1

Graphes

« *Des arêtes, des sommets, des poids.* »

1.1 Fête de Noël sans conflit

On considère une grande famille de n personnes avec beaucoup de gens qui ne s'entendent pas, représentée par une matrice $A = (a_{i,j}) \in \mathscr{M}_n$ telle que $a_{i,j} = a_{j,i} = 1$ si i et j ne peuvent pas se voir, 0 sinon. On a deux maisons de famille et on veut partager les n personnes entre ces deux maisons pour les fêtes de Noël, de façon que deux personnes qui ne s'entendent pas soient toujours dans des maisons différentes.

Question 1

Montrer par un exemple qu'il n'est pas toujours possible de répartir les membres de la famille entre les deux maisons pour éviter tout conflit.

Question 2

On va mettre le résultat dans un tableau `maison` de taille n tel que `maison.(i-1)=1` si la personne i est dans la première maison et `maison.(i-1)=-1` si la personne i est dans l'autre maison. Écrire une fonction `partage (a:int array array) : int array` qui teste s'il est possible de faire un partage sans conflit et propose un partage lorsque cela et possible.

Question 3

Deux personnes i et j sont dites en relation d'influence s'il existe une suite k_1, ..., k_l telle que $a_{i,k_1} = a_{k_1,k_2} = ... = a_{k_{l-1},k_l} = a_{k_l,j} = 1$. Montrer que cette relation est une relation d'équivalence. Soit N le nombre de classes d'équivalence, si l'on pose par convention $a_{i,i} = 1$ pour tout i. Montrer qu'un partage sans conflit est possible si et

seulement si il n'existe pas de suite i_1, ..., i_{2l+1} telle que $a_{i_1,i_2} = ... = a_{i_{2l+1},i_1} = 1$. Montrer que le nombre de partages sans conflits est soit 0, soit $2N$.

Question 4

On suppose maintenant qu'il y a trois maisons de famille. Proposer un algorithme qui fasse un partage sans conflits entre les trois maisons lorsque cela est possible. Qu'en pensez-vous ?

CORRIGÉ

Question 1

Il suffit d'une famille de trois personnes qui sont chacune en conflit avec les deux autres.

Question 2

On met la personne 1 dans la maison 1. On met toutes les personnes avec lesquelles elle est en conflit, s'il en existe, dans la maison -1. Puis on la marque comme « vue ». À chaque étape, on cherche une personne i qui a déjà été mise dans une maison mais n'est pas encore vue.

Premier cas de figure : il existe une telle personne i. On vérifie que i n'est pas en conflit avec des gens déjà placés dans la même maison, et on met tous les gens en conflit avec i encore non placés dans l'autre maison. Puis on marque la personne i comme « vue ».

Deuxième cas de figure : il n'existe pas de tel i. On prend alors une personne quelconque qui n'a pas encore été vue, et on la met dans la maison 1, puis on continue.

En termes plus techniques, on peut dire qu'on définit un graphe à partir de la matrice des conflits, avec un sommet pour chaque personne, une arête reliant deux sommets si les personnes correspondantes sont en conflit. L'algorithme consiste à traiter ce graphe composante connexe par composante connexe. Dans chaque composante, toutes les affectations des personnes dans les maisons sont déterminées dès qu'on a placé la première personne (qui par défaut est mise dans la maison 1).

On peut démontrer facilement l'invariant suivant :

Invariant : si i est vue, alors i est placée dans l'une des deux maisons, et toutes les personnes avec lesquelles i est en conflit se trouvent dans l'autre maison.

Ainsi, l'algorithme, lorsqu'il trouve une solution, trouve toujours une solution correcte.

Inversement, l'algorithme échoue lorsqu'il place i dans une maison où se trouve déjà une personne j en conflit avec i. Donc j est placée dans une maison mais n'a pas encore été vue, et l'algorithme n'a donc utilisé que le premier cas de figure entre le moment où j a été placée et celui où i a été placée. On a donc une chaîne de conflits telle qu'il ne peut exister de partage sans conflit (voir question 3).

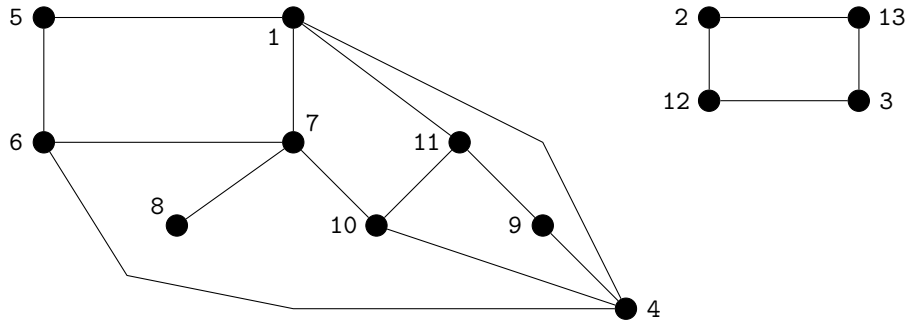
Ceci démontre que l'algorithme résout bien le problème posé.

```
let partage (a:int array array) : int array =
  let n = Array.length a in
```

```
let maison = Array.make n 0 in
let vu = Array.make n false in

for i=0 to n-1 do
  if not vu.(i) then ( (* nouvelle composante connexe *)
    maison.(i) <- 1;
    vu.(i) <- true;
    for j=i+1 to n-1 do (* ajout des personnes en conflit *)
      if a.(i).(j) = 1 then
        maison.(j) <- -1
    done;

    (* recherche d'un personne non vue dans une maison *)
    for j=i+1 to n-1 do
      if maison.(j) <> 0 && not vu.(j) then (
        (* on vérifie que j n'est pas en conflit *)
        for k=i+1 to n-1 do
          if k <> j && maison.(k) = maison.(j) && a.(k).(j) = 1 then
            failwith "partage impossible"
        done;

        (* on met les ennemis de j dans l'autre maison *)
        for k=i+1 to n-1 do
          if k <> j && a.(k).(j) = 1 then
            maison.(k) <- -maison.(j)
        done;

        vu.(j) <- true;
      )
    done
  )
done;
maison
```

Il est possible de regrouper certains tests, au détriment de la lisibilité du programme.

