COMPOSITION D'INFORMATIQUE n°3

Sujet 1 – Corrigé

⋆ ⋆ ⋆

Jeu de synchronisation

1 Machines synchronisées

Question 1 Sur une machine à un seul état q, les transitions sont exactement les $\delta(q,a) = q$ pour $a \in \Sigma$, et on en déduit que tous les mots sont synchronisants.

Question 2 On remarque que si $|u| \equiv 0[2]$, alors $\delta^*(q_i,u) = q_i$ et si $|u| \equiv 1[2]$, alors $\delta^*(q_i,u) = q_{1-i}$ (ce résultat se prouve aisément par récurrence). On en déduit que pour tout mot u, $\delta^*(q_0,u) \neq \delta^*(q_1,u)$, donc qu'il n'existe pas de mot synchronisant.

Question 3 On remarque que la lecture d'un a emmène à l'état q_1 ou l'état q_2. De plus, la lecture de da depuis l'un de ces deux états emmène nécessairement vers l'état q_1. On en déduit que ada est un mot synchronisant pour M_2. D'autres mots peuvent convenir, comme bca, bdb, aca, bcb, acb, adb.

Question 4 On calcule une transition tant qu'on n'est pas arrivé sur le caractère de fin du mot `u`. On utilise la fonction `code` pour passer d'un caractère à un entier dans le bon intervalle.

```
int delta_etoile(machine M, int q, char* u){
    for (int i=0; u[i] != '\0'; i++){
        q = M.delta[q][code(u[i])];
    }
    return q;
}
```

Question 5 Il faut tester pour chaque état que la lecture du mot emmène vers un unique état. Pour cela, on fait le calcul depuis l'état 0, puis on utilise une boucle pour vérifier que l'image depuis chaque autre état mène bien au même état.

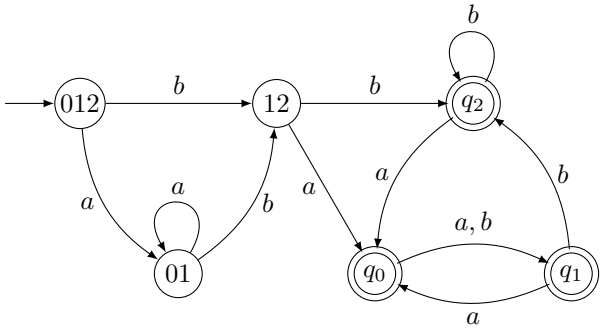
```
bool synchronisant(machine M, char* u){
    int p = delta_etoile(M, 0, u);
    for (int q=1; q<M.Q; q++){
        if (delta_etoile(M, q, u) != p) return false;
    }
    return true;
}
```

Question 6 On remarque que s'il existe un mot synchronisant u pour M, tel que $\forall q \in Q$, $\delta^*(q,u) = q_0$, alors $\widehat{\delta}^*(Q,u) = \{q_0\}$. On en déduit qu'il existe un mot synchronisant pour M si et seulement si un singleton est accessible depuis l'état $Q \in \widehat{Q}$ dans $\widehat{M}$.

Question 7 On définit l'automate déterministe $(\widehat{Q}, \Sigma, \widehat{\delta}, Q, \{\{q\} | q \in Q\})$. D'après la proposition précédente, cet automate reconnaît bien l'ensemble de tous les mots synchronisants de M. On en déduit que $LS(M)$ est reconnaissable.

Question 8 On détermine l'automate des parties sous forme de tableau, puis on le représente en enlevant les états non utiles (l'état initial étant Q) :

	a	b
$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$
$\{q_1, q_2\}$	$\{q_0\}$	$\{q_2\}$
$\{q_0\}$	$\{q_1\}$	$\{q_1\}$
$\{q_1\}$	$\{q_0\}$	$\{q_2\}$
$\{q_2\}$	$\{q_0\}$	$\{q_2\}$



Notons qu'on peut simplifier cet automate en fusionnant q_0, q_1 et q_2 d'une part et 012 et 01 d'autre part.

2 Existence de mot synchronisant

2.1 Machine des paires

Question 9 On a nécessairement $|\delta^*(Q, u_0)| \geqslant |\delta^*(Q, u_1)| \geqslant \ldots \geqslant |\delta^*(Q, u_r)| = 1$. En effet la machine étant déterministe, en lisant un mot depuis un état, on ne peut atteindre qu'un seul état. En lisant un mot depuis un ensemble d'états, le nombre d'états atteignables ne peut que diminuer (en cas de collisions). De plus, comme $u = u_r$ est synchronisant, $|\delta^*(Q, u_r)| = 1$.

Question 10 Le sens direct est le plus simple, car $u_{p,q} = u$ convient (u est synchronisant).

Réciproquement, définissons les suites :

- $u_0 = v_0 = \varepsilon$ et $Q_0 = Q$;
- pour $i \in \mathbb{N}$, si $|Q_i| = 1$, alors on s'arrête, sinon, il existe deux états p et q de Q_i et un mot $u_{i+1} = u_{p,q}$ tel que $|Q_{i+1}| = |\delta^*(Q_i, u_{i+1})| < |Q_i|$. On pose alors $v_{i+1} = v_i u_{i+1}$.

La suite des $(|Q_i|)$ est strictement décroissante et minorée par 1, donc à partir d'un rang $j < n$, on a $|Q_j| = 1$, donc le mot v_j est synchronisant (car $Q_j = \delta^*(Q_0, v_j)$ par définition des suites et de δ^*).

Question 11 Il s'agit ici de coder un parcours en profondeur depuis un état de $\widetilde{M}$ afin de déterminer s'il est possible d'atteindre un singleton. On code pour cela une fonction auxiliaire récursive qui prend en plus un tableau de booléens de taille $\widetilde{n}$ permettant de déterminer si un état a déjà été vu ou non. La fonction renvoie un booléen qui vaut `true` si et seulement si le parcours a permis d'atteindre un état singleton depuis $\{p, q\}$ en passant par des sommets non vus.

```
bool DFS(machine M, int* vus, int p, int q){
    if (p == q) return true;
    int X = phi(p, q);
    if (!vus[X]){
        vus[X] = true;
        for (int i=0; i<M.Sigma; i++){
            int pi = M.delta[p][i];
            int qi = M.delta[q][i];
            if (DFS(M, vus, pi, qi)) return true;
        }
    }
    return false;
}
```

Dès lors, la fonction principale consiste juste à créer le tableau de booléens et à déterminer la valeur de retour pour un appel à `DFS`.

```
bool paire_synchronisee(machine M, int p, int q){
    int nt = (M.Q * (M.Q + 1)) / 2;
    int* vus = malloc(nt * sizeof(vus));
    for (int X=0; X<nt; X++) vus[X] = false;
    bool b = DFS(M, vus, p, q);
    free(vus);
    return b;
}
```

Question 12 On lance un appel à la fonction précédente pour chaque paire d'états. Si l'un des appels renvoie `false`, la machine n'est pas synchronisée. Sinon, elle l'est.

```
bool synchronisee(machine M){
    for (int p=0; p<M.Q; p++){
        for (int q=p+1; q<M.Q; q++){
            if (!paire_synchronisee(M, p, q)) return false;
        }
    }
    return true;
}
```

Question 13 La fonction `paire_synchronisee` consiste à faire un parcours en profondeur, donc de complexité linéaire en nombre d'arêtes + nombre de sommets, soit $\mathcal{O}(|Q|^2|\Sigma)$. On l'appelle autant de fois qu'il y a de paires, c'est-à-dire $\binom{|Q|}{2}$ fois. La complexité totale est donc en $\mathcal{O}(|Q|^4|\Sigma|)$.

Question 14 Au lieu de faire autant de parcours qu'il y a de paires, on peut ne faire qu'un seul parcours, mais dans l'automate transposé (où on a retourné toutes les transitions). L'idée est alors de faire un parcours depuis tous les états singletons puis de vérifier qu'on a atteint tous les autres états. Comme il s'agit d'un seul parcours, on aurait bien une complexité en $\mathcal{O}(|Q|^2|\Sigma|)$.

