REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Larbi Ben M'Hidi Oum El Bouaghi



**Cours Méthodes algorithmiques**
**1ere année master AD**

# Chapitre 1
# Méthode gloutonne

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Larbi Ben M'Hidi Oum El Bouaghi

**Cours Méthodes algorithmiques**
**1ere année master AD**

# Chapitre 1
# Méthode gloutonne



**Définition**



**Stratégie**



**Schéma**



**Exemples**

# Les algorithmes gloutons

**Définition des algorithmes gloutons** :

Les algorithmes gloutons

sont une classe d'algorithmes qui,

pour résoudre un problème, prennent des décisions successives en choisissant à chaque étape **l'option qui semble la meilleure**,

sans revenir en arrière.

Ils sont utilisés pour les problèmes **d'optimisation**.

# Les algorithms gloutons

**Définition des algorithmes gloutons** :

- **Un algorithme glouton**

  fait un choix localement optimal dans l'espoir que ce choix mènera à la solution globalement optimale

- **Les algorithmes gloutons**

  ne conduisent pas toujours à la solution optimale

# Les algorithms gloutons

**Stratégie Gloutonne**

- La **stratégie gloutonne** est une approche algorithmique qui repose sur la prise de décisions séquentielles.
- À chaque étape, l'algorithme fait un choix en fonction d'une **règle locale d'optimalité**,
- c'est-à-dire qu'il sélectionne la meilleure option disponible à l'instant présent, sans se préoccuper des conséquences futures.

# La stratégie gloutonne

## Choix glouton

À chaque étape, l'algorithme sélectionne l'option qui semble localement la meilleure (selon une certaine mesure : coût minimal, profit maximal, etc).

## Récursivité ou itérativité

Après chaque décision, le problème restant est réduit et l'algorithme recommence à choisir la meilleure option pour le sous-problème suivant.

## Finalité

La solution complète est obtenue une fois que toutes les décisions successives ont été prises

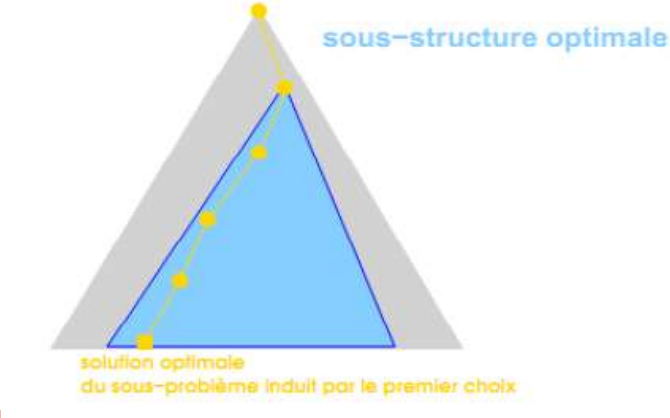
# La stratégie gloutonne

**Preuve d'Optimalité**

Pour qu'un algorithme glouton soit efficace et donne une solution optimale, le problème doit respecter deux propriétés clés :

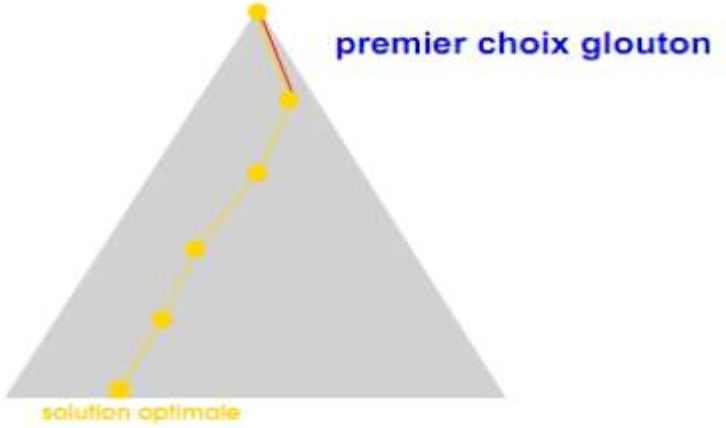
**Propriété de sous-structure optimale** :

Le problème doit pouvoir être décomposé en sous-problèmes dont les solutions optimales peuvent être combinées pour former la solution optimale globale. Autrement dit, résoudre une partie du problème de manière optimale garantit une solution optimale globale.



