Option Informatique
Arbres couvrants minimaux

Corrigé

30 novembre 2006

1 Petit rappel sur les graphes

Question 1 *Soit $G = (V, E)$ un graphe non orienté à n sommets. Justifier (rapidement) l'équivalence des assertions suivantes :*

1. *G est connexe et acyclique.*
2. *G est connexe et comporte $n-1$ arêtes reliant des sommets distincts.*
3. *G est acyclique et comporte $n-1$ arêtes reliant des sommets distincts.*
4. *G est acyclique, mais ne le reste pas si on lui ajoute une arête.*
5. *G est connexe, mais ne le reste pas si on lui enlève une arête.*
6. *deux sommets quelconques de G ont reliés par un et un seul chemin élémentaire.*

- $4 \Rightarrow 3$: la graphe a au moins $n-1$ sommmets, sinon il n'est pas connexe, et il suffit d'ajouter une arête vers un sommet isolé. Il a au plus $n-1$ arêtes, sinon on peut le considérer comme un graphe acyclique à $n-1$ arêtes auquel on a rajouté k arêtes, qui ne peut donc plus être acyclique par hypothèse.
- $3 \Rightarrow 2$: Considérons un graphe acyclique à (n-1) arêtes, arête par arête. Chaque nouvelle arête atteint un sommet qui n'a pas été atteint auparavant.
- $2 \Rightarrow 5$: Un argument de dénombrement permet de conclure.
- $5 \Rightarrow 1$: par contrapposée.
- $1 \Rightarrow 4$: par contrapposée, si on enlève une arête (a, b) et que (a, b) restent reliés, en remettant cette arête on obtient un cycle.
- $6 \Leftrightarrow 1$: par définition.

2 Problème de l'arbre couvrant

Question 2 *Soit un graphe $G = (V, E)$ acyclique, et $e \notin E$. Montrer que $G' = (V, E \cup \{e\})$ est acyclique si et seulement si e relie deux composantes connexes de G.*

C'est l'équivalence $\neg 5 \Leftrightarrow \neg 1$ qu'on tire aisément de la question 1.

3 Algorithme de Kruskal

Question 3 *Écrire l'explosion d'un graphe en un ensemble de ses sous-graphes élémentaires. Évaluer sa complexité.*
(`explose : graphe → graphe list`)

```
let explose g = map (function x → Sommets=[x] ; Aretes=[]) g.Sommets ; ;
```

Complexité $O(|E|)$.

Question 4 *Écrire une fonction `fusion` réalisant la fusion par l'arête a de deux graphes g_1 et g_2 d'une forêt (liste de graphes), de type `graphe list → graphe → graphe → arete → graphe list`. Évaluer sa complexité.*

```
let rec fusion foret g1 g2 a =
  let newforet = subtract foret [g1 ; g2]
  in  Sommets = union g1.Sommets g2.Sommets ; Aretes = a : :(union g1.Aretes g2.Aretes) : :newforet ; ;
```

Complexité $O(|E_1| + |E_2| + |\text{foret}|)$.

Question 5 *Écrire* `cherche_arbre`*, de type* `int → graphe list → graphe`*, qui à tout élément* `x` *et à toute forêt* `f` *associe le graphe de* `f` *auquel appartient* `x`*. Évaluer sa complexité.*

```
let rec cherche_arbre x = function
  | [] → failwith "impossible"
  | g : :q →
      if (mem x g.Sommets)
      then g
      else cherche_arbre x q
; ;
```

Complexité $O(|\text{foret}|)$.

Question 6 *Écrire la recherche de* ***la première*** *arête d'une liste reliant deux arbres d'une forêt (*`cherche_areteK : graphe list → arete list → graphe * graphe * arete`*).*

```
let rec cherche_areteK foret = function
  | [] → failwith "Graphe non connexe"
  | (Edge(x,y,_) as a) : :q →
      let a_x = cherche_arbre x foret and a_y = cherche_arbre y foret in
        if (a_x <> a_y)
        then (a_x, a_y, a)
        else cherche_areteK foret q
; ;
```

Complexité $O(|E| \times |\text{foret}|)$.