Question 15 Soit $M = (Q, \Sigma, \delta)$ une machine et q_0 un état de M. On pose $M' = (Q, \Sigma, \delta')$ telle que :

- $\forall q \in Q \setminus \{q_0\}$, $\forall a \in \Sigma$, $\delta'(q, a) = \delta(q, a)$;
- $\forall a \in \Sigma$, $\delta'(q_0, a) = q_0$.

On remarque qu'avec cette construction, si u est un mot synchronisant pour M', alors $\forall q \in Q$, $\delta'^*(q, u) = q_0$. On en déduit que le chemin menant de q à $\delta^*(q, u)$ passe forcément par q_0. Réciproquement, si le chemin menant de q à $\delta^*(q, u)$ passe forcément par q_0, alors u est synchronisant pour M'.

Il s'agit bien d'une réduction many-one de `ÉTAT DE PASSAGE` à `SYNCHRONISÉE`.

2.2 Bornes sur les mots synchronisants

Question 16 On commence par montrer qu'une paire d'états $\{p, q\} \subseteq Q$ peut toujours être synchronisée par un mot $u_{p,q}$ de longueur $\leqslant \binom{n}{2}$. En effet, dans la machine $\widetilde{M}$, un chemin élémentaire de $\{p, q\}$ à un singleton comporte au plus $\binom{n}{2} + 1$ états (les $\binom{n}{2}$ paires et un singleton en dernier état). Le mot lu le long de ce chemin est bien un mot synchronisant $\{p, q\}$.

De plus, en reprenant les notations de la question 10, chaque mot u_i pour $i > 0$ est un mot synchronisant une paire, et le mot synchronisant construit est la concaténation d'au plus $n - 1$ des u_i, donc est de taille $\leqslant (n-1)\binom{n}{2} = \frac{n(n-1)^2}{2}$.

Question 17 Montrons que $u = a(b^{n-1}a)^{n-2}$ est un mot synchronisant pour $\mathcal{C}_n$, en considérant les $\delta^*(Q, u_i)$, pour u_i certains préfixes de u bien choisis. Posons, pour $i \in [\![0, n-2]\!]$, $u_i = a(b^{n-1}a)^i$. Montrons par récurrence sur i que $\delta^*(Q, u_i) = \{1, 2, \ldots, n - i - 1\}$.

- pour $i = 0$, par définition de δ, on a bien $\delta^*(Q, u_0) = \delta(Q, a) = \{1, \ldots, n - 1\}$;
- supposons le résultat vrai pour $i \geqslant 0$ fixé. Sachant que pour $q \in Q$, $\delta(q, b) = (q + 1) \bmod n$, on en déduit par une récurrence rapide que $\delta^*(\{1, 2, \ldots, n - i - 1\}, b^{n-1}) = \{0, 1, \ldots, n - i - 2\}$. Dès lors, $\delta^*(Q, u_{i+1}) = \delta(\delta^*(\delta^*(Q, u_i), b^{n-1}), a) = \delta(\{0, 1, \ldots, n - i - 2\}, a) = \{1, 2, \ldots, n - i - 2\}$.

On conclut par récurrence. Finalement, $\delta^*(Q, u) = \delta^*(Q, u_{n-2}) = \{1\}$, donc u est synchronisant pour $\mathcal{C}_n$, et est bien de taille $1 + n \times (n - 2) = (n - 1)^2$.

Question 18 On commence par remarquer que pour $X \subseteq Q$, alors $\delta(X, b) = \{(q+1) \bmod n \mid q \in X\}$, donc $|\Delta(X)| = |\Delta(\delta(X, b))|$. Ainsi, $\alpha = a$.

De plus, $\delta(X, a) = X \setminus \{0\}$. On en déduit les propriétés suivantes :

- $0 \in X$ et $0 \in \Delta(\delta(X, a))$, sinon $\Delta(X) = \Delta(\delta(X, a))$;
- $1 \in \delta(X, a)$, car $\delta(0, a) = 1$, donc $1 \notin \Delta(\delta(X, a))$;
- en combinant les deux points précédents, $\Delta(\delta(X, a)) = \{n - j, n - j + 1, \ldots, n - 1, 0\}$.

