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])
; ;
```

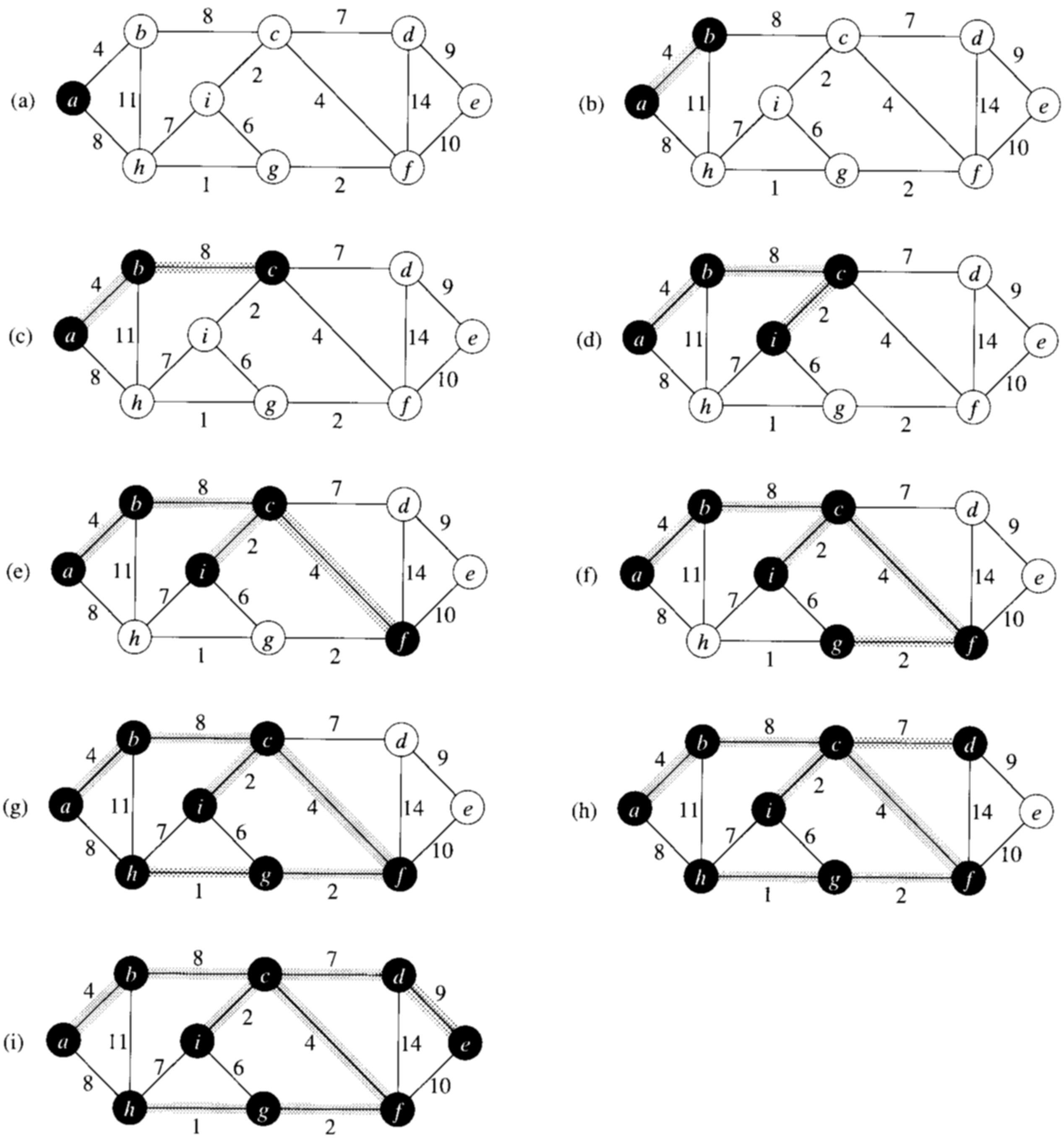

FIG. 3 – Algorithme de Prim