

Consignes aux candidates et candidats

On identifiera une même grandeur écrite dans deux polices de caractères différentes, en italique du point de vue mathématique (par exemple n) et en **Computer Modern** à chasse fixe du point de vue informatique (par exemple `n`).

Les candidates et candidats devront répondre aux questions de programmation en utilisant le langage **OCaml**. S'ils écrivent une fonction non demandée par l'énoncé, ils devront préciser son rôle, ainsi que sa signature (son type). Les candidates et candidats sont également invités, lorsque c'est pertinent, à décrire le fonctionnement global de leurs programmes. On autorisera 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 sera interdite.

Lorsqu'une question de programmation demande l'écriture d'une fonction, la signature de la fonction demandée est indiquée à la fin de la question. Les candidates et candidats pourront écrire une fonction dont la signature est compatible avec celle demandée. Par exemple, si l'énoncé demande l'écriture d'une fonction `'int list -> int` qui renvoie le premier élément d'une liste d'entiers, écrire une fonction `'a list -> 'a` qui renvoie le premier élément d'une liste d'éléments quelconques sera considéré comme correct.

Une annexe disponible en fin de sujet rappelle différentes fonctions en OCaml.

Définitions et notations

Pour un ensemble fini E , on note $|E|$ le cardinal de E . On note $\mathcal{P}_2(E)$ l'ensemble des paires d'éléments de E , c'est-à-dire les sous-ensembles de E de cardinal 2.

On définit un **graphe non orienté** comme un couple $G = (S, A)$ tel que S est un ensemble fini d'éléments appelés *sommets* et A un ensemble de paires d'éléments distincts de S appelées *arêtes*, c'est-à-dire $A \subset \mathcal{P}_2(S)$. S'il n'y a pas d'ambiguïté, une paire $\{s, t\}$ sera notée st . Si st est une arête du graphe, on dit que s et t sont *adjacents*. Le graphe

$$G_1 = \left(\{s_0, s_1, s_2, s_3, s_4, s_5\}, \{s_0s_2, s_0s_3, s_0s_4, s_1s_2, s_1s_3, s_2s_3, s_2s_5, s_3s_4\} \right)$$

est représenté sur la figure 1.

On appelle **sous-graphe** de $G = (S, A)$ un graphe $H = (X, B)$ tel que $X \subset S$ et $B \subset A \cap \mathcal{P}_2(X)$. Si $X \subset S$, on appelle **sous-graphe induit** par X le graphe $G[X] = (X, A \cap \mathcal{P}_2(X))$.

Pour $(s, t) \in V^2$, on appelle **chaîne de s à t** une suite de sommets distincts $(s_0, s_1, \dots, s_k)$ telle que $s_0 = s$, $s_k = t$ et pour $i \in \llbracket 1, k \rrbracket$, $s_{i-1}s_i \in A$. On appelle **cycle** de G une suite de sommets $(s_0, s_1, \dots, s_k)$ telle que $k \geq 3$, $(s_0, \dots, s_{k-1})$ est une chaîne, $s_0 = s_k$ et $s_0s_{k-1} \in A$.

Un graphe est dit **connexe** s'il existe toujours une chaîne entre deux sommets distincts. Si $G = (S, A)$, on appelle **composante connexe** de G un sous-ensemble X de S tel que $G[X]$ est connexe et pour tout $s \in S \setminus X$, $G[X \cup \{s\}]$ n'est pas connexe.

Un graphe est dit un **arbre** (à différencier des arbres enracinés) s'il est connexe et ne contient pas de cycle. On dit qu'un sous-graphe $T = (X, B)$ de $G = (S, A)$ est un **arbre couvrant** si $S = X$ et T est un arbre.

On appelle **graphe pondéré** un triplet (S, A, f) tel que (S, A) est un graphe et $f : A \rightarrow \mathbb{R}^+$ est une fonction qui à chaque arête associe un *poids* qui est un réel positif. La figure 1 contient une représentation graphique d'un graphe pondéré G_2 .