# La stratégie gloutonne

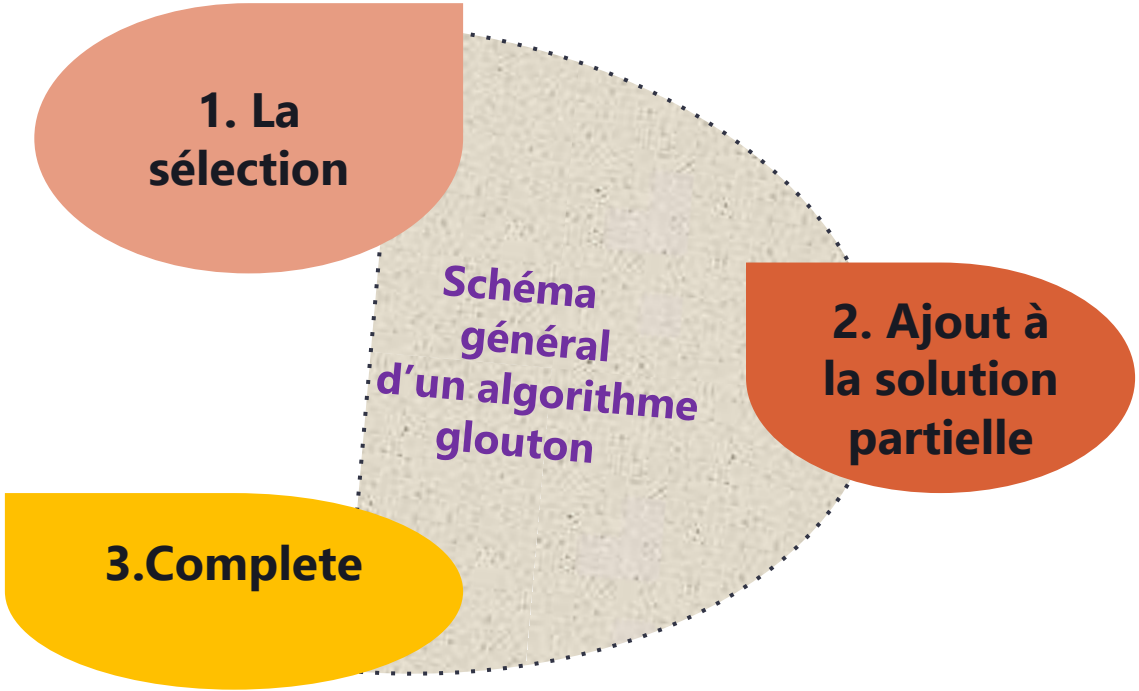
- **Propriété du choix glouton** :
  Le choix localement optimal à chaque étape doit conduire à une solution optimale globale.
- La meilleure solution peut être atteinte simplement en prenant des décisions locales sans avoir besoin de revenir sur des choix passés ou d'explorer d'autres options.



# Schéma général d’un algorithme glouton

- La méthode gloutonne se base sur un critère de sélection local pour construire une solution optimale.
- À chaque étape, un choix est fait parmi les éléments restants de E en fonction d’un **critère glouton**.
- Voici les principales étapes du schéma général :

# Schéma général d’un algorithme glouton



# Schéma général d’un algorithme glouton

**1. Sélection**

- Une fonction **select** est définie pour choisir, parmi les candidats restants, l’élément qui semble être le meilleur selon le critère local.
- Ce critère est souvent basé sur une mesure simple (par exemple, le poids minimal, la valeur maximale, etc).
- Ce choix est dit **optimal localement**, car il maximise ou minimise une caractéristique immédiate, mais sans considérer la solution globale.

# Schéma général d’un algorithme glouton

**2. Ajout à la solution partielle**

- Après avoir sélectionné un élément selon le critère glouton, cet élément est ajouté à la solution partielle.
- La solution partielle est donc augmentée à chaque itération

# Schéma général d’un algorithme glouton

**3. Vérification de la complétion (complete?) :**

À chaque itération, la fonction **complete?** teste si la solution partielle obtenue est une solution complète au problème.

Autrement dit, elle vérifie si l’ensemble des éléments sélectionnés constitue une solution valide. À ce stade, on ne se soucie pas encore de l'optimalité globale, mais seulement de la complétude de la solution.

Si la solution est complète, l'algorithme s'arrête.

Sinon, l'algorithme continue d’ajouter des éléments.

# Exemples

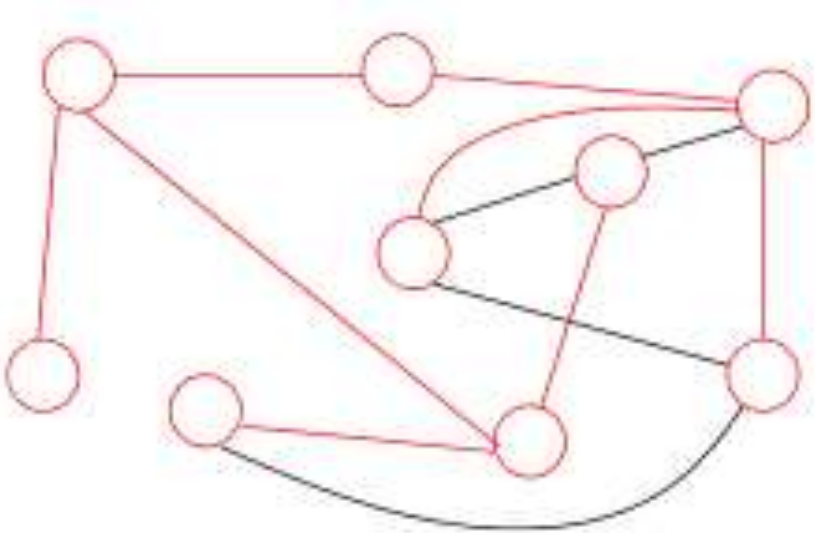
**Problèmes de Graphes :**
**Arbre couvrant minimal**

