École Nationale Supérieure de l'Informatique pour l'Industrie et l'Entreprise

PRFO 2023-2024

Rapport : Programmation fonctionnelle

GOELLER Karim

Rapport : Programmation fonctionnelle

GOELLER Karim

January 3, 2024

Contents

1 Introduction

Ce projet a pour finalité d'aider M. Forest à maximiser le flot de café entre le bar de l'ENSIIE et son bureau. Entre le bureau de M. Forest et le bar, il y a des salles et des halls, qui sont reliés entre eux par des couloirs de différentes tailles.

La maximisation se fera en deux phases distinctes :

- Phase 1 : on cherchera le chemin le plus rapide (le plus court) entre le bureau et le bar
- Phase 2 : On prendra en compte la capacité réelle des couloirs, on cherchera à maximiser le flot sortant du bar et arrivant au bureau.

2 Phase 1

Il nous faut implémenter différentes structures et fonctions pour réussir cette phase. Nous modéliserons l'école par un graphe orienté non pondéré pour la phase 1 (sans prendre en compte leur capacité pour l'instant). Les salles et les halls seront les sommets, reliés entre eux par des couloirs qui constitueront les arêtes. Pour trouver tous les chemins les plus courts, nous utiliserons un parcours en largeur. Dans cette fonction, nous nous servirons d'une file d'attente dont j'expliciterai la structure dans le paragraphe 2.2 .

2.1 Le type graphe

Pour cette partie, nous n'aurons pas besoin d'un graphe pondéré, mais cela sera nécessaire dans la phase 2. Cependant j'utiliserai la même structure de graphe pour les deux parties. Donc dans la phase 1, toutes les pondérations seront fixées à zéro et n'interviendront pas dans les étapes du processus.
Pour implémenter les graphes, nous utiliserons un dictionnaire (une Map) dont les clés seront les sommets du graphe et la valeur associée à chaque clé sera un autre dictionnaire. Dans ce second dictionnaire, les clés seront un sommet successeur et la valeur associée sera un doublet d'entiers, dont j'expliciterai l'utilité dans la phase 2. Pour l'instant, ils seront initialisés à (0, 0) et n'interviennent pas.
en Ocaml mes graphes seront de type : `type graph = ((int*int) NodeMap.t) NodeMap.t`

2.2 File d'attente

Un parcour en largeur nécessite de connaitre quel est le noeud courant et quel chemin on a parcouru pour arriver à ce noeud. Cette méthode garantit que les noeuds sont visités de manière systématique, en commençant par ceux qui sont les plus proches du point de départ et en progressant ensuite vers les noeuds plus éloignés. Pour implémenter cette file d'attente, nous utiliserons une FIFO (First In, First Out). Dans notre cas, elle consistera en une liste de doublets, où le premier élément est le noeud courant et le second représente le chemin

parcouru jusqu'à ce noeud. Grâce à cette structure FIFO, dès qu'un noeud est visité, tous ses voisins non visités sont ajoutés à la file, assurant ainsi une exploration uniforme et complète du graphe. On arrête de parcourir le graphe lorsque tous les sommets ont été visités ou que la destination a été trouvée.

2.3 Parcour en largeur

Dans le cadre de la première phase de notre projet, nous avons choisi d'utiliser un parcours en largeur (BFS = Breadth-First Search) pour identifier les plus courts chemins entre le bureau de M. Forest et le bar. Cet algorithme débute à partir du noeud source, ici le bureau, et s'étend progressivement en explorant systématiquement tous les noeuds adjacents. Chaque successeur non visité du noeud courant est ajouté à la file d'attente, avec une mise à jour du chemin parcouru pour y parvenir, assurant ainsi que les noeuds sont abordés selon l'ordre de leur découverte.

Lorsqu'un noeud atteint le point de destination, à savoir le bar, l'algorithme évalue le chemin utilisé. Si ce dernier s'avère être plus court que les précédents chemins identifiés, il est alors enregistré comme l'un des chemins les plus courts. L'algorithme continue son exploration jusqu'à avoir couvert tous les chemins possibles. Cette approche garantit une exploration complète et efficace.

