TD1 : Parcours dans un graphe orienté

Adaptation de la partie II du sujet Centrale-Supélec MP – Option informatique – 2017

On appelle *graphe orienté* tout couple

$$G = (S, A),$$

où S est un ensemble fini dont les éléments sont appelés *sommets*, et

$$A \subseteq S \times S$$

est un ensemble dont les éléments sont appelés *arcs*.

Pour un arc $(s, s') \in A$, le sommet s est appelé son *origine* et s' son *extrémité*.

On considère par exemple le graphe

$$S_0 = \{0, 1, 2, 3, 4, 5\}$$

et

$$A_0 = \{(0,0), (0,3), (0,2), (0,1), (1,1), (1,2),\\ (2,1), (2,3), (2,4), (3,2), (4,1), (4,5), (5,1)\}.$$

Le graphe $G_0 = (S_0, A_0)$ est représenté figure 1.

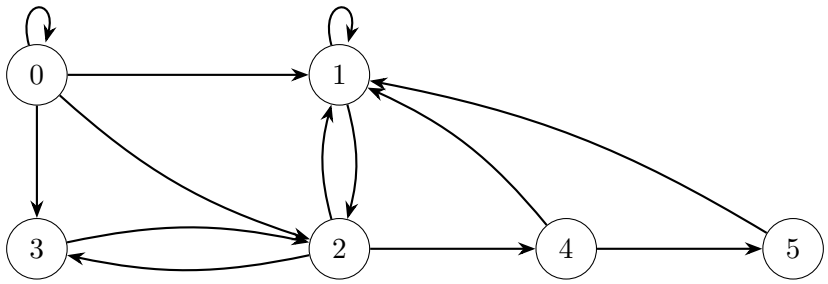

FIGURE 1 – Le graphe orienté G_0

Soient s et s' deux sommets d'un graphe orienté $G = (S, A)$.

On appelle *chemin de s vers s' de longueur ℓ* toute suite de sommets

$$s_0, s_1, \ldots, s_\ell$$

telle que

$$s_0 = s, \qquad s_\ell = s'$$

et

$$\forall i \in [\![0, \ell - 1]\!], \qquad (s_i, s_{i+1}) \in A.$$

La longueur du chemin est donc le nombre d'arcs qui le constituent.

On dit que s' est *accessible depuis s* lorsqu'il existe un chemin de s vers s'.

En particulier, tout sommet $s \in S$ est accessible depuis lui-même par un chemin de longueur nulle.

Dans les programmes à écrire, l'ensemble des sommets d'un graphe sera toujours un intervalle d'entiers $S = [\![0, n-1]\!]$.

Les sommets seront représentés par le type OCaml :

```
type sommet = int
```

Le graphe sera représenté par un tableau de listes d'adjacence `v` de type :

```
sommet list array
```

Pour tout sommet $s \in S$, la liste `v.(s)` contient, dans un ordre quelconque, tous les sommets s' tels que

$$(s, s') \in A.$$

Ainsi, le graphe G_0 précédent pourra être représenté en OCaml par :

```
let v0 = [|
  [0; 3; 2; 1];
  [1; 2];
  [1; 3; 4];
  [2];
  [1; 5];
  [1]
|]
```

A – Implantation d'une file

On veut implanter une file d'attente à l'aide d'un tableau circulaire. On définit pour cela le type :

```
type 'a file = {
  tab : 'a array;
  mutable deb : int;
  mutable fin : int;
  mutable vide : bool
}
```

Le champ `deb` indique l'indice du premier élément de la file et `fin` l'indice qui suit celui du dernier élément. Le booléen `vide` indique si la file est vide.

Les éléments de la file sont rangés à partir de la case `deb` jusqu'à la case précédant `fin`, en repartant à la case 0 lorsqu'on atteint la fin du tableau.

Il est donc possible d'avoir

$$\texttt{fin} < \texttt{deb}.$$

Par exemple, la situation de la figure 2 correspond à une file contenant, dans cet ordre,

$$4,\ 0,\ 1,\ 12,\ 8,$$

avec `deb = 9` et `fin = 2`.

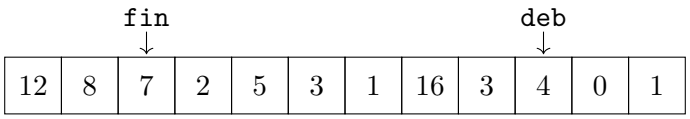

FIGURE 2 – Exemple de file circulaire avec `fin` < `deb`

On rappelle qu'un champ `mutable` peut voir sa valeur modifiée. Par exemple,

```
f.deb <- 0
```

affecte la valeur 0 au champ `deb` de la file `f`.

1) Écrire une fonction

```
ajoute : 'a file -> 'a -> unit
```

telle que `ajoute f x` ajoute `x` à la fin de la file d'attente `f`.

Si l'opération est impossible parce que la file est pleine, la fonction devra produire un message d'erreur, par exemple à l'aide de

```
failwith "File pleine"
```

2) Écrire une fonction

```
retire : 'a file -> 'a
```

telle que `retire f` retire l'élément placé en tête de la file `f` et le renvoie.

