ALGORITHMIQUE DES GRAPHES - PARCOURS, COUVERTURE - INGENIEURS 1 - EISTI

Ecole Internationale des Sciences du Traitement de l'Information

7 avril 2009

Type Abstrait Arête

```
Type abstrait AreteValuee
Concept
Ce type permet de modéliser les arêtes valuées d'un graphe
On représente les sommets par des numéros entre 1 et n où n
est le nombre de sommets.

Opérations de base
Constructeur AreteValuee : creerAreteValuee(Entier o, Entier d, Reel val
    ) : AreteValuee
Observateur AreteValuee : recValuation() : Reel
Observateur AreteValuee : recOrigine() : Entier
Observateur AreteValuee : recDestination() : Entier

Axiomes
estEgal(recOrigine(creerAreteValuee(o,d,val)),o)
estEgal(recDestination(creerAreteValuee(o,d,val)),d)
estEgal(recValuation(creerAreteValuee(o,d,val)),val)
```

Type Abstrait Graphe

```
Type abstrait Graphe
Concept
Ce type permet de modéliser les graphes.
Les sommets sont numérotés de 1 à n.

Opérations de base
Constructeur Graphe : creerGraphe(Entier nbSommets) : Graphe
Transformateur Graphe : ajouterArete(AreteValuee a) : Graphe
Transformateur Graphe : marquer(Entier noS) : Graphe
Transformateur Graphe : demarquer(Entier noS) : Graphe

Observateur Graphe : estMarque(Entier noS) : Booleen
Observateur Graphe : recAretes() : Vecteur
Observateur Graphe : recNbSommets() : Entier
Observateur Graphe : recNbAretes() : Entier
Observateur Graphe : recArete(Entier noSD, Entier noSA) : AreteValuee

Pré-conditions
definie(creerGraphe(nb)) ⟹ nb > 0
definie(ajouterArete(g,a)) ⟹ recOrigine(a) ≥ 1 ET recOrigine(a) ≤
    recNbSommets(g)
definie(ajouterArete(g,a)) ⟹ recDestination(a) ≥ 1 ET recDestination(
    a) ≤ recNbSommets(g)
definie(marquer(g,noS)) ⟹ noS ≥ 1 ET noS ≤ recNbSommets(g)
definie(demarquer(g,noS)) ⟹ noS ≥ 1 ET noS ≤ recNbSommets(g)


Axiomes
estEgal(recNbSommets(creerGraphe(nb)),nb)
estEgal(recNbAretes(creerGraphe(nb)),0)
```

Parcours en profondeur

```
// Parcourir en profondeur de toutes les composantes connexes d'un
      graphe
PPG(Graphe g)
DEBUT
    on démarque tous les sommets
    POUR chaque sommet s de g
          SI s n'est pas marqué ALORS
             PPGRec(g,s)
          FINSI
    FINPOUR
FIN

// Parcourir en profondeur d'un graphe g à partir d'un sommet s
PPGRec (graphe g, sommet s)
DEBUT
    marquer(s)
    POUR CHAQUE élément sfils successeurs de s FAIRE
          SI nonMarqué(sfils) ALORS
             PPGRec(g,sfils)
          FINSI
    FINPOUR
FIN
```

Parcours en largeur

```
// Parcourir en largeur d'un graphe g à partir d'un sommet s
PLG(Graphe g,Sommet s)
DEBUT
   on crée une file vide f
   on démarque tous les sommets de g
   on enfile s dans f
   on marque s
   TANTQUE f n'est pas vide FAIRE
     x = sommet de f
     on défile f
     POUR tous les successeurs y de x non marqués
          on enfile y dans f
          on marque y
     FINPOUR
   FINTANTQUE
FIN
```

