GRAPHES DE FLOT – VERSION ++

I Introduction aux graphes de flot

Définition – Graphe de flot

Un *graphe de flot* est un quintuplet $G = (S, A, c, s, t)$ où :

- (S, A) est un graphe orienté sans boucle (sans arc de la forme (v, v)) ;
- $s \in S$ est une source de (S, A) (degré entrant nul) et $t \in S$ est un puits de (S, A) (degré sortant nul) ;
- $c : A \to \mathbb{R}_+$ est une fonction dite de *capacité*.

On étend c à tous les couples (u, v) de sommets en posant $c(u, v) = 0$ si $(u, v) \notin A$.

La figure 1 représente un graphe de flot G_0. Les capacités sont représentées sur les arcs.

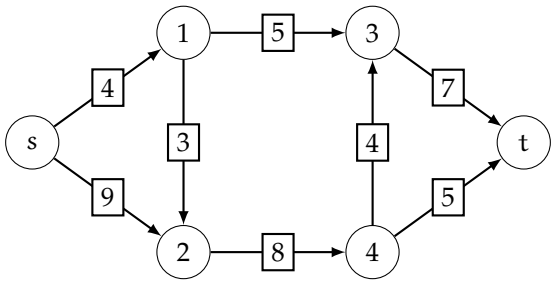


FIGURE 1 – Le graphe de flot G_0.

Définition – Flot

Soit $G = (S, A, c, s, t)$ un graphe de flot. On appelle *flot* sur G une fonction $f : S^2 \to \mathbb{R}$ vérifiant les trois propriétés suivantes :

- pour $u, v \in S$, $f(u, v) = -f(v, u)$ — **antisymétrie**
- pour $u, v \in S$, $f(u, v) \leqslant c(u, v)$ — **respect de la capacité**
- pour $u \in S \setminus \{s, t\}$, $\sum_{v \in S} f(u, v) = 0$ — **conservation**

On appelle *débit* du flot f la valeur $|f| = \sum_{u \in S} f(s, u)$.

La figure 2 représente le graphe de flot G_0 et un flot f compatible avec G. Pour chaque arc (u, v), on représente sur l'arc $f(u, v)/c(u, v)$. Les autres valeurs nulles et négatives de f peuvent se déduire des valeurs positives.

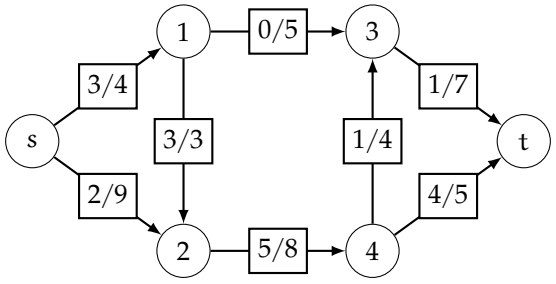


FIGURE 2 – Le graphe de flot G_0 et un flot f.

Dans la suite du sujet, G désignera toujours un graphe de flot et f un flot sur G.

▶ **Question 1** Justifier que $f(u,v) > 0$ implique $(u,v) \in A$ et $f(u,v) < 0$ implique $(v,u) \in A$.

Définition – Flux

Pour $u \in S$, on appelle :

- *flux sortant de* u la quantité $\varphi_+(u) = \sum_{v \in S, f(u,v)>0} f(u,v)$;
- *flux entrant de* u la quantité $\varphi_-(u) = \sum_{v \in S, f(u,v)<0} f(v,u)$;
- *flux net de* u la quantité $\varphi(u) = \varphi_+(u) - \varphi_-(u)$.

▶ **Question 2** Montrer que la condition **conservation** est équivalente à la propriété suivante :

$$\forall u \in S \setminus \{s,t\},\ \varphi(u) = 0$$

▶ **Question 3** Montrer que $|f| = \varphi(s) = -\varphi(t)$.

Le problème MaxFlow, auquel on s'intéresse dans le reste du sujet, est défini comme suit :

Entrée : un graphe de flot G

Sortie : un flot f maximal sur G, c'est-à-dire un flot tel que $|f|$ soit maximal.

▶ **Question 4** Représenter graphiquement une solution à MaxFlow pour l'instance représentée sur la figure 1. On justifiera que le flot proposé est maximal.

II Algorithme de Ford-Fulkerson

Définition – Saturation

- La *capacité disponible* d'un arc $(u,v) \in A$ est la quantité $c(u,v) - f(u,v)$.
- Un arc est dit *saturé* si sa capacité disponible vaut zéro.
- La *capacité disponible* d'un chemin γ de s à t est le minimum des capacités disponibles de ses arcs.
- Un chemin de s à t est dit *saturé* si sa capacité disponible vaut zéro.

