I Mines-Telecom

I.1 Exercice 1

Soit t un tableau de taille n dont les éléments sont entre 0 et $n-1$ (inclus).
On veut déterminer si t contient un doublon, c'est-à-dire un élément apparaissant plusieurs fois.

1. Donner un algorithme en complexité temporelle $\mathrm{O}(n)$ pour résoudre ce problème. Quelle est la complexité spatiale ?
2. Peut-on adapter l'algorithme précédent si les éléments de t ne sont pas entre 0 et $n-1$? pas forcément entiers ?
3. On reprend l'hypothèse où les éléments de t sont entre 0 et $n-1$.
   Décrire un algorithme en complexité $\mathrm{O}(n)$ en temps et $\mathrm{O}(1)$ en mémoire. On pourra modifier t.

I.2 Exercice 2

Soit $w \in \Sigma^\star$. On dit que le mot w est un palindrome si $\tilde{w} = w$.

1. Écrire une fonction palindrome de signature `string -> bool` qui teste, en temps linéaire, si un mot est un palindrome.

Pour un alphabet Σ, on note $\mathrm{Pal}(\Sigma)$ l'ensemble des palindromes de $\Sigma^\star$.

2. Montrer que si Σ est un alphabet à une lettre, alors $\mathrm{Pal}(\Sigma)$ est rationnel.
3. Montrer que si Σ contient au moins deux lettres, alors $\mathrm{Pal}(\Sigma)$ n'est pas rationnel.
   On pourra utiliser un automate et un mot de $\mathrm{Pal}(\Sigma) \cap a^\star b a^\star$.

Soit $L \subset \Sigma^\star$ un langage reconnu par l'automate $A = (Q, I, F, T)$.

Pour $(q, q') \in Q^2$, on note $L_{q,q'}$ le langage de tous les mots w qui étiquettent un chemin dans A partant de q et arrivant en q'.

4. Montrer que $L_{q,q'}$ est reconnaissable et exprimer le langage L_A en fonction de langages $L_{q,q'}$.
5. Montrer que $\mathrm{Pal}(\Sigma) \cap \left(\Sigma^2\right)^\star = \{u\tilde{u} \mid u \in \Sigma^\star\}$.

Soit L un langage rationnel reconnu par un automate $A = (Q, I, F, T)$. On définit les langages $D(L) = \{w\tilde{w} \mid w \in L\}$ et $R(L) = \{w \in \Sigma^\star \mid w\tilde{w} \in L\}$.

6. Décrire simplement les langages $D\left(a^\star b\right)$ et $R\left(a^\star b^\star a^\star\right)$.
7. Les langages $D(L)$ et $R(L)$ sont-ils reconnaissables? On pourra faire intervenir les langages $L_{q,q'}$, définis ci-dessus.

II Mines-Telecom

II.1 Exercice 1

On considère ici des arbres binaires stricts (chaque noeud possède 0 ou 2 fils). Étant donné un arbre a et un sommet s de a, on définit $p(s, a)$ comme la profondeur de s dans a.

1. Démontrer l'égalité suivante (appelée égalité de Kraft), pour tout arbre binaire strict a :

$$\sum_{f \in \mathcal{F}(a)} 2^{-p(f,a)} = 1$$

   où $\mathcal{F}(a)$ est l'ensemble des feuilles de a.

2. Est-ce que cette égalité subsiste pour un arbre binaire non strict?

II.2 Exercice 2

Soit $G = (V, E)$ un graphe non orienté à n sommets et p arêtes. Pour $S \subseteq V$, on définit la fonction de densité par :

$$\rho(S) = \frac{|E(S)|}{|S|}$$

où $E(S)$ est l'ensemble des arêtes de G ayant leurs deux extrémités dans S.

4. Quelles sont les valeurs minimum et maximum de $\rho(S)$, en fonction de $|S|$?
5. Quel est le lien entre $\rho(S)$ et le degré moyen des sommets dans S ?

On s'intéresse au problème suivant :

DENSEST

Entrée : un graphe $G = (V, E)$.
Sortie : un ensemble $S \subseteq V$ tel que $\rho(S)$ soit maximum.

