Corrigé Centrale 2024

Q1. Prendre les arêtes $\{s_1, s_2\}$, $\{s_3, s_0\}$, $\{s_0, s_2\}$ et $\{s_0, s_4\}$.
Q2. Montrons ces résultats par récurrence sur $|A|$. Supposons que $|A| = 0$. Alors, si S est connexe, il ne peut avoir qu'un sommet donc $|S| = 1$ donc $|S| - 1 = 0 \leq |A|$. De plus, $|S| \geq 1$ donc $|A| \leq |S| - 1$.
Soit $n \in \mathbb{N}$. Supposons le résultat si $|A| \leq n$. Considérons $G = (S, A)$ un graphe tel que $|A| = n + 1$.
Supposons que G est connexe. Retirons une arête $a = \{x, y\}$ à A. Notons $A' = A \backslash \{a\}$ et $G' = (S, A')$. Soit $z \in S$. Il existe une chaîne de z à x dans A. Si cette chaîne passe par l'arête a, z est relié à y dans A' ; sinon, z est relié à x dans A'. Tout sommet de S est donc relié à x ou à y dans G'. Si x et y sont reliés dans A', G' est connexe ; sinon, G' possède deux composantes connexes : $G_x = (S_x, A_x)$ la composante des sommets reliés à x et $G_y = (S_y, A_y)$ celle des sommets reliés à y. Si G' est connexe, en vertu de l'hypothèse de récurrence, $|S| - 1 \leq |A'| = |A| - 1 \leq |A|$; sinon, en vertu de l'hypothèse de récurrence appliquée aux graphes G_x et G_y connexes, $|S_x| - 1 \leq |A_x|$ et $|S_y| - 1 \leq |A_y|$; en outre, $|S| = |S_x| + |S_y|$, $|A'| = |A_x| + |A_y|$ et $|A| = |A'| + 1$; finalement, $|S| - 1 = 1 + |S_x| - 1 + |S_y| - 1 \leq 1 + |A_x| + |A_y| = |A|$.
Supposons que G est sans cycle. Considérons une arête $a = \{x, y\} \in A$. Notons $A' = A \backslash \{a\}$, $G' = (S, A')$, G_x et G_y les composantes connexes de x, y dans G'. Si $G_x = G_y$, alors x et y sont reliés dans G' donc l'arête a ajoutée à une chaîne de x à y dans A' complète un cycle dans $G = (S, A)$. De ce fait, G_x et G_y sont distinctes et donc d'intersection vide. De plus, G_x et G_y sont également sans cycle. Notons $G'' = (S \backslash (S_x \cup S_y), A' \backslash (A_x \cup A_y))$. G'' est aussi sans cycle. De par l'hypothèse de récurrence, $|A_x| \leq |S_x| - 1$, $|A_y| \leq |S_y| - 1$ et $|A' \backslash (A_x \cup A_y)| \leq |S \backslash (S_x \cup S_y)| - 1$; de ce fait, $|A| - 1 = |A'| = |A_x| + |A_y| + |A' \backslash (A_x \cup A_y)| \leq |S_x| - 1 + |S_y| - 1 + |S \backslash (S_x \cup S_y)| - 1 = |S| - 1$.
Par récurrence, les deux résultats à établir sont prouvés.
Q3. Supposons que G est un arbre. Par définition, G est connexe et sans cycle. D'après la question 2, $|A| \geq |S| - 1$ et $|A| \leq |S| - 1$ donc $|A| = |S| - 1$.
Supposons que G est connexe et $|A| = |S| - 1$. Supposons que G possède un cycle partant de x dont la dernière arête est $a = \{y, x\}$. Soit $A' = A \backslash \{a\}$. Ce cycle privé de sa dernière arête permet de relié x à y dans A'. Tout sommet de S est relié à x ou à y donc G' est encore connexe donc $|S| - 1 \leq |A'| = |A| - 1 = |S| - 2$ ce qui est faux. Par conséquent, G n'a pas de cycle donc G est un arbre.
Supposons que G est sans cycle et $|A| = |S| - 1$. Supposons qu'il existe deux sommets x et y non reliés dans G. Soient $A' = A \cup \{\{x, y\}\}$ et $G' = (S, A')$. G étant connexe, il n'existe pas de cycle dans G' n'utilisant pas $\{x, y\}$; un cycle dans G' utilisant $\{x, y\}$ permettrait de relier x et y dans G'. Par conséquent, G' est aussi sans cycle donc $|A'| \leq |S| - 1$ donc $|A| + 1 \leq |A|$ ce qui est faux donc G est connexe donc G est un arbre.