Dans un graphe pondéré $G = (S, A, f)$, le **poids** d'un sous-graphe $H = (X, B, f)$ de G est la somme des poids des arêtes qui le composent, c'est-à-dire $f(H) = \sum_{a \in B} f(a)$.



FIGURE 1 – Les graphes G_1 et G_2 .

1 Difficulté du calcul de l'arbre optimal

Dans cette partie, on présente le problème du calcul de l'arbre optimal, appelé *arbre de Steiner*, et on veut montrer que c'est un problème difficile à résoudre.

Soit $G = (S, A)$ un graphe non orienté connexe. On appelle **arbre de Steiner de sommets terminaux** X un sous-graphe de G de la forme $T = (Y, B)$ tel que T est un arbre et $X \subset Y$. Les sommets de $Y \setminus X$ sont appelés les *sommets de Steiner*. Le *problème de l'arbre de Steiner* consiste, étant donné un graphe $G = (S, A)$, un ensemble $X \subset S$ et un entier k , à déterminer s'il existe un arbre de Steiner de sommets terminaux X ayant moins de k sommets de Steiner.

Q 1. Représenter graphiquement un arbre de Steiner du graphe G_1 de la figure 1, ayant $X = \{s_1, s_4, s_5\}$ comme sommets terminaux et ayant moins de 2 sommets de Steiner. Justifier qu'il n'en existe pas ayant moins de 1 sommet de Steiner.

Pour montrer la difficulté de ce problème, on se ramène au problème de satisfaisabilité d'une formule booléenne. On rappelle qu'un *littéral* est une variable ou la négation d'une variable. Une *clause* est une disjonction ($\vee$) de littéraux. Une *forme normale conjonctive (FNC)* est une conjonction ($\wedge$) de clauses. Une formule φ est dite en *3-FNC* si elle est en forme normale conjonctive telle que chaque clause contient au plus trois littéraux. On admet qu'étant donnée une formule propositionnelle φ en 3-FNC, il est difficile de savoir si φ possède un modèle, c'est-à-dire une valuation qui la satisfait.

1.1 De la satisfaisabilité au stable

Soit $G = (S, A)$ un graphe non orienté. On dit qu'une partie $X \subset S$ est un **stable** si c'est un ensemble de sommets deux à deux non adjacents, c'est-à-dire si pour $(s, t) \in X^2$, $st \notin A$.

Q 2. Déterminer, un stable de cardinal maximal dans le graphe G_1 représenté figure 1.

Soit $\mathcal{V} = \{v_1, \dots, v_n\}$ un ensemble de variables et φ une formule booléenne en 3-FNC, de la forme $\varphi = \bigwedge_{j=1}^m C_j$, où $C_j = \bigvee_{k=1}^{m_j} \ell_{jk}$, les ℓ_{jk} étant des littéraux (de la forme v_i ou $\neg v_i$) et $m_j \in \llbracket 1, 3 \rrbracket$.

On définit le graphe $G_\varphi = (S_\varphi, A_\varphi)$ par :

- S_φ contient un sommet par littéral apparaissant dans une clause, c'est-à-dire $S_\varphi = \bigcup_{j=1}^m S_j$ où :

$$S_j = \{v_i^{(j)} \mid v_i \text{ apparaît dans } C_j\} \cup \{\bar{v}_i^{(j)} \mid \neg v_i \text{ apparaît dans } C_j\}$$

- A_φ contient les arêtes entre chaque sommet associé à une même clause, et entre les sommets associés à une variable et sa négation :

$$A_\varphi = \{st \mid (s, t) \in S_j^2, s \neq t\} \cup \{v_i^{(j)} \bar{v}_i^{(j')} \mid i \in \llbracket 1, n \rrbracket, j \neq j'\}$$

La figure 2 représente le graphe G_{φ_0} pour $\varphi_0 = (v_1 \vee \neg v_2 \vee \neg v_3) \wedge (\neg v_1 \vee v_2) \wedge (v_1 \vee v_2 \vee v_3)$.