Question 7 *En déduire une implémentation de la recherche d'arbre couvrant, et évaluer sa complexité. (*`spanningtree : graphe → graphe`*).*

```
let spanningtree g =
  let rec aux ssgraphes aretes = match ssgraphes with
    | [] → failwith "Boulette !"
    | [spantree] → spantree
    | _ →
        let (g1, g2, a) = cherche_areteK ssgraphes aretes"
        in aux (fusion ssgraphes g1 g2 a) aretes
  in aux (explose g) g.Aretes ; ;
```

Complexité $O(|E| \times |V|^2)$.

Question 8 *Écrire une fonction* `tri_aretes` *de type* `arete list → arete list` *effectuant le tri d'une liste d'arêtes par ordre de poids croissant.*

```
let tri_aretes l_aretes =
  sort_sort
    (fun (Edge(_,_,a)) (Edge(_,_,b)) → a <= b)
    l_aretes
; ;
```

Complexité $O(|v| \log_2(|v|))$.

Question 9 *Éxécuter (sur un bout de papier) l'algorithme de Kruskal sur le graphe de la* FIG. *1 pour trouver un arbre couvrant de poids minimal.*

Voir FIG. 1 et 2.

Question 10 *A-t-on unicité de l'arbre couvrant de poids minimal ? Donner un arbre de même poids que celui de la question précédente, mais différant d'une arête de celui que vous avez trouvé.*

Non, il suffit de remplacer l'arête (b, c) par l'arête (a, h).