Arbre couvrant de poids minimum

Motivations

Arbre couvrant de poids minimum

Motivations

1. Données : graphe non orienté avec arêtes valuées (coûts de transport, probabilités, poids, longueurs)

Arbre couvrant de poids minimum

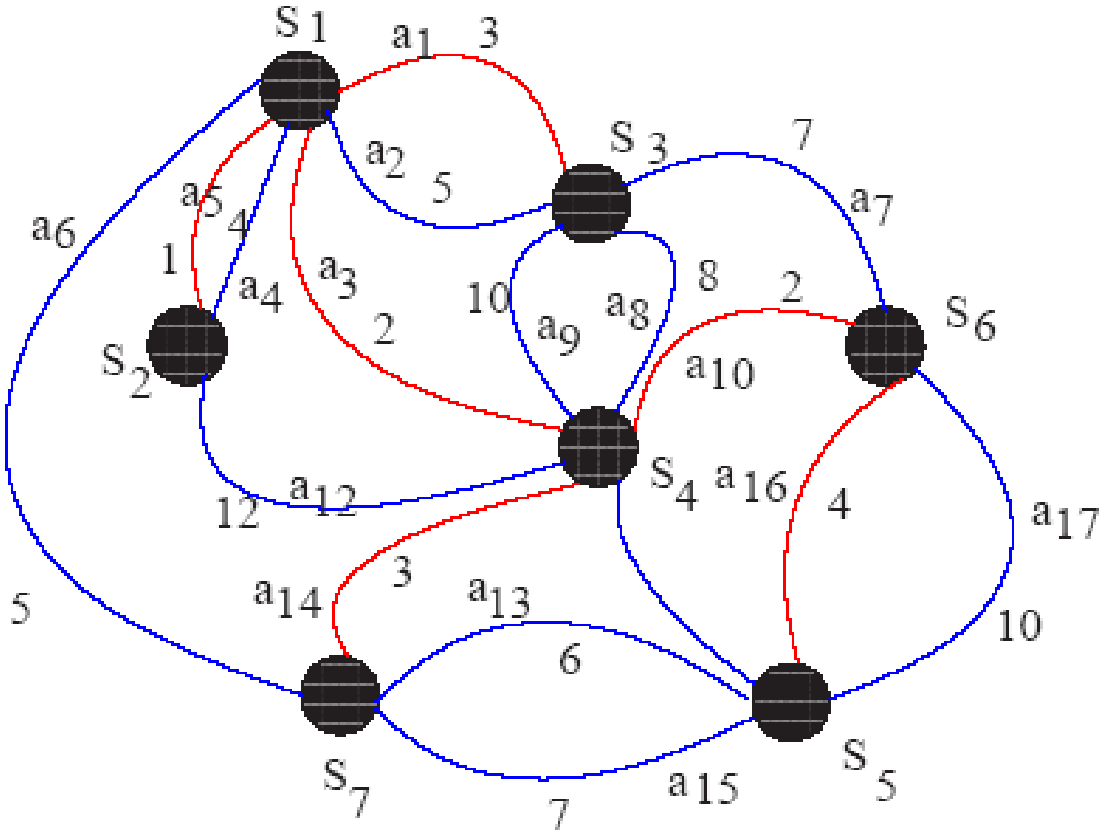
Motivations

1. Données : graphe non orienté avec arêtes valuées (coûts de transport, probabilités, poids, longueurs)
2. Objectif : trouver un arbre partiel de coût minimum avec les mêmes sommets que le graphe initial.

Arbre couvrant de poids minimum

Motivations

1. Données : graphe non orienté avec arêtes valuées (coûts de transport, probabilités, poids, longueurs)
2. Objectif : trouver un arbre partiel de coût minimum avec les mêmes sommets que le graphe initial.



Algorithme de Kruskal

Principes de l'algorithme

Algorithme de Kruskal

Principes de l'algorithme

1. Poser les n sommets d'un graphe connexe

Algorithme de Kruskal

Principes de l'algorithme

1. Poser les n sommets d'un graphe connexe
2. Choisir une arête de poids minimum

Algorithme de Kruskal

Principes de l'algorithme

1. Poser les n sommets d'un graphe connexe
2. Choisir une arête de poids minimum
3. Répéter la procédure parmi les arêtes restantes
   - de poids minimum
   - ne faisant pas de boucles avec les précédentes

Etapes de l'algorithme

Algorithme de Kruskal

Principes de l'algorithme

1. Poser les n sommets d'un graphe connexe
2. Choisir une arête de poids minimum
3. Répéter la procédure parmi les arêtes restantes
   - de poids minimum
   - ne faisant pas de boucles avec les précédentes

