Un arbre maximier de Noël prêt pour la récolte !

Tri maximier

1 Préambule sur les graphes

1.1 Rappels sur les graphes : définitions

On rappelle qu'un ***graphe*** $G = (S, \mathscr{A})$ est la donnée d'un ensemble fini $S = \{s_1, \cdots, s_n\}$ de ***sommets*** aussi appelés ***nœuds*** et d'une partie $\mathscr{A} = \{a_1, \cdots, a_p\}$ de S^2 d'***arêtes***. Les arêtes relient donc deux sommets et un graphe apparaît comme un ensemble fini de points reliés les uns aux autres à l'aide d'arêtes.
On dit qu'un graphe est ***connexe*** si pour tout couple (s_i, s_j) de sommets, on peut trouver un chemin d'arêtes $\gamma = (r_0 = s_i, r_1, \cdots, r_{n-1}, r_n = s_j)$, pour tout entier $k \in [\![0, n-1]\!]$, le couple (r_k, r_{k+1}) étant une arête dans $\mathscr{A}$. On dira alors que le chemin d'arêtes $\gamma = (r_0, \cdots, r_n)$ relie le sommet r_0 au sommet r_n. L'entier n est appelé la ***longueur du chemin*** γ.
On dit qu'un sommet r est une ***racine*** d'un graphe si pour tout $s \in S$, il existe un chemin d'arêtes γ reliant r à s.
Pour toute arête $(s_i, s_j) \in \mathscr{A}$, on dit que le sommet s_j est un ***successeur du sommet*** s_i ou que le sommet s_j est un ***sommet-fils du sommet*** s_i, ou que le sommet s_i est un ***prédécesseur du sommet*** s_j ou que le sommet s_i est un ***sommet-père du sommet*** s_j.
On dit qu'un graphe est un ***arbre***, si on ne peut trouver dans ce graphe, un chemin d'arêtes $\gamma = (r_0, \cdots, r_n)$ avec $n \geqslant 1$ tel que $r_0 = r_n$. Autrement dit, un arbre est un graphe dans lequel il n'existe aucune boucle possible formée par un chemin d'arêtes.
Dans un arbre connexe, on dispose d'une ***distance entre les sommets*** : pour tous sommets s_i et s_j, il existe un chemin d'arêtes γ de longueur minimale reliant s_i à s_j. Cette distance $d(s_i, s_j)$ vérifie les trois propriétés suivantes :

- pour tous sommets a et b, $d(a, b) \geqslant 0$ et si $d(a, b) = 0$, alors $a = b$
- pour tous sommets a, b et c, alors :

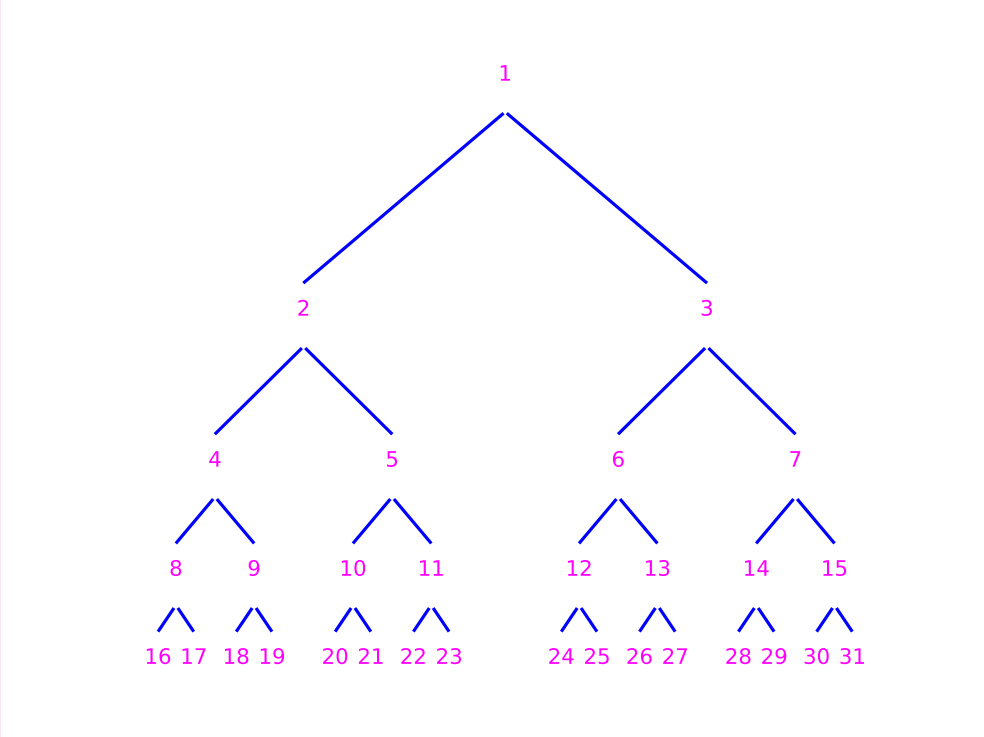
$$d(a, c) \leqslant d(a, b) + d(b, c).$$