Si un chemin γ de s à t n'est pas saturé, il dispose d'une capacité disponible $m > 0$. On définit alors l'action de *saturation* du chemin γ comme la modification suivante de f :

- pour chaque arc (u,v) de γ :
  - $f(u,v) \leftarrow f(u,v) + m$;
  - $f(v,u) \leftarrow f(v,u) - m$

On remarquera que, après saturation du chemin γ, f est toujours un flot sur G et que γ est désormais saturé.

La figure 3 montre le résultat de la saturation du chemin $(s,2,4,3,t)$ dans le graphe G_0 à partir du flot f de la figure 2. On a augmenté le débit de 3 le long du chemin. Les nouveaux arcs saturés sont $(2,4)$ et $(4,3)$.

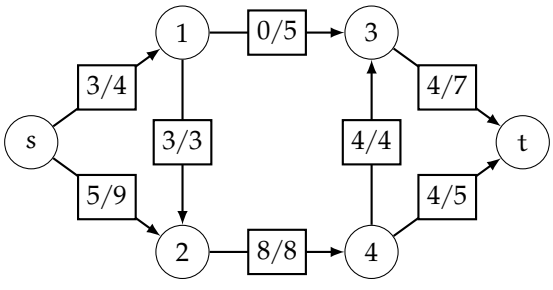


Figure 3 – Le graphe de flot G_0 et le flot f après saturation du chemin $(s,2,4,3,t)$.

On considère l'algorithme suivant :

Algorithme 1 Algorithme glouton pour MaxFlow

Entrées : Un graphe de flot $G = (S, A, c, s, t)$
  $f(u, v) \leftarrow 0$ pour tout $(u, v) \in S^2$
  tant que il existe un chemin non saturé de s à t **faire**
    saturer ce chemin
  renvoyer f

▶ **Question 5** Expliquer comment trouver un chemin non saturé de s à t dans un graphe de flot, étant donné un flot f.

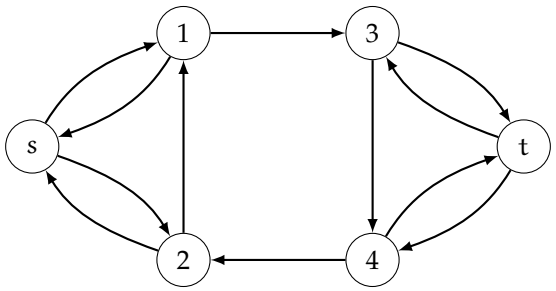
▶ **Question 6** Terminer l'exécution de l'algorithme 1 sur le graphe G_0 à partir du flot f de la figure 3 et représenter graphiquement le résultat.

La question précédente et la question 4 montrent que l'algorithme 1 ne renvoie pas toujours un flot maximal. Pour corriger ce problème, il faut s'autoriser à faire « refluer » le flot en arrière le long d'un arc.

Définition – Graphe résiduel, chemin améliorant

On définit le *graphe résiduel* $G_f = (S, A_f)$ où $A_f = \{(u, v) \in S^2 \mid c(u, v) - f(u, v) > 0\}$. Un chemin de s à t dans G_f est appelé *chemin améliorant pour* f.

On a représenté ci-dessous le graphe résiduel associé au flot de la figure 3 :



▶ **Question 7** Justifier que si $(u, v) \in A_f$, alors $(u, v) \in A$ ou $(v, u) \in A$.

Remarque

On fera bien attention à ne pas oublier dans la suite que le graphe résiduel peut contenir des arcs qui n'étaient pas dans le graphe G initial.

▶ **Question 8** Représenter graphiquement le graphe résiduel de G_0 pour le flot obtenu à la question 6.

L'algorithme de Ford-Fulkerson est alors le suivant :

Algorithme 2 Ford-Fulkerson

Entrées : Un graphe de flot $G = (S, A, c, s, t)$
Sorties : Un flot maximal f sur G
  $f(u, v) \leftarrow 0$ pour tout $(u, v) \in A$
  tant que il existe un chemin améliorant pour f **faire**
    saturer ce chemin
  renvoyer f

▶ **Question 9** Terminer l'exécution de l'algorithme de Ford-Fulkerson sur le graphe G_0 avec le flot obtenu à la question 6.

