Optimisation de réseau

Option informatique

MP 2024

CONCOURS CENTRALE•SUPÉLEC

4 heures Calculatrice autorisée

Créer un réseau routier entre différentes villes, en s'autorisant des intersections intermédiaires, dont la longueur totale est minimale est un problème très difficile à résoudre. Ce problème est une généralisation de deux problèmes pourtant faciles à résoudre : la recherche d'arbre couvrant de poids minimal, correspondant au cas où il n'y a pas d'intersections intermédiaires, et la recherche de plus court chemin, correspondant au cas où on ne souhaite relier que deux villes.

En 1836, un an seulement après l'apparition d'une ligne de train en Allemagne, Carl Friedrich Gauß s'interrogeait dans une lettre à l'astronome Christian Schumacher sur la meilleure manière de relier les villes de Hambourg, Brême, Hanovre et Brunswick par un réseau ferré.

Figure 1 Les quatre villes allemandes et une manière théoriquement optimale de les relier.

De tels arbres trouvent des applications dans la construction de réseaux (électroniques, routiers, ...). Ils peuvent également apparaître dans des phénomènes naturels : des films de savon reliant des tiges entre deux plaques peuvent former un arbre avec des points intermédiaires.

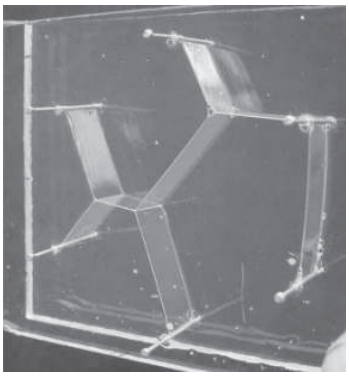

Figure 2 Films de savon reliant des tiges métalliques entre deux plaques de plexiglas. Il y a trois sommets intermédiaires.[1]

La partie I de ce problème étudie des généralités sur les arbres couvrants et une fonction de tri. La partie II présente l'algorithme de Kruskal inversé pour trouver un arbre couvrant de poids minimal dans un graphe

[1] Crédits : Dutta, P., Khastgir, S. P. & Roy, A. (2010). Steiner trees and spanning trees in six-pin soap films. American Journal of Physics, 78(2), 215-221.

connexe. La partie III montre la complexité du problème d'optimisation de réseau en se ramenant au problème de satisfiabilité. Enfin, la partie IV présente comment approcher une solution au problème d'optimisation en utilisant des arbres couvrants.

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 $G_1 = (\{s_0, s_1, s_2, s_3, s_4, s_5\}, \{s_0s_2, s_0s_3, s_0s_4, s_1s_2, s_1s_3, s_2s_3, s_2s_5, s_3s_4\})$ est représenté sur la figure 3.

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 $G[X] = (X, A \cap \mathcal{P}_2(X))$, c'est-à-dire le graphe ayant X comme ensemble de sommets et contenant toutes les arêtes de G dont les deux extrémités sont dans X.

Pour $(s, t) \in V^2$, on appelle **chaîne** de s à t une suite de sommets distincts $(s_0, s_1, ..., s_k)$ telle que $s_0 = s$, $s_k = t$ et pour $i \in [\![1, k]\!]$, $s_{i-1}s_i \in A$. On appelle **cycle** de G une suite de sommets $(s_0, s_1, ..., s_k)$ telle que $k \geqslant 3$, $(s_0, ..., s_{k-1})$ 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 $G = (S, A)$ qu'il 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 $f : A \to \mathbb{R}_+$ est une fonction qui à chaque arête associe un **poids** qui est un réel positif. La figure 3 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 3 Les graphes G_1 et G_2

I Généralités

Q 1. Dessiner sans justifier un arbre couvrant de G_2 de poids minimal.

Q 2. Soit $G = (S, A)$ un graphe non orienté. Montrer les deux propriétés suivantes :

— Si G est connexe, alors $|A| \geqslant |S| - 1$.

— Si G est sans cycle, alors $|A| \leqslant |S| - 1$.

Q 3. En déduire l'équivalence entre les trois propriétés suivantes, pour $G = (S, A)$ un graphe non orienté :

— G est un arbre.

— G est connexe et $|A| = |S| - 1$.

— G est sans cycle et $|A| = |S| - 1$.

Pour les deux questions suivantes, on suppose que $G = (S, A, f)$ est un graphe pondéré tel que f est injective (toutes les arêtes ont un poids différent).

Q 4. Montrer que G possède un unique arbre couvrant de poids minimal.

Q 5. Soit $T^* = (S, B^*)$ l'unique arbre couvrant de poids minimal de G. On suppose que $a \in A$ est une arête dont le poids est maximal dans un cycle de G. Montrer que $a \notin B^*$.

Les trois questions qui suivent ont pour objectif d'implémenter un algorithme de tri efficace.