Q4. Supposons que G est connexe (hypothèse manquante à l'énoncé).
Si G est connexe et $|A| = 0$, $|S| = 1$ donc $|A| = |S| - 1$ donc G est un arbre donc G possède des arbres couvrant.
Soit $n \in \mathbb{N}$. Supposons que, si $G = (S, A)$ est connexe et $|A| \leq n$, G admet un arbre couvrant. Soit $G = (S, A)$ un graphe connexe tel que $|A| = n + 1$. Supposons que G est un arbre ; alors G admet un arbre couvrant (lui-même !). Sinon, G admet un cycle. Soit a une arête de ce cycle. Soit $A' = A \backslash \{a\}$. $G' = (S, A')$ est connexe et $|A'| \leq n$ donc G' admet un arbre couvrant qui est un arbre couvrant de G.
Par récurrence, G admet des arbres couvrants. Les sous-graphes de G sont finis donc les arbres couvrant aussi. Il existe donc un arbre couvrant de G ayant un poids minimal.
Montrons, par récurrence sur $|A|$, qu'un arbre couvrant de poids minimal est unique.
Ce résultat est trivial si $|A| \leq 1$.
Supposons que ce résultat vaut si $G = (S, A)$ est connexe et $|A| \leq n$. Soit $G = (S, A)$ connexe tel que $|A| = n + 1$. Soit $a = \{x, y\}$ l'arête de plus grand poids de A. Soient $A' = A \backslash \{a\}$, $G' = (S, A')$, $G_x = (S_x, A_x)$ et $G_y = (S_y, A_y)$ les composantes connexes de x et y. Supposons que x et y ne sont pas reliés dans G'. Tout arbre couvrant de G est constitué dans d'un arbre couvrant de G_x, d'un arbre couvrant de G_y et de a. Un arbre couvrant minimal de G est donc constitué d'un arbre couvrant de G_x, d'un arbre couvrant de G_y et de a. Par récurrence, les arbres couvrants minimaux de G_x, G_y étant uniques, celui de G est unique. Supposons que x et y sont reliés dans G'. Considérons un arbre couvrant $T = (S, B)$ de G. Supposons que celui-ci contient l'arête a. $T = (S, B \backslash \{a\})$ possède deux composantes connexes : celles de x et de y que l'on notera T_x et T_y. Comme x et y sont reliés dans G', il existe une arête a' reliant T_x à T_y qui a, par hypothèse, un poids strictement inférieur à celui de a. Alors $(S, (B \backslash \{a\}) \cup \{a'\})$ est un arbre couvrant de G de poids strictement inférieur à celui de T. Par conséquent, tout arbre couvrant de G ne contient pas a donc c'est un arbre couvrant de G'. Par hypothèse de récurrence, il est unique.
Q5. Nous l'avons prouvé dans la question précédente. On considère un arbre (S, B) contenant a. $(S, B \backslash \{a\})$ contient deux composantes connexes G_1, G_2. Dans G, les sommets de G_1 et de G_2 sont reliés. Comme l'arête a est maximal dans le cycle, il existe une arête a' reliant un sommet de G_1 et un sommet de G_2 dont le poids est strictement inférieur à celui de a. $(S, (B \backslash \{a\}) \cup \{a'\})$ est un arbre couvrant de poids strictement inférieur à celui de (S, B). Par conséquent, $a \notin B^*$.
Q6.

```
let rec separation lst = match lst with
   | [] -> ([],[])
   | x::q ->
   if q=[] then ([x],[])
   else let y::q0=q in let (lst1,lst2)=separation q0 in ( x::lst1 , y::lst2);;
```

