Problème 2. Algorithmique

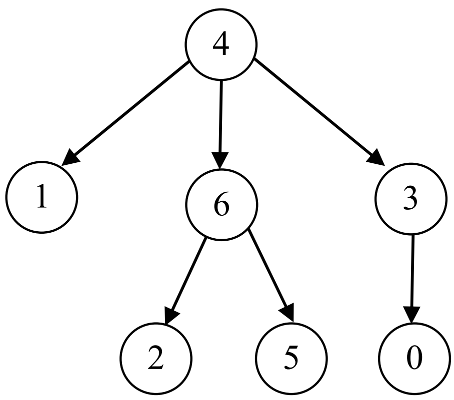
L'objectif de ce problème est de compter le nombre d'arbres *enracinés, non ordonnés* et *étiquetés* de nombre de nœuds donné. Pour cela, on étudie un codage particulier de ces arbres appelé *codage de Prüfer*.

Un arbre possède un nombre fini d'éléments appelés *nœuds*. Les arbres considérés dans ce problème possèdent tous au moins un nœud. Un *arbre enraciné non ordonné A* est défini récursivement de la façon suivante : il est constitué d'un nœud particulier appelé *racine* de A et d'un ensemble fini **non ordonné**, éventuellement vide, d'*arbres enracinés non ordonnés* appelés *sous-arbres* de A. Les racines des sous-arbres de A sont les *fils* de la racine de A et la racine de A est le *père* de ces derniers. Dans un arbre, deux nœuds sont dits *frères* s'ils ont même père. L'*arité* d'un nœud est son nombre de fils ; dans ce problème, l'arité d'un nœud peut être quelconque. Les nœuds d'arité 0 sont les *feuilles* de l'arbre.

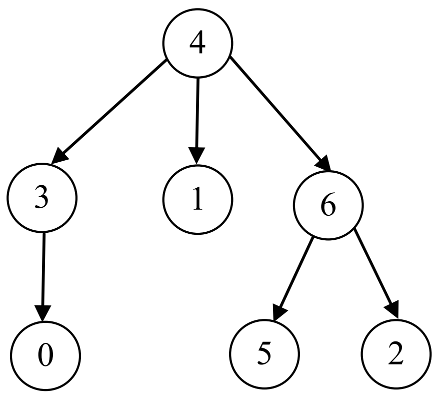
Un arbre est dit *étiqueté* si à chaque nœud est associé un entier positif ou nul, ces entiers étant deux à deux distincts ; l'entier associé à un nœud est l'*étiquette* du nœud. On pourra nommer un nœud par son étiquette ; si i est un entier, on pourra donc parler du nœud i pour le nœud d'étiquette i.

Dans ce problème, le terme d'*arbre* désignera toujours un arbre enraciné non ordonné étiqueté.

Les deux dessins ci-dessous sont deux représentations graphiques d'un **même** arbre nommé A_1. L'étiquette de la racine de A_1 est 4 ; l'ensemble des étiquettes des fils de la racine est $\{1, 3, 6\}$; l'ensemble des étiquettes des fils du nœud d'étiquette 6 est $\{2, 5\}$; le nœud d'étiquette 3 possède un seul fils : le nœud d'étiquette 0 ; les nœuds d'étiquettes 0, 1, 2, 5 n'ont pas de fils. Les représentations graphiques d'un arbre donné diffèrent par l'ordre dans lequel on dessine les fils d'un même nœud.

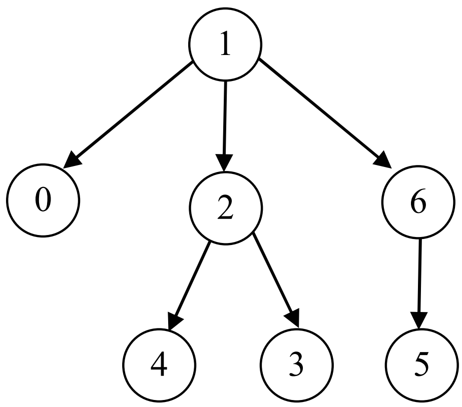


L'arbre A_1, première représentation



L'arbre A_1, seconde représentation

L'arbre A_2 représenté ci-contre est différent de l'arbre A_1.