On peut donc parler de sphère de centre ω et de rayon r comme l'ensemble :

$$\Big\{ s \in S \mid d(\omega, s) = r \Big\}.$$

On appelle ***profondeur d'un arbre*** de racine r, la distance maximale séparant les sommets de la racine, cette distance étant augmentée d'une unité.
Dans un arbre admettant comme racine r, alors le sommet r est le seul sommet n'admettant pas de sommet-père. Dans un arbre, il n'y a qu'une seule racine car sinon, on pourrait former une boucle d'arêtes entre deux racines différentes de l'arbre.
On appelle ***niveau du graphe*** , toute sphère de centre r et de rayon donné, la distance mesurée entre deux sommets étant le nombre d'arêtes minimal présent dans un chemin

reliant ces deux sommets. Le premier niveau, celui de numéro 0, est uniquement constitué de la racine de l'arbre. Le deuxième niveau, celui de numéro 1 comporte le(s) sommet(s)-fils de la racine.
On dit qu'un arbre est ***binaire*** si tout sommet admet au maximum deux successeurs.
Dans un arbre, on appelle ***feuille***, tout sommet n'admettant aucun sommet-fils. Dans un arbre de profondeur n et de racine r, les sommets s tels que $d(r,s) = n-1$ sont des feuilles. La réciproque est fausse : il peut exister des feuilles à distance de r strictement inférieure à $n-1$.
On appelle ***arbre binaire complet***, tout arbre tel que pour tout sommet s qui n'est pas une feuille, alors le sommet s admet exactement deux sommets-fils.
Voici un exemple d'arbre binaire complet, de profondeur égale à 5 :

