TP4 : Programmation dynamique

La plus longue sous-séquence commune à deux séquences

Nous avons deux séquences (c'est-à-dire une série ordonnée) d'objets `a` et `b` (chaînes de caractères, listes d'entiers, ...), le but est de trouver une séquence `c` de longueur maximale telle que `c` soit une sous-séquence de `a` et `b`. Les caractères de la séquence `c` sont donc communs aux caractères de la séquence `a` et aux caractères de la séquence `b`, apparaissent dans le même ordre que dans les séquences `a` et `b` mais pas nécessairement de manière continue.

Par exemple, en prenant `a="ngjgdgkdlkjfdkkkk"` et `b="glