# Arbres Couvrants de Poids Minimum
Correction

***Solution 2* (Algorithme de Prim)**

7. Les opérations sur les tas :
   — `tas_vide ()` : retourne un nouveau tas.
   — `est_vide (`$T$`)` : indique si $T$ est vide.
   — `maj (`$T$`, `$c$`, `$ps$`)` : ajoute au tas $T$ le sommet pointé par $ps$ de cout $c$.
   — `supp_min (`$T$`)` : retourne le sommet de coût minimum supprimé de $T$.

   **Spécifications :**
   La procédure `Prim (t_graph_dyn` $G$`, t_vect_entiers` $res$`)` construit un ARPM de $G$ (s'il est connexe) sous la forme du vecteur de pères $res$.

```
algorithme procedure prim
   parametres locaux
      t_graph_dyn G
   parametres globaux
      t_vect_entiers res

   variables
      t_vect_reels cout
      t_tas h
      t_listsom ps
      t_listadj pa
      entier s, sa
debut
   pour s ← 1 jusqu'a g.ordre faire
      res[s] ← 0     /* inutile si le graphe est connexe */
      cout[s] ← ∞
   fin pour
   res[G.lsom↑.som] ← -1
   cout[G.lsom↑.som] ← 0

   h ← tas_vide()
   maj(h, 0, G.lsom)

   faire
      ps ← supp_min(h)
      s ← ps↑.som
      cout[s] ← - cout[s]      /* marque les sommets traités */
      pa ← ps↑.succ
      tant que (pa <> NUL) faire
         sa ← pa↑.vsom↑.som
         si cout[sa] > pa↑.cout alors
            cout[sa] ← pa↑.cout
            res[sa] ← src
            maj(h, pa↑.cout, pa↑.vsom)
         fin si
         pa ← pa↑.suiv
      fin tant que
   tant que non est_vide(h)

fin algorithme procedure prim
```

*Remarque :* S'il reste des sommets non "marqués" à la fin de l'algorithme, alors le graphe n'était pas connexe.

***Solution 3*** **(Algorithme de Kruskal)**

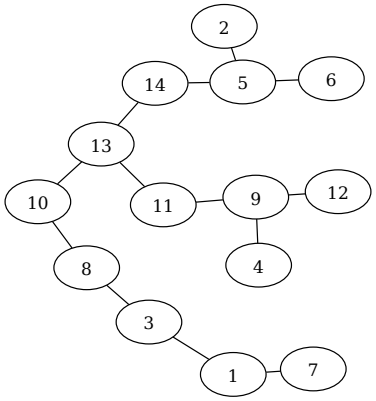


Figure 1 – ARPM produit par Kruskal

5. Voir la figure 1

7. Le tas contiendra des arêtes :

```
types
   t_arete = enregistrement
      entier   src, dst
      reel     cout
   fin enregistrement t_arete
```

Les opérations sur les tas :
— `tas_vide ()` : retourne un nouveau tas.
— `est_vide (`$T$`)` : indique si $T$ est vide.
— `ajout (`$T$`, `$a$`)` : ajoute au tas $T$ l'arête $a$.
— `supp_min (`$T$`)` : retourne l'arête de coût minimum supprimé de $T$.

```
algorithme fonction trouver : entier
  parametres locaux
    t_vect_entiers cc
    entier s

debut
  tant que (cc[s] <> s) faire
    s ← cc[s]
  fin tant que
  retourne (s)
fin algorithme fonction trouver


algorithme fonction reunir : booleen      /* indique si x et y ont été réunis */
  parametres locaux
    entier x,  y
  parametres globaux
    t_vect_entiers cc

  variables
    entier rx,  ry
debut
  rx ← trouver (cc, x)
  ry ← trouver (cc, y)
  si rx <> ry alors
```

```
        cc[ry] ← rx
      fin si
      retourne (rx <> ry)
    fin algorithme fonction reunir

algorithme procedure kruskal
  parametres locaux
    t_graph_dyn G
  parametres globaux
    t_mat_entiers T

  variables
    tas aretes
    t_vect_entiers cc
    entier s, sa, nba
    t_listsom ps
    t_listadj pa
    t_arete a

debut
  aretes ← tas_vide()
  ps ← G.lsom
  tant que ps <> NUL faire
    s ← ps↑.som
    cc[s] ← s
    pour sa ← 1 jusqu'a G.ordre faire
      T[s, sa] ← 0
    fin pour

    a.src ← s
    pa ← ps↑.succ
    tant que pa <> NUL faire
      a.dst ← pa↑.vsom↑.som
      si a.scr < a.dst alors
        a.cout ← pa↑.cout
        ajout (aretes, a)
      fin si
      pa ← pa↑.suiv
    fin tant que

    ps ← ps↑.suiv
  fin tant que

  nba ← 0
  tant que nba < g.ordre - 1 faire
    a ← supp_min (aretes)
    si reunir (a.src, a.dst, cc) alors
      T[a.src, a.dst] ← 1
      T[a.dst, a.src] ← 1
      nba ← nba + 1
    fin si
  fin tant que

fin algorithme procedure kruskal
```

*Remarque :* si le graphe peut ne pas être connexe, il faut ajouter un test **non** `est_vide(aretes)` à la boucle qui construit la solution. Il suffira alors de tester `nba = G.ordre-1` en sortie de boucle pour vérifier la connexité.