III Programmation

III.1 Préliminaires

On travaille dans le langage C, et l'on suppose avoir inclus tous les entêtes standard :

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <assert.h>
#include <limits.h> // fournit INT_MAX, INT_MIN (entre autres)
```

On suppose de plus disposer des deux fonctions suivantes (dont la spécification devrait être évidente) :

```c
int min(int x, int y);
int max(int x, int y);
```

On représente en C un graphe de flot par la structure suivante :

```c
typedef int vertex;

struct flow_graph {
    int n;
    vertex s;
    vertex t;
    vertex **adj;
    int *degrees;
    int **capacity;
};

typedef struct flow_graph flow_graph;
```

- L'ensemble S des sommets est $[0 \dots n-1]$.
- `s` et `t` sont respectivement le sommet s source et le sommet t puits.
- `degrees` est un tableau de taille n tel que le degré sortant du sommet u soit égal à `degrees[u]`.
- `adj` est un tableau de taille n, dont les éléments sont des tableaux d'entiers.
- Pour $u \in [0 \dots n-1]$, le tableau `adj[u]` est de taille `degrees[u]` et contient les successeurs de u.
- `capacity` est une matrice de taille $n \times n$ (un tableau de n tableaux de n entiers chacun).
- `capacity[u][v]` indique la valeur de $c(u, v)$.

Remarque

Important : on supposera que l'arc (v, u) existe dans le graphe dès que l'arc (u, v) existe (quitte à ce que $c(v, u)$ soit égal à zéro). Autrement dit, si v apparaît dans le tableau `adj[u]`, alors u apparaît dans le tableau `adj[v]`.

▶ **Question 10** Écrire une fonction `zeroes` prenant en entrée un entier n (supposé strictement positif) et renvoyant une « matrice » de taille $n \times n$ initialisée avec des zéros. Par *matrice*, on entend un pointeur vers un bloc alloué `mat` de taille n, telle que `mat[i]` soit un bloc alloué de taille n rempli de zéros, et ce pour $0 \leqslant i < n$.

```c
int **zeroes(int n);
```

▶ **Question 11** Écrire une fonction `free_matrix` permettant de libérer la mémoire associée à une matrice du type de celle renvoyée par `zeroes`.

```c
void free_matrix(int **mat, int n);
```

Dans toute la suite, un flot sera représenté par une matrice d'entiers. On pourra toujours supposer que, si une fonction prend en entrée un graphe de flot G et un flot `f` représenté par un `f` de type `int**`, alors `f` a la bonne dimension (est une matrice $n \times n$, autrement dit).

III.2 Structure de file

On suppose que l'on dispose d'une structure de file avec l'interface suivante :

```
// création d'une file vide
queue *queue_create(void);

// libération de la mémoire
void queue_free(queue *q);

// nombre d'éléments présents dans la file
int queue_length(queue *q);

// extraction du plus ancien élément (erreur si vide)
vertex queue_pop(queue *q);

// ajout d'un élément (la capacité de la file n'est pas limitée)
void queue_push(queue *q, vertex v);
```

On suppose de plus que l'on a les garanties de complexité suivantes :

- `queue_create`, `queue_push`, `queue_pop` et `queue_length` sont en $O(1)$;
- `queue_free` est en $O(n)$, où n est la longueur de la file.

III.3 Implémentation de l'algorithme de Ford-Fulkerson

▶ **Question 12** Écrire une fonction `bfs_residual` prenant en entrée un graphe de flot G à n sommets et un flot f sur G, et effectuant une recherche en largeur d'un chemin de s à t dans le graphe résiduel G_f. La fonction renverra un pointeur vers un bloc alloué `parent` de taille n codant l'arbre associé à ce parcours de la manière suivante :

- `parents[g->s]` vaut `g->s` ;
- pour chaque autre sommet u visité lors du parcours, `parents[u]` vaut v, où v est le sommet depuis lequel on a exploré u ;
- pour les sommets u n'ayant pas été explorés, `parents[u]` vaut -1.

On demande une complexité en $O(|S| + |A|)$.

```
vertex *bfs_residual(flow_graph *g, int **f);
```

▶ **Question 13** Écrire une fonction `path_capacity` qui prend en entrée un graphe de flot G, un flot f et un arbre de parcours `parent` tel que renvoyé par la fonction `bfs_residual` et renvoie la capacité disponible du chemin reliant s à t dans l'arbre `parent`. Si un tel chemin n'existe pas, la fonction renverra zéro.

```
int path_capacity(flow_graph *g, int **f, vertex *parent);
```

▶ **Question 14** Écrire une fonction `saturate_path` prenant en entrée un graphe de flot G, un flot f et un arbre de parcours `parent` et ayant pour effet de saturer le chemin éventuel reliant s à t dans l'arbre `parent` (en modifiant f). Cette fonction renverra `true` s'il y avait un chemin, `false` sinon.