Q 6. Écrire une fonction `separation` telle que si `lst` est une liste, alors `separation lst` renvoie un couple `(lst1, lst2)` tel que chaque élément de `lst` apparaît soit dans `lst1`, soit dans `lst2`, et les tailles de `lst1` et `lst2` diffèrent d'au plus 1.

```
separation : 'a list -> 'a list * 'a list
```

Q 7. Écrire une fonction `fusion` telle que si `lst1` et `lst2` sont deux listes triées par ordre croissant, alors `fusion lst1 lst2` renvoie une liste triée par ordre croissant contenant tous les éléments de `lst1` et `lst2`.

```
fusion : 'a list -> 'a list -> 'a list
```

Q 8. En déduire une fonction `tri_fusion` qui prend en argument une liste et renvoie une liste triée par ordre croissant contenant les mêmes éléments. Quelle est la complexité temporelle de cette fonction ? (On ne demande pas de justifier.)

```
tri_fusion : 'a list -> 'a list
```

II Algorithme de Kruskal inversé

Q 9. Rappeler le principe de l'algorithme de Kruskal et sa complexité temporelle en fonction de $|S|$ et $|A|$ lorsqu'il est appliqué à un graphe pondéré connexe $G = (S, A, f)$.

L'algorithme de Kruskal inversé est une variante de l'algorithme de Kruskal qui permet de trouver un arbre couvrant de poids minimal dans un graphe pondéré connexe. En voici une description en pseudo-code.

Entrées : Graphe pondéré $G = (S, A, f)$ non orienté connexe
$B \leftarrow$ copie de A
Trier B par ordre décroissant de poids
pour $a \in B$ par ordre décroissant de poids **faire**
    si $(S, B \setminus \{a\})$ est connexe **alors**
        $B \leftarrow B \setminus \{a\}$
    fin si
fin pour
renvoyer (S, B)

Algorithme 1 Algorithme de Kruskal

Q 10. Appliquer l'algorithme de Kruskal inversé au graphe G_3 de la figure figure 4. On ne demande pas la description de l'exécution détaillée, mais simplement une représentation graphique de l'arbre obtenu.

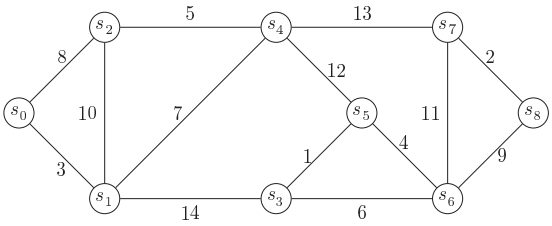

Figure 4 Le graphe pondéré G_3

On implémente un graphe pondéré par tableau de listes d'adjacences en OCaml, selon le type suivant :

```
type graphe = (int * float) list array
```

Si $G = (S, A, f)$ est implémenté par un objet `g` de type `graphe`, alors :

— l'ensemble des sommets est $S = [\![0, n-1]\!]$, où $n = |S|$;

— `g` est un tableau de taille n tel que pour tout $s \in S$, `g.(s)` est une liste contenant des couples (t, p) tels que $st \in A$ et $f(st) = p$.

Les listes d'adjacence ne sont pas nécessairement supposée triées par ordre croissant des sommets ou des poids. Par exemple, le graphe G_2 de la figure 3 peut être créé par la commande :

```
let g2 = [|[(2, 7.); (3, 6.); (1, 10.); (4, 7.)]; [(2, 5.); (0, 10.)];
          [(0, 7.); (1, 5.); (3, 9.)]; [(0, 6.); (2, 9.)]; [(0, 7.)]|]
```

On rappelle que les opérations de comparaisons (`max`, `min`, `<`, `>`, …) peuvent s'effectuer sur des uplets selon l'ordre lexicographique.

Q 11. Écrire une fonction `tri_aretes` telle que si `g` est la représentation d'un graphe pondéré $G = (S, A, f)$, alors `tri_aretes g` renvoie la liste des triplets $(f(st), s, t)$, triée par ordre **décroissant** des $f(st)$, tels que $st \in A$ et $s < t$. Cette liste ne devra pas contenir de doublons.

```
tri_aretes : graphe -> (float * int * int) list
```

Q 12. Écrire une fonction `connectes` telle que si `g` est la représentation d'un graphe pondéré $G = (S, A, f)$, $(s, t) \in S^2$ et `poids_max` est un flottant, alors `connectes g s t poids_max` renvoie un booléen qui vaut `true` si et seulement s'il existe un chemin de s à t dans G ne passant que par des arêtes dont le poids est strictement inférieur à `poids_max`. La fonction devra avoir une complexité linéaire en $|S| + |A|$ et on demande de justifier brièvement.

```
connectes : graphe -> int -> int -> float -> bool
```

Q 13. En déduire une fonction `kruskal_inverse` qui prend en argument un graphe $G = (S, A, f)$ et renvoie le graphe obtenu selon le principe de l'algorithme de Kruskal inversé. On supposera que le graphe G donné en argument est connexe et que la fonction de pondération f est injective.

```
kruskal_inverse : graphe -> graphe
```

Q 14. Montrer que si G est connexe, le graphe renvoyé par l'algorithme de Kruskal inversé appliqué à G est un arbre couvrant de G.