Si la file est vide, la fonction devra produire un message d'erreur.

3) Quelle est la complexité temporelle des fonctions `ajoute` et `retire` ?

B – Parcours du graphe et calcul de distance

On considère désormais l'algorithme 1 appliqué à un graphe orienté $G = (S, A)$ et à un ensemble $E \subseteq S$ de sommets de départ.

On note $n = |S|$.

Le tableau D est destiné à contenir des distances. La valeur particulière ∞ indique qu'aucune distance n'a encore été attribuée au sommet considéré.

Le tableau P contient des prédécesseurs. On utilise deux valeurs particulières :

— *vide*, lorsqu'aucun prédécesseur n'a encore été déterminé ;

— *rien*, pour les sommets appartenant à l'ensemble initial E.

Créer une file d'attente F, vide au départ.
Créer un tableau D, indexé par S, dont toutes les cases sont initialisées à ∞.
Créer un tableau P, indexé par S, dont toutes les cases sont initialisées à *vide*.
Créer une variable c initialisée à n.

pour tout $s \in E$ **faire**
    insérer s à la fin de F ;
    fixer $D[s]$ à 0 ;
    fixer $P[s]$ à *rien* ;
    diminuer c de 1.
fin pour

tant que F n'est pas vide **faire**
    extraire le sommet s situé en tête de F ;
    pour tout s' tel que $(s, s') \in A$ et $D[s'] = \infty$ **faire**
        fixer $D[s']$ à $D[s] + 1$;
        fixer $P[s']$ à s ;
        insérer s' à la fin de F ;
        diminuer c de 1.
    fin pour
fin tant que

Renvoyer (c, D, P).

Algorithme 1

1 Justifier que l'algorithme 1 termine toujours.

2 Déterminer la complexité temporelle de l'algorithme 1 en fonction de $|S|$ et $|A|$.

On supposera que les opérations d'insertion et d'extraction dans la file s'effectuent en temps constant.

3 Justifier qu'au début de chaque passage dans la boucle

« tant que F n'est pas vide »,

si F contient, dans cet ordre, les sommets

$$s_1, s_2, \ldots, s_r,$$

alors

$$D[s_1] \leq D[s_2] \leq \cdots \leq D[s_r]$$

et

$$D[s_r] - D[s_1] \leq 1.$$

Pour tout sommet s de G, on note d_s la *distance de E à s*, c'est-à-dire la longueur d'un plus court chemin partant d'un sommet de E et arrivant en s.

Plus précisément,

$$d_s = \min \left\{ \ell \in \mathbb{N} \;\middle|\; \begin{array}{l} \text{il existe } s_0 \in E \text{ et un chemin} \\ s_0, s_1, \ldots, s_\ell = s \end{array} \right\}.$$

On adopte la convention

$$d_s = \infty$$

s'il n'existe aucun chemin d'un sommet de E vers s.

4 Justifier qu'à la fin de l'algorithme, pour tout sommet $s \in S$,

$$D[s] \neq \infty$$

si et seulement si s est accessible depuis au moins un sommet de E.

Montrer également que, pour tout sommet s,

$$d_s \leq D[s].$$

Que représente alors la variable c à la fin de l'algorithme ?

5 Montrer qu'en réalité, à la fin de l'algorithme,

$$\boxed{D[s] = d_s}$$

pour tout sommet $s \in S$.

On pourra raisonner par l'absurde en considérant un sommet s tel que

$$D[s] > d_s$$

et un plus court chemin

$$s_0, s_1, \ldots, s_{d_s} = s$$

avec $s_0 \in E$.

Lorsque $s \notin E$ est accessible depuis E, que représente alors $P[s]$?

C – Implantation du parcours

Pour l'implantation en OCaml, on décide de coder les valeurs particulières de la façon suivante :

$$\infty \longleftrightarrow -1.$$

Dans le tableau des prédécesseurs P, on utilisera :

$$\textit{vide} \longleftrightarrow -2, \qquad \textit{rien} \longleftrightarrow -1.$$

Ainsi :

— $P[s] = -2$ signifie que le sommet s n'a jamais été découvert ;

— $P[s] = -1$ signifie que s appartient à l'ensemble initial E ;

— $P[s] \geq 0$ désigne le prédécesseur de s choisi par le parcours.

1 Écrire une fonction

```
accessibles : sommet list array -> sommet list -> int * int array * sommet array
```

qui reçoit :

— un graphe orienté sous la forme de son tableau de listes d'adjacence ;

— un ensemble E de sommets, représenté par une liste ;

et renvoie le triplet (c, D, P) calculé par l'algorithme 1.

On pourra supposer que les sommets présents dans la liste représentant E sont deux à deux distincts et appartiennent tous au graphe.

2 Écrire une fonction

```
chemin : sommet -> sommet array -> sommet list
```

qui reçoit un sommet s et le tableau P obtenu par la fonction `accessible`, et renvoie la liste des sommets constituant un chemin de longueur minimale d'un sommet de E jusqu'à s.

Si aucun sommet de E ne permet d'atteindre s, la fonction devra produire un message d'erreur, par exemple :

```
failwith "Sommet inaccessible"
```
