CORRIGÉ COMPLET

Ce corrigé concerne l'union des deux sujets.

I Introduction aux graphes de flot

▶ **Question 1** Si $f(u,v) > 0$, la condition **respect de la capacité** impose $0 < f(u,v) \leqslant c(u,v)$. Comme $c(u,v) = 0$ si $(u,v) \notin A$, on a donc bien $(u,v) \in A$. D'autre part, $f(u,v) < 0$ donne $f(v,u) > 0$ par **antisymétrie**, donc $(v,u) \in A$.

▶ **Question 2** On a, pour $u \in S \setminus \{s,t\}$:

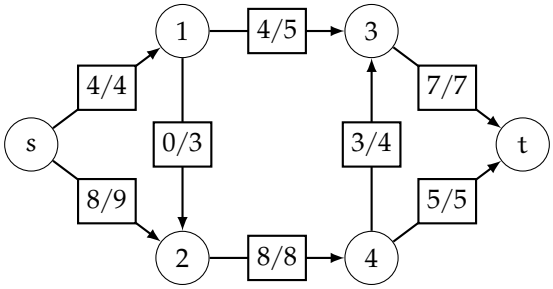
$$\begin{aligned}
\varphi(u) &= \varphi_+(u) - \varphi_-(u) \\
&= \sum_{\substack{v \in S \\ f(u,v)>0}} f(u,v) - \sum_{\substack{v \in S \\ f(u,v)<0}} f(v,u) \\
&= \sum_{\substack{v \in S \\ f(u,v)>0}} f(u,v) + \sum_{\substack{v \in S \\ f(u,v)<0}} f(u,v) && \textbf{antisymétrie} \\
&= \sum_{v \in S} f(u,v) \\
&= 0 && \textbf{conservation}
\end{aligned}$$

▶ **Question 3** Comme s est une source, il est clair que $\varphi_-(s) = 0$. On en déduit que $|f| = \varphi_+(s) = \varphi(s)$. Par ailleurs, on a $\sum_{u \in S} \varphi(u) = \varphi(s) + \varphi(t)$ d'après la question précédente, et

$$\begin{aligned}
\sum_{u \in S} \varphi(u) &= \sum_{u \in S} \varphi_+(u) - \sum_{u \in S} \varphi_-(s) \\
&= \sum_{\substack{(u,v) \in S^2 \\ f(u,v)>0}} f(u,v) - \sum_{\substack{(u,v) \in S^2 \\ f(v,u)>0}} f(v,u) \\
&= 0
\end{aligned}$$

On en déduit $\varphi(t) = -\varphi(s) = -|f|$.

▶ **Question 4** On propose le flot suivant :

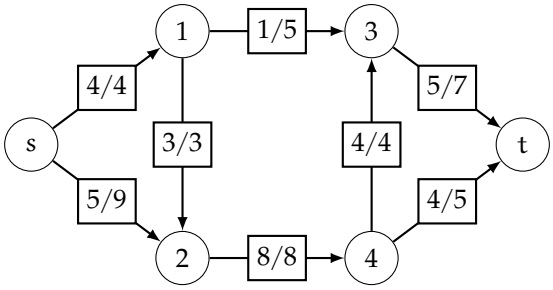


Il s'agit bien d'un flot maximal car $|f| = -\varphi(t) = \sum_{u \in S} c(u,t) = 12$. Un flot de débit strictement supérieur entraînerait un non-respect de la capacité sur une des arêtes entrantes de t.

II Algorithme de Ford-Fulkerson

▶ **Question 5** On peut effectuer un parcours de graphe depuis s en ne s'autorisant que les arêtes non saturées, en stockant l'arbre de parcours (sous forme par exemple d'un tableau $parent$) pour pouvoir reconstruire un chemin à la fin. Si l'on trouve un chemin menant à t, il s'agira bien d'un chemin non saturé ; dans le cas contraire, il n'y a pas de chemin non saturé. Tout ceci peut se faire en temps $O(|S| + |A|)$ pour le parcours et $O(|S|)$ pour la reconstruction, donc $O(|S| + |A|)$ au total.