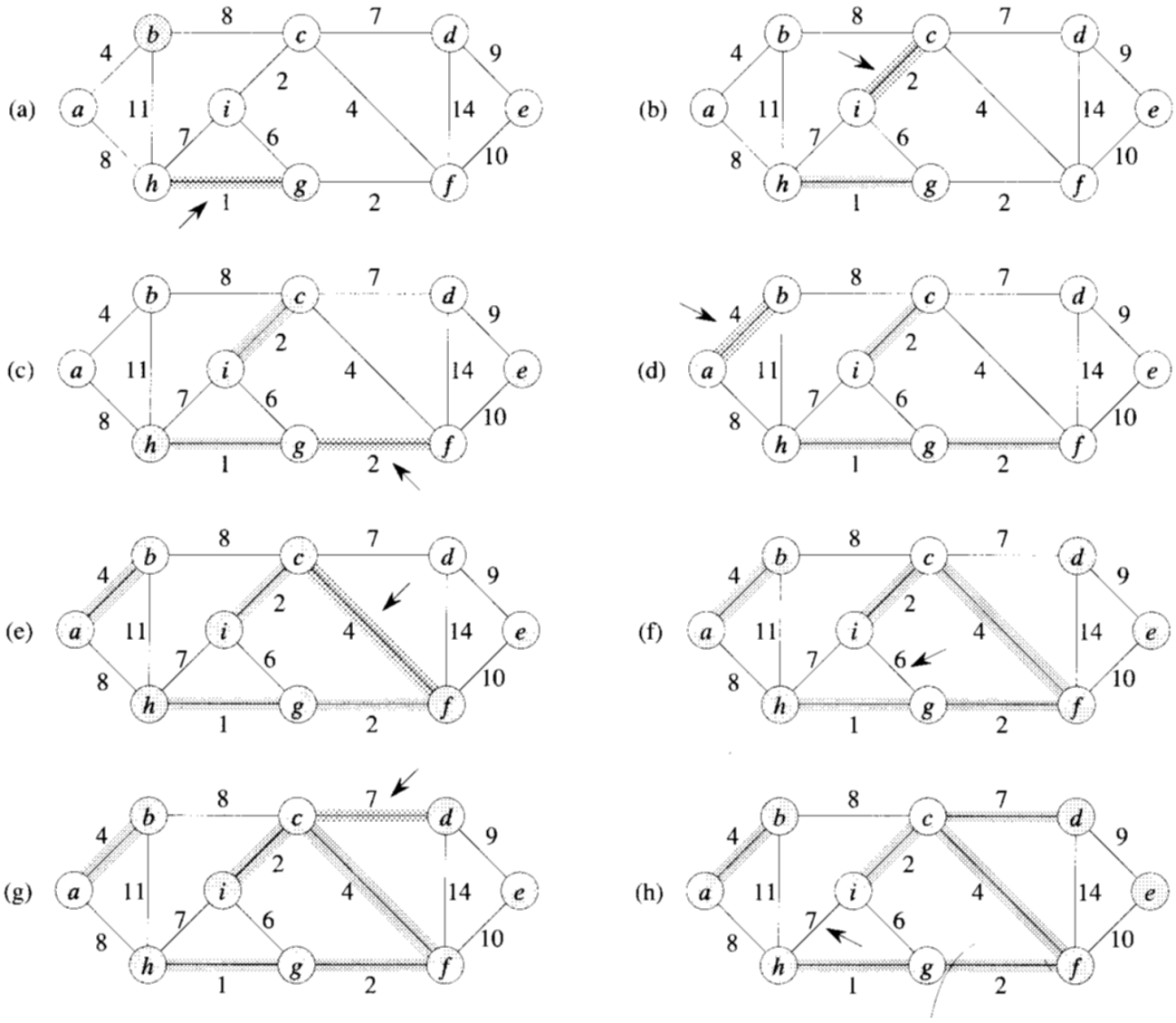

FIG. 1 – Algorithme de Kruskal

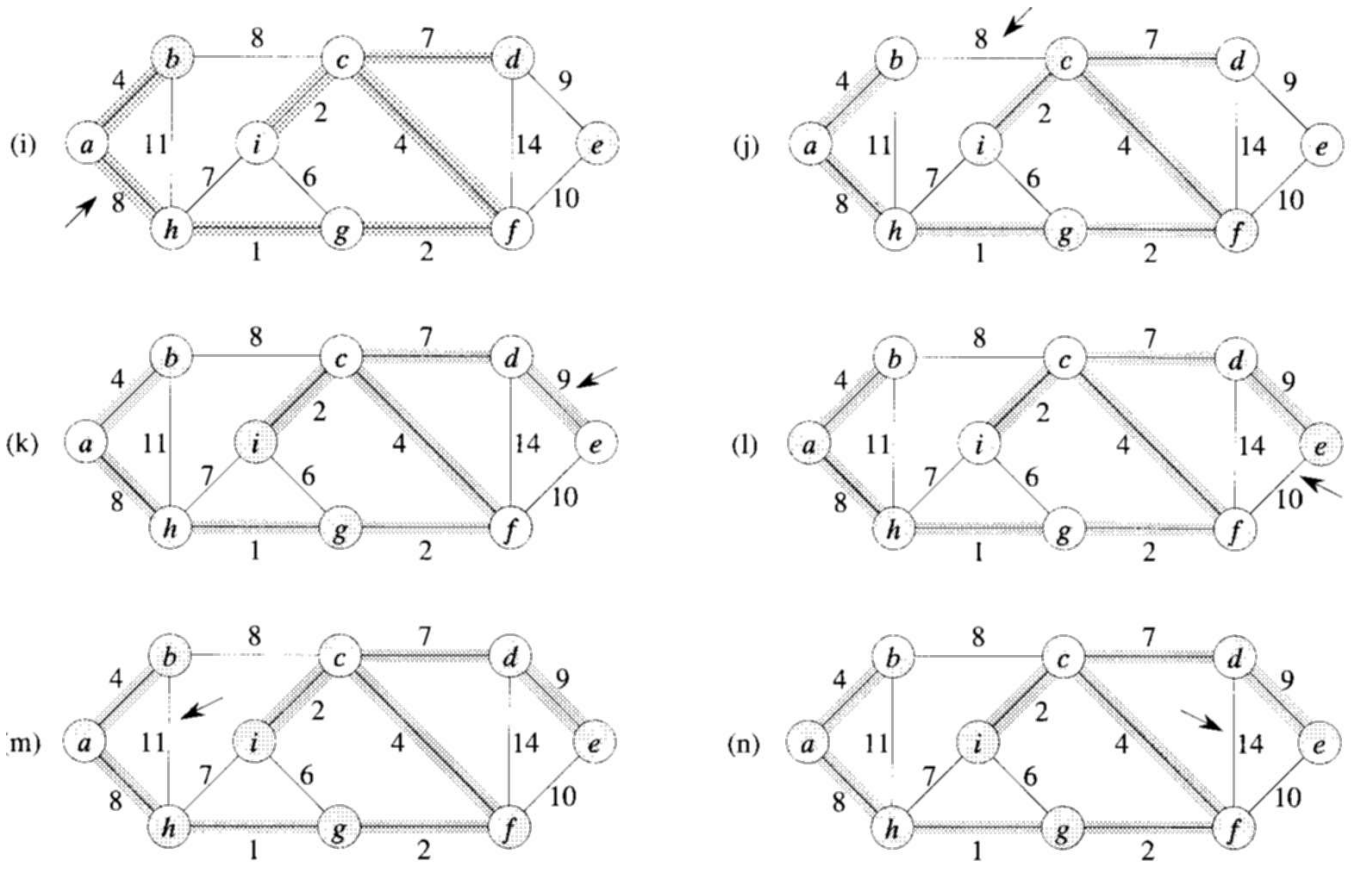

FIG. 2 – Algorithme de Kruskal (suite)

Question 11 *Prouver le lemme suivant :*

> ***Lemme 1*** *Si F_1 et F_2 sont deux sous-arbres couvrants et si $\alpha \in F_1, \alpha \notin F_2$, alors il existe une arête $\beta \in F_2$ telle que $F_1 \backslash \{\alpha\} \cup \{\beta\}$ soit un sous-arbre couvrant.*

Proof: Notons $\alpha = x, y$. $F_1 \backslash \{\alpha\}$ définit deux composantes connexe simples (celle de x, celle de y). Dans le chemin simple reliant x à y dans F_2, on peut choisir une arête β reliant un sommet de la composante connexe de x à un sommet de celle de y. On constate que β convient. □

Question 12 *En déduire une preuve de la validité de l'algorithme de Kruskal.*

Il est clair que l'algorithme de Kruskal termine et fournit un sous-arbre couvrant. Prouvons qu'il est minimal.

Proof: Soit F le résultat de l'algorithme et U une solution minimale ayant le plus d'arêtes communes avec F. U et F ont même cardinal. Si $U \neq F$, on prend $\alpha \in U \backslash F$ de telle sorte que $v(\alpha)$ soit minimal. Le lemme donne une arête