Remarks After the lab session, send your code to my e-mail `ni-luh-dewi.sintiari@ens-lyon.fr`. Please send it in one single file named `TP4-nom_prenom`, and in the email subject write "TP4 assignment".

Prüfer codes

A tree $T = (V, E)$ on $n \geq 3$ vertices can be represented by its Prüfer code. A *Prüfer code* is a sequence $(a_0, a_1, \ldots, a_{n-3})$ such that $a_i \in \{0, 1, \ldots, n-1\}$ for all i. Figures 2–6 present some trees and the corresponding Prüfer codes.

Exercise 1.

(a) Given n, what is the number of the corresponding Prüfer codes? How does it relate to the number of trees on n vertices?

(b) Figure 1 gives a pseudocode of a function that converts a Prüfer code into a tree. Implement it as a function `prufer_to_tree(P)`. How many operations do you need to find the element v from lines `1` and `2`? (Note that finding an element in a list requires to iterate through it, and hence takes linear time.) What is the overall complexity of your function? To achieve a better performance, create a list C of length n such that $C[k]$ is the number of occurrences of k in P. Update this list at each passage of the loop in order to find the element v (line `1`) in linear time.

(c) We now have a way to study random trees: we can take a random Prüfer code and construct the corresponding tree. Do some experiments to estimate what is the average diameter of a random tree. How fast does it grow with n? (Hint: it grows like n^α for some $0 < \alpha \leq 1$. We want to know the value of α.)

(d) There is a different way to study random trees: we take all $n(n-1)/2$ possible edges of a graph with n vertices, permute them randomly, and use the Kruskal algorithm to find a spanning tree (the weights of edges are equal to 0). Estimate the average diameter of a tree obtained in this way. Is this the same model as above?

(e) Write a function `tree_to_prufer(E)` that takes a tree as an input and outputs the corresponding Prüfer code. (Hint: consider the leaf of the tree with the smallest number and look at its neighbor.)

(f) The element v from line `1` of the algorithm can be found in $O(\log n)$ time (which is faster than linear). This can be achieved using a data structure known as a *binary heap*, and implemented in the `heapq` module of Python. Read about binary heaps and implement them in your function.

Data: A Prüfer code P (given as a list of numbers)
Result: A list of edges E of a tree
$E \leftarrow [];$
$n \leftarrow \mathrm{len}(P) + 2;$
$L \leftarrow [0, 1, \dots, n-1];$
for $i \in \{0, 1, \dots, n-3\}$ **do**
1 $\quad v \leftarrow$ the smallest number that is in L but not in $P[i : n-2];$
2 $\quad L.\mathrm{remove}(v);$
$\quad E.\mathrm{append}\big((v, P[i])\big);$
end
$E.\mathrm{append}\big((L[0], L[1])\big);$

Figure 1: Computing a tree from its Prüfer code.

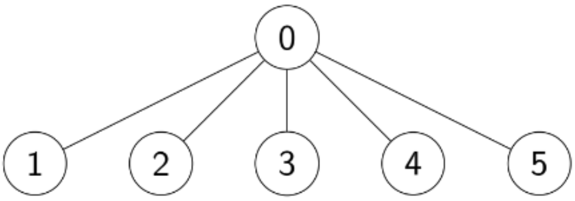

Figure 2: Tree with code $(0, 0, 0, 0)$.

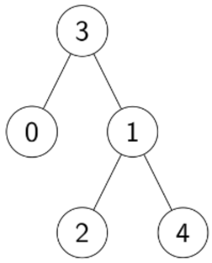

Figure 3: Tree with code $(3, 1, 1)$.

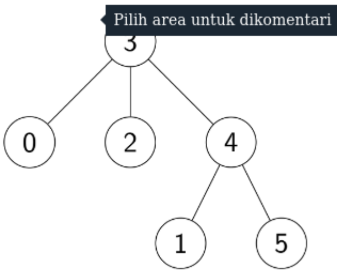

Figure 4: Tree with code $(3, 4, 3, 4)$.

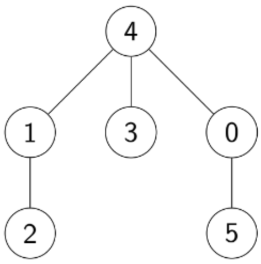

Figure 5: Tree with code $(1, 4, 4, 0)$.

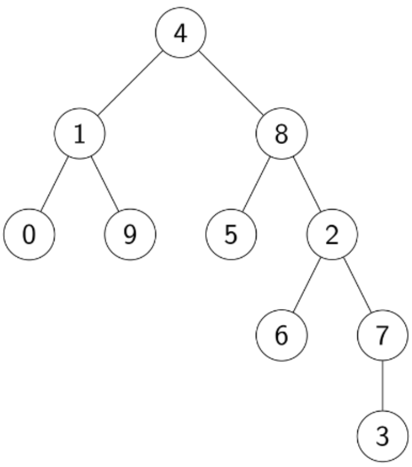

Figure 6: Tree with code $(1, 7, 8, 2, 2, 8, 4, 1)$.