TP – Projet: ordonnancement de tâches

À rendre pour le **Lundi 06 Décembre 2021 à 08h00 (heure de Paris)**
Aucun délai supplémentaire ne sera accordé.

— Consignes du projet —

- Le projet est à réaliser en groupe : **monôme ou binôme**. Des codes et/ou rapports identiques entre des groupes ne seront pas acceptés.
- Le code Python à rendre doit :
  1. s'exécuter sans erreur dans l'environment de travail utilisé pour les TPs. Tout code ne s'exécutant pas sera considéré comme faux ;
  2. être lisible et compréhensible par un humain, *i.e.* les variables et fonctions doivent avoir des noms en accord avec ce qu'elles représentent.
- Le projet sera à rendre par e-mail. L'e-mail de rendu devra contenir :
  - en destinataire : **votre enseignant de TP et l'ensemble des membres du groupe** ;
  - en objet : **[RO][Project] Prenom1 Nom1 - Prenom2 Nom2** ;
  - en pièce jointe : une archive au format **zip** nommée :

    project_Prenom1_NOM1_Prenom2_NOM2.zip

    Cette archive doit contenir tous les éléments du projet : données, code Python et rapport au format **pdf**. Le rapport devra contenir les réponses de toutes les questions ne nécessitant pas la réalisation de code (sauf mention du contraire).
- Les **noms et signatures des fonctions** demandée à compléter ne devrons **pas être modifiés**. Vous êtes cependant libre de créer autant de fonctions auxiliaires que vous le souhaitez.

Les projets dont le rendu ne respecteront pas ces consignes ne seront pas considérés comme rendu.

Énoncé du problème

Une entreprise de construction mécanique décide d'augmenter sa capacité de production. Il faut construire et équiper un nouvel atelier et embaucher du personnel. On cherche à définir un planning des opérations et estimer les dates au plus tôt de démarrage de cette nouvelle production.
Un exemple de tâches à ordonnancer est donné ci-dessous (Tableau 1).

Tâches	Libellé	Durée (semaines)	Antériorité
1	Construction de l'atelier	10	—
2	Achat et réception des machines	8	—
3	Réglage des machines	4	1, 2
4	Définition des modalités d'organisation du travail	5	—
5	Définition des types et nombres de postes	2	4, 3
6	Recrutement du personnel	4	5
7	Conception des gammes de production	8	4, 2
8	Essais et tests de production	2	3, 6, 7

Table 1: Listes des tâches, durées et contraintes.

1 Extraction et visualisation des données

Nous souhaitons extraire les fichiers de données du dossier `Project/data` et les visualiser sous forme de graphe à l'aide de `Cytoscape`.

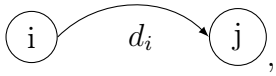
1.1 Extraction des données

Nous allons ici nous intéresser à l'extraction des fichiers de données et à leur modélisation sous forme de graphes « potentiels-tâches », dont la structure est rappelée ci-dessous (Définition 1-1).

Définition 1-1 : Graphe « potentiel-tâche »

À partir de la liste des tâches, on construit un graphe sans circuit (DAG) de la façon suivante:

- À $\forall$ tâche i, on associe un sommet 

;
- On ajoutera l'arc (i, j) de longueur d_i, noté par 

, si la tâche i de durée d_i doit précéder la tâche j ;

- On ajoute deux tâches fictives α (début) et ω (fin) ;
- α est reliée aux sommets sans prédécesseurs ;
- ω est reliée aux sommets sans successeurs ;
- On représente le graphe par niveaux après avoir calculé sa fonction rang.

Pour la suite, nous considérons que les fichiers de données suivent le format de données `CSV`. Le format `CSV` est un format de données couramment utilisé en entreprise pour représenter des bases de données.

Ici, nos fichiers suivrons le format `CSV`, avec des tabulations (`'\t'`) comme séparateurs, suivant (sans les commentaires) :

```
#Task\tLabel\tDuration\tAnteriority
1\tConstruction de l'atelier\t10\t
2\tAchat et réception des machines\t8\t
3\tRéglage des machines\t4\t1,2
...
8\tEssais et tests de production\t2\t3,6,7
```

Extrait du fichier `data/task_scheduling.data` représentant l'instance du tableau 1 au format `CSV`.

Pour plus de détails, vous pouvez regarder le fichier `data/task_scheduling.data` décrivant la liste des tâches du tableau 1.

Question 1.

> Pour un chemin d'accès `file_path` à un fichier respectant le format ci-dessus, compléter la fonction `extract_network` du fichier `project_file_to_network.py`. Cette fonction extrait les données du fichier de chemin d'accès `file_path` et retourne un graphe « potentiel-tâche ». Ce graphe doit être un graphe dirigé `NetworkX`.

