# TD1 algorithmes gloutons

# (Greedy algorithms)

## Exercice1 : PROBLÈME DE SÉQUENCEMENT DES TÂCHES

Étant donné un ensemble de travaux pour lesquels chaque travail a une date limite et les bénéfices associés si le travail est terminé avant la date limite. Il est également indiqué que chaque travail prend une seule unité de temps, de sorte que le délai minimum possible pour tout travail est 1. Comment maximiser le profit total si un seul travail peut être planifié à la fois.

**Algorithme glouton utilisé**

1. Triez tous les travaux par ordre décroissant de profit.
2. Initialisez la séquence de résultats avec le premier travail dans les travaux triés.
3. Faire pour les travaux restant (n-1)
    - Si le travail en cours peut tenir dans la séquence de résultats actuelle sans manquer la date limite, ajoutez-le au résultat. Sinon ignorer le travail en cours.

**Exemple :**

Supposons que nous ayons les tâches suivantes avec leurs délais et profits associés :

| Tâche | Délai (Deadline) | Profit |
|---|---|---|
| A | 2 | 100 |
| B | 1 | 50 |
| C | 2 | 60 |
| D | 1 | 70 |
| E | 3 | 80 |

**Solution :**

Nous allons résoudre ce problème en utilisant l'algorithme glouton de planification de tâches. Voici les étapes de résolution :

1. Triez les tâches par ordre décroissant de profit. Cela nous donne l'ordre de sélection des tâches.

| **Tâche** | **Délai (Deadline)** | **Profit** |
|---|---|---|
| A | 2 | 100 |
| E | 3 | 80 |
| D | 1 | 70 |
| C | 2 | 60 |
| B | 1 | 50 |

2. Créez un calendrier de planification vide avec des emplacements pour les délais possibles. Nous pouvons utiliser un tableau pour cela.

| **Délai** | **Tâche Sélectionnée** |
|---|---|
| **0** | |
| 1 | |
| 2 | |
| 3 | |

3. Commencez à sélectionner les tâches dans l'ordre décroissant de profit et placez-les dans le calendrier de manière à respecter les délais autant que possible.

- Sélectionnez la tâche A avec un profit de 100 et un délai de 2. Placez-la dans le délai 2.

| **Délai** | **Tâche Sélectionnée** |
|---|---|
| **0** | |
| 1 | |
| 2 | A |
| 3 | |

- Sélectionnez la tâche E avec un profit de 80 et un délai de 3. Placez-la dans le délai 3.

| **Délai** | **Tâche Sélectionnée** |
|---|---|
| **0** | |
| 1 | C |
| 2 | A |
| 3 | E |

- Sélectionnez la tâche E avec un profit de 70 et un délai de 1. Placez-la dans le délai 2.

| **Délai** | **Tâche Sélectionnée** |
|---|---|
| **0** | |
| 1 | D |
| 2 | A |
| 3 | E |

- Sélectionnez la tâche C avec un profit de 60 et un délai de 2. Placez-la dans le délai 0.

| Délai | Tâche Sélectionnée |
|---|---|
| **0** | **C** |
| 1 | D |
| 2 | A |
| 3 | E |

- Les tâches B restante ne peuvent pas être planifiée car elles ne respectent pas les délais.

4. Le calendrier de planification final est le suivant :

| Délai | Tâche Sélectionnée |
|---|---|
| **0** | **C** |
| 1 | D |
| 2 | A |
| 3 | E |

**Résultat :**

Les tâches planifiées dans le calendrier respectent les délais et maximisent le profit total. Le profit total est de 100 (pour A) + 70 (pour D) + 80 (pour E) +60(pour C)= 310.

Dans cet exemple, l'algorithme glouton a été utilisé pour résoudre le problème de planification de tâches et a produit un plan de séquençage qui maximise le profit tout en respectant les délais.

## Exercice 2 : remplissage de conteneurs ( bin packing problem)

Le problème de bin packing (ou problème d'emballage dans des conteneurs) est un problème d'optimisation combinatoire dans lequel vous devez emballer un ensemble d'objets de tailles différentes dans un nombre minimal de conteneurs (ou bacs) de capacité fixe.

L'objectif principal est de minimiser le nombre total de conteneurs utilisés, en maximisant l'utilisation de l'espace disponible dans chaque conteneur.