Q7.

```
let rec fusion lst1 lst2 = match lst1,lst2 with
   | [],lst2 -> lst2
   | lst1,[] -> lst1
```

```
   | x::l1, y::l2 -> if x<y then x::(fusion l1 lst2) else y::(fusion lst1 l2);;
```

Q8.

```
let rec tri_fusion l = match l with
   | [] -> []
   | [x] -> [x]
   | _ -> let (l1,l2)=separation l in fusion ((tri_fusion l1) (tri_fusion l2));;
```

La complexité de cette fonction est en $O(n\ln(n))$ où n est la taille de la liste en argument.
Q9. L'algorithme de Kruskal part d'un graphe sans arête $(S, B) = (S, \emptyset)$; il trie les arêtes par poids croissant ; pour chaque arête a dans l'ordre croissant, si $(S, B \cup \{a\})$ est sans cycle, on remplace B par $B \cup \{a\}$. Le tri des arêtes peut se faire en $O(|A| \ln |A|)$; le vérification du fait qu'une arête n'ajoute pas de cycle et la modification éventuelle du graphe peut se faire en $O(\ln |S|)$. On obtient une complexité en $O(|A| \ln |A| + |A| \ln |S|) = O(|A| \ln(|A| \times |S|))$.
Q10. On va retirer les arêtes $\{s_1, s_3\}$, $\{s_4, s_7\}$, $\{s_7, s_6\}$, $\{s_2, s_1\}$, $\{s_0, s_2\}$, $\{s_3, s_6\}$.
Q11.

```
let tri_aretes g =
   let rec aux voisins i liste = match voisins with
      | [] -> if i=Array.length g-1 then liste else aux g.(i+1) (i+1) liste
      | (j,p)::q -> if i<j then aux q i ((p,i,j)::liste) else aux q i liste
   in tri_fusion (aux g.(0) 0 []);;
```

Q12.

```
let connectes g s t poids_max =
   let b=Array.make (Array.length g) false in b.(s)<-true;
   let rec aux1 voisins parcours = match voisins with
      | [] -> parcours
      | (x,p)::q -> if not(b.(x)) && p<poids_max then aux1 q (x::parcours) in
   let rec aux2 liste = match liste with
      | [] -> ()
      | x::q -> if b.(x) then aux2 q else begin b.(x)<-true; aux2 (aux1 g.(x) q) end
   in b.(t);;
```

Cette fonction effectue un parcours en profondeur de graphe. Sa complexité est donc en $O(|S| + |A|)$ (en effet, chaque arête crée au plus un appel récursif à "aux1" et chaque sommet crée au plus un appel récursif à "aux2").
Q13.

```
let kruskal_inverse g =
   let b=List.rev (tri_aretes g) and let t=Array.make (Array.length g) [] in
   let rec aux1 l = match l with
      | [] -> []
      | (p,s,t)::q -> if connectes g s t p then aux q else (p,s,t)::(aux q)
   and aux2 l = match l with
      | [] -> g0
      | (p,s,t)::q -> g0.(s) <- (t,p)::g0.(s); g0.(t) <- (s,p)::g0.(t); aux q
   in aux2 aux1 b;;
```