β de F, $U' = U \backslash \{\alpha\} \cup \{\beta\}$ est un arbre couvrant. Toutes les arêtes de U moins lourdes que α sont dans F par définition de α. La construction de l'algorithme donne alors : $v(\beta) \leq v(\alpha)$ (sinon, α aurait été traitée avant β et aurait été acceptée). Ainsi U' est-il encore minimal, et moins différent de F que U, ce qui contredit sa définition. □

Question 13 *Modifier le programme de la question 7 pour qu'il donne un arbre couvrant de poids minimal. Comment évolue la complexité ?*
(`spanningtree_Kruskal : graphe → graphe`).

```
let spanningtree_Kruskal g =
  ...
  in aux (explose g) (tri_aretes g.Aretes) ; ;
```

4 Minimalité

Question 14 *Montrer que si les poids des arêtes sont distincts deux à deux, il n'existe qu'un seul arbre recouvrant minimal.*

Soit deux arbres minimaux F et F' différant au moins sur l'arête $\alpha \in F \backslash F'$. Alors, d'après la question 11, il existe une arête β de F' qui relie les deux composantes connexes que reliait α dans F. Supposons sans perte de généralité $v(\alpha) < v(\beta)$. Alors $F' \backslash \{\beta\} \cup \{alpha\}$ est couvrant de poids strictement inférieur à F', contradiction.

