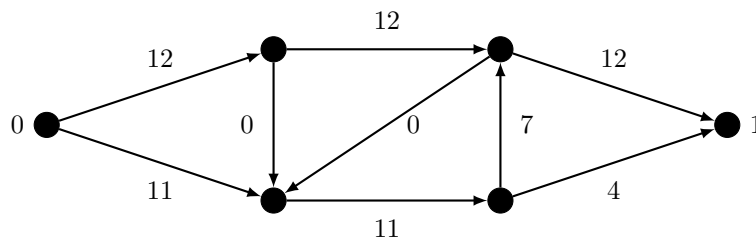


DM 2 corrigé : Flots

Option informatique

I Définitions

1. Dans le graphe ci-dessous, on a indiqué les capacités sur chaque arc. Trouver un flot de valeur maximum et recopier ce graphe en écrivant le flot passant par chaque arc, à la place des capacités. On ne demande aucune justification.
► On trouve un flot maximum de valeur $12+11 = 23$ (remarque: il n'y a pas unicité du flot maximum):



II Algorithme de Ford-Fulkerson

1. Comment modifier f de façon à obtenir un flot $f_{\vec{P}}$ de \vec{G} de valeur $|f| + c_f(\vec{P})$?

► Pour tout arc (u, v) de \vec{P} :

- Si $(u, v) \in \vec{E}$, on augmente $f(u, v)$ de $c_f(\vec{P})$.
- Si $(v, u) \in \vec{E}$, on diminue $f(v, u)$ de $c_f(\vec{P})$.

Si \vec{P} utilise les sommets $0 = v_1 \rightarrow v_2 \dots \rightarrow v_k = 1$ alors la modification de f sur (v_{i-1}, v_i) est compensée par la modification de f sur (v_i, v_{i+1}) : la « loi de Kirchhoff » est toujours vérifiée.

On représente $\vec{G}_f = (V, \vec{E}_f)$ en Caml par une matrice **g** de taille $|V| \times |V|$ contenant $c_f(u, v)$ sur la ligne u , colonne v . Dans toutes les questions de programmation on supposera les capacités entières: **g** sera donc de type **int array array**. La représentation de \vec{G} est la même que celle de \vec{G}_f pour $f = 0$.

On représente \vec{P} par un tableau **pere** des prédécesseurs tel que: (u, v) est un arc de $\vec{P} \implies u = \text{pere}.(v)$.

2. Écrire une fonction **cmin** : **int array array** -> **int array** -> **int** telle que, si **g** est une matrice représentant \vec{G}_f et **pere** un tableau des prédécesseurs de \vec{P} , **cmin g pere** renvoie $c_f(\vec{P})$.

```
let cmin g pere =  
  let rec aux v =  
    let p = pere.(v) in  
    if p = 0 then g.(0).(v)  
    else min g.(p).(v) (aux p) in  
  aux 1;;
```

3. Écrire une fonction **augment** : **int array array** -> **int array** -> **int** telle que, si **g** est une matrice représentant \vec{G}_f et **pere** un tableau des prédécesseurs de \vec{P} , **augment g pere** modifie **g** pour qu'il représente $\vec{G}_{f_{\vec{P}}}$, où $f_{\vec{P}}$ est le flot de la question II.1. De plus, **augment g pere** renverra $c_f(\vec{P})$.

```
let augment g pere =  
  let cm = cmin g pere in  
  let rec aux v =  
    if v = 0 then cm  
    else (let p = pere.(v) in  
          g.(p).(v) <- g.(p).(v) - cm;  
          g.(v).(p) <- g.(v).(p) + cm;  
          aux p) in  
  aux 1;;
```

On considère l'algorithme suivant (où $f_{\vec{P}}$ est obtenu par la question II.1):

Algorithme de Ford-Fulkerson

Initialement: $\vec{G}_f = \vec{G}$
 Tant qu'il existe un chemin \vec{P} de 0 à 1 dans \vec{G}_f :
 Modifier \vec{G}_f en $\vec{G}_{f_{\vec{P}}}$

4. Écrire une fonction `ford : int array array -> (int array array -> int array) -> int` telle que:

- si `g` est une matrice représentant \vec{G} ,
- si `chemin` est une fonction telle que `chemin g` renvoie un tableau des prédécesseurs d'un chemin (s'il existe) du sommet 0 au sommet 1 dans `g` (si un tel chemin n'existe pas, le tableau `pere` renvoyé par `chemin g` est tel que `pere.(1) = -1`),

