Exercice 1

On considère l'alphabet $\Sigma = \{a, b, c\}$ et L l'ensemble des mots constitués d'un nombre strictement positif de a, suivis d'un c, suivi d'un mélange quelconque de a et b.

1. Proposer une expression régulière r décrivant le langage L.
2. Construire l'automate de Glushkov associé à l'expression régulière r.
3. L'automate précédent est-il déterministe ? La réponse à cette question est-elle toujours la même si on considère une autre expression régulière (pas nécessairement associée au langage L) ? Justifier.
4. Rappeler l'énoncé du lemme de pompage (appelé aussi lemme de l'étoile).
5. Montrer que le langage $L' = \{uu \mid u \in \{a, b\}^*\}$ n'est pas rationnel.
6. Le langage $L'' = \{uuv \mid u, v \in \{a, b\}^*\}$ est-il rationnel ? Justifier. Même question pour le langage $L''' = \{uvuw \mid u \in \{a, b\}^*, v, w \in \{c\}^*\}$

Exercice 2: L'exercice suivant est à traiter dans le langage C.

Dans cet exercice, on considère des arbres binaires non étiquetés. On dit qu'un arbre est **binaire strict** si tout nœud interne a exactement deux enfants, ou autrement dit si tout nœud (interne ou non) a zéro ou deux enfants. On dit qu'un arbre est **parfait** s'il est binaire strict et que toutes ses feuilles sont à même profondeur. On dit qu'il est **presque-parfait** si tous les niveaux de profondeur sont remplis par des nœuds, sauf éventuellement le dernier, rempli par la gauche.

1. Combien existe-t-il d'arbres presque-parfaits de taille 6 ? Les représenter graphiquement.
2. Citer une structure de données qui peut s'implémenter avec des arbres presque-parfaits.
3. Proposer une définition par induction d'un arbre parfait de hauteur h, pour $h \in \mathbb{N}$. Justifier rigoureusement que cette définition coïncide avec celle de l'énoncé.

On implémente un arbre binaire en C par le type `arbre` défini de la manière suivante :

```c
struct Noeud {
    struct Noeud* gauche;
    struct Noeud* droite;
};

typedef struct Noeud arbre;
```

Ainsi, un arbre contient un pointeur vers son fils gauche et son fils droit.

4. Écrire une fonction `int hauteur(arbre* a)` qui prend en argument un pointeur vers un arbre et calcule et renvoie sa hauteur.
5. En déduire une fonction `bool est_parfait(arbre* a)` qui prend en argument un pointeur vers un arbre et renvoie un booléen qui vaut `true` si et seulement si l'arbre pointé par `a` est parfait. Quelle est sa complexité temporelle pour un arbre `a` qui est effectivement parfait ?
6. Écrire une fonction `arbre* plus_grand_presque_parfait(arbre* a)` qui prend en argument un pointeur vers un arbre a et renvoie un pointeur vers le plus grand sous-arbre de a qui est presque-parfait. La complexité de cette fonction doit être linéaire en la taille de l'arbre a.
   Indication : que peut-on dire des enfants gauche et droit d'un arbre presque-parfait de hauteur h ?

Exercice 3

Définition

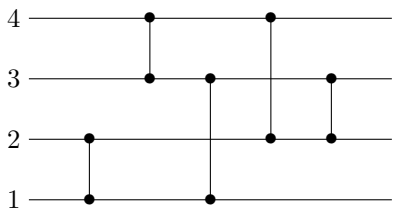
Soit T un tableau de taille n. On appelle **module de comparaison et d'échange** (ou simplement module) un couple (i, j) où $i, j \in [\![1, n]\!]$, $i < j$. On appelle **réseau de tri** une suite finie de modules.
On définit l'**action** d'un module (i, j) sur T par l'échange des cellules $T[i]$ et $T[j]$ si et seulement si $T[j] < T[i]$ (et aucune modification sinon). On définit l'**action** d'un réseau de tri sur T comme l'action de chacun de ses modules dans l'ordre où ils apparaissent.

On représente graphiquement un réseau de tri pour un tableau T de la manière suivante :

- chaque indice de T est représenté par une ligne horizontale ;
- les modules sont représentés de gauche à droite, chacun d'entre eux étant représenté par une ligne verticale reliant les lignes des deux indices correspondant.

