TP

9

CALCULS D'ATTRACTEURS

Objectif et représentation des données

On se donne une arène de jeu pour un jeu à deux joueurs. Cela signifie qu'on modélise les états de jeu par un graphe orienté chaque sommet étant "contrôlé" par un et un seul des deux joueurs, celui dont c'est le tour de jeu.
Le graphe possède n sommets numérotés de 0 à $n-1$ et m arêtes. Il sera représenté sous forme d'une liste d'adjacence `lAdj`, c'est-à-dire que `lAdj[i]` contient la liste des états successeurs de l'état `i`.
Le contrôle des états est modélisé par une liste `controle` de n entiers valant 0 ou 1 : la valeur de `controle[i]` indique quel joueur contrôle l'état de jeu numéro `i`.

Pour les calculs de complexité on répondra au mieux en fonction de m et n.

Notons que passer en revue toutes les arêtes est théoriquement en $O(n+m)$ puisqu'il faut passer en revue tous les sommets et les arêtes de chaque sommet, mais on pourra ici supposer que le graphe est "connexe" (d'un seul tenant), donc $n \leqslant m$, et remplacer ce $O(n+m)$ par $O(m)$.

Premières manipulations

Q1. Écrire une fonction `nombreAretes(lAdj)` qui renvoie le nombre d'arêtes du graphe. Indiquer la complexité de cette fonction.

Q2. Écrire une fonction `estBiparti(lAdj,controle)` qui indique si le graphe est biparti c'est-à-dire si les successeurs d'un état contrôlé par un joueur sont tous contrôlés par l'autre joueur. La fonction renverra un booléen. Indiquer la complexité de cette fonction.

Calcul d'attracteur sur un graphe biparti

On suppose dans cette section que le graphe de jeu est biparti. On rappelle que dans ce cas, l'attracteur peut se calculer de façon progressive en suivant les formules (où (s,t) désigne l'arc allant du sommet s au sommet t) :

$$\begin{aligned}
\mathrm{Attr}_0(joueur) &= \mathrm{W}_{joueur} \\
\mathrm{Attr}_{i+1}(joueur) &= \mathrm{Attr}_i(joueur) \\
&\quad \cup \left\{ s \in \mathrm{S}_{joueur} \mid \exists (s,t) \in \mathrm{A} \text{ et } t \in \mathrm{Attr}_i(joueur) \right\} \\
&\quad \cup \left\{ s \in \mathrm{S}_{adversaire} \mid s \notin \mathrm{W}_{joueur}, \forall (s,t) \in \mathrm{A}, t \in \mathrm{Attr}_i(joueur) \right\}.
\end{aligned}$$

où W_{joueur} représente un ensemble de sommets tels que si le chemin sur le graphe (la partie) y passe alors *joueur* gagne la partie (condition de victoire de *joueur*).

On remarquera, qu'à chaque étape, suivant que c'est à *joueur* ou à *adversaire* de jouer, seul l'un des deux ressembles précédents à ajouter à $\mathrm{Attr}_i(joueur)$ est non vide.

Alors $\text{Attr}(joueur) = \bigcup_{i \geqslant 0} \text{Attr}_i(joueur)$, est l' **attracteur** pour *joueur*.

On considère un des deux joueurs (désigné par la variable `joueur`, de valeur 0 ou 1), et on souhaite déterminer ses positions gagnantes et des stratégies pour les atteindre. Dans un premier temps, on veut constituer un dictionnaire `attr` des positions gagnantes, qui indique le nombre de coups au bout desquels on est sûr de gagner. Ainsi `attr[i]=k` signifie que l'état `i` est une position gagnante pour ce joueur et qu'il est sûr de gagner en `k` coups.

Pour la suite W_{joueur} sera l'ensemble des sommets contrôlés par l'adversaire et n'ayant pas de successeur.

Q3. Écrire une fonction `initialiseAttracteur(listeAdj,controle,joueur)` qui initialise le dictionnaire `attr` en déterminant les états où le joueur dont le numéro est `joueur` est victorieux. Ce sont les premières entrées du dictionnaire, et elles se voient attribuer la valeur 0. Indiquer la complexité de cette fonction.

Q4. On rédige une petite fonction utilitaire. Soit `nouveaux` une liste d'états qui ne figure pas dans le dictionnaire `attr`. Écrire la fonction `grossirAttracteur(attr,nouveaux,rang)` qui rajoute les éléments de `nouveaux` au dictionnaire `attr` en leur attribuant la valeur `rang`.

Q5. Encore une petite fonction utilitaire. Écrire la fonction `unSuccesseurDansA(listeAdj,sommet,attr)` qui indique si le sommet `sommet` du graphe possède un successeur figurant dans le dictionnaire `attr`.

Q6. On suppose que l'attracteur de *joueur* est correctement calculé jusqu'à la valeur $i =$`rang` et que c'est à *joueur* de jouer.
Écrire une fonction `nouveauxAttr1(listeAdj,attr,rang,controle,joueur)` qui détermine et intègre les éléments de $\left\{s \in S_{joueur} \mid \exists (s,t) \in A \text{ et } t \in \text{Attr}_i(joueur)\right\}$

Q7. Faire le travail analogue avec une fonction `nouveauxAttr2(listeAdj,attr,rang,controle,joueur)` pour les éléments de $\left\{s \in S_{adversaire} \mid s \notin W_{joueur}, \forall (s,t) \in A, t \in \text{Attr}_i(joueur)\right\}$ lorsque c'est à *adversaire* de jouer.

Q8. Faire un calcul complet de l'attracteur de *joueur* à travers une fonction `calculAttracteur(listeAdj,controle,joueur)` qui détermine et renvoie le dictionnaire `attr`.

Tester sur le jeu de Chomp $(2,3)$ dont le graphe et le contrôle sont donnés : sommets 0 à 8 pour le joueur 0 (ceux qui étaient indexés par a dans le cours) et 9 à 16 pour le joueur 1 (ceux qui étaient indexés par e). C'est le joueur 0 qui commence.

Q9. Quelle est la complexité d'un tel calcul d'attracteur ?

Q10. Adapter le code pour qu'il donne une stratégie gagnante pour toute position gagnante.