TP Python: le retour de Ford-Fulkerson

1 Définitions et vérifications

Dans tout le TP, on considère des graphes orientés, dont les arcs sont munis d'un poids entier. Vous pouvez récupérer une petite annexe sur le site web.

Réseau de transport. Un graphe $G = (V, E)$ est appelé un *réseau de transport* si :

— il existe dans V deux sommets particuliers, la source s et le puits t ;

— il existe une fonction $c : E \to \mathbb{N}^*$ appelé capacité et évaluant les arcs E de G.

On étend c en une fonction $V \times V \to \mathbb{N}$ en convenant que $c(v_i, v_j) = 0$ si $(v_i, v_j) \notin E$.

Flot. Un *flot* de G est une fonction $f : V \times V \to \mathbb{Z}$, qui associe à chaque couple (u, v) de V^2 une quantité de flot $f(u, v)$. On définit pour f une fonction bilan $\mathcal{B} : V \to \mathbb{Z}$ telle que :

$$\forall y \in V, \qquad \mathcal{B}(y) = \left(\sum_{x \in V} f(y, x)\right) - \left(\sum_{x \in V} f(x, y)\right)$$

En voyant le flot $f(u, v)$ comme une quantité de matière qui transite de u à v, $\mathcal{B}(y)$ n'est autre que la différence entre la quantité de matière partant de y et la quantité de matière arrivant en y. Le flot f doit vérifier les conditions suivantes, pour un certain $\nu \geq 0$:

$$\begin{cases} \mathcal{B}(s) = \nu \\ \mathcal{B}(t) = -\nu \\ \forall u \in V \backslash \{s, t\}, \quad \mathcal{B}(u) = 0 \end{cases} \tag{1}$$

et

$$\forall (u, v) \in V^2, \quad 0 \leq f(u, v) \leq c(u, v) \tag{2}$$

L'entier ν est appelé la *valeur du flot* f. Autrement dit, f est un flot de valeur $\nu \geq 0$ si une quantité de matière ν transite dans le réseau de transport de la source au puits, en passant par les nœuds intermédiaires sans excéder la capacité de chaque arc.

Représentation. Si V est de cardinal n, on représente le réseau de transport G ou un flot f sur G comme une matrice carrée $n \times n$ à coefficients entiers. On convient que $V = [\![0, n-1]\!]$, que la source est le sommet 0 et le puits le sommet $n - 1$. En Python, on convient qu'une matrice est représentée comme une liste de listes.

Question 1. Écrire une fonction `est_flot(G,F)` prenant en entrée deux matrices carrées de même taille `G` et `F`, avec `G` un réseau de transport, et renvoyant un booléen indiquant si `F` est un flot sur `G`.

```
>>> est_flot(G1, F1)
True
>>> est_flot(G1, F13)
False
>>> est_flot(G1, F12)
False
```

Question 2. Écrire une fonction `valeur_flot(F)` prenant en entrée un flot et renvoyant sa valeur (l'entier ν défini dans l'énoncé).

```
>>> valeur_flot(F1)
17
```

2 Coupe et théorème Max-Flow, Min-Cut

Coupe. On définit une coupe dans un réseau de transport G comme une partition de V en deux sous-ensembles $V = S \cup T$, avec $s \in S$ et $t \in T$. On définit la valeur de la coupe comme la quantité $c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$.

Question 3. On représente l'ensemble S comme une liste de booléens `S` de taille n avec n le nombre de sommets de G : pour $i \in [\![0, n-1]\!]$, `S[i]` vaut `True` si et seulement si $i \in S$. Écrire une fonction `valeur_coupe(G,S)` renvoyant la valeur de la coupe (S,T) définie par `S`.

```
>>> valeur_coupe(G1, [True, True, True, True, False])
17
>>> valeur_coupe(G1, [True, False, True, True, False])
21
```

Il est facile de voir que la valeur d'un flot quelconque est inférieur à la valeur de toute coupe. Le théorème Max-Flow, Min-Cut stipule que dans un réseau de transport, il y a égalité entre valeur maximale d'un flot et valeur minimale d'une coupe. Les tests précédents montre que `F1` est un flot maximal de `G1`.

Le reste du TP est dévolu à l'implémentation de l'algorithme de Ford-Fulkerson, qui permet de calculer un flot maximal d'un réseau de transport, et fournit gratuitement une coupe minimale.

