I Lemme de l'étoile pour les langages hors-contexte

On admet la version suivante du lemme de l'étoile pour les langages hors-contextes :

Théorème : Lemme de l'étoile hors-contexte

Si L est un langage hors-contexte alors il existe un entier n tel que, pour tout mot $t \in L$ tel que $|t| \geq n$, on peut écrire $t = uvwxy$ avec :

- $|vwx| \leq n$;
- $vx \neq \varepsilon$;
- $\forall i \in \mathbb{N}, uv^iwx^iy \in L$.

Soient $L_1 = \{a^n b^n c^p \mid n, p \in \mathbb{N}\}$ et $L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$.

1. Montrer que L_1 est un langage hors-contexte.
2. Montrer que L_2 n'est pas un langage hors-contexte.
3. Montrer que l'ensemble des langages hors-contextes n'est pas stable par intersection ni par passage au complémentaire.

II Algorithmes de Borůvka

Soit $G = (S, A)$ un graphe non-orienté connexe et pondéré par $w : A \longrightarrow \mathbb{R}$. On note $n = |S|$ et $p = |A|$. On suppose que tous les poids de G sont distincts (c'est-à-dire : w injective) et que $S = \{0, ..., n-1\}$.

II.1 Théorie

1. Montrer que G possède un arbre couvrant de poids minimum. Avec quelle complexité pourrait-on en trouver un ?
2. Montrer que G possède un unique arbre couvrant de poids minimum.

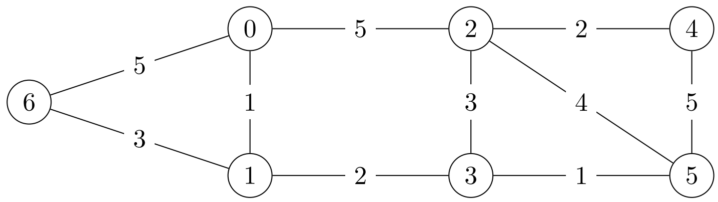
On appelle T^* l'unique arbre couvrant de poids minimum de G.
Soit $X \subset S$. On dit qu'une arête est sûre pour X si elle est de poids minimum parmi les arêtes ayant exactement une extrémité dans X. Autrement dit, une arête e est sûre pour X si $w(e) = \min\{w(e') \mid \{u, v\} \in A, u \in X, v \notin X\}$.
L'objectif de l'algorithme de Borůvka est de construire un arbre couvrant de poids minimum T en conservant une partition F de S, correspondant aux composantes connexes de T.
À chaque étape, on ajoute une arête sûre pour chaque composante connexe de F :

Algorithme de Borůvka

```
F ⟵ {{x} | x ∈ S}
T ⟵ ∅
Tant que |F| > 1 :
    E ⟵ ∅
    Pour C ∈ F :
        e ⟵ arête sûre pour C
        E ⟵ E ∪ {e}
    F ⟵ partition de S obtenue en fusionnant les composantes
      connexes de F avec les arêtes de E
    T ⟵ T ∪ E
Renvoyer T
```

L'étape de fusion des composantes connexes consiste, pour chaque arête $e = \{u, v\}$ de E, à remplacer dans F les composantes connexes C_1 et C_2 contenant u et v par leur union $C_1 \cup C_2$.

3. Appliquer l'algorithme de Borůvka sur le graphe suivant, en donnant à chaque l'ensemble des arêtes de T à la fin de chaque passage dans la boucle **Tant que** :



4. Montrer que l'algorithme de Borůvka termine, en utilisant un variant de boucle.
5. Soit $X \subset S$ et e une arête sûre pour X. Montrer que T^* contient e.
6. Montrer que l'algorithme de Borůvka renvoie bien T^*.

II.2 Implémentation

On va utiliser une structure d'Union-Find pour représenter les composantes connexes de F, sous la forme d'un tableau `uf` de taille n tel que `uf.(x)` soit le père de `x` dans l'arbre contenant `x`. Si `x` est une racine, `uf.(x)` contiendra `x`.
On n'utilisera pas d'optimisation de type union par rang ou compression de chemin.

