Devoir maison n°5

À rendre le lundi 24/11

⋆ ⋆ ⋆

Jeux impartiaux et automates à repli

Les deux parties sont indépendantes.

Les questions de programmation seront traitées en langage C. On supposera que les entêtes `stdlib.h`, `stdbool.h` et `stdio.h` ont été chargées.

1 Étude de jeux impartiaux

1.1 Introduction

Un jeu **impartial** est un jeu en tour par tour dans lequel les coups possibles ne dépendent que de la position et non du joueur dont c'est le tour. Cela signifie qu'il y a une symétrie entre les deux joueurs sur le graphe du jeu. On suppose pour ce sujet qu'un joueur perd s'il est le premier à ne plus pouvoir jouer. Formellement, si $G = (S, A)$ est un graphe orienté, un jeu impartial sur G peut être défini comme (G_I, s_0, T_1, T_2) où :

- $G_I = (S \times \{1, 2\}, A_I)$ est un graphe orienté biparti tel que :

$$A_I = \{((s, 1), (t, 2)) \mid (s, t) \in A\} \cup \{((s, 2), (t, 1)) \mid (s, t) \in A\}$$

- En notant T l'ensemble des sommets de degré sortant nul de G, on a :

$$T_1 = T \cap (S \times \{2\}) \qquad T_2 = T \cap (S \times \{1\})$$

- $s_0 \in S \times \{1, 2\}$ est le sommet initial.

On remarque en particulier qu'il n'y a pas de match nul dans un jeu impartial.

Question 1 Parmi les jeux suivants, indiquez, en justifiant brièvement, s'ils sont impartiaux ou non : Morpion, Chomp, Domineering, Hex, Nim.

Ces jeux ont été définis en cours ou en TD.

Pour un jeu impartial défini avec les notations précédentes, on appelle **graphe simplifié** le graphe $G = (S, A)$. Par exemple, la figure 1 représente le graphe simplifié du jeu de Nim $\star 4$.

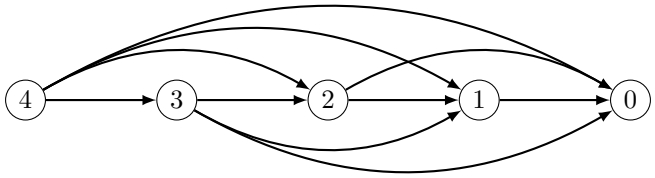

FIGURE 1 – Le graphe simplifié de $\star 4$

Question 2 Représenter graphiquement le graphe simplifié de $*3 + *2$. On choisira des noms pertinents pour les sommets.

On définit le jeu de Wythoff de la manière suivante : deux boîtes contiennent des pièces. À son tour, un joueur peut soit enlever un nombre non nul de pièces dans l'une des boîtes, soit enlever le même nombre non nul de pièces dans chacune des boîtes. On note $W_{m,n}$ le jeu de Wythoff commençant par deux boîtes contenant m et n pièces respectivement. Le joueur qui retire la dernière pièce gagne la partie (car alors l'autre ne peut plus jouer).

Question 3 Donner une définition formelle du graphe simplifié de $W_{m,n}$.

Question 4 Écrire une fonction récursive `bool wythoff_naif(int m, int n)` qui détermine si le premier joueur à jouer a une stratégie gagnante dans le jeu $W_{m,n}$. On demande de ne pas utiliser de mémoire supplémentaire autre que la pile d'appels récursifs.

Question 5 On note $C(m, n)$ le nombre total d'appels récursifs à `wythoff_naif` effectués lors de l'appel `wythoff_naif(m, n)`. Montrer que $C(m, n) \geqslant 2^{\max(m,n)}$.

Question 6 Quelle technique de programmation peut être utilisée pour améliorer la complexité temporelle de la fonction précédente, aux dépens de la complexité spatiale ?

Implémenter une fonction `bool wythoff(int m, int n)` effectuant ce calcul. Indiquer sa complexité temporelle.

1.2 Nombre de Grundy

On définit l'opération mex (pour **minimum exclu**) pour un ensemble $X \subsetneq \mathbb{N}$, comme le plus petit entier naturel qui n'est pas dans X, c'est-à-dire :

$$\mathrm{mex}(X) = \min(\mathbb{N} \setminus X)$$

Question 7 Écrire une fonction `int mex(int* X, int n)` qui prend en argument un tableau d'entiers X de taille n et renvoie $\mathrm{mex}(X)$. La fonction devra avoir une complexité linéaire en n et on demande de justifier.