**Problème de du rendu de monnaie**



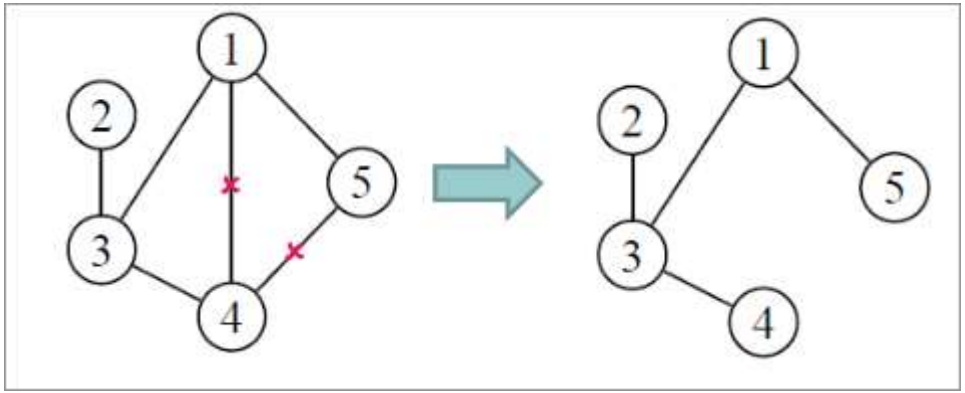
# Exemple1

**Problèmes de Graphes : Arbre couvrant minimal**



# Arbre couvrant minimal

Un **arbre couvrant minimal** (ACM) est un sous-ensemble d'arêtes d'un graphe connecté et non orienté qui relie tous les sommets ensemble avec un coût total minimal (somme des poids des arêtes).



# Arbre couvrant minimal

Il existe plusieurs problèmes liés aux arbres couvrants minimaux:

- Problème de base : Trouver un arbre couvrant minimal
- Problème du second meilleur arbre couvrant minimal
- Problème de l'ACM avec contraintes
- Problème de Steiner : ACM avec sommets spécifiques

# Arbre couvrant minimal

**Problème de base : Trouver un arbre couvrant minimal**

Étant donné un graphe connecté, non orienté et pondéré, trouvez un sous-ensemble d'arêtes formant un arbre couvrant minimal.

Chaque arête du graphe a un poids, et l'objectif est de minimiser la somme des poids des arêtes utilisées pour couvrir tous les sommets.

# Arbre couvrant minimal

**Problème de base : Trouver un arbre couvrant minimal**

**Algorithmes classiques :**

•**Algorithme de Kruskal** : Fonctionne en ajoutant les arêtes les moins coûteuses tout en évitant les cycles grâce à la structure **Union-Find**.

•**Algorithme de Prim** : Démarre à un sommet donné et étend l'arbre en ajoutant les arêtes de moindre coût reliant un nouveau sommet à l'arbre.

# Arbre couvrant minimal

**Problème de base : Trouver un arbre couvrant minimal**

**Application :**

•Réseaux électriques ou de communication (construire un réseau avec un coût minimal).



•Problème de construction d'infrastructure comme les routes ou les pipelines.



# Arbre couvrant minimal

**Problème du second meilleur arbre couvrant minimal**
Trouvez le deuxième arbre couvrant minimal d'un graphe.

C'est-à-dire, trouvez un arbre couvrant qui a un coût légèrement supérieur au premier ACM, mais qui reste le plus petit possible parmi tous les autres arbres couvrants.

# Arbre couvrant minimal

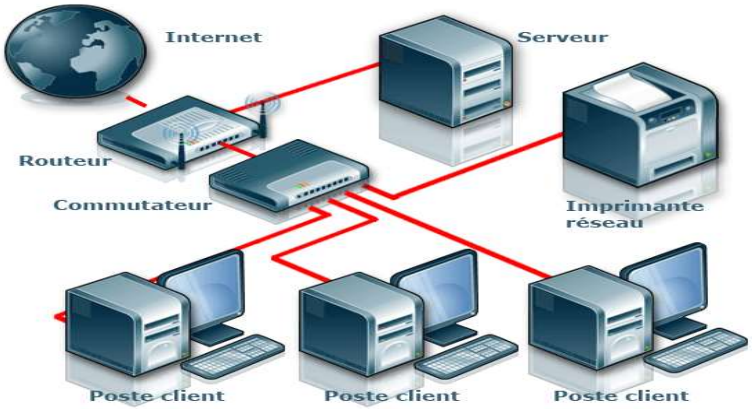
**Problème du second meilleur arbre couvrant minimal**

L’idée est de Trouver l'ACM avec un algorithme comme Kruskal ou Prim. Ensuite, essayer de trouver des arêtes à remplacer dans l'ACM tout en maintenant un arbre couvrant valide, en recalculant le coût total.

# Arbre couvrant minimal

**Problème du second meilleur arbre couvrant minimal**

Il est efficace dans le cas des Scénarios de redondance dans les réseaux, où le premier chemin peut être indisponible en raison de défaillances.



# Arbre couvrant minimal

**Problème de l'ACM avec contraintes**

On doit trouver un arbre couvrant minimal dans un graphe, mais certaines contraintes supplémentaires doivent être respectées.

Par exemple :
•Certaines arêtes doivent absolument être présentes ou absentes dans l'ACM.
•Les arêtes ont des coûts qui changent avec le temps, et l'ACM doit être recalculé périodiquement.

# Arbre couvrant minimal

**Problème de l'ACM avec contraintes**

**Approche :**

•**Algorithme de Kruskal avec modifications** : S'assurer que les arêtes obligatoires sont incluses dès le début et exclure celles qui ne doivent pas être prises.

•**Algorithmes dynamiques** : Si les poids des arêtes changent, il peut être nécessaire de réexécuter les algorithmes à chaque itération, mais des optimisations spécifiques peuvent être introduites.

# Arbre couvrant minimal

## Problème de l'ACM avec contraintes

Ce types résout les problèmes liés aux Réseaux en temps réel où les coûts des connexions varient (réseaux télécoms ou systèmes de transport).



# Arbre couvrant minimal

**Problème de Steiner : Arbre couvrant minimal avec sommets spécifiques**

Étant donné un graphe avec des poids sur les arêtes, trouvez le sous-ensemble des arêtes qui connectent un sous-ensemble spécifique de sommets tout en minimisant le coût total des arêtes utilisées.