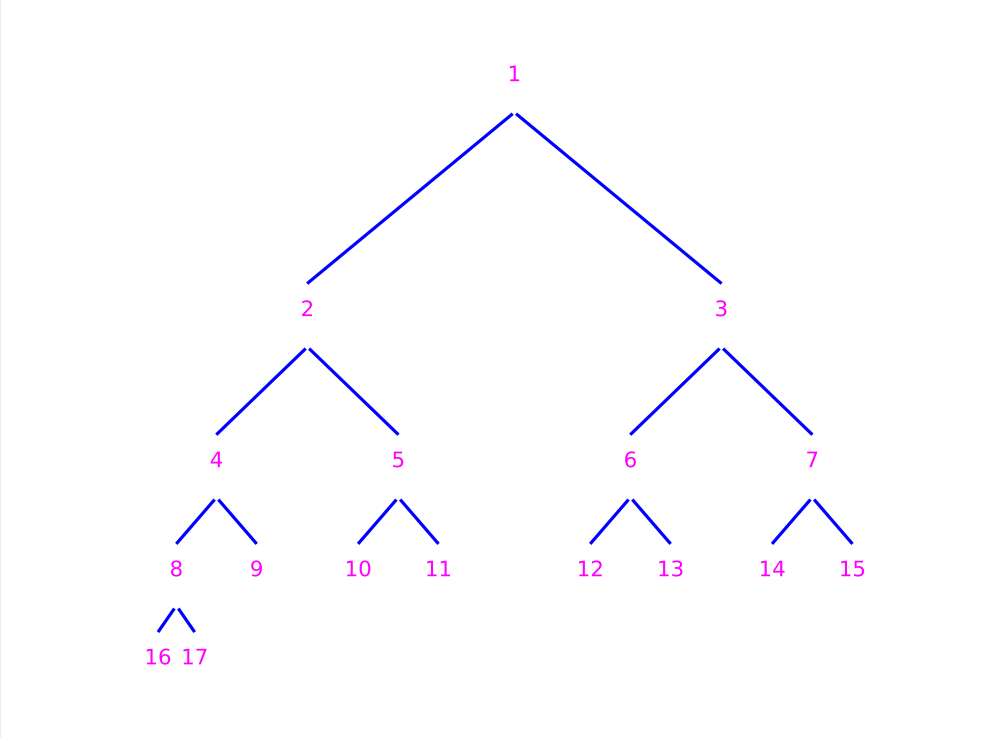


Dans un arbre binaire complet de racine r, chaque niveau k – égal à la sphère de centre r et de rayon k – contient exactement 2^k sommets (ou bien aucun sommet si l'entier k est trop grand, supérieur ou égal à la profondeur de l'arbre et donc strictement supérieur à la distance maximale entre la racine et les sommets de l'arbre).
Un arbre binaire complet de profondeur n admet exactement 2^{n-1} feuilles. En effet, dans un tel arbre, les feuilles sont exactement les sommets à distance maximale de la racine (égale à $n-1$) de la racine. Plus précisément, pour tout $k \in [\![1, n]\!]$, chaque niveau k contient exactement 2^{k-1} sommets et l'arbre binaire complet de profondeur n contient finalement :

$$\sum_{k=0}^{n-1} 2^k = 2^n - 1 \text{ sommets.}$$

On dit qu'un arbre binaire est ***semi-complet*** , si – en notant n la profondeur de l'arbre et r la racine de l'arbre, tous les nœuds s tels que $d(r,s) < n-1$ admettent exactement

deux sommets et le dernier niveau est partiellement rempli, les feuilles du dernier niveau étant placées à gauche. Autrement dit, tous les niveaux k sauf le dernier comportent exactement 2^k sommets et le dernier niveau comporte des sommets placés sur la gauche. Voici un exemple d'arbre binaire semi-complet de profondeur encore égale à 5 :



Dans la suite, on utilisera des arbres binaires semi-complets pondérés pour signifier que chaque nœud ou sommet de l'arbre est associé à une valeur numérique. La page de couverture laisse apparaître un tel arbre pondéré de profondeur égale à 4.

1.2 Préambule fruitier

Il y a les pommiers qui produisent des pommes, il y a des cerisiers qui produisent des cerises, il y a des bananiers qui produisent des bananes, et puis il y a des maximiers qui produisent des maximums.
On appelle ***maximier***, un arbre binaire connexe et semi-complet pondéré tel que la valeur associée à chaque noeud est toujours supérieure ou égale à la valeur de ses fils. On en déduit que dans n'importe quel maximier, la valeur maximale se trouve à la racine de l'arbre et la valeur minimale se trouve sur l'une des feuilles.
On peut remarquer que n'importe quel sous-arbre d'un maximier est encore un maximier. Au lieu de maximier, on parle aussi de ***tas***.

2 Principes du tri maximier

2.1 Données du problème et objectifs

On considère une liste $L = [a_0, \cdots, a_{n-1}]$ de valeurs numériques. L'objectif est de présenter un algorithme de tri de la liste L. On souhaite que ce tri soit de bonne complexité....

2.2 Principes de l'algorithme

On se donne une liste $L = [a_0, \cdots, a_{n-1}]$ de valeurs numériques.