7. Écrire une fonction `create : int -> int array` telle que `create n` renvoie un tableau de taille `n` initialisé avec les entiers de 0 à `n - 1`.
8. Écrire une fonction `find : int array -> int -> int` telle que `find uf x` renvoie la racine de l'arbre contenant `x` dans la structure d'Union-Find représentée par le tableau `uf`.
9. Écrire une fonction `meme_cc : int array -> int -> int -> bool` telle que `meme_cc uf x y` détermine si `x` et `y` sont dans la même composante connexe.
10. Écrire une fonction `n_cc : int array -> int` telle que `n_cc uf` renvoie le nombre de composantes connexes dans `uf`.

Le graphe G est représenté par une liste d'adjacence `g` telle que `g.(i)` contient une liste des arêtes partant de `i`, où chaque arête est un couple (w, j) où w est le poids de l'arête et i et j les extrémités de l'arête.

11. Écrire une fonction `aretes_sures : (float * int) list array -> int array -> (float * int * int) array` telle que
`aretes_sures g uf` renvoie un tableau `ans` de taille n où, si `i` est une racine dans `uf`, `ans.(i)` contient l'arête sûre pour la composante connexe de `i`.
Si `i` n'est pas une racine, `ans.(i)` contiendra `(max_float, -1, -1)`.
12. Écrire une fonction `boruvka : (float * int) list array -> (float * int) list array` renvoyant l'arbre couvrant de poids minimum de G par l'algorithme de Borůvka.
13. Quitte à utiliser l'optimisation par compression de chemin et union par rang, on suppose que `union` et `find` sont en O(1). Montrer que la complexité de l'algorithme de Borůvka est en $O(p \log n)$. Comparer avec l'algorithme de Kruskal.

III Théorème de Chomsky-Schützenberger

III.1 Langage de Dyck

Soit $n \in \mathbb{N}^*$. On définit $\Sigma_n = \{a_1, \overline{a_1}, a_2, \overline{a_2}, ..., a_n, \overline{a_n}\}$. Les lettres a_i seront appelées parenthèses ouvrantes et les $\overline{a_i}$ sont les parenthèses fermantes.
Soit $G_n = (\Sigma_n, \{S\}, R, S)$, où R contient les règles :

$$S \longrightarrow a_1 S \overline{a_1} S \mid a_2 S \overline{a_2} S \mid ... \mid a_n S \overline{a_n} S \mid \varepsilon$$

On définit le langage D_n de Dyck d'ordre n comme celui engendré par G_n.

On note Pref(u) l'ensemble des préfixes d'un mot u.

1. Représenter graphiquement un arbre de dérivation de $u = a_1 a_2 \overline{a_2} a_3 \overline{a_3} \overline{a_1}$ pour G_3.
2. Soit $L = \{u \in \Sigma_1^* \mid \forall v \in \text{Pref}(u), |v|_{a_1} \geq |v|_{\overline{a_1}} \text{ et } |u|_{a_1} = |u|_{\overline{a_1}}\}$. Montrer que $D_1 = L$.
3. En déduire que D_1 n'est pas régulier.
4. Est-il vrai que pour tout $n \geq 1$, $D_n = \{u \in \Sigma_n \mid \forall v \in \text{Pref}(u), \forall i \in [\![1, n]\!], |v|_{a_i} \geq |v|_{\overline{a_i}} \text{ et } |u|_{a_i} = |u|_{\overline{a_i}}\}$? Justifier.

On définit le type suivant :

```
type lettre = O of int | F of int
```

tel qu'une lettre a_i sera représentée par `O i` et une lettre $\overline{a_i}$ par `F i`.

5. Écrire une fonction `dyck : lettre list -> bool` qui détermine si un mot u est un mot d'un langage de Dyck D_n pour un certain $n \in \mathbb{N}$, en complexité linéaire en la taille de u.

Soient Σ et Γ deux alphabets. On appelle morphisme de mots une fonction $\varphi : \Sigma^* \longrightarrow \Gamma^*$ telle que pour tout $u, v \in \Sigma^*$, $\varphi(uv) = \varphi(u)\varphi(v)$.
Remarque : il suffit de définir un morphisme de mots sur les lettres, car si $u = u_1 ... u_n \in \Sigma^*$ alors $\varphi(u) = \varphi(u_1)...\varphi(u_n)$.
Si L est un langage, on note $\varphi(L) = \{\varphi(u) \mid u \in L\}$.