Q 15. Montrer que l'arbre couvrant renvoyé par l'algorithme de Kruskal inversé appliqué à un graphe pondéré connexe G est de poids minimal. On pourra commencer par traiter le cas où la fonction de pondération est injective.

Q 16. Déterminer la complexité temporelle de la fonction `kruskal_inverse`. Commenter.

III 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 17. Représenter graphiquement un arbre de Steiner du graphe G_1 de la figure 3, 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.

III.A – 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 18. Déterminer, en justifiant, un stable de cardinal maximal dans le graphe G_1 représenté figure 3.

Soit $\mathcal{V} = \{v_1, ..., v_n\}$ 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 $m_j \in [\![1, 3]\!]$.

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 :

$$S_j = \left\{ v_i^{(j)} \mid v_i \text{ apparait dans } C_j \right\} \cup \left\{ \overline{v_i}^{(j)} \mid \neg v_i \text{ apparait dans } C_j \right\}$$

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

$$A_\varphi = \left\{ st \mid (s, t) \in S_j^2, s \neq t \right\} \cup \left\{ v_i^{(j)} \overline{v_i}^{(j')} \mid i \in [\![1, n]\!], j \neq j' \right\}$$

La figure 5 représente le graphe G_{φ_0} pour $\varphi_0 = (v_1 \vee \neg v_2 \vee \neg v_3) \wedge (\neg v_1 \vee v_2) \wedge (v_1 \vee v_2 \vee v_3)$.

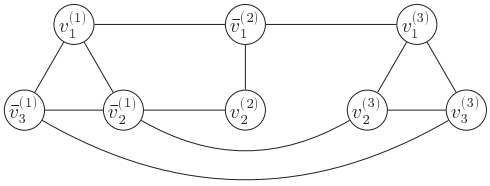

Figure 5 Le graphe G_{φ_0} pour $\varphi_0 = (v_1 \vee \neg v_2 \vee \neg v_3) \wedge (\neg v_1 \vee v_2) \wedge (v_1 \vee v_2 \vee v_3)$.

Q 19. 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 20. 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 21. De même, montrer que si φ est satisfaisable, alors il existe un stable de taille m dans G_φ.

III.B – 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 sommet de S, et des arêtes entre chaque sommet s_a vers chacune des extrémités de a, c'est-à-dire :

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

Par exemple, la figure 6 est une représentation de G'_1 où G_1 est le graphe de la figure 3.

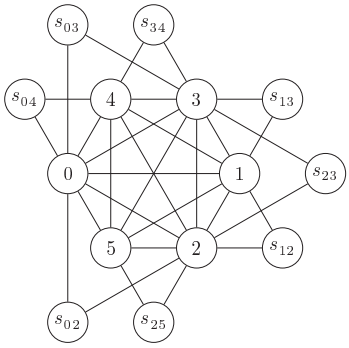

Figure 6 Une représentation de G'_1. Pour simplifier, on a renommé chaque sommet s_i par i.

Q 22. 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 23. 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 24. 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.

IV 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.

IV.A – 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 25. Écrire une fonction `matrice_adjacence` qui prend en argument un graphe et renvoie sa matrice d'adjacence.

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

Soit $G = (S, A, f)$ un graphe pondéré. Si $c = (s_0, s_1, ..., s_k)$ 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 $k \in [\![0, n]\!]$ définies de la manière suivante : pour $(s,t) \in S^2$, $M^{(k)}_{st}$ 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 26. Que vaut $M^{(0)}$?

Q 27. Montrer que pour $k \in [\![0, n-1]\!]$ et $(s,t) \in S^2$, $M^{(k+1)}_{st} = \min(M^{(k)}_{st}, M^{(k)}_{sk} + M^{(k)}_{kt})$.

Q 28. 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.

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

Q 29. Déterminer la complexité temporelle de la fonction `floyd_warshall`.

Q 30. Écrire une fonction `chaine_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.

```
chaine_min : int array array -> int -> int -> int list
```

IV.B – 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 :

— 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 $g(st) = d_G(s,t)$) ;

— calculer $T_1 = (X, B_1, g)$ un arbre couvrant de poids minimal de H_1 ;

— 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 $H_2 = (Y, A_2, f)$.

— calculer et renvoyer $T = (Y, B, f)$ un arbre couvrant de poids minimal de H_2.

Q 31. Montrer que l'arbre T est un arbre de Steiner de G ayant X pour sommets terminaux.

Q 32. 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 33. Montrer que $f(T) \leqslant g(T_1)$, puis en déduire que $f(T) \leqslant 2f(T^*)$.

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 $num : X \to [\![0, |X|-1]\!]$.

Q 34. 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 $X \to [\![0, |X|-1]\!]$ choisie arbitrairement.

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

Q 35. 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.

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

Q 36. 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 $m \times n$.

• • • FIN • • •