Arbres couvrants de poids minimal

Stéphane Grandcolas

Aix-Marseille Université

2021-2022

Graphes

Plan du cours :

- définitions, exemples,
- plus courts chemins : Dijkstra,
- plus courts chemins : Bellman-Ford,
- plus courts chemins : Floyd et Warshall,
- arbres couvrants de poids minimal (ACM) : (Prim, Kruskal),

Arbres couvrants minimaux (ACM)

$G = (S, A, w)$ un graphe pondéré, connexe et non orienté.

$T = (S, A', w)$ est un *arbre couvrant* de G si

- $A' \subseteq A$,
- T est connexe et sans cycle (c'est un arbre).

Arbres couvrants minimaux (ACM)

$G = (S, A, w)$ un graphe pondéré, connexe et non orienté.

$T = (S, A', w)$ est un *arbre couvrant* de G si

- $A' \subseteq A$,
- T est connexe et sans cycle (c'est un arbre).

T est un *arbre couvrant de poids minimal* si son poids

$$\sum_{(u,v)\in A'} w(u, v)$$

est minimal *(il n'existe pas d'arbre couvrant de G de poids plus petit)*

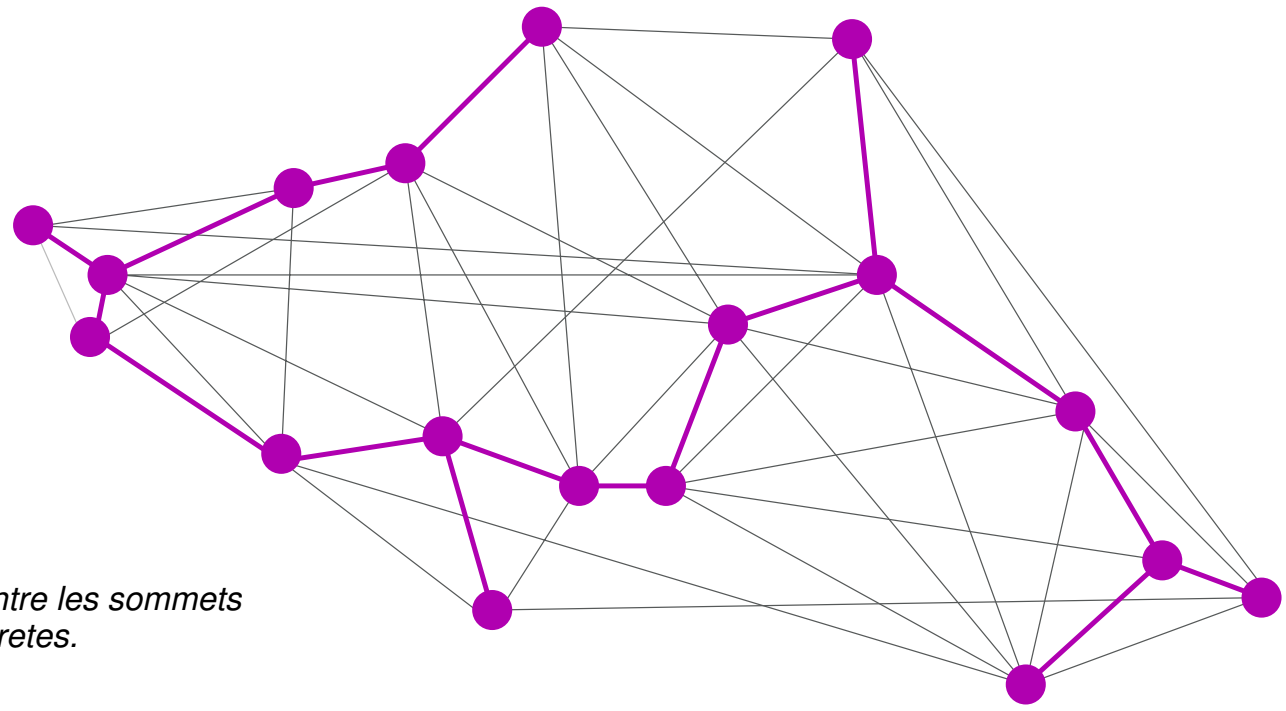
Arbres couvrants minimaux (ACM)

Applications :

- construction de réseaux (téléphonique, électrique, hydraulique, routier,...) : minimiser le coût des lignes,
- algorithmes d'approximation : problème du voyageur de commerce, arbres de Steiner : minimiser le coût parcours ou de l'arbre,
- k-clustering : suppression des $k-1$ arêtes les plus couteuses,
- ...

Arbres couvrants minimaux (ACM)

Exemple :



la distance Euclidienne entre les sommets représente le poids des aretes.

Arbres couvrants minimaux (ACM)

Problème d'optimisation :

- énumération exhaustive $\mathcal{O}(m^n)$,
- **algorithme glouton :**
  - choix *glouton*,
  - optimalité à démontrer.

Exemple rendu de monnaie :

- monnaie 5, 2, 1 : le choix de la pièce de plus grande valeur produit la solution optimale,
- monnaie 5, 4, 1 : solution non optimale $8 = 5 + 1 + 1 + 1$.

ACM : algorithme générique

Construire-ACM(G)
in : $G = (S, A, w)$ un graphe pondéré,
1 $E := \emptyset$,
2 **tant que** E n'est pas un arbre couvrant pour G **faire**
3 Choisir une *arête sûre* $(u, v) \in A$,
4 $E := E \cup \{(u, v)\}$,
5 **renvoyer** E,

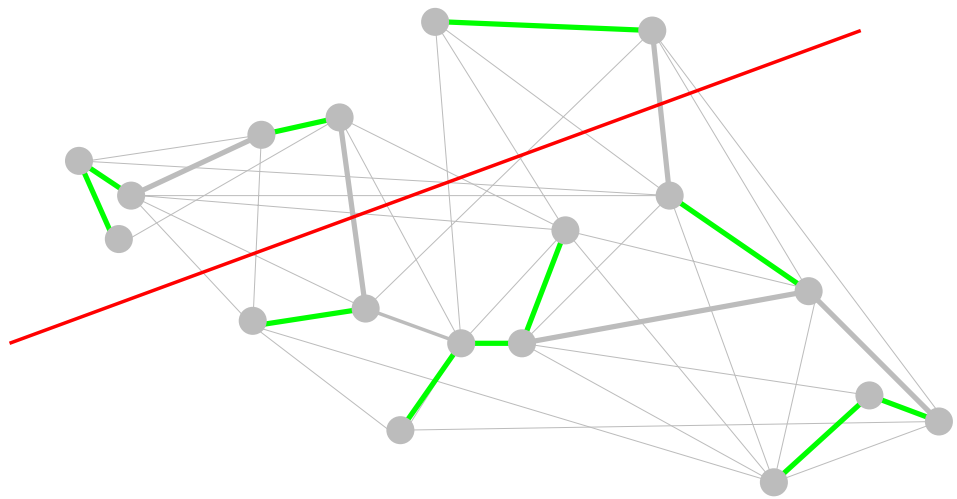
arête sûre (u, v) : il existe un arbre couvrant de poids minimal T tel que

$$E \cup \{(u, v)\} \subset T$$

ACM : arête sûre

- T un ACM
- U un sous-ensemble de T (arêtes vertes)
- $(P, S - P)$ une coupe qui respecte U*

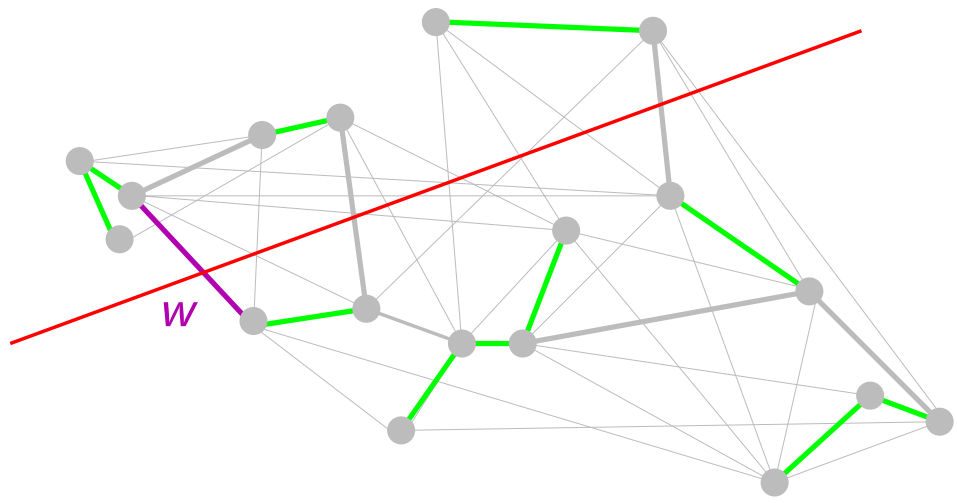
Propriété. Une arête de poids minimal traversant la coupe est sûre.