Le problème peut être formulé comme suit :

- Vous disposez d'un ensemble d'objets, chacun ayant une taille spécifique.
- Vous avez un nombre limité de conteneurs, tous de même capacité.
- Chaque objet doit être placé dans un conteneur sans dépasser sa capacité.

**L'objectif** est de trouver une répartition des objets dans les conteneurs qui minimise le nombre total de conteneurs utilisés.

Le problème de bin packing est connu pour être NP-difficile, ce qui signifie qu'il n'existe pas d'algorithme efficace pour le résoudre de manière optimale en un temps raisonnable pour un grand nombre d'objets.

**Exemple**

Supposons que vous ayez les objets suivants à emballer dans des conteneurs de capacité maximale C :
Objets : [5, 8, 4, 2, 6, 7, 5]
Capacité maximale du conteneur (C) : 10

**Questions :**

1. Donner la borne inferieure du nombre minimum des conteneurs
2. Utilisez les algorithmes First Fit, Best Fit, Worst Fit et Next Fit pour déterminer comment ces objets seront répartis dans les conteneurs.
3. Trier d'abord les objets par ordre décroissant et réappliquer l'algorithme First fit.

**Solution :**

1. La borne inferieure est égale à la somme des objets divisée par la capacité du conteneur :

$$NBmin = \frac{\sum Wi}{n} = \frac{5+8+4+2+6+7+5}{10} = 3.7 \approx 4\ conteneurs$$

Donc le nombre optimal de conteneurs est $\text{opt} \geq 4$

***2. Application des algorithmes gloutons first fit, best fit, wirst fit and next fit***

*Remarque : Dans cet exercice, nous utilisons une approche "offline" pour illustrer les algorithmes, ce qui signifie que nous connaissons tous les objets à l'avance et nous les plaçons dans les conteneurs de manière optimale.*

**First Fit :**

1. Initialisez un conteneur vide avec une capacité maximale de 10.
2. Parcourez les objets un par un dans l'ordre donné.
3. Pour chaque objet, essayez de le placer dans le premier conteneur existant où il convient. Si aucun conteneur ne convient, créez un nouveau conteneur.
4. Répétez cette étape pour chaque objet.
   : [5, 8, 4, 2, 6, 7, 5]

**Résultat du First Fit :**

- Conteneur 1 : [ 5 ,4]
- Conteneur 2 : [ 8,2]
- Conteneur 3 : [ 6]
- Conteneur 4 : [7 ]
- Conteneur 5 : [ 5]

**Nombre de conteneurs utilisés est 5**

**Best Fit :**

1. Initialisez un conteneur vide avec une capacité maximale de 10.
2. Parcourez les objets un par un dans l'ordre donné.
3. Pour chaque objet, trouvez le conteneur existant avec la capacité minimale permettant d'y placer l'objet. Si aucun conteneur ne convient, créez un nouveau conteneur.
4. Répétez cette étape pour chaque objet.

**Résultat du Best Fit :**

- Conteneur 1 : [5, 5]
- Conteneur 2 : [8,2]
- Conteneur 3 : [4,6]
- Conteneur 4 : [7]

**La meilleure solution = 4 conteneurs**

**Worst Fit :**

1. Initialisez un conteneur vide avec une capacité maximale de 10.
2. Parcourez les objets un par un dans l'ordre donné.
3. Pour chaque objet, trouvez le conteneur existant avec la capacité maximale permettant d'y placer l'objet. Si aucun conteneur ne convient, créez un nouveau conteneur.
4. Répétez cette étape pour chaque objet.

**Résultat du Worst Fit :**

- Conteneur 1 : [5, 4]
- Conteneur 2 : [8, 2]
- Conteneur 3 : [6]
- Conteneur 4 : [7]
- Conteneur 5 : [5]

**Nombre de conteneurs utilisés est 5**

Next Fit :

1. Initialisez un conteneur vide avec une capacité maximale de 10.
2. Parcourez les objets un par un dans l'ordre donné.
3. Essayez de placer chaque objet dans le conteneur actuel. Si l'objet ne convient pas, créez un nouveau conteneur et **fermer le conteneur en cours**.
4. Répétez cette étape pour chaque objet.