Les autres sommets peuvent être utilisés, mais ne doivent pas nécessairement être connectés.

# Arbre couvrant minimal

**Problème de Steiner : Arbre couvrant minimal avec sommets spécifiques**
**Application :**
•Réseaux de communication où seulement certains sites ou utilisateurs spécifiques doivent être connectés, mais d'autres sites intermédiaires peuvent être utilisés pour réduire le coût.



# Arbre couvrant minimal

**Problème de l'ACM dans un graphe dynamique**

Vous avez un graphe dont les arêtes ou les poids des arêtes peuvent changer au fil du temps (par exemple, des arêtes peuvent être ajoutées ou supprimées).

Vous devez maintenir un ACM efficace à chaque modification du graphe.

# Arbre couvrant minimal

**Problème de l'ACM dans un graphe dynamique**

Utiliser des techniques d'algorithmes dynamiques pour recalculer un ACM efficacement à chaque changement plutôt que de recalculer tout depuis le début.

•Des techniques comme **l'arbre des coupes** peuvent être utilisées pour des mises à jour plus rapides.

# Arbre couvrant minimal

**Problème de l'ACM dans un graphe dynamique**
**Application :**
•Réseaux dynamiques comme les réseaux de transport, où les chemins disponibles changent constamment en raison des conditions de circulation ou des accidents.
•Réseaux de communication qui s'adaptent aux changements de topologie.



# Algorithme de Kruskal

L'algorithme de Kruskal est un exemple classique d'algorithme glouton, souvent utilisé pour résoudre le problème de l'arbre couvrant minimal dans un graphe non orienté et pondéré.

L'objectif est de trouver un sous-ensemble d'arêtes qui forme un arbre couvrant (un arbre qui connecte tous les sommets) et minimise la somme des poids des arêtes.

# Algorithme de Kruskal

***Étapes de l'algorithme de Kruskal***

1. Initialisation : Chaque sommet est un composant distinct.
2. Tri des arêtes : Trie des arêtes selon le poids.
3. Sélection des arêtes : Sélection des arêtes les moins coûteuses (qui ne forment pas des cycles).
4. Arrêt : S'arrête quand n - 1 arêtes sont sélectionnées.

# Algorithme de Kruskal

## *Caractéristiques gloutonnes de l'algorithme de Kruskal*

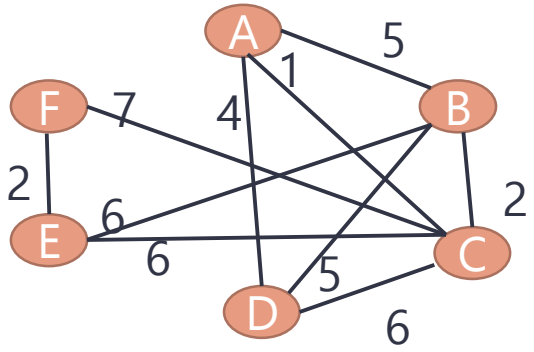
**Choix local optimal :** À chaque étape, l'algorithme fait un choix localement optimal en sélectionnant l'arête de poids minimal.

**Propriété de sous-structure optimale :** L'algorithme garantit que la solution obtenue est globale (l'arbre couvrant minimal) en construisant la solution par des choix locaux optimaux successifs.

# Algorithme de Kruskal

## *Exemple d'application de l'algorithme de Kruskal*

Supposons un graphe avec les sommets A, B, C, D,E,F et les arêtes suivantes :



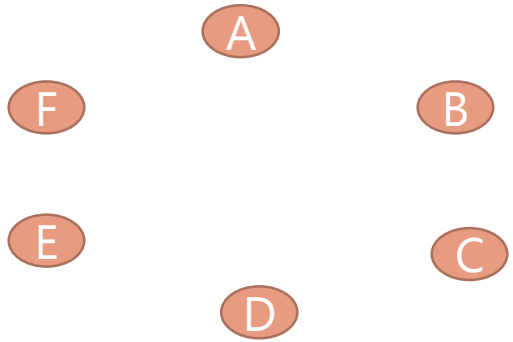
# Algorithme de Kruskal

## *Exemple d'application de l'algorithme de Kruskal*

Trier les arêtes :

| Poids | Arêtes |
|---|---|
| 1 | AC |
| 2 | BC,EF |
| 4 | AD |
| 5 | AB,BD, |
| 6 | BE,CD,CE |
| 7 | CF |

| Sommet | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| CC | A | B | C | D | E | F |



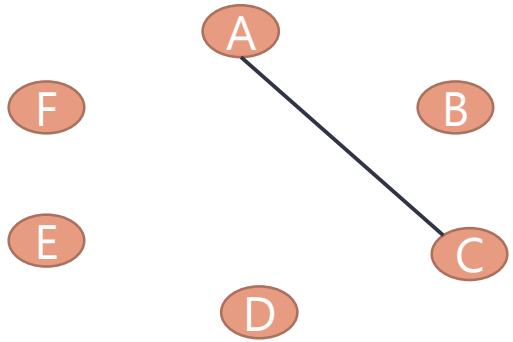
# Algorithme de Kruskal

## *Exemple d'application de l'algorithme de Kruskal*

Trier les arêtes :

| Poids | Arêtes |
|---|---|
| 1 | AC |
| 2 | BC,EF |
| 4 | AD |
| 5 | AB,BD, |
| 6 | BE,CD,CE |
| 7 | CF |

| Sommet | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| CC | a | B | a | D | E | F |



