**Rapport**

**1. Introduction**

Le problème F2∣∣Cmax consiste à ordonnancer n tâches sur deux machines M1 et M2 de manière à minimiser le **temps de fin d'exécution total**. Le temps d'exécution de chaque tâche est fixé pour M1 et M2, et l'ordre d'exécution doit être le même sur les deux machines (système flow shop).

L'algorithme de Johnson est une méthode efficace et optimale pour résoudre ce problème. Ce rapport décrit le fonctionnement du code Python qui implémente cet algorithme.

**2. Description du code**

Le code est structuré en plusieurs étapes, chacune correspondant à une étape clé de l'algorithme de Johnson.

**3. Explications du code**

**3.1 Fonction johnson\_algorithm**

**Entrée** :

* Une liste de tuples tasks où chaque tuple (P1,P2) représente le temps d'exécution d'une tâche sur M1 (machine 1) et M2 (machine 2).

**Sortie** :

* Une séquence optimale des tâches.
* Le Cmax, qui est le temps total pour compléter toutes les tâches.

**3.2 Étapes de l'algorithme**

1. **Séparation des tâches** :  
   Les tâches sont réparties dans deux ensembles :
   * **S1** : Contient les tâches où le temps sur M1 (P1) est inférieur ou égal au temps sur M2 (P2).
   * **S2** : Contient les tâches où P1>P2.
2. **Tri des ensembles** :
   * S1 est trié par P1 en ordre croissant.
   * S2 est trié par P2 en ordre décroissant.
3. **Construction de la séquence optimale** :  
   Les tâches de S1 sont placées en premier, suivies des tâches de S2.
4. **Calcul du Cmax** :  
   On simule l'exécution des tâches sur M1 et M2, en prenant en compte les dépendances entre les deux machines :
   * M1 exécute les tâches séquentiellement.
   * M2 ne peut commencer une tâche qu'après la fin de la même tâche sur M1.

**3.3 Exemple avec les données**

**Données d'entrée** :

tasks=[(3,6),(2,8),(7,4),(5,3)]

**Étapes de calcul** :

1. **Classification des tâches** :
   * S1=[T2, T1] car P1≤P2.
   * S2=[T3, T4] car P1>P2.
2. **Tri des tâches** :
   * S1 trié par P1 croissant : [T2, T1].
   * S2 trié par P2 décroissant : [T3, T4].
3. **Séquence optimale** :  
   Séquence=[T2, T1, T3, T4].
4. **Calcul du Cmax** :  
   En simulant les exécutions sur M1 et M2 :
   * Temps sur M1 : [2, 5, 12, 17].
   * Temps sur M2 : [10, 15, 19, 23].
   * Cmax=23.

**Résultat** :  
Séquence optimale : [2,1,3,4].  
Cmax : 23.

**4. Analyse des performances**

L'algorithme de Johnson est très efficace pour F2∣∣Cmax grâce à sa complexité temporelle de O(n log n), dominée par l'étape de tri.

**Avantages** :

* Optimal pour deux machines.
* Simple et rapide à mettre en œuvre.

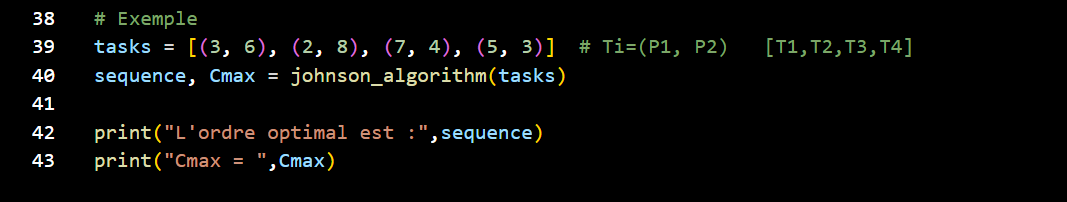
**Limites** :

* Ne s'applique qu'aux problèmes avec exactement deux machines.
* Pour plus de deux machines, il faut recourir à d'autres approches (heuristiques ou branch-and-bound).

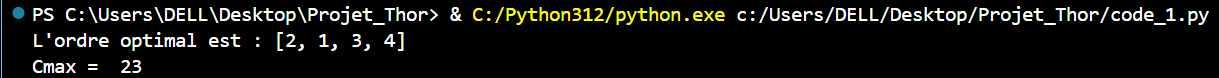
**5. Utilisation**

Le code est conçu pour être réutilisable et testé avec n'importe quelle liste de tâches. Il suffit de fournir une liste de tuples représentant les temps d'exécution pour chaque tâche.

**Exemple d'exécution :**



**Sortie** :



**6. Conclusion**

Le code présenté met en œuvre l'algorithme de Johnson pour résoudre le problème F2∣∣Cmax. Il fournit une solution optimale en termes d'ordre des tâches et du temps total d'exécution (Cmax). Grâce à sa simplicité et son efficacité, il est bien adapté pour des contextes académiques ou pratiques impliquant deux machines.