**Résultat du Next Fit :**

- Conteneur 1 : [ 5] x
- Conteneur 2 : [ 8] x
- Conteneur 3 : [4,2 ]x
- Conteneur 4 : [ 6]x

- Conteneur 5 : [ 7]x
- Conteneur 6: [ 5]x

**Nombre de conteneurs utilisés est 6**

3. Le tri des objets :
   [5, 8, 4, 2, 6, 7, 5]➔ [8,7,6,5,5,4,2]

**Résultat du First Fit decreasing :**

- Conteneur 1 : [ 8 ,2 ]
- Conteneur 2 : [ 7 ]
- Conteneur 3 : [ 6, 4 ]
- Conteneur 4 : [ 5, 5 ]

La solution est optimale

Nous remarquons que que les résultats peuvent varier en fonction de l'ordre dans lequel les objets sont traités, ce qui peut influencer le nombre total de conteneurs utilisés.

**Exercice 3 : Rendu de Monnaie (Change-Making Problem)**

Supposons que vous travailliez dans un magasin et que vous deviez rendre de la monnaie à un client. Vous disposez de pièces de monnaie de différentes valeurs, telles que 1 €, 2 €, 5 €, et 10 €. Le client doit recevoir un montant de 18 € en retour. Utilisez un algorithme glouton pour déterminer comment rendre la monnaie en utilisant le moins de pièces possible.

**Solution :**

Pour résoudre ce problème en utilisant un algorithme glouton, nous allons essayer de rendre la monnaie en utilisant d'abord les pièces de plus grande valeur autant que possible.

1. Nous avons les pièces de 10 €, 5 €, 2 € et 1 €.
2. Nous commençons par la pièce de 10 €, car c'est la plus grande.
3. Tant que le montant à rendre est supérieur ou égal à 10 €, nous rendons une pièce de 10 €.
4. Ensuite, nous passons à la pièce de 5 € et continuons jusqu'à ce que le montant restant soit inférieur à 5 €.
5. Ensuite, nous utilisons les pièces de 2 € de la même manière.
6. Enfin, nous utilisons les pièces de 1 € jusqu'à ce que le montant soit rendu en totalité.

**Application de l'algorithme :**

Montant à rendre : 18 €

1. Utilisons une pièce de 10 €, le montant restant est maintenant de 8 €.
2. Utilisons une pièce de 5 €, le montant restant est maintenant de 3 €.
3. Utilisons une pièce de 2 €, le montant restant est maintenant de 1 €.
4. Utilisons une pièce de 1 €, le montant restant est maintenant de 0 €.

**Résultat :**

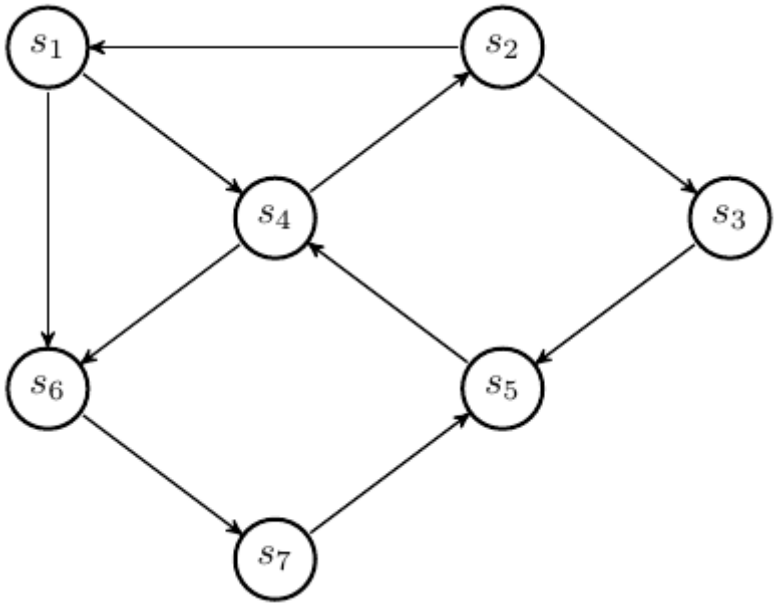
Le montant de 18 € a été rendu en utilisant un total de 4 pièces (10 €, 5 €, 2 € et 1 €) en utilisant l'algorithme glouton. Cela démontre comment cet algorithme peut être utilisé pour rendre la monnaie en minimisant le nombre de pièces utilisées.