Question 15 *Soit $H = (F, W)$ un arbre recouvrant minimal de $G = (E, V)$. Montrer qu'il existe $a \in W$ et $b \in E \backslash W$ telles que $(F, V \cup \{b\} \backslash \{a\})$ soit un arbre recouvrant de G de poids minimal parmi les arbres recouvrants qui ne sont pas de poids minimal.* On supposera dans la fin de cette section que les arêtes sont de poids deux à deux distincts.

Il suffit de considérer pour b l'arête de poids minimal parmi celles qui relient les mêmes composantes connexes que a. La question 11 permet de conclure.

Question 16 *Déduire de ce qui précède un algorithme construisant un arbre G de poids minimal parmi les non minimaux. Complexité de cet algorithme ?*

On remarque qu'il s'agit simplement d'appliquer l'algorithme construisant l'arbre minimal dans le graphe ne comportant pas l'arête considérée.

```
let non_minimal g a = (spanningtree_Kruskal Sommets = g.Sommets ;
  Aretes = subtract g.Aretes a) ; ;
```

Question 17 *A-t-on unicité de l'arbre de poids minimal parmi les non-minimaux ?*

Oui, puisque les poids des arêtes sont deux à deux distincts, et d'après la quesiton 14.

5 Algorithme de Prim

Question 18 *Exécuter (à la main) l'algorithme de Prim sur le graphe de la* FIG. *1.*

Voir la FIG. 3.

Question 19 *Démontrer que l'algorithme de Prim renvoie bien un arbre couvrant minimal.*

Il est clair que l'algorithme de Prim termine et fournit un sous-arbre couvrant. Prouvons qu'il est minimal.

Proof: Soit U une solution minimale telle que le nombre d'étapes de l'algorithme pendant lesquelles l'arbre en cours de construction est un sous-arbre de U soit maximale. Considérons précisement cette étape où l'algorithme rajoute à $F \subset U$ une arête $\alpha = x, y$ qui n'est pas dans U, x est dans le graphe défini par F et non y. Il existe un chemin reliant x à y dans U. Soit β la première arête traversée par ce chemin, qui sort de F. On a $v(\alpha) \leq v(\beta)$ par définition de α dans l'algorithme. Remplaçons β par α dans U pour obtenir U'. U' est encore un arbre-couvrant minimal (supprimer β sépare U en deux composantes connexes que α réunit). Cela contredit la définition de U. □

Question 20 *Implémenter l'algorithme de Prim. On écrira tout d'abord une fonction* `poids` *qui renvoie le poids d'un graphe passé en paramètre, puis une fonction* `cherche_areteP` *qui prend en arguments une liste de sommets et une liste d'arêtes, et qui renvoie la première arête de la liste qui ne relie pas deux sommets cette première liste.*

```
let poids g =
  let rec aux = function
    | [] → 0
    |(Edge(_,_,p)) : :q → p + aux q
  in aux (g.Aretes)
; ;

let rec cherche_areteP sommets = function
  | [] → failwith "Graphe non connexe"
  | (Edge(x,y,_) as a) : :q →
     let tmpx = mem x sommets and tmpy = mem y sommets in
       if ((tmpx && (not tmpy)) || ((not tmpx) && tmpy))
       then a
       else cherche_areteP sommets q
; ;

let spanningtree_Prim g =
  let rec aux sommets aretes spantree = function
    | [] →  Sommets = sommets ; Aretes = spantree
    | reste →
        let (Edge(x,y,_) as a) = cherche_areteP sommets aretes
        in aux (union sommets [x ;y]) aretes (a : :spantree) (subtract reste [x ;y])
        in let deb = g.Sommets.(0)
        in aux [deb] (tri_aretes g.Aretes) [] (subtract (g.Sommets) [deb])
; ;
```

