EXERCICES SUR LES GRAPHES

Dans tous les exercices, on représente les graphes par des tableaux de listes d'adjacence. Les sommets sont donc numérotés de 0 à n.

Sur le site du lycée (rubrique MP - informatique), vous trouverez un fichier contenant des exemples de graphes orientés et non orientés pour tester vos fonctions.

---

1) En cours, nous avons écrit une fonction itérative qui calcule un parcours dans un graphe. On peut l'écrire récursivement avec des objets non mutables, au prix d'une complexité un peu plus importante, en modifiant légèrement l'algorithme.

Classiquement, on ajoute dans l'ensemble ouvert des sommets qui ne sont ni dans l'ensemble ouvert, ni dans l'ensemble fermé. On peut éviter de tester ces appartenances au moment de l'ajout, en ajoutant les sommets grossièrement, quitte à ce qu'ils soient présents plusieurs fois dans l'ensemble ouvert ou présents déjà dans l'ensemble fermé.

Cependant, pour éviter de passer deux fois par le même sommet, au moment où en retire un, on vérifie seulement son appartenance à l'ensemble fermé :

- s'il est déjà dans l'ensemble fermé, alors ça signifie qu'on est déjà passé par ce sommet, donc on le laisse tomber et on passe au sommet suivant ;
- s'il est présent plusieurs fois dans l'ensemble ouvert, ce n'est pas grave : la première fois, il passe dans l'ensemble fermé, donc les apparitions suivantes seront ignorées d'après ce qui précède.

En utilisant le type liste pour représenter les ensembles, écrivez deux versions des parcours de graphes, l'une en profondeur, l'autre en largeur. Elles seront récursives encapsulées et elles calculeront l'ensemble fermé, ensemble des points accessibles depuis le sommet initial :

`let rec pprf graphe ouvert ferme = ...` et `let rec plar graphe ouvert ferme = ...`

Objectif : 8 lignes maximum.

2) Utilisez ces fonctions en les modifiant si nécessaire pour déterminer les composantes connexes d'un graphe **non orienté** : on veut calculer la liste des ensembles fermés successifs. Pour cela, on choisit un point au hasard dans le graphe, on calcule sa composante connexe, c'est-à-dire les sommets qui lui sont accessibles, on les retire du graphe et on recommence avec un autre sommet jusqu'à épuisement du graphe. Tout ça peut se faire récursivement. Il faut juste un moyen pour supprimer à peu de frais un sommet et ses arêtes dans un graphe non orienté.

3) L'avantage de versions récursives des parcours est qu'on peut alors construire en même temps l'arbre couvrant associé.

Les arbres n-aires ont pour type

```
type arbre = S of int * arbre list;;
```

Ce sont donc des arbres homogènes dont les étiquettes sont des entiers.

Si s est un sommet et ℓ est la liste de ses successeurs, alors on peut construire récursivement l'arbre couvrant associé au parcours issu de s : on construit récursivement les arbres issus des successeurs de s s'ils n'ont pas déjà été construits (on obtient une forêt), puis connaissant ces arbres, on peut les définir comme les fils de l'arbre issu de s.

On abandonne donc l'ensemble ouvert et on ne garde que l'ensemble fermé (indispensable pour éviter de repasser deux fois par le même sommet). Plutôt que verser tous les sommets dans un ensemble ouvert, ce qui fait perdre de vue leurs prédécesseurs, on épuise les successeurs d'un sommet avant de passer aux successeurs d'un autre : de cette façon, on garde la trace des relations père-fils.

Écrivez des fonctions qui calculent les arbres couvrants associés aux parcours en profondeur ou en largeur à partir d'un sommet. Cette question nécessitera sans doute des fonctions auxiliaires.