DS4 (3 heures)

Lisez tout le texte avant de commencer. La plus grande importance sera attachée à la clarté, à la précision et à la concision de la rédaction. Toute réponse non justifiée ne sera pas prise en compte. Si vous repérez ce qui vous semble être une erreur d'énoncé, signalez la sur votre copie et poursuivez votre composition en expliquant les raisons de vos éventuelles initiatives. L'usage de tout dispositif électronique est interdit.

Exercice 1

On s'intéresse aux arbres binaires de recherche (ABR) de taille n, c'est-à-dire comportant n nœuds, étiquetés par des éléments distincts et comparables.

Question 1. Selon qu'il est *équilibré* ou *filiforme*, rappeler la complexité au pire de recherche d'un élément dans un ABR.

Question 2. On souhaite construire un ABR en insérant successivement n éléments distincts et comparables. Donner un ordre d'insertion des éléments de $[\![1, 7]\!]$ associé au pire cas et un ordre associé au meilleur cas.

Afin d'éviter le pire cas, on insère les n éléments en les choisissant dans un ordre aléatoire en admettant que les $n!$ permutations possibles des éléments sont des ordres d'insertion équiprobables. On note H_n la variable aléatoire correspondant à la hauteur de l'arbre obtenu, $X_n = 2^{H_n}$, R_n le rang de la racine de l'arbre parmi les n éléments, c'est-à-dire sa position si les n éléments étaient triés, et $Z_{n,i}$ la variable aléatoire valant 1 si $R_n = i$ et 0 sinon.

Question 3. Montrer que si $R_n = i$ alors $X_n = 2\max(X_{i-1}, X_{n-i})$.

Question 4. Montrer que $X_n = 2\sum_{i=1}^{n} Z_{n,i} \times \max(X_{i-1}, X_{n-i})$.

Question 5. En observant que $Z_{n,i}$ est indépendante de X_{i-1} et X_{n-i}, montrer que $\mathbb{E}[X_n] \leqslant \frac{4}{n}\sum_{k=0}^{n-1}\mathbb{E}[X_k]$. On admet qu'une telle relation de récurrence implique que $\mathbb{E}[X_n] \leqslant \frac{1}{4}\binom{n+3}{n}$.

Question 6. L'inégalité de Jensen garantit que si f est une fonction convexe sur un intervalle I et X est une variable aléatoire sur I alors $f(\mathbb{E}[X]) \leqslant \mathbb{E}[f(X)]$. Déduire de ce qui précède que $\mathbb{E}[H_n] \leqslant O(\log n)$ et conclure.

Exercice 2

Le problème de décision Stable est défini comme suit.

Étant donné un graphe $G = (S, A)$ non orienté et $k \in \mathbb{N}$,
existe-t-il $S' \subset S$ tel que $|S'| = k$ et deux sommets de S' ne sont jamais adjacents dans G ?

Un tel sous-ensemble S' s'appelle un *stable* de taille k.

Question 1. Montrer que Stable $\in$ NP.

On cherche à présent à montrer que 3Sat $\leqslant$ Stable. Soit $\varphi = \bigwedge_{i=1}^{m} C_i$ une formule du calcul propositionnel dont les clauses C_i contiennent au plus 3 littéraux et impliquant un ensemble $V = \{v_1, \dots, v_n\}$ de variables propositionnelles. On définit le graphe non orienté $G_\varphi = (S, A)$ comme suit.

- Pour chaque occurrence de littéral dans φ, on construit un sommet de S.
- Si $s, t \in S$, on ajoute l'arête $\{s, t\}$ à A si et seulement si (s et t sont des sommets représentant deux littéraux d'une même clause) ou (s et t sont des sommets étiquetés par deux littéraux qui sont la négation l'un de l'autre).

Question 2. Montrer que si G_φ possède un stable de taille m alors φ est satisfiable par une valuation qu'on explicitera. La réciproque est-elle vraie ?

Question 3. Montrer que Stable est NP-complet.

Question 4. Le problème Clique est défini comme suit.

Étant donné un graphe $G = (S, A)$ non orienté et $k \in \mathbb{N}$, existe-t-il un sous-graphe complet de G ayant k sommets ?

Montrer que Clique est NP-complet.