2.2.1 Plantation du maximier

On commence par lire les occurrences de la liste L de la gauche vers la droite et à chaque lecture, on crée un sommet supplémentaire dans un arbre initialement vide, de façon que l'arbre en construction soit toujours un arbre maximier.
Pour ce faire, on commence par placer la première occurrence a_0 en la racine de l'arbre – arbre réduit alors à un seul sommet.
Étant donnée une étape de construction de l'arbre, à la lecture de l'occurrence a_k de la liste L, on crée une nouvelle feuille où l'on insère a_k, puis on procède à une percolation pour échanger éventuellement les valeurs numériques d'un sommet s avec celle de son sommet-père et ainsi de suite avec les sommets aux niveaux plus hauts, pour finalement avoir un maximier.
À la fin de la lecture de la liste L, on dispose d'un arbre binaire maximier A admettant exactement n sommets.
La page de couverture présente un arbre maximier obtenu suite à cette plantation, en liaison avec la liste de valeurs numériques suivante :

$$L = \left[\gamma, 833, \binom{10}{7}, \sqrt{2025}, \pi, \nu_2(18), \zeta(2), \ln(17!), \varphi, 1/3, e^{-6}, 0\right].$$

2.2.2 Cueillette des fruits

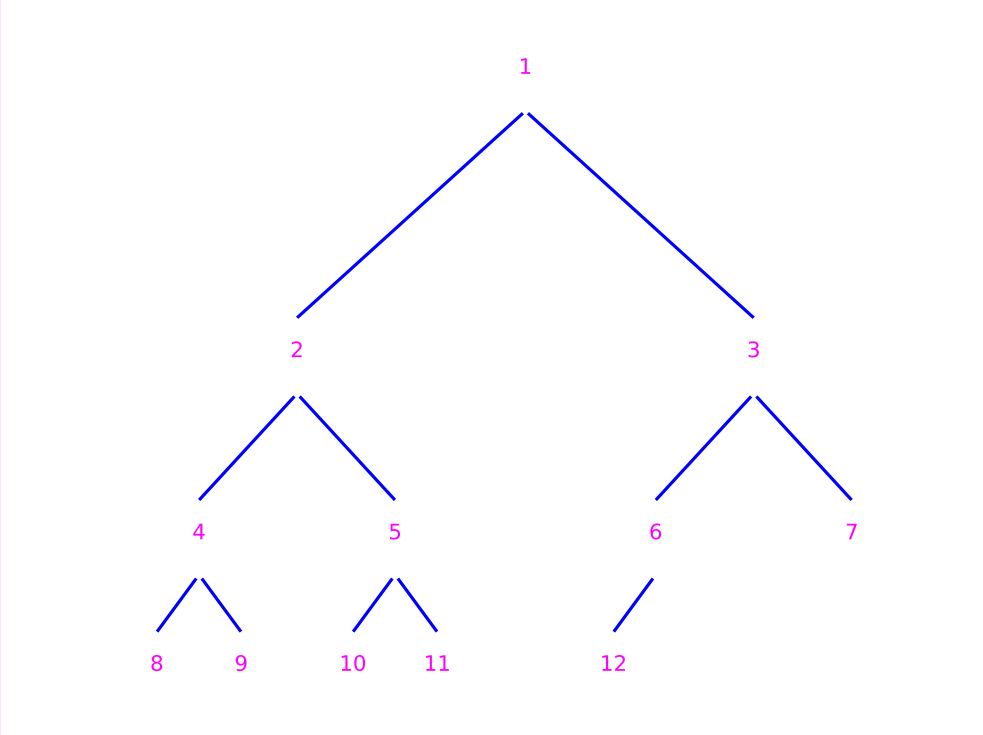
À partir de cet arbre maximier A fraîchement planté, on procède maintenant à la cueillette des fruits, c'est-à-dire à la cueillette des maxima.
Le fruit du maximier est toujours à cueillir à la racine : c'est là que se trouve le maximum. On considère donc une liste vide M. On pioche le maximum dans l'arbre A et on le place au début de la liste M. Une fois ce fruit cueilli, à chaque étape, on choisit le fils de la racine qui admet le plus grand nombre, ce nombre prenant la place de la racine. La valeur transitée laisse vide l'un des sommets-fils, qui prend alors la valeur maximale entre ses deux sommets-fils, ainsi de suite.
Une fois tous ces changements effectués, l'une des feuilles se trouve sans valeur associée car sa valeur associée initiale a transité vers un autre sommet. Cette feuille dépourvue de valeur n'est pas forcément la feuille la plus à droite dans l'arbre : l'arbre pondéré obtenu est toujours un arbre connexe mais pas nécessairement semi-complet.

