Université de Msila
Faculté Mathématiques et Informatique
Département d'Informatique

Cours de Théorie des graphes

2eme année Informatique

Dr Nasser Eddine MOUHOUB

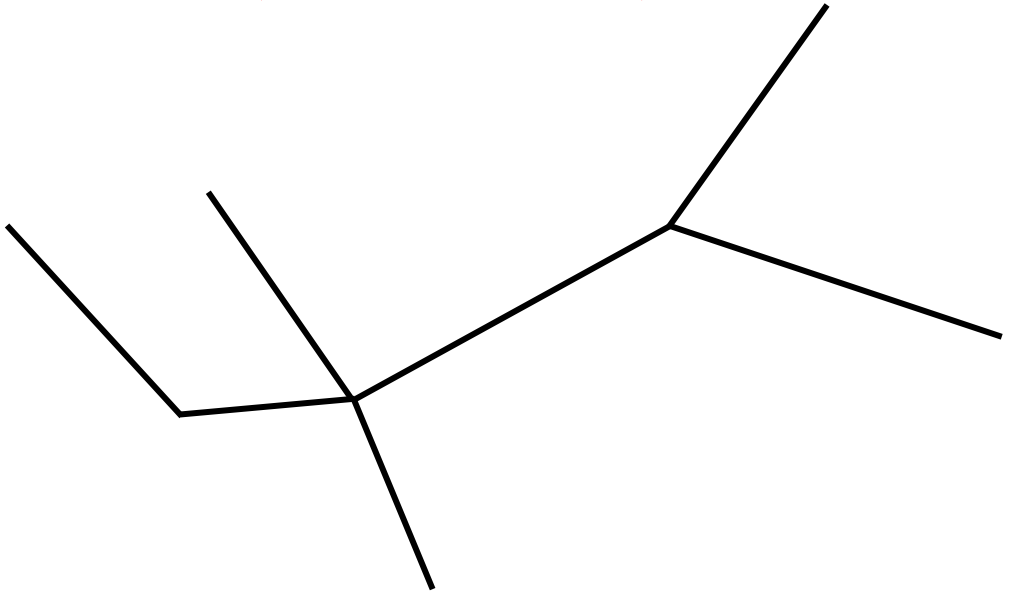
2015 / 2016

Chapitre III

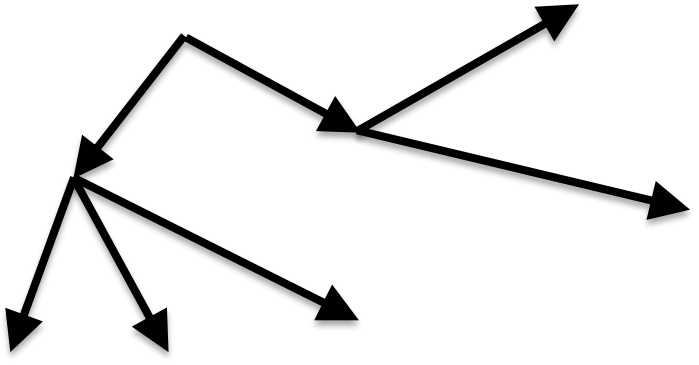
Les arbres

Les arbres

Un arbre (non orienté) !

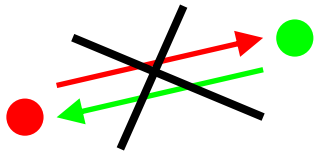


Une arborescence (orientée) !

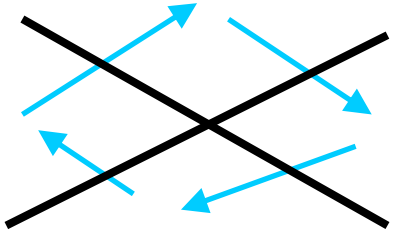


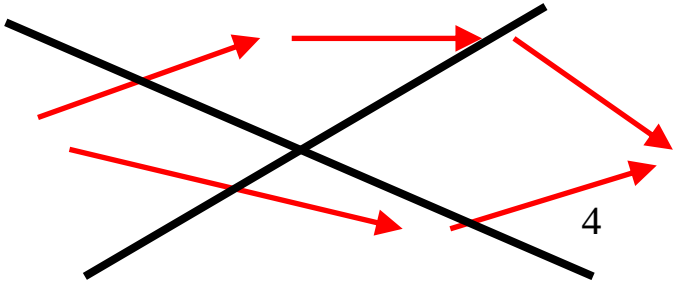
Les arbres

- **Définitions :**
  - **Un arbre est un graphe non orienté dans lequel il existe une et une seule chaine entre toute paire de sommets.**
    - **Cette chaine sera donc simple, la plus courte, . . .**
  - **Une arborescence est un graphe orienté quasi-fortement connexe tel qu'il existe un et un seul chemin orienté de la racine vers tout autre sommet.**
    - D'abord, il n'y a qu'une seule racine !
    - On n'a pas de chemins multiples, ni de circuits !



Les arbres d'algorithmique sont des arborescences !





- **Théorème**
- Les définitions suivantes sont équivalentes: pour tout graphe *G = (V, E) à n sommets.*

  1. *G est un arbre,*
  2. *G est connexe et sans cycles,*
  3. *G est connexe et comporte n-1 arêtes,*
  4. *G est sans cycles et comporte n-1 arêtes,*
  5. chaque paire *{u, v} de sommets distincts est reliée par une seule chaîne simple (et le graphe est sans boucles).*

Quelques définitions des arbres

Définition 6:

Un graphe est connexe minimal s'il est connexe et n'a pas plus d'arêtes qu'aucun autre graphe connexe !

L'idée: si nous enlevons une arête de la chaine unique, nous cassons la connexité!

Définition 7:

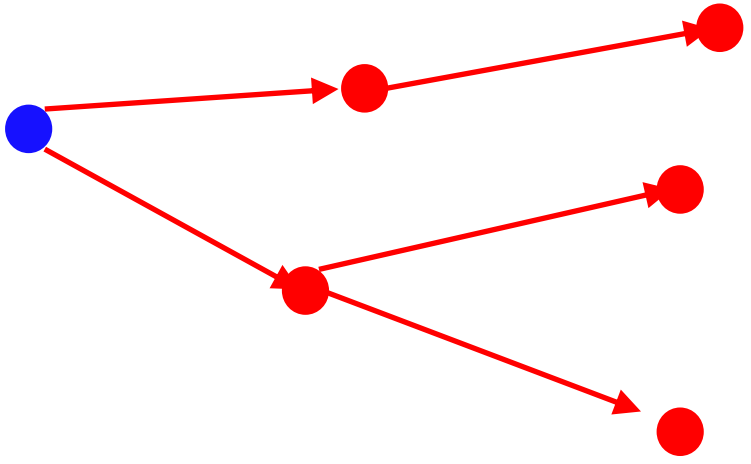
Un graphe est connexe minimal s'il est connexe et comporte n-1 arêtes.

Définition 8:

Un graphe est sans cycles, maximal s'il est sans cycles et n'a pas moins d'arêtes qu'aucun autre graphe sans cycles !

Les arborescences

- **Une arborescence peut être caractérisée comme suit.**
  - **Elle possède (n-1) arcs.**
  - **Tous les sommets sauf un ont un degré entrant unitaire.**
  - **Un seul sommet a un degré entrant nul.**



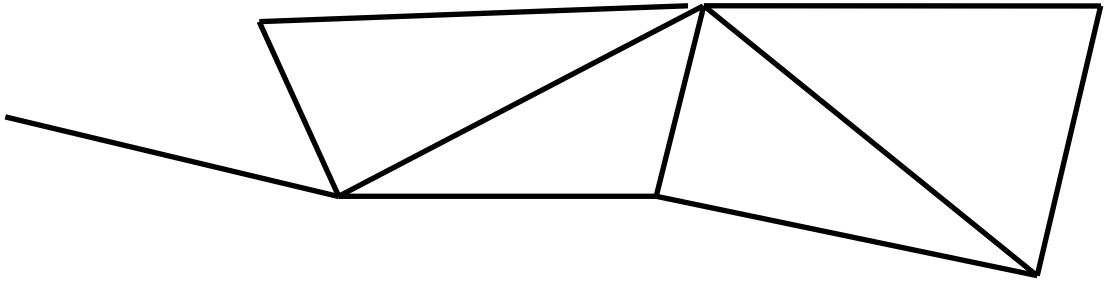
Les arborescences

- **Toute arborescence peut être transformée en un arbre !**
  - **Il suffit de changer les arcs en arêtes.**
  - **Nous aurons (n-1) arêtes.**

- **Tout arbre peut être transformé en une arborescence en nous laissant le choix de la racine !**

Les arbres de recouvrement

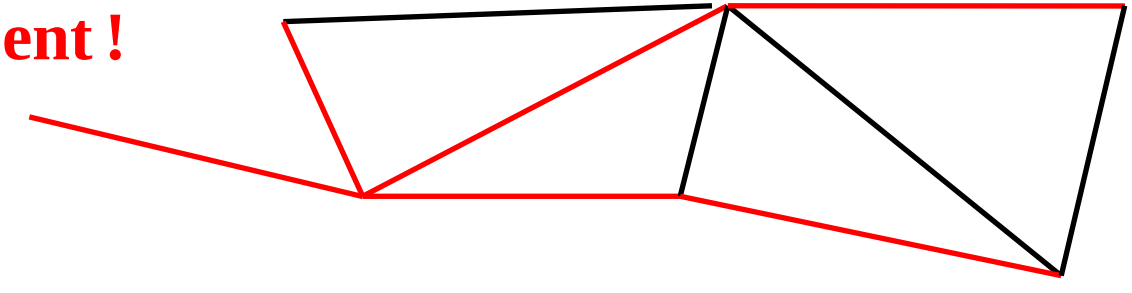
- **Un arbre de recouvrement (AR) d'un graphe G connexe est**
  - **un sous-graphe de G qui comporte tous les sommets**
  - **et qui a la propriété d'être un arbre.**
- **Propriétés :**
  - **Nous préservons la connexité !**
  - **Nous n'avons pas de cycles !**
  - **Nous avons un nombre minimal d'arêtes !**
  - **Le choix de l'AR n'est pas unique en général !**



