PRFO23 2023-2024

Rapport : Cafe-Flow-Maximizer

Nom: **Lapu**
Prénom: **Matthias**

1 Introduction

Le but de ce projet est de maximiser le flot de café entre une source et un puit en utilisant l'algorithme de Dinitz.

2 Structure

Pour ce faire, nous avons premièrement implémenté un graphe pondéré afin de résoudre ce problème. Dans cette implémentation, le graphe est un dictionnaire de dictionnaire avec comme clé la pondération d'un nœud vers l'autre.

```

    module NodeMap = Map.Make(X)

    type graph = int NodeMap.t NodeMap.t

```

Par la suite, celui-ci a été remplacé par une forme permettant d'avoir le minimum et le maximum, afin d'implémenter plus facilement l'algorithme de Dinitz.

```

    module NodeMap = Map.Make(X)

    type graph = (int*int) NodeMap.t NodeMap.t

```

De nombreuses fonctions classiques pour une implémentation de graphe tels que :

- l'ajout ou la suppression d'un noeud ou d'une arête
- test d'existence ou d'appartenance
- récupération des successeurs
- fold sur les noeuds, sur les successeurs
- etc.

ont été crée.

3 Phase 1

Lors de la phase 1, il était demandé de trouver les chemins les plus courts allant de la source vers le puits. Pour ce faire, nous utilisons un ensemble de chemin contenant tous les chemins possibles empruntés lors du parcours du graphe.

```
        module SetOfPath = Set.Make(struct
        type t = (node * int * int) list
        let compare = compare
        end)
```

En effectuant un fold sur le graphe à partir d'un nœud de départ et d'arriver, on peut donc parcourir le graphe et stocké dans une liste le chemin emprunté grâce au dernier noeud lu (qui sera modifié à chaque appel). La condition d'arrêt est lorsque l'ensemble reste le même d'un appel à l'autre, ce qui implique que nous avons parcouru l'entièreté du graphe. C'est en faisant cela que nous énumérons tous les chemins possibles, puis filtrons pour avoir uniquement les chemins qui vont du nœud de départ au nœud d'arriver.

Enfin, il nous suffit d'appliquer un fold sur la taille des chemins pour prendre uniquement les chemins ayant la taille minimale.

4 Phase 2

Afin d'appliquer l'algorithme de Dinitz, nous sommes partis du principe qu'il n'était pas nécessaire de faire plusieurs graphes. Le graphe de niveau peut être représenté par un ensemble de chemin, ce qui a déjà été implémenter lors de la phase 1. Il a cependant été nécessaire de modifier cette fonction pour qu'elle n'ajoute pas les chemins possédant des arcs saturés.

Par la suite, il a été nécessaire de créer des fonctions afin de déterminer la valeur correspondant au "bottleneck" dans un chemin.

L'algorithme de Dinitz demande également d'ajouter la valeur de celui-ci à chaque chemin jusqu'à saturation, pour ce faire nous avons donc créé une fonction permettant d'ajouter jusqu'à saturation, afin de pouvoir l'appeler sur chaque chemin de l'ensemble. L'ensemble de chemin ne sera pas modifié, il permettra uniquement d'avoir accès aux nœuds afin de modifier le graphe directement.

À l'aide de la récursion, lorsque le "bottleneck" a été appliqué sur l'ensemble, on reconstruit l'ensemble des chemins possible avec le graphe modifié. Car cet ensemble ne comporte pas d'arête saturée, on peut donc appliquer un nouveau "bottleneck".

Enfin, lorsque l'ensemble des chemins est vide, c'est-à-dire qu'il n'est plus possible de partir du nœud de départ au nœud d'arrivé, on retourne le graphe.

5 Test

Un fichier de test relativement générique a été crée pour vérifier le fonctionnement des fonctions implémentés, spécialement lorsqu'il y a des boucles dans le graphe, que le chemin n'existe pas ou encore pour afficher et vérifier l'état du graphe à différents moments de l'exécution du programme.

6 Limite du programme

Les limites du programme se situent premièrement dans les tests. Bien que des efforts aient été faits pour rester aussi général que possible, il est possible que certaines situations limites aient été oubliés. Ensuite, la récursion importante liée à la création de l'ensemble contenant les chemins, doit certainement être problématique lorsque des graphes importants sont étudiés. De plus, le choix des listes pour chaque chemins parcourus dans le graphe est également un choix coûteux. En effet, le chemin est dans le sens inverse, il est donc nécessaire d'appliquer un List.rev pour avoir la liste dans le bon sens. Enfin , les listes ne sont pas le choix le plus optimal, et il doit exister un meilleur moyen pour stocker les chemins (Queue ...) .