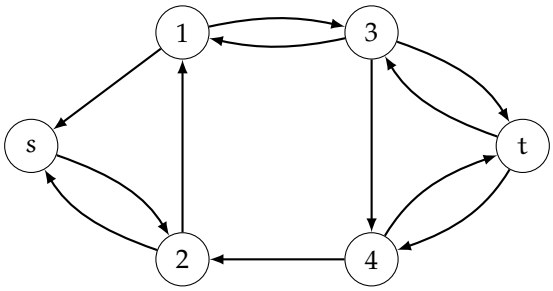
▶ **Question 6** Le seul chemin non saturé est $(s, 1, 3, t)$: on le sature, et l'on obtient le flot suivant :



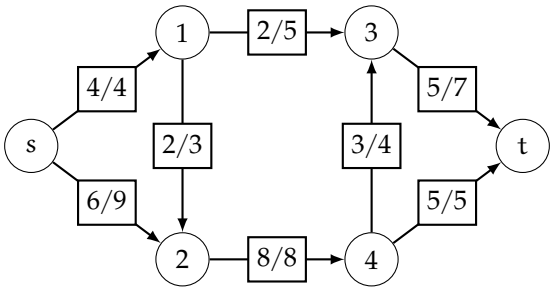
Avec ce flot, tous les chemins de s à t sont saturés, donc l'algorithme termine. Ce flot n'est clairement pas maximal car son débit est égal à 9 (et on a déjà trouvé un flot de débit 12).

▶ **Question 7** Pour que $(u, v) \in A_f$, il est nécessaire (mais non suffisant !) que $c(u, v) > 0$ ou $f(u, v) < 0$. La première condition implique $(u, v) \in A$ et, d'après la question 1, la deuxième implique $(v, u) \in A$.

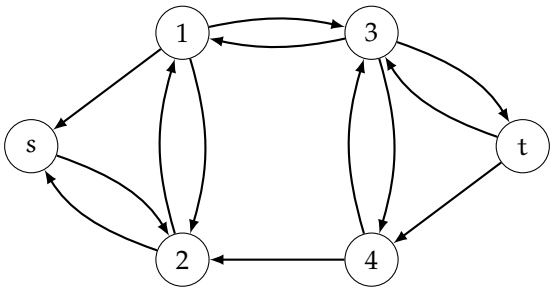
▶ **Question 8** On obtient le graphe résiduel suivant :



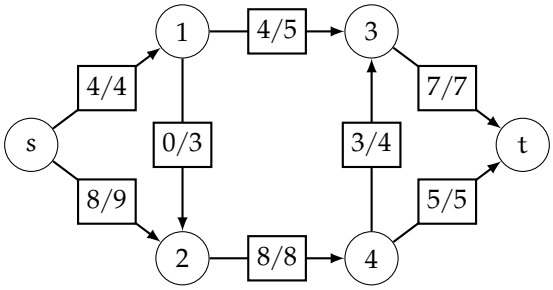
▶ **Question 9** Dans le graphe précédent, il existe deux chemins améliorants : $(s, 2, 1, 3, t)$ et $(s, 2, 1, 3, 4, t)$. Si on sature ce dernier, on obtient le flot :



Le graphe résiduel est alors :



Il ne reste qu'un seul chemin améliorant, $(s, 2, 1, 3, t)$. En saturant ce chemin, on obtient :



Il s'agit bien d'un flot maximal.

III Programmation

III.1 Préliminaires

▶ Question 10

```c
int **zeroes(int n){
    int **mat = malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        mat[i] = malloc(n * sizeof(int));
        for (int j = 0; j < n; j++) {
            mat[i][j] = 0;
        }
    }
    return mat;
}
```

▶ Question 11

```c
void free_matrix(int **mat, int n){
    for (int i = 0; i < n; i++) free(mat[i]);
    free(mat);
}
```

III.2 Structure de file

▶ Question 12

```c
cell *cell_new(vertex v){
    cell *c = malloc(sizeof(cell));
    c->data = v;
    c->next = NULL;
    return c;
}
```

▶ Question 13

```c
queue *queue_create(void){
    queue *q = malloc(sizeof(queue));
    q->len = 0;
    q->left = NULL;
    q->right = NULL;
    return q;
}
```

▶ **Question 14**

```c
int queue_length(queue *q){
    return q->len;
}
```

▶ **Question 15**

```c
void queue_push(queue *q, vertex v){
    cell *c = cell_new(v);
    if (q->len == 0) {
        q->right = c;
        q->left = c;
    } else {
        q->right->next = c;
        q->right = c;
    }
    q->len++;
}
```

▶ **Question 16**

```c
vertex queue_pop(queue *q){
    assert(q->len > 0);
    vertex result = q->left->data;
    if (q->len == 1) {
        free(q->left);
        q->left = NULL;
        q->right = NULL;
    } else {
        cell *tmp = q->left->next;
        free(q->left);
        q->left = tmp;
    }
    q->len--;
    return result;
}
```