*i.e. aucune arête de U ne traverse la coupe

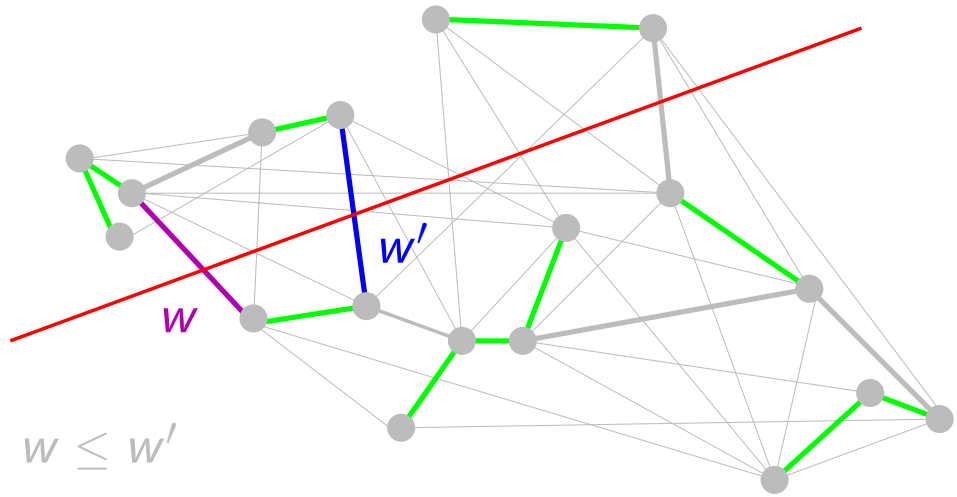
ACM : arête sûre

Hypothèse : T ne contient pas l'arête violette (arête de poids minimal qui traverse la coupe)



ACM : arête sûre

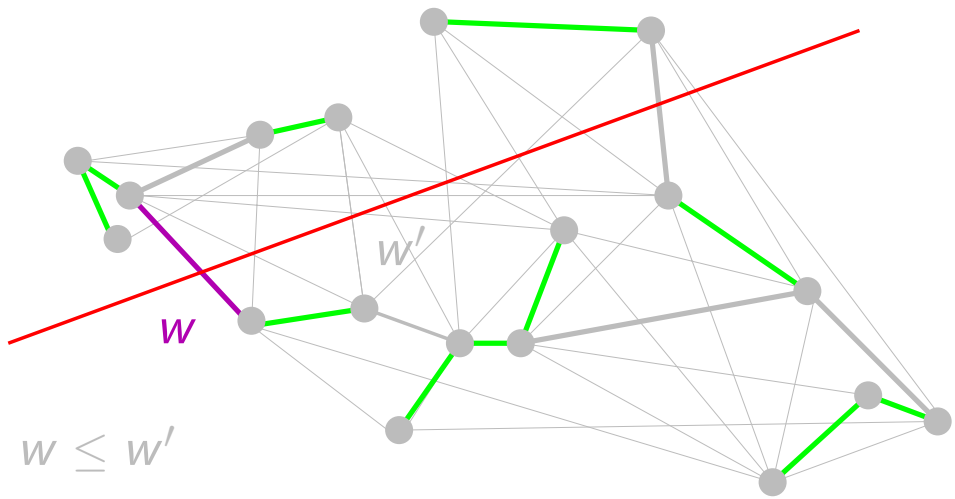
- **ajout** de l'arête violette : création d'un cycle



ACM : arête sûre

- **suppression** de l'arête bleue : arbre T'

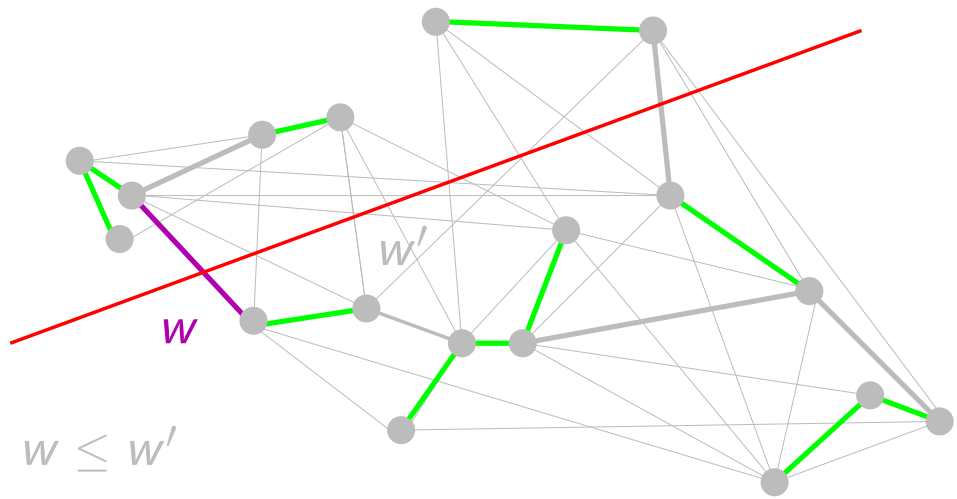
$$poids(T') = poids(T) + w - w' \leq poids(T)$$



ACM : arête sûre

- puisque T est un arbre couvrant de poids minimal

$$poids(T) \leq poids(T')$$

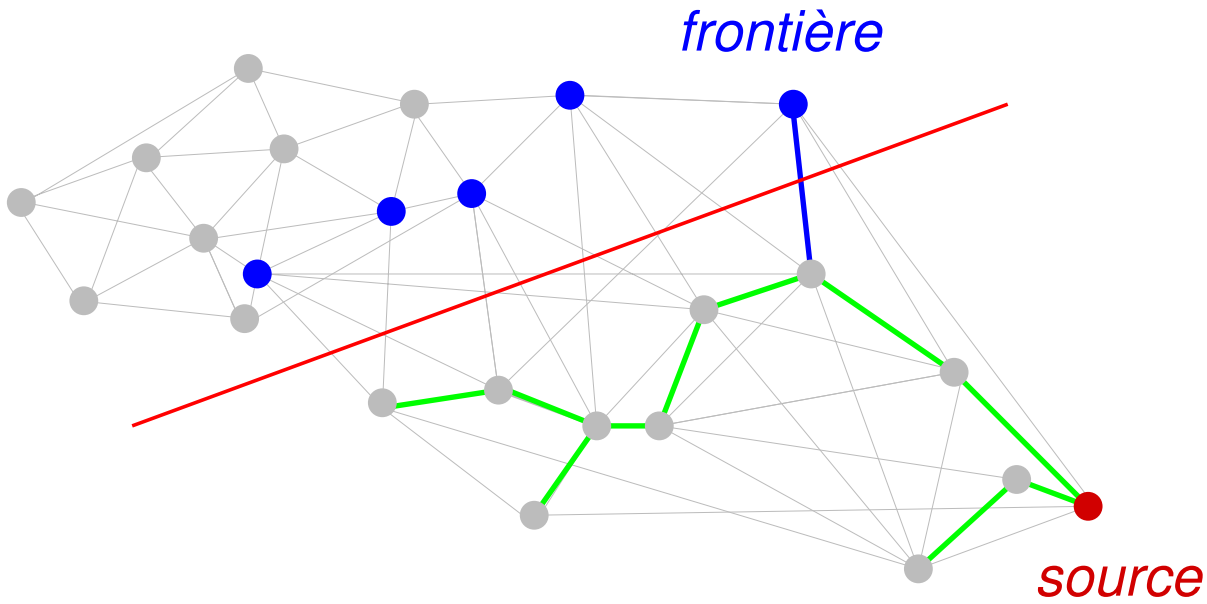


Conclusion : $poids(T') = poids(T)$, donc T' est un ACM.

ACM : algorithme de Prim

Semblable à l'algorithme de Dijkstra :

- **sommet source** s,
- **extension de l'arbre :** ajout d'une arête traversante de poids minimal.



ACM : algorithme de Prim

algorithme DIJKSTRA(G, s)
in : $G = (S, V[], w)$ un graphe pondéré, s un sommet de G
(F une file de priorité)

1 $F := S$, {*i.e. initialement F contient tous les sommets*}
2 **pour chaque sommet** $u \in S$ **faire**
3 $d[u] := \infty$,
4 $d[s] := 0$,
5 **tant que** $F \neq \emptyset$ **faire**
6 $u :=$EXTRAIRE_LE_MIN(F),
7 **pour chaque sommet** $v \in V[u]$ **faire**
8 **si** $d[v] > d[u] + w(u, v)$ **alors**
9 $d[v] := d[u] + w(u, v)$,
9 $pred[v] := u$,
10 **fin pour**,
11 **fin tant que**,
12 **renvoyer** *pred*,

ACM : algorithme de Prim

algorithme ACM-PRIM(G, s)
in : $G = (S, V[], w)$ un graphe pondéré, s un sommet de G
(F une file de priorité)

