Corrigé du TD1

Parcours dans un graphe orienté

A – Implantation d'une file

On note n la taille du tableau utilisé pour représenter la file.
La file est vide lorsque `vide = true`. Lorsqu'elle n'est pas vide, la situation

`deb = fin`

signifie donc que le tableau est entièrement rempli.

1)

Pour ajouter un élément, on vérifie tout d'abord que la file n'est pas pleine. On écrit ensuite l'élément à la position `fin`, puis on décale `fin` circulairement.

```
let ajoute f x =
  let n = Array.length f.tab in
  if f.vide then begin
     f.vide <- false ;
     f.fin <- (1+f.deb) mod n;
     f.tab.(f.deb) <- x
  end
  if (not f.vide) && f.deb = f.fin then
    failwith "File pleine"
  else begin
    f.tab.(f.fin) <- x;
    f.fin <- (f.fin + 1) mod n;
  end
```

2)

Si la file n'est pas vide, l'élément à retirer se trouve dans la case d'indice `deb`. Après son extraction, on décale `deb` circulairement.

La file devient vide exactement lorsque, après cette modification,

`deb = fin`.

```
let retire f =
  if f.vide then
    failwith "File vide"
  else begin
    let x = f.tab.(f.deb) in
    let n = Array.length f.tab in
    f.deb <- (f.deb + 1) mod n;
    if f.deb = f.fin then
      f.vide <- true;
    x
  end
```

3)

Les fonctions `ajoute` et `retire` n'effectuent qu'un nombre constant d'opérations élémentaires : accès à un tableau, affectations, comparaisons et opérations arithmétiques.

Leur complexité temporelle est donc

$\boxed{\Theta(1)}$.

B – Parcours du graphe et calcul de distance

1)

Montrons que l'algorithme termine.

Un sommet s' est ajouté à la file dans la boucle principale uniquement lorsque

$$D[s'] = \infty.$$

Immédiatement après son insertion, on affecte à $D[s']$ une valeur finie. Ainsi, un même sommet ne peut être ajouté à la file qu'une seule fois.

Les sommets de E ne sont eux aussi placés dans la file qu'une seule fois.

Comme le graphe possède un nombre fini de sommets, au plus

$$|S|$$

insertions sont donc effectuées dans la file.

À chaque passage dans la boucle principale, un sommet est retiré de la file. Comme seuls un nombre fini de sommets peuvent être insérés, la file finit nécessairement par devenir vide.

Ainsi,

$$\boxed{\text{l'algorithme termine toujours.}}$$

2)

L'initialisation des tableaux D et P nécessite

$$\Theta(|S|)$$

opérations.

Chaque sommet est inséré dans la file au plus une fois et en est donc extrait au plus une fois.

Lorsqu'un sommet s est extrait de la file, on parcourt l'ensemble de ses voisins sortants. La liste d'adjacence de chaque sommet étant donc parcourue au plus une fois, le nombre total d'arcs examinés est

$$\sum_{s \in S} \deg^+(s) = |A|.$$

Les opérations d'insertion et d'extraction dans la file étant supposées de coût constant, la complexité totale est donc

$$\boxed{\Theta(|S| + |A|)}.$$

3)

Montrons l'invariant suivant :

au début de chaque passage dans la boucle principale, si la file contient

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

dans cet ordre, alors

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

et

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

Initialisation.

Après l'insertion des sommets de E, tous les sommets présents dans la file vérifient $D[s] = 0$. La propriété est donc vraie.

Hérédité.

Supposons la propriété vraie au début d'une itération et notons $s = s_1$ le sommet extrait de la tête de la file.

Tous les nouveaux sommets s' découverts au cours de cette itération reçoivent la valeur

$$D[s'] = D[s] + 1.$$

D'après l'hypothèse de récurrence, tous les sommets qui restent dans la file après le retrait de s ont une distance égale à

$$D[s] \qquad \text{ou} \qquad D[s] + 1.$$

Les nouveaux sommets, qui sont ajoutés à la fin de la file, ont précisément la valeur

$$D[s] + 1.$$

Par conséquent, après les insertions, les valeurs de D restent rangées dans l'ordre croissant dans la file et ne peuvent prendre que deux valeurs consécutives.

L'invariant est donc conservé.

Ainsi, au début de chaque passage dans la boucle,

$$\boxed{D[s_1] \leq D[s_2] \leq \cdots \leq D[s_r]}$$

et

$$\boxed{D[s_r] - D[s_1] \leq 1.}$$

4)

Montrons tout d'abord que

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

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

Sens direct.

On peut montrer l'invariant suivant dans l'algorithme : si $D[u]$ est fini alors u est accessible depuis E. En effet, au premier tour les éléments convernés sont dans E et si on fixe une valeur finie de $D[u]$ pour un certain sommet u c'est qu'il est le voisin d'un sommet qui a déjà une valeur finie et qui est donc accessible depuis E par invariant d'où u est lui même accessible depuis E.

Ainsi,

$$D[s] \neq \infty \quad \Longrightarrow \quad s \text{ est accessible depuis } E.$$

Sens réciproque.