```
bool saturate_path(flow_graph *g, int **f, vertex *parent);
```

▶ **Question 15** Écrire une fonction `step` prenant en entrée un graphe de flot G et un flot f et ayant le comportement suivant :

- s'il n'existe pas de chemin améliorant pour f, alors f n'est pas modifié et la fonction renvoie `false` ;
- sinon, la fonction détermine un chemin améliorant, modifie f en saturant ce chemin et renvoie `true`.

```
bool step(flow_graph *g, int **f);
```

▸ **Question 16** Déterminer la complexité de la fonction `step`.

▸ **Question 17** Écrire une fonction `ford_fulkerson` prenant en entrée un graphe de flot G et renvoyant le flot f calculé par l'algorithme de Ford-Fulkerson.

```
int **ford_fulkerson(flow_graph *g);
```

IV Complexité et terminaison

▸ **Question 18** Dans cette question, on suppose que les capacités sont entières, mais on ne fait pas d'hypothèse sur l'algorithme de parcours utilisé pour trouver un chemin améliorant, à part que cette recherche se fait en $O(|S| + |A|)$. Montrer que l'algorithme de Ford-Fulkerson termine, et majorer sa complexité temporelle en fonction de $|S|$, $|A|$ et M, où M est le débit d'un flot maximal sur G.

Remarque

On ne demande pas ici de montrer que l'algorithme renvoie un flot maximal.

V Correction de l'algorithme

Définition – Coupe

Soit $G = (S, A, c, s, t)$ un graphe de flot. On appelle *coupe* de G un ensemble $X \subseteq S$ tel que $s \in X$ et $t \in \overline{X}$, où $\overline{X}$ désigne le complémentaire de X dans S.
La *capacité* d'une coupe X est la quantité $C(X) = \sum_{(u,v)\in X\times\overline{X}} c(u,v)$.

▸ **Question 19** Dans le graphe G_0 donné figure 1, déterminer la capacité de la coupe $X = \{s, 1, 3\}$.

▸ **Question 20** Soit f un flot de X. Montrer que $|f| = \sum_{(u,v)\in X\times\overline{X}} f(u,v) \leqslant C(X)$.

▸ **Question 21** Montrer l'équivalence entre les trois propriétés suivantes :

1. f est un flot maximal ;
2. il n'existe pas de chemin améliorant pour f ;
3. il existe une coupe X telle que $|f| = C(X)$.

Remarque

Ce résultat est connu sous le nom de théorème MaxFlow-MinCut.

▸ **Question 22** En déduire la correction de l'algorithme de Ford-Fulkerson.

VI Réduction du couplage maximum

▸ **Question 23** Expliquer comment résoudre le problème du couplage maximum dans un graphe (non orienté) biparti en trouvant un flot maximal dans un graphe de flot dont on détaillera la construction. Justifier la correction de votre algorithme et déterminer sa complexité temporelle.

VII Algorithme de Edmonds-Karp

Dans cette partie, on suppose explicitement que la recherche d'un chemin améliorant s'est faite à l'aide d'un parcours en largeur. Cette variante de l'algorithme de Ford-Fulkerson (celle que nous avons programmée plus haut) est connue sous le nom d'algorithme de Edmonds-Karp. On pose les définitions suivantes :

- f_i est le flot obtenu après i étapes de saturation (i passages dans la boucle principale de l'algorithme) – en particulier, f_0 est le flot initial, identiquement nul ;
- $G_i = (A_i, S)$ est le graphe résiduel associé au flot f_i ;
- pour chaque sommet v, $niveau_i(v)$ est la distance de s à v (en nombre d'arcs) dans le graphe G_i, en posant $niveau_i(v) = \infty$ si v n'est pas accessible depuis s dans G_i. On remarquera que cette distance est également la profondeur de v dans un arbre de parcours en largeur de G_i enraciné en s.

▶ **Question 24** Montrer que :

- si $(u, v) \in A_i \setminus A_{i+1}$, alors $niveau_i(v) = niveau_i(u) + 1$;
- si $(u, v) \in A_{i+1} \setminus A_i$, alors $niveau_i(u) = niveau_i(v) + 1$.

▶ **Question 25** Montrer que $niveau_i(v) \geqslant niveau_{i-1}(v)$ pour tout sommet v et tout entier $i > 0$.
Indication : on procédera par récurrence forte sur $niveau_i(v)$.

▶ **Question 26** En déduire qu'un arc (u, v) disparaît au plus $|S|/2$ fois du graphe résiduel au cours de l'exécution de l'algorithme d'Edmonds-Karp. Autrement dit, il existe au plus $|S|/2$ entiers i tels que $(u, v) \in A_i \setminus A_{i+1}$.

▶ **Question 27** Montrer finalement que la complexité temporelle de l'algorithme de Edmonds-Karp est en $O(np(n + p))$, où l'on note $n = |S|$ et $p = |A|$.