3 Parcours en profondeur

L'algorithme de Ford-Fulkerson consiste à partir du flot nul, et à effectuer des parcours en profondeur sur le *graphe des résidus* (défini plus tard) pour tenter d'augmenter la valeur du flot. Lorsque ceci n'est plus possible, le flot est maximal. Cette section est dédiée à l'implémentation du parcours en profondeur.

Question 4. *Reconstruction d'un chemin entre u et v.* On considère un tableau π des prédecesseurs dans un parcours depuis u. On suppose v accessible de sorte qu'en remontant de v à u via π, on peut construire un chemin de u à v à l'envers. Écrire une fonction `reconstruit(pi, u, v)` renvoyant le chemin de u à v, sous forme de liste. On pourra (ce n'est pas obligatoire) considérer que $\pi[u] = u$.

```
>>> reconstruit([0, 2, 5, 5, 3, 0, 4], 0, 6)
[0, 5, 3, 4, 6]
```

On rappelle que l'on considère des graphes en représentation matricielle, dont les poids sont positifs. Avec `G` la matrice associée à G, on considère qu'un arc (u,v) est présent dans G si `G[u][v]` est *strictement positif*.

Question 5. *Parcours en profondeur*. Écrire une fonction `parcours(G, u, v)` prenant en paramètre le graphe G et deux sommets u et v, et renvoyant un couple `(b, L)` tel que :

- s'il existe un chemin de u à v dans G (n'empruntant que des arcs de poids strictement positifs) alors `b` vaut `True`, et `L` est la liste des sommets sur un tel chemin ;
- sinon, `b` vaut `False`, et `L` est une liste de booléens de taille n, indiquant l'ensemble des sommets accessibles depuis `u`.

```
>>> parcours(G1, 0, 4)
(True, [0, 4])
>>> parcours(G2, 0, 4)
(True, [0, 2, 4])
>>> parcours(G3, 0, 4)
(False, [True, True, True, True, False])
```

Indication : on pourra utiliser une liste `a_traiter` de sommets, qu'on considérera comme une pile via les méthodes `append` et `pop`. On fera aussi usage d'un tableau de booléens `deja_vu`, et du tableau `pi` des prédecesseurs. On impose une complexité $O(n^2)$
Remarque : il se peut que votre algorithme ne fonctionne pas exactement comme le mien, ce n'est pas grave, vérifier qu'il est néanmoins correct !

Question 6. *Valeur d'augmentation le long d'un chemin*. Si l'algorithme précédent renvoie effectivement un chemin, on définit la *valeur d'augmentation* le long du chemin comme le minimum des poids des arcs le long du chemin. Écrire une fonction `calcule_valeur(G, chem)` renvoyant cette valeur.

```
>>> calcule_valeur(G1, [0, 4])
4
>>> calcule_valeur(G2, [0, 2, 4])
9
```

4 Algorithme de Ford Fulkerson

Graphe des résidus. Si G est un réseau de transport et f un flot sur G, on définit le *graphe des résidus* comme le graphe R ayant même sommets que G, et donc les poids des arcs sont donnés par la fonction r définie comme suit :

$$\forall (u,v) \in V^2, \qquad r(u,v) = c(u,v) - f(u,v) + f(v,u)$$

Question 7. *Calcul du graphe des résidus.* Écrire une fonction `graphe_residu(G,F)` prenant en entrée un réseau de transport et un flot et renvoyant le graphe des résidus.

```
>>> graphe_residu(G1, F1)==G3
True
```

L'algorithme de Ford-Fulkerson pour le calcul d'un flot maximal peut être résumé comme suit :

Algorithme 1 : Ford et Fulkerson

Entrée : Un réseau de transport $G = (V, E)$
Sortie : Un flot maximal f
pour tout $(u, v) \in V^2$ **faire**
  $f(u, v) \leftarrow 0$
$s \leftarrow 0; t \leftarrow n - 1$ avec n le nombre de sommets de G;
tant que *Vrai* **faire**
  Calculer le graphe des résidus R de G vis à vis du flot f;
  si *il existe un chemin C de s à t dans R* **alors**
    Calculer la valeur ω d'augmentation de C;
    actualiser le flot f en rajoutant la valeur ω le long des arcs de C
  sinon
    retourner (f)

Tous les ingrédients pour l'implémentation ont déja été écrits, sauf l'actualisation du flot avec la valeur d'augmentation donnée par le chemin trouvé dans le graphe des résidus. Pour ω cette valeur d'augmentation et (u, v) un arc du chemin, on distingue deux cas :

— si $c(u, v) - f(u, v) \geq \omega$, on peut simplement augmenter $f(u, v)$ de ω ;
— sinon, cela signifie que $f(v, u) > 0$. Il faut alors diminuer $f(v, u)$ et éventuellement augmenter $f(u, v)$.