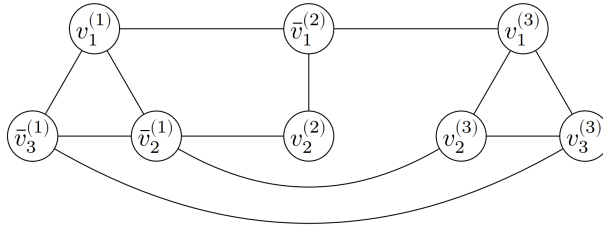


FIGURE 2 – Le graphe G_{φ_0} pour $\varphi_0 = (v_1 \vee \neg v_2 \vee \neg v_3) \wedge (\neg v_1 \vee v_2) \wedge (v_1 \vee v_2 \vee v_3)$.

Q 3. Tracer le graphe G_{φ_1} pour $\varphi_1 = (\neg v_0 \vee v_2) \wedge (v_0 \vee \neg v_1 \vee \neg v_2) \wedge (v_1 \vee \neg v_2) \wedge (\neg v_0 \vee \neg v_1 \vee v_2)$.

Q 4. Montrer que s'il existe un stable X de taille m dans G_φ , alors φ est satisfaisable. On expliquera comment construire un modèle de φ en fonction de X .

Q 5. De même, montrer que si φ est satisfaisable, alors il existe un stable de taille m dans G_φ .

1.2 Du stable à l'arbre de Steiner

Si $G = (S, A)$ est un graphe non orienté, on pose $G' = (S_G, A_G)$ un graphe tel que :

- S_G contient S et un nouveau sommet par arête de G , c'est-à-dire : $S_G = S \cup \{s_a \mid a \in A\}$
- A_G contient toutes les arêtes entre deux sommets de S , et des arêtes entre chaque sommet s_a vers chacune des extrémités de a , c'est-à-dire :

$$A_G = \{st \mid (s, t) \in S^2\} \cup \bigcup_{st \in A} \{s s_{st}, t s_{st}\}$$

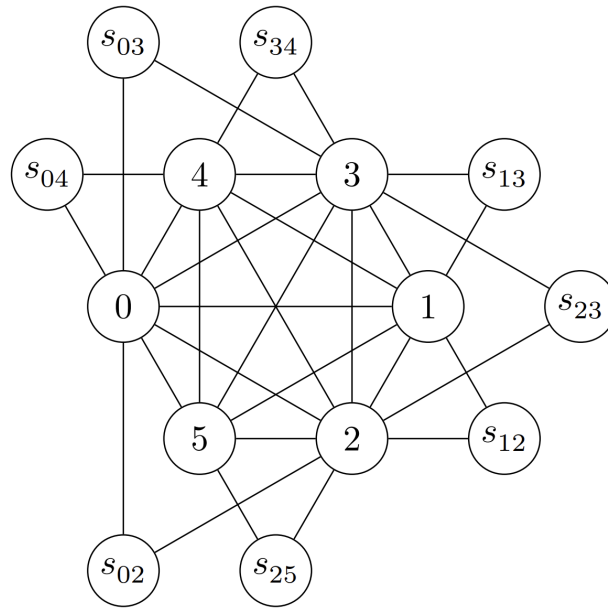


FIGURE 3 – Une représentation de G'_1 . Pour simplifier, on a renommé chaque s_i par i .

Par exemple, la figure 3 est une représentation de G'_1 où G_1 est le graphe de la figure 1.

Q 6. Montrer que s'il existe un stable de G de cardinal k , alors le graphe $G' = (S_G, A_G)$ admet un arbre de Steiner ayant $X = \{s_a \mid a \in A\}$ comme sommets terminaux et $|S| - k$ sommets de Steiner.

Q 7. Réciproquement, montrer que si le graphe $G' = (S_G, A_G)$ admet un arbre de Steiner ayant $X = \{s_a \mid a \in A\}$ comme sommets terminaux et k sommets de Steiner, alors G admet un stable de cardinal $|S| - k$.

Q 8. En déduire que si on sait trouver un arbre de Steiner ayant un nombre minimal de sommets de Steiner en temps polynomial, on peut déterminer la satisfaisabilité d'une formule propositionnelle en 3-FNC en temps polynomial.

