TP15

ALGORITHMES GLOUTONS

Planning d'un séminaire

Vous assistez à un séminaire présentant un certains nombre de conférences qui vous intéressent. Vous souhaitez voir **le maximum de conférences**. Il s'agit donc, en fonction des horaires des conférences, d'établir votre planning de la journée.

Exemple 1 : on a quatre conférences C_1, C_2, C_3 et C_4 et leurs créneaux sont indiquées sous la forme d'un intervalle.

$$C_1 : [3,4[, \quad C_2 : [0,1[, \quad C_3 : [2,3[, \quad C_4 : [1,2[.$$

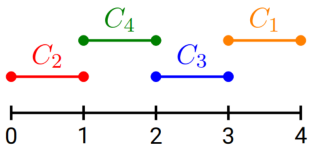

FIGURE 1 – Situation 1

Ici c'est très simple, vous allez pouvoir assister à toutes les conférences dans l'ordre suivant : C_2, C_4, C_3, C_1.

Exemple 2 : $C_1 : [2,4[, \quad C_2 : [0,1[, \quad C_3 : [1,3[, \quad C_4 : [0,2[.$

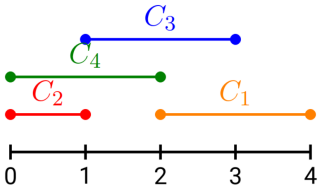

FIGURE 2 – Situation 2

Ici, plusieurs conférences se superposent, vous n'allez pas pouvoir toutes les voir. Il y a plusieurs solutions :

$$C_2 \text{ puis } C_1 \,; \quad C_2 \text{ puis } C_3 \,; \quad C_4 \text{ puis } C_1 \,; \quad C_3.$$

Se posent alors plusieurs questions :

- Comment choisir une de ces solutions ?
  Idée 1 : on classe les conférences par horaire de début croissante, on voit la première puis on regarde quelle conférence on peut mettre à la suite. Ceci élimine C_3 et $[C_2, C_1]$.

- Problème : un algorithme doit renvoyer une réponse unique et bien déterminée. Or ici, il nous reste deux solutions : $[C_2, C_3]$ et $[C_4, C_1]$.

  Remarque : toutes les conférences vous intéressent autant, quelles que soient leurs durées. La solution $[C_4, C_1]$ n'est pas à privilégier de ce point de vue. On cherche un algorithme permettant de voir **le maximum de conférences** (en nombre, pas en durée).

- Comment choisir la première conférence entre C_2 et C_4 ?
  Idée 2 : on complète en regardant les horaires de fin : $fin_2 < fin_4$. Laquelle choisissez-vous ?

Exemple 3 : $C_1 : [0, 3[, \quad C_2 : [1, 2[, \quad C_3 : [2, 3[$.

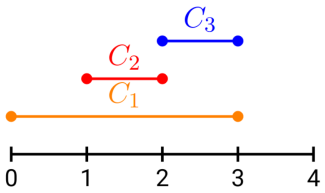

FIGURE 3 – Situation 3

Appliquons les principes précédents :

- On commence par C_1. Il n'y a pas de conférence possible après. Cette solution n'est pas satisfaisante car on préfère voir C_2 et C_3.
- Idée 3 : on procède uniquement par heure de fin croissante.

$$fin_2 \leqslant fin_1 \leqslant fin_3.$$

On commence donc par C_2 puis on aura C_3 (C_1 est éliminée car incompatible avec C_2). C'est cette idée que nous allons programmer. On peut montrer qu'elle fournit une solution optimale.

Algorithme :

- Classer les conférences par heure de fin croissante.
- Choisir la première conférence de la liste, l'ajouter au planning.
- S'intéresser à la deuxième conférence de la liste. Si elle est compatible on l'ajoute au planning. Sinon, on passe à la conférence suivante.
- Recommencer jusqu'à avoir passé en revue toutes les conférences.

1. Appliquer l'algorithme à la main sur cet exemple :

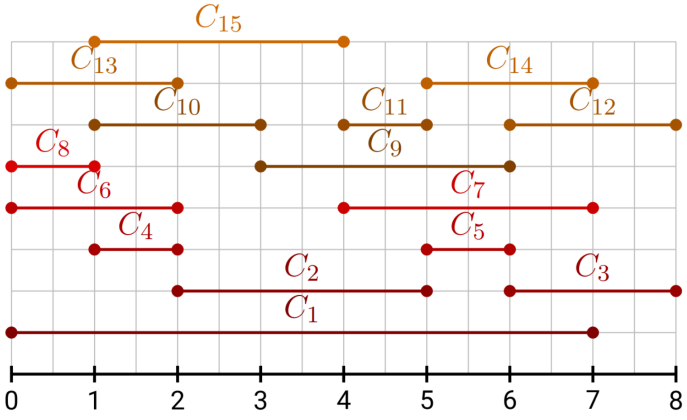

FIGURE 4 – Une situation plus complexe.

2. Compléter le programme suivant :

```
conferences = [[0,7,'C1'], [2,5,'C2'], [6,8,'C3'], [1,2,'C4'],
[5,6,'C5'], [0,2,'C6'], [4,7,'C7'], [0,1,'C8'], [3,6,'C9'],
[1,3,'C10'],[4,5,'C11'], [6,8,'C12'], [0,2,'C13'], [5,7,'C14'],
[1,4,'C15']]

# Trier les conférences par date de fin croissante.
conferences_triees = sorted(conferences, key = lambda L:L[1] )
print(conferences_triees)

n = ..................... # nombre de conférences

planning = []

i = ... # indice de la première conférence considérée

planning.append(.............................)

for j in range(...........):

    fin = ..........................

    debut_j = ..........................

    if ........................... :

        ............................

        ............................



print(planning)
```