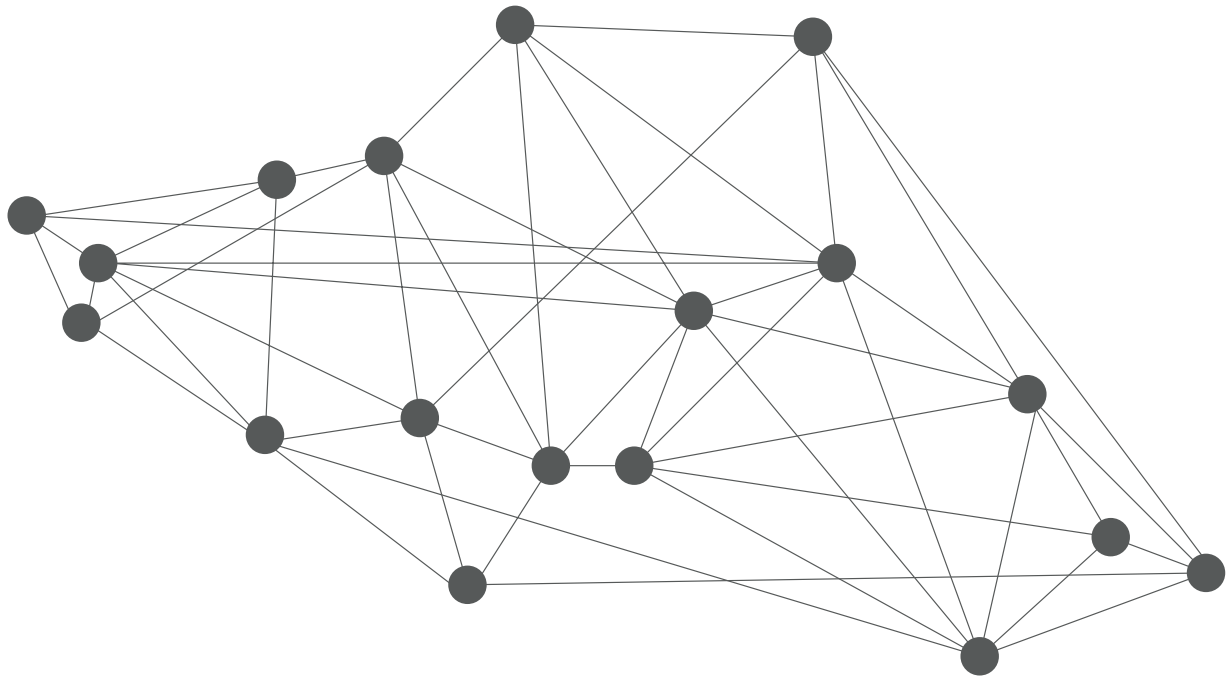
1 $F := S$, {*i.e. initialement F contient tous les sommets*}
2 **pour chaque sommet** $u \in S$ **faire**
3 $d[u] := \infty$,
4 $d[s] := 0$,
5 **tant que** $F \neq \emptyset$ **faire**
6 $u :=$EXTRAIRE_LE_MIN(F),
7 **pour chaque sommet** $v \in V[u]$ **faire**
8 **si** $d[v] > w(u, v)$ **alors**
9 $d[v] := w(u, v)$,
9 $pred[v] := u$,
10 **fin pour**,
11 **fin tant que**,
12 **renvoyer** *pred*,

ACM : Kruskal

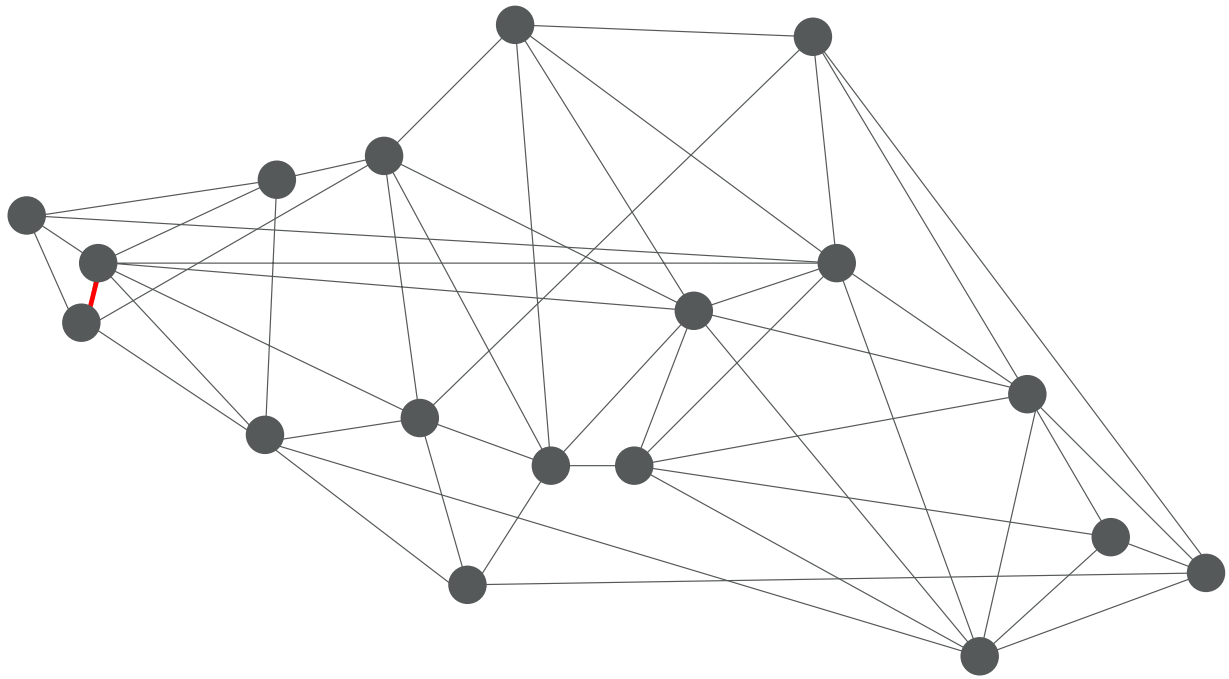
Algorithme de Kruskal

- *initialisation* : chaque sommet constitue sa propre composante connexe
- *itération* : ajout d'une arête de poids minimal qui ne crée pas de cycle (l'ajout regroupe deux composantes connexes)