3 Phase 2

Lors de la Phase 2, nous prenons en compte la capacité des différents couloirs, ce qui implique l'utilisation de pondérations. La pondération de chaque couloir est représentée par un doublet défini comme suit :

- Premier élément : il représente le flux actuellement en transit dans le couloir. Initialement, pour chaque couloir, cette valeur est fixée à 0 au démarrage de l'algorithme.
- Deuxième élément : il indique le flux maximal que le couloir peut accueillir. Cette valeur reste constante tout au long de l'exécution de l'algorithme.

Pour résoudre ce problème, nous utiliserons l'algorithme de Dinic, qui s'appuie sur la création de graphes intermédiaires tels que le graphe résiduel et le graphe de niveau qui seront détaillés par la suite. Ces structures permettent à l'algorithme de Dinic d'optimiser la recherche de chemins et de maximiser efficacement le flux à travers le réseau de couloir de l'ENSIIE.

3.1 Graphe Résiduel

Le graphe résiduel représente le graphe original mais avec des capacités de flux ajustées en fonction des flux déjà présents dans le graphe. Dans ce graphe, chaque arête possède une

capacité résiduelle, qui correspond à la différence entre la capacité maximale de l'arête et le flux actuellement en transit. Ce graphe résiduel est constamment mis à jour à chaque itération de l'algorithme, reflétant ainsi les changements dans les flux à travers le réseau de couloirs.Dans le graphe résiduel, si une arête (u,v) présente dans le graphe original possède un flux f et une capacité c et que le flux résiduel (c-f) est positif, alors l'arête (u,v) y sera incluse avec un flux de (c-f). Parallèlement, une arête inverse (v,u) sera créée avec un flux équivalent à f. Si le flux résiduel de l'arête (u,v) est nul, alors cette arête n'est pas ajoutée au graphe résiduel.
Pour le construire, on effectue un `fold_edge` pour parcourir toutes les arêtes du graphe. Puis, on vérifie la capacité résiduelle de chaque arête ; si elle est strictement positive, on ajoute les arêtes correspondantes au graphe résiduel.