▶ **Question 17**

```c
void cell_free(cell *c){
    if (c != NULL) {
        cell_free(c->next);
        free(c);
    }
}
```

▶ **Question 18**

```c
void queue_free(queue *q){
    cell_free(q->left);
    free(q);
}
```

▶ **Question 19** Insérer à gauche ne pose aucun problème. En revanche, pour extraire à droite, il faudrait modifier `right` pour qu'il pointe vers l'avant-dernière cellule. Le problème est qu'on ne dispose pas de pointeur vers cette cellule, ni de moyen efficace d'en récupérer un : il va falloir parcourir toute la liste à partir de la gauche, pour un coût en $O(n)$. Il faudrait une liste doublement chaînée pour faire cette opération efficacement.

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

▶ **Question 20**

- Il est inutile de construire explicitement le graphe résiduel : un arc de ce graphe correspond soit à un arc (u, v) du graphe initial, soit à un arc « renversé » (dont on nous a garanti qu'ils étaient tous présents).
- On peut aussi se passer du tableau de booléens usuel pour marquer les sommets visités : le tableau `parent` peut remplir ce rôle.
- En dehors de cela, c'est un parcours en largeur standard avec une file (qu'on n'oubliera pas de libérer à la fin de la fonction).

```
vertex *bfs_residual(flow_graph *g, int **f){
    queue *q = queue_create();
    queue_push(q, g->s);
    vertex *parent = malloc(g->n * sizeof(vertex));
    for (vertex i = 0; i < g->n; i++) parent[i] = -1;
    parent[g->s] = g->s;
    while (queue_length(q) > 0) {
        vertex v = queue_pop(q);
        for (int i = 0; i < g->degrees[v]; i++) {
            vertex w = g->adj[v][i];
            if (parent[w] != -1 || g->capacity[v][w] - f[v][w] <= 0) continue;
            parent[w] = v;
            queue_push(q, w);
        }
    }
    queue_free(q);
    return parent;
}
```

▶ **Question 21** On remonte dans l'arbre codé par le tableau `parent` ; `t` est accessible si et seulement si `parent[v]` n'est pas égal à -1.

```
int path_capacity(flow_graph *g, int **f, vertex *parent){
    int free_capacity = INT_MAX;
    int v = g->t;
    if (parent[v] == -1) return 0;
    while (v != g->s) {
        vertex p = parent[v];
        free_capacity = min(free_capacity, g->capacity[p][v] - f[p][v]);
        v = p;
    }
    return free_capacity;
}
```

▶ **Question 22** Très similaire à la question précédente.

```
bool saturate_path(flow_graph *g, int **f, vertex *parent){
    int free_capacity = path_capacity(g, f, parent);
    if (free_capacity == 0) return false;
    int v = g->t;
    while (v != g->s) {
        vertex p = parent[v];
        f[p][v] += free_capacity;
        f[v][p] -= free_capacity;
        v = p;
    }
    return true;
}
```

▸ **Question 23**

```
bool step(flow_graph *g, int **f){
    int *parent = bfs_residual(g, f);
    bool result = saturate_path(g, f, parent);
    free(parent);
    return result;
}
```

▸ **Question 24**

- Dans l'appel à `bfs_residual`, chaque sommet est inséré dans la file au plus une fois, et donc extrait au plus une fois. Or le coût du traitement lors de cette extraction est proportionnel au degré sortant du sommet dans le graphe résiduel, qui est au plus le double de son degré sortant dans le graphe initial. Au total, la boucle **while** s'exécute donc en temps $O\left(\sum_{v\in S} d^+v\right) = O(|A|)$. En ajoutant la création du tableau `parent`, on obtient du $O(|S|+|A|)$.
- L'appel à `saturate_path` fait parcourir deux fois un chemin de longueur au plus $|S|$ avec des opérations en temps constant sur chaque sommet rencontré. On a donc du $O(|S|)$.
- Finalement, `step` s'exécute en $O(|A|+|S|)$.

▸ **Question 25** Il suffit d'appeler la fonction `step` jusqu'à ce qu'elle renvoie `false`.

```
int **ford_fulkerson(flow_graph *g){
    int **f = zeroes(g->n);
    while (step(g, f)) {}
    return f;
}
```

IV Complexité et terminaison

▸ **Question 26** On remarque que la saturation d'un chemin améliorant augmente le débit du flot d'une valeur entière strictement positive. Le nombre de saturations effectuées est donc majoré par M. Sachant que chaque appel à `step` (et donc chaque tour de boucle) a une complexité en $O(|S|+|A|)$, on en déduit la terminaison et une complexité totale en $O(|S|^2 + M(|S|+|A|))$ (le $O(|S|^2)$ venant de la création de la matrice `f`, et pouvant d'ailleurs être évité en la remplaçant par un dictionnaire).