On considère maintenant la feuille la plus à droite, comportant un certain nombre. On prélève ce nombre v, on supprime la feuille associée et on remplit la feuille vacante (sans pour l'instant de valeur associée) avec ce nombre v en faisant jouer la percolation pour effectuer les changements adéquats (à savoir échanges des valeurs entre sommets adjacents et ce autant de fois que nécessaire pour garantir un arbre maximier) pour obtenir finalement un arbre maximier avec un élément de moins.
On réitère l'opération avec une nouvelle cueillette, en remplissant la liste M par la gauche.
La liste M obtenue une fois que l'arbre maximier est vide est la liste L triée.

3 Encodage du tri maximier

3.1 Modélisation des arbres maximier

On modélise les arbres par des listes. Par exemple, l'arbre semi-complet suivant :



sera modélisé par la liste : $A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$.
Ainsi, une liste $A = [\omega_0, \cdots, \omega_{n-1}]$ à n composantes correspond à un arbre binaire semi-complet comportant n sommets, les sommets dans l'ordre de lecture (on commence la lecture de haut en bas, puis par niveau de gauche à droite) sont associés dans cet ordre aux valeurs $\omega_0, \omega_1, \cdots, \omega_{n-1}$.
Si $A = [\omega_0, \cdots, \omega_{n-1}]$ est une liste associée à un arbre binaire semi-complet, si k est un entier entre 1 et $n-1$, l'élément $A[k]$ correspond à la valeur du $k^{ème}$ sommet que l'on

note s_k et ce sommet s_k admet un sommet-père dont la valeur associée est égale à :

$$A\left[\left\lfloor\frac{k-1}{2}\right\rfloor\right].$$

Inversement, si k est un entier entre 0 et $n-1$, l'élément $A[k]$ est associé à un sommet s_k. Ce sommet s_k est soit une feuille, soit il admet un seul sommet-fils (gauche), soit il admet deux sommets-fils.
Le cas échéant, le sommet-fils gauche du sommet s_k est associé à la valeur $A[2k+1]$ et le sommet-fils droit est associé à la valeur $A[2k+2]$.

3.2 Encodage de la plantation du maximier

Voici un script qui permet la mise à jour du maximier lorsqu'on insère une nouvelle feuille :

```
def Pousse_Maximier(Arbre,Feuille,Place) :
    n=len(Arbre)
    if Place==n :
        Arbre.append(Feuille)
    else :
        Arbre[Place]=Feuille
    Test_Modif=True
    while Test_Modif and Place>0 :
        Test_Modif=False
        Noeud_Pere=(Place-1)//2
        if Arbre[Noeud_Pere]<Arbre[Place] :
            Arbre[Noeud_Pere],Arbre[Place]=Arbre[Place],
    Arbre[Noeud_Pere]
            Place=Noeud_Pere
            Test_Modif=True
    return Arbre
```

Voici maintenant un script qui procède à la plantation proprement dite de l'arbre :

```
def Plantation_Maximier(L) :
    Arbre=[]
    for k in range(len(L)) :
        Pousse_Maximier(Arbre,L[k],k)
    return Arbre
```