Pour un jeu impartial de graphe simplifié $G = (S, A)$, supposé sans cycle, et $s \in S$, on note $V(s)$ l'ensemble des voisins de s. On définit la fonction **nombre de Grundy** $g : S \to \mathbb{N}$ par induction par :

$$\forall s \in S, g(s) = \mathrm{mex}\{g(t) \mid t \in V(s)\}$$

Question 8 L'absence de cas de base explicite est-elle un problème pour la définition par induction de g ? Expliquer.

Question 9 Calculer $g((3, 2))$ dans le graphe simplifié associé à $*3 + *2$.

Question 10 Montrer que le premier joueur à jouer depuis $s_0 \in S$ a une stratégie gagnante si et seulement si $g(s_0) \neq 0$.

On représente un graphe simplifié à l'aide du type suivant :

```c
struct Graphe {
    int n;
    int* degres;
    int** adj;
};

typedef struct Graphe graphe;
```

tel que si `G` est un objet de type `graphe` représentant $G = (S, A)$, alors `G.n` $= |S| = n$, $S = [\![0, n-1]\!]$, pour $s \in S$, `G.degres[s]` est le degré sortant de s et `G.adj` est un tableau de taille n tel que pour $s \in S$, `G.adj[s]` est un tableau de taille `G.degres[s]` contenant les voisins de s dans un ordre quelconque.

Question 11 Écrire une fonction `int* grundy(graphe G)` qui prend en argument un graphe simplifié $G = (S, A)$ de jeu impartial et renvoie un tableau `g` de taille $|S|$ tel que pour $s \in S$, `g[s]` $= g(s)$.

Cette fonction devra avoir une complexité temporelle en $\mathcal{O}(|S| + |A|)$ et on demande de justifier.

2 Automates à repli

Une **occurrence** d'un mot $u = a_1 \dots a_k$ (appelée **motif**) dans un autre mot $v = b_1 \dots b_n$ est un entier naturel $i \in [\![1, n]\!]$ tel que pour $j \in [\![0, k-1]\!]$, $a_{k-j} = b_{i-j}$. En d'autres termes, l'occurrence indique la position de la dernière lettre du motif u au sein du mot v (les positions commençant à 1).

Par exemple, si $\Sigma = \{a, b, c, d, e\}$, $u = abc$ et $v = abcabcdababcdabcdabde$, alors il y a quatre occurrences de u dans v, à savoir aux indices 3, 6, 12 et 16.

Pour toute la suite, on suppose que les mots sont formés de caractères ASCII compris entre 0 et 255.

Question 12 Écrire une fonction `void recherche_naive(char* u, char* v)` qui **affiche** toutes les occurrences de u dans v. On ne cherchera pas à faire d'optimisation particulière.

Question 13 Déterminer la complexité de la fonction précédente en fonction des longueurs de u et v.

2.1 Automates finis déterministes à repli

Un **automate fini déterministe à repli** (ou AFDR) sur Σ est un quadruplet $\mathcal{A} = (k, F, \delta, \rho)$ tel que :

- $k \in \mathbb{N}^*$ représente le **nombre d'états** de $\mathcal{A}$; l'ensemble des **états** de $\mathcal{A}$ est $Q_{\mathcal{A}} = [\![0, k-1]\!]$ et 0 est appelé **état initial** ;
- $F \subseteq Q_{\mathcal{A}}$ est un ensemble d'états appelés **finals** ;
- $\delta : Q_{\mathcal{A}} \times \Sigma \to Q_{\mathcal{A}}$ est une **fonction partielle de transition** (c'est-à-dire une fonction dont le domaine de définition est un sous-ensemble de $Q_{\mathcal{A}} \times \Sigma$) ; on impose que $\delta(0, a)$ est défini pour tout $a \in \Sigma$;
- $\rho : Q_{\mathcal{A}} \setminus \{0\} \to Q_{\mathcal{A}}$ est une application (c'est-à-dire une fonction totale) appelée **fonction de repli** telle que pour tout $q \in Q_{\mathcal{A}} \setminus \{0\}$, $\rho(q) < q$. On prolonge ρ en convenant $\rho(0) = 0$.