2 Approximation du problème

Étant donné un graphe pondéré $G = (S, A, f)$ et un sous-ensemble $X \subset S$, le *problème de l'arbre de Steiner pondéré* consiste à déterminer un sous-graphe $T = (Y, B, f)$ tel que $X \subset Y \subset S$, T est un arbre et $f(T)$ est minimal.

On peut voir que le problème de l'arbre de Steiner pondéré est plus difficile que le cas non pondéré, qui n'est qu'un cas particulier où tous les poids sont égaux à 1. Malgré sa difficulté, il est possible de calculer en temps polynomial un arbre de Steiner pondéré dont le poids est moins du double d'un arbre optimal.

2.1 Algorithme de Floyd-Warshall

L'algorithme de Floyd-Warshall est un algorithme de programmation dynamique permettant de calculer toutes les distances pondérées entre deux sommets d'un graphe pondéré.

On rappelle que la représentation de G par *matrice d'adjacence* est une matrice M de dimensions $n \times n$, où $n = |S|$, telle que pour $(s, t) \in S^2$, $M_{st} = f(st)$ s'il existe $st \in A$, et $M_{st} = +\infty$ sinon.

En OCaml, `infinity` est un flottant qui se comporte comme $+\infty$: plus grand que tous les autres flottants et absorbant par addition ou par multiplication par un flottant strictement positif.

Q 9. Écrire une fonction `matrice_adjacence` qui prend en argument un graphe et renvoie sa matrice d'adjacence.

```
1 matrice_adjacence : graphe -> float array array
```

Soit $G = (S, A, f)$ un graphe pondéré. Si $c = (s_0, s_1, \dots, s_k)$ est une chaîne de s_0 à s_k , le poids de c est $f(c) = \sum_{i=1}^k f(s_{i-1}s_i)$. Pour $(s, t) \in S^2$, on appelle **distance de s à t** , notée $d_G(s, t)$ (ou $d(s, t)$ s'il n'y a pas d'ambiguïté sur le graphe), le poids minimal d'une chaîne de s à t . S'il n'existe pas de telle chaîne, on pose $d(s, t) = +\infty$ par convention. On remarque qu'avec une telle définition, pour $s \in S$, $d(s, s) = 0$.

L'algorithme de Floyd-Warshall permet de calculer les distances des sommets deux à deux en considérant une suite de matrices $M^{(k)}$, pour $k \in \llbracket 0, n \rrbracket$ définies de la manière suivante : pour $(s, t) \in S^2$, $M_{st}^{(k)}$ est le poids minimal d'une chaîne de s à t dont les sommets intermédiaires (autres que les extrémités) sont strictement inférieurs à k .

Q 10. Que vaut $M^{(0)}$?

Q 11. Montrer que pour $k \in \llbracket 0, n - 1 \rrbracket$ et $(s, t) \in S^2$,

$$M_{st}^{(k+1)} = \min(M_{st}^{(k)}, M_{sk}^{(k)} + M_{kt}^{(k)}).$$

Q 12. Écrire une fonction `floyd_warshall` qui prend en argument une matrice d'adjacence $n \times n$ d'un graphe $G = (S, A, f)$ et renvoie un couple (`dist`, `pred`) de matrices $n \times n$ telles que pour $(s, t) \in S^2$:

- `dist.(s).(t)` est un flottant correspondant à $d(s, t)$;
- `pred.(s).(t)` est le prédécesseur de t dans un chemin de poids minimal de s à t si un tel chemin existe, et -1 sinon.

```
1 floyd_warshall : float array array -> float array array * int array array
```

Q 13. Déterminer la complexité temporelle de la fonction `floyd_warshall`.