Q14. Par définition, le graphe (S, B) reste connexe à chaque passage de boucle. Par conséquent, le couple (S, B) renvoyé est connexe. Supposons qu'il possède un cycle. Soit a l'arête de ce cycle de poids maximal. Alors, lors du traitement de cette arête par l'algorithme de Kruskal, $(S, B\backslash\{a\})$ étant connexe, l'arête a ne devrait pas figurer dans (S, B) à la fin de l'algorithme. Ceci étant absurde, (S, B) est sans cycle. Par conséquent, (S, B) est un arbre couvrant.
Q15. Supposons que la fonction de pondération est injective. Si $|A| \leq 1$, l'algorithme de Kruskal inversé renvoie trivialement un arbre couvrant de poids minimal. Supposons ce résultat si $|A| \leq n$ et soit $G = (S, A)$ un graphe connexe tel que $|A| = n + 1$. Soit $a = \{x, y\}$ l'arête de poids maximal. Si a n'est pas sur un cycle de G, l'unique arbre couvrant de G est l'union de ceux des composantes de G_x de x et G_y de y auquel on ajoute l'arête a. Par récurrence, l'algorithme de Kruskal inverse appliqué à $A\backslash\{a\}$ conservera a (puisque $(S, A\backslash\{a\})$ n'est pas connexe) et obtiendra des arbres couvrants de poids minimaux de G_x et de G_y. L'algorithme de Kruskal inverse renvoie donc l'arbre couvrant de poids minimal de G. Si a est sur un cycle de G, l'algorithme de Kruskal l'enlève de A et renverra un arbre couvrant de poids minimal de $(S, A\backslash\{a\})$ qui, d'après la question 5, est un arbre couvrant de poids minimal de G. Par récurrence, l'algorithme de Kruskal inverse renvoie l'arbre couvrant de poids minimal de G.
Supposons que la fonction de poids f ne soit pas injective. Soit $T = (S, B)$ l'arbre choisi par l'algorithme de Kruskal. Il existe une suite de fonctions injectives $g_n : A \to \mathbb{R}$ dont la limite (simple) est f et telle que l'ordre choisi pour trier les arêtes de A selon le poids f soit celui selon le poids g_n. L'algorithme de Kruskal pour les poids f et g_n choisissent les mêmes arêtes donc cet algorithme renvoie un arbre couvrant de poids minimal pour g_n. Si $T' = (S, B')$ est un autre arbre couvrant, pour tout $n \in \mathbb{N}$, $g_n(B) < g_n(B')$ donc, par passage à la limite dans les inégalités, $f(B) \leq f(B')$. T est donc un arbre couvrant de poids minimal pour f.
Q16. Le tri des arêtes est en $O(|A| \ln(|A|))$; la vérification de la connexité de $(S, B\backslash\{a\})$ est en $O(|A| + |S|)$. Au total, la complexité est en $O(|A| \ln(|A|)) + O(|A|(|A| + |S|)) = O(|A|^2)$. Cette complexité est moins bonne que la complexité de l'algorithme de Kruskal. Le problème vient du lancement à chaque passage de boucle d'un parcours de graphe.
Q17. Notez que le graphe dessiné n'a pas de sommet s_5. En revanche, la définition du graphe figurant au milieu de cette page 2 contient bien un sommet s_5 qui n'est relié qu'à s_2. Un arbre de Steiner ayant s_1, s_4, s_5 comme sommets terminaux est, par exemple (Y, B) où $Y = \{s_1, s_4, s_5, s_3, s_2\}$ et $B = \{s_4s_3, s_3s_1, s_3s_1, s_2s_5\}$. Cet arbre a donc comme sommets de Steiner s_3 et s_2. Si (Y, B) est un arbre ayant un seul sommet de Steiner, ce sommet doit être relié, comme Y contient s_4, s_1, s_5 et que ces sommets ne sont pas voisins, le seul sommet de Steiner doit être voisin de s_4, s_1, s_5. Un tel sommet n'existe pas. Il n'existe donc pas d'arbre de Steiner ayant un seul sommet de Steiner.

Q18. Si l'on reprend la définition et non le graphe dessiné, $\{s_4, s_1, s_5\}$ est un stable de cardinal 3. Si un ensemble X de sommets contient 4 éléments, il contient s_0, s_3 ou s_2. Chacun de ces sommets a au moins 3 voisins donc possède un voisin parme les éléments de X. Ce n'est donc pas un stable. Un stable a donc au plus 3 éléments.