`ford g chemin` applique l'algorithme de Ford-Fulkerson (en utilisant `chemin` pour trouver le chemin \vec{P}) et renvoie la valeur du flot f obtenu à la fin de l'algorithme. (On pourra modifier `g` sans faire de copie.)

```
let rec ford g chemin =
  let pere = chemin g in
  if pere.(1) = -1 then 0
  else let cm = augment g pere in
        cm + ford g chemin;;
```

5. On suppose que toutes les capacités de \vec{G} sont entières, c'est à dire $c : \vec{E} \rightarrow \mathbb{N}$. Soit $|f^*|$ la valeur maximum d'un flot de \vec{G} . Montrer que le nombre d'itérations de la boucle « Tant que » de l'algorithme de Ford-Fulkerson est au plus $|f^*|$.

► Si les capacités de \vec{G}_f sont des entiers, alors $c_f(\vec{P})$ est un entier et les capacités de $\vec{G}_{f_{\vec{P}}}$ sont aussi des entiers.

Donc, tout au long de l'algorithme de Ford-Fulkerson, les valeurs de $c_f(\vec{P})$ sont des entiers ≥ 1 . Comme la valeur du flot est 0 initialement, $|f^*|$ au plus à la fin, et augmente de 1 au moins à chaque itération, il y a bien au plus $|f^*|$ itérations. Il peut y avoir exactement $|f^*|$ itérations, par exemple si \vec{G} contient seulement deux sommets 0, 1 et un arc de capacité 1 de 0 vers 1.

Remarque: si les capacités sont irrationnelles (ce qu'on ne considérera pas dans ce DM), l'algorithme de Ford-Fulkerson peut faire « boucle infinie »...

III Correction

Soit $S \subseteq V$ une **coupe** de \vec{G} , c'est à dire un ensemble de sommets tel que $0 \in S$ mais $1 \notin S$. On définit:

- $f(S) = \sum_{\vec{e} \in S^+} f(\vec{e}) - \sum_{\vec{e} \in S^-} f(\vec{e})$ (« flot sortant de S »)
- $c(S) = \sum_{\vec{e} \in S^+} c(\vec{e})$

On peut remarquer que la valeur d'un flot s'écrit aussi $|f| = f(\{0\})$.

1. Soit S une coupe de \vec{G} et f un flot de \vec{G} . Montrer que $f(S) \leq c(S)$.

► $f(S) = \sum_{\vec{e} \in S^+} f(\vec{e}) - \sum_{\vec{e} \in S^-} f(\vec{e}) \leq \sum_{\vec{e} \in S^+} f(\vec{e}) \leq \sum_{\vec{e} \in S^+} c(\vec{e}) = c(S)$.

2. Soit S une coupe de \vec{G} et f un flot de \vec{G} . Montrer que $f(S) = |f|$.

On pourra raisonner par récurrence en précisant clairement l'hypothèse de récurrence.

► Soit f un flot de \vec{G} . Soit $H(k)$: « S est une coupe de \vec{G} avec k sommets $\implies f(S) = |f|$ ». $H(1)$ est vrai car alors $S = \{0\}$.

Supposons $H(k)$ vraie pour un $k \geq 1$. Soit S une coupe de \vec{G} avec $k+1$ sommets et $v \in S$. Alors, en utilisant la définition: $f(S) = f(S \setminus \{v\}) + f(\{v\})$. Or $f(S \setminus \{v\}) = |f|$ par hypothèse de récurrence et $f(\{v\}) = 0$ par la « loi de Kirchoff ». Donc $f(S) = |f|$: $H(k+1)$ est vraie.

3. En déduire que si il existe une coupe S et un flot f tel que $f(S) = c(S)$ alors f est un flot de valeur maximum et S est une coupe de capacité minimum (c'est le théorème max-flow min-cut). Justifiez à présent votre réponse à la question I.1.

► Supposons qu'il existe une coupe S et un flot f tel que $f(S) = c(S)$. Soit f^* un flot de valeur maximum. Alors $|f^*| = f^*(S) \leq c(S) = f(S) = |f|$. Donc f est un flot de valeur maximum et un raisonnement similaire montre que S est une coupe de capacité minimum.

- En déduire que si l'algorithme de Ford-Fulkerson termine, le dernier flot calculé est un flot de valeur maximum de \vec{G} .
 ► Soit S l'ensemble des sommets accessibles depuis 0 dans \vec{G}_f à la fin de l'algorithme de Ford-Fulkerson. Alors S contient 0 mais pas 1 donc est une coupe.
 Comme il n'y a aucun arc de capacité résiduelle strictement positive qui sort de S , $f(S) = c(S)$, ce qui permet de conclure d'après la question précédente.