Les arbres de recouvrement

- **Un arbre de recouvrement (AR) d'un graphe G connexe est**
  - **un sous-graphe de G qui comporte tous les sommets**
  - **et qui a la propriété d'être un arbre.**
- **Propriétés :**
  - **Nous préservons la connexité !**
  - **Nous n'avons pas de cycles !**
  - **Nous avons un nombre minimal d'arêtes !**

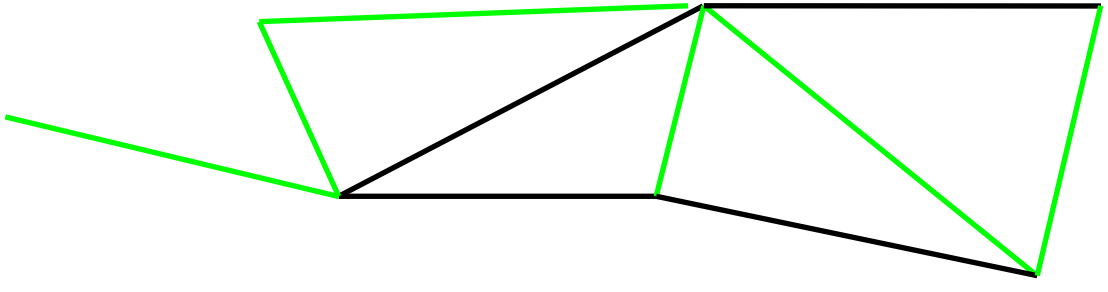
Un autre arbre de recouvrement !



Les arbres de recouvrement

- **Un arbre de recouvrement (AR) d'un graphe G connexe est**
  - **un sous-graphe de G qui comporte tous les sommets**
  - **et qui a la propriété d'être un arbre.**
- **Propriétés :**
  - **Nous préservons la connexité !**
  - **Nous n'avons pas de cycles !**
  - **Nous avons un nombre minimal d'arêtes !**
  - **Le choix de l'AR n'est pas unique en général !**

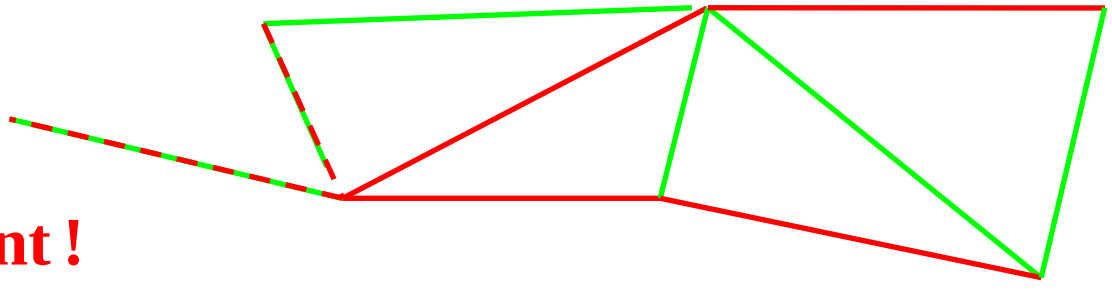
Un arbre de recouvrement !



Les arbres de recouvrement

- **Un arbre de recouvrement (AR) d'un graphe G connexe est**
  - **un sous-graphe de G qui comporte tous les sommets**
  - **et qui a la propriété d'être un arbre.**
- **Propriétés :**
  - **Nous préservons la connexité !**
  - **Nous n'avons pas de cycles !**
  - **Nous avons un nombre minimal d'arêtes !**
  - **Le choix de l'AR n'est pas unique en général !**

Un arbre de recouvrement !



Un autre arbre de recouvrement !

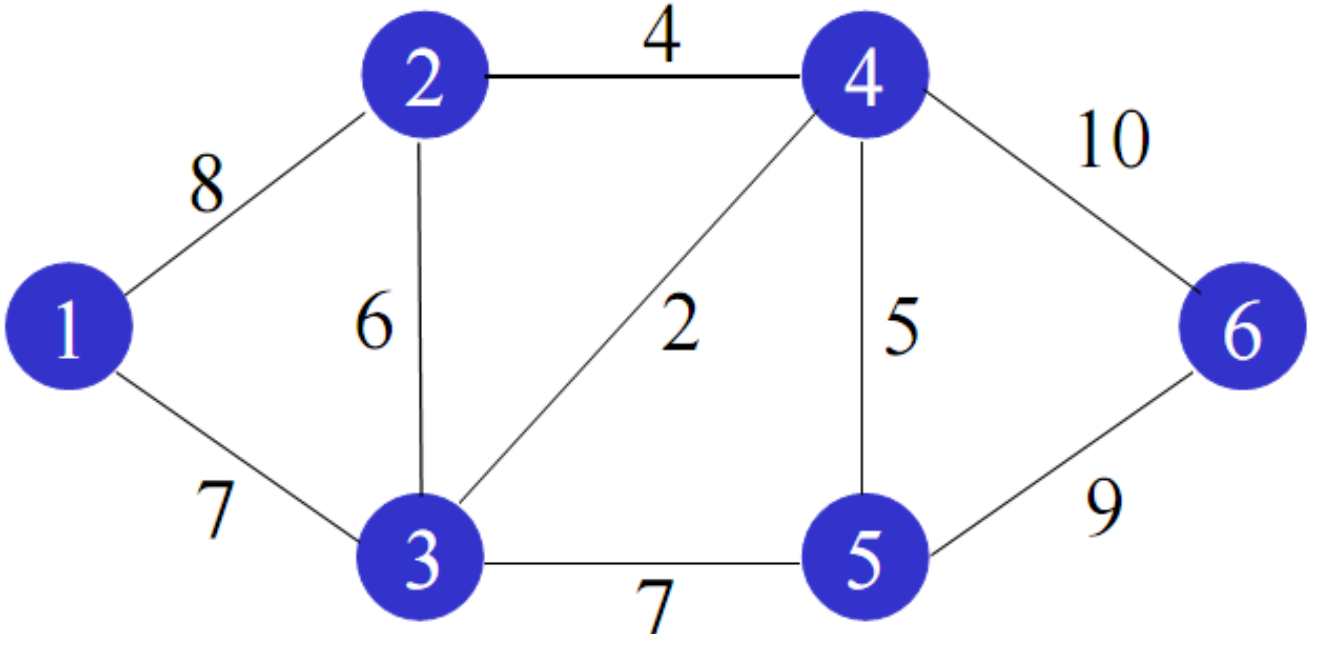
Arbre de poids minimal ou arbre de recouvrement minimal

- **Problème :**

- Tracer l'arbre le plus court qui touche tous les nœuds, c-à-d. minimiser la longueur totale des arcs contenus dans l'arbre.
- 1. Algorithme de Kruskal;
- 2. Algorithme de Prim

Algorithme de Kruskal

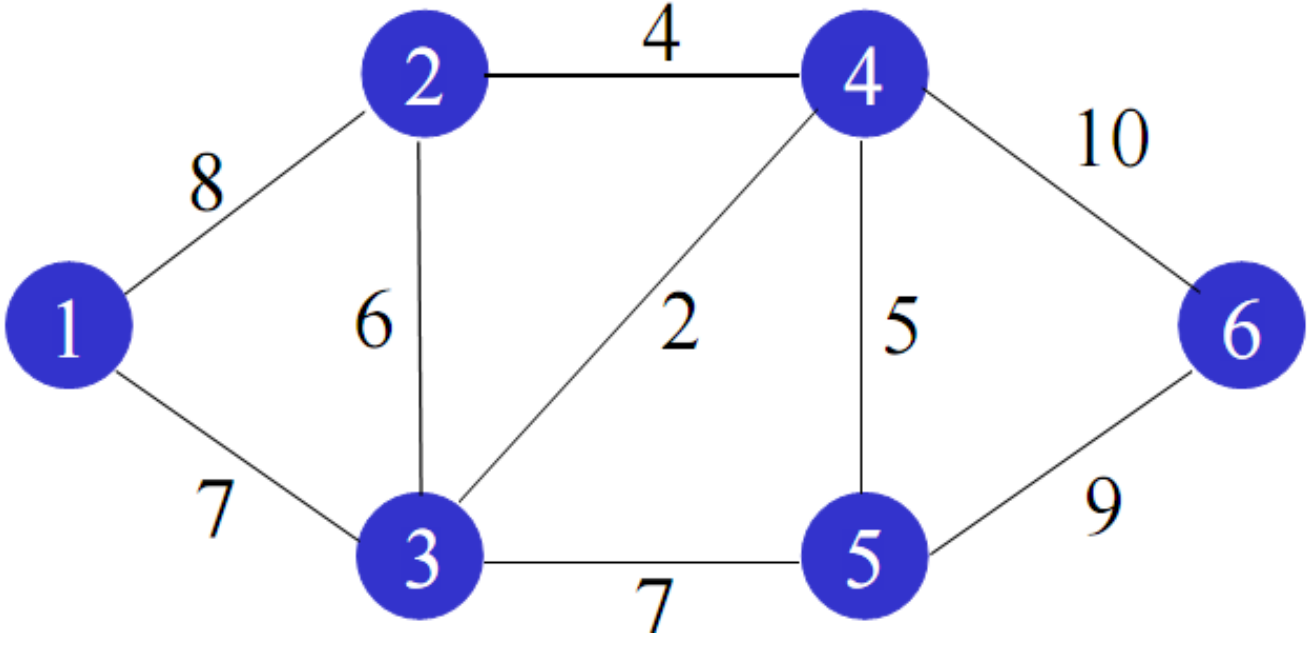
1. Trier les arcs en ordre croissant de poids ;
2. Construire un arbre en sélectionnant les arcs selon l'ordre établi à l'étape 1.