Etapes de l'algorithme

1. Trier les arêtes par ordre croissant

Algorithme de Kruskal

Principes de l'algorithme

1. Poser les n sommets d'un graphe connexe
2. Choisir une arête de poids minimum
3. Répéter la procédure parmi les arêtes restantes
   - de poids minimum
   - ne faisant pas de boucles avec les précédentes

Etapes de l'algorithme

1. Trier les arêtes par ordre croissant
2. Si on n'a pas sélectionné $n-1$ arêtes, considérer la première non sélectionnée (ordre du tri) ; tester si elle fait une boucle, la sélectionner ou la rejeter.

Implémentation de l'algorithme de Kruskal

Comment déterminer si une arête forme un cycle ?

Implémentation de l'algorithme de Kruskal

Comment déterminer si une arête forme un cycle ?

Composantes connexes

Implémentation de l'algorithme de Kruskal

Comment déterminer si une arête forme un cycle ?

Composantes connexes

1. Au début chaque sommet constitue une composante connexe (pas d'arêtes)

Implémentation de l'algorithme de Kruskal

Comment déterminer si une arête forme un cycle ?

Composantes connexes

1. Au début chaque sommet constitue une composante connexe (pas d'arêtes)
2. On associe un indice i à chaque sommet. Pour une arête $\{x, y\}$, on compare les indices de x et y.
   - si ils sont égaux $\longrightarrow$ cycle
   - sinon indice de y = indice de x

Implémentation de l'algorithme de Kruskal

Comment déterminer si une arête forme un cycle ?

Composantes connexes

1. Au début chaque sommet constitue une composante connexe (pas d'arêtes)
2. On associe un indice i à chaque sommet. Pour une arête $\{x, y\}$, on compare les indices de x et y.
   - si ils sont égaux $\longrightarrow$ cycle
   - sinon indice de y = indice de x
3. On construit un tableau d'entiers de 1 à n (cc) pour stocker composantes connexes.

Implémentation de l'algorithme de Kruskal

Comment déterminer si une arête forme un cycle ?

Composantes connexes

1. Au début chaque sommet constitue une composante connexe (pas d'arêtes)
2. On associe un indice i à chaque sommet. Pour une arête $\{x, y\}$, on compare les indices de x et y.
   - si ils sont égaux $\longrightarrow$ cycle
   - sinon indice de y = indice de x
3. On construit un tableau d'entiers de 1 à n (cc) pour stocker composantes connexes.
4. On construit un tableau qui représente le graphe par ses composantes connexes (ordre de poids croissant).

Implémentation de l'algorithme de Kruskal

Comment déterminer si une arête forme un cycle ?

Composantes connexes

1. Au début chaque sommet constitue une composante connexe (pas d'arêtes)
2. On associe un indice i à chaque sommet. Pour une arête $\{x, y\}$, on compare les indices de x et y.
   - si ils sont égaux $\longrightarrow$ cycle
   - sinon indice de y = indice de x
3. On construit un tableau d'entiers de 1 à n (cc) pour stocker composantes connexes.
4. On construit un tableau qui représente le graphe par ses composantes connexes (ordre de poids croissant).
5. Le résultat final est un tableau correspondant aux $n-1$ arêtes du graphe (arbre de recouvrement de poids minimum).

