumerate_leaf_dfs(node)`.

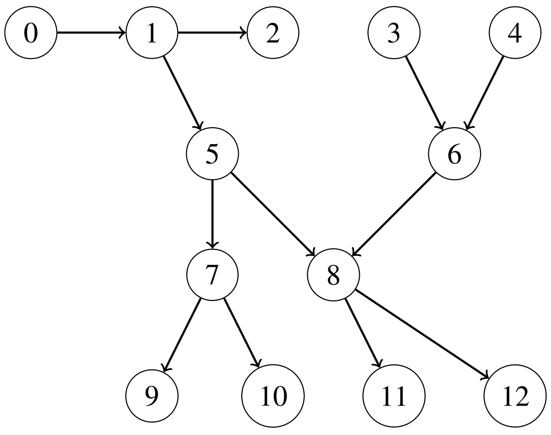
Question 6. *Propose a non-recursive algorithm inspired by Question* **??**. *What are its characteristics ?*

We will now explain how to improve on those algorithms. We will work with a set of nodes to explore. At each step the algorithm will take one node n from this set, output a leaf reachable from n and then add back some nodes to the set of nodes to explore.

Note that this is somewhat similar to the DFS algorithm to enumerate nodes in a graph implemented with a stack. The main difference are that in the previous algorithm we were outputting the node n node (and not a leaf reachable from n) and that the set of nodes to add back was just the set of children of n (here it is more complicated).

For the algorithm to work correctly, we can output any leaf f reachable from n but we need the set of nodes S to add back to be such that the set of leaves reachable from n is the disjoint union of $\{f\}$ and the set of leaves reachable from nodes in S.

Let us consider an example with the graph below. If the algorithm consider $n = 1$, we could return the leaf $f = 9$ and the set of nodes that are left to be explored would $S = \{10, 8, 2\}$. Note that this is not the only solution we could return $f = 2$ and $S = \{5\}$ or $S = \{7, 8\}$, etc.



Our goal is to have an algorithm with preprocessing and constant delay enumeration. As we will see, there are two difficulties :

— first we need to be able to retrieve from a node n in $O(1)$ both a reachable leaf and a set of nodes that remain to be explored ;

— second, we need a way to represent the set of nodes such that we can add back the nodes to explored in $O(1)$.

Let us start by tackling this second point :

Question 7. *Propose (give the general idea and the pseudo-code) a* ***functional*** *datastructure that can represent a "collection" of elements of type α and support the following operations in $O(1)$:*

— *test whether the collection is empty,*

— *add a list of elements (lists are represented as functional lists),*

— *retrieve an element (we suppose the collection is not empty).*

The term collection here means a set of elements where the order is unimportant and where we don't care about uniqueness (i.e. we will never try to add twice an element).

Question 8. *Implement this algorithm in the file* `Collection.py`.

Now, let us try to solve the first point.

Question 9. *Explain how we can compute during the preprocessing an array that associates each node n (we suppose nodes are numbered $0 \ldots N$) with a pair of a leaf f and a set S such that each leaf reachable from n except f is reachable from exactly one node in S.*
Tips : *Since the graph is acyclic you can work recursively and use memoization. Use the fact that all nodes have output degree 2.*

Question 10. *Give the idea (or pseudo-code) for the complete algorithm to solve the enumeration problem. Distinguish clearly the preprocessing part and enumeration part.*

Question 11. *Implement this algorithm in the function* `enumerate_leaf_fast(node)`.

Path enumeration

Enumeration task 3 : path enumeration. The problem that we will concern with now is that we are given a multitree $\mathcal{T}$ that we can preprocess and then we are given a node n in $\mathcal{T}$ and we need to enumerate the paths starting from n and ending in a leaf. Since enumerating all paths at the list of nodes crossed in each path is too easy (this is very similar to enumerating nodes) we will not enumerate the paths but rather we will suppose that each node n has a label $\ell(n)$ taken from some set $\pm \cup \{\epsilon\}$ and for each path $n_1 \to \cdots \to n_k$ we will output the string $\ell(n_1) \ldots \ell(n_k)$ where all elements of Σ as seen as chars and ϵ designates the empty word.

Question 12. *Propose a recursive algorithm to solve this enumeration task, what are its characteristics ? What happens if no label is ϵ ?*

Question 13. *Implement this enumeration algorithm in the function* `enumerate_path_dfs(node)`.

We want to adapt the algorithm from question **??** to work with our problem. For that we want an algorithm that can enumerate with constant delay, from a node n all nodes v reachable from n such that $\ell(v) \neq \epsilon$ and each node u on the path from n to v is such that $\ell(u) = \epsilon$.

Question 14. *Explain how such an algorithm would work, you can provide the pseudo code for this but you are invited to explain how you can modify an algorithm seen in class or earlier in this lab to achieve this result.*

Question 15. *Explain how can solve the path enumeration problem with constant delay.*

Question 16. *Implement this enumeration algorithm in the function* `enumerate_path_fast(node)`.

Bonus question

Question 17. *Assuming that we cannot multiply boolean matrices in time $O(n^2)$, show that there cannot exist a linear preprocessing and constant delay algorithm for the enumeration of leaves.*