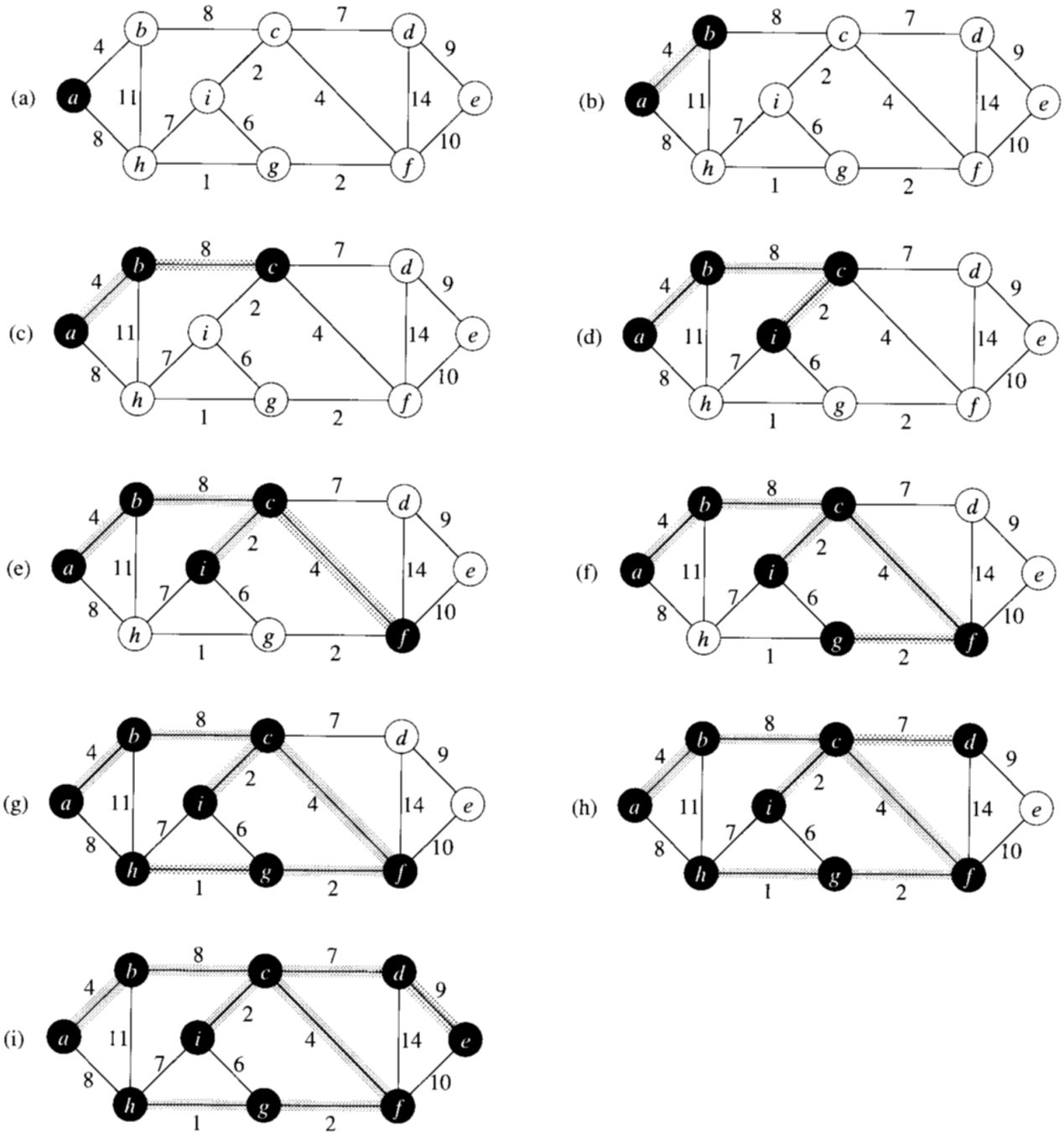

FIG. 3 – Algorithme de Prim