Exemple de couverture

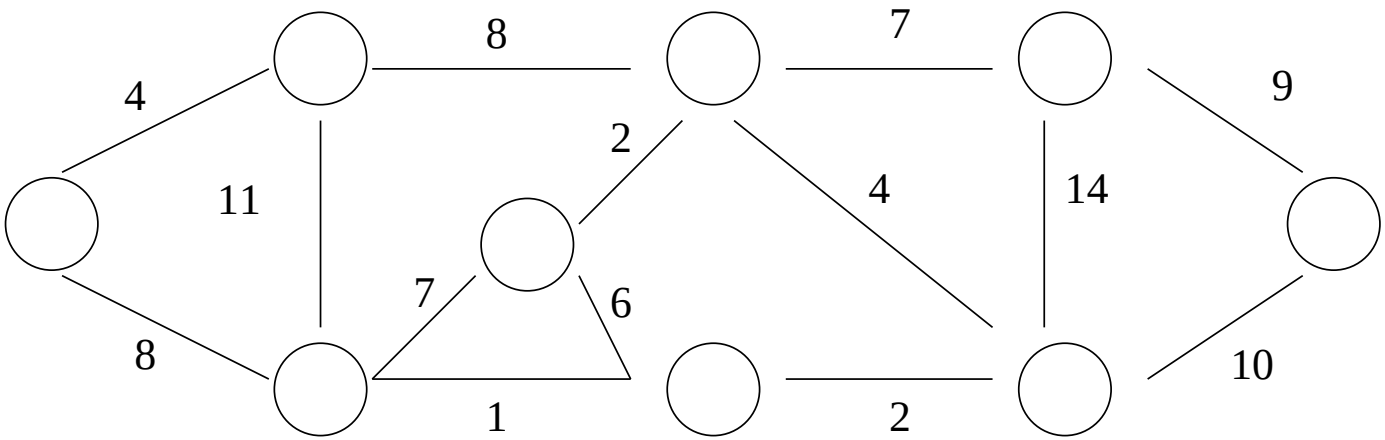


FIG.: Algorithme de Kruskal

Exemple de couverture

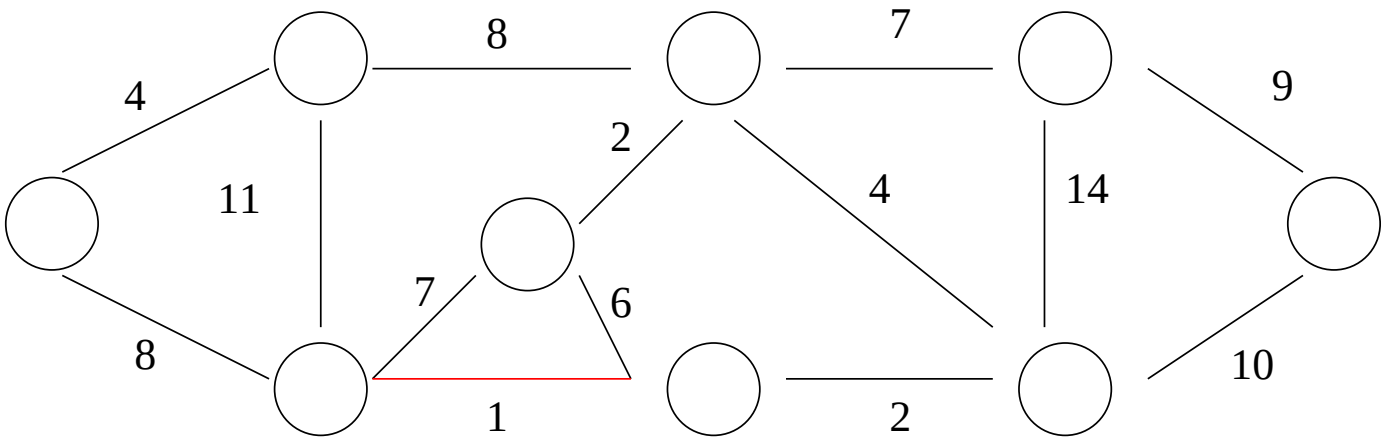


FIG.: Algorithme de Kruskal

Exemple de couverture

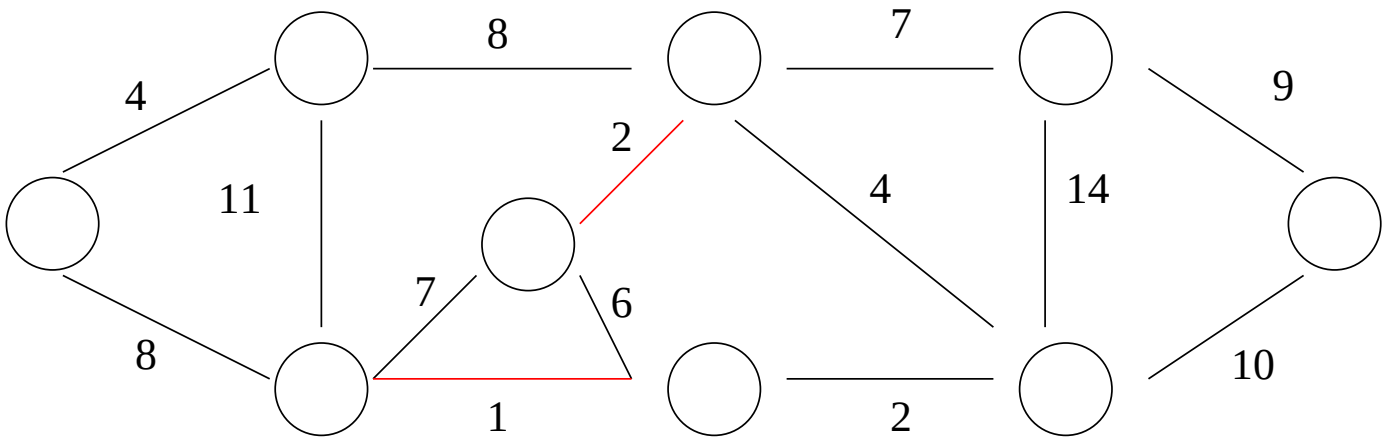


FIG.: Algorithme de Kruskal

Exemple de couverture

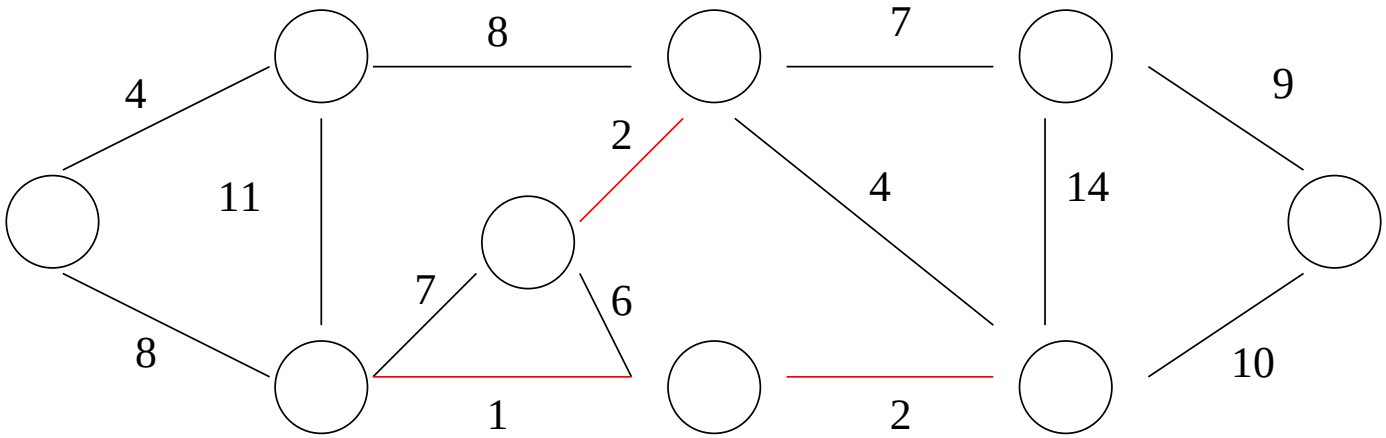