L'arbre A_2

On dira qu'un arbre est un *arbre étiqueté consécutivement* s'il s'agit d'un arbre étiqueté et que l'ensemble de ses étiquettes forme un intervalle d'entiers de plus petite valeur 0 ; autrement dit, pour un arbre ayant n nœuds et étiqueté consécutivement, l'ensemble des étiquettes est $\{0, 1, 2, \ldots, n - 1\}$. Les arbres A_1 et A_2 sont des arbres étiquetés consécutivement.

Première partie : d'un codage racine-fils-frères d'un arbre au codage de Prüfer

❒ 17 – Donner la liste des arbres possédant trois nœuds et étiquetés consécutivement.

Soit A un arbre étiqueté consécutivement ayant n nœuds. Pour coder A, on définit un codage nommé *codage racine-fils-frères*. Pour cela, on fixe une représentation graphique de A ; on code A à l'aide de :

- l'étiquette de la racine (qui ne dépend pas de la représentation) ;
- un tableau nommé *fils* ; pour i compris entre 0 et $n-1$, la case d'indice i du tableau *fils* contient la valeur –1 si le nœud i est une feuille de l'arbre et, sinon, l'étiquette du fils du nœud i se situant le plus à gauche dans la représentation graphique choisie ;
- un tableau nommé *freres* ; pour i compris entre 0 et $n-1$, la case d'indice i du tableau *freres* contient la valeur –1 si le nœud i n'a aucun frère sur sa droite et, sinon, l'étiquette de son frère qui se trouve le premier sur sa droite.

Pour l'arbre A_1, si on choisit la première représentation, on obtient le codage suivant :

- la racine est le nœud 4 ;
- pour le tableau *fils* : les cases d'indices 0, 1, 2 et 5 contiennent la valeur –1, la case d'indice 3 contient 0, la case d'indice 4 contient 1, la case d'indice 6 contient 2 ;
- pour le tableau *freres* : les cases d'indices 0, 3, 4 et 5 contiennent la valeur –1, la case d'indice 1 contient 6, la case d'indice 2 contient 5, la case d'indice 6 contient 3.

Ainsi, l'arbre A_1 est représenté par la valeur 4 pour la racine et par les deux tableaux ci-dessous :

indice	0	1	2	3	4	5	6
fils	–1	–1	–1	0	1	–1	2

indice	0	1	2	3	4	5	6
freres	–1	6	5	–1	–1	–1	3

On définit aussi deux tableaux qui peuvent être calculés à partir du codage racine-fils-frères :

- un tableau nommé *peres* ; pour i compris entre 0 et $n-1$, la case d'indice i contient la valeur –1 s'il s'agit de la racine de l'arbre et, dans les autres cas, l'étiquette du père du nœud i ; pour l'arbre A_1, la case d'indice 4 contient la valeur –1, la case d'indice 0 contient 3, les cases d'indices 1, 3 et 6 contiennent 4, les cases d'indices 2 et 5 contiennent la valeur 6 ;
- un tableau nommé *arites* ; pour i compris entre 0 et $n-1$, la case d'indice i de ce tableau contient l'arité du nœud i ; pour l'arbre A_1, les cases d'indices 0, 1, 2 et 5 contiennent la valeur 0, la case d'indice 3 contient 1, la case d'indice 4 contient 3, la case d'indice 6 contient 2.

Pour l'arbre A_1, les tableaux *peres* et *arites* sont représentés ci-dessous :

indice	0	1	2	3	4	5	6
peres	3	4	6	4	–1	6	4

indice	0	1	2	3	4	5	6
arites	0	0	0	1	3	0	2

Indications pour la programmation en Pascal

On définit la constante et le type suivant :

```
const MAX = 100;
type Tableau = array[0 .. MAX - 1] of Integer;
```

La constante `MAX` est un majorant du nombre de nœuds des arbres considérés.

Fin des indications pour la programmation en Pascal

❒ 18 – Il s'agit d'écrire en langage de programmation une fonction nommée *calculer_peres* qui, à partir du codage racine-fils-frères d'un arbre étiqueté consécutivement, calcule le tableau *peres* correspondant à cet arbre.