On propose un algorithme glouton pour DENSEST :

- Itérativement retirer un sommet de degré minimum (ainsi que tous les sommets adjacents) jusqu'à ce qu'il n'y ait plus de sommet.
- À chacune de ces itérations, calculer la valeur de ρ et conserver le maximum.

6. Expliquer comment on pourrait implémenter cet algorithme en complexité temporelle $\mathrm{O}(n + p)$.

Soit S^* tel que $\rho(S^*)$ soit maximum, $v^* \in S^*$ le premier sommet de S^* retiré par l'algorithme glouton et S' l'ensemble des sommets restants juste avant de retirer v^*.

7. Montrer que $\rho(S') \geq \dfrac{\deg_{S'}(v^*)}{2}$, où $\deg_{S'}(v^*)$ est le degré de v^* dans S'.
8. Justifier que $\rho(S^*) \geq \rho(S^* \setminus \{v^*\})$.
9. En déduire que $\deg_{S^*}(v^*) \geq \rho(S^*)$.
10. En déduire que l'algorithme glouton est une 2-approximation pour DENSEST.

III CCP : Déduction naturelle + Arbre de segments

III.1 Sujet A

On rappelle les règles suivantes :

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to_i \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B} \to_e \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_i \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_e^g \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge_e^d \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A} \bot_e$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_i^g \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee_i^d \qquad \frac{\Gamma, A \vdash C \qquad \Gamma, B \vdash C \qquad \Gamma \vdash A \vee B}{\Gamma \vdash C} \vee_e \qquad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \neg_i \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash \bot} \neg_e$$

Prouver les séquents suivants, en utilisant les règles ci-dessus.

1. $P \wedge Q \vdash Q \wedge P$
2. $P \to Q, P \vee Q \vdash Q$
3. $\vdash \neg(P \wedge \neg P)$

III.2 Sujet B (programmation)

Soit `a` un tableau d'entiers.
On souhaite concevoir, à partir de `a`, une structure de donnée `t` permettant de réaliser efficacement les opérations suivantes :

- `min_range i j t` : renvoie le minimum des éléments de `a` entre les indices `i` et `j`.
- `set i e t` : met à jour la structure pour que `a.(i)` soit remplacé par `e`.

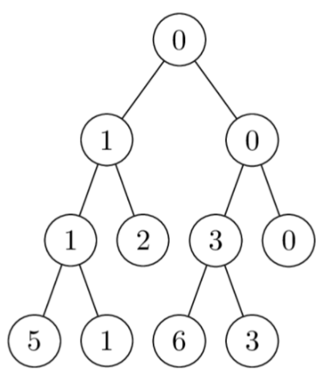
1. Proposer une solution naïve et donner sa complexité.

Dans la suite, on va utiliser un arbre de segments :

Définition : Arbre de segments

Un arbre de segments (pour un tableau `a`) est un arbre binaire dont :
- Les feuilles sont les éléments de `a`
- Chaque noeud est étiqueté par un triplet `(m, i, j)` tel que son sous-arbre contienne les feuilles `a.(i), ... , a.(j)` et `m` est le minimum de ces valeurs.

Par exemple, voici un arbre de segments obtenu à partir du tableau `[|5; 1; 2; 6; 3; 0|]`, où on a représenté seulement les minimums (premiers éléments de chaque triplet) :



Ainsi, les feuilles sont bien les éléments du tableau `[|5; 1; 2; 6; 3; 0|]` et chaque noeud correspond à un minimum sur une certaine plage du tableau.
Remarque : Il y a d'autres arbres de segments possibles pour le même tableau.
On utilisera le type suivant :

```
type tree = E | N of int * int * int * tree * tree
```

Ainsi, un sous-arbre `N(m, i, j, g, d)` possède `a.(i)`, `a.(i + 1)`, ..., `a.(j)` comme feuilles, de minimum `m` et de sous-arbres `g`, `d`.