Finalement, $\Delta(X) = \Delta(\delta(X, a)) \setminus \{0\} = \{n - j, n - j + 1, \ldots, n - 1\}$.

Question 19 Soit $u = a_1 a_2 \ldots a_k$ un mot synchronisant pour $\mathcal{C}_n$ de taille minimale. Pour $n = 1$, $u = \varepsilon$, de taille $0 = (1-1)^2$, est synchronisant. Pour $n = 2$, ε n'est pas synchronisant (car il y a deux états), mais $u = a$, de taille $1 = (2-1)^2$ est synchronisant. Pour la suite, supposons $n \geqslant 3$.

On commence par remarquer que $a_1 = a$, car $\delta(Q, b) = Q$, donc si bv est synchronisant, alors v aussi. Par minimalité de la taille de u, $a_1 = a$.

Posons, pour $i = 1, \ldots, k$, $X_i = \delta^*(Q, a_1 a_2 \ldots a_i)$. On a $X_1 = \{1, 2, \ldots, n - 1\}$ (car $a_1 = a$) et $X_k = \{1\}$ (car u est synchronisant et que 1 est le seul état qui a deux transitions entrantes étiquetées par une même lettre).

Soit alors $i \in [\![2, k]\!]$ et $j \in [\![1, n - 2]\!]$ tels que $|\Delta(X_{i-1})| = j$ et $|\Delta(X_i)| = j + 1$. Par la question précédente, $\Delta(X_{i-1}) = \{n - j, n - j + 1, \ldots, n - 1\}$ et $\Delta(X_i) = \{n - j, n - j + 1, \ldots, n - 1, 0\}$.

De plus, soit $\ell > 0$ tel que $|\Delta(X_{i+\ell})| = j + 2$. Alors $\ell \geqslant n$, car b^{n-1} est le mot le plus court qui peut transformer $\Delta(X_i)$ en $\{n - j - 1, n - j, \ldots, n - 1\}$ (on applique b $n - 1$ fois pour faire « tourner » le « trou » autour de l'automate).

Finalement, $k \geqslant 1 + (n - 2) \times n = (n - 1)^2$ (il faut augmenter la taille de $\Delta(X_i)$ un total de $n - 2$ fois, de 1 à $n - 1$).

3 Jeu de synchronisation

Question 20 Il suffit en fait qu'il existe une partie gagnée par 1 pour que M soit synchronisée. Si 1 a une stratégie gagnante, une telle partie existe nécessairement. Soit $a_1, \ldots, a_k$ les lettres choisies par les joueurs

lors d'une partie gagnée par 1 et les X_i définis comme dans l'énoncé. Sachant que $X_{i+1} = \delta(X_i, a_{i+1})$, une récurrence rapide permet de montrer que $X_i = \delta^*(X_0, a_1 \dots a_i)$. Sachant que $X_0 = Q$, pour $u = a_1 \dots a_k$, on a $X_k = \delta^*(Q, u)$ et $|X_k| = 1$ (car 1 remporte la partie). Le mot u est bien synchronisant pour M.

Question 21 Sens direct : raisonnons par contraposée et supposons qu'il existe $p, q \in Q$ tel que le joueur 1 n'a pas de stratégie gagnante depuis $\{p, q\}$. Ainsi, le joueur 2 a une stratégie gagnante depuis $\{p, q\}$. La stratégie gagnante pour 2 depuis $X = Q$ est alors la suivante : le joueur 2 applique la stratégie gagnante depuis $\{p, q\}$ en ignorant les autres pièces (on suppose que si l'une des pièces initialement sur l'état p ou q se retrouve sur un état en même temps qu'une autre pièce, c'est cette autre pièce qui est enlevée).

Sens réciproque : supposons que le joueur 1 peut gagner depuis toute configuration $\{p, q\} \subseteq Q$. La stratégie gagnante pour 1 depuis Q est la suivante :

> **Tant que** il reste au moins deux états $p \neq q$ qui contiennent une pièce. **Faire**
> Appliquer la stratégie gagnante depuis $\{p, q\}$, jusqu'à ce que les pièces sur p et q se retrouvent sur le même état.

À chaque passage dans la boucle **Tant que**, au moins une pièce est retirée du jeu (peut-être plus). Ainsi, il s'agit bien d'une stratégie gagnante pour 1.