Caml : Écrire en Caml une fonction `calculer_peres` telle que, si on considère un arbre A possédant n nœuds et étiqueté consécutivement et si :

- `racine` est un entier qui contient l'étiquette de la racine de A,
- `fils` et `freres` sont deux vecteurs de longueur n qui représentent respectivement les tableaux *fils* et *freres* d'un codage racine-fils-frères de A,

alors `calculer_peres racine fils freres` renvoie un vecteur de longueur n correspondant au tableau *peres* défini plus haut.

Pascal : Écrire en Pascal une fonction `calculer_peres` telle que, si on considère un arbre A étiqueté consécutivement et si :

- `racine` est un entier qui contient l'étiquette de la racine de A,
- `fils` et `freres` sont de type `Tableau` et représentent respectivement les tableaux *fils* et *freres* d'un codage racine-fils-frères de A,
- `n` est un entier qui contient le nombre de nœuds de A,

alors `calculer_peres(racine, fils, freres, n)` renvoie un tableau de type `Tableau` contenant, entre les indices 0 et $n - 1$, le tableau *peres* défini plus haut.

❒ 19 – Indiquer, en fonction du nombre de nœuds de l'arbre considéré, la complexité de la fonction *calculer_peres*.

❒ 20 – Il s'agit d'écrire en langage de programmation une fonction nommée *calculer_arites* qui, à partir du codage racine-fils-frères d'un arbre étiqueté consécutivement, renvoie le tableau *arites* correspondant à cet arbre.
Caml : Écrire en Caml une fonction `calculer_arites` telle que, pour un arbre A possédant n nœuds et étiqueté consécutivement, si `fils` et `freres` sont deux vecteurs de longueur n qui représentent respectivement les tableaux *fils* et *freres* d'un codage racine-fils-frères de A, alors `calculer_arites fils freres` renvoie un vecteur correspondant au tableau *arites* défini plus haut.

Pascal : Écrire en Pascal une fonction `calculer_arites` telle que, pour un arbre A étiqueté consécutivement, si :

- `fils` et `freres` sont de type `Tableau` et représentent respectivement les tableaux *fils* et *freres* d'un codage racine-fils-frères de A,
- `n` est un entier qui contient le nombre de nœuds de A,

alors `calculer_arites(fils, freres, n)` renvoie un tableau de type `Tableau` contenant entre les indices 0 et $n - 1$ les arités des nœuds de l'arbre.

❒ 21 – Indiquer, en fonction du nombre de nœuds de l'arbre considéré, la complexité de la fonction *calculer_arites*.

❒ 22 – Il s'agit d'écrire en langage de programmation une fonction *inserer* qui prend en arguments un tableau *table* d'entiers non nécessairement distincts triés par valeurs décroissantes et un entier d ; cette fonction modifie le tableau *table* pour insérer l'entier d en respectant l'ordre décroissant. L'entier d est inséré même s'il figure déjà dans *table*.
Caml : Écrire en Caml une fonction `inserer` telle que, si :

- `table` est un vecteur d'entiers,
- `nb` est un entier positif ou nul ne dépassant pas la dimension du vecteur `table` diminuée de 1,
- `d` est un entier,
- on suppose que le vecteur `table` contient des entiers classés par valeurs décroissantes dans les cases d'indices compris entre 0 et `nb — 1`, les autres cases du vecteur `table` étant ignorées,

alors `inserer table nb d` insère la donnée `d` dans le vecteur `table` en respectant l'ordre décroissant. La fonction renvoie `nb + 1`, c'est-à-dire le nouveau nombre de données figurant dans `table.`

Pascal : Écrire en Pascal une fonction `inserer` telle que, si :

- `table` est de type `Tableau`,
- `nb` est un entier positif ou nul ne dépassant pas `MAX — 1`,
- `d` est un entier ;
- on suppose que le tableau `table` contient entre les indices 0 et `nb — 1` des entiers classés par valeurs décroissantes, les autres cases du tableau `table` étant ignorées,