3.1.1 Exemple

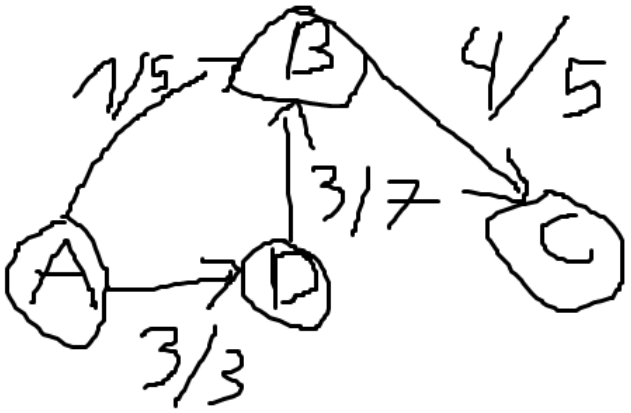

Figure 1: Graphe d'entrée

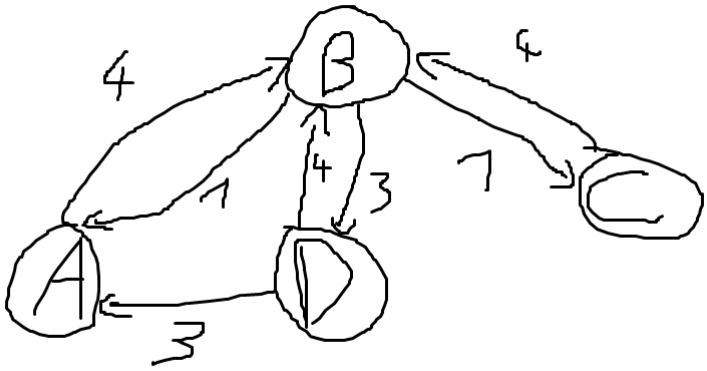

Figure 2: Graphe résiduel associé

3.2 Graphe de Niveau

Un graphe de niveau, dérivé du graphe résiduel, est essentiel pour identifier efficacement les chemins augmentants. Un chemin augmentant est un chemin de la source à la destination dans le graphe résiduel qui peut transporter un flux supplémentaire.Dans le graphe de niveau, chaque arête connecte uniquement des noeuds dont les niveaux sont consécutifs. Le niveau représente la distance minimale d'un noeud par rapport à la source, mesurée en nombre d'arêtes traversées. En se concentrant sur les arêtes qui connectent des noeuds de niveaux consécutifs, le graphe de niveau permet de simplifier la recherche des chemins augmentants, en s'assurant que ces chemins progressent vers la destination sans boucles ni détours inutiles.
La construction du graphe de niveau débute par l'initialisation d'une carte des niveaux, c'est un dictionnaire où chaque noeud du graphe est associé à son niveau, initialisé avec une valeur de niveau de -1, signifiant qu'ils ne sont pas encore visités, est mis à chaque noeud. Ensuite, un parcours en largeur est effectué à partir du noeud source. À chaque étape, le niveau du noeud courant est mis à jour dans la carte des niveaux, assurant que chaque noeud est visité une seule fois et que son niveau le plus bas possible est enregistré.
Pour chaque noeud visité, ses voisins dans le graphe résiduel sont examinés. Si un voisin n'a pas encore été visité ou si son niveau actuel peut être réduit, il est ajouté à la file d'attente du BFS avec un niveau incrémenté. Cette approche garantit que seuls les chemins valides et augmentants sont pris en compte, car elle élimine les chemins qui reviennent en arrière ou qui ne contribuent pas à l'augmentation du flux.
Une fois le parcours en largeur terminé, le graphe de niveau est construit en ajoutant des arêtes entre les noeuds dont les niveaux sont adjacents et pour lesquels il existe un flux résiduel positif dans le graphe résiduel. Ainsi, le graphe de niveau résultant fournit un moyen efficace de naviguer dans le réseau, en concentrant la recherche sur les chemins qui sont les plus susceptibles d'augmenter le flux global. Cet outil est indispensable pour l'optimisation de l'algorithme de Dinic et joue un rôle important dans la maximisation du flux.