V Correction de l'algorithme

▸ **Question 27** On a $C(X) = c(s,2) + c(1,2) + c(3,t) = 15$.

▸ **Question 28** D'une part, d'après les questions **??** et **??**, on a $|f| = \varphi(s) = \sum_{u\in X} \varphi(u)$. D'autre part :

$$\begin{aligned}
\sum_{u\in X} \varphi(u) &= \sum_{u\in X} \left(\varphi_+(u) - \varphi_-(u)\right) \\
&= \sum_{u\in X} \left(\sum_{\substack{v\in X \\ f(u,v)>0}} f(u,v) + \sum_{\substack{v\in \overline{X} \\ f(u,v)>0}} f(u,v) - \sum_{\substack{v\in X \\ f(u,v)<0}} f(v,u) - \sum_{\substack{v\in \overline{X} \\ f(u,v)<0}} f(v,u) \right) \\
&= \sum_{u\in X} \left(\sum_{\substack{v\in \overline{X} \\ f(u,v)>0}} f(u,v) + \sum_{\substack{v\in \overline{X} \\ f(u,v)<0}} f(u,v) \right) + \sum_{\substack{(u,v)\in X\times X \\ f(u,v)>0}} f(u,v) - \sum_{\substack{(u,v)\in X\times X \\ f(u,v)<0}} f(v,u) \\
&= C(X) + + \sum_{\substack{(u,v)\in X\times X \\ f(u,v)>0}} f(u,v) - \sum_{\substack{(u,v)\in X\times X \\ f(v,u)>0}} f(v,u) \\
&= C(X)
\end{aligned}$$

Comme le flot respecte la capacité des arêtes, l'inégalité demandée suit.

▸ **Question 29**

(1) $\implies$ (2) S'il existait un chemin améliorant, alors on pourrait augmenter strictement la valeur du flot en le saturant.

(2) $\implies$ (3) Prenons pour X l'ensemble des sommets accessibles depuis s dans le graphe résiduel. Comme il n'y a pas de chemin améliorant, $t \notin X$ et X est donc une coupe. D'autre part, les arcs (u,v) avec $u \in X$ et $v \notin X$ ne sont pas dans G_f (sinon v serait accessible depuis s), donc pour un tel arc on a $c(u,v) = f(u,v)$. En appliquant la question précédent, on obtient alors $|f| = \sum_{(u,v)\in X\times\overline{X}} f(u,v) = \sum_{(u,v)\in X\times\overline{X}} c(u,v) = C(X)$.

(3) $\implies$ (1) Considérons un autre flot f'. D'après la question précédente, on a $|f'| \leqslant C(X)$, donc $|f'| \leqslant |f|$. Ainsi, $|f|$ est maximal.

▸ **Question 30** L'algorithme termine (d'après la question 26) et renvoie un flot ne contenant pas de chemin améliorant. L'implication (2) $\implies$ (1) montre que ce flot est bien maximal.

VI Réduction du couplage maximum

▸ **Question 31** Soit $G = (X \sqcup Y, A)$ un graphe biparti. On définit le graphe de flot $G' = (S, A', c, s, c)$ par :

- $S = X \sqcup Y \cup \{s\} \cup \{t\}$, où s et t sont deux nouveaux sommets ;
- pour $(x,y) \in X \times Y$, on a $(x,y) \in A'$ si et seulement si $\{x,y\} \in A$ (on oriente les arêtes de G de X vers Y) ;
- $(s,x) \in A'$ pour tout $x \in X$;
- $(y,t) \in A'$ pour tout $y \in Y$;
- c est constante égale à 1 sur A'.

À un flot f, on associe le couplage $C_f = \{xy \in A \mid f(x,y) = 1\}$.

- Si $x_0, y_0, x_1, y_1, \ldots, x_k, y_k$ est un chemin augmentant pour C_f, alors on a $f(x_i, y_i) = 0$ et $f(y_i, x_{i+1}) = -1$ pour chaque i (les arêtes x_iy_i ne sont pas dans C_f, les arêtes y_ix_{i+1} y sont). Comme les capacités valent 1 dans tous les cas, le chemin $s, x_0, y_0, \ldots, x_k, y_k, t$ est améliorant pour f.
- Inversement, on voit de même qu'en supprimant s et t d'un chemin améliorant pour f, on obtient un chemin augmentant pour C_f.