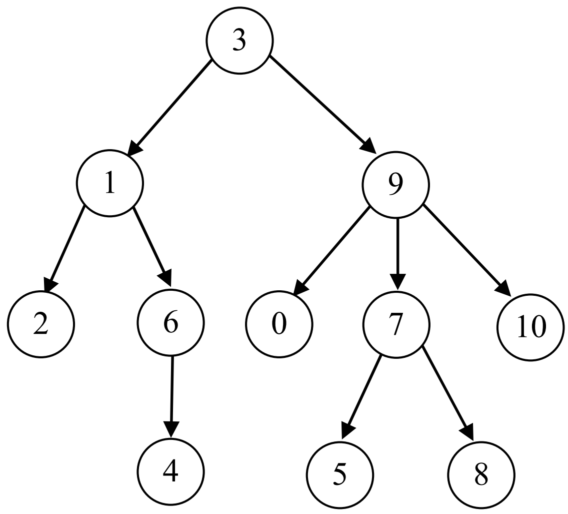
alors `inserer(table, nb, d)` insère la donnée `d` dans le tableau `table` en respectant l'ordre décroissant. La fonction renvoie `nb + 1`, c'est-à-dire le nouveau nombre de données figurant dans `table`.

❒ 23 – Indiquer, en fonction du nombre *nb* d'entiers contenus dans un tableau trié *table*, la complexité de la fonction *inserer* quand elle insère un nouvel entier dans *table*.

Soit A un arbre possédant n nœuds ; on note $E(A)$ l'ensemble des étiquettes de A ; les étiquettes de A étant toutes distinctes, l'ensemble $E(A)$ possède n éléments. Le codage de Prüfer d'un arbre étiqueté ayant n nœuds est une suite de $n - 1$ entiers appartenant à $E(A)$, suite notée $Pr(A)$; ce codage est défini récursivement de la façon suivante. Si A est réduit à un nœud, sa racine, son codage de Prüfer est la suite vide. Sinon, soit f la feuille de A d'étiquette minimum et soit p le père de f ; on note A' l'arbre obtenu en enlevant de A la feuille f ; par définition, le codage de Prüfer de A est la suite dont le premier élément est l'étiquette de p, ce premier élément étant suivi du codage de Prüfer de A'.

Ainsi, le codage de Prüfer de l'arbre A_1 est : 3, 4, 6, 4, 6, 4 ; le codage de Prüfer de l'arbre A_2 est : 1, 2, 2, 1, 6, 1.

❒ 24 – Indiquer le codage de Prüfer de l'arbre A_3 ci-contre.



L'arbre A_3

❒ 25 – On considère un arbre A étiqueté consécutivement. Il s'agit d'écrire en langage de programmation une fonction qui calcule le codage de Prüfer de A. La fonction commencera par calculer les tableaux *peres* et *arites* ; puis elle construira un tableau contenant les feuilles de l'arbre initial classées par étiquettes décroissantes ; après cette partie préparatoire, la fonction calculera le codage de Prüfer.

Caml : Écrire en Caml une fonction `calculer_Prufer` telle que, si on considère un arbre A étiqueté consécutivement et si :

- `racine` est un entier qui contient l'étiquette de la racine de A,
- `fils` et `freres` sont deux vecteurs de longueur n qui représentent respectivement les tableaux *fils* et *freres* d'un codage racine-fils-frères de A,

alors `calculer_Prufer racine fils freres` renvoie un vecteur de longueur $n - 1$ contenant le codage de Prüfer de l'arbre A.

Pascal : Écrire en Pascal une fonction `calculer_Prufer` telle que, si on considère un arbre A étiqueté consécutivement et si :

- `racine` est un entier qui contient l'étiquette de la racine de A,
- `fils` et `freres` sont de type `Tableau` et représentent respectivement les tableaux *fils* et *freres* d'un codage racine-fils-frères de A,
- `n` est un entier qui contient le nombre de nœuds de A,

alors `calculer_Prufer(racine, fils, freres, n)` renvoie un tableau, de type `Tableau`, contenant le codage de Prüfer de l'arbre A entre les indices 0 et $n - 2$.

❒ 26 – Indiquer la complexité du calcul du codage de Prüfer d'un arbre A possédant n nœuds, étiqueté consécutivement et codé avec le codage racine-fils-frères.

