TP noté

Option informatique, deuxième année

Julien Reichert

Ce TP est à faire en deux heures sans assistance. Il sera rendu sous la forme d'un fichier d'extension .ml contenant l'ensemble des programmes écrits ainsi que les données nécessaires (exemples et déclarations de types) pour l'exécuter, ainsi que la réponse à l'exercice 6 et d'éventuels éléments jugés utiles, soit en commentaire dans le fichier .ml, soit sous la forme d'un document d'extension .txt (à la rigueur .pdf ou .odt, en évitant .doc), soit en tant que scan d'une copie manuelle.

Contexte

Les réformes territoriales incitant à la fusion des petites communes, une petite zone géographique décide de rassembler progressivement ses villages en un seul. Le principe de la fusion est que le nom de la nouvelle commune soit celui de celle des deux qui avait le plus d'habitants (au hasard dans le cas - peu probable - d'une égalité). Ainsi, on pourrait s'attendre qu'à la fin du processus le village le plus peuplé soit celui dont le nom demeure, mais finalement tout est une question d'ordre...

Préliminaires

L'exemple de travail est disponible sous la forme d'un fichier d'extension .ml dont le lien est fourni à côté de celui du présent énoncé.

La structure de données utilisée pour représenter un village est le type enregistrement suivant : [1]

```
type village = { nom : string; mutable population : int };;
```

La zone géographique est un graphe non orienté dont les sommets sont les villages et les arêtes relient deux villages tels qu'on puisse passer de l'un à l'autre par la route sans qu'il n'y ait d'autre village sur le chemin.

Les fusions ne peuvent concerner que deux villages voisins et consistent donc à rassembler deux sommets en un, les arêtes incidentes au nouveau sommet le reliant alors à tout sommet qui était relié à au moins un des deux anciens sommets. En ce qui concerne l'objet de type `village`, il aura pour attribut `population` la somme des deux anciennes valeurs.

La structure de données choisie est la suivante :

```
type comcom = { mutable communes : village list;  mutable routes : (string * string) list };;
```

Dans la mesure où le graphe est non orienté, le premier élément de chaque couple décrivant une arête sera celui de nom minimal selon l'ordre lexicographique (rappel : le symbole `<` fonctionne en Caml pour les chaînes de caractères).

Guide de survie pour l'utilisation de fonctions pseudo-aléatoires : `Random.int` (utilisable sans importer le module) retourne un nombre aléatoire entre `0` et son argument (exclu). On n'utilisera pas d'autre fonction du module.

1. Il sera constaté que la population est une rubrique mutable, donnant lieu à une variante en fin d'énoncé.

Fusions aléatoires

Exercice 1 : Écrire en Caml une fonction `fusion v1 v2` qui retourne le village obtenu par la fusion de `v1` et `v2`.

Exercice 2 : Écrire en Caml une fonction `random_list l` qui retourne un élément de la liste `l` au hasard avec équiprobabilité (en admettant que la distribution de `Random.int` soit uniforme, ce qui est presque vrai).

Exercice 3 : Écrire en Caml une fonction `maj cc v1 v2` qui retourne la nouvelle version du graphe `cc` mis à jour par la fusion des villages `v1` et `v2`.

Exercice 4 : Écrire en Caml une fonction `fusions cc` qui retourne le nom de la commune obtenue une fois que tous les villages ont fusionné suivant le principe de l'énoncé et en faisant successivement les fusions de deux villages au hasard, c'est-à-dire en choisissant les villages selon une arête sélectionnée arbitrairement.

Exercice 5 : Appeler mille fois la fonction précédente et compter le nombre de fois que chaque nom est sorti. La structure de données est laissée au choix, mais il faudra y faire figurer tous les villages, pas seulement ceux dont le nom est sorti au moins une fois, ce qui incite à utiliser des tableaux, voire des tables de hachage.

Considérations théoriques

Exercice 6 : Donner une CNS pour qu'un village puisse être celui dont le nom est maintenu jusqu'à la fin.

Exercice 7 : Écrire en Caml une fonction `fusions_orientees cc v` qui détermine si le village `v` peut être celui dont le nom est maintenu jusqu'à la fin. La méthode est laissée libre, en particulier l'exercice précédent peut être ignoré. Il n'est pas non plus nécessaire de fournir l'ordre des fusions, seul un booléen suffit.

Exercice 8 : À l'aide de l'exercice précédent, écrire en Caml une fonction `possibles cc` qui donne la liste des villages au sein de `cc` dont le nom peut être maintenu jusqu'à la fin.

Démographie

Malheureusement, les villages se dépeuplent. Pour simplifier, nous considérons une baisse constante de quinze habitants par an dans chaque village d'origine.
Ceci pose un problème dans la mesure où il faut alors traiter différemment un village suivant le nombre de fusions. Une idée, assez classique en informatique théorique, est d'encoder cette information sans préjudice de correction des algorithmes, donc ici des comparaisons. Le principe est le suivant : la population qui sera enregistrée est une valeur virtuelle contenant cent fois la vraie valeur de la population plus le nombre de fusions (ce qui ne fausse aucune comparaison car le nombre de villages est inférieur à cent, le problème des retenues étant esquivé[2]).

Exercice 9 : Écrire en Caml une fonction `encode cc` qui transforme `cc` en multipliant par cent toutes les données de population. Il s'agira ici de l'état initial.

Exercice 10 : Écrire en Caml une fonction `annee cc` qui met à jour la population de tous les villages de `cc` en considérant qu'un an a passé. Si une population devait devenir négative, on peut déclencher une erreur ou supprimer le village (choix à éviter, c'est bien plus compliqué). L'état de `cc` peut être intermédiaire, avec des fusions déjà effectuées.

Exercice 11 : Reprendre les exercices 4 et 5 avec le contexte de cette section, en considérant une fusion par an.

2. Il y a eu un passage délicat de ma thèse dans lequel je devais gérer l'encodage de quatre valeurs dans un seul nombre et faire attention aux retenues - petit hommage !