**Exercice 4 : algorithme Greedy pour la coloration de graphe**

L'idée principale de cet algorithme est de parcourir les sommets du graphe un par un et d'assigner une couleur au sommet en choisissant la plus petite couleur disponible qui n'est pas déjà utilisée par ses voisins.
Voici les étapes de l'algorithme de coloration séquentielle :

1. Commencez par trier les sommets du graphe de manière à les parcourir dans un ordre spécifique. Cet ordre peut affecter la qualité de la coloration finale.
2. Initialisez un tableau de couleurs pour suivre la couleur assignée à chaque sommet. Au départ, tous les sommets ont une couleur non assignée.
3. Parcourez les sommets du graphe dans l'ordre spécifique.
4. Pour chaque sommet, examinez les couleurs déjà attribuées à ses voisins. Choisissez la plus petite couleur qui n'est pas déjà utilisée par ses voisins et attribuez-la au sommet en cours de traitement.
5. Répétez cette étape pour tous les sommets du graphe.
6. Une fois que tous les sommets ont reçu une couleur, le tableau de couleurs représente une coloration valide du graphe.

**Appliquer l'algorithme sur le graphe suivant :**




**Exercice 5: L'algorithme de l'insertion la moins coûteuse (Cheapest Insertion)**

L'algorithme de l'insertion la moins coûteuse (Cheapest Insertion) est une méthode gloutonne pour résoudre le problème du voyageur de commerce (TSP). L'idée de base est de commencer avec un circuit partiel contenant un seul sommet, puis d'ajouter progressivement d'autres sommets au circuit de manière à minimiser le coût total. Voici comment fonctionne l'algorithme :

1. Choisissez arbitrairement un sommet de départ et ajoutez-le au circuit partiel.
2. Tant que tous les sommets ne sont pas inclus dans le circuit partiel, trouvez le sommet non inclus (appelons-le S) qui, lorsqu'il est inséré dans le circuit partiel entre

deux sommets existants A et B, minimise le coût supplémentaire **(distance(S, A) + distance(S, B) - distance(A, B)).**

3. Insérez le sommet S entre les sommets A et B pour former un nouveau circuit partiel.
4. Répétez les étapes 2 et 3 jusqu'à ce que tous les sommets soient inclus dans le circuit.
5. Ajoutez l'arc final pour fermer le circuit.

Voici un exemple résolu pour illustrer l'algorithme Cheapest Insertion :

**Exemple :**

Considérons un ensemble de sommets avec leurs coordonnées et les distances entre eux :
Sommets : A, B, C, D, E
Coordonnées :

- A(0, 0)
- B(1, 2)
- C(2, 4)
- D(3, 1)
- E(4, 3)

**Étape 1 : Initialisation**

**Nous calculons les distances**
Distance de Manhattan = |x1 - x2| + |y1 - y2|

Voici la matrice de distances en utilisant la distance de Manhattan pour les points A, B, C, D et E :

| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 3 | 6 | 4 | 7 |
| B | 3 | 0 | 3 | 3 | 4 |
| C | 6 | 3 | 0 | 4 | 3 |
| D | 4 | 3 | 4 | 0 | 3 |
| E | 7 | 4 | 3 | 3 | 0 |

Pour appliquer l'algorithme de l'insertion la moins coûteuse (Cheapest Insertion) sur la matrice de distances fournie, en partant du point A, suivez les étapes suivantes :

**Étape 1 : Initialisation**
Nous commençons avec un circuit partiel contenant le point de départ A :
Circuit partiel : A

**Étape 2 à 4 : Ajout de Sommets**
Nous cherchons le sommet non inclus (S) qui, lorsqu'il est inséré entre deux sommets existants (A et B), minimise le coût supplémentaire.