Question 22 Soit $n \geqslant 3$. Montrons qu'il existe, dans $\mathcal{C}_n$, une stratégie gagnante pour 2 depuis la configuration $\{1, 3\}$. Par la question précédente, cela montrera qu'il existe une stratégie gagnante pour 2 depuis $X = Q$.

La stratégie est la suivante :

- pour $X = \{0, 2\}$ ou $X = \{0, n-2\}$, on pose $g(X) = b$;
- sinon, $g(X) = a$.

En effet, avec cette stratégie, le joueur 2 garantit qu'il y aura toujours un état sans pièce entre les deux états qui contiennent une pièce.

Soit f une stratégie pour 1 et $(X_i)_{i \in \mathbb{N}}$ une (f, g)-partie depuis $\{1, 3\}$, dont les lettres sont les $(a_i)_{i \in \mathbb{N}^*}$. Montrons que pour $i \in \mathbb{N}$, $X_i = \{p_i, q_i\}$, avec $p_i - q_i \equiv 2 \bmod n$ (en particulier, $|X_i| > 1$), et $0 \in X_i \Rightarrow i$ est impair (c'est-à-dire que c'est au joueur 2 de jouer).

- la propriété est vraie pour $i = 0$ car $X_0 = \{1, 3\}$;
- supposons la propriété vraie pour $i \in \mathbb{N}$ et distinguons :
  - si c'est au joueur 1 de jouer, comme $0 \notin X_i$, alors si $a_{i+1} = a$, $X_{i+1} = X_i$ vérifie toujours les propriétés, et si $a_{i+1} = b$, alors $X_{i+1} = \{p_i + 1 \bmod n, q_i + 1 \bmod n\}$ vérifie toujours les propriétés ;
  - si c'est au joueur 2 de jouer, si $0 \in X_i$, alors $a_{i+1} = b$ et $X_{i+1} = \{p_i + 1 \bmod n, q_i + 1 \bmod n\}$ vérifie toujours les propriétés (en particulier, $0 \notin X_{i+1}$), sinon, $a_{i+1} = a$ et $X_{i+1} = X_i$ vérifie toujours les propriétés.

On déduit le résultat par récurrence, ce qui montre que la partie est gagnée par le joueur 2.

Question 23 (a) $\Rightarrow$ (c) M étant faiblement acyclique, elle possède un ordre topologique (c'est-à-dire une énumération de $Q = \{q_0, q_1, \dots, q_{n-1}\}$ telle que $q_i \preccurlyeq q_j \Rightarrow i \leqslant j$). Supposons que q_{n-1} est l'unique puits. Pour $X \subseteq Q$, notons $\min X = q_i$ où $i = \min\{j \in [\![0, n-1]\!] \mid q_j \in X\}$. On a les deux propriétés suivantes :

- pour $a \in \Sigma$, $\min \delta(X, a) \succcurlyeq \min X$, par définition de l'ordre topologique ;
- si $X \neq \{q_{n-1}\}$, il existe $a_X \in \Sigma$, $\min \delta(X, a_X) \succ \min X$, car si $X \neq \{q_{n-1}\}$, $\min X \neq q_{n-1}$, donc possède un voisin qui lui est strictement supérieur (car seul le puits n'a pas d'élément strictement supérieur).

On peut alors montrer par récurrence que la stratégie $f : X \mapsto a_X$ est gagnante pour le joueur 1 (avec $X_{\{q_{n-1}\}}$ choisi de manière quelconque) : à son tour, le joueur 1 pourra faire augmenter strictement l'élément minimal où il reste une pièce, jusqu'à atteindre q_{n-1} comme unique état avec une pièce.

(b) $\Rightarrow$ (a) M étant faiblement acyclique, elle possède un ordre topologique $(q_0, \dots, q_{n-1})$. Alors, si M est synchronisée, q_{n-1} est l'unique puits de M (s'il y avait un autre puits, M ne pourrait pas être synchronisée).