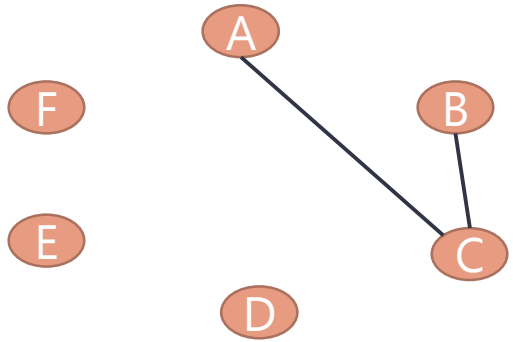
# Algorithme de Kruskal

## *Exemple d'application de l'algorithme de Kruskal*

Trier les arêtes :

| Poids | Arêtes |
|---|---|
| 1 | AC |
| 2 | BC,EF |
| 4 | AD |
| 5 | AB,BD, |
| 6 | BE,CD,CE |
| 7 | CF |

| Sommet | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| CC | a | a | a | D | E | F |



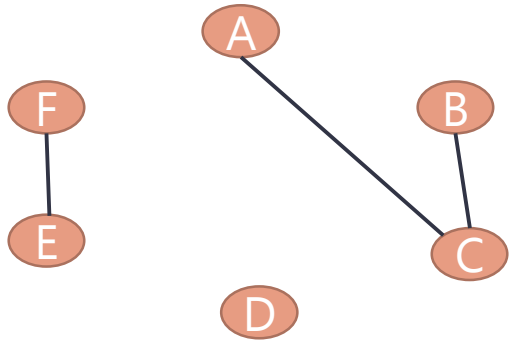
# Algorithme de Kruskal

## *Exemple d'application de l'algorithme de Kruskal*

Trier les arêtes :

| Poids | Arêtes |
|---|---|
| 1 | AC |
| 2 | BC,EF |
| 4 | AD |
| 5 | AB,BD, |
| 6 | BE,CD,CE |
| 7 | CF |

| Sommet | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| CC | a | a | a | D | e | e |



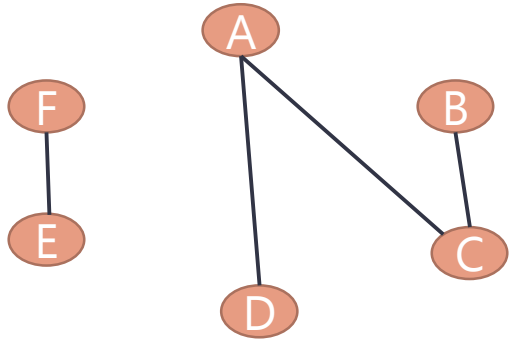
# Algorithme de Kruskal

## *Exemple d'application de l'algorithme de Kruskal*

Trier les arêtes :

| Poids | Arêtes |
|---|---|
| 1 | AC |
| 2 | BC,EF |
| 4 | AD |
| 5 | AB,BD, |
| 6 | BE,CD,CE |
| 7 | CF |

| Sommet | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| CC | a | a | a | a | e | e |



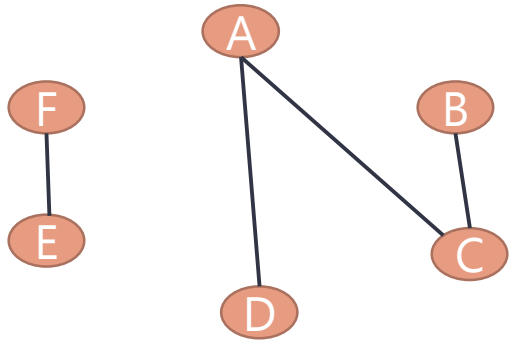
# Algorithme de Kruskal

## *Exemple d'application de l'algorithme de Kruskal*

Trier les arêtes :

| Poids | Arêtes |
|---|---|
| 1 | AC |
| 2 | BC,EF |
| 4 | AD |
| 5 | ~~AB~~,~~BD~~, |
| 6 | BE,CD,CE |
| 7 | CF |

| Sommet | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| CC | a | a | a | a | e | e |



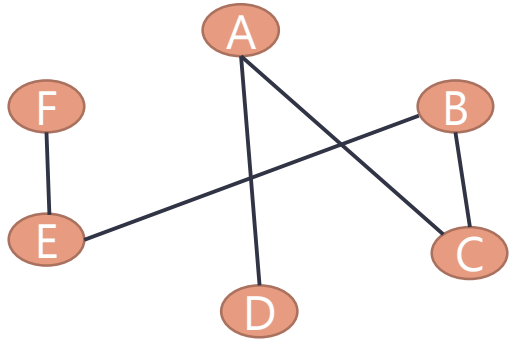
# Algorithme de Kruskal

## *Exemple d'application de l'algorithme de Kruskal*

Trier les arêtes :

| Poids | Arêtes |
|---|---|
| 1 | AC |
| 2 | BC,EF |
| 4 | AD |
| 5 | ~~AB~~,~~BD~~, |
| 6 | BE,CD,CE |
| 7 | CF |

| Sommet | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| CC | a | a | a | a | a | e |