Algorithme de Kruskal

1. Trier les arcs

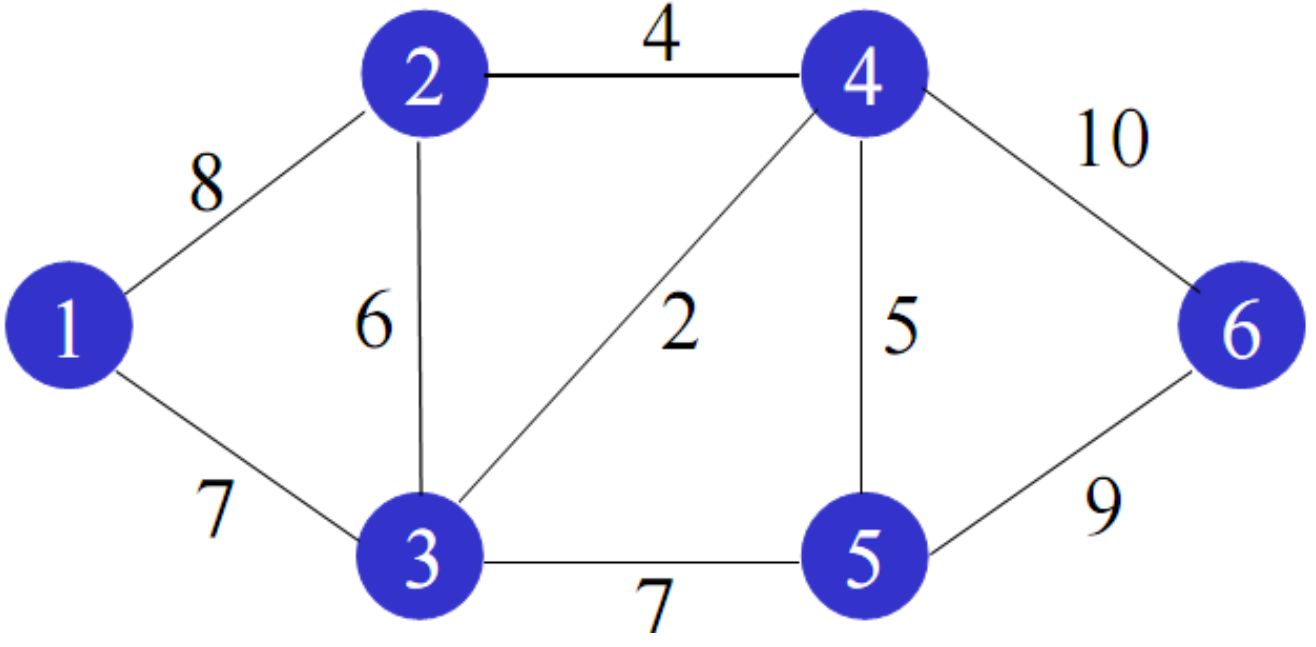
{ (3,4), (2,4), (4,5), (2,3), (1,3), (3,5), (1,2), (5,6),(4,6) }



Algorithme de Kruskal

2. Construire un arbre T = { }

{ (3,4), (2,4), (4,5), (2,3), (1,3), (3,5), (1,2), (5,6),(4,6) }

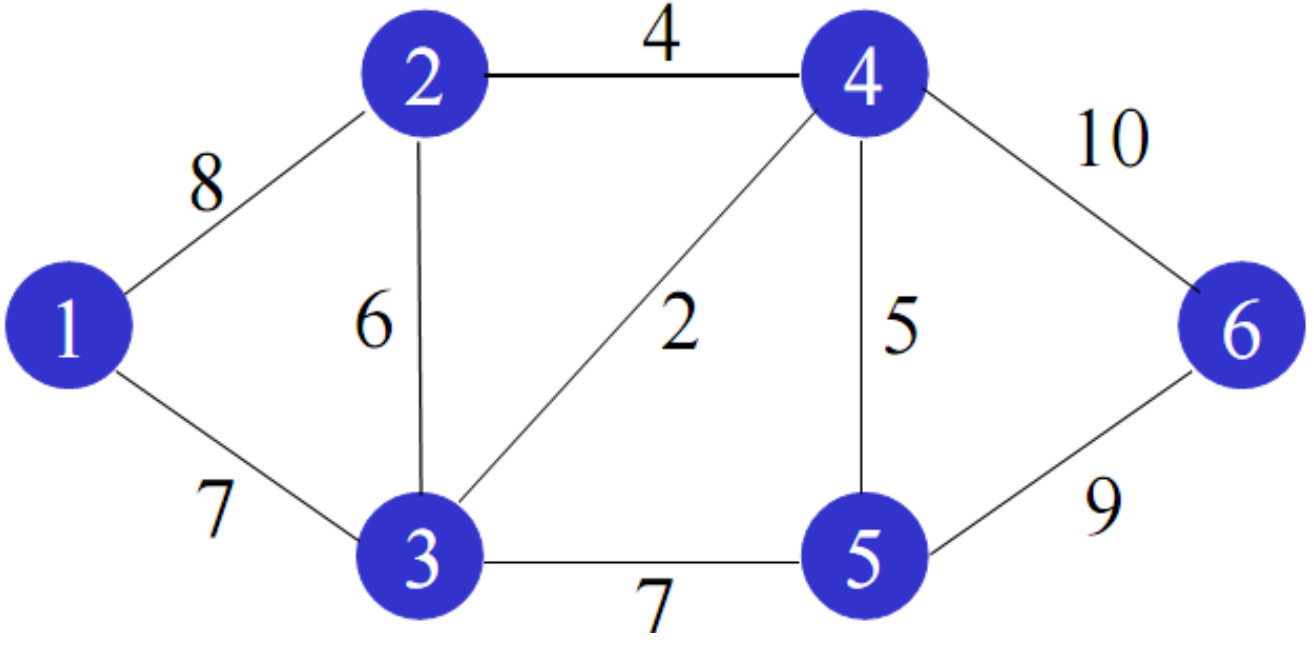


Algorithme de Kruskal

On évalue (3,4) :

{ ~~(3,4)~~, (2,4), (4,5), (2,3), (1,3), (3,5), (1,2), (5,6),(4,6)
}

T= { (3,4) }

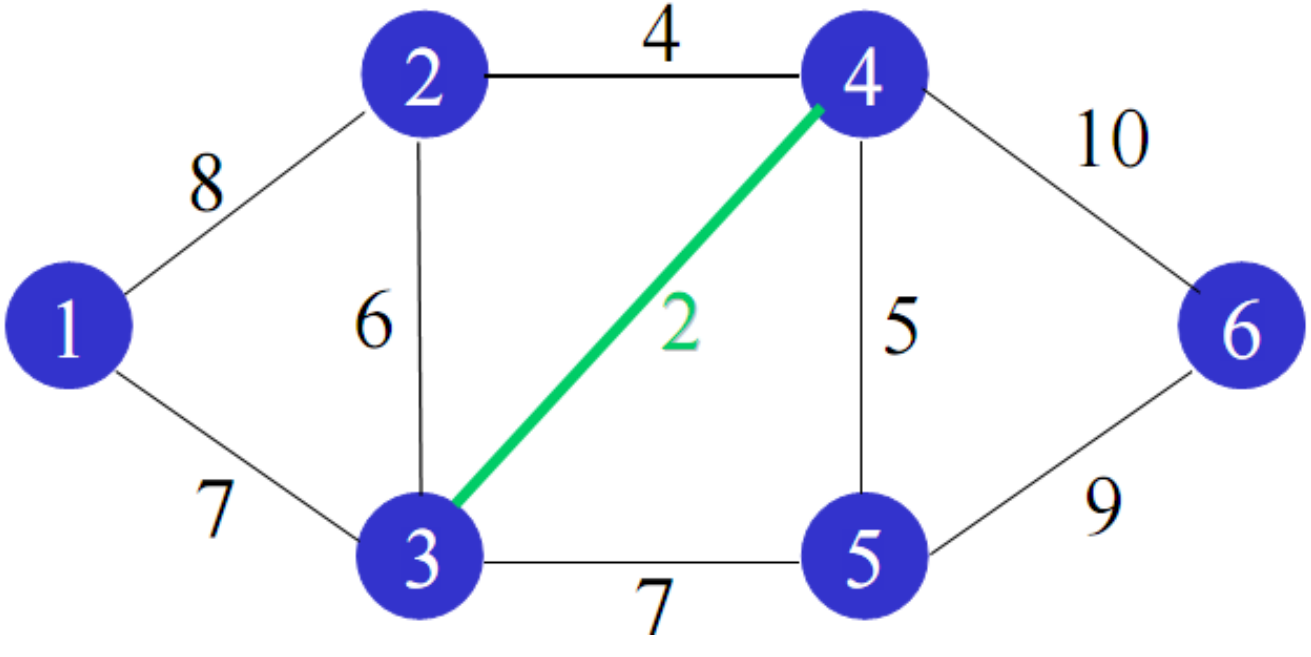


Algorithme de Kruskal

On évalue (3,4) :