Exercice 3

Le langage de programmation est OCaml. On autorise toutes les fonctions des modules Array et List, ainsi que les fonctions de la bibliothèque standard (celles qui s'écrivent sans nom de module, comme max, incr ainsi que les opérateurs comme ( )). Sauf précision de l'énoncé, l'utilisation d'autres modules est interdite.

Préliminaires

Un *graphe de flot* est une structure permettant de modéliser tout type de flux : réseau de transport routier, circulation de fluides dans des canalisations, distribution d'électricité... Sa définition s'apparente à celle d'un graphe orienté pondéré où les poids sont considérés comme des capacités de débit. On appelle *graphe de flot* un quintuplet $G = (S, A, c, s, t)$ où :

- (S, A) est un graphe orienté sans boucle, c'est-à-dire sans arête de la forme (u, u) ;
- $c : A \to \mathbb{R}_+^*$ est une fonction dite de *capacité* ;
- il existe deux sommets particuliers : $s \in S$ est une *source* de (S, A), sommet sans arête entrante, et $t \in S$ est un *puits* de (S, A), sommet sans arête sortante ;
- s'il existe une arête $(u, v) \in A$ alors $(v, u) \notin A$: il n'existe pas de cycle de taille 2 .

La figure 1 représente un graphe de flot G_0.

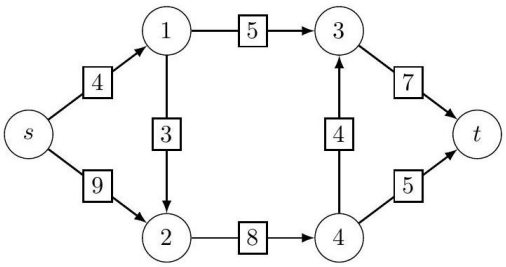

Figure 1 - Le graphe de flot G_0.

Si $G = (S, A; c, s, t)$ est un graphe de flot, on appelle *flot de* G une fonction $\phi : A \to \mathbb{R}^+$vérifiant :

- respect de la capacité : pour $(u, v) \in S^2, \phi(u, v) \leqslant c(u, v)$;
- conservation du flot : pour $u \in S \setminus \{s, t\}$, $\sum_{(v,u)\in A} \phi(u, v) - \sum_{(u,v)\in A} \phi(u, v) = 0$.

La figure 2 représente le graphe de flot G_0 et un flot ϕ_0 compatible avec G. Sur chaque arête (u, v), on mentionne l'information $\phi_0(u, v)/c(u, v)$.

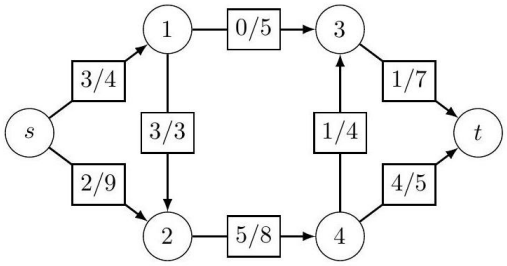

Figure 2 - Le graphe de flot G_0 et un flot ϕ_0.

On appelle *débit* de ϕ la valeur $|\phi| = \sum_{(s,u)\in A} \phi(s, u)$, c'est-à-dire la somme des flots sortant de s.

Flots

Question 1. Montrer qu'il n'existe pas de flot de G_0 de débit 13.

Pour la suite du sujet, on considère $G = (S, A, c, s, t)$ un graphe de flot et ϕ un flot de G. Si $u \in S$, on appelle :

- *flux entrant* de u la valeur $\phi_-(u) = \sum_{(v,u)\in A} \phi(v, u)$;
- *flux sortant* de u la valeur $\phi_+(u) = \sum_{(u,v)\in A} \phi(u, v)$;
- *flux net* de u la valeur $\Delta\phi(u) = \phi_+(u) - \phi_-(u)$.

Ainsi, la condition de conservation du flot ϕ est équivalente à la propriété suivante : pour tout $u \in S \setminus \{s, t\}$, $\Delta\phi(u) = 0$.

Question 2. Montrer que $|\phi| = \Delta\phi(s)$ puis que $\Delta\phi(s) = -\Delta\phi(t)$.