Seconde partie : d'un codage de Prüfer d'un arbre à un codage racine-fils-frères

❒ 27 – On suppose qu'on connaît le codage de Prüfer d'un arbre A étiqueté consécutivement. Il s'agit d'écrire une fonction *calculer_arites_par_Prufer* qui calcule les arités des nœuds de l'arbre A à partir de ce codage.
Caml : Écrire en Caml une fonction `calculer_arites_par_Prufer` telle que, pour un arbre A possédant n nœuds et étiqueté consécutivement, si `Prufer` est un vecteur de longueur $n - 1$ contenant le codage de Prüfer de A, alors `calculer_arites_par_Prufer Prufer` renvoie un vecteur de longueur n contenant les arités des nœuds de A.
Avant d'écrire la fonction `calculer_arites_par_Prufer`, on en donnera rapidement le principe.

Pascal : Écrire en Pascal une fonction `calculer_arites_par_Prufer` telle que, pour un arbre A étiqueté consécutivement, si :
- `Prufer` est de type `Tableau` et contient le codage de Prüfer de A,
- `n` est un entier qui contient le nombre de nœuds de A,

alors `calculer_arites_par_Prufer(Prufer, n)` renvoie un tableau, de type `Tableau`, contenant les arités des nœuds de A.
Avant d'écrire la fonction `calculer_arites_par_Prufer`, on en donnera rapidement le principe.

❒ 28 – Déterminer un arbre A étiqueté consécutivement dont le codage de Prüfer $Pr(A)$ est : 2, 3, 0, 2, 2. On détaillera la démarche utilisée.

❒ 29 – On considère un arbre A ; on suppose que l'ensemble des étiquettes $E(A)$ de A est $\{1, 3, 5, 6, 7, 9, 10, 12, 13\}$; l'arbre A n'est donc pas étiqueté consécutivement ; on suppose enfin que le codage de Prüfer $Pr(A)$ de A est : 3, 10, 3, 7, 7, 5, 7, 5. Déterminer l'arbre A. On décrira succinctement la démarche utilisée.

❒ 30 – Il s'agit d'écrire en langage de programmation une fonction *calculer_arbre* qui, à partir du codage de Prüfer d'un arbre A **étiqueté consécutivement**, calcule un codage racine-fils-frères de A.
Caml : Écrire en Caml une fonction `calculer_arbre` telle que, pour un arbre A possédant n nœuds et étiqueté consécutivement, si :
- `Prufer` est un vecteur de longueur $n - 1$ contenant le codage de Prüfer de A,
- `fils` et `freres` sont deux vecteurs de longueur n,

alors `calculer_arbre Prufer fils freres` modifie les vecteurs `fils` et `freres` pour qu'ils correspondent respectivement aux tableaux *fils* et *freres* d'un codage racine-fils-frères de A et renvoie l'étiquette de la racine de A.

Pascal : Écrire en Pascal une fonction `calculer_arbre` telle que, pour un arbre A possédant n nœuds et étiqueté consécutivement, si :
- `Prufer` est de type `Tableau` et contient le codage de Prüfer de A,
- `fils` et `freres` sont de type `Tableau`,
- `n` est un entier qui contient le nombre de nœuds de A,

alors `calculer_arbre(Prufer, fils, freres, n)` modifie les tableaux `fils` et `freres` pour qu'ils correspondent respectivement aux tableaux *fils* et *freres* d'un codage racine-fils-frères de A et renvoie l'étiquette de la racine de A.

Soit $\mathcal{E}$ un ensemble de n entiers distincts positifs ou nuls ; soit $\mathcal{S}(\mathcal{E})$ l'ensemble des suites de longueur $n - 1$ dont tous les éléments sont dans $\mathcal{E}$, distincts ou non ; soit enfin $\mathcal{A}(\mathcal{E})$ l'ensemble des arbres enracinés non ordonnés, possédant n nœuds et étiquetés par les éléments de $\mathcal{E}$.

❒ 31 – Montrer que l'application Pr qui, à un arbre appartenant à $\mathcal{A}(\mathcal{E})$, associe son codage de Prüfer est une bijection entre $\mathcal{A}(\mathcal{E})$ et $\mathcal{S}(\mathcal{E})$.

❒ 32 – Déterminer le cardinal de $\mathcal{A}(\mathcal{E})$.