Q 14. Écrire une fonction `chaîne_min` qui prend en argument une matrice `pred` telle que décrite à la Q 28 et deux sommets s et t et renvoie une chaîne de poids minimal de s à t sous la forme d'une liste de ses sommets. La fonction renverra une liste vide s'il n'existe pas de tel chemin.

```
1 chaîne_min : int array array -> int -> int -> int list
```

2.2 Solution approchée

En considérant $G = (S, A, f)$ un graphe pondéré non orienté connexe et $X \subset S$, la solution approchée pour le problème de l'arbre de Steiner est donnée par l'algorithme suivant :

1. poser $H_1 = (X, \mathcal{P}_2(X), g)$: H_1 est un graphe complet entre les sommets de X , dont les pondérations sont données par les distances dans G (c'est-à-dire $g(st) = d_G(s, t)$) ;
2. calculer $T_1 = (X, B_1, g)$ un arbre couvrant de poids minimal de H_1 ;
3. pour chaque arête $st \in B_1 \setminus A$, la remplacer dans T_1 par une chaîne de poids minimal dans G , en rajoutant les sommets nécessaires. Le sous-graphe de G obtenu sera noté $H_2 = (Y, A_2, f)$;
4. calculer et renvoyer $T = (Y, B, f)$ un arbre couvrant de poids minimal de H_2 .

Q 15. Montrer que l'arbre T est un arbre de Steiner de G ayant X pour sommets terminaux.

Q 16. Montrer que si T possède des sommets de Steiner de degré 1, leur suppression permet d'obtenir un arbre de Steiner ayant X pour sommets terminaux de poids inférieur.

On pose T^* un arbre de Steiner de poids minimal parmi ceux ayant X comme sommets terminaux.

Q 17. Montrer que $f(T) \leq g(T_1)$, puis en déduire que $f(T) \leq 2 f(T^*)$.

On représente une partie $X \subset S$ comme un tableau de booléens `tab_X` de taille $n = |S|$ tel que pour $s \in S$, $s \in X$ si et seulement si `tab_X.(s)` vaut `true`.

Pour créer un graphe induit par une partie X en OCaml, il est nécessaire de renuméroter des sommets, via une bijection $\text{num} : X \rightarrow \llbracket 0, |X| - 1 \rrbracket$.

Q 18. Écrire une fonction `renumeroter` qui prend en argument une partie $X \subset S$ et renvoie un tableau `num` d'entiers de taille n tel que pour $s \in S$, `num.(s)` vaut -1 si $s \notin X$ et `num.(s) = num(s)` sinon, où `num` est une bijection de $X \rightarrow \llbracket 0, |X| - 1 \rrbracket$ choisie arbitrairement.

```
1 renumeroter : bool array -> int array
```

Q 19. En déduire une fonction `steiner_approche` qui, étant donné un graphe $G = (S, A, f)$ et une partie $X \subset S$ calcule un arbre de Steiner ayant X pour sommets terminaux selon l'algorithme présenté précédemment. On ne demande pas de supprimer les sommets de Steiner de degré 1.

```
1 steiner_approche : graphe -> bool array -> graphe
```

Q 20. Déterminer la complexité temporelle de la fonction `steiner_approche` en fonction de $|S|$, $|A|$ et $|X|$.

Annexe : rappels de programmation

Cette annexe rappelle le fonctionnement de différentes fonctions des modules `List` et `Array` :

- `List.iter (f: 'a -> unit)(lst: 'a list): unit` fait un appel à la fonction `f` pour chaque élément de la liste `lst`. Cette fonction s'exécute en temps linéaire en la taille de la liste, si la fonction `f` se calcule en temps constant ;
- `List.rev (lst: 'a list): 'a list` renvoie une liste contenant les mêmes éléments que `lst`, mais dans l'ordre inverse. Cette fonction s'exécute en temps linéaire en la taille de la liste ;
- `Array.length (tab: 'a array): int` renvoie la longueur du tableau `tab`. Cette fonction s'exécute en temps constant ;
- `Array.make (n: int)(x: 'a): 'a array` renvoie un tableau de longueur n dont toutes les cases contiennent la valeur `x`. Cette fonction s'exécute en temps linéaire en n ;
- `Array.make_matrix (m: int)(n: int)(x: 'a): 'a array array` renvoie une matrice de dimension $m \times n$ dont toutes les cases contiennent la valeur `x`. Cette fonction s'exécute en temps linéaire en $m \times n$.