Pour la suite, vous pourrez utiliser la fonction `get_networks_data` pour itérer sur les jeux de données. Cette fonction prend en entrée un booléen `all_data` indiquant si l'on itère sur une instance ou sur toutes les instances. Par défaut, `all_data` est fixée à faux. Pour rappel, cette fonction s'utilise telle que (voir TP2 et TP3 pour plus d'informations) :

```
for graph in get_networks_data(all_data):
    scheduling = solve_task_scheduling(graph)
    ...
```

1.2 Visualisation des données

Maintenant que nous pouvons extraire les données, il faut pouvoir les visualiser.

Question 2.

> Exporter le graphe « potentiel-tâche » du fichier `data/task_scheduling.data` au format `GraphML`.

Question 3.

> Visualiser le fichier `GraphML` de la question précédente à l'aide de `Cytoscape`. Après avoir correctement mis en forme le graphe, vous exporterez le visuel obtenu en tant qu'image.

Les images ainsi obtenues devront être présentées dans un rapport au format `.pdf`.

2 Modélisation linéaire du problème d'ordonnancement

Pour la suite, nous utiliserons les notations vue en TD. Ces notations sont rappelés ci-dessous (Définition 2-1).

Définition 2-1 : Définitions diverses

- *La date au plus tôt* t_i de début de la tâche i: $t_i = \max_{(j,i)\in E}(t_j + d_j)$ (c.-à-d. la longueur du *plus long chemin* de α à i $l(PLC(\alpha \to i))$).
- La *durée minimale* t_ω : $l(PLC(\alpha \to \omega))$.
- Si la durée du projet est fixé à t_ω, la *date au plus tard* T_i pour commencer la tâche i sans influencer t_ω est : $T_\omega = t_\omega$; $T_i = \min_{j\in\Gamma_i}(T_j - d_i) \Rightarrow T_i = t_\omega - l(PLC(i \to \omega))$
- La *marge totale* (mt_i) de la la tâche i : $mt_i = T_i - t_i$. Pour une durée de projet fixée à t_ω la marge totale d'une tâche correspond au retard maximale que peut prendre son début sans retarder la date t_ω.
- Si $mt_i = 0$, la tâche i est dite *tâche critique*.
- La *marge libre* (ml_i) de la tâche i : $ml_i = \min_{(i,j)\in E}(t_j - t_i - d_i)$. Pour un calendrier donné, la marge libre d'une tâche correspond au retard maximal que peut prendre son début sans modifier les dates de début des autres tâches.

Lors du TD sur l'ordonnancement de tâches, vous avez calculé les différents éléments (t_i, T_i, mt_i et ml_i) en utilisant des approches basées programmation dynamique. L'objectif de ce projet est de recalculer ces éléments en utilisant les techniques de programmation linéaire vue en cours et un solveur adéquat. Dans les deux sections suivantes, vous allez définir 2 approches de programmation linéaire différentes pour résoudre ce problème.

Toutes les réponses aux questions ne demandant pas de code (questions 4–8–10–13–16–18) devront figurer dans votre rapport.

2.1 Durée minimale

Dans cette section, nous vous proposons de développer deux approches permettant de déterminer les temps les plus tôt et tard des tâches à ordonnancer : `PLC` en Section 2.1.1 et `Bellman-Ford` en Section 2.1.2. Le choix de l'approche à utiliser pour la résolution du problème sera donné en argument dans le terminal.
Pour utiliser l'approche `PLC`, on utilisera :

```
(.venv)$ python3.8 project.py --method PLC
```

tandis que pour utiliser l'approche `Bellman-Ford`, on utilisera :

```
(.venv)$ python3.8 project.py --method Bellman-Ford
```

Par défaut, c'est l'approche `PLC` qui sera utilisée.

2.1.1 Recherche d'un chemin de poids maximal

La première approche que vous devez développer est basée sur la recherche d'un chemin élémentaire entre 2 sommets i et j dans un graphe orienté comme vu lors du TP4.

Question 4.

Nous souhaitons déterminer le chemin le plus long entre deux sommets i et j du graphe. Pour cela, nous souhaitons utiliser la programmation linéaire en suivant l'approche vue en cours (TP4).

1. Décriver les variables linéaires nécessaires pour résoudre ce problème.
2. Quelle est la fonction objectif du modèle linéaire ?
3. Donner les contraintes permettant de définir un chemin élémentaire entre deux sommets i et j.

Question 5.

Compléter la fonction `plc` du fichier `project.py`. Cette fonction implémente et résout le modèle linéaire définie à la question précédente telle que :

Entrées : un graphe dirigé `NetworkX`, un sommet source i et un sommet cible j.

Sortie : la valeur de la fonction objectif du programme linéaire.