{ ~~(3,4)~~, (2,4), (4,5), (2,3), (1,3), (3,5), (1,2), (5,6),(4,6)
}

T= { (3,4) }

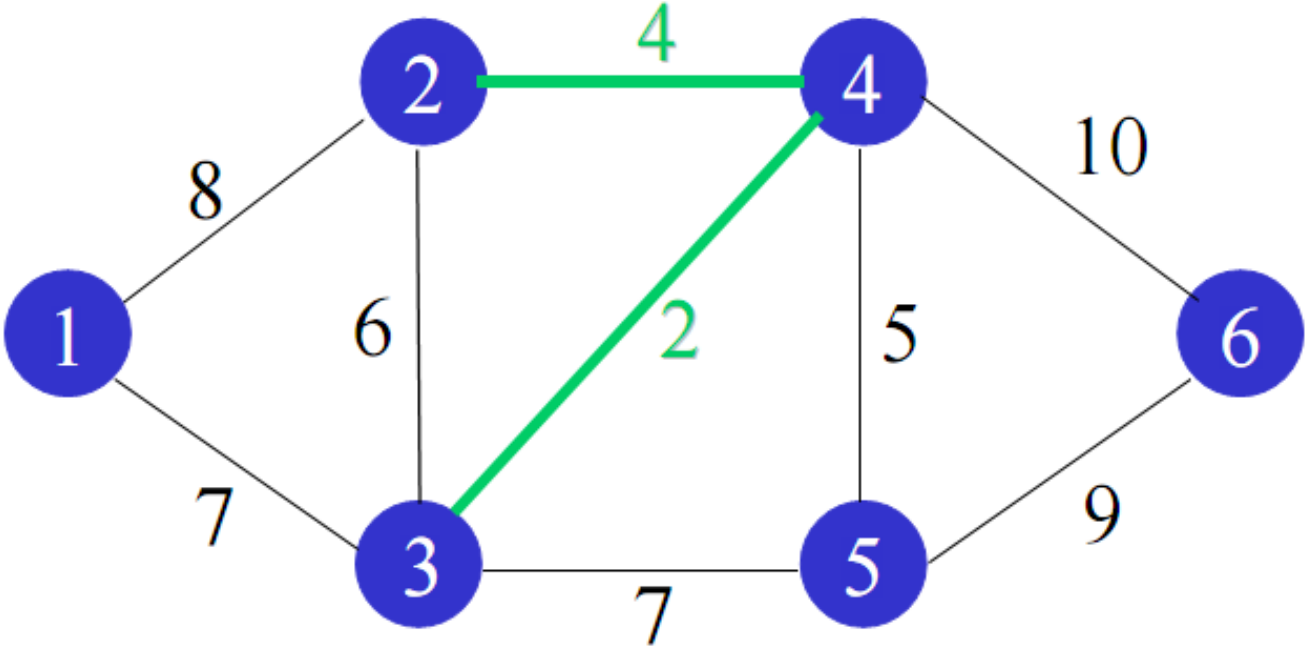


Algorithme de Kruskal

On évalue (2,4) :

{ ~~(3,4)~~, ~~(2,4)~~, (4,5), (2,3), (1,3), (3,5), (1,2), (5,6),(4,6)
}

T= {(3,4), (2,4)}

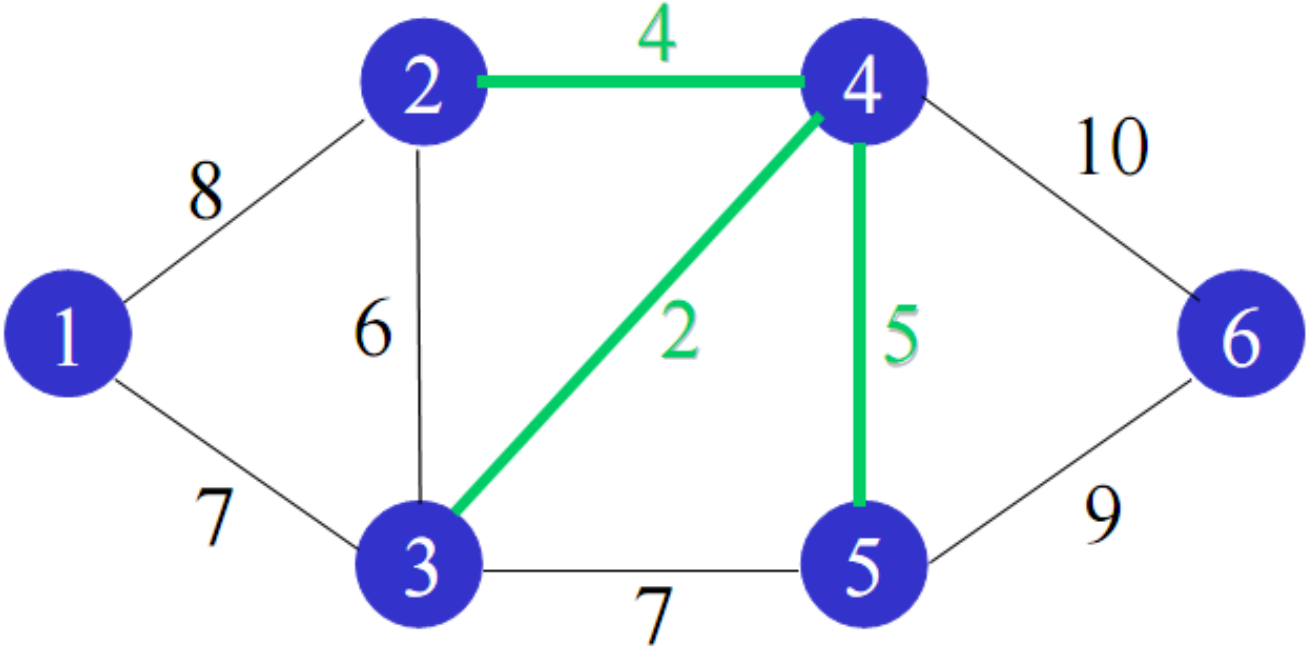


Algorithme de Kruskal

On évalue (4,5):

{ ~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, (2,3), (1,3), (3,5), (1,2), (5,6),(4,6)
}

T= { (3,4), (2,4), (4,5) }

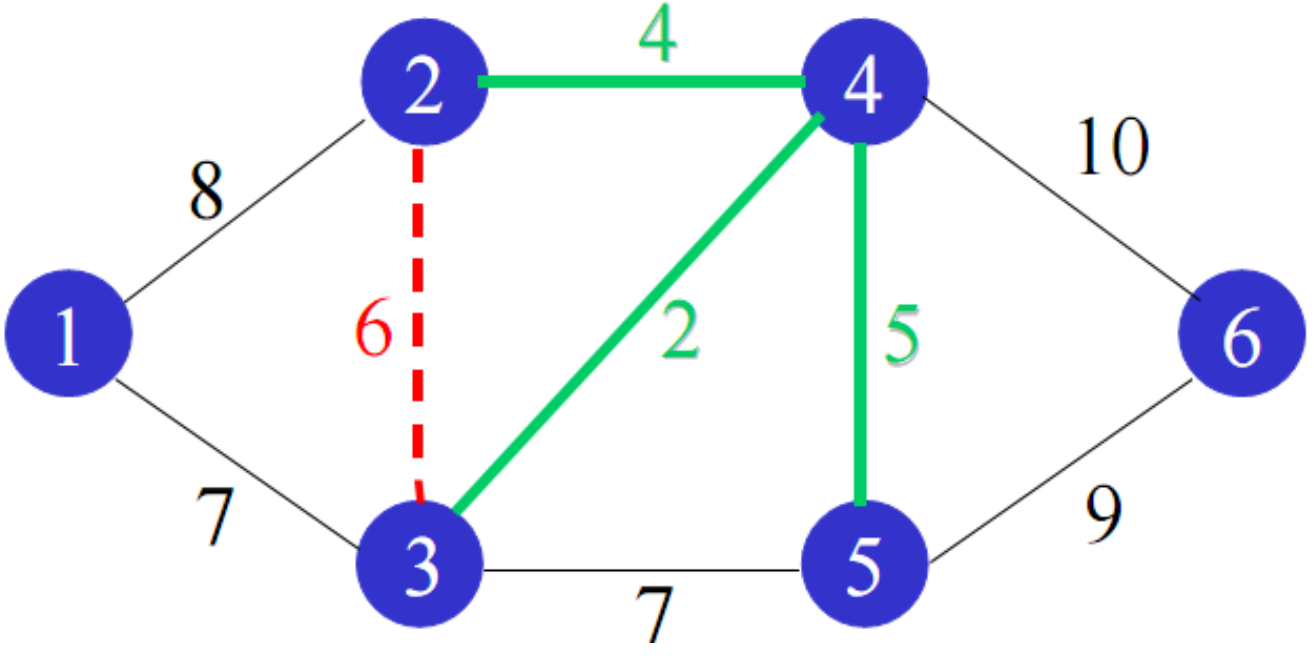


Algorithme de Kruskal

On évalue (2,3):

{ ~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, (1,3), (3,5), (1,2), (5,6),(4,6)
}

T= { (3,4), (2,4), (4,5) }

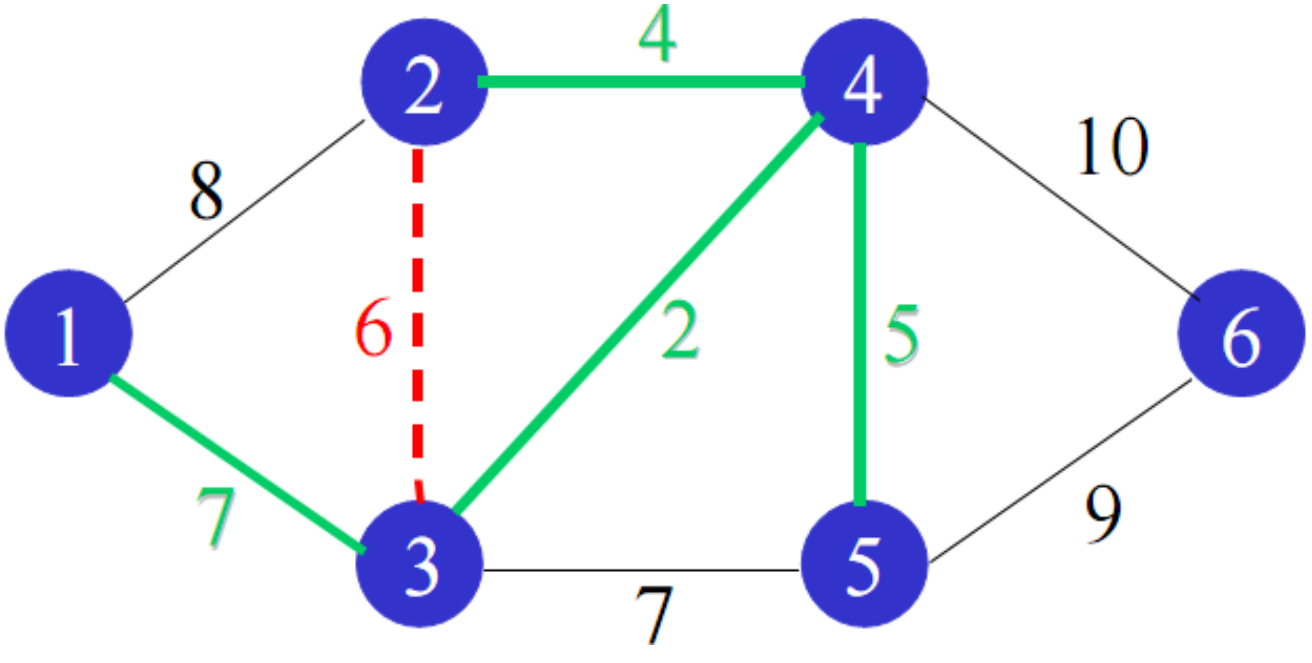


Algorithme de Kruskal

On évalue (1,3):