De part la définition de la fonction bilan, on remarque que cette opération permet d'augmenter la valeur du flot de la quantité ω.

Question 8. *Mise à jour du flot.* Écrire une fonction `mise_a_jour(G, F, c, w)` prenant en paramètre un réseau de transport, un flot, un chemin de la source au puits, et `w` la valeur d'augmentation du chemin dans le graphe des résidus, et modifiant le flot de manière à augmenter sa valeur de `w`. On suivra la discussion précédente.

Question 9. Déduire des questions précédentes une implémentation `ford_fulkerson(G)` de l'algorithme de Ford-Fulkerson. Votre fonction renvoie un flot maximal.

```
>>> ford_fulkerson(G1)==F1
True
```

Correction de l'algorithme et preuve du théorème Max-Flow, Min-Cut. Lorsque l'algorithme de Ford-Fulkerson lancé sur G s'arrête avec un flot f, il n'y a pas de chemin de la source au puits dans le graphe R des résidus. En notant S l'ensemble des sommets atteignables depuis s dans R, et $T = V \backslash S$, on montre facilement que (S, T) est une coupe de G, de capacité égale à la valeur du flot f. Ceci montre à la fois la correction de l'algorithme de Ford-Fulkerson, et le théorème Max-Flow, Min-Cut.

Un mot sur la terminaison et la complexité. Si les capacités des arcs ne sont pas à valeurs entières, l'algorithme de Ford-Fulkerson ne termine pas en général. Ici, on a supposé les arcs à poids entiers, notons C la capacité maximale d'un arc du graphe. Chaque étape de l'algorithme de Ford-Fulkerson augmente d'au moins 1 la valeur du flot, et il est clair que la source ne peut délivrer une quantité de flot strictement supérieure à Cn (et même $C(n-1)$). Ainsi, $O(Cn)$ étapes sont effectuées. Chaque étape ayant un coût $O(n^2)$, on en déduit une complexité $O(Cn^3)$. Ce résultat n'est pas très bon car la complexité est exponentielle en la taille $\log C$ de C. Ceci dit, l'algorithme admet plusieurs variantes de complexités polynomiales en la taille de l'entrée, voir la suite.

Question 10. *Vérification.* Écrire une fonction `flot_maximal(G,F)` prenant en entrée un graphe et un flot, et indiquant si le flot associé est maximal. On calculera le graphe des résidus associé, et on vérifiera qu'un parcours lancé depuis la source n'atteint pas le puits, et que la valeur de la coupe définie par le parcours est égale à la valeur du flot.

```
>>> flot_maximal(G1, F1)
True
```

5 Optimisation

Il existe plusieurs variantes rendant la complexité de l'algorithme de Ford-Fulkerson polynomiale. Citons-en une simple : l'algorithme d'Edmonds-Karp. Celui-ci consiste à privilégier les chemins augmentant « courts » (en nombre d'arcs), donc à utiliser un parcours en largeur plutôt qu'un parcours en profondeur pour la recherche des chemins augmentants dans le graphe résiduel : chaque chemin s'obtient en temps $O(|V|^2)$. De plus chaque parcours qui augmente la quantité de flot va *saturer* (au moins) un arc, et on s'aperçoit que la distance entre la soure et l'arc saturé augmente à chaque parcours. On peut montrer que la complexité de l'algorithme devient $O(|V|^5)$ avec cette approche ($O(|V|\,|E|^2)$ avec une implémentation « creuse »).

Question 11. Vous trouverez une implémentation de file sous forme de classe dans le fichier joint. Les fonctions sont les suivantes (`f` est une file) :

- `file()` crée et renvoie une file vide ;
- `f.file_vide()` renvoie un booléen indiquant si `f` est vide ;
- `f.enfiler(x)` rajoute `x` à la file `f` ;
- `f.defiler()` supprime de `f` l'élément en tête de file (si `f` est non vide), et le renvoie.

Reprendre l'algorithme de parcours du graphe pour effectuer un parcours en largeur à la place d'un parcours en profondeur.