(c) $\Rightarrow$ (b) : a déjà été montré à la question 20

4 Recherche de stratégie gagnante

Question 24 L'ensemble Attr_i correspond aux couples (X, j) tels que dans une partie partant de la configuration X (de cardinal 1 ou 2) où c'est au joueur $j \in \{1, 2\}$ de jouer, le joueur 1 a une stratégie gagnante lui permettant de gagner en moins de i coups.

Question 25 Par définition, cette suite est croissante pour l'inclusion. De plus, pour $i \in \mathbb{N}$, $\mathrm{Attr}_i \subseteq (\mathcal{P}_2(Q) \cup \mathcal{P}_1(Q)) \times \{1, 2\}$ qui est un ensemble fini. L'ensemble des cardinaux des Attr_i est donc une suite croissante majorée, donc ultimement stationnaire. Par croissance, la suite $(\mathrm{Attr}_i)_{i\in\mathbb{N}}$ est elle-même stationnaire.

Question 26 En utilisant la question 10, cela revient à montrer que le joueur 1 a une stratégie gagnante sur M depuis une configuration $X = \{p, q\} \in \mathcal{P}_2(Q) \cup \mathcal{P}_1(Q)$ si et seulement si $(X, 1) \in \mathrm{Attr}$.

Sens direct : par récurrence sur la taille maximale k d'une partie gagnée par le joueur 1 en suivant la stratégie gagnante depuis $\{p, q\}$:

- si $k = 0$, alors le joueur 1 gagne sans jouer de coup, donc $|X| = 1$ et $(X, 1) \in \mathrm{Attr}$;
- supposons le résultat vrai pour tout $\ell \leqslant k$, $k \geqslant 0$ fixé et soit X tel que le joueur 1 peut garantir sa victoire en $k + 1$ coups ou moins. Après le premier coup de 1 menant en X', si 1 n'a pas déjà gagné, le joueur 2 ne peut pas empêcher la victoire de 1, donc tous les coups possibles pour 2 mènent à une configuration X'' de laquelle le joueur 1 a une stratégie gagnante en $k-1$ coups ou moins. Par hypothèse de récurrence, $(X'', 1) \in \mathrm{Attr}$, et par définition des Attr_i, $(X', 2) \in \mathrm{Attr}$ et donc $(X, 1) \in \mathrm{Attr}$.

On conclut par récurrence.

Sens réciproque : soit $(X, 1) \in \mathrm{Attr}$. On définit f la stratégie suivante pour le joueur 1 : pour $X \subseteq \widetilde{Q}$, soit i minimal tel que $(X, 1) \in \mathrm{Attr}_i$ (on choisit $f(X)$ arbitrairement si ça n'est pas le cas, mais cela ne doit pas arriver). Par définition, si $|X| > 1$, il existe $a \in \Sigma$ tel que $(\widetilde{\delta}(X, a), 2) \in \mathrm{Attr}_{i-1}$. On pose alors $f(X) = a$. Par définition de l'attracteur, il s'agit bien d'une stratégie gagnante.

Question 27 On va encoder un sommet (X, j) par $\varphi(X) + (j - 1) \times \widetilde{n}$. On commence par déterminer les degrés sortants des sommets de G_M, c'est-à-dire les degrés entrants dans $\widetilde{M}$. Pour ce faire, on parcourt toutes les transitions et on augmente le degré des sommets atteints par une transition.

```
int* degres_GM(machine M){
    int nt = (M.Q * (M.Q + 1)) / 2;
    int* deg = calloc(2 * nt, sizeof(int));
    for (int p=0; p<M.Q; p++){
        for (int q=p; q<M.Q; q++){
            for (int i=0; i<M.Sigma; i++){
                int Xi = phi(M.delta[p][i], M.delta[q][i]);
                deg[Xi]++;
                deg[Xi + nt]++;
            }
        }
    }
    return deg;
}
```

