Emploi du temps

Objectif : L'objectif de ce tp est :

- d'utiliser un algorithme glouton pour résoudre un problème de gestion de salles
- d'écrire un algorithme de tri sur plusieurs colonnes

1 Lycée

Un lycée miniature possède plusieurs salles de classes et différents créneaux de cours. La Fig 1 donne une représentation graphique des créneaux.

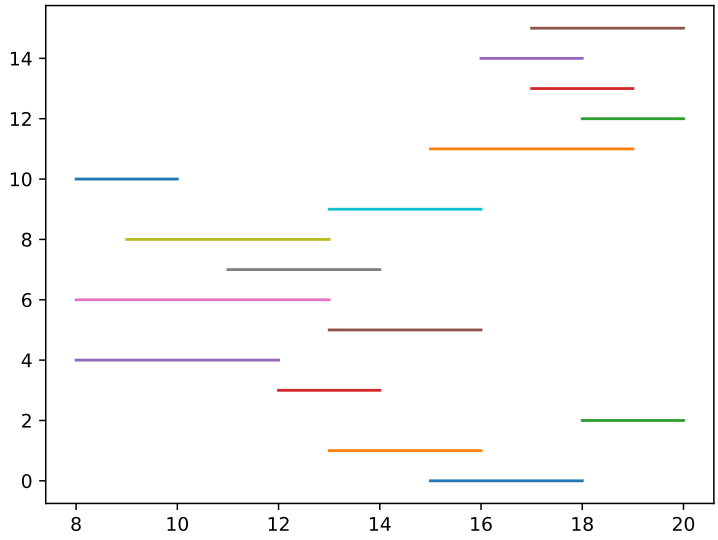

Figure 1 – Créneaux des cours

Le fichier `.py` mis à disposition permet de charger les créneaux de cours sous la forme d'une liste `cours` de 2-uplets dont le premier élément correspond à l'heure de début de cours et le deuxième à la l'heure de fin de cours.

```
cours = [(15, 17), (13, 15), (18, 19), (12, 13), (8, 11), (13, 15),
         (8, 12), (11, 13), (9, 12), (13, 15), (8, 9), (15, 18),
         (18, 19), (17, 18), (16, 17), (17, 19)]
```

A l'aide d'algorithmes gloutons, on souhaite trouver une solution aux revendications suivantes :

- un inspecteur souhaite suivre le plus de cours différents dans la journée
- l'équipe de ménage souhaite que les EDT utilisent le moins de salles possible.

Il s'agit de deux questions différentes à ne pas traiter en même temps (et donc pas forcément compatibles).

Q - 1 : *Obtenir la Fig 1.*

2 Un peu de tri

La liste `cours` n'est pas triée. Il existe plusieurs façon de le faire :

- par heure de début de cours
- par durée des cours
- par heure de fin de cours

Il se trouve aussi que plusieurs cours ont une même heure de début et une durée différente.

Dans cette partie du Tp, la méthode `lexsort` du module `np` sera utilisée.

```
>>> L_1 = [2, 3, 5, 3, 2, 0]
>>> L_2 = [1, 3, 0, 2, 1, 3]
>>> L_3 = [4, 2, 1, 3, 1, 2]
>>> print(np.lexsort((L_1,)))
[5 0 4 1 3 2]
>>> print(np.lexsort((L_2, L_1)))
[5 0 4 3 1 2]
>>> print(np.lexsort((L_1, L_2)))
[2 0 4 3 5 1]
>>> print(np.lexsort((L_3, L_1, L_2)))
[2 4 0 3 5 1]
```

Q - 2 : *Après avoir testé `lexsort` avec les tableaux ci-contre, déterminer comment fonctionne cette méthode.*

Q - 3 : *Écrire une fonction `tri_cours(L, choix)` qui prend en argument deux listes `L` et `choix`, telles que `L` contient la liste des créneaux de cours comme définie dans la partie précédente et `choix` contient deux entiers (0 pour l'heure de début de cours, 1 pour la durée du cours et 2 pour l'heure de fin du cours). La fonction `tri_cours(L, choix)`, renvoie une liste `M`, triée en premier sur le critère `choix[0]` et en deuxième sur le critère `choix[1]`.*

Pour cela, on pourra :

- créer trois listes `deb`, `dur` et `fin` idoines,
- créer une liste contenant les trois listes précédentes,
- récupérer les indices après avoir appliqué la méthode `lexsort`,
- créer une liste vide `M` et la remplir correctement.

3 Revendication de l'inspecteur

Q - 4 : *Écrire une fonction `journee_ig(L)` qui prend en argument la liste `L` de cours triée suivant un choix prédéfini et qui renvoie la liste des cours, sous la forme des 2-uplets de `L`, auxquels peut assister l'IG.*

4 Revendication de l'équipe de ménage

Q - 5 : *Écrire une fonction `remplissage_salles(L)` qui prend en argument la liste `L` de cours triée suivant un choix prédéfini et qui renvoie une liste `salles` de listes où la ième liste contient, pour la salle `i`, les cours sous forme des 2-uplets de `L`.*

5 Tris

Q - 6 : *Écrire une fonction `tris(data, choix)` qui prend en arguments une liste de listes `data` et un liste d'entiers relatifs `choix` et qui renvoie `data` trié sur la ligne `choix[0]`, et en cas d'égalité sur la ligne `choix[1]`. Si `choix[i]` est négatif, le tri est décroisssant.*