{ ~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~, (3,5), (1,2), (5,6),(4,6) }

T= { (3,4), (2,4), (4,5), (1,3) }

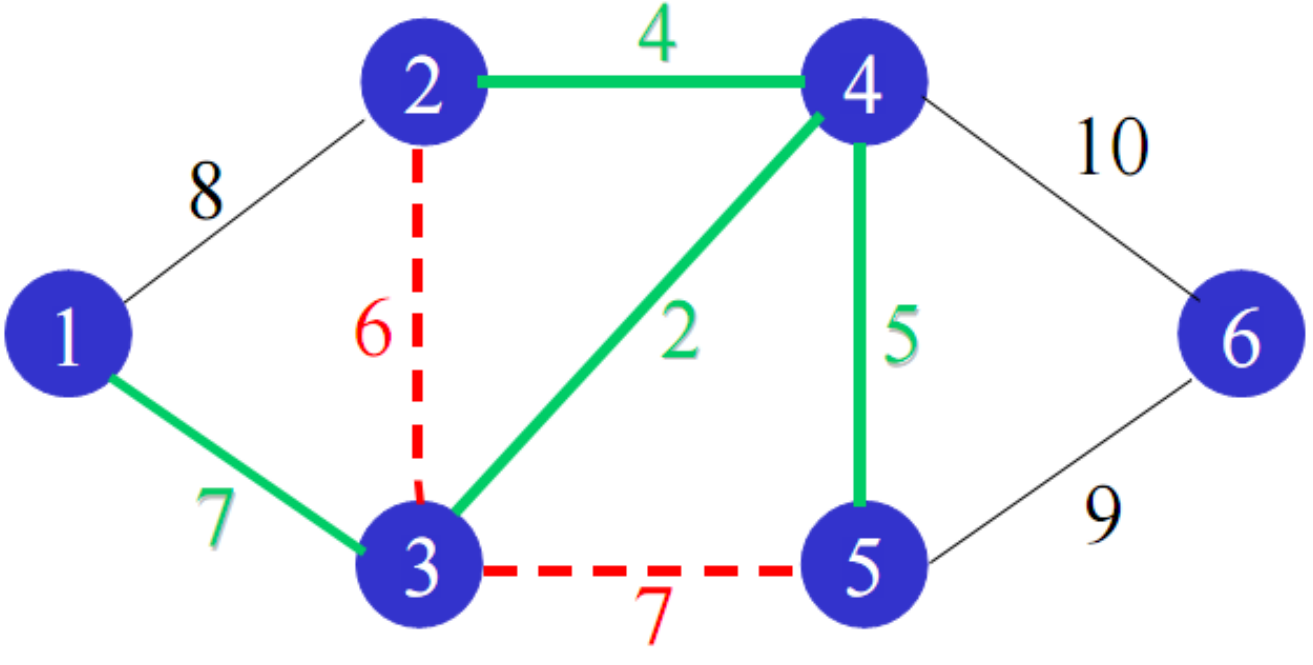


Algorithme de Kruskal

On évalue (3,5):

{ ~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~ , ~~(3,5)~~ , (1,2), (5,6),(4,6)
}

T= { (3,4), (2,4), (4,5), (1,3) }

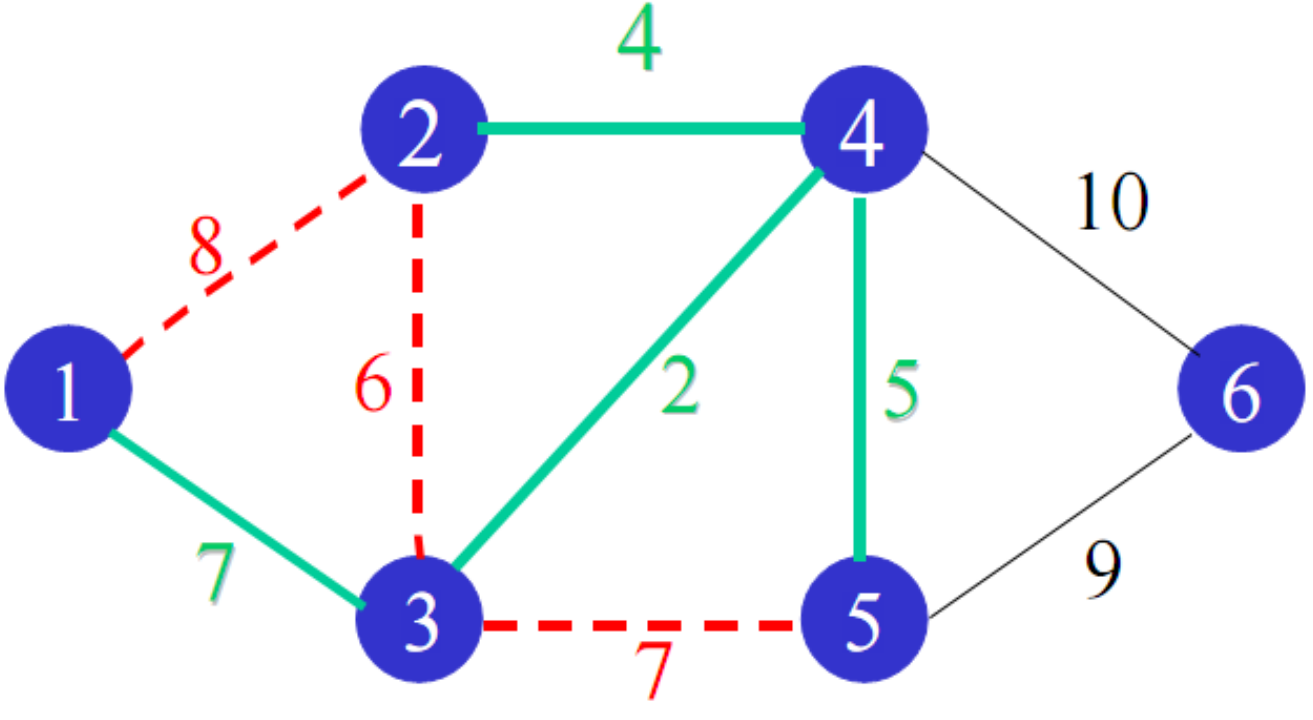


Algorithme de Kruskal

On évalue (1,2):

{ ~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~ , ~~(3,5)~~, ~~(1,2)~~, (5,6) ,(4,6)
}

T= { (3,4), (2,4), (4,5), (1,3) }

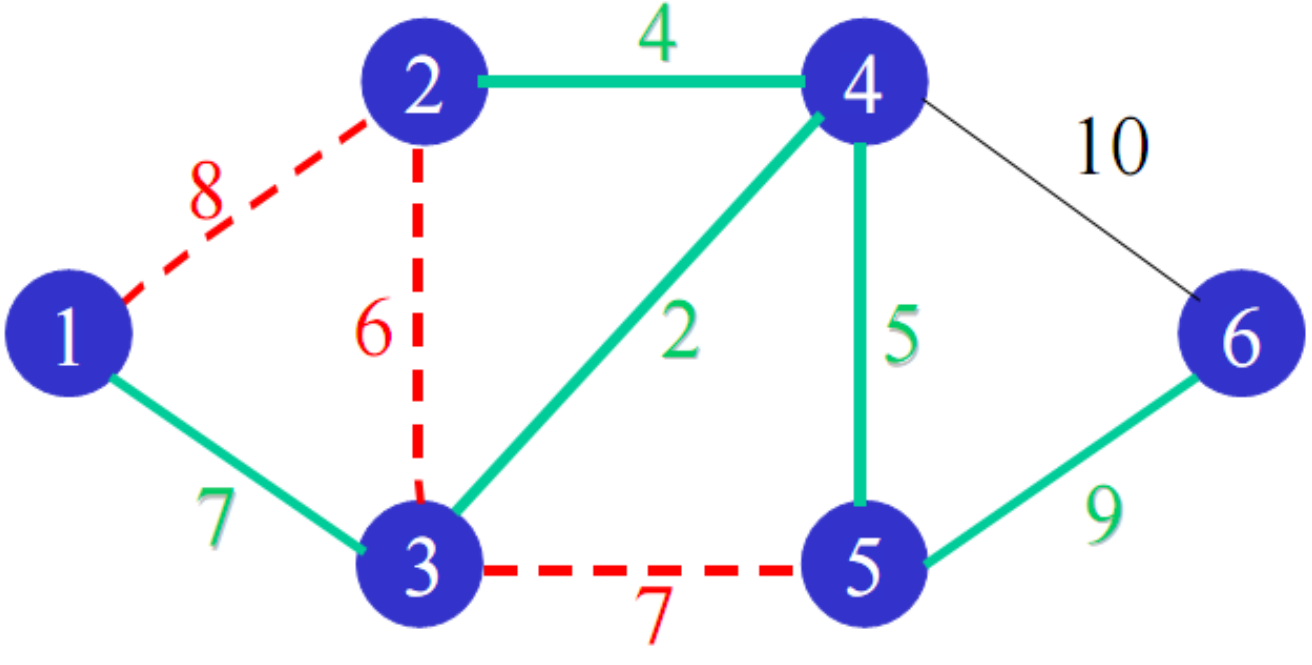


Algorithme de Kruskal

On évalue (5,6):