Exemple

Le réseau de tri $(1, 2), (3, 4), (1, 3), (2, 4), (2, 3)$ est représenté par :



1. Quelle est l'action du réseau de tri donné en exemple sur le tableau $[2, 4, 5, 1]$? Ce réseau permet-il de trier tout tableau de taille 4 ?
2. Décrire comment construire un réseau de tri naïf qui trie tout tableau de taille n. Combien de modules contient-il asymptotiquement ?
3. Décrire un réseau qui fusionne deux tableaux triés de taille 2. Faire de même pour un réseau qui fusionne deux tableaux triés de taille $4 = 2^2$.
4. On considère deux tableaux triés de taille paire $[a_1, \ldots, a_{2n}]$ et $[b_1, \ldots, b_{2n}]$. On note $[c_1, c_2, \ldots, c_{2n}]$ le résultat de la fusion de $[a_1, a_3, \ldots, a_{2n-1}]$ et $[b_1, b_3, \ldots, b_{2n-1}]$; et on note $[d_1, d_2, \ldots, d_{2n}]$ le résultat de la fusion de $[a_2, a_4, \ldots, a_{2n}]$ et $[b_2, b_4, \ldots, b_{2n}]$. Montrer que le tableau suivant est trié :

$$[c_1, \min(c_2, d_1), \max(c_2, d_1), \min(c_3, d_2), \max(c_3, d_2), \ldots, \min(c_{2n}, d_{2n-1}), \max(c_{2n}, d_{2n-1}), d_{2n}]$$

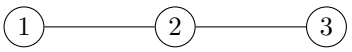
5. En déduire un réseau qui fusionne deux tableaux triés de même taille $n = 2^k$. Combien de modules ce réseau comprend-il ?
6. On suppose disposer de $n = 2^k$ fils d'exécutions, indicés de 1 à n. On souhaite effectuer le tri d'un tableau t de taille $2n$ en le minimum de temps, en supposant que $[t_1, \ldots, t_n]$ et $[t_{n+1}, \ldots, t_{2n}]$ sont des portions triées, en s'inspirant du réseau de la question précédente. Quels problèmes de synchronisation peuvent se produire ? Proposer une solution.
7. Déterminer un réseau de tri efficace qui trie un tableau quelconque de taille $n = 2^k$. Combien de modules ce réseau comprend-il ?

Exercice 4

Définition

Soit $G = (V, E)$ un graphe non-orienté. On appelle **séquence** une suite finie non-vide σ dont la longueur sera notée $|\sigma| > 0$. On appelle **chemin** dans G une séquence $c = v_1 \dots v_{|c|}$ d'éléments de V telle que pour $1 \leqslant i < |c|$, on a $\{v_i, v_{i+1}\} \in E$. On dit qu'un chemin dans G est **simple** s'il n'existe pas $1 \leqslant i < j \leqslant n$ tels que $\{v_i, v_{i+1}\} = \{v_j, v_{j+1}\}$. Un **cycle** dans G est un chemin simple $c = v_1 \dots v_n$ de longueur $n > 1$ tel que $v_1 = v_n$.
On dit que deux séquences $u_1 \dots u_n$ et $v_1 \dots v_n$ de même longueur se **rencontrent** s'il existe $i \in [\![1; n]\!]$ tel que $u_i = v_i$. Une **séquence d'élimination** pour G est une séquence $\sigma = v_1 \dots v_{|\sigma|}$ d'éléments de V telle que, pour tout chemin c dans G tel que $|c| = |\sigma|$, les séquences σ et c se rencontrent.

1. Construire une séquence d'élimination pour le graphe suivant :



2. Montrer que si G a un cycle, alors G n'admet pas de séquence d'élimination.
3. Pour $k \in \mathbb{N}^*$, on note L_k le graphe (V, E) tel que $V = [\![1; k]\!]$ et $E = \{\{i, i+1\} | 1 \leqslant i < k\}$. Construire une séquence d'élimination pour L_k.

Définition

Une **séquence d'élimination partielle** pour G et pour un sous-ensemble $X \subseteq V$ est une séquence
$\sigma = v_1 \dots v_{|\sigma|}$ d'éléments de V telle que, pour tout chemin c dans G tel que $|c| = |\sigma|$, si le dernier élément de c est dans X, alors σ et c se rencontrent.
On note $\phi_E : \mathcal{P}(V) \longrightarrow \mathcal{P}(V)$ la fonction définie par :

$$\phi_E(X) := V \setminus \{v | \exists v' \in V \setminus X, \{v, v'\} \in E\}$$

4. Montrer que pour $X \subseteq V$, σ une séquence d'élimination partielle pour G et X et $v \in V$, alors la séquence σ' obtenue en concaténant σ et v est une séquence d'élimination partielle pour G et $\phi_E(X) \cup \{v\}$.
5. En déduire un algorithme pour décider si un graphe admet une séquence d'élimination, et si oui, la calculer. Discuter de l'efficacité de l'algorithme et de la longueur de la séquence obtenue.
Indication : On construira un graphe orienté sur $\mathcal{P}(V)$.
6. Pour tout $k \in \mathbb{N}^*$, on note S_k le graphe $(\{0\} \cup \{1, \dots, k\} \cup \{1', \dots, k'\}, \{\{0, i\} | 1 \leqslant i \leqslant k\} \cup \{\{i, i'\} | 1 \leqslant i \leqslant k\})$. Construire une séquence d'élimination pour S_k.
7. Montrer que le graphe suivant n'admet pas de séquence d'élimination :