Un graphe orienté (S, A) est représenté par tableau de listes d'adjacence de type `graphe` L'ensemble des capacités associées à un graphe de flot est représenté par une matrice d'adjacence de type `capa`.

```
type graphe = int list array;;
type capa = float array array;;
```

Ainsi, si $G = (S, A, c, s, t)$ est représenté par un objet `g` de type `graphe` et un objet `c` de type `capa`, alors :

- $S = [\![0, n-1]\!]$ où $n =$ `Array.length g` est la taille de `g` ;
- par convention, $s = 0$ et $t = n - 1$ sont la source et le puits ;
- pour $u \in S$, `g.(u)` contient la liste des voisins de u ;
- c est une matrice de dimensions $n \times n$;
- pour $(u, v) \in S^2$, `c.(u).(v)` $= c(u, v) > 0$ si $(u, v) \in A$ et `c.(u).(v)` $= 0$ sinon.

Par exemple, le graphe de flot G_0 de la figure 1 peut être représenté par les objets :

```
let g = [| [1; 2]; [2; 3]; [4]; [5]; [3; 5]; [] |];;

let c0 = [| [|0.; 4.; 9.; 0.; 0.; 0.|];
            [|0.; 0.; 3.; 5.; 0.; 0.|];
            [|0.; 0.; 0.; 0.; 8.; 0.|];
            [|0.; 0.; 0.; 0.; 0.; 7.|];
            [|0.; 0.; 0.; 4.; 0.; 5.|];
            [|0.; 0.; 0.; 0.; 0.; 0.|] |];;
```

Un flot est représenté par une matrice de flottants de type `flot`.

```
type flot = float array array; ;
```

Le flot ϕ_0 de la figure 2 peut être représenté par :

```
let phi0 = [| [|0.; 3.; 2.; 0.; 0.; 0.|];
              [|0.; 0.; 3.; 0.; 0.; 0.|];
              [|0.; 0.; 0.; 0.; 5.; 0.|];
              [|0.; 0.; 0.; 0.; 0.; 1.|];
              [|0.; 0.; 0.; 1.; 0.; 4.|];
              [|0.; 0.; 0.; 0.; 0.; 0.|] |];;
```

Pour toute la suite, on suppose que lorsqu'une fonction prend en argument plusieurs tableaux, ces tableaux ont la même dimension et on ne demande pas de le vérifier.

Question 3. Écrire une fonction `delta_phi (g: graphe) (phi: flot) : float array` qui calcule un tableau `dphi` de taille $n = |S|$ tel que pour $u \in S$, `dphi.(u)` $= \Delta\phi(u)$. On attend une complexité linéaire en $|S| + |A|$.

Question 4. En déduire une fonction `conservation(g : graphe) (phi: flot) : bool` qui renvoie `true` si et seulement si le flot `phi` respecte la règle de conservation du flot dans le graphe de flot représenté par `g`.
Cette règle ne dépend pas des capacités, donc on ne met pas `c` en argument.

Calcul de flot maximal

On considère le problème d'optimisation FlotMaximal suivant.

- **Instance :** un graphe de flot $G = (S, A, c, s, t)$.
- **Solution :** un flot ϕ de G.
- **Optimisation :** maximiser $|\phi|$.

Question 5. Représenter graphiquement une solution à FlotMaximal sur le graphe G_0 de la figure 1. Justifier qu'il s'agit d'une solution maximale.

Saturation des chemins

On dit qu'une arête $(u, v) \in A$ est *saturée* si $\phi(u, v) = c(u, v)$. On dit qu'un chemin de s à t est *saturé* s'il contient une arête saturée. Le flot ϕ est dit *saturé* si tous les chemins de s à t sont saturés.

Pour un chemin σ de s à t, la *capacité restante* de ce chemin, notée $c_\phi(\sigma)$, est le minimum des valeurs $c(u, v) - \phi(u, v)$ pour (u, v) une arête de ce chemin. Ainsi, un chemin saturé est un chemin de capacité restante nulle. On remarque que pour $u \neq v \in S^2$, (u, v) étant un chemin de longueur 1, la quantité $c_\phi(u, v)$ est bien définie et égale à $c(u, v) - \phi(u, v)$.