{ ~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~,~~(2,3)~~, ~~(1,3)~~ ,~~(3,5)~~, ~~(1,2)~~, ~~(5,6)~~ ,(4,6)
}

T= { (3,4), (2,4), (4,5), (1,3), (5,6) }

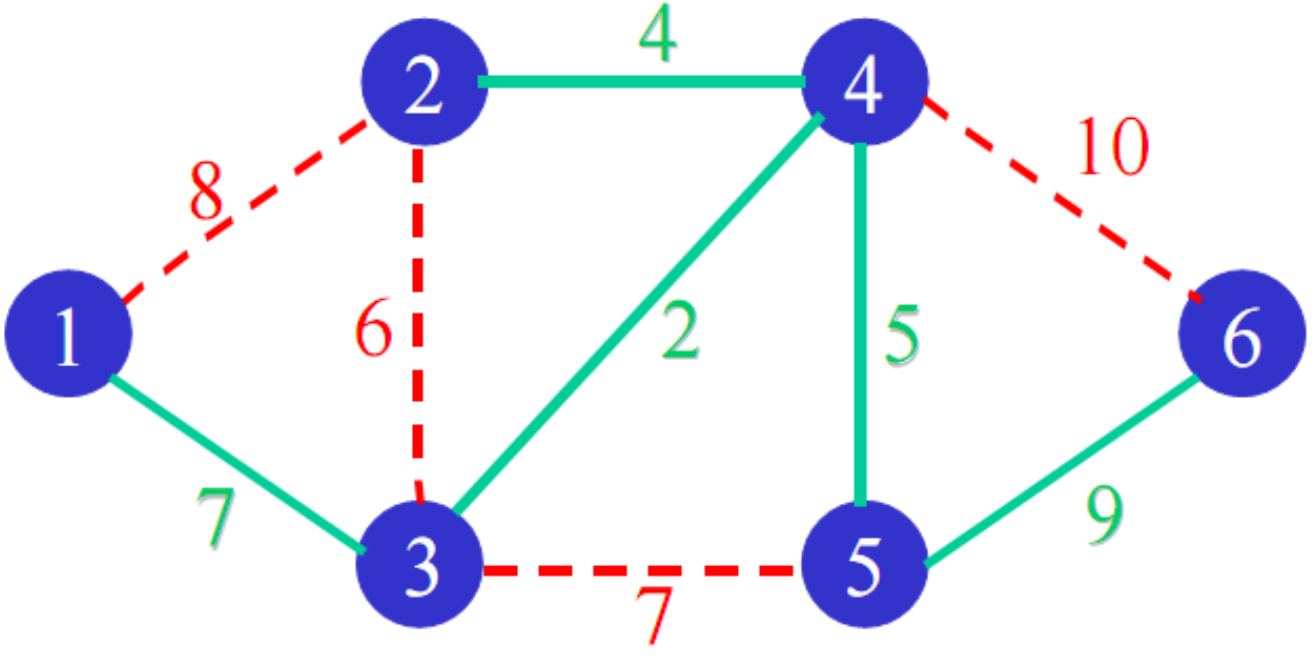


Algorithme de Kruskal

On évalue (4,6):

{ ~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~, ~~(3,5)~~, ~~(1,2)~~, ~~(5,6)~~, ~~(4,6)~~}

T= { (3,4), (2,4), (4,5), (1,3), (5,6) }



Algorithme de Prim

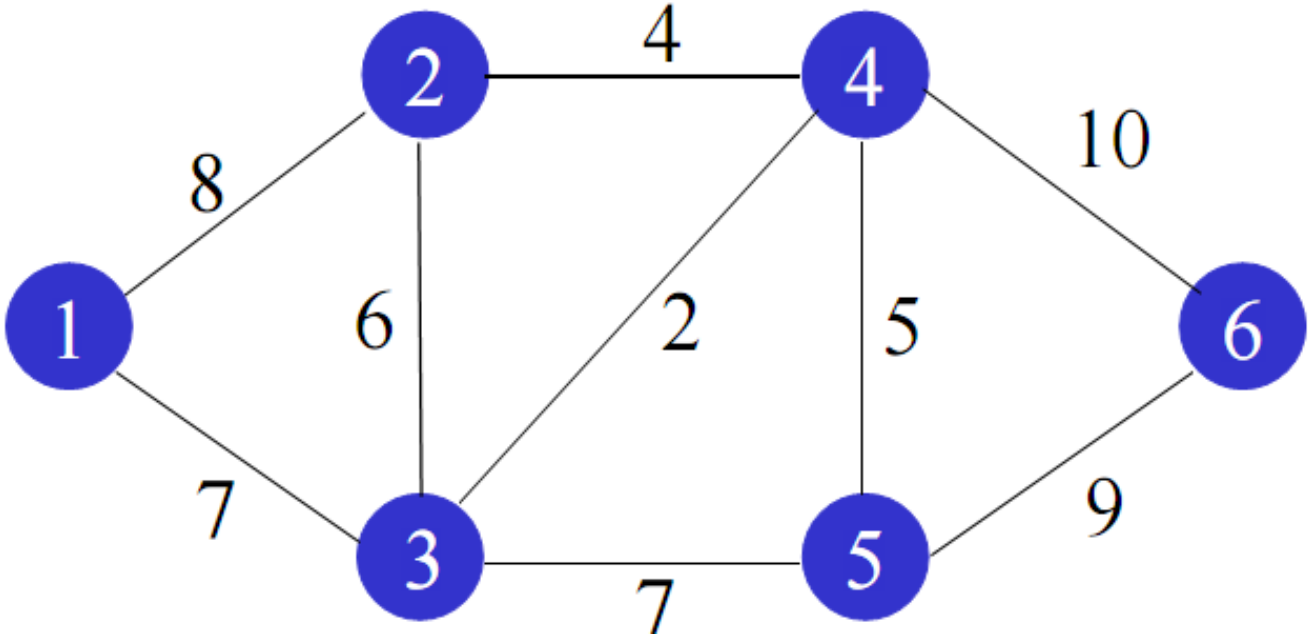
- 0. Initialiser T={}, S={} et W= V (sommets)
- 1. Choisir un nœud i $\in$ W, poser S = { i } et W = W \ { i }
- 2. Choisir un nœud j $\in$ W tel que l'arc (i, j), où i $\in$ S, ait le $c_{i,j}$ le plus petit;
- 3. Si (i,j) ne forme pas de cycle alors inclure cet arc dans
- l'arbre, i.e. T = T $\cup$ (i,j), j $\in$ S et W = W \ { j } et aller à
- l'étape 4;
- sinon exclure l'arc et retourner à l'étape 2;
- 4. Si S = V alors l'arbre de poids minimal est trouvé; sinon
- retourner à l 'étape 2.

