



Epreuve d'informatique

1^{er} semestre AU : 2023 – 2024

Date : Janvier 2024

Durée : 2H

Nombre de page : 4

Resource-Constrained Project Scheduling Problem

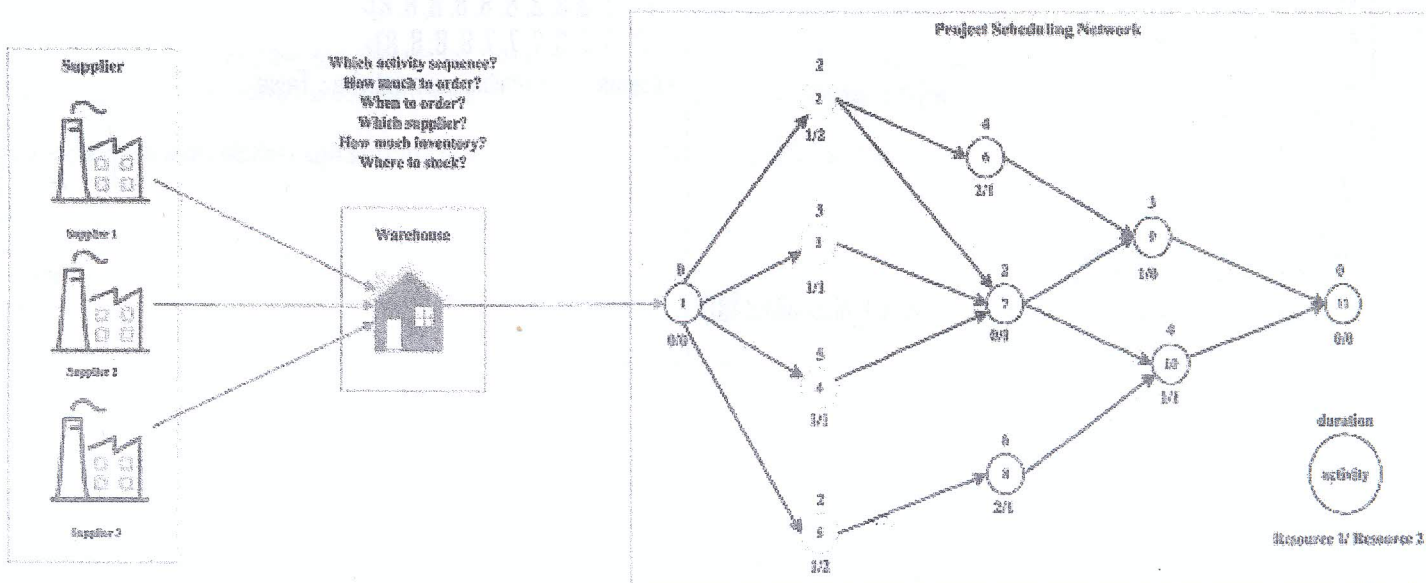
Le problème d'ordonnancement de projet à moyens limités (RCPSP : Resource-Constrained Project Scheduling Problem), est l'un des problèmes d'ordonnancement les plus connus, du fait de l'intérêt que lui ont accordé les chercheurs du domaine de la recherche opérationnelle et de ses nombreuses applications industrielles.

Dans ce sujet nous nous intéressons à mettre en place ce projet RCPSP.

Formellement, le problème d'ordonnancement de projet sous contraintes de ressources (RCPSP) est un problème d'optimisation combinatoire défini par un 6-uplet (V, p, E, R, B, b) , où V est un ensemble d'activités, p est un vecteur de durées d'exécution, E est un ensemble de relations de précédences, R est un ensemble de ressources, B est un vecteur de capacités (disponibilités des ressources), et b est une matrice de demandes (consommations de ressource).

Soit n le nombre d'activités et m le nombre de ressources disponibles. Les activités constituant le projet sont identifiées de 1 à n . Le but est d'ordonnancer les n activités sur les m ressources tout en respectant deux contraintes :

- La contrainte de précédence : une activité est exécutée si et seulement si toutes ses activités précédentes sont exécutées et terminées.
- La contrainte de ressource : une activité est exécutée si et seulement s'il y a une disponibilité de ressource de l'instant t (le début d'exécution de l'activité) jusqu'à $t + \text{durée de l'exécution de l'activité}$.