Un AFDR est représenté en C par le type suivant :

```c
struct AFDR {
    int k;
    bool* F;
    int** delta;
    int* rho;
};

typedef struct AFDR afdr;
```

où si `A` est de type `afdr` représentant l'AFDR $\mathcal{A} = (k, F, \delta, \rho)$, alors :

- `A.k` $= k$;
- `A.F` est un tableau de booléens de taille k tel que `A.F[q]` vaut `true` si et seulement si $q \in F$;
- `A.delta` est une matrice d'entiers de taille $k \times 256$ telle que `A.delta[q][a]` contient `-1` si $\delta(q, a)$ n'est pas défini, et contient $\delta(q, a)$ sinon ;
- `A.rho` est un tableau d'entiers de taille k tel que `A.rho[0]` contient `0` et `A.rho[q]` contient $\rho(q)$ si $q \neq 0$.

Si `a` est un caractère de type `char`, on autorisera son utilisation comme indice de tableau compris entre 0 et 255, comme si c'était un entier. On suppose de plus disposer de fonctions :

- `afdr copie_afdr(afdr A)` qui prend en argument un AFDR et renvoie une copie non superficielle (c'est-à-dire sans liaison de données) ;
- `void liberer_afdr(afdr A)` qui libère la mémoire allouée pour un AFDR.

On dessine un AFDR comme un automate fini déterministe classique, en faisant figurer par des flèches en pointillés les replis d'un état vers un autre. Les états finals sont figurés par un double cercle.

Par exemple, si on définit l'automate $\mathcal{A}_1 = (4, \{3\}, \delta_1, \rho_1)$ sur l'alphabet $\Sigma = \{a, b\}$ où les fonctions δ_1 et ρ_1 sont définies par les tables de la figure 2, alors l'automate $\mathcal{A}_1$ pourra être représenté par la figure 3.

δ_1

q	$\delta_1(q,a)$	$\delta_1(q,b)$
0	1	0
1		2
2	3	
3		

ρ_1

q	$\rho_1(q)$
0	
1	0
2	0
3	1

FIGURE 2 – Les tables de transition et de repli de $\mathcal{A}_1$.

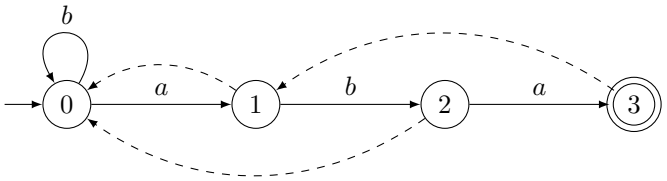

FIGURE 3 – L'automate $\mathcal{A}_1$.

Question 14 Soit $\mathcal{A} = (k, F, \delta, \rho)$ un AFDR. Montrer que $\forall (q, a) \in Q_{\mathcal{A}} \times \Sigma, \exists j \in \mathbb{N}, \delta(\rho^j(q), a)$ est défini (ρ^j est l'itérée j fois de ρ).

Pour $\mathcal{A} = (k, F, \delta, \rho)$, on note $\widetilde{\rho} : Q_{\mathcal{A}} \times \Sigma \to Q_{\mathcal{A}}$ l'application qui à un couple $(q, a) \in Q_{\mathcal{A}} \times \Sigma$ associe l'état $\rho^j(q)$ où j est le plus petit entier (éventuellement nul) tel que $\delta(\rho^j(q), a)$ est défini.

On dit que $\mathcal{A}$ accepte un mot $u = a_1 \ldots a_p \in \Sigma^*$ s'il existe une suite finie $q_0, q'_1, q_1, q'_2, \ldots, q'_{p-1}, q_{p-1}, q'_p, q_p$ d'états de $\mathcal{A}$ avec :

- $q_0 = 0$;
- pour $i \in [\![1, p]\!]$, $q'_i = \widetilde{\rho}(q_{i-1}, a_i)$;
- pour $i \in [\![1, p]\!]$, $q_i = \delta(q'_i, a_i)$;
- $q_p \in F$.

Le langage accepté par $\mathcal{A}$, noté $L(\mathcal{A})$ est l'ensemble des mots acceptés par $\mathcal{A}$.