Si un chemin σ de s à t n'est pas saturé, on définit l'action de *saturation du chemin σ pour ϕ* comme une modification de ϕ qui consiste, pour chaque arête (u, v) du chemin, à augmenter $\phi(u, v)$ de $c_\phi(\sigma)$. 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 ϕ_0. On a augmenté le flot de 3 le long du chemin. Les arêtes saturées sont $(2, 4)$ et $(4, 3)$.

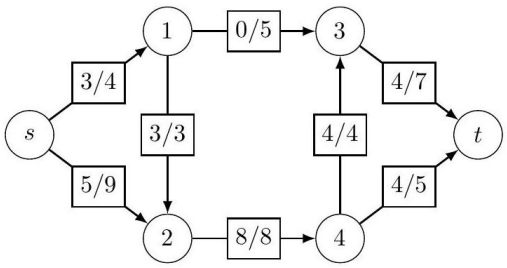

Figure 3 - Le graphe de flot G_0 et le flot ϕ_1 après saturation du chemin $(s, 2, 4, 3, t)$ pour le flot ϕ_0.

On considère l'algorithme glouton suivant.

Algorithme 1 : algorithme glouton

Entrée : graphe de flot $G = (S, A, c, s, t)$
Sortie : un flot ϕ

```
pour tout (u, v) ∈ A, φ(u, v) ← 0
tant que il existe un chemin σ non saturé de s à t faire
    saturer σ pour φ
renvoyer φ
```

Question 6. Terminer l'exécution de l'algorithme sur le graphe G_0 à partir du flot ϕ_1 de la figure 3 en détaillant les chemins non saturés trouvés et leur capacité restante. Représenter graphiquement le résultat.

Algorithme de Ford-Fulkerson

L'algorithme glouton ne renvoie pas toujours un flot maximal. Pour corriger le problème, il faut envisager de pouvoir faire « refluer » le flot en arrière dans une arête.

Pour $G = (S, A, s, t, c)$ un graphe de flot et ϕ un flot de G, on définit le *graphe résiduel* $G_\phi = (S, A_\phi, r_\phi)$ comme un graphe pondéré orienté tel que $(u, v) \in A_\phi$ si et seulement si l'une des deux conditions est vérifiée :

- $(u, v) \in A$ et $\phi(u, v) < c(u, v)$, auquel cas on pose $r_\phi(u, v) = c_\phi(u, v) = c(u, v) - \phi(u, v)$;
- $(v, u) \in A$ et $\phi(v, u) > 0$, auquel cas on pose $r_\phi(u, v) = \phi(v, u)$.

Le premier type d'arêtes correspond aux arêtes de G non saturées auxquelles on associe une nouvelle capacité correspondant à la capacité restante. Le deuxième type d'arêtes correspond aux arêtes renvoyées de G dont le flot traversant est strictement positif ; la nouvelle capacité correspond à la quantité de flot qu'on peut faire refluer.

Attention, le graphe G_ϕ peut contenir des arêtes n'existant pas dans G. Par ailleurs, ce n'est *a priori* pas un graphe de flot car il n'a pas forcément de source ou de puits.

Question 7. Représenter graphiquement le graphe résiduel de G_0 avec le flot ϕ_1 de la figure 3. On représentera les valeurs de r_ϕ comme des capacités d'un graphe de flot.

Question 8. Écrire une fonction `residuel (g: graphe) (c: capa) (phi: flot) : graphe` qui construit la partie graphe (S, A_ϕ) d'un graphe résiduel pour un flot `phi` dans un graphe de flot représenté par `g` et `c`. Quelle est sa complexité temporelle ?

Question 9. On appelle *chemin améliorant* pour ϕ un chemin de s à t dans le graphe résiduel G_ϕ. Écrire une fonction `parcours (g: graphe) : int array` qui calcule l'arborescence d'un parcours en profondeur dans un graphe orienté (S, A), en partant du sommet 0. Le résultat prendra la forme d'un tableau `parent` tel que :

- `parent.(0)` vaut 0 ;
- si u est accessible depuis 0 alors `parent.(u)` est le parent de u dans l'arborescence ;
- si u n'est pas accessible depuis 0 alors `parent.(u)` vaut -1.