3.3 Encodage de la cueillette

Voici un script qui code la cueillette du fruit :

```
def Cueillette(Arbre) :
    k=len(Arbre)-1
    if k==0 :
        return []
    else :
        Place=0
        while 2*Place+2 <= k  :
            Noeud_Fils=2*Place+1
            if Arbre[Noeud_Fils]>Arbre[Noeud_Fils+1] :
                Place_Fils=Noeud_Fils
            else :
                Place_Fils=Noeud_Fils+1
            Arbre[Place]=Arbre[Place_Fils]
            Place=Place_Fils

        if Place==k :
            return Arbre[:-1]
        else :
            return Pousse_Maximier(Arbre[:-1],Arbre[-1],
    Place)
```

3.4 Encodage du tri maximier

Voici un script qui code finalement le tri voulu :

```
def Tri_Maximier(L) :
    Arbre=Plantation_Maximier(L)
    M=[]
    while Arbre!=[] :
        M=[Arbre[0]]+M
        Arbre=Cueillette(Arbre)
    return M
```

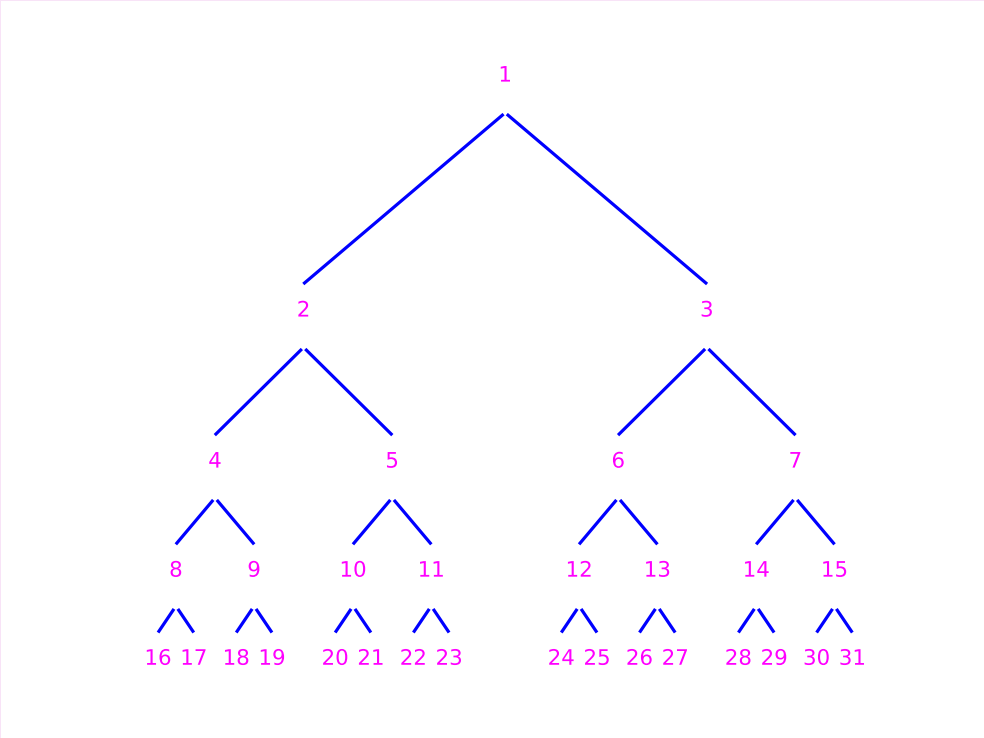
4 Quelques remarques sur le tri maximier

4.1 Placement des nœuds et décomposition binaire

Étant donné un arbre maximier A associé à la liste $A = [\omega_0, \cdots, \omega_{n-1}]$ des poids associés aux nœuds, pour tout $k \in [\![0, n-1]\!]$, il y a un très fort lien entre la décomposition binaire de l'entier $k+1$ et la position du sommet s_k dans l'arbre A.
En effet, en notant :

$$k+1 = \sum_{i=0}^{r-1} \varepsilon_i \cdot 2^i \longleftrightarrow [\varepsilon_{r-1}, \varepsilon_{r-2}, \cdots, \varepsilon_0]$$

