NE PAS ÉCRIRE SUR LE SUJET

Exercice 1

Dans cet exercice, on autorise les doublons dans un arbre binaire de recherche (ABR) et pour le cas d'égalité, on choisira le sous-arbre gauche. On ne cherche pas à équilibrer les arbres.

1. Rappeler la définition d'un ABR.
2. Insérer successivement et une à une dans un ABR initialement vide toutes les lettres du mot *bacddabdbae*, selon l'ordre alphabétique. Quelle est la hauteur de l'arbre ainsi obtenu ?
3. Montrer par induction que le parcours en profondeur infixe d'un ABR est croissant.
4. Proposer un algorithme qui permet de compter le nombre d'occurrences d'un élément x dans un ABR. Quelle est sa complexité ?
5. On souhaite supprimer *une* occurrence d'un élément x dans un ABR. Expliquer le principe d'un algorithme résolvant ce problème et le mettre en œuvre sur l'arbre obtenu à la question 2, en supprimant successivement une occurrence de e, b, b, c et d. Quelle est la complexité ?

Exercice 2

Consignes :

- Le sujet fournit un fichier source `grundy.c`. Il contient le type défini dans cet énoncé, des jeux exemples, et les prototypes des fonctions à écrire.
- On pourra compiler le fichier par la commande `gcc -o nom_exec grundy.c`, en se plaçant dans le bon répertoire.
- On pourra exécuter le fichier exécutable obtenu par `./nom_exec`.

Définition

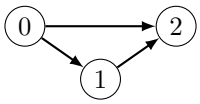
Un **jeu** est un triplet (S, A, s_0) où S est un ensemble fini d'**états**, $A \subseteq S^2$ est un ensemble de **transitions** et $s_0 \in S$ est un **état initial**, tels que le graphe (S, A) est acyclique.

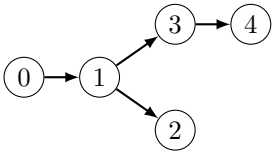
Une **stratégie** est une fonction partielle $\varphi : S \rightharpoonup S$ telle que pour tout état s où elle est définie, $(s, \varphi(s)) \in A$. On peut faire jouer deux stratégies φ_0 et φ_1 l'une contre l'autre en les faisant jouer alternativement. On définit ainsi la séquence d'états $s_1 = \varphi_0(s_0)$, $s_2 = \varphi_1(s_1)$, $s_3 = \varphi_0(s_2)$, …, $s_k = \varphi_{(k-1) \bmod 2}(s_{k-1})$. On admet que la séquence (s_k) est finie.

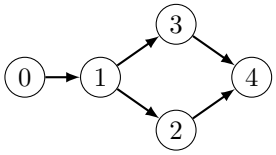
Pour deux stratégies φ_0 et φ_1 jouées par les joueurs 0 et 1, le joueur perdant est le premier joueur pour lequel la stratégie n'est plus définie.

Une **stratégie gagnante** pour le joueur $i \in \{0, 1\}$ est une stratégie qui garantit que le joueur i gagne quand il joue suivant cette stratégie, quelle que soit la stratégie jouée par l'autre joueur.

1. Parmi les jeux suivants (où l'état initial est le sommet 0), quels sont ceux qui ont une stratégie gagnante pour le joueur 0 ? Pour le joueur 1 ?







On représente un jeu en C par le type suivant :

```c
struct Jeu {
    int taille;
    int graphe[100][100];
};

typedef struct Jeu jeu;
```

Ainsi, si $J = (S, A, s_0)$ est un jeu avec $|S| = n$ représenté par un élément `J` de type `jeu`, alors :

- $S = [\![0, n-1]\!]$, $n \leqslant 100$ et `J.taille` $= n$;
- $s_0 = 0$;
- `J.graphe` est un tableau de 100 tableaux d'entiers de taille 100 tel que si $s \in S$, alors `J.graphe[s]` contient des entiers correspondant aux voisins de s, suivis d'une valeur sentinelle 100, suivie de valeurs quelconques. Si $i \geqslant n$, alors `J.graphe[i]` est un tableau quelconque.

Les trois graphes de la question précédente sont définis comme des variables globales.

2. Faut-il libérer la mémoire allouée pour créer les jeux exemples, une fois l'exécution terminée ? Justifier.
3. Écrire une fonction `int nombre_coups_possibles(jeu J, int s)` qui prend en argument un jeu et un état s et renvoie le nombre de coups possibles depuis l'état s.
4. Écrire une fonction `bool est_voisin(jeu J, int s, int t)` telle que `est_voisin(J, s, t)` renvoie un booléen qui vaut `true` si et seulement si t est un état accessible en un coup depuis s dans le jeu J.

Pour un jeu (S, A, s_0), on définit, $s \in S$, sa **valeur de Grundy** $G(s) \in \mathbb{N}$ par :

$$G(s) = \min(\mathbb{N} \setminus \{G(t) \mid (s,t) \in A\})$$

5. Écrire une fonction `int plus_petit_absent(int* tab, int k)` qui prend en argument un pointeur vers un tableau d'entiers positifs ou nuls et un entier k et renvoie le plus petit entier naturel qui n'apparaît pas parmi les k premiers éléments du tableau.
   On attend une complexité linéaire en k, mais on se contentera d'une fonction de complexité quadratique.

On veut écrire une fonction de programmation dynamique `void grundy(jeu J, int G[100])` qui prend en argument un jeu (S, A, s_0) et un tableau `G` et modifie `G` de telle sorte qu'après l'appel, pour tout $s \in S$, `G[s]` contient la valeur $G(s)$.

6. Compléter la fonction `nb_grundy` du fichier `grundy.c` en conséquence. Quelle est la complexité de la fonction `grundy` ?
7. Montrer que le joueur 0 a une stratégie gagnante dans le jeu $J = (S, A, s_0)$ si et seulement si $G(s_0) \neq 0$.

NE PAS ÉCRIRE SUR LE SUJET