Dès lors, la construction du graphe consiste à parcourir toutes les arêtes et à transposer le graphe. On garde en mémoire le degré courant de chaque sommet pour savoir où rajouter un élément dans la « liste » d'adjacence.

```
graphe construire_GM(machine M){
    int nt = (M.Q * (M.Q + 1)) / 2;
    int* deg = degres_GM(M);
    int* deg_courant = calloc(nt, sizeof(int));
    int** A = malloc(2 * nt * sizeof(int*));
    for (int s=0; s<2*nt; s++) A[s] = malloc(deg[s] * sizeof(int));
    for (int p=0; p<M.Q; p++){
        for (int q=p; q<M.Q; q++){
            int X = phi(p, q);
            for (int i=0; i<M.Sigma; i++){
                int Xi = phi(M.delta[p][i], M.delta[q][i]);
                int j = deg_courant[Xi];
                deg_courant[Xi]++;
                A[Xi][j] = X + nt;
                A[Xi + nt][j] = X;
            }
        }
    }
    free(deg_courant);
    graphe GM = {.S = 2 * nt, .deg = deg, .A = A};
    return GM;
}
```

Question 28 En suivant l'indication, on va calculer le tableau `attr`. Pour ce faire, on commence par calculer les degrés entrants des sommets de G_M :

```
int* degres_entrants(graphe G){
    int* deg_e = calloc(G.S, sizeof(int));
    for (int s=0; s<G.S; s++){
        for (int i=0; i<G.deg[s]; i++) deg_e[G.A[s][i]]++;
    }
    return deg_e;
}
```

Dès lors, on implémente un parcours en profondeur dont le principe est le suivant : on utilise `attr` comme un tableau des sommets vus, et lorsqu'on traite un sommet s, on étudie chacun de ses voisins t et :

- si t est de la forme $(X, 1)$ (donc $< \widetilde{n}$), on relance un parcours depuis t (car dans $\widetilde{M}$, il y a un élément de l'attracteur dans $\widetilde{\delta}(X, \Sigma) \times \{2\}$) ;
- si t est de la forme $(X, 2)$ (donc $\geqslant \widetilde{n}$) et que tous ses prédécesseurs ont été vus (c'est à cela que sert le tableau des degrés entrants, qu'on modifie au fur et à mesure), on relance un parcours depuis t (car tous les éléments de $\widetilde{\delta}(X, \Sigma) \times \{1\}$ sont dans l'attracteur).

```
void DFS_attr(graphe GM, bool* attr, int* deg_e, int s){
    if (attr[s]) return;
    int nt = GM.S / 2;
    attr[s] = true;
    for (int i=0; i<GM.deg[s]; i++){
        int t = GM.A[s][i];
        deg_e[t]--;
        if (t < nt || (t >= nt && deg_e[t] == 0)){
            DFS_attr(GM, attr, deg_e, t);
        }
    }
}
```

Finalement, la fonction `gagnant` lance un parcours depuis chaque (X, j) où X est un singleton. On vérifie

ensuite que tous les sommets sont dans l'attracteur, auquel cas c'est le joueur 1 qui a une stratégie gagnante, ou non.

```
int gagnant (machine M){
    graphe GM = construire_GM(M);
    int* deg_e = degres_entrants(GM);
    bool* attr = calloc(GM.S, sizeof(bool));
    for (int q=0; q<M.Q; q++){
        DFS_attr(GM, attr, deg_e, phi(q, q));
        DFS_attr(GM, attr, deg_e, phi(q, q) + (GM.S / 2));
    }
    int j = 1;
    for (int s=0; s<GM.S; s++){
        if (!attr[s]){
            j = 2;
            break;
        }
    }
    free(attr);
    free(deg_e);
    liberer_graphe(GM);
    return j;
}
```

Question 29 La construction de G_M se fait en parcourant deux fois les arêtes de $\widetilde{M}$, donc en $\mathcal{O}(|Q|^2|\Sigma|)$. Le calcul du joueur gagnant consiste à calculer les degrés entrants (parcours de toutes les arêtes à nouveau), puis à faire un parcours en profondeur (en $\mathcal{O}(|S|+|A|)$).

La complexité totale est $\mathcal{O}(|Q|^2|\Sigma|)$.

Question 30 La conjecture de Černý, énoncée en 1964 et toujours un problème ouvert à ce jour, affirme que toute machine synchronisée à n états possède un mot synchronisant de taille inférieure ou égale à $(n-1)^2$. Cette question n'a donc pas de preuve connue !

⋆ ⋆ ⋆