6. Montrer que si φ est un morphisme de mots, alors $\varphi(\varepsilon) = \varepsilon$.
7. Donner une expression régulière de $\varphi(D_1)$ pour le morphisme de mots φ défini par $\varphi(a_1) = aa$ et $\varphi(\overline{a_1}) = \varepsilon$.
8. Soit φ un morphisme de mots. Montrer que si L est un langage hors-contexte alors $\varphi(L)$ est un langage hors-contexte.

III.2 Théorème de Chomsky-Schützenberger

Soit $G = (\Sigma, V, R, S)$ une grammaire hors-contexte. On dit que G est en forme normale de Chomsky si toutes les règles de production sont de l'une des formes suivantes :

- $X \longrightarrow a$, avec $a \in \Sigma$
- $X \longrightarrow YZ$, avec $Y, Z \in V$

On admet que si G est une grammaire quelconque, alors il existe une grammaire G' en forme normale de Chomsky telle que $L(G') = L(G) \setminus \{\varepsilon\}$.

9. Donner sans justification le langage engendré par la grammaire G_0 en forme normale de Chomsky définie par les règles suivantes :
   - $S \longrightarrow AX \mid AB$
   - $X \longrightarrow SB$
   - $A \longrightarrow a$
   - $B \longrightarrow b$

On veut démontrer :

Théorème : Chomsky-Schützenberger

Un langage L est hors-contexte si et seulement il existe un langage régulier K, un langage de Dyck D_n et un morphisme de mots φ tels que $L = \varphi(D_n \cap K)$.

10. On suppose L hors-contexte et K régulier sur un même alphabet Σ. En considérant $A = (\Sigma, Q, \delta, q_0, F)$ un automate fini déterministe reconnaissant K et $G = (\Sigma, V, R, S)$ une grammaire en forme normale de Chomsky engendrant L, montrer que $L \cap K$ est un langage hors-contexte.
    Pour cela, on construira une grammaire G' ayant un symbole initial S' et une variable $X_{p,q}$ pour toute variable $X \in V$ et tout états $p, q \in Q$.
11. En déduire un sens du théorème.

Soit $G = (\Sigma, V, R, S)$ une grammaire hors-contexte en forme normale de Chomsky. On numérote les règles de la forme $X \longrightarrow YZ$ par $r_1, r_2, ..., r_k$. On pose $G' = (\Sigma', V, R', S)$ où :

- $\Sigma' = \Sigma \cup \{\overline{a} \mid a \in \Sigma\} \cup \displaystyle\bigcup_{i=1}^{k} \{a_i, \overline{a_i}, b_i, \overline{b_i}, c_i, \overline{c_i}\}$;
- $R' = \{X \longrightarrow a_i b_i Y \overline{b_i} c_i Z \overline{c_i}\,\overline{a_i}, \text{pour } i \in [\![1, k]\!] \text{ et } r_i = X \longrightarrow YZ\} \cup \{X \longrightarrow a\overline{a}, \text{pour } X \longrightarrow a \in P\}$.

12. Montrer que $L(G')$ est inclus dans un langage de Dyck d'ordre n, pour n bien choisi.

On pose $\varphi : \Sigma'^* \longrightarrow \Sigma^*$ le morphisme de mots défini par :

- pour $a \in \Sigma$, $\varphi(a) = a$ et $\varphi(\overline{a}) = \varepsilon$;

- pour $i \in [\![1, k]\!]$, $\varphi(a_i) = \varphi(\overline{a_i}) = \varphi(b_i) = \varphi(\overline{b_i})\varphi(c_i) = \varphi(\overline{c_i}) = \varepsilon$.

13. Montrer que $L(G) = \varphi(L(G'))$.

Pour L un langage sur un alphabet Σ, on note $P(L) \subset \Sigma$ l'ensemble des premières lettres des mots de L, $F(L) \subset \Sigma^2$ l'ensemble des facteurs de taille 2 des mots de L et $N(L) = \Sigma^2 \setminus F(L)$.

14. On pose $K = P(L(G'))\Sigma'^* \setminus \Sigma'^* N(L(G'))\Sigma'^*$. Montrer que K est un langage régulier.

15. Montrer le théorème de Chomsky-Schützenberger.