Q19. Ce graphe a sommets 10 sommets : $\neg v_0^{(1)}$, $v_2^{(1)}$, $v_0^{(2)}$, $\neg v_1^{(2)}$, $\neg v_2^{(2)}$, $v_1^{(3)}$, $\neg v_2^{(3)}$, $\neg v_0^{(4)}$, $\neg v_1^{(4)}$, $v_2^{(4)}$. Il y a des arêtes $\neg v_0^{(1)}$ et $v_2^{(1)}$, entre $v_0^{(2)}$ et $\neg v_1^{(2)}$, $v_0^{(2)}$ et $\neg v_2^{(2)}$, $\neg v_1^{(2)}$ et $\neg v_2^{(2)}$, entre $v_1^{(3)}$ et $\neg v_2^{(3)}$, entre $\neg v_0^{(4)}$ et $\neg v_1^{(4)}$, $\neg v_0^{(4)}$ et $v_2^{(4)}$, $\neg v_1^{(4)}$ et $v_2^{(4)}$, ainsi que des arêtes entre $\neg v_0^{(1)}$ et $v_0^{(2)}$, $\neg v_0^{(4)}$ et $v_0^{(2)}$, entre $\neg v_1^{(2)}$ et $v_1^{(3)}$, $v_1^{(3)}$ et $\neg v_1^{(4)}$, entre $v_2^{(1)}$ et $\neg v_2^{(2)}$, $v_2^{(1)}$ et $\neg v_2^{(3)}$, $\neg v_2^{(2)}$ et $v_2^{(4)}$, $\neg v_2^{(3)}$ et $v_2^{(4)}$.

Q20. Soit X un stable de G_φ. Comme $v_i^{(j)}$ et $\neg v_i^{(j')}$ sont voisins pour tout $j \neq j'$ et pour $j = j'$, il ne peut exister j et j' tel que $v_i^{(j)}$ et $\neg v_i^{(j')}$ sont dans X. S'il existe j tel que $v_i^{(j)}$ est dans X, on pose $d(v_i) = V$; s'il existe j tel que $\neg v_i^{(j)}$ est dans X, on pose $d(v_i) = F$; s'il n'existe pas j tel que $v_i^{(j)}$ ou $\neg v_i^{(j)}$ est dans X, on peut poser arbitrairement $d(v_i)$ comme F ou V.
Comme, pour tout j, les littéraux intervenant dans la clause C_j sont voisins, il ne peut y avoir qu'au plus un littéral appartenant à X. Comme X a m éléments, il y a un unique littéral l de chaque clause C_j dans X. Par définition de la distribution d, $d(l) = V$ donc $d(C_j) = V$. Finalement, $d(\varphi) = V$. d est un modèle de φ.

Q21. Supposons que φ est satisfiable. Soit d un modèle de φ. Pour tout j, C_j est satisfaite par d donc il existe k_j et un littéral l_{j,k_j} de C_j satisfait par d. On pose alors $X = \{l_{j,k_j} \,/\, 1 \leq j \leq m\}$. X possède m éléments. Pour chaque $1 \leq j \neq j' \leq m$, comme $j \neq j'$, l_{j,k_j} et $l_{j',k_{j'}}$ ne peuvent être adjacents qu'à la condition que ces littéraux soient négation l'un de l'autre. Or ces deux littéraux sont satisfaits par la distribution d donc ils ne peuvent être adjacents. X est donc un stable de cardinal m.

Q22. Soit Z un stable de G. $G'[S\backslash Z]$ est connexe. On considère $(S\backslash Z, B)$ un arbre couvrant de $G'[S\backslash Z]$. Pour tout $a \in A_G\backslash A$, si $a = st$, on ne peut avoir $s \in Z$ et $t \in Z$ car Z est un stable. On choisit un élément parmi s et t n'étant pas dans Z que l'on notera $\tilde{s}_a$.
On pose $Y = X \cup (S\backslash Z)$ et $C = B \cup \{s_a \tilde{s}_a\}$; autrement dit, on ajoute aux arêtes de B une arête reliant tout élément s_a de X à l'une des extrémités de a n'étant ps dans Z.
Un cycle de (Y, C) ne peut passer que par des sommets de $S\backslash Z$ car $(S\backslash Z, B)$ est un arbre ; un cycle de (Y, C) ne peut passer par des sommets de X un tel sommet n'est adjacent que d'une arête dans (Y, C).
(Y, C) est donc un arbre. Ses sommets de Steiner sont les éléments de $Y\backslash X = S\backslash Z$; il y en a donc $|S| - k$.