3.2.1 Exemple

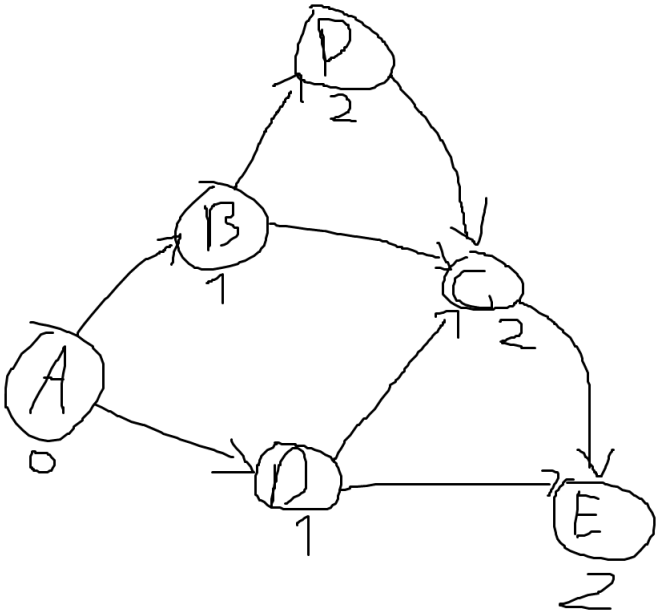

Figure 3: Graphe d'entrée

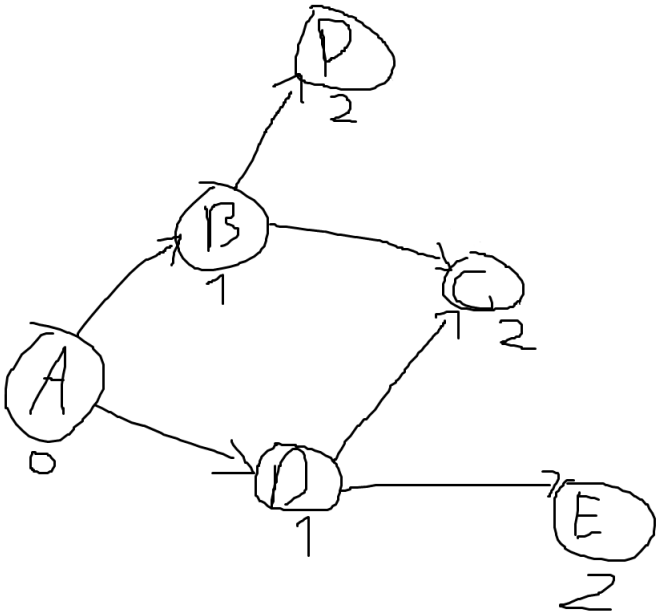

Figure 4: Graphe de niveau associé

3.3 Détermination et Ajout du Flot Bloquant

La phase de détermination et d'ajout du flot bloquant débute par l'identification du flot bloquant sur un chemin augmentant identifié dans le graphe de niveau. Le flot bloquant est défini comme étant le flux le plus faible rencontré sur ce chemin, et sa détermination s'effectue à travers une fonction récursive qui parcourt le chemin en question.

Une fois ce flot bloquant identifié, une mise à jour du graphe est nécessaire pour incorporer ce nouveau flux. Ce processus consiste à augmenter le flux de chaque arête le long du chemin augmentant par la valeur du flot bloquant, tout en ajustant simultanément le flux dans la direction opposée pour préserver l'équilibre des flux. Cette étape de mise à jour est primordiale pour le maintien de la précision du graphe résiduel, qui doit refléter fidèlement les capacités de flux restantes après chaque itération de l'algorithme.
L'ajustement du flux selon le flot bloquant est une stratégie clé pour maximiser l'efficacité de l'utilisation des capacités de flux dans le réseau. Il garantit que chaque chemin augmentant contribue de manière optimale à l'augmentation du flux global, permettant ainsi de se rapprocher progressivement du flux maximal possible dans le réseau.

3.4 Algorithme de Dinic

L'algorithme de Dinic, en utilisant les graphes résiduels et de niveau, optimise la recherche de chemins augmentants et la mise à jour des flux. À chaque itération, il construit un graphe de niveau, trouve les chemins augmentants et les flots bloquants, et met à jour le graphe résiduel en conséquence. Cette méthode itérative est répétée jusqu'à ce qu'aucun chemin augmentant ne soit trouvé dans le graphe de niveau, indiquant que le flux maximal a été atteint.

Ces étapes de l'algorithme de Dinic illustrent une approche efficace pour maximiser le flux de café entre le bar et le bureau de M. Forest, en prenant en compte les capacités variables des couloirs de l'ENSIIE.

4 Conclusion

Pour s'assurer que mon implémentation répond aux exigences, il serait pertinent de la tester avec une série de graphes standards. L'objectif serait de contrôler sa performance en termes de consommation de ressources et de rapidité d'exécution. Concrètement, cela implique de soumettre le programme à divers cas de graphes représentatifs, afin d'évaluer son efficacité en termes de mémoire et de temps de traitement. Cette démarche permettrait de confirmer la viabilité de l'implémentation dans des contextes pratiques et d'identifier les éventuelles optimisations nécessaires.

Concernant la complexité temporelle, des ajustements pourraient être nécessaires, surtout pour la recherche de flot bloquant. Actuellement, mon approche se concentre sur un chemin unique dans le graphe de niveau. Une optimisation pourrait être d'utiliser tous les chemins les plus courts pour la mise à jour du graphe résiduel. De plus, pour réduire la complexité temporelle, envisager de construire le graphe de niveau directement durant le parcours en largeur et non à partir d'un dictionnaire où les niveaux sont stocker pourrait être une solution plus efficace.

Enfin, pour réduire la complexité en mémoire, limiter l'utilisation de structures intermédiaires telles que les dictionnaires pour stocker les niveaux ou les successeurs pourrait alléger le programme. Simplifier ces structures pourrait rendre le programme plus rapide et moins gourmand en mémoire.

Pour optimiser le projet, une stratégie consisterait à compresser les chemins dans le graphe résiduel. Cela implique de regrouper les séquences de noeuds et d'arêtes similaires pour réduire le nombre total à traiter. Par ailleurs, stocker les résultats précédents dans une base de données pourrait permettre de réutiliser les calculs déjà effectués lors de légères modifications du graphe d'entrée, évitant ainsi de recalculer entièrement le flux.