**Itération 1 :**

- Pour A, B : Distance(S, A) + Distance(S, B) - Distance(A, B) = Distance(B, S) + Distance(S, A) - Distance(A, B)
    - Pour B, C : 3 + 6 - 3 = 6
    - Pour A, C : 6 + 3 - 6 = 3

Donc, nous insérons le sommet C entre A et B.
Circuit partiel : A - C - B

**Itération 2 :**

- Pour A, D : 4 + 3 - 4 = 3
- Pour C, D : 4 + 6 - 3 = 7
- Pour B, D : 3 + 4 - 3 = 4

Donc, nous insérons le sommet D entre A et B.
Circuit partiel : A - D - C - B
**Itération 3 :**

- Pour A, E : 7 + 3 - 7 = 3
- Pour D, E : 3 + 3 - 4 = 2
- Pour C, E : 3 + 4 - 3 = 4
- Pour B, E : 4 + 3 - 3 = 4

Donc, nous insérons le sommet E entre D et A.
Circuit partiel : A - E - D - C - B
**Étape 5 : Fermeture du Circuit**
Nous ajoutons l'arc final pour fermer le circuit en retournant au point de départ, A.
Circuit final : A - E - D - C - B - A
**Résultat :**
Le circuit final trouvé par l'algorithme Cheapest Insertion, en partant du point A, est le suivant :
Circuit final : **A - E - D - C - B - A**
Ce circuit visite tous les points une fois et revient au point de départ, minimisant ainsi la distance totale parcourue.

**Exercice 6 Le problème du "Shortest Superstring"**
Le problème du "Shortest Superstring" consiste à trouver la plus courte superchaîne qui contient toutes les chaînes données d'un ensemble donné. Voici un exemple résolu pour illustrer ce problème :
Le problème du "Shortest Superstring" est un problème d'optimisation combinatoire dans lequel l'objectif est de trouver la plus courte superchaîne qui contient toutes les chaînes données en entrée. Cela peut être utile dans des domaines tels que la bioinformatique, où l'on cherche à assembler des fragments d'ADN.
Supposons que nous ayons les chaînes d'ADN suivantes : "ACGT", "GTAC", "TACG", "CGTA". Nous devons trouver la plus courte superchaîne qui les contient toutes en utilisant l'alphabet des nucléotides ACGT.
Voici un exemple résolu :
**Étape 1 : Initialisation**

- Nous commençons avec toutes les chaînes non fusionnées.
- Notre superchaîne actuelle est vide.

**Étape 2 : Fusion des Chaînes**
Nous devons choisir les deux chaînes qui se chevauchent le plus pour les fusionner. Pour faciliter la tache créer un graphe de chevauchements ou utiliser une matrice de chevauchements. La taille de chevauchement et la taille du grand suffixe ou préfixe identique entre les deux chaines.

- "ACGT" et "GTAC" se chevauchent sur "GTACGT" (en utilisant les 4 derniers caractères de la première et les 4 premiers caractères de la deuxième).
- Nous fusionnons donc ces deux chaînes en "ACGTACGT".

Superchaîne actuelle : "ACGTACGT" Chaînes non fusionnées : ["TACG", "CGTA", "ACGTACGT"]

**Étape 3 : Répétition de la Fusion**

Nous répétons l'étape précédente jusqu'à ce que toutes les chaînes soient fusionnées.

- "ACGTACGT" et "TACG" se chevauchent sur "TACGTACGT" (en utilisant les 4 derniers caractères de la première et les 4 premiers caractères de la deuxième).
- Nous fusionnons donc ces deux chaînes en "ACGTACGTACG".

Superchaîne actuelle : "ACGTACGTACG" Chaînes non fusionnées : ["CGTA", "ACGTACGTACG"]

- "ACGTACGTACG" et "CGTA" se chevauchent sur "ACGTACGTACGTA" (en utilisant les 3 derniers caractères de la première et les 3 premiers caractères de la deuxième).
- Nous fusionnons donc ces deux chaînes en "ACGTACGTACGTA".

Superchaîne actuelle : "ACGTACGTACGTA" Chaînes non fusionnées : []

**Étape 4 : Superchaîne Terminée**

Toutes les chaînes ont été fusionnées, et notre superchaîne est maintenant complète.
Superchaîne finale : "ACGTACGTACGTA"

**Résultat :**

La plus courte superchaîne qui contient toutes les chaînes données est "ACGTACGTACGTA". C'est la réponse au problème du Shortest Superstring pour cet ensemble de chaînes d'ADN.

Veuillez noter que le problème du Shortest Superstring est NP-difficile, ce qui signifie qu'il peut être difficile de le résoudre de manière optimale pour des ensembles de chaînes plus importants. Des algorithmes heuristiques sont souvent utilisés pour obtenir des solutions acceptables dans des délais raisonnables.