Travail demandé :

On considère la classe *Ressource* définie comme suit :

```
class Ressource :
    def __init__(self, label, duree = 120, cap = 10) :

        self.label = label           # identifiant de la ressource de type str
        self.capacity = cap          # capacité maximum de la ressource de type int
        self.disponible = [cap]*duree # une liste de disponibilité de ressources tout au long de la durée de consommation
```

et on considère la classe *Activite* définie comme suit :

```
class Activite :
    def __init__(self, identifiant, duree, ressource, consommation, debut) :
        self.id = identifiant        # identifiant de l'activité de type int
        self.duree = duree           # la durée d'exécution de l'activité de type int
        self.debut = debut           # le temps du début d'activité de type int
        self.ressource = ressource   # la ressource sur laquelle l'activité s'exécute de type Ressource
        self.consommation = consommation # la consommation de l'activité alloué pour sa ressource
```

Dans la classe *Ressource* :

Q1. Ajouter la méthode *__repr__* qui renvoie une chaîne de la forme <label – disponible>

Q2. Ajouter la méthode *est_disponible* qui prend en paramètre un objet *Activite* et qui renvoie True s'il y a une disponibilité dans la ressource à partir du début de l'activité donnée en paramètre et tout au long de sa durée d'exécution, False sinon.

Q3. Ajouter la méthode *metAJour* qui prend en paramètre un objet *Activite* et qui modifie l'attribut *disponible* en diminuant la consommation de l'activité pendant son exécution.

Exemple :

Exécution :	Trace d'exécution
<pre>RD = Ressource('RD', 10, 8) A1 = Activite(1, 4, RD, 5, 0) A2 = Activite(2, 3, RD, 1, 3) print(RD) RD.metAJour(A1) print(RD) RD.metAJour(A2) print(RD) A3 = Activite(3, 5, RD, 4, 2) print("ressource disponible pour activite : ", RD.est_disponible(A3))</pre>	<pre><RD-[8, 8, 8, 8, 8, 8, 8, 8, 8, 8]> <RD-[3, 3, 3, 3, 8, 8, 8, 8, 8, 8]> <RD-[3, 3, 3, 2, 7, 7, 8, 8, 8, 8]> ressource disponible pour activite : False</pre>

Dans la classe *Activite* :

Q4. Ajouter la méthode `__repr__` qui renvoie une chaine de la forme :

```
<activite id commence à debut, dure duree et consomme consommation de la ressource label>
```

Q5. Ajouter la méthode `consommeRessource` qui met à jour sa ressource avec sa consommation. Cette méthode déclenche une exception si la ressource n’est pas encore disponible.

Exemple :

Exécution :	Trace d'exécution
<pre>RI = Ressource('R1' , 10 , 6) A4 = Activite(4 , 3 , RI , 5 , 2) print(A4) A4.consommeRessource() print(A4.ressource)</pre>	<pre>activite 4 commence à 2, dure 3 et consomme 5 de la ressource R1> <R1-[6, 6, 1, 1, 1, 6, 6, 6, 6, 6]></pre>

Soit la classe *ActiviteOrdonnee* qui hérite de la classe *Activite* et ayant en plus les attributs suivants :

- **Lsucc** : une liste d’objets *ActiviteOrdonnee* représentant les successeurs de l’activité
- **Spred** : un ensemble d’objets *ActiviteOrdonnee* représentant les prédécesseurs de l’activité initialement vide
- **Etat** : un booléen représentant l’état de l’activité initialement False (devient True si l’activité est exécutée)

Q6. Créer la classe *ActiviteOrdonnee* avec un constructeur prenant en paramètres l’identifiant, la durée, la ressource, la consommation et la liste de ses activités successeurs.(l’attribut début est initialement égale à 0).

Q7. Dans la classe *ActiviteOrdonnee*, ajouter la méthode `estPrete` qui renvoie True si toutes les activités prédécesseurs sont exécutés (leurs états égale à True).

On considère la classe *File* définissant une structure de données de type FIFO (First In First Out)

```
class File :
    def __init__(self) :
        self.data = [ ]
```

Dans la classe *File*

Q8. Ajouter la méthode `enfiler` qui ajoute un **objet V** (donnée en paramètre) à la fin de la liste data. Dans le cas où l’objet **V** existe déjà dans data, cette méthode le supprime et l’ajoute à la fin de la liste.