On garantira une complexité $\mathcal{O}(|S| + |A|)$ qu'on justifiera brièvement.

Question 10. Écrire une fonction `chemin (parent: int array) : int list option` qui prend en argument un tableau `parent` décrivant une arborescence de parcours et qui renvoie une option de liste telle que :

- s'il n'existe pas de chemin de s (le sommet 0) à t (le sommet $n - 1$) alors la fonction renvoie `None` ;
- sinon, la fonction renvoie `Some sigma` où `sigma` est une liste décrivant un chemin σ de s à t (donc commençant par 0 et terminant par $n - 1$).

La *capacité résiduelle* d'un chemin améliorant σ, notée $r_\phi(\sigma)$, correspond au minimum des $c'(u,v)$ pour (u,v) une arête de ce chemin.

Question 11. Écrire une fonction `residu (sigma: int list) (c: capa) (phi: flot) : float` qui calcule la capacité résiduelle d'un chemin σ, étant données une liste décrivant ce chemin et des matrices décrivant la capacité et le flot.

L'action d'*amélioration de ϕ par rapport à σ* est une modification de ϕ qui consiste, pour chaque arête (u,v) du chemin, à modifier ϕ selon le principe suivant :

- si $(u,v) \in A$ et $c(u,v) \geqslant \phi(u,v) + r_\phi(\sigma)$, alors on ajoute $r_\phi(\sigma)$ à $\phi(u,v)$;
- sinon, on retire $r_\phi(\sigma)$ à $\phi(v,u)$.

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

Algorithme 2 : algorithme de Ford-Fulkerson

Entrée : graphe de flot $G = (S, A, c, s, t)$
Sortie : un flot ϕ

```
pour tout (u, v) ∈ A, φ(u, v) ← 0
tant que il existe un chemin améliorant σ pour φ faire
    améliorer φ par rapport à σ
renvoyer φ
```

Question 12. Terminer l'exécution de l'algorithme de Ford-Fulkerson sur le graphe G_0 avec le flot ϕ_1 représenté figure 3. On représentera les graphes résiduels, les flots intermédiaires et on donnera les chemins améliorants et leurs capacités résiduelles. On pourra repartir du flot obtenu à la question 6.

Question 13. Écrire une fonction `ameliorer (sigma: int list) (c: capa) (phi: flot) : unit` qui améliore un chemin supposé améliorant `sigma` pour un flot `phi`. La fonction ne devra rien renvoyer.

Question 14. En déduire une fonction `ford_fulkerson (g: graphe) (c: capa) : flot` qui renvoie un flot maximal selon l'algorithme de Ford-Fulkerson.

Question 15. Montrer que si les capacités sont entières, l'algorithme de Ford-Fulkerson termine toujours et déterminer sa complexité temporelle en fonction de $|S|$, $|A|$ et le débit du flot maximal $|\phi^*|$.

Flot maximal, coupe minimale

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 $t \in \overline{X}$. La *capacité* d'une coupe X est :

$$C(X) = \sum_{(u,v) \in A \cap (X \times \overline{X})} c(u,v)$$

Si ϕ est un flot pour G alors le *flux* d'une coupe X est :

$$\phi(X) = \sum_{(u,v) \in A \cap (X \times \overline{X})} \phi(u,v) - \sum_{(v,u) \in A \cap (\overline{X} \times X)} \phi(v,u)$$

c'est-à-dire la différence entre la somme des flots qui sortent de X et la somme des flots qui rentrent dans X.

Question 16. Dans le graphe G_0 donné figure 1, déterminer la capacité de la coupe $X = \{s, 1, 3\}$.

Question 17. Soit ϕ un flot et X une coupe de G. Montrer que $\phi(X) \leqslant C(X)$ puis que $|\phi| = \phi(X)$.
Indication : on pourra montrer que si $x \in X \setminus \{s\}$, alors $\phi(X) = \phi(X \setminus \{x\})$.

Question 18. Montrer l'équivalence entre les propriétés suivantes :

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

Indication : on pourra s'intéresser à l'ensemble des sommets accessibles depuis s dans G_ϕ.

Question 19. En déduire que s'il termine, l'algorithme de Ford-Fulkerson renvoie un flot maximal.