sa décomposition binaire avec $\varepsilon_{r-1} = 1$ et les ε_i valant 0 ou 1, le nombre r de chiffres binaires dans cette décomposition nous indique le numéro du niveau où se situe le sommet s_k. Plus précisément, le sommet s_k sera situé sur le niveau numéro $r-1$.
Ensuite, en considérant les chiffres dans cet ordre $\varepsilon_0, \varepsilon_1, \cdots, \varepsilon_{r-2}$, on peut associer au chiffre 0 la fonctionnalité « prendre le fils gauche » et associer au chiffre 1 la fonctionnalité « prendre le fils droit » .
Par exemple, dans l'arbre suivant où les valeurs affichées correspondent aux poids des nœuds, chaque sommet s_k étant ici associé au poids $k+1$:



considérons le sommet s_k, avec $k = 24$, associé ici au poids $A[k] = 25$.
L'entier $k+1$ valant 25, voici sa décomposition binaire :

$$25 = 16 + 8 + 1 \longleftrightarrow [1, 1, 0, 0, 1].$$

Le nombre 25 admet cinq chiffres binaires : le sommet s_k se situe au quatrième niveau dans l'arbre.
On considère alors la liste $\varepsilon_0 = 1, \varepsilon_1 = 0, \varepsilon_2 = 0, \varepsilon_3 = 1$. Ces chiffres nous indiquent le parcours à effectuer à partir de la racine pour atteindre le sommet s_k. Voici le parcours :

- on part de la racine s_0
- comme $\varepsilon_0 = 1$, on prend le sommet-fils droit
- comme $\varepsilon_1 = 0$, on prend le sommet-fils gauche
- comme $\varepsilon_2 = 0$, on prend encore le sommet-fils gauche
- finalement, comme $\varepsilon_3 = 1$, on prend le sommet-fils droit : on atteint la bonne position.

4.2 Complexité du tri maximier

On cherche une estimation du nombre d'opérations élémentaires utilisées dans l'algorithme du tri maximier, en fonction de la longueur n de la liste $L = [a_0, \cdots, a_{n-1}]$ de départ à trier.
En premier lieu, le nombre n admet un nombre de chiffres dans sa décomposition binaire égal à :

$$c_n = \lfloor \log_2(n) \rfloor + 1.$$

En effet, il existe un entier naturel k tel que $2^{k-1} \leqslant n < 2^k$, puis $k - 1 \leqslant \log_2(n) < k$, donc $k = \lfloor \log_2(n) \rfloor + 1$.
De plus, les entiers 2^{k-1} et $2^k - 1$ admettent exactement k chiffres dans leur décomposition binaire, l'entier 2^{k-1} étant le plus petit entier admettant k chiffres en binaire (un 1 suivi de $(k-1)$ zéros) alors que l'entier $2^k - 1$ est le plus grand entier admettant k chiffres en binaire (k chiffres 1).
On en déduit que l'entier n admet également k chiffres en binaire, d'où la formule.
Ce nombre c_n de chiffres en binaire nous donne la profondeur de l'arbre maximier construit à l'issue de la plantation de l'arbre A à partir de la liste L.
Dans le pire des cas, à chaque lecture d'une occurrence de la liste L, cela génère c_n comparaisons dans la percolation afin de placer correctement l'occurrence lue dans l'arbre en construction.
Le procédé de plantation de l'arbre génère donc un nombre d'opérations élémentaires égal à $\mathscr{O}(n \cdot c_n) = \mathscr{O}(n \cdot \ln n)$.
Ensuite, chaque cueillette génère au maximum $\mathscr{O}(c_n) = \mathscr{O}(\ln n)$ opérations élémentaires issues des transitions des occurrences une fois la racine cueillie pour remplir la liste M avec le fruit maximum.
La phase de cueillette et donc de la constitution de la liste triée M génère encore un nombre d'opérations en $\mathscr{O}(n \cdot \ln n)$.
Finalement, la complexité de l'algorithme de ce tri est en $\mathscr{O}(n \cdot \ln n)$ dans le pire des cas, ce qui en fait un algorithme aussi performant que le tri fusion.