Q23. Supposons que G' possède un arbre de Steiner (Y, B) ayant X comme sommets terminaux et k sommets de Steiner. On pose $Z = S\backslash(Y\backslash X)$. $Z \subset S$. Supposons que deux éléments de Z sont adjacents. Alors ils sont adjacents à un élément de X. Or un élément de X n'est relié qu'à deux éléments de S et, X étant l'ensemble des sommets terminaux, l'un de ces deux éléments est un sommet de l'arbre de Steiner donc est dans $Y\backslash X$ donc n'est pas dans Z. Ainsi, les éléments de Z ne peuvent être adjacents. Z est donc un stable de cardinal $|Z| = |S| - |Y\backslash X| = |S| - k$.

Q24. Etant donner une formule propositionnelle φ en $3-FNC$, on peut construire en temps polynômial le graphe G_φ défini précédemment. On peut construire en temps polynômial le graphe G'_φ défini précédemment. Par hypothèse, on peut trouver en temps polynômial un arbre de Steiner ayant un nombre minimal k de sommets de Steiner. On peut donc trouver un stable de cardinal maximal $|S| - k$. Si $|S| - k \geq m$ où m est le nombre de clauses de φ, alors G_φ a un stable de cardinal au moins m donc un stable de cardinal m donc φ est satisfiable. Si $|S| - k < m$, G_φ n'admet pas de stable de cardinal m donc φ n'est pas satisfiable. Ainsi, on peut déterminer en temps polynômial si φ est satisfiable.

Q25.

```
let matrice_adjacence g = let n=Array.length g in
   let m=Array.make_matrix n n infinity in
   let rec aux l i = match l with
      | [] -> ()
      | (j,p)::q -> m.(i).(j) <- p; aux q i;
   for k=0 to n-1 do
      aux g.(k) k
   done;
   m;;
```

Q26. $M^{(0)}$ est la matrice d'adjance M.

Q27. Soient $0 \leq k \leq n-1$ et $(s, t) \in S^2$. Un chemin minimal de s à t dont les sommets sont strictement inférieurs à k est un chemin de s à t dont les sommets sont strictement inférieurs à $k+1$ donc $M_{st}^{(k+1)} \leq M_{st}^{(k)}$; la concaténation d'un chemin de poids minimal de s à k et d'un chemin de poids minimal de k à t dont les sommets sont strictement inférieurs à k forme un chemin dont les sommets sont strictement inférieurs à $k+1$; cette concaténation a poids $M_{sk}^{(k)} + M_{kt}^{(k)}$ donc $M_{st}^{(k+1)} \leq M_{sk}^{(k)} + M_{kt}^{(k)}$. Ainsi, $M_{st}^{(k+1)} \leq \min(M_{st}^{(k)}, M_{sk}^{(k)} + M_{kt}^{(k)})$.
Soit un chemin de poids minimal de s à t ne passant que par des sommets strictement inférieurs à $k+1$. Si ces sommets sont tous strictement inférieurs à k, $M_{st}^{(k)} \leq M_{st}^{(k+1)}$ donc, dans ce cas, $M_{st}^{(k+1)} = \min(M_{st}^{(k)}, M_{sk}^{(k)} + M_{kt}^{(k)})$. Si l'un de ces sommets est k, s'il y a plusieurs fois k, un cycle partant de k arrivant en k ayant poids au plus 0, on peut supposer, qui à supprimer un ou plusieurs cycles en k, que k n'apparaît qu'une fois dans ce chemin de poids minimal. Ce chemin est donc la concaténation d'un chemin de s à k ne passant que par des sommets strictement inférieurs à k et d'un chemin de k à t ne passant que par des sommets strictement inférieurs à k. On a donc, dans ce cas, $M_{sk}^{(k)} + M_{kt}^{(k)} \leq M_{st}^{(k+1)}$ donc $M_{st}^{(k+1)} = \min(M_{st}^{(k)}, M_{sk}^{(k)} + M_{kt}^{(k)})$.