Q9. Ajouter la méthode `défiler` qui supprime et renvoie le premier objet de la liste data.

Exemple :

Exécution :	Trace d'exécution
<pre>FI = File() FI.enfiler(4) FI.enfiler(2) FI.enfiler(6) FI.enfiler(3) print(FI.data) FI.enfiler(2) print(FI.data)</pre>	<pre>[4, 2, 6, 3] [4, 6, 3, 2]</pre>

On considère la classe **Ordonnancement** définie comme suit :

```
class Ordonnancement :
    def __init__(self , L_act):
        self.L_activite = L_act      # liste d'objets ActiviteOrdonnee
        self.dict_ressource = dict() # un dictionnaire des ressources
```

Dans la classe Ordonnancement

Q10. Ajouter la méthode **chargerRessource** qui récupère les ressources des activités de l'attribut **self.L_activite** et remplit l'attribut **dict_ressource** tel que les clés sont les label des ressources et les valeurs sont leurs listes des disponibilités.

Q11. Ajouter la méthode **chargerSetPredecesseur** qui remplit l'ensemble des prédécesseur (attribut **Spred**) pour chaque activité de l'attribut **self.L_activite**.

Q12. Ajouter la méthode **RCPSP** qui suit l'algorithme suivant :

Algorithme RCPSP

```
Créer une liste SO initialement contenant la première activité
Met à jour l'état de la première activité à True
Créer un objet file
Enfiler les successeurs de la première activité
Pour i de 1 à N-1 faire ( N étant le nombre d'activités)
    Répéter
        Défiler une activité nommée act
        Si act n'est pas encore prête (contrainte de précédence) alors
            Enfiler act de nouveau et répéter
        Sinon
            Initialiser le début de act en tant que la fin du dernier prédécesseur exécuté
            Parcourir la liste des disponibilités de la ressource de act pour calculer le début de act
            Consommer la ressource de act
            Modifier l'état de act
            Ajouter act a SO
            Enfiler les successeurs de act
            Passer à l'ordonnancement de l'activité suivante
Fin Pour
Renvoie SO
```


Annexe

<u>Méthode</u>	<u>Rôle</u>
<code>ch.isupper()</code>	Retourner le booléen True si la chaîne ch est <u>toute</u> en majuscule, False , sinon.
<code>ch.islower()</code>	Retourner le booléen True si la chaîne ch est <u>toute</u> en minuscule, False , sinon.
<code>ch.isalpha()</code>	Retourner le booléen True si la chaîne ch est formée par des lettres alphabétiques <u>uniquement</u> , False , sinon.
<code>ch.isnumeric()</code>	Retourner le booléen True si la chaîne ch est formée par des caractères numériques <u>uniquement</u> , False , sinon.
<code>ch.isalnum()</code>	Retourner le booléen True si la chaîne ch est formée par des lettres alphabétiques ou des caractères numériques <u>uniquement</u> , False , sinon.
<code>ch.lower()</code>	Afficher la conversion de la chaîne ch <u>toute</u> en minuscule.
<code>ch.upper()</code>	Afficher la conversion de la chaîne ch <u>toute</u> en majuscule.
<code>ch.split()</code>	Retourner la liste formée par les sous-chaînes se trouvant dans la chaîne ch et étant séparées par le caractère espace ' '. "Hello world !".split() retourne ['Hello', 'world', '!']
<code>ch.find(ch1)</code>	Retourner l'indice de la 1 ^{ère} occurrence de la chaîne ch1 dans la chaîne ch (et -1 si ch1 n'apparaît pas dans ch)
<code>L.append(x)</code>	Ajouter l'élément x à la fin de la liste L .
<code>L.pop(i)</code>	Retourner et supprimer l'élément se trouvant à la position i dans la liste L .
<code>L.index(x)</code>	Retourner l'indice de la première occurrence de l'élément x dans la liste L s'il existe, un message d'erreur, sinon.
<code>L.remove(x)</code>	Supprimer la 1 ^{ère} occurrence de l'élément x dans la liste L s'il existe, sinon, retourner un message d'erreur.
<code>D.keys()</code>	Retourner une liste contenant les clés du dictionnaire D .
<code>D.values()</code>	Retourner une liste contenant les valeurs du dictionnaire D .
<code>D.items()</code>	Retourner une liste contenant les tuples (clé, valeur) du dictionnaire D .