FIG.: Algorithme de Kruskal

Exemple de couverture

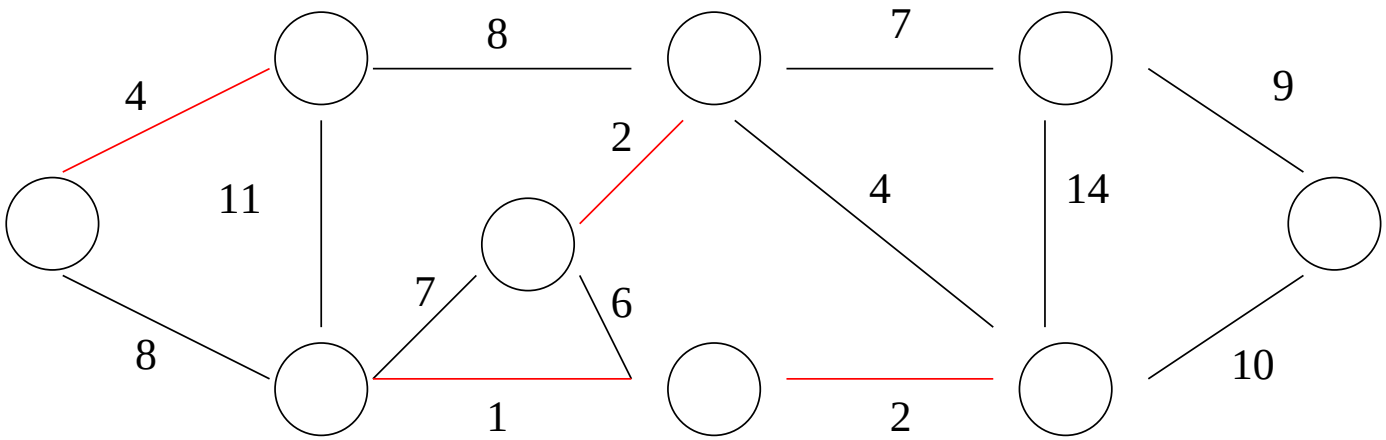


FIG.: Algorithme de Kruskal

Exemple de couverture

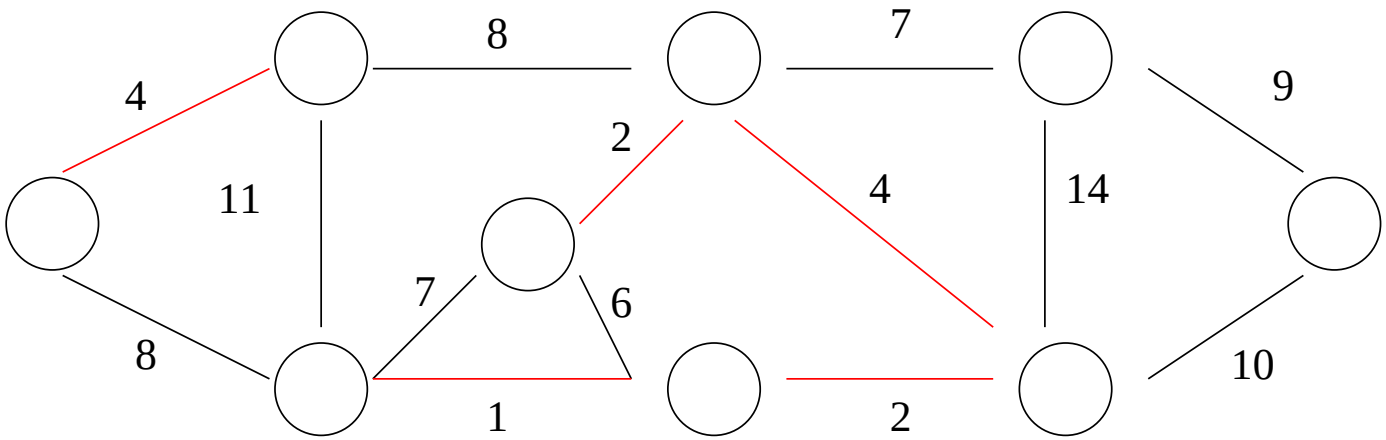


FIG.: Algorithme de Kruskal

Exemple de couverture

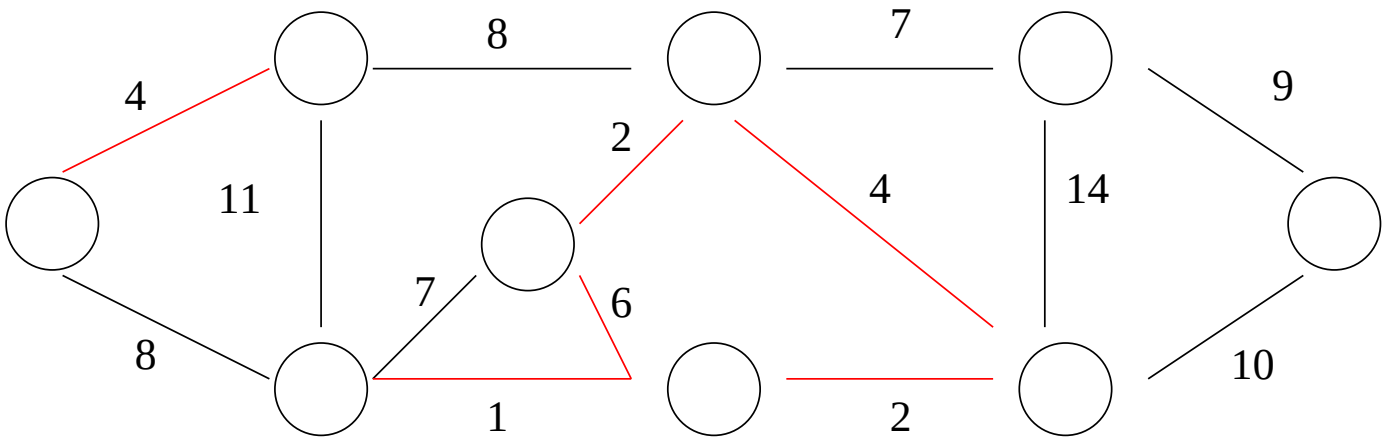


FIG.: Algorithme de Kruskal

Exemple de couverture