L'exemple suivant peut clarifier le fonctionnement de l'algorithme. Dans la figure suivante, chaque point représente une personne, et deux personnes sont reliées par un trait si elles sont en conflit.



Le tableau ci-dessous donne, pour chaque personne, l'étape à laquelle elle a été placée, ainsi que la maison dans laquelle elle a été placée.

personne	maison 1	maison -1
1	1	
2	7	
3	9	
4		4
5		2
6	3	
7		2
8	6	
9	5	
10	5	
11		2
12		8
13		8

Question 3

Réflexivité : $a_i, i = 1$ pour tout i par convention.
Symétrie : s'il existe une suite de personnes de i à j telle que chaque personne est en conflit avec la suivante et la précédente, la même suite dans l'ordre inverse prouve que $a_{i,j} = 1$.
Transitivité : s'il existe une suite de personnes en conflit de i à j et une autre suite de j à k, la concaténation des deux suites donne une suite de personnes en conflit de i à k.

On a donc une relation d'équivalence. Les classes d'équivalence correspondent à ce que nous avons appelé les « composantes connexes ».

Clairement, s'il existe une suite de i_1 à lui même telle que décrite dans l'énoncé et si i_1 est dans la maison 1, alors i_2 doit être placé dans la maison -1, i_3 dans la maison 1, ..., i_k dans la maison $(-1)^{k-1}$, donc i_{2l+1} dans la maison 1 et $i_1 = i_{2l+2}$ dans la maison -1 : contradiction.

Inversement, si la fonction de la question 2 aboutit à la réponse « partage impossible », c'est parce qu'une personne k dans la composante connexe de i se trouve contrainte d'être placée dans les deux maisons à la fois. Il existe donc deux suites de personnes en conflit allant de i à k, l'une de longueur paire et l'autre de longueur impaire. La concaténation de la première suite avec la deuxième suite en ordre inverse donne une suite de i à i de longueur impaire.

Les différentes classes d'équivalence étant indépendantes les unes des autres, le nombre total de partages sans conflits est le produit du nombre de partages sans conflit pour chaque classe. Ce nombre, pour la classe d'équivalence de i est soit 0 s'il existe une suite impaire de i à i, soit 2 sinon : en effet, le premier élément de la classe d'équivalence peut être placé au choix dans la maison 1 ou -1, et toutes les autres affectations sont ensuite déterminées par la parité de la longueur de la suite reliant une personne à i. Le produit est donc 0 ou 2^N.

Question 4

La première idée consiste à mettre la personne 1 dans la maison 1, son premier ennemi dans la maison 2, et ses ennemis successifs, soit dans la maison 2 s'ils ne sont pas en conflit avec quelqu'un se trouvant déjà dans la maison 2, soit dans la maison 3 sinon. Cela conduirait à une généralisation facile de l'algorithme de la question 2, mais malheureusement cet algorithme ne marche pas : il existe des conflits pour lesquels il ne trouve pas de solution alors qu'il en existe une. Par exemple :

$a_{1,2} = a_{1,3} = 1$;
$a_{2,1} = a_{2,3} = a_{2,4} = a_{2,5} = 1$,
$a_{3,1} = a_{3,2} = a_{3,5} = 1$,
$a_{4,2} = a_{4,5} = 1$,
$a_{5,2} = a_{5,3} = a_{5,4} = 1$.
(Les entrées non définies sont 0). Un algorithme similaire à celui de la question 2 ferait les affectations suivantes :
`maison.(0) <- 1`,
`maison.(1) <- 2`,
`maison.(2) <- 3`,
`maison.(3) <- 1`,
`maison.(4)` « impossible »,
alors que les affectations de 1, 2, 3, 4, 5 à 1, 2, 3, 2, 1 satisfont les contraintes.

De fait, il n'existe pas de modification astucieuse de l'algorithme qui le rende correct : ce problème, connu sous le nom de coloriage d'un graphe par trois couleurs, est bien connu en informatique théorique pour être difficile. Il n'existe actuellement pas d'algorithme efficace pour le résoudre, c'est à dire qui fasse beaucoup mieux que regarder les 3^n affectations possibles des personnes dans les maisons.

Plus généralement, le problème de coloriage d'un graphe avec le nombre minimum de couleurs, de sorte que deux sommets liés par une arête soient toujours de couleurs différentes, est le problème du nombre chromatique. Un problème célèbre, resté longtemps conjecture, est de démontrer que tout graphe planaire (représentable sur un plan sans que ses arêtes se coupent) a un nombre chromatique au plus égal à 4, ou encore que toute carte planaire peut être coloriée avec au plus 4 couleurs de façon que deux pays ayant une frontière commune soient toujours de couleurs différentes : ceci a finalement été démontré en réduisant l'ensemble de tous les graphes planaires possibles à un nombre fini mais élevé de cas de graphes (environ 1900), qui furent ensuite examinés un par un par ordinateur.

⋆⋆⋆⋆⋆⋆⋆