Q28.

```
let floyd_warshall m = let n=Array.length g in
   let dist=Array.make_matrix n n infinity and pred=Array.make_matrix n n -1 in
   for i=0 to n-1 do
      for j=0 to n-1 do
         if i=j then dist.(i).(j)<-0;
         if m.(i).(j)<infinity then begin dist.(i).(j)=m.(i).(j); pred.(i).(j)<-i end
      done
   done;
   for k=0 to n-1 do
      for i=0 to n-1 do
         for j=0 to n-1 do
```

```
            if dist.(i).(k)+dist.(k).(j)<dist.(i).(j) then begin
            dist.(i).(j)<-dist.(i).(k)+dist.(k).(j);
            pred.(i).(j)<-pred.(k).(j) end
         done
      done
   done;
   (dist,pred);;
```

Q29. Cette fonction contient deux boucles de taille $O(n^2)$ et $O(n^3)$ dont chaque instruction a complexité bornée. Sa complexité temporelle est donc $O(n^3)$.
Q30.

```
let chaine_min pred s t =
   let rec aux k l = if k=s then k::l else aux pred.(k).(t) (k::q)
   in aux t [];;
```

Q31. Par construction, H_2 est un sous-graphe de G ; T est un arbre couvrant de H_2 donc un sous-graphe de G qui est un arbre ; enfin, $X \subset Y$ donc c'est un arbre de Steiner de sommets terminaux X.
Q32. Dans un arbre, la suppression d'un sommet de degré un maintient la structure d'arbre. La suppression d'un sommet de Steiner de degré un maintient donc la structure d'arbre de Steiner de sommets terminaux X. En outre, cet arbre possède une arête de moins donc a son poids qui diminue du poids de cette arête. On obtient donc un arbre de Steiner de sommets terminaux X et de poids inférieur.
Q33. Le remplacement d'une arête de $B_1 \backslash A$ par une chaîne de poids minimal ne modifie pas le poids global du graphe donc $g(T_1) = f(H_2)$. Comme $T = (Y, B, f)$ est un sous-graphe de $H_2 = (Y, A_2, f)$, $f(T) \leq f(H_2)$ donc $f(T) \leq g(T_1)$. **Q34.**

```
let renumeroter tab = let n=Array.length tab in let num=Array.make n -1 and i=ref 0 in
   for s=0 to n-1 do
      if tab.(s) then begin num.(s)<- !i; incr i end
   done;
   num;;
```

Q35.

```
let steiner_approche g x = let num=renumeroter x in
   let h1=Array.make (Array.length x) [] in
   let m=matrice_adjacence g in
   let (dist,pred)=floyd_warshall m in
   for s=0 to n-1 do
      for t=0 to n-1 do
         if x.(s) && x.(t) then
            begin let t0=num.(t) and s0=num.(s) in
            h.(t0) <- (s0,dist.(t0).(s0))::h1.(t0)
      done;
   done;
   let t1=kruskal_inverse h1 in
   let k=ref Array.length x and let aretes= ref [] in
   let aux l s = let t::q=l in match q with
      | [] -> ()
      | t::q0 ->
      if not x.(t) then begin num.(!k+1)<-t; incr k end;
      if not(List.mem (m.(s).(t)) !aretes) then aretes:=(s,t)::(!aretes);
      aux q0
   and aux2 l i = match l with
      | [] -> ()
      | (j,p)::q -> let c=chaine_min pred i j in aux c i in
   for i=0 to m-1 do
      aux2 t1.(i) i
   done;
   let h2=Array.make (!k) [] in
   for i=0 to Array.length h1 do
      h2.(i)<-h1.(i)
   done;
   let trouve t l = match l with
      | [] -> -1
      | (u,p)::q -> if u=t then p else trouve t q in
   let rec aux3 a = match a with
      | [] -> ()
      | (s,t)::q -> let p=trouve t g.(s) in h2.(t)<-(s,p)::h2.(t); h2.(s)<-(t,p)::h2.(t); aux q
   in kruskal_inverse h2;;
```

Q36. L'algorithme de Floyd-Warshall est en $O(|S|^3)$; la création de H_1 est en $O(|X|^2)$; l'algorithme de Kruskal est en $O(|A|^2)$; l'ajoute de chemin de poids minimal est en $O(|A| \times |X|)$. Au total, la complexité est la somme de toutes ces complexités donc en $O(|S|^3 + |A|^2)$.