Algorithme de Prim

S = { }

W = { 1, 2, 3, 4, 5, 6 }

T = { }

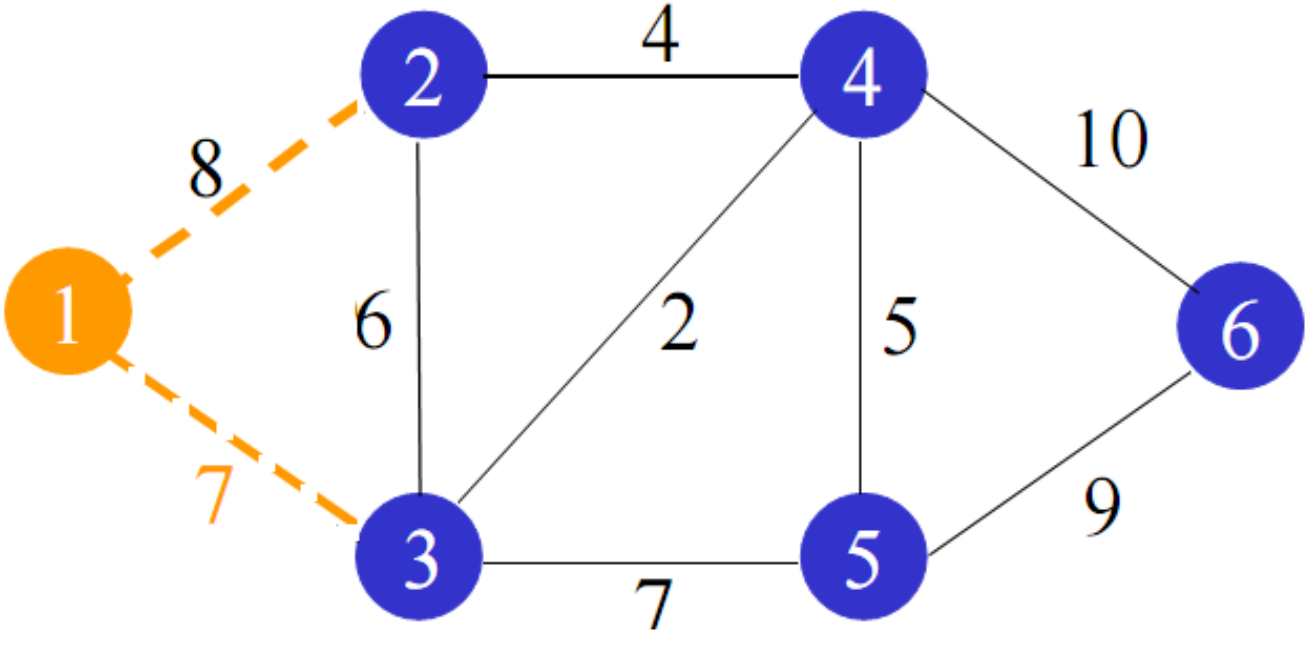


Algorithme de Prim

S = { 1 }

W = { 2, 3, 4, 5, 6 }

T = { }

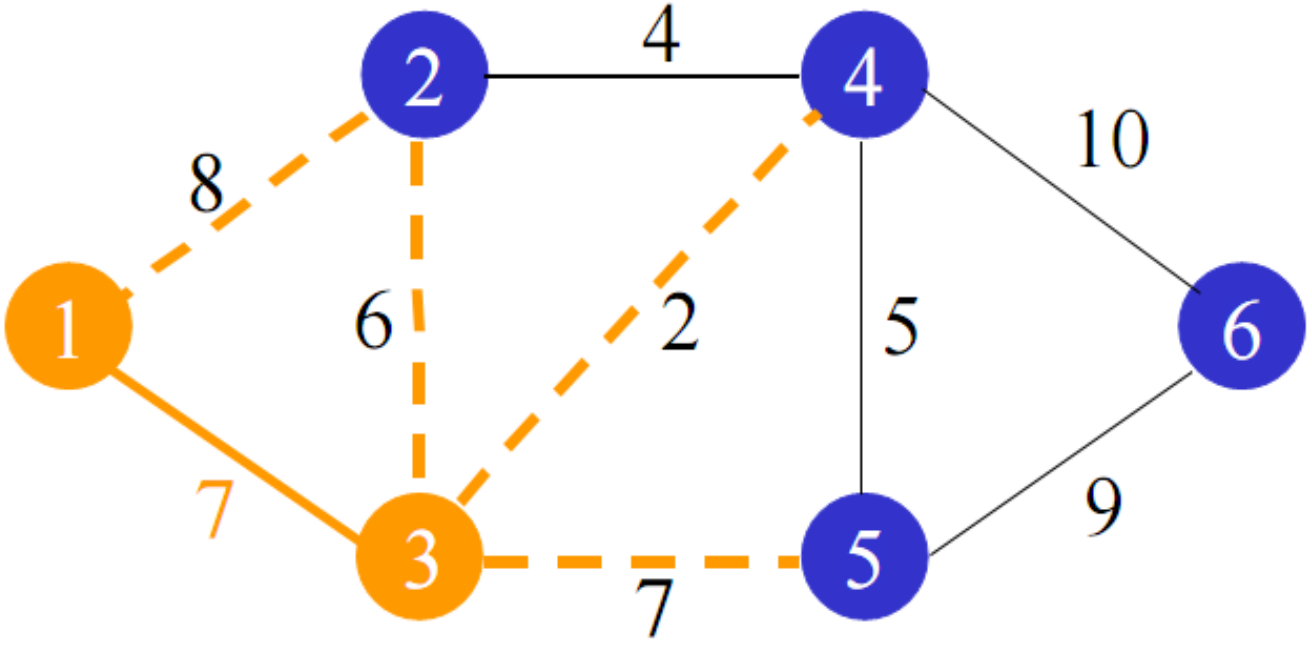


Algorithme de Prim

S = { 1,3 }

W = { 2, 4, 5, 6 }

T = { (1,3) }

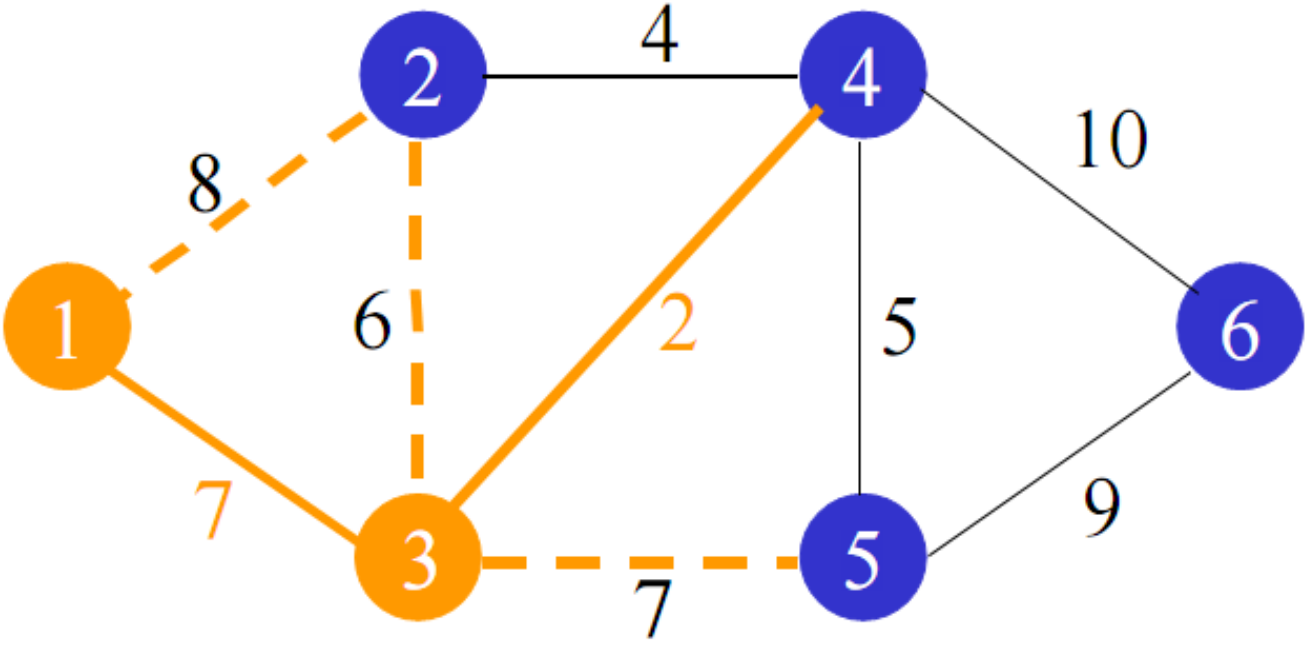


Algorithme de Prim

S = { 1,3 }

W = { 2, 4, 5, 6 }

T = { (1,3), (3,4) }

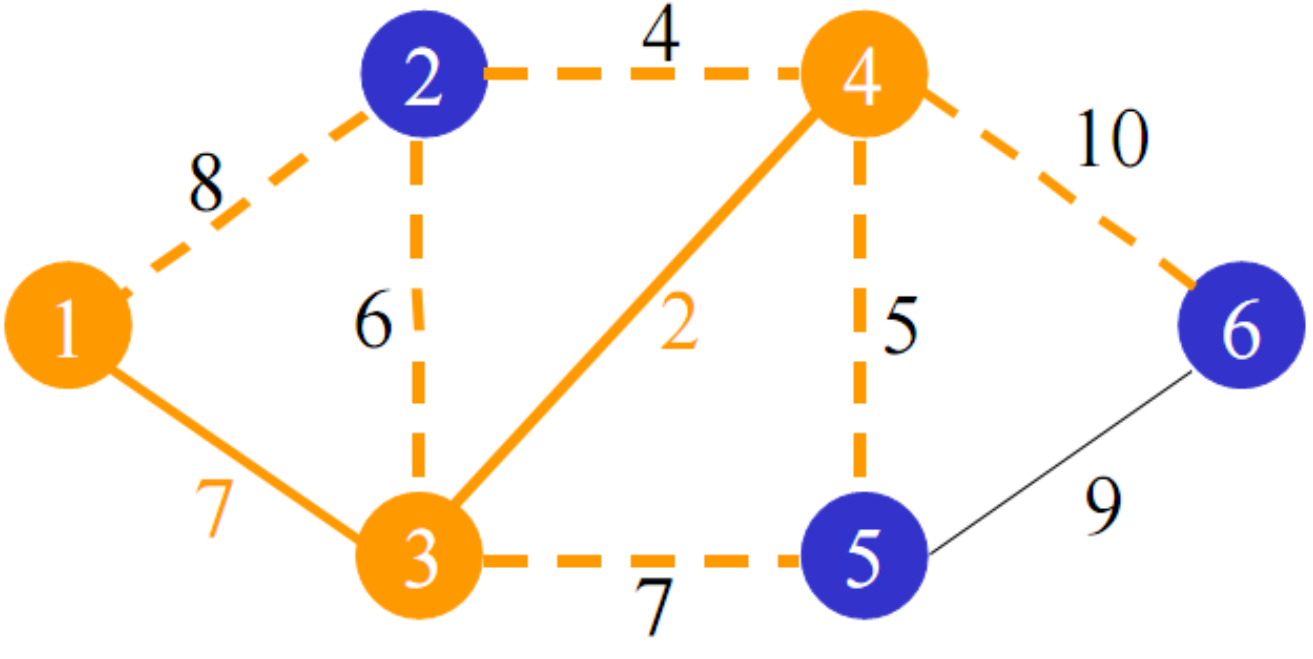


Algorithme de Prim

S = { 1, 3, 4}

W = { 2, 5, 6 }

T = { (1,3), (3,4) }

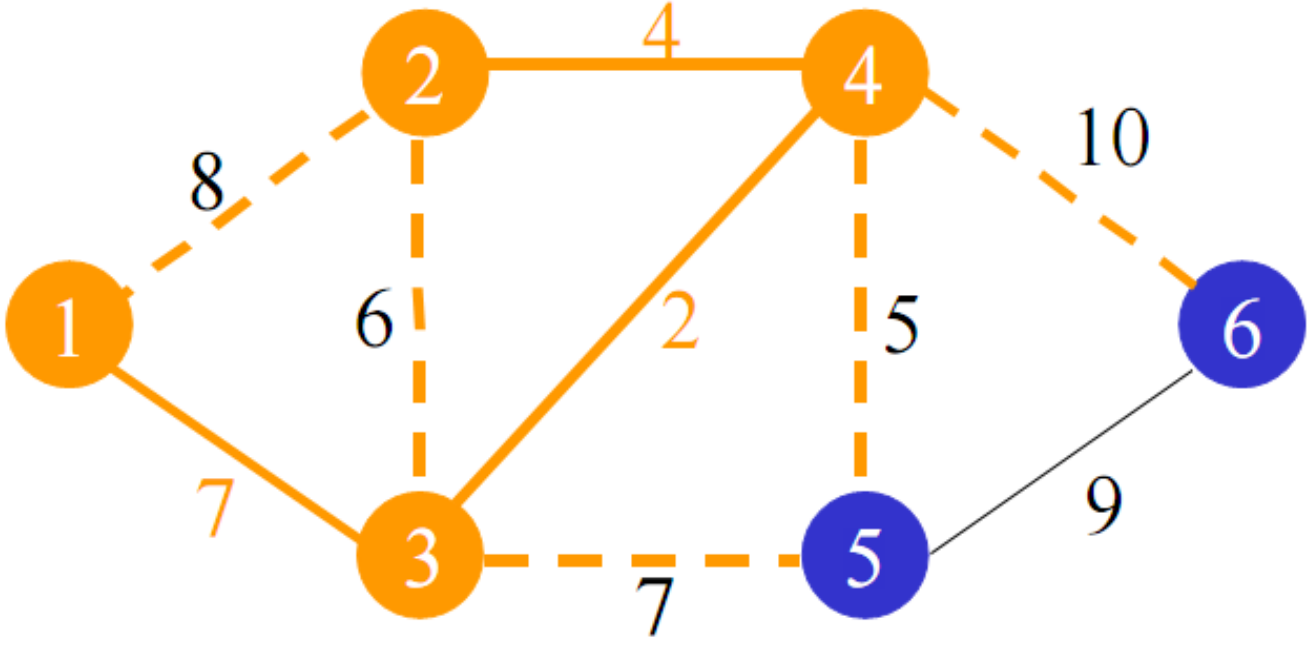