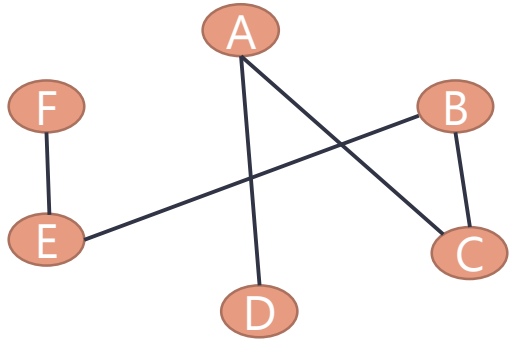
# Algorithme de Kruskal

## *Exemple d'application de l'algorithme de Kruskal*

Trier les arêtes :

| Poids | Arêtes |
|---|---|
| 1 | AC |
| 2 | BC,EF |
| 4 | AD |
| 5 | ~~AB~~,~~BD~~, |
| 6 | BE,~~CD~~,~~CE~~ |
| 7 | ~~CF~~ |

| Sommet | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| CC | a | a | a | a | a | e |

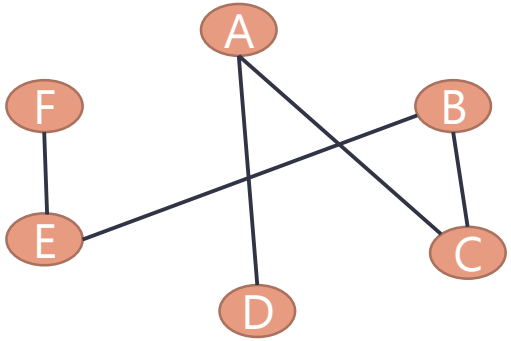


# Algorithme de Kruskal

## *Exemple d'application de l'algorithme de Kruskal*

Trier les arêtes :

| Poids | Arêtes |
|---|---|
| 1 | AC |
| 2 | BC, EF |
| 4 | AD |
| 5 | ~~AB~~, ~~BD~~, |
| 6 | BE, ~~CD~~, ~~CE~~ |
| 7 | ~~CF~~ |




Le poids de l'arbre couvrant :
1+2+2+4+6=15

# Algorithme de Prim

L'idée de base de l'algorithme est de construire l'arbre couvrant minimal de manière incrémentale, en commençant par un sommet quelconque et en ajoutant, à chaque étape, l'arête de poids minimum reliant un sommet déjà dans l'arbre à un sommet non encore dans l'arbre.

Ce processus est répété jusqu'à ce que tous les sommets soient inclus dans l'arbre couvrant.

# Algorithme de Prim

**Description de l'algorithme**

**1.Initialisation** :

1. Choisir un sommet de départ arbitraire.
2. Marquer ce sommet comme visité.
3. Initialiser une structure de données (comme une file de priorité) pour garder la trace des arêtes incidentes aux sommets visités.

# Algorithme de Prim

**Description de l'algorithme**

**2. Étape itérative** :

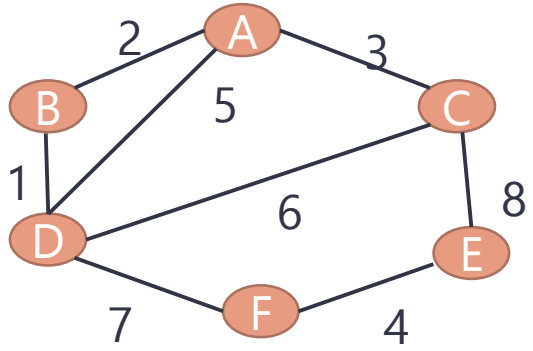
1. À chaque étape, choisir l'arête de poids minimal connectant un sommet visité à un sommet non visité.
2. Ajouter cette arête à l'arbre couvrant minimal.
3. Marquer le nouveau sommet comme visité.
4. Mettre à jour la structure de données en incluant les nouvelles arêtes incidentes aux sommets visités.

**3. Fin** :

1. Répéter l'étape itérative jusqu'à ce que tous les sommets soient visités.

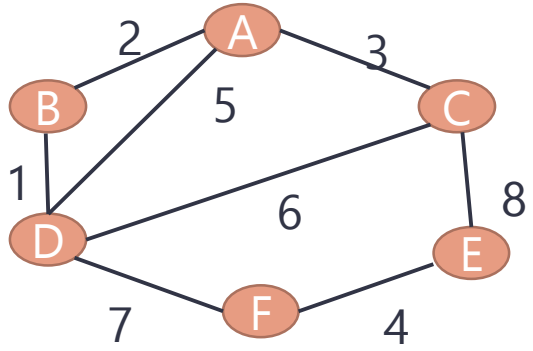
# Algorithme de Prim

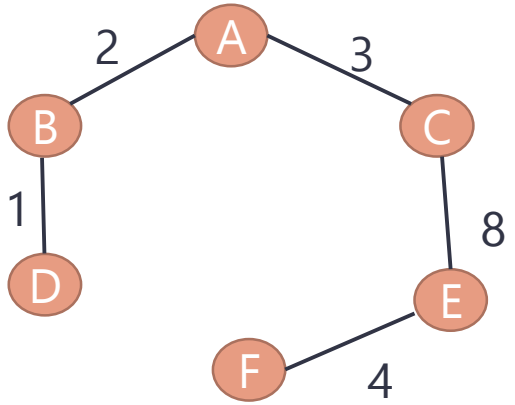
Appliquer l'algorithme de Prim pour trouver l'arbre couvrant minimal