Par exemple, l'AFDR $\mathcal{A}_1$ accepte le mot *ababa* comme le montre la suite d'états $0, 0, 1, 1, 2, 2, 3, 1, 2, 2, 3$.

On remarque qu'un AFDR dont la fonction de transition δ est définie partout peut être vu comme un automate fini déterministe classique, et que cet automate est **complet**. En effet, les puissances non nulles de

la fonction de repli ρ ne sont utilisées que si la fonction de transition n'est pas définie partout. On notera un tel automate un AFDC.

Question 15 Représenter graphiquement sans justification un AFDC sur $\Sigma = \{0, 1\}$ acceptant le même langage que $\mathcal{A}_1$ et ayant le même nombre d'états que $\mathcal{A}_1$.

Question 16 Donner sans justification une description concise du langage accepté par $\mathcal{A}_1$.

Question 17 Écrire une fonction `afdr enleve_repli(afdr A)` qui renvoie un AFDC acceptant le même langage que $\mathcal{A}$. Cette fonction doit avoir une complexité temporelle en $\mathcal{O}(k)$ où k est le nombre d'états de $\mathcal{A}$ et il faudra justifier cette complexité.

Question 18 Écrire une fonction `void occurrences(afdr A, char* u)` qui prend en argument un AFDC $\mathcal{A}$ et un mot $u = a_1 \dots a_n \in \Sigma^*$ et **affiche** tous les entiers $i \in [\![1, n]\!]$ tels que le préfixe $a_1 \dots a_i$ est accepté par $\mathcal{A}$. La fonction devra avoir une complexité linéaire en n.

2.2 Automate de Knuth-Morris-Pratt

L'**automate de Knuth-Morris-Pratt** (ou automate KMP) associé à un motif $u = a_1 \dots a_k$ sur l'alphabet Σ est un AFDR $\mathcal{A}_u^{\text{KMP}} = (k', F, \delta, \rho)$ où :

- $k' = k + 1$;
- $F = \{k\}$;
- pour $i \in [\![1, k]\!]$, $\delta(i-1, a_i) = i$, pour $a \in \Sigma \setminus \{a_1\}$, $\delta(0, a) = 0$ et aucune autre transition n'est définie ;
- pour $i \in [\![1, k]\!]$, $\rho(i)$ est le plus grand entier $j \in [\![0, i-1]\!]$ tel que $a_1 \dots a_j$ est un **suffixe** de $a_1 \dots a_i$.

On peut vérifier que l'automate $\mathcal{A}_1$ de la question précédente est l'automate KMP associé à aba sur l'alphabet $\Sigma = \{a, b\}$.

Question 19 Représenter graphiquement sans justification l'automate KMP $\mathcal{A}_{ababc}^{\text{KMP}}$ sur $\Sigma = \{a, b, c\}$.

Question 20 Donner sans justification une description concise du langage accepté par l'automate $\mathcal{A}_u^{\text{KMP}}$ pour un motif u quelconque.

Si $u = a_1 \dots a_k$ est un motif et $\mathcal{A}_u^{\text{KMP}} = (k', F, \delta, \rho)$ est son automate KMP, alors pour $i \in [\![1, k]\!]$, on note j_i le plus petit entier tel que $\delta(\rho^{j_i}(\rho(i-1)), a_i)$ est défini.

On admet que pour $i \in [\![2, k]\!]$, $\rho(i) = \delta(\rho^{j_i}(\rho(i-1)), a_i)$.

Question 21 En déduire une fonction `afdr KMP(char* u)` qui prend en entrée un motif u et renvoie l'automate KMP associé.

Question 22 Montrer que pour $i \in [\![1, k]\!]$, $\rho(i) \leqslant \rho(i-1) + 1 - j_i$ et en déduire que $\sum_{i=1}^{k} j_i = \mathcal{O}(k)$.

Question 23 Déterminer en justifiant la complexité de la fonction `KMP` en fonction de la longueur k du motif en argument.

Question 24 Écrire une fonction `void recherche_KMP(char* u, char* v)` qui **affiche** la liste des occurrences de u dans v. Déterminer sa complexité et comparer avec celle de la fonction `recherche_naive`.

⋆ ⋆ ⋆