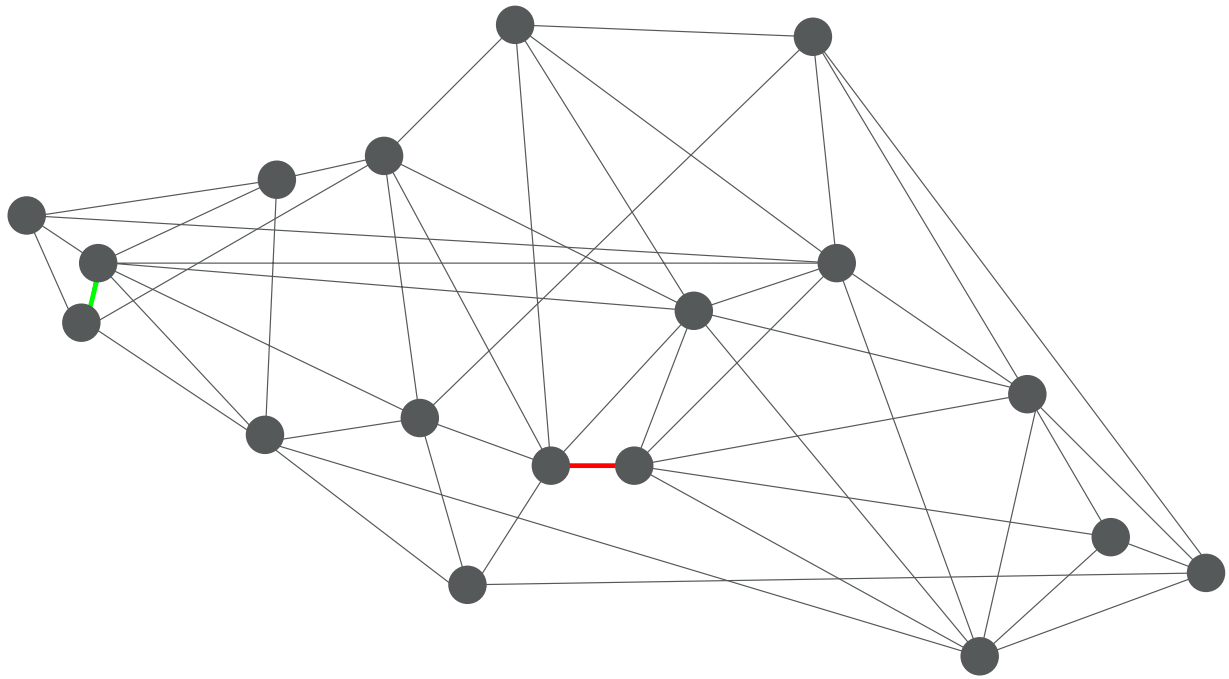
ACM : Kruskal



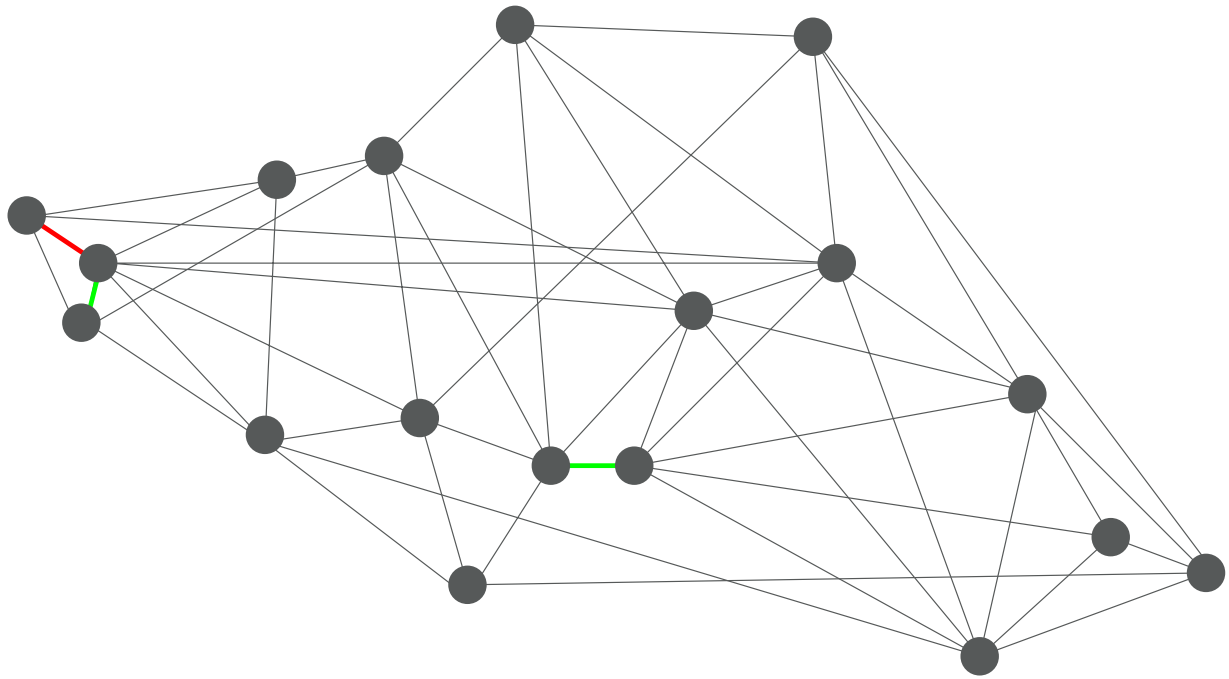
ACM : Kruskal



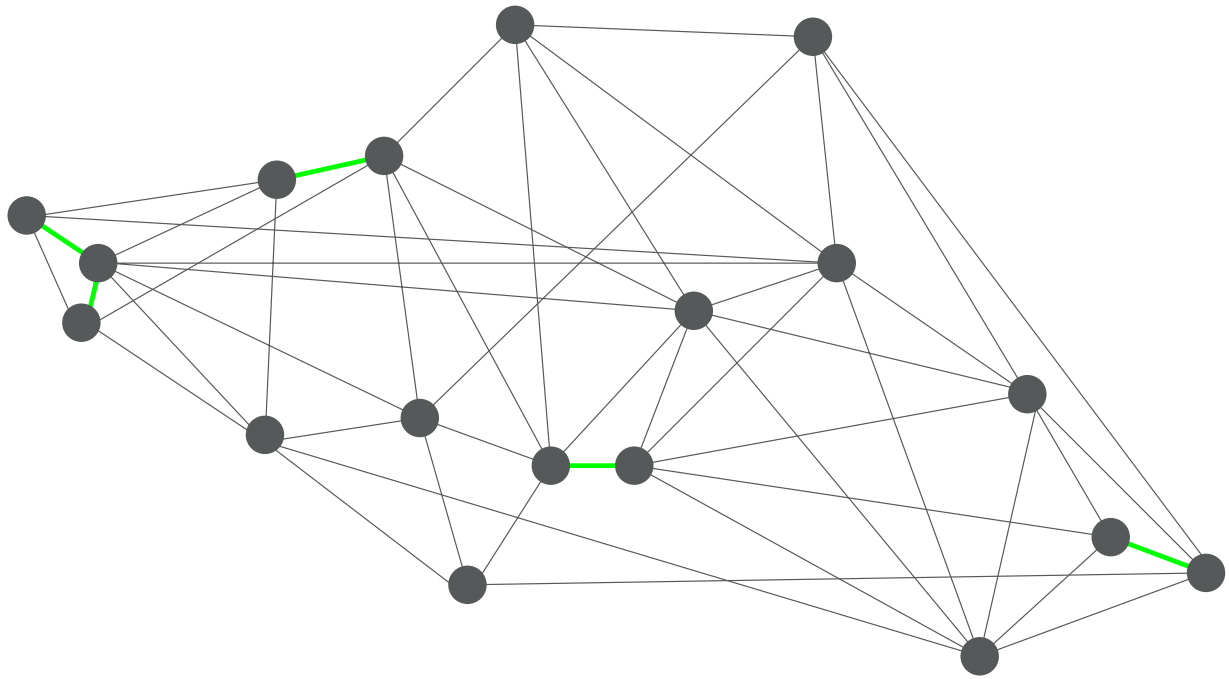
ACM : Kruskal



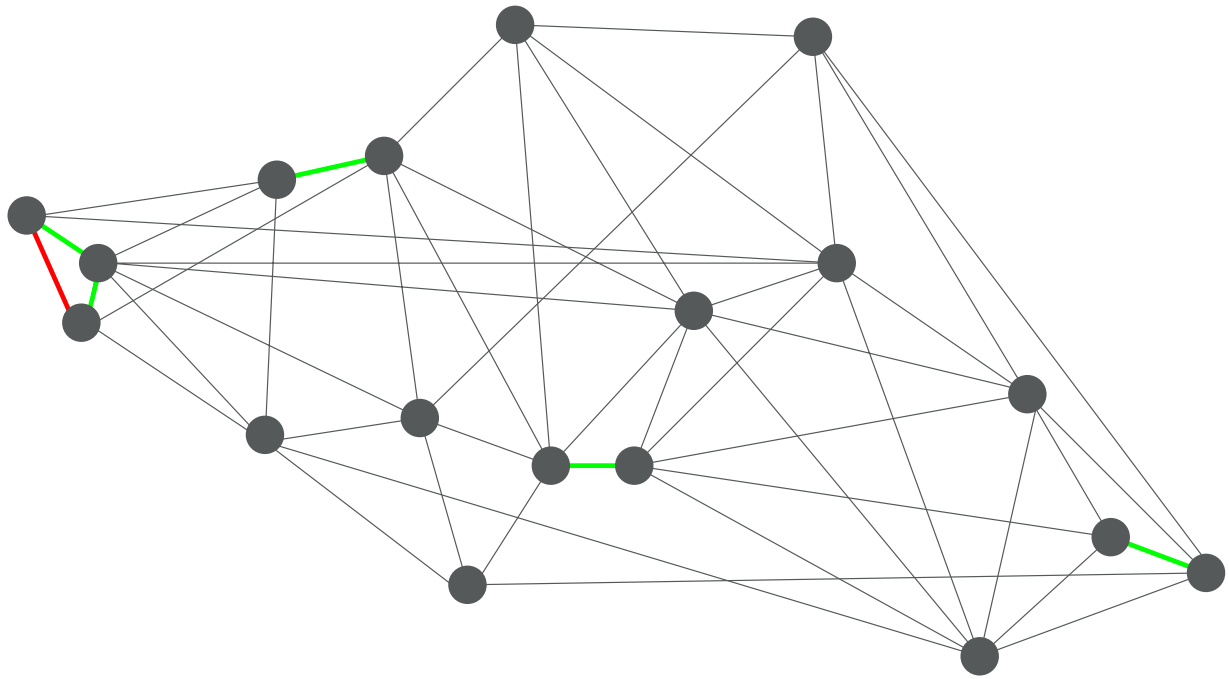
ACM : Kruskal



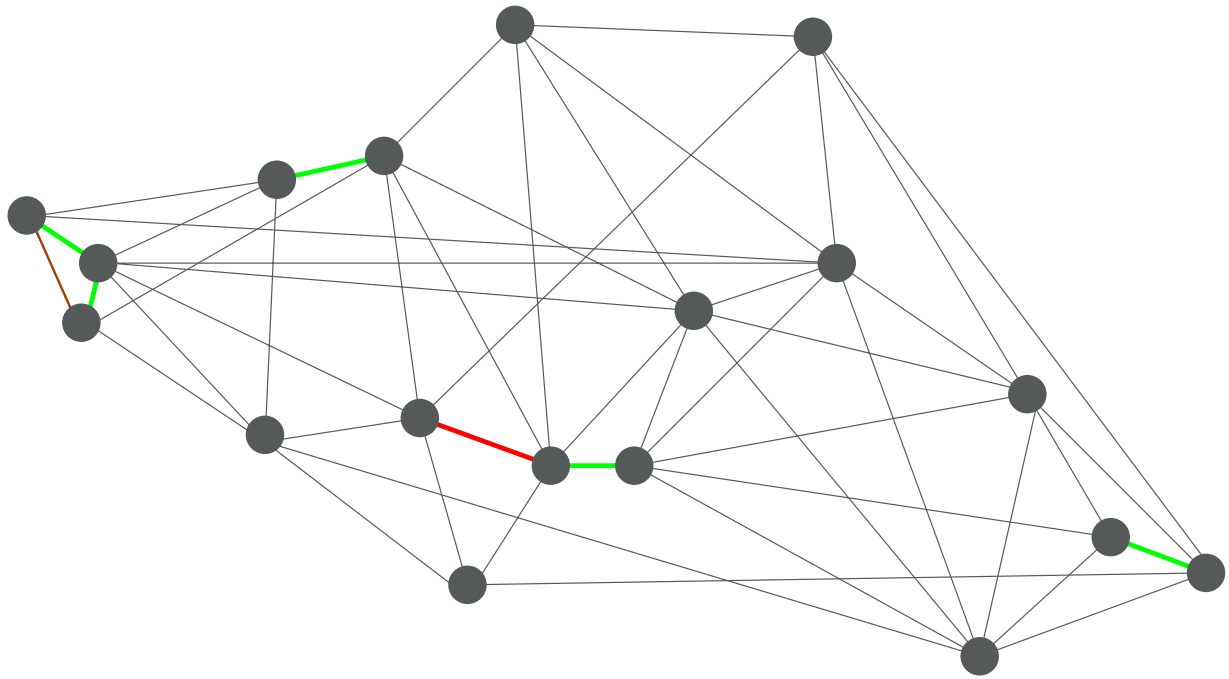
ACM : Kruskal



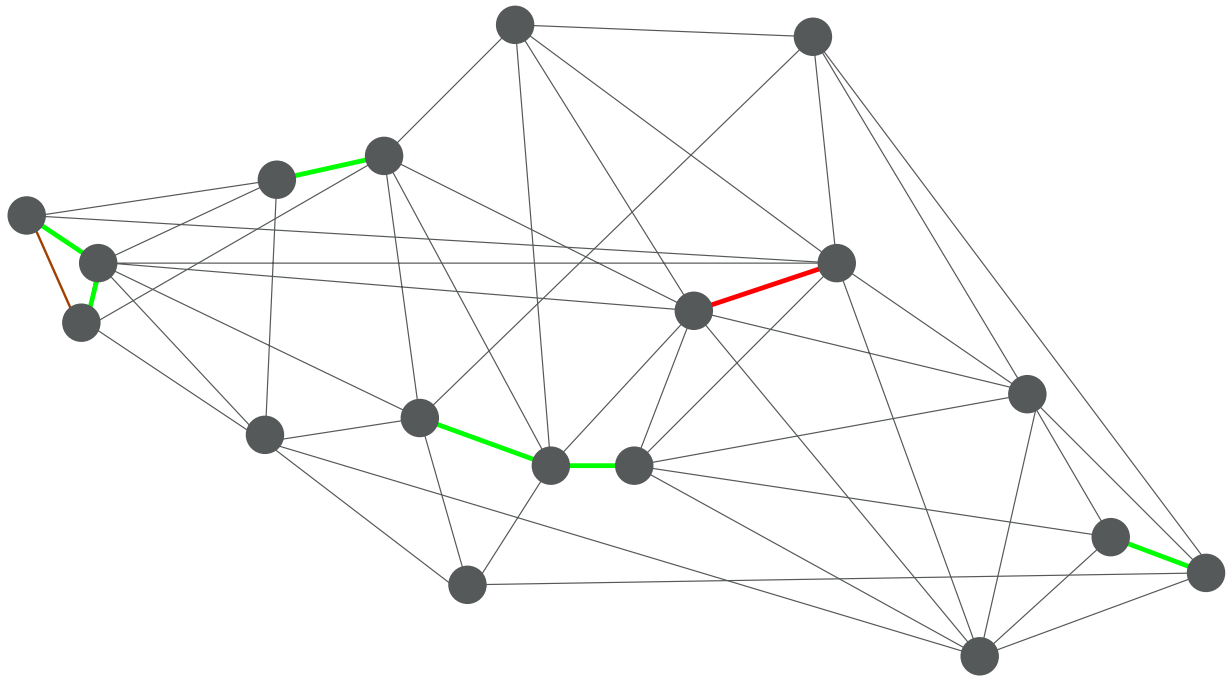
ACM : Kruskal



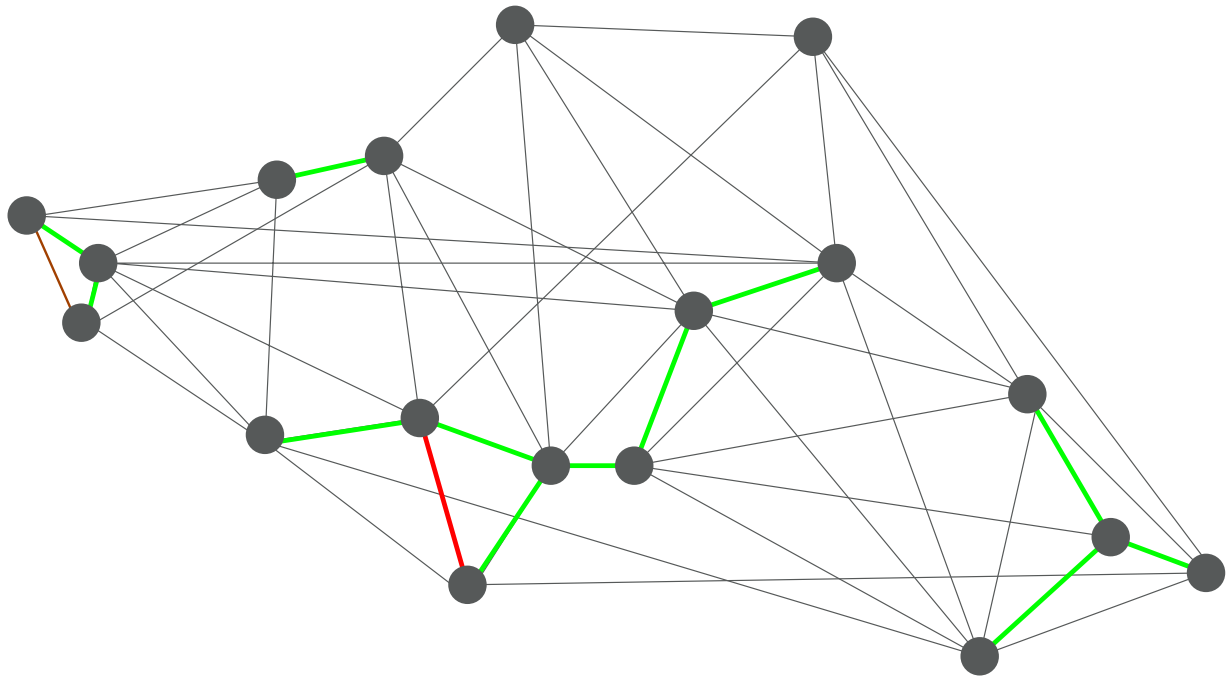
ACM : Kruskal