IV Recherche de chemin

Dans cette partie, on étudie plusieurs façon de trouver un chemin \vec{P} dans l'algorithme de Ford-Fulkerson (c'est à dire des implémentations de fonction `chemin` utilisé par la fonction `ford` de la question II.4).

Parcours en profondeur

- Écrire une fonction `dfs : int array array -> int array` telle que, si g est une matrice représentant un graphe résiduel \vec{G}_f , `dfs` renvoie le tableau `pere` des prédécesseurs d'un parcours en profondeur depuis le sommet 0. Ainsi `pere.(u)` est le sommet qui a permis de visiter u (si u n'a pas été visité on donnera à `pere.(u)` la valeur -1).

```
let dfs g =
  let n = Array.length g in
  let pere = Array.make n (-1) in
  let rec aux u =
    for v = 0 to n - 1 do
      if g.(u).(v) > 0 && pere.(v) = -1
      then (pere.(v) <- u;
            aux v)
    done in
  aux 0;
  pere;;
```

- Quelle est la complexité dans le pire des cas de `ford g dfs`, en fonction du nombre de sommets $|V|$ et de la valeur maximum $|f^*|$ d'un flot?
 ► `dfs` est en complexité $O(|V|^2)$ (un appel récursif au plus par sommet, effectuant $|V|$ itérations de boucle `for`).
 D'après 2.5, `ford g dfs` a complexité $O(|V|^2|f^*|)$.

On peut améliorer l'algorithme précédent en essayant de trouver des chemins qui augmentent de beaucoup le flot.

- Écrire une fonction `cmax : int array array -> int` telle que `cmax m` renvoie la valeur maximum d'une case de la matrice m .

```
let cmax g =
  let res = ref g.(0).(0) in
  let n = Array.length g in
  for u = 0 to n - 1 do
    for v = 0 to n - 1 do
      res := max !res g.(u).(v)
    done;
  done; !res;;
```

- Comment modifier `dfs` en une fonction `kdfs` telle que `kdfs g k` permette de trouver un chemin (s'il existe) de 0 à 1 n'utilisant que des arcs (u, v) tels que $c_f(u, v) \geq k$?
 ► Il suffit de ne s'appliquer récursivement que si l'arc est de capacité résiduelle supérieure à k :

```
let kdfs g k =
  let n = Array.length g in
  let pere = Array.make n (-1) in
  let rec aux p u =
    if pere.(u) = -1 then (pere.(u) <- p;
                          for v = 0 to n - 1 do
                            if g.(u).(v) >= k then aux u v
                          done) in
  aux 0 0;
  pere;;
```

- Écrire alors une fonction `kford : int array array -> int` renvoyant la valeur maximum d'un flot dans un graphe, en utilisant la version modifiée suivante de l'algorithme de Ford-Fulkerson ($k/2$ renvoyant, comme en Caml, la division euclidienne de k par 2):

Algorithme de Ford-Fulkerson modifié

Initialement: $\vec{G}_f = \vec{G}$
 $k \leftarrow$ capacité maximum d'un arc de \vec{G}
 Tant que $k > 0$:
 Tant qu'il existe un chemin \vec{P} de 0 à 1 dans \vec{G}_f tel que $c_f(\vec{P}) \geq k$:
 Modifier \vec{G}_f en $\vec{G}_{f_{\vec{P}}}$
 $k \leftarrow k/2$

```
let rec kford k g =
  if k = 0 then 0
  else let pere = kdfs g k in
        if pere.(1) = -1 then kford (k/2) g
        else let cm = augment g pere in
              cm + kford k g;;
```

Parcours en largeur

6. Indiquer par quelles instructions Caml remplacer les deux commentaires de la fonction `bfs` suivante de façon à ce que, si `g` est un graphe résiduel représenté par matrice d'adjacence, `bfs g` renvoie le tableau `pere` des prédécesseurs d'un parcours en largeur depuis le sommet 0 :

```
let bfs g =
  let n = Array.length g in
  let pere = Array.make n (-1) in
  let f = file_new () in
  (* initialiser f et pere *)
  while not f.is_empty () do
    let u = f.take () in
    (* à compléter *)
  done;
  pere;;
```

►

```
let bfs g =
  let n = Array.length g in
  let pere = Array.make n (-1) in
  let f = file_new () in
  f.add 0;
  pere.(0) <- 0;
  while not f.is_empty () do
    let u = f.take () in
    for v = 0 to n - 1 do
      if g.(u).(v) > 0 && pere.(v) = -1 then
        (pere.(v) <- u;
         f.add v)
    done;
  done;
  pere;;
```

(Ne pas passer trop de temps sur les trois questions suivantes: elles sont difficiles.)