# Algorithme de Prim

Appliquer l'algorithme de Prim pour trouver l'arbre couvrant minimal



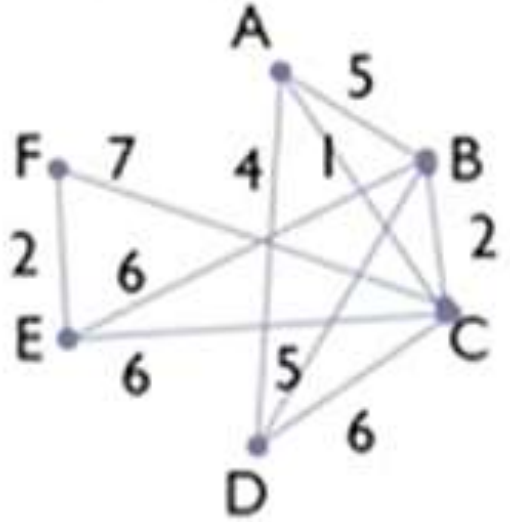



# Algorithme de Prim

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | | 2 A | 3 A | 5 A | | |
| B | | | 3A | 1 B | | |
| D | | | 3A | | | 7D |
| F | | | | | 4 F | |
| E | | | | | 4 F | |

Donc L'arbre couvrant minimal obtenu avec l'algorithme de Prim est constitué des arêtes : (A,B), (C,A), (D,B), (D,F), (F,E) de poids respectivement 2 + 3 +1 +7 + 4 =17

# Algorithme de Prim



# Algorithme de Prim

| sommet | A | | B | | C | | D | | E | | F | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Dist | proche | Dist | proche | Dist | proche | Dist | proche | Dist | proche | Dist | proche |
| init | ∞ | / | ∞ | / | ∞ | / | ∞ | / | ∞ | / | ∞ | / |
| A | | | 5 | A | 1 | A | 4 | A | | | | |
| C | | | 2 | C | | | 4 | A | 6 | C | 7 | C |
| B | | | | | | | 4 | A | 6 | C | 7 | C |
| D | | | | | | | | | 6 | C | 7 | C |
| E | | | | | | | | | | | 2 | E |

Une arête de moins que le nombre de sommets :
{{A,C},{B,C},{A,D},{C,E},{E,F}}

Poids de l'arbre couvrant = 1+2+4+6+2=15

# Algorithme de Prim

**Preuve de correction** :

Soit T un arbre couvrant construit par l'algorithme de Prim.
Supposons, pour le raisonnement par l'absurde, que T n'est pas minimal. Cela signifie qu'il existe un arbre couvrant minimal $T^*$ avec un poids total strictement inférieur à celui de T.

•Considérons la première arête (u,v) que l'algorithme de Prim ajoute à T mais qui n'appartient pas à $T^*$. En remplaçant (u,v) par une autre arête dans $T^*$, nous introduisons un cycle.

•Cependant, étant donné que l'arête (u,v)a le poids minimal parmi toutes les arêtes disponibles, cela conduit à une contradiction car cela signifierait que $T^*$ contient une arête de poids supérieur, ce qui est impossible dans un arbre couvrant minimal.

# Exemple2

problème du rendu de monnaie

# Rendu de monnaie

## Présentation du problème du rendu de monnaie

Le problème du rendu de monnaie est un problème fondamental en optimisation.

Il consiste à rendre une somme donnée à l'aide d'un ensemble de pièces de valeurs fixes, tout en utilisant le moins de pièces possible.

# Rendu de monnaie

**Énoncé formel :**

- On dispose d'un ensemble de pièces avec des valeurs $C=\{c_1,c_2,\ldots,c_n\}$, où $c_1<c_2<\cdots<c_n$ sont des entiers positifs.

- L'objectif est de rendre une somme S avec un nombre minimal de pièces issues de cet ensemble. Chaque pièce peut être utilisée autant de fois que nécessaire.

# Rendu de monnaie

**Étapes de l'algorithme :**

**1.Initialisation :**

1. On a une somme S à rendre avec un ensemble de pièces $C=\{c_1,c_2,\ldots,c_n\}$, triées en ordre décroissant $c_n \geq c_n-1 \geq \cdots \geq c_1$

2. On initialise une variable count pour comptabiliser le nombre de pièces utilisées.

# Rendu de monnaie

**2. Sélection gloutonne :**

1. On choisit la pièce de valeur ci la plus grande telle que $c_i \leq S$
2. On détermine combien de fois cette pièce peut être utilisée sans dépasser S, soit $k = S/. c_i$
3. On met à jour count avec count=count+k et on réduit la somme restante S de $k \times c_i$

**3. Répétition :**

1. Tant que $S \neq 0$, on recommence en sélectionnant la pièce la plus adaptée à la nouvelle somme.

**4. Arrêt :**

1. L'algorithme se termine quand S=0. Le nombre total de pièces utilisées est donné par la variable count.

# Rendu de monnaie

## Exemple de résolution

**Problème :** Nous disposons de pièces de valeurs 1, 5, et 10 unités, et devons rendre une somme de S=28.

# Rendu de monnaie

**Étapes de résolution avec la méthode gloutonne :**

**Étape 1 :** La somme à rendre est $S=28$.

- La pièce la plus grande inférieure ou égale à 28 est 10.
- On utilise 2 pièces de 10, ce qui laisse une somme restante de $28-2\times10=8$
- Le nombre total de pièces est de 2.

# Rendu de monnaie

**Étapes de résolution avec la méthode gloutonne :**

**Étape 2 :**

La somme restante est $S=8$.
La pièce la plus grande disponible est 5.
On utilise 1 pièce de 5, il reste donc $8-5=3$.
Le nombre total de pièces est maintenant de 3.

# Rendu de monnaie

**Étapes de résolution avec la méthode gloutonne :**
**Étape 3 :**

La somme restante est S=3.
On utilise 3 pièces de 1 (car ⌊3/1⌋=3.
La somme est maintenant réduite à 0.