Supposons s accessible depuis E. Il existe donc un chemin

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

avec

$$s_0 \in E.$$

Le sommet s_0 est placé initialement dans la file.

Lorsqu'un sommet s_i est extrait de la file, l'algorithme examine l'arc

$$(s_i, s_{i+1}).$$

Si s_{i+1} n'avait pas encore été découvert, il l'est alors et reçoit une valeur finie.

Par récurrence sur i, tous les sommets du chemin sont donc découverts, en particulier s.

Ainsi,

$$s \text{ accessible depuis } E \quad \Longrightarrow \quad D[s] \neq \infty.$$

On a donc

$$\boxed{D[s] \neq \infty \iff s \text{ est accessible depuis } E.}$$

Montrons maintenant que

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

On remarque que pour toute arête (s, s'), on a $d_{s'} \leq 1 + d_s$. Ceci permet de garantir l'invariant de l'algorithme suivant : $d_x \leq D[x]$ pour tout sommet x. Cet invariant est bien vérifié par l'initialisation et lorsqu'on affecte $D[s'] = 1 + D[s] \geq 1 + d_s$ par invariance et cette dernière valeur est supérieure ou égale à $d_{s'}$ d'après la remarque précédente.

Enfin, la variable c est initialisée à $|S|$ puis diminuée de 1 exactement une fois lors de la découverte de chaque sommet.

À la fin de l'algorithme,

$$\boxed{c = \text{nombre de sommets non accessibles depuis } E.}$$

5)

Nous savons déjà, grâce à la question précédente, que pour tout sommet s,

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

Il reste à montrer l'inégalité réciproque.

Supposons par l'absurde qu'il existe un sommet s tel que $D[s] > d_s$. et fixons un tel sommet qui minimise la valeur de d_s parmi les candidats.

Considérons un plus court chemin

$$s_0, s_1, \ldots, s_k = s$$

avec

$$s_0 \in E.$$

Tout préfixe de ce chemin est lui-même un plus court chemin depuis E. En effet, si l'on pouvait atteindre un sommet s_i en moins de i arcs, on pourrait remplacer le début du chemin par ce chemin plus court et obtenir un chemin vers s de longueur strictement inférieure à d_s, ce qui est impossible.

On a donc, $d_{s_{k-1}} = D[s_{k-1}]$.

Lorsque le sommet s_{k-1} est extrait de la file, l'algorithme examine l'arc

$$(s_{k-1}, s_k = s).$$

Deux cas sont possibles.

— Si s n'a pas encore été découvert, l'algorithme lui attribue $D[s] = D[s_{k-1}] + 1 = d_s$, ce qui est exclu.

— Si s a déjà été découvert, la question 3 garantit qu'un sommet découvert antérieurement ne peut avoir une valeur strictement supérieure à celle que l'on attribuerait à un sommet nouvellement découvert depuis s_{k-1}. On a donc $D[s] \leq D[s_{k-1}] + 1 = d_s$ ce qui est également exclu.

Lorsque $s \notin E$ est accessible depuis E, $P[s]$ est un prédécesseur de s sur un plus court chemin issu de E.

Plus précisément,

$$\boxed{(P[s], s) \in A \qquad \text{et} \qquad d_{P[s]} = d_s - 1.}$$

C – Implantation du parcours

1)

On utilise la structure de file définie dans la partie A.
Le tableau D est initialisé avec la valeur -1, qui représente ∞.
Le tableau P est initialisé avec la valeur -2, qui représente *vide*.

```
let accessibles v e =
  let n = Array.length v in

  let d = Array.make n (-1) in
  let p = Array.make n (-2) in

  let f = {
    tab = Array.make n 0;
    deb = 0;
    fin = 0;
    vide = true
  } in

  let c = ref n in

  List.iter
    (fun s ->
      ajoute f s;
      d.(s) <- 0;
      p.(s) <- -1;
      decr c)
    e;

  while not f.vide do
    let s = retire f in

    List.iter
      (fun s' ->
        if d.(s') = -1 then begin
          d.(s') <- d.(s) + 1;
          p.(s') <- s;
          ajoute f s';
          decr c
        end)
      v.(s)
  done;

  (!c, d, p)
```

La taille n choisie pour la file est suffisante : un sommet n'est présent dans la file qu'une seule fois et il y a n sommets au total.

La complexité de cette fonction est

$$\boxed{\Theta(|S| + |A|)}.$$

2)

Le tableau P permet de remonter depuis s jusqu'à un sommet de E.
Trois situations sont possibles :

— $P[s] = -2$: le sommet est inaccessible depuis E ;

— $P[s] = -1$: le sommet s appartient à E ;

— $P[s] \geq 0$: $P[s]$ est le prédécesseur de s sur un plus court chemin.

On peut construire le chemin à l'aide d'une fonction auxiliaire récursive.

```
let chemin s p =
  let rec aux u acc =
    if p.(u) = -2 then
      failwith "Sommet inaccessible"
    else if p.(u) = -1 then
      u :: acc
    else
      aux p.(u) (u :: acc)
  in
  aux s []
```