On note $d_f(v)$ le nombre minimum d'arcs d'un chemin du sommet 0 au sommet v dans un graphe résiduel \vec{G}_f . On suppose que les chemins \vec{P} de l'algorithme de Ford-Fulkerson sont obtenus par parcours en largeur.

- Soit \vec{P} un chemin de 0 à 1 trouvé par parcours en largeur dans un graphe résiduel \vec{G}_f . Soit $\vec{G}_{f_{\vec{P}}}$ obtenu depuis \vec{G}_f comme à la question II.1. Soit $v \in V$. Montrer que $d_f(v) \leq d_{f_{\vec{P}}}(v)$.
- Supposons qu'un arc (u, v) disparaisse de \vec{G}_f à un moment de l'algorithme de Ford-Fulkerson ((u, v) est dans \vec{G}_f mais pas dans $\vec{G}_{f_{\vec{P}}}$) puis réapparaisse plus tard dans un certain $\vec{G}_{f'}$ ($(u, v) \notin \vec{G}_{f'}$ mais $(u, v) \in \vec{G}_{f'}$). Montrer que $d_{f'}(u) \geq d_f(u) + 2$.
- En déduire que l'algorithme de Ford-Fulkerson termine après au plus $|V||\vec{E}|$ itérations de la boucle « Tant que ».

Plus large chemin (widest path)

Si \vec{P} est un chemin de 0 à 1 dans un graphe résiduel \vec{G}_f , on appelle $c_f(\vec{P})$ la **largeur** de \vec{P} . On cherche dans cette sous-partie un **plus large chemin** (de largeur maximum).

10. On rappelle l'algorithme de Dijkstra pour trouver les distances depuis un sommet r dans un graphe pondéré par w (à la fin de l'algorithme, $\text{dist.}(v)$ contient la distance de r à v):

Algorithme de Dijkstra

```
Initialement: next contient tous les sommets
                 $\text{dist.}(r) \leftarrow 0$  et  $\text{dist.}(v) \leftarrow \infty, \forall v \neq r$ 
Tant que next  $\neq \emptyset$ :
    Extraire  $u$  de next tel que  $\text{dist.}(u)$  soit minimum
    Pour tout voisin  $v$  de  $u$ :
         $\text{dist.}(v) \leftarrow \min \text{dist.}(v) (\text{dist.}(u) + w(u, v))$ 
```

En s'inspirant de cet algorithme, écrire en pseudo-code un algorithme permettant de trouver la largeur d'un plus large chemin de 0 à 1 dans un graphe résiduel \vec{G}_f . Prouver que votre algorithme est correct. **Remarque : utiliser un arbre couvrant de poids maximum (Kruskal/Prim) serait préférable...**

►

Algorithme de plus large chemin

```
Initialement: next contient tous les sommets
                 $\text{dist.}(0) \leftarrow \infty$  et  $\text{dist.}(v) \leftarrow -\infty, \forall v \neq 0$ 
Tant que next  $\neq \emptyset$ :
    Extraire  $u$  de next tel que  $\text{dist.}(u)$  soit maximum
    Pour tout voisin  $v$  de  $u$ :
         $\text{dist.}(v) \leftarrow \max \text{dist.}(v) (\min \text{dist.}(u) w(u, v))$ 
```

Soit C un plus large chemin de 0 à $v \neq 0$. Soit u le prédécesseur de v dans C et C' le sous-chemin de C de 0 à u . Alors la largeur de C est égale soit à la largeur de C' , soit à $w(u, v)$ (et en fait au maximum des deux).

11. Proposez une structure de donnée pertinente pour implémenter **next** dans la question précédente. Quelle serait alors la complexité de votre algorithme?
- On peut utiliser un tableau de booléen, comme dans le cours, ce qui donnerait une complexité $O(|V|^2)$ ($O(|V|)$ pour chaque extraction de minimum).