De la longueur des plus long chemins entre deux sommets, il est possible de déterminer les temps les plus tôt t_i et les plus tards T_i pour effectuer une tâche i. Pour rappel, nous savons que pour un graphe « potentiel-tâche » $G = (V, E)$:

$$\forall i \in V, t_i = plc(G, \alpha, i)$$
$$\forall i \in V, T_i = t_\omega - plc(G, i, \omega)$$

Question 6.

Compléter la fonction `plc_compute_earliest_time` du fichier `project.py`. Cette fonction est définie telle que :

Entrées : un graphe dirigé `NetworkX`, un sommet i.

Sortie : la date au plus tôt t_i pour effectuer la tâche i.

Question 7.

Compléter la fonction `plc_compute_latest_time` du fichier `project.py`. Cette fonction est définie telle que :

Entrées : un graphe dirigé `NetworkX`, la durée minimale t_ω, un sommet i.

Sortie : la date au plus tard T_i pour effectuer la tâche i sans influencer la durée du projet t_ω.

Pour la suite, nous nommerons cette approche : `PLC`.

2.1.2 L'invariant de Bellman-Ford

La seconde approche que vous allez développer est basée sur la propriété de Bellman-Ford. L'idée est de définir un chemin de poids maximal en utilisant la propriété de Bellman-Ford :

$$\forall (u, v) \in E,\ t_u \leq t_v - d(u)$$

où $d(u)$ représente la durée de la tâche u.

Question 8.

Calculons tout d'abord les temps les plus tôt t_i pour effectuer chaque tâche i.

1. Quelles sont les variables linéaires du modèle ? Détailler leur spécification.
2. Quelle contrainte linéaire doit-on mettre sur t_α ? ***Indice :*** α doit s'effectuer avant la première tâche.
3. Quelle est la fonction objectif du modèle ? ***Indice :*** On souhaite que t_i soit le plus proche possible de t_α.
4. Donner le modèle linéaire complet permettant de calculer t_i en utilisant la propriété de Bellman-Ford.

Question 9.

Compléter la fonction `bf_compute_earliest_time` du fichier `project.py`. Cette fonction implémente et résout le modèle linéaire défini à la question précédente, elle est définie telle que :

Entrées : un graphe dirigé `NetworkX`, un sommet i.

Sortie : la date au plus tôt t_i pour effectuer la tâche i.

Question 10.

Calculons ensuite les temps les plus tard T_i pour effectuer chaque tâche i. Nous supposerons connue la valeur t_ω, représentant le temps au plus tôt pour effectuer la tâche ω.

1. Quelles sont les variables linéaires du modèle ? Détailler leur spécification.
2. Quelle contrainte linéaire doit-on mettre sur T_ω ? ***Indice :*** La tâche ω est une tâche critique.
3. Quelle est la fonction objectif du modèle ? ***Indice :*** On souhaite que T_i soit le plus proche possible de T_ω.
4. Donner le modèle linéaire complet permettant de calculer T_i en utilisant la propriété de Bellman-Ford.

Question 11.

Compléter la fonction `bf_compute_latest_time` du fichier `project.py`. Cette fonction implémente et résout le modèle linéaire défini à la question précédente, elle est définie telle que :

Entrées : un graphe dirigé `NetworkX`, la durée minimale t_ω, un sommet i.

Sortie : la date au plus tard T_i pour effectuer la tâche i sans influencer la durée du projet t_ω.

Pour la suite, nous nommerons cette approche : `Bellman-Ford`.

2.2 Ordonnancement de tâches

Afin de réaliser un ordonnancement des tâches, il est nécessaire de déterminer les dates les plus tôt et les plus tard pour chaque tâche i. Pour cela, vous réaliserez des fonctions permettant de calculer ces dates pour une tâche i fixée.

Question 12.

Compléter la fonction `solve_task_scheduling` du fichier `project.py`. Cette fonction est définie telle que :

Entrées : un graphe dirigé `NetworkX` décrivant un graphe « potentiel-tâche » et un nom d'approche (`PLC` ou `Bellman-Ford`).

Sortie : un dictionnaire associant à chaque tâche i le couple (t_i, T_i) des dates les plus tôt et tard de i.

Attention, vous veillerez à bien calculer les valeurs t_i et T_i avec la méthode fournie en paramètre :

PLC : `plc_compute_earliest_time` et `plc_compute_latest_time` ;

Bellman-Ford : `bf_compute_earliest_time` et `bf_compute_latest_time`.

Pour la suite, nous appellerons *ordonnancement* le dictionnaire associant à chaque tâche i le couple (t_i, T_i) des dates les plus tôt et tard de i.

