Feuille de révisions n°6 - Programmation en OCaml : manipulation de graphes

Dans toute cette feuille de révision on suppose définis les types OCaml suivants, permettant respectivement la représentation de graphes (orientés ou non) par table de listes d'adjacences, par matrice d'adjacence.

```
type graphe_adj =  (* graphes par table de listes d'adjacence *)
  int list array

type graphe_mat =  (* graphes par matrice d'adjacence *)
  bool array array
```

1 Pied à l'étrier

R. 6-1 Définir une fonction OCaml prenant en arguments un graphe `g` (représenté par matrice d'adjacence), deux sommets entiers `i` et `j` et retournant si l'arête $\{\texttt{i}, \texttt{j}\}$ est une arête du graphe `g`.

R. 6-2 Définir une fonction OCaml prenant en arguments un graphe `g` (représenté par table de listes d'adjacence), deux sommets entiers `i` et `j` et retournant si l'arête $\{\texttt{i}, \texttt{j}\}$ est une arête du graphe `g`.

R. 6-3 Définir une fonction OCaml prenant en arguments deux graphes (représentés par matrice d'adjacence) et testant s'ils sont égaux.

R. 6-4 Définir une fonction OCaml prenant en arguments deux graphes (représentés par table de listes d'adjacence)et testant s'ils sont égaux.

R. 6-5 Définir une fonction OCaml prenant en arguments un entier `n` et une liste `l` de couples d'entiers et retournant le graphe orienté (représenté par table de listes d'adjacence) dont les sommets sont les entiers de l'intervalle $[\![0, \texttt{n}-1]\!]$ et les arcs sont les éléments de `l`. On peut supposer que `l` est sans doublons. On déclenchera une erreur si `l` contient un couple qui ne peut être un arc possible de ce graphe.

R. 6-6 Définir une fonction OCaml prenant en arguments un entier `n` et une liste `l` de couples d'entiers et retournant le graphe non orienté (représenté par table de listes d'adjacence) dont les sommets sont les entiers de l'intervalle $[\![0, \texttt{n}-1]\!]$ et les arêtes sont données par les couples de `l`. On peut supposer que les couples de `l` représentent deux à deux des arêtes différentes. On déclenchera une erreur si `l` contient un couple qui ne correspond pas à une arête possible de ce graphe.

R. 6-7 Définir une fonction OCaml prenant en arguments un entier `n` et une liste `l` de couples d'entiers et retournant le graphe orienté (représenté par matrice d'adjacence) dont les sommets sont les entiers de l'intervalle $[\![0, \texttt{n}-1]\!]$ et les arcs sont données par les éléments de `l`.

R. 6-8 Définir une fonction OCaml prenant en argument un graphe orienté (représenté par table de listes d'adjacence) et retournant la liste de ses arcs.

R. 6-9 Définir une fonction OCaml prenant en argument un graphe non orienté (représenté par table de listes d'adjacence) et retournant la liste de ses arêtes représentées par des couples $(\texttt{i}, \texttt{j})$ avec $\texttt{i} < \texttt{j}$.

R. 6-10 Définir une fonction OCaml prenant en argument un graphe orienté (représenté par matrice d'adjacence) et retournant la liste de ses arcs.

R. 6-11 Définir une fonction OCaml prenant en argument un graphe orienté g (représenté par table de listes d'adjacence) et retournant le graphe non orienté (représenté par table de listes d'adjacence) ayant même ensemble de sommets et contenant une arrête $\{x, y\}$ si et seulement si le graphe g contient l'arc (x, y) ou l'arc (y, x).

R. 6-12 La matrice d'accessibilité d'un graphe g non orienté à n sommets est une matrice de booléens de dimension $n \times n$ contenant `true` dans la case d'indice (i, j) si et seulement il existe un chemin du sommet i au sommet j dans le graphe. Définir une fonction prenant en arguments un graphe (représenté par matrice d'adjacence) et retournant sa matrice d'accessibilité. On s'autorise une complexité en $\mathcal{O}(n^4)$.

R. 6-13 La matrice d'accessibilité d'un graphe g non orienté à n sommets est une matrice de booléens de dimension $n \times n$ contenant `true` dans la case d'indice (i, j) si et seulement s'il existe un chemin du sommet i au sommet j dans le graphe. Définir une fonction prenant en arguments un graphe (représenté par matrice d'adjacence) et retournant sa matrice d'accessibilité. On s'autorise une complexité en $\mathcal{O}(n^3)$, on pourra pour cela s'aider de l'algorithme de Roy-Warshall.

2 Parcours de graphes

Dans cette partie les graphes seront tous représentés par table de listes d'adjacence (*i.e.* avec le type `graphe_adj`).

R. 6-14 Définir une fonction OCaml calculant un parcours en profondeur (une permutation des sommets, de type `int list`) du graphe passé en argument. On s'efforcera d'utiliser la pile des appels récursifs, pour stocker les éléments encore à traiter. La complexité de l'implémentation devra être en $\mathcal{O}(n + m)$ où n est le nombre de sommets du graphe et m est le nombre d'arêtes.

R. 6-15 Définir une fonction OCaml calculant un parcours en profondeur (une permutation des sommets, de type `int list`) du graphe passé en argument. On s'efforcera d'utiliser le module `Stack`, pour stocker les éléments encore à traiter. La complexité de l'implémentation devra être en $\mathcal{O}(n+m)$ où n est le nombre de sommets du graphe et m est le nombre d'arêtes.

R. 6-16 Définir une fonction OCaml calculant un parcours en largeur (une permutation des sommets, de type `int list`) du graphe passé en argument. On s'efforcera d'utiliser le module `Queue`, pour stocker les éléments encore à traiter. La complexité de l'implémentation devra être en $\mathcal{O}(n+m)$ où n est le nombre de sommets du graphe et m est le nombre d'arêtes.

3 Application des parcours

Dans cette partie les graphes seront représentés par table de listes d'adjacence (ou table de liste de successeurs pour les graphes orientés), *i.e.* par le type OCaml `graphe_adj` défini plus haut.

R. 6-17 Définir une fonction OCaml prenant en argument un graphe non orienté et deux sommets de ce graphe, et testant si l'un est accessible depuis l'autre.

R. 6-18 Définir une fonction OCaml prenant en argument un graphe non orienté et testant si celui-ci est connexe.

R. 6-19 Définir une fonction OCaml prenant en argument un graphe non orienté et testant si celui-ci est un arbre.

R. 6-20 Définir une fonction OCaml prenant en argument un graphe non orienté et retournant sa décomposition en composantes connexes sous la forme d'un tableau associant à chaque sommet un numéro de composante.

R. 6-21 Définir une fonction OCaml prenant en argument un graphe non orienté et retournant si ce graphe est biparti.

R. 6-22 Définir une fonction OCaml prenant en arguments un graphe non orienté , deux sommets u et v et retournant la distance, en nombre d'arêtes séparant u et v.

R. 6-23 Définir une fonction OCaml prenant en argument un graphe orienté supposé sans circuit et retournant, **à l'aide d'un parcours** un tri topologique sous la forme d'une liste de sommets.

R. 6-24 Définir une fonction OCaml prenant en argument un graphe orienté et retournant un tri préfixe des sommets sous la forme d'une liste de sommets.

R. 6-25 Définir une fonction OCaml prenant en argument un graphe orienté et retournant un booléen indiquant s'il admet un circuit.