On en déduit par le lemme de Berge et la question 29 que f est maximal si et seulement si C_f est de cardinalité maximale.

La construction de G' se fait en temps $O(n+p)$ sans problème particulier, et comme $|f| \leqslant |S|$ (il y a $|X| \leqslant |S|$ arcs sortant de s), la complexité totale est en $O(n+p+n^2+n(n+p)) = O(n^2+np)$.

VII Algorithme de Edmonds-Karp

▸ **Question 32** Si (u,v) a disparu entre A_i et A_{i+1}, alors il a été saturé, et elle faisait donc partie du chemin augmentant. Or ce chemin est un chemin de la racine s vers la feuille t dans l'arbre de parcours en largeur, donc u est le père de v : $niveau_i(v) = niveau_i(u) + 1$.
Inversement, si (u,v) est apparu dans A_{i+1}, c'est qu'on a diminué le flux $f(u,v)$, donc augmenté $f(v,u)$. Cette fois, c'est (v,u) qui était une arête du plus court chemin, et v est le père de u : $niveau_i(u) = niveau_i(v) + 1$.

▸ **Question 33** Comme suggéré par l'énoncé, on fixe $i \geqslant 1$ et l'on prend comme hypothèse de récurrence H_k : « pour tout v tel que $niveau_i(v) \leqslant k$, on a $niveau_{i-1}(v) \leqslant niveau_i(v)$ ».

- Pour $k = 0$, c'est immédiat puisqu'alors $v = s$ et donc $niveau_{i-1}(v) = niveau_i(v) = 0$.
- Supposons donc H_k pour un certain k, et soit v tel que $niveau_i(v) = k+1$. Considérons le chemin $\gamma : s, \ldots, u, v$ reliant s à v dans l'arbre i. On a $niveau_i(u) = niveau_i(v) - 1$, donc l'hypothèse de récurrence nous donne $niveau_{i-1}(u) \leqslant niveau_i(u)$. Il suffit donc de montrer que $niveau_{i-1}(v) - 1 \leqslant niveau_{i-1}(u)$.
  - Si l'arc (u,v) est dans A_{i-1}, alors $niveau_{i-1}(v) \leqslant niveau_{i-1}(u) + 1$ puisqu'on peut ajouter cet arc à un plus court chemin pour u.
  - Sinon, on a $(u,v) \in A_i \setminus A_{i-1}$, donc la question précédente nous donne $niveau_{i-1}(u) = niveau_{i-1}(v)+1$, et donc $niveau_{i-1}(u) > niveau_{i-1}(v) - 1$.

On en déduit que si $niveau_i(v)$ est fini, alors $niveau_i(v) \geqslant niveau_{i-1}(v)$. Cela reste évidemment vrai si $niveau_i(v) = \infty$.

▶ **Question 34** Considérons deux disparitions de l'arc (u, v) : $(u, v) \in A_i \setminus A_{i+1}$ et $(u, v) \in A_j \setminus A_{j+1}$, avec $j > i$. Il existe nécessairement un k entre i et j tel que $(u, v) \in A_{k+1} \setminus A_k$ (l'arc doit réapparaître avant sa deuxième disparition). Mais alors :

- $niveau_i(v) = niveau_i(u) + 1$ (question 32) ;
- $niveau_k(u) = niveau_k(v) + 1$ (*idem*) ;
- $niveau_i(v) \leqslant niveau_k(v)$ (question 33).

On en déduit $niveau_i(u) + 1 \leqslant niveau_k(u) - 1$, et donc $niveau_j(u) \geqslant niveau_k(u) \geqslant niveau_i(u) + 2$. Ainsi, le niveau de u augmente d'au moins 2 entre deux disparitions. Au bout de $|S|/2$ disparitions au plus, on aura donc $niveau(u) \geqslant |S|$, ce qui implique $niveau(u) = \infty$ (un plus court chemin dans un graphe à n sommets est de longueur au plus $n - 1$).

▶ **Question 35** À chaque étape, on sature au moins un arc, qui disparaît du graphe résiduel. Il y a p arcs et chacun de ces arcs ne peut disparaître que $n/2$ fois, donc on a au plus $np/2$ disparitions et donc $np/2$ tours de boucle. Un tour de boucle se fait en temps $O(n + p)$, et l'initialisation du flot f en temps $O(n^2)$, donc on obtient au total du $O\left(n^2 + np(n + p)\right) = O\big(np(n + p)\big)$.