Résolution du problème de couplage maximum

On veut montrer qu'on peut résoudre le problème de recherche de couplage maximum dans un graphe biparti en utilisant l'algorithme de Ford-Fulkerson. On considère un graphe $G = (S, A)$ biparti et non orienté, avec $S = X \cup Y$, $X \cap Y = \varnothing$ et $A \subseteq \{\{x, y\} \mid x \in X, y \in Y\}$.

On définit le *réseau* de G comme un graphe de flot $R_G = (S', A', s, t, c)$ tel que :

- $S' = S \cup \{s, t\}$, où s et t sont deux nouveaux sommets ;
- A' contient les arêtes suivantes :
  - les (s, x) pour $x \in X$;
  - les (y, t) pour $y \in t$;
  - les (x, y) pour $\{x, y\} \in A$.
- Toutes les capacités des arêtes sont égales à 1.

Par exemple, la figure 4 représente un graphe biparti G_1 et son réseau associé.

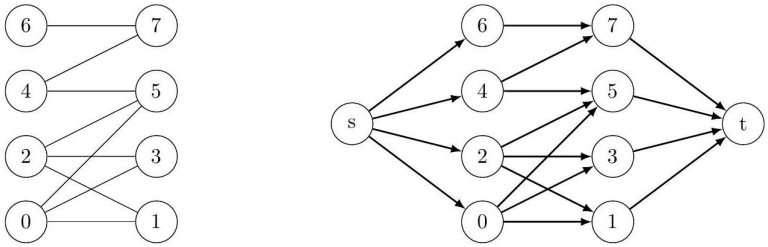

Figure 4 - Le graphe biparti G_1 et le réseau R_{G_1}. Les capacités ne sont pas représentées car égales à 1 .

Question 20. Résoudre le problème du couplage maximum en appliquant l'algorithme de Ford-Fulkerson à R_G.

Question 21. Justifier la correction et déterminer la complexité temporelle de l'algorithme précédent.

Algorithme d'Edmonds-Karp

On souhaite améliorer la complexité de l'algorithme de Ford-Fulkerson dans le pire cas, qui peut être très désavantageux si on choisit les mauvais chemins améliorants.

Question 22. Montrer que le nombre de passages dans la boucle **tant que** de l'algorithme de Ford-Fulkerson peut être égal à 2 ou à 2000 selon le choix des chemins améliorants dans le graphe de flot G_2 représenté figure 5 .

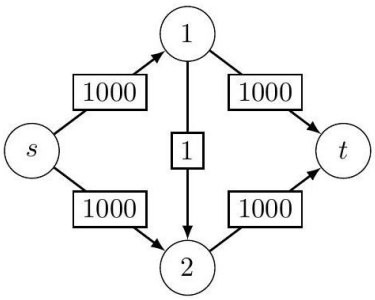

Figure 5 - Le graphe de flot G_2.

L'algorithme d'Edmonds-Karp est une variante de l'algorithme de Ford-Fulkerson qui consiste à choisir un *plus court* chemin améliorant à chaque itération.

Implémentation de file

On souhaite dans cette partie implémenter une structure de file pour implémenter un parcours en largeur. On utilise pour cela une structure de liste simplement chaînée, en gardant en mémoire le maillon de début et le maillon de fin. Pour faciliter la manipulation d'une file vide, on ajoute un maillon particulier dit *sentinelle*. La figure 6 représente schématiquement la structure, pour une file contenant les valeurs 5, 12, 7 et 3.

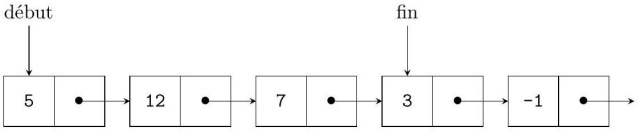

Figure 6 - Une liste simplement chaînée avec marquage du début et de la fin. Le dernier maillon est la sentinelle.