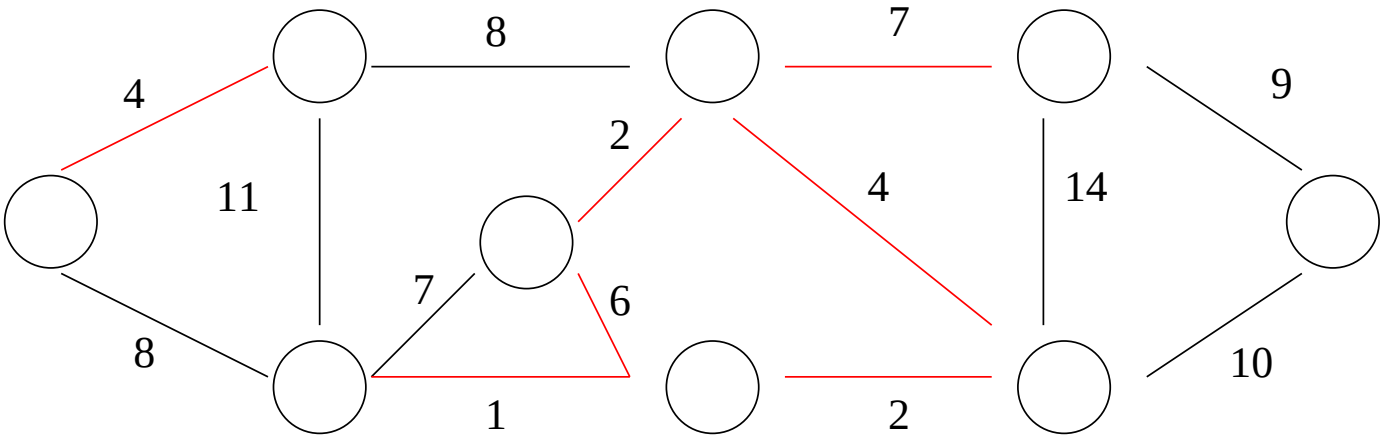


FIG.: Algorithme de Kruskal

Exemple de couverture

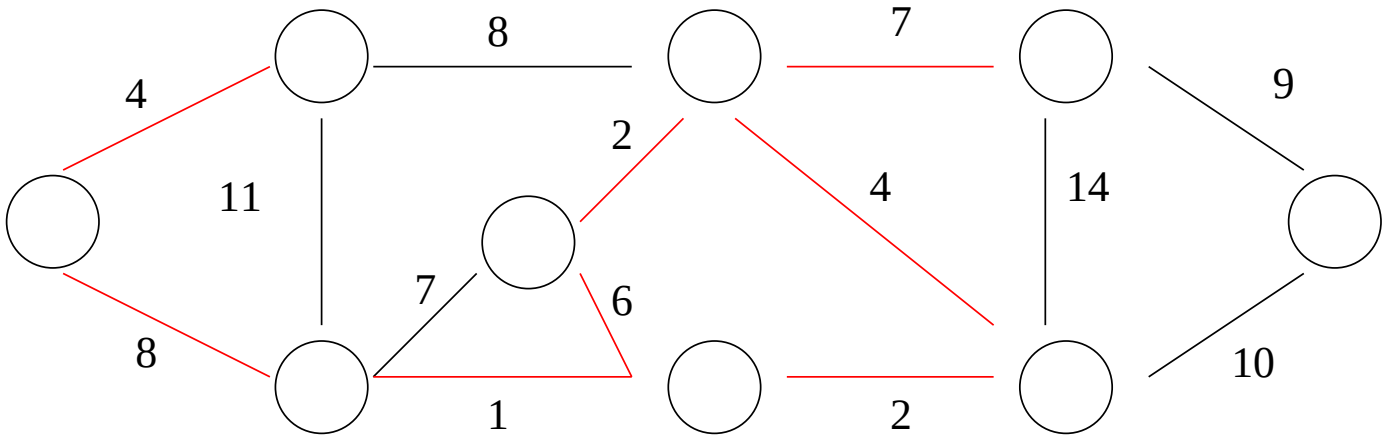


FIG.: Algorithme de Kruskal

Exemple de couverture

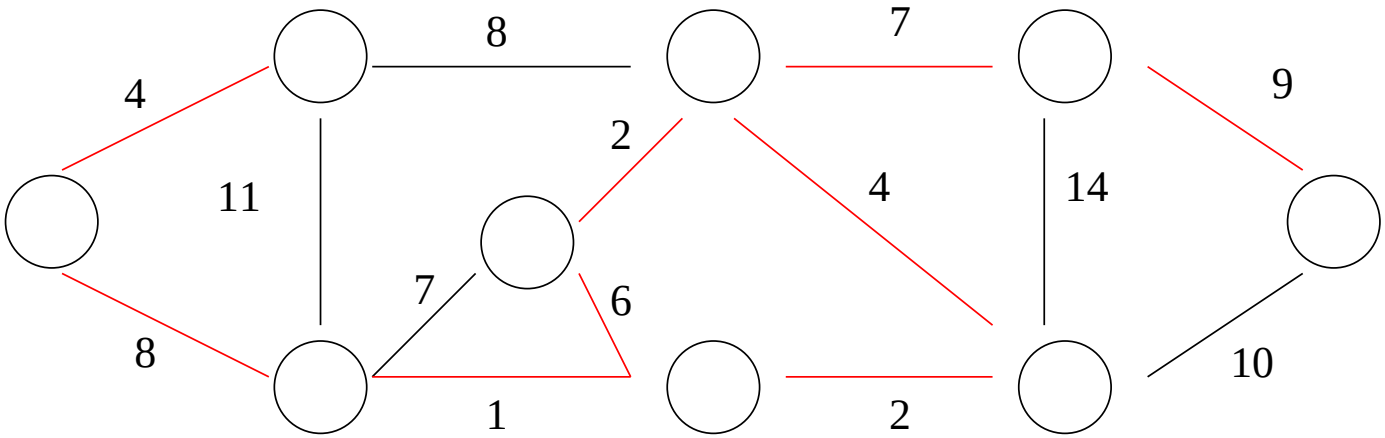


FIG.: Algorithme de Kruskal

Complexité de l'algorithme

Complexité

$$\theta(n^2 + m\ log_2(m))$$

$$m = |A|$$

$$n = |S|$$

Applications des algorithmes

Réseaux de communications ou d'interconnexions

- Réalisation d'un réseau connexe de coût minimum, dont on connaît le coût des liaisons directes des paires d'objets
- Structure d'un arbre optimale car coût positifs

Applications des algorithmes

Réseaux de communications ou d'interconnexions

- Réalisation d'un réseau connexe de coût minimum, dont on connaît le coût des liaisons directes des paires d'objets
- Structure d'un arbre optimale car coût positifs

Applicaton en traitement de l'image

- Image N&B = réseau de pixels ($512x512$)
- Chaque image : 1 niveau de gris et 8 voisins
- Problème : Déterminer des parties connexes ayant des points de niveaux de gris très voisins.