Le nombre total de pièces est donc de 6.

# Rendu de monnaie

## Correctitude de la méthode gloutonne

La méthode gloutonne est efficace dans de nombreux cas, mais elle n’est pas toujours garantie de fournir une solution optimale.

Elle donne une solution correcte seulement si l'ensemble des pièces respecte certaines propriétés spécifiques.

# Rendu de monnaie

**Cas où l'algorithme échoue :**

Considérons un ensemble de pièces C={1,3,4} et une somme à rendre de S=6.

En suivant l'algorithme glouton :

- On commence par utiliser la pièce de 4, ce qui laisse 6−4=2.
- Puis on utilise 2 pièces de 1 pour rendre les 2 unités restantes.
- Résultat : 1 pièce de 4 et 2 pièces de 1, donc un total de 3 pièces.

Cependant, la solution optimale serait d'utiliser 2 pièces de 3, soit un total de 2 pièces, ce qui montre que l'algorithme glouton n'est pas optimal dans ce cas.

# Exemple 3

problème de location de camion

# Location du camion

Imaginons que vous devez transporter une certaine quantité de marchandises d'un entrepôt à un autre en utilisant des camions de différentes tailles (ou capacités) et coûts de location.

*L'objectif est de minimiser le coût total de location tout en transportant toute la marchandise.*

# Location du camion

**Stratégie gloutonne**

L'idée principale de cet algorithme est de toujours choisir le camion avec le **meilleur rapport coût/capacité** (coût par unité de marchandise transportée) jusqu'à ce que tout le volume soit transporté.

# Location du camion

**Stratégie gloutonne**
**Étapes :**

**1.Trier** les camions par coût unitaire croissant (coût par unité de marchandise transportée).

**2. Sélectionner** le camion ayant le **meilleur rapport coût/unité** et le remplir autant que possible.

**3. Répéter** le processus avec les camions suivants jusqu'à ce que tout le volume de marchandises soit transporté.

# Location du camion

**Algorithme glouton pour la location de camion**

**1.Entrée** :

1. Une liste de camions C, chaque camion $c_i$ ayant :
   - une capacité $cap_i$ (quantité de marchandise qu'il peut transporter),
   - un coût $cost_i$
2. Une quantité totale de marchandises M à transporter.

**2.Sortie** : Un sous-ensemble de camions S et leur nombre associé, qui minimise le coût total de transport.

# Location du camion

**Étapes de l'algorithme** :

1.Calculer le **rapport coût/capacité** $r_i = cost_i / cap_i$ pour chaque camion.

**2.** Trier les camions C par ordre croissant de $r_i$
3. Initialiser une variable $M_{restant} = M$ (la marchandise restante à transporter).

4. Initialiser un sous-ensemble vide de camions sélectionnés S.

# Location du camion

**Étapes de l'algorithme**

5. Pour chaque camion $c_i$ dans l'ordre trié :

Si $cap_i \leq M_{restant}$ , ajouter le camion $c_i$ à la solution S et mettre à jour $M_{restant} = M_{restant} - cap_i$

Si capi > Mrestant , fractionner la capacité du camion $c_i$ pour transporter $M_{restant}$, ajouter cette fraction à S, et mettre $M_{restant} = 0$

6. Sortir de la boucle quand $M_{restant} = 0$

7. Retourner le sous-ensemble S des camions et le coût total correspondant.

# Location du camion

Fondement mathématique

Fonction objectif (optimisation)

Le but est de minimiser le coût total de location pour transporter exactement M unités de marchandises. Cela peut être exprimé comme une **fonction objectif** :

$$\text{Minimiser} \sum_{i=1}^{n} x_i \cdot cost_i$$

où $x_i$ représente le nombre d'unités de marchandises transportées par le camion i, et $cost_i$ est le coût de location du camion i.

# Location du camion

Fondement mathématique

Fonction objectif (optimisation)

On doit respecter la contrainte suivante : $$\sum_{i=1}^{n} x_i \cdot cap_i = M$$

c'est-à-dire que la somme des marchandises transportées par tous les camions doit être égale à M, la quantité totale de marchandise

# Location du camion

**Propriété de solution optimale**
Le problème de location de camion satisfait la propriété dite de:

**Greedy choice property** : À chaque étape, en choisissant le camion avec le plus bas coût par unité de marchandise, nous ne compromettons pas la possibilité d'obtenir une solution optimale globale.

**Optimal substructure** : Un sous-problème du problème original (par exemple, transporter $M - capi$ unités de marchandises après avoir choisi un camion $c_i$) est lui-même un problème de même nature, et donc la même approche gloutonne peut être appliquée de manière récursive ou itérative.

# Location du camion

**Exemple :**

Supposons que vous avez trois camions avec les caractéristiques suivantes :
Camion 1 : Capacité = 10 tonnes, Coût = 100 unités
Camion 2 : Capacité = 20 tonnes, Coût = 180 unités
Camion 3 : Capacité = 30 tonnes, Coût = 270 unités
Vous devez transporter 50 tonnes.

# Location du camion

Calcul des rapports $r_i$ :

Camion 1 : $r_1$=10 unités/tonne

Camion 2 : $r_2$=9 unités/tonne

Camion 3 : $r_3$=9 unités/tonne

Tri des camions par $r_i$ : $C_2$,$C_3$,$C_1$

Sélection :

On choisit le camion 2 pour transporter 20 tonnes (coût = 180)

Il reste 30 tonnes à transporter

On choisit ensuite le camion 3 pour 30 tonnes (coût = 270)

Coût total = 180 + 270 = 450 unités