2. Écrire une fonction `make : int array -> tree` qui construit un arbre de segments à partir d'un tableau d'entiers. On fera en sorte que l'arbre construit soit de hauteur logarithmique en la taille du tableau.
3. Écrire une fonction `set i e t` qui met à jour `t` en remplaçant `a.(i)` par `e`.
   Quelle est la complexité de cette fonction?
4. Écrire une fonction `min_range i j t` renvoyant le minimum des éléments de `a` entre les indices `i` et `j`.
5. Montrer que la complexité de `min_range i j t` est $\mathrm{O}(\log(n))$, où n est la taille de `a`.
6. On s'intéresse à un autre problème : calculer efficacement une somme d'éléments entre les indices `i` et `j`, dans un tableau.
   Adapter les fonctions précédentes pour y parvenir.

IV CCP : Kraft + Dyck

IV.1 Sujet A

On considère ici des arbres binaires stricts (chaque noeud possède 0 ou 2 fils). Étant donné un arbre a et un sommet s de a, on définit $p(s, a)$ comme la profondeur de s dans a.

1. Démontrer l'égalité suivante (appelée égalité de Kraft), pour tout arbre binaire strict a :

$$\sum_{f \in \mathcal{F}(a)} 2^{-p(f,a)} = 1$$

où $\mathcal{F}(a)$ est l'ensemble des feuilles de a.

2. Est-ce que cette égalité subsiste pour un arbre binaire non strict?

IV.2 Sujet B (programmation)

On appelle mot de Dyck un mot m sur l'alphabet $\{a, b\}$ tel que :

- m contient autant de a que de b;
- tout préfixe m contient plus de a que de b;

Par exemple, $m = aababb$ est un mot de Dyck, car il possède trois a et trois b, et ses préfixes sont ε (le mot vide), a, aa, aab, aaba, aabab et m, et aucun ne contient strictement plus de b que de a.

1. Parmi les mots suivants, indiquer ceux qui sont de Dyck. On ne demande pas de preuve.

$$\varepsilon \quad aabbbaab \quad aaab \quad abab \quad aaabbb$$

On représente dans la suite en OCaml un mot sur l'alphabet $\{a, b\}$ par la liste de ses lettres. On définit les types suivants :

```
type lettre = A | B
```

```
type mot = lettre list
```

Le mot $m = aababb$ sera donc représenté par la liste `[A; A; B; A; B; B]`.

2. Écrire une fonction `verifie_dyck` de type `mot -> bool` renvoyant un booléen indiquant si le mot passé en entrée est de Dyck.

On admet la propriété suivante : tout mot m de Dyck non vide se décompose de manière unique sous la forme $m = aubv$, où u et v sont des mots de Dyck. On pourra utiliser sans démonstration la caractérisation suivante : aub est le plus petit préfixe non vide de m contenant autant de a que de b.

3. Donner sans démonstration la décomposition de chacun des mots de Dyck suivants :

$$ab \quad aababb \quad ababab \quad aabbab$$

4. Écrire une fonction `decompo_dyck` de type `mot -> mot * mot`, prenant en entrée un mot m supposé non vide et de Dyck (il est inutile de le vérifier), et renvoyant le couple de mots de Dyck (u, v) tel que $m = aubv$.

Il existe une bijection naturelle entre les arbres binaires stricts (tout nœud de l'arbre possède 0 ou 2 fils) et les mots de Dyck, basée sur la décomposition précédente :

- au mot vide est associé l'arbre réduit à une feuille;
- à un mot de Dyck non vide $aubv$ où u et v sont de Dyck, on associe l'arbre binaire où le sous-arbre gauche est associé au mot u et le sous-arbre droit au mot v.

5. Dessiner l'arbre associé au mot de Dyck $m = aababb$. Les nœuds ne portent pas d'étiquettes.

On définit le type suivant :

```
type arbre = F | N of arbre * arbre;;
```

6. Écrire une fonction `mot_a_arbre` de type `mot -> arbre` renvoyant l'arbre binaire strict associé à un mot de Dyck (on ne vérifiera pas que le mot passé en entrée est bien de Dyck).

7. Écrire une fonction `arbre_a_mot` de type `arbre -> mot` faisant l'inverse.
8. Montrer que le language L des mots de Dyck n'est pas rationnel.
9. Soit $C(n)$ le nombre de mots de Dyck de longueur n. Trouver une équation de récurrence sur $C(n)$, puis montrer que $C(n) = \dfrac{1}{n+1}\dbinom{2n}{n}$.