Une autre façon de trouver un plus large chemin est de réutiliser la fonction `kdfs` de la question IV.4 en cherchant par dichotomie le k maximum tel que `kdfs g k` trouve un chemin de 0 à 1 dans g :

12. Écrire une fonction `maxkdfs : int array array -> int array` telle que, si g représente un graphe résiduel, `maxkdfs g` renvoie le tableau des prédécesseurs d'un plus large chemin de 0 à 1 dans g , en utilisant une méthode par dichotomie.
- On appelle `maxkdfs g 0 (cmax g)` pour avoir la fonction souhaitée:

```
(* renvoie le k maximum avec  $k_1 \leq k < k_2$  *)
let rec maxkdfs g k1 k2 =
  let pere = kdfs g m in
  if k2 = k1 + 1 then pere
  else let m = (k1 + k2)/2 in (*  $k_1 < m < k_2$  *)
       if pere.(1) = -1 then maxkdfs g k1 m
       else maxkdfs g m k2;;
```

13. Quelle est la complexité de `maxkdfs g`?

►

- `cmax g` est en $O(|V|^2)$
- `maxkdfs g 0 (cmax g)` effectue $O(\log_2(\text{cmax } g))$ appels (voir démo de la complexité de la recherche dichotomique, déjà faite) à `kdfs g m` qui est en $O(|V|^2)$

D'où une complexité totale $O(|V|^2 \log(\text{cmax } g))$.

V Applications

Connectivité

On dit que des chemins sont **disjoints** s'ils n'ont pas d'arc en commun. Soit $\vec{G}' = (V', \vec{E}')$ un graphe orienté (sans capacité) et u, v deux sommets de \vec{G}' . La **connectivité** de u à v est le nombre maximum de chemins disjoints de u à v , dans \vec{G}' .

1. Expliquer comment utiliser l'algorithme de Ford-Fulkerson du II pour déterminer la connectivité de u à v .
 - On utilise u à la place de 0, v à la place de 1, et on met une capacité 1 sur chaque arête. La valeur maximum d'un flot est alors la connectivité de u à v .

La **connectivité** de \vec{G}' est la connectivité minimum d'un sommet quelconque de \vec{G}' à un autre.

2. Montrer que \vec{G}' a un sommet s de degré sortant inférieur ou égal à $\frac{|\vec{E}'|}{|V'|}$.
 - $\frac{|\vec{E}'|}{|V'|}$ est le degré sortant moyen des sommets de \vec{G}' : il existe un sommet dont le degré sortant est supérieur ou égal à la moyenne.
3. En utilisant la question III.3, en déduire que la connectivité de \vec{G}' peut être déterminée en $O(\frac{|\vec{E}'|}{|V'|}C)$, où C est la complexité de la méthode utilisée par Ford-Fulkerson pour trouver un chemin.
 - Soient u et v des sommets tels que la connectivité c de \vec{G}' soit celle de u à v . D'après III.3, c'est aussi la capacité minimum d'une coupe S séparant u et v . Si $s \in S$ alors une utilisation de l'algorithme de Ford-Fulkerson avec $0 = s$ et $1 = v$ donnera c . Si $s \notin S$ alors une utilisation de l'algorithme de Ford-Fulkerson avec $0 = u$ et $1 = s$ donnera c . Il suffit donc d'appliquer Ford-Fulkerson avec $0 = s$ et pour tout autre sommet 1, puis appliquer Ford-Fulkerson avec $1 = s$ et pour tout autre sommet 0. Le maximum des valeurs trouvées est alors c . De plus chaque utilisation de Ford-Fulkerson effectue $O(\frac{|\vec{E}'|}{|V'|})$ recherches de chemin car chaque arc sortant de s ne peut être utilisé qu'une seule fois. D'où le résultat.

Couplage

Un **couplage** dans un graphe non-orienté est un ensemble M d'arêtes dont toutes les extrémités sont différentes (si $e_1 \in M$ et $e_2 \in M$ alors $e_1 \cap e_2 = \emptyset$). Un couplage avec un nombre maximum d'arêtes est un **couplage maximum**.

4. Soit $G = (V, E)$ un graphe non-orienté **biparti**, c'est à dire que $V = A \sqcup B$ et toute arête de G a une extrémité dans A et une dans B . Expliquer comment on pourrait utiliser l'algorithme de Ford-Fulkerson pour trouver un couplage maximum dans G .
Indice: on pourra orienter les arêtes de G , rajouter un sommet 0 et un sommet 1 à G (on suppose $0 \notin V$, $1 \notin V$) et donner une capacité à chaque arc.