On implémente un maillon par un objet du type `maillon` tel que si `m` est de type `maillon` alors `m.valeur` représente la valeur du maillon, et `m. suivant` représente le maillon suivant (sauf pour le cas particulier de la sentinelle).

```
type maillon = {valeur : int; mutable suivant : maillon};;
```

On définit la sentinelle par :

```
let rec sentinelle = {valeur = -1; suivant = sentinelle};;
```

On ne demande pas de comprendre cette syntaxe utilisant `rec` pour autre chose qu'une fonction, et on pourra utiliser la variable `sentinelle` comme si elle était définie de manière globale.
On implémente une file par un objet du type `file` tel que si `f` est de type `file`, alors :

- si la file correspondante est vide, `f.debut` et `f.fin` sont tous les deux égaux au maillon sentinelle ;
- sinon, `f.debut` est égal au premier maillon de la liste et `f.debut` est égal au dernier maillon de la liste, c'est-à-dire celui qui précède le maillon sentinelle.

```
type file = {mutable debut : maillon; mutable fin : maillon};;
```

Pour toute la suite, on suppose que les valeurs contenues dans la file sont des entiers positifs ou nuls.

Question 23. Quelle opération est problématique en termes de complexité si on choisit d'enfiler des éléments en les rajoutant au début de la liste et de défiler en supprimant à la fin de la liste ?

Pour la suite, on enfile des éléments à la fin de la liste et on défile au début de la liste.

Question 24. Écrire une fonction `creer () : file` qui crée une file vide.

Question 25. Écrire une fonction `est_vide (f: file) : bool` qui teste si une file est vide.

Question 26. Écrire une fonction `enfiler (f : file) (x : int) : unit` qui enfile un élément dans une file. On prendra garde à bien gérer le cas où la file est initialement vide.

Question 27. Écrire une fonction `defiler(f : file) : int` qui défile et renvoie un élément dans une file. La fonction renverra une erreur si la file est déjà vide.

Question 28. Quelle est la complexité temporelle des opérations précédentes ?

Algorithme d'Edmonds-Karp

L'algorithme d'Edmonds-Karp est le suivant.

Algorithme 3 : algorithme d'Edmonds-Karp

Entrée : graphe de flot $G = (S, A, c, s, t)$
Sortie : un flot ϕ

```
pour tout (u, v) ∈ A, φ(u, v) ← 0
tant que il existe un chemin améliorant pour φ faire
    poser σ un plus court chemin améliorant
    améliorer φ par rapport à σ
renvoyer φ
```

La recherche de plus court chemin peut se faire avec l'algorithme de parcours en largeur.

Question 29. Écrire une fonction `parcours_largeur (g: graphe) : int array` qui calcule l'arborescence d'un parcours en largeur dans un graphe orienté (S, A), en partant du sommet 0. Le résultat prendra la forme d'un tableau parent comme défini à la question 9.

Dans la suite de cette partie, on cherche à déterminer la complexité de l'algorithme précédent. On note $\phi_0, \phi_1, \dots, \phi_k$ la suite des flots à chaque itération de la boucle **tant que** lors de l'exécution de l'algorithme d'Edmonds-Karp, et $\sigma_1, \dots, \sigma_k$ les plus courts chemins améliorants correspondants. Pour $i \in [\![0, k]\!]$ et $u \in S$, on note $d_i(u)$ la *distance* de s à u dans le graphe résiduel G_{ϕ_i}. On commence par montrer que la longueur des chemins σ_i ne peut qu'augmenter avec i.

Question 30. Pour cette question, on suppose qu'il existe $i \in [\![1, k-1]\!]$ et un sommet v dans le chemin σ_{i+1} tel que $d_i(v) > d_{i+1}(v)$. On pose v un sommet qui minimise $d_{i+1}(v)$ parmi ceux qui vérifient cette propriété. On définit u comme le prédécesseur de v dans σ_{i+1}. Par une disjonction de cas, montrer qu'on arrive à une contradiction.

Question 31. On note $|\sigma_i|$ la longueur d'un chemin σ_i. On appelle *arête critique* de σ_i une arête (u, v) de G_{ϕ_i} telle que $r_{\phi_i}(u, v) = r_{\phi_i}(\sigma_i)$. Montrer qu'une arête (u, v) ne peut être critique qu'au plus $|S|/2$ fois au cours de l'algorithme.

Question 32. En déduire la complexité temporelle de l'algorithme d'Edmonds-Karp en fonction de $|S|$ et $|A|$, en supposant $|A| \geqslant |S|$.