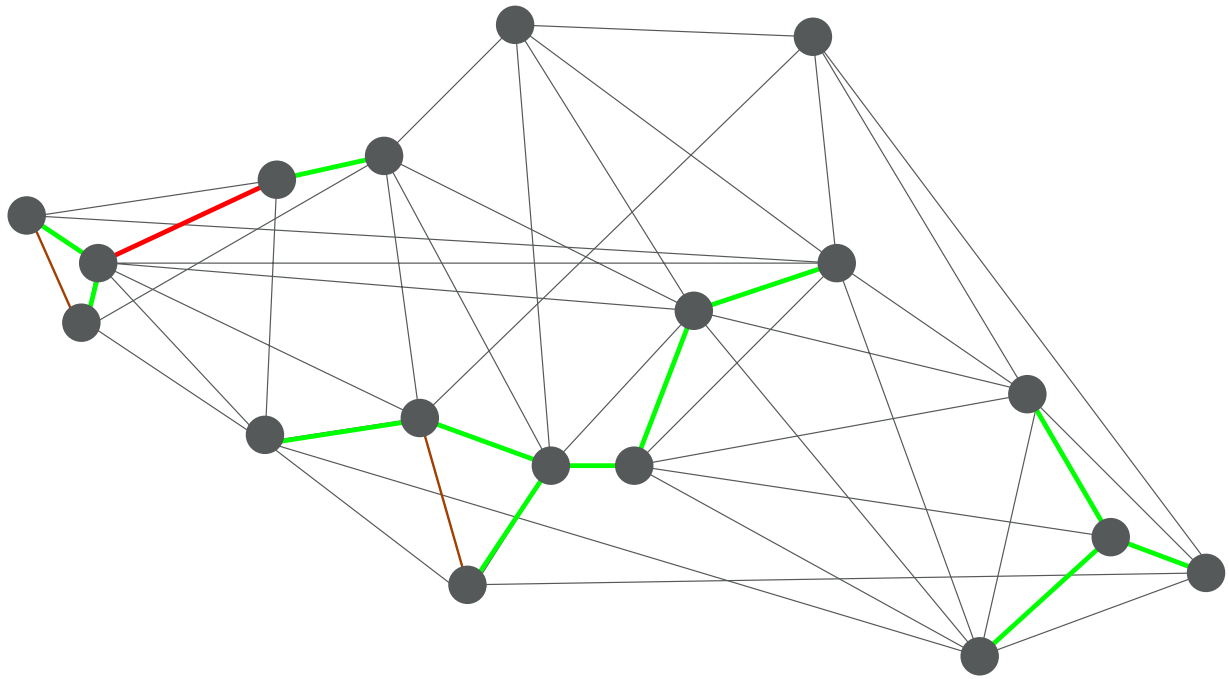
ACM : Kruskal



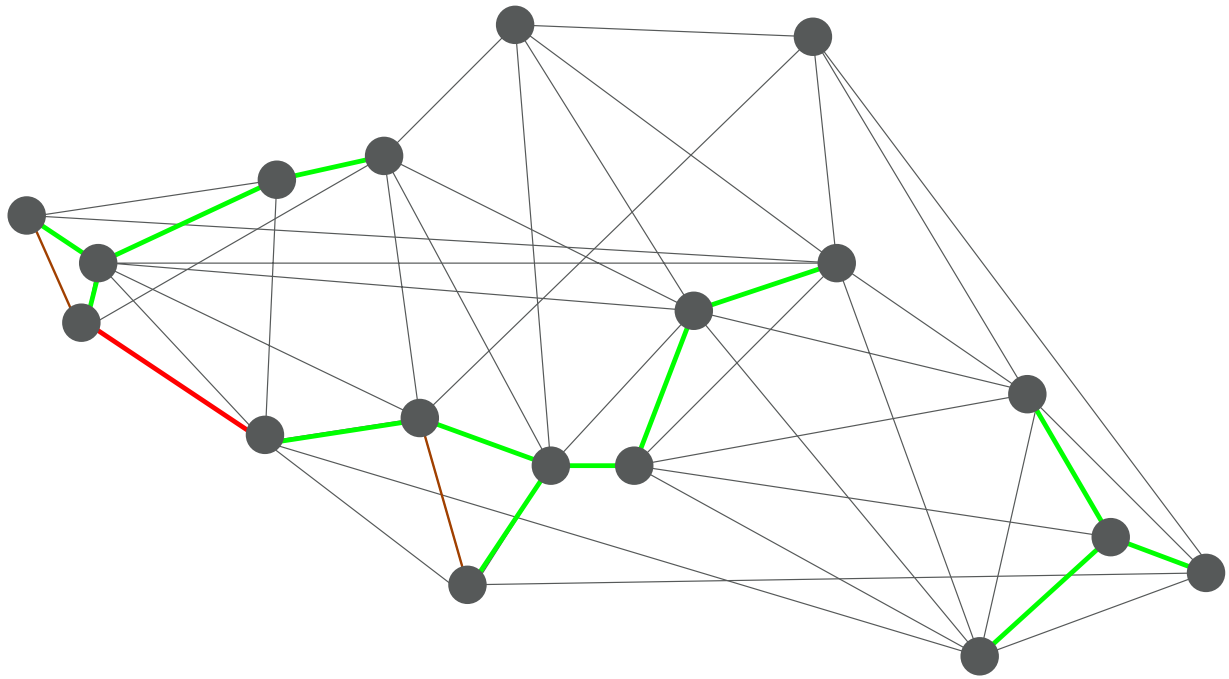
ACM : Kruskal



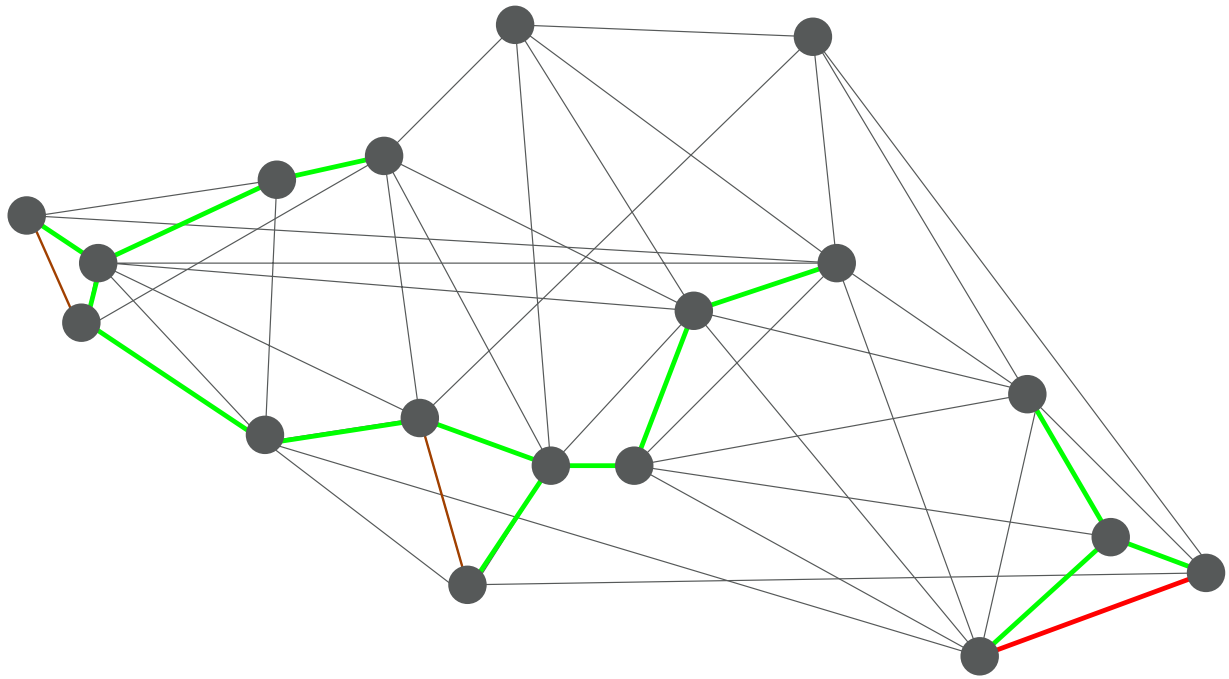
ACM : Kruskal



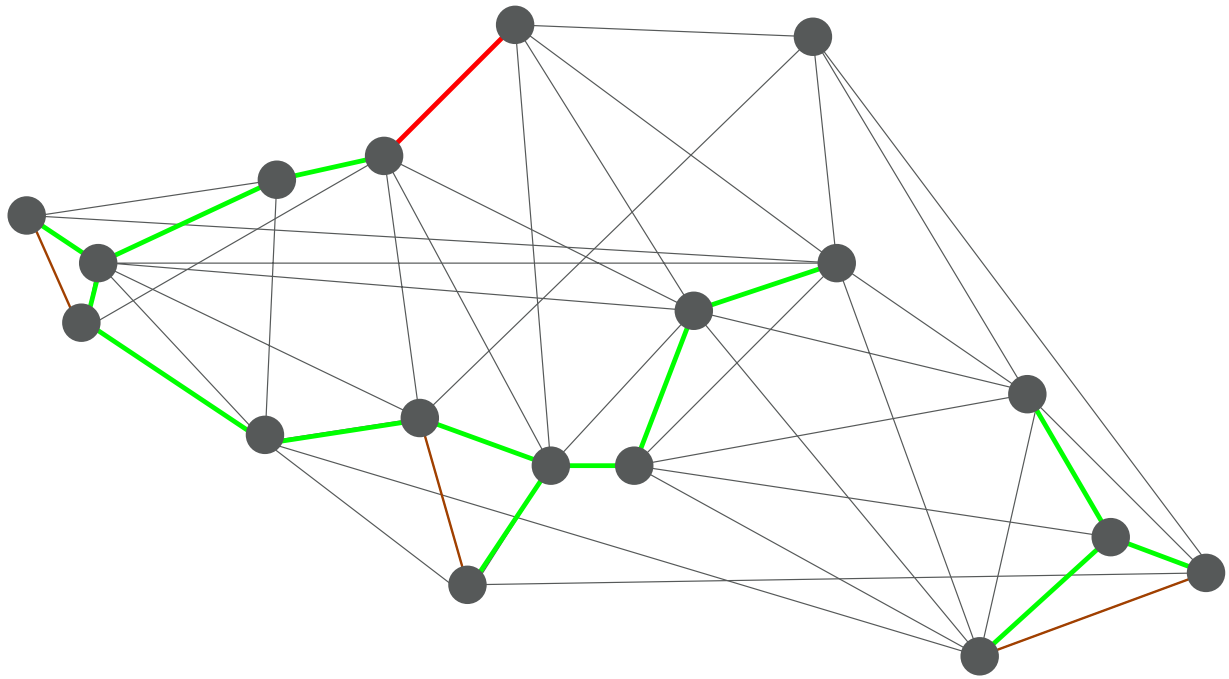
ACM : Kruskal



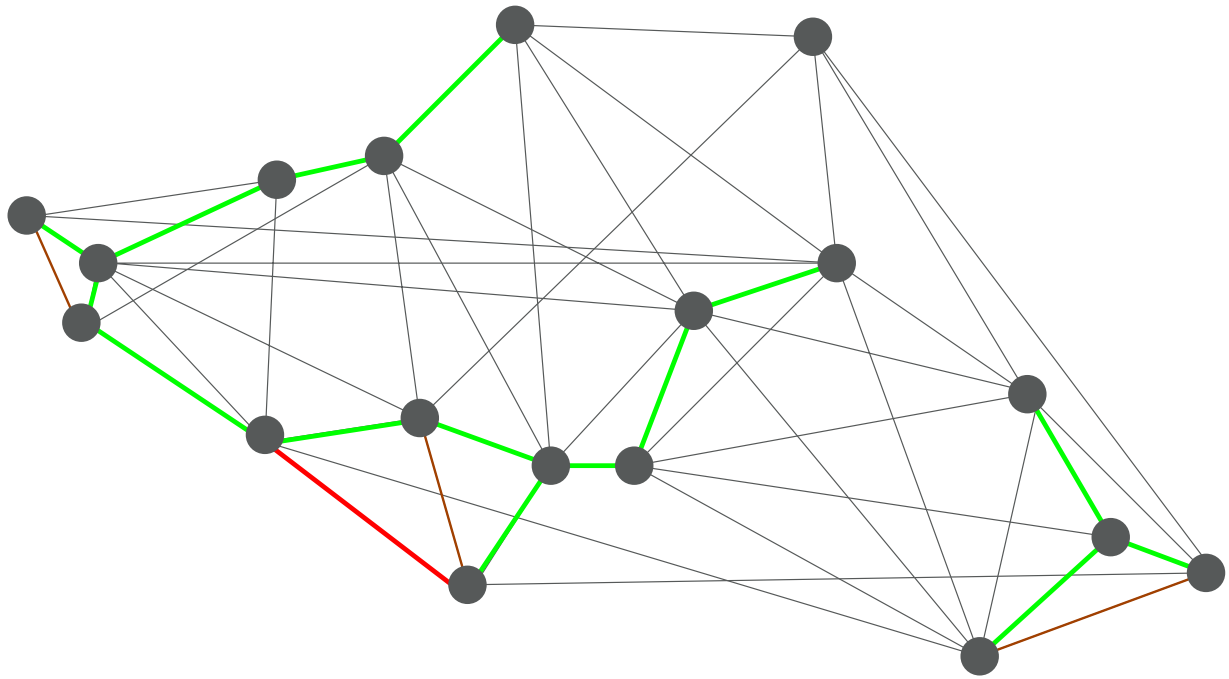
ACM : Kruskal



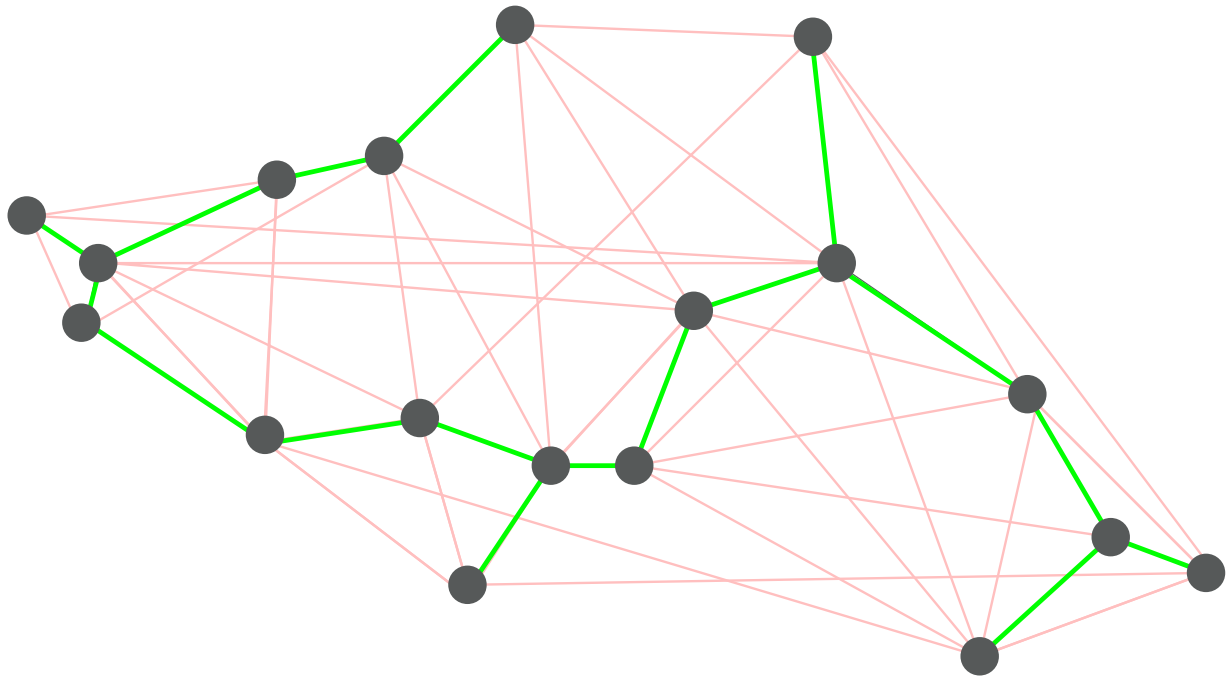
ACM : Kruskal



ACM : Kruskal



ACM : Kruskal



ACM : Kruskal

fonction **ACM-Kruskal**(G, w)