Algorithme de Prim

S = { 1, 3, 4}

W = { 2, 5, 6 }

T = { (1,3), (3,4), (2,4) }

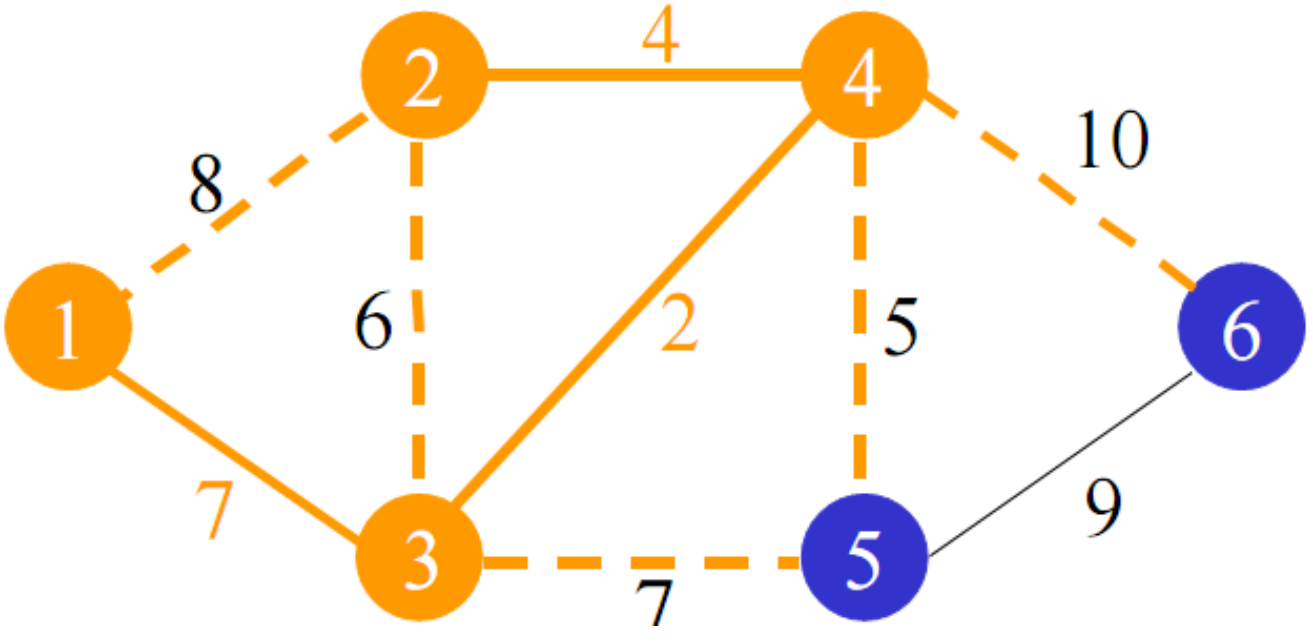


Algorithme de Prim

S = { 1, 3, 4, 2 }

W = { 5, 6 }

T = { (1,3), (3,4), (2,4) }

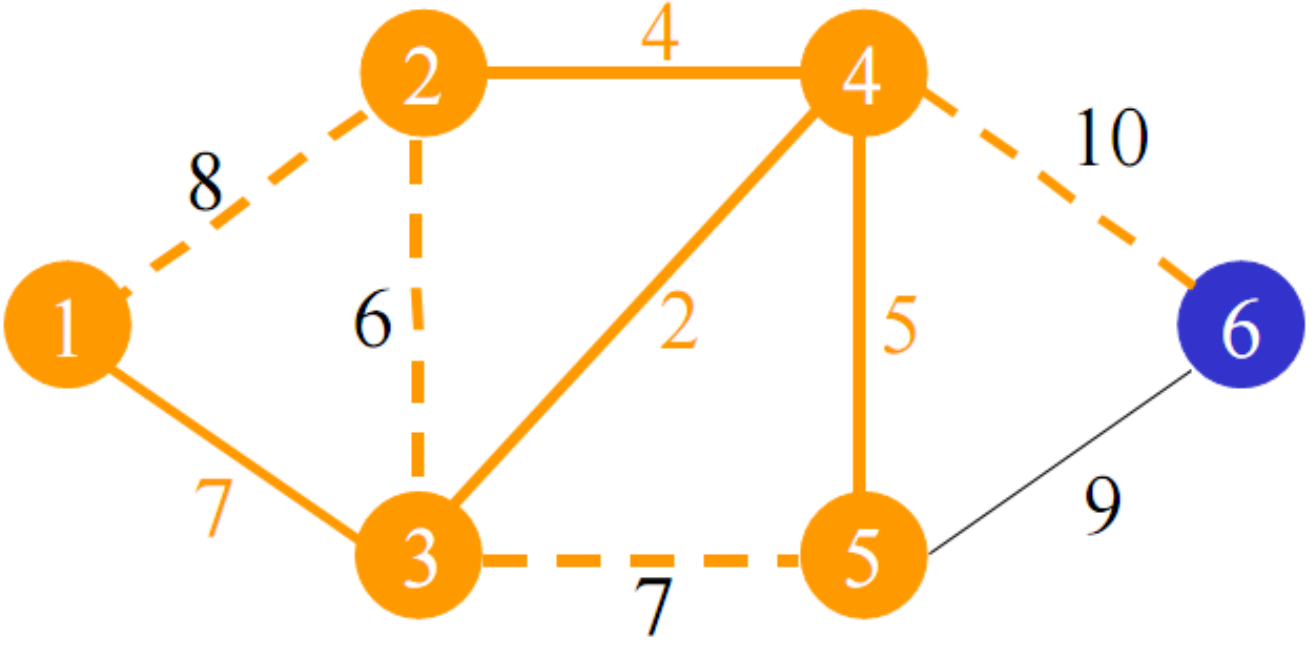


Algorithme de Prim

S = { 1, 3, 4, 2 }

W = { 6 }

T = { (1,3), (3,4), (2,4), (4,5) }

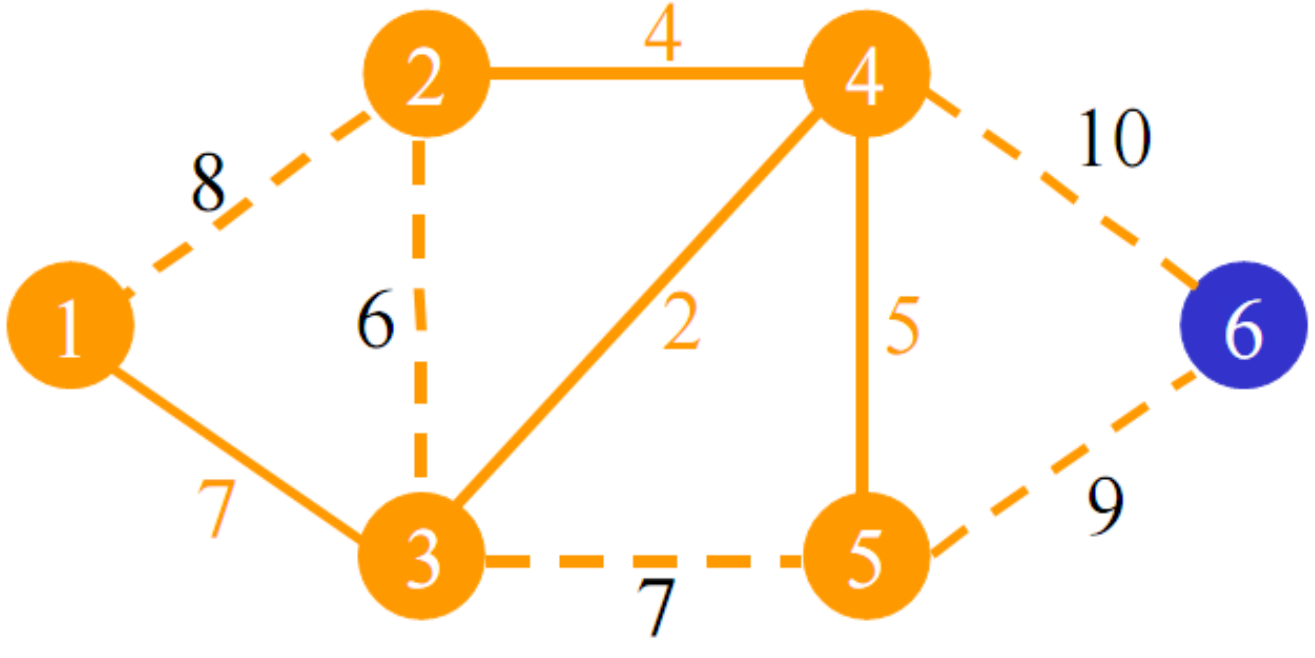


Algorithme de Prim

S = { 1, 3, 4, 2, 5 }

W = { 6 }

T = { (1,3), (3,4), (2,4), (4,5) }



Algorithme de Prim

S = { 1, 3, 4, 2, 5 }

W = { 6 }

T = { (1,3), (3,4), (2,4), (4,5) }