Une fois l'ordonnancement réalisé et le calendrier au plus tôt adopté, l'entreprise souhaite classifier les tâches selon 3 catégories :

1. tâches pour lesquelles le moindre retard de démarrage allonge le délai de réalisation du projet.
2. tâches pour lesquelles le moindre retard de démarrage perturbe le calendrier ;
3. tâches pour lesquelles un léger retard ne modifie pas le planning.

Question 13.

Selon quels critères peut-on classifier une tâche dans ces trois catégories ?

Nous souhaitons avoir accès à toutes les informations permettant de classifier nos tâches. Pour cela, nous voulons afficher les résultats de l'ordonnancement sous la forme d'un tableau suivant le format :

Tâche i	t_i	T_i	mt_i	ml_i
1	?	?	?	?
...				
n	?	?	?	?

Table 2: Format du tableau demandé.

Question 14.

Compléter les fonctions `compute_slack` et `compute_free_slack` du fichier `project.py`. Ces fonctions sont définies telles que :

Entrées : un graphe « potentiel-tâche », un ordonnancement et un sommet i.

Sorties : réciproquement, la marge total mt_i et la marge libre ml_i du sommet i.

Question 15.

Compléter la fonction `solve_task_scheduling_extended`. Cette fonction est définie telle que :

Entrées : un graphe « potentiel-tâche » et un ordonnancement.

Sortie : un dictionnaire associant chaque tâche i à un tuple de la forme (t_i, T_i, mt_i, ml_i).

Question 16.

Afficher dans le terminal de commande la sortie de la fonction `solve_task_scheduling_extended` sous la forme d'un tableau (voir Tableau 2). Vous donnerez le tableau résultant dans le rapport.

L'entreprise souhaite finalement présenter l'ordonnancement proposé à ses investisseurs. Elle décide donc de leur présenter une visualisation du graphe « potentiel-tâche » contenant le plus d'informations possibles.

Question 17.

Compléter la fonction `visualize_task_scheduling_results` définie telle que :

Entrées : un chemin d'accès `file_path`, un graphe « potentiel-tâche », l'ordonnancement étendu avec les marges totales et libres'.

Sortie : crée un fichier de chemin `file_path` au format `GraphML` contenant toutes les informations nécessaires pour faire une visualisation respectant les critères de l'entreprise.

Question 18.

Visualiser le fichier `GraphML` de la question précédente à l'aide de `Cytoscape`. Après avoir correctement mis en forme le graphe, vous exporterez le visuel obtenu en tant qu'image.

3 Analyses des approches

Dans cette section, nous allons étudier les deux modèles linéaires que vous avez défini dans la Section 2.1. ***Toutes les réponses aux questions de cette section devront figurer dans votre rapport.***

3.1 Analyse théorique des modèles linéaires

Soit le graphe « potentiel-tâche » $G = (V, E)$ avec V l'ensemble des sommets et E l'ensemble des arcs.

Question 19.

> Donner le nombre de contraintes et de variables du modèle linéaire de l'approche `PLC` en fonction de V et E.
> Quelle est la complexité en terme de nombre de contraintes et de variables du calcul de l'ordonnancement en utilisant cette approche ?

Question 20.

> Donner le nombre de contraintes et de variables des modèles linéaires de l'approche `Bellman-Ford` en fonction de V et E.
> Quelle est la complexité en terme de nombre de contraintes et de variables du calcul de l'ordonnancement en utilisant cette approche ?

Question 21.

> Comparer les résultats de l'analyse théorique de chaque approche. Doit-on favoriser une approche vis-à-vis d'une autre ? Si oui, laquelle et dans quelle circonstance ?

3.2 Résultats en pratique

Il y a souvent une différence entre les analyses théoriques et ce qui est observé en pratique. Ces différences peuvent venir de nombreux paramètres, parmi lesquels : la structure initiale des données, l'instanciation du modèle linéaire ou encore des heuristiques du solveur linéaire.

Voyons donc comment se comporte vos implémentations des deux approches en pratique. Vous pourrez exécuter votre projet sur l'ensemble des instances proposées en l'exécutant avec la commande suivante :

```
(.venv)$ python3.8 project.py --all-data [options]
```

Question 22.

> Analyser l'évolution du nombre de contraintes et de variables du modèle linéaire de l'approche `PLC` en pratique sur les instances fournies.

Question 23.

> Analyser l'évolution du nombre de contraintes et de variables du modèle linéaire de l'approche `Bellman-Ford` en pratique sur les instances fournies.

Question 24.

Comparer les résultats de l'analyse pratique de chaque approche. On attend de vous que vous vous intéressiez aux temps de calculs, nombres de variables et contraintes en fonction de la taille des instances et de l'approche utilisée.
Concluer sur l'avantage d'une approche vis-à-vis d'une autre.