(**in :** $G = (S, A)$ un graphe, w les poids des arcs)

```
1  T := ∅,
2  pour chaque sommet v ∈ S faire
3        CreerEnsemble(v),
4  U := A,
5  tant que T n'est pas un ACM (i.e. |T| < |S| − 1) faire
6        choisir une arête (u, v) ∈ U de poids minimal,
7        si (Ensemble(u)≠Ensemble(v)) alors
8              T := T ∪ {(u, v)},
9              Reunir(Ensemble(u), Ensemble(v)),
10       U := U − (u, v),
11 renvoyer T,
```

ACM : Kruskal

Choix d'une arête de poids minimal :

- tri initial des arêtes
- parcours de la suite des arêtes triée

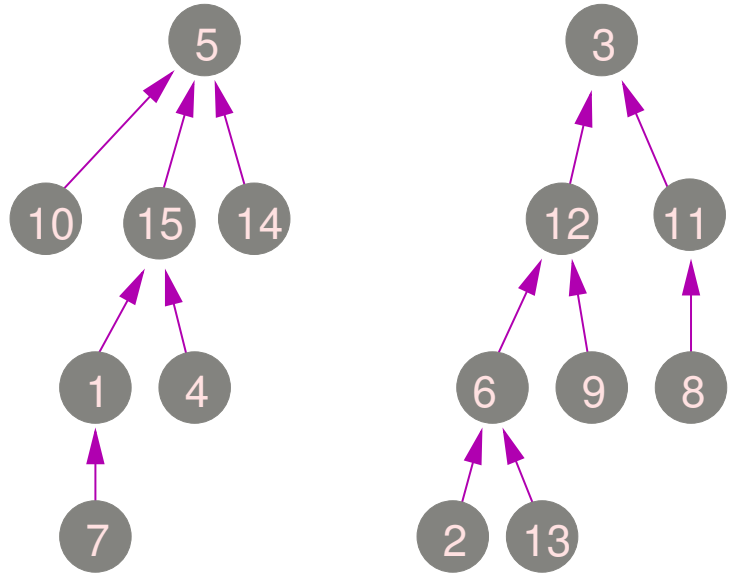
Complexité.

$$O(|A| \times \log|A|)$$

si on utilise une structure d'ensembles disjoints pour représenter les composantes en construction.

ACM : Kruskal : ensembles disjoints

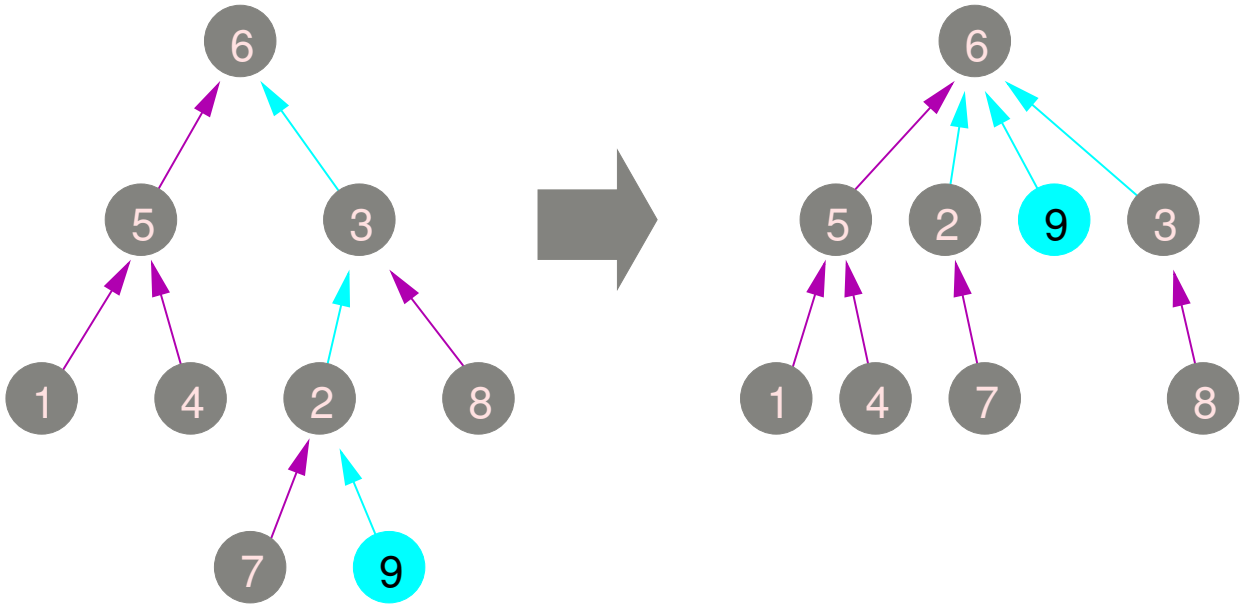
Représentation d'ensembles disjoints par des forêts



pointeurs vers les pères

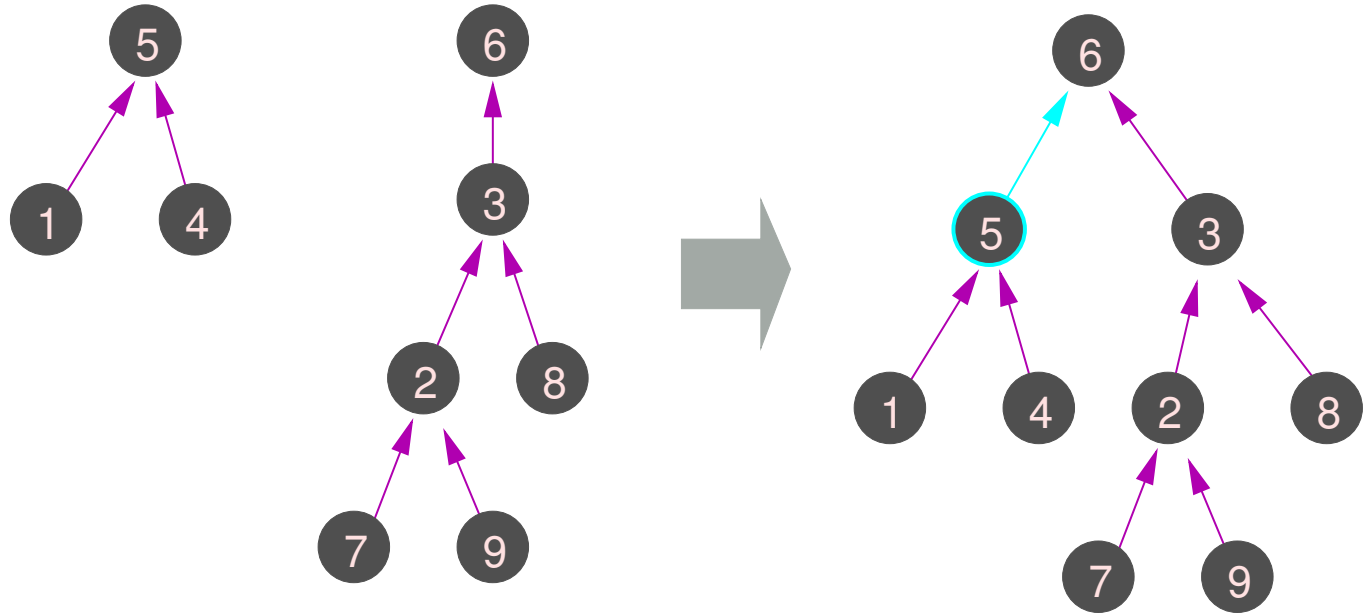
ACM : Kruskal : ensembles disjoints

Compression des chemins



ACM : Kruskal : ensembles disjoints

Union par rangs



Le *rang* est un majorant de la hauteur.

ACM : Kruskal : ensembles disjoints

fonction **CreerEns**(x)
    Pere[x] :=x,
    Rang[x] :=0,
fin fonction

fonction **Ensemble**(x)
    si ($x \neq$Pere[x]) **alors**
        Pere[x] :=Ensemble(Pere[x]),
    Renvoyer (Pere[x]),
fin fonction

ACM : Kruskal : ensembles disjoints

```
procedure Reunir(x, y)
    si (Rang[x] > Rang[y]) alors
        Pere[y] :=x,
    sinon
        Pere[x] :=y,
        si (Rang[x] = Rang[y]) alors
            Rang[y] :=Rang[y]+1,
    fin si
fin procedure
```

ACM : Kruskal : ensembles disjoints

Coût amorti d'une séquence de k opérations

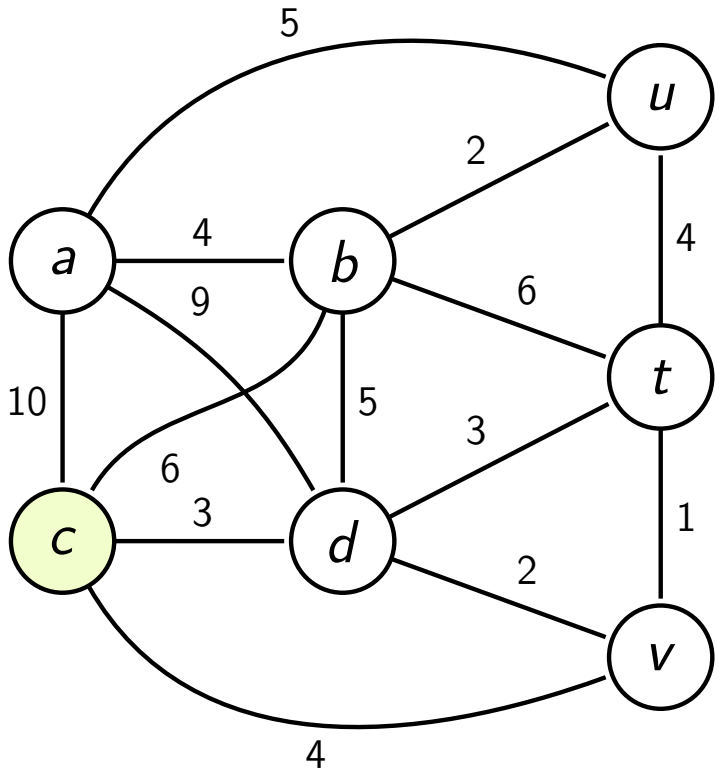
$$O(\alpha(k)) = O(5 \cdot k)$$

Le coût amorti *par opération* est donc **constant.**

$\alpha(k)$: réciproque de la fonction d'Ackermann $A(k, k)$.

La fonction d'Ackermann croît très vite.

ACM : exemple



$(t, v) : 1$
$(b, u) : 2$
$(d, v) : 2$
$(c, d) : 3$
$(d, t) : 3$
$(u, t) : 4$
$(v, c) : 4$
$(a, b) : 4$
$(b, d) : 5$
$(a, u) : 5$
$(b, t) : 6$
$(b, c) : 6$
$(a, d) : 9$
$(a, c) : 10$

Problème du voyageur de commerce

$G = (S, A, w)$ un graphe pondéré, complet, non orienté.

$S = \{s_1, \ldots, s_n\}$

Problème : trouver le *parcours* $(s_{i_1}, \ldots, s_{i_n})$ qui minimise

$$\sum_{j=1}^{n-1} w(s_{i_j}, s_{i_{j+1}}) + w(s_{i_n}, s_{i_1})$$

Problème du voyageur de commerce

$G = (S, A, w)$ un graphe pondéré, complet, non orienté.

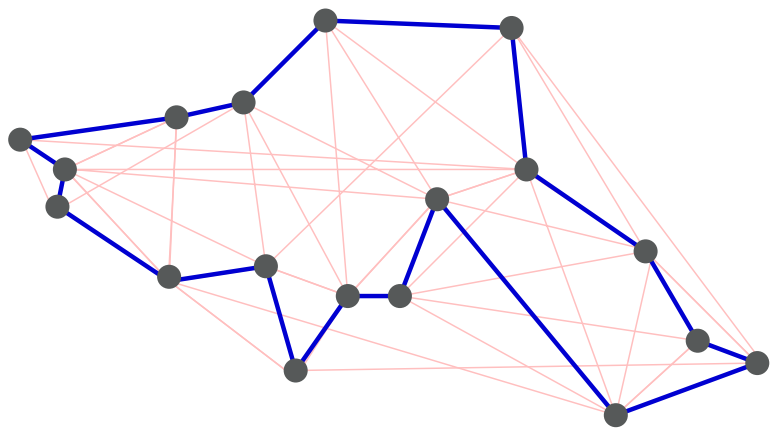
$S = \{s_1, \ldots, s_n\}$

Problème : trouver le *parcours* $(s_{i_1}, \ldots, s_{i_n})$ qui minimise

$$\sum_{j=1}^{n-1} w(s_{i_j}, s_{i_{j+1}}) + w(s_{i_n}, s_{i_1})$$

Pas d'algorithme polynomial connu.

Enumération de tous les parcours : $\mathcal{O}((n-1)!)$



Problème du voyageur de commerce

Algorithme d'approximation (facteur 2) :

- calcul d'un arbre couvrant T de poids minimal,

  la longueur d'un parcours quelconque est au plus $poids(T)$
- parcours en profondeur de l'arbre,

  chaque arête est parcourue deux fois
- élimination des occurences multiples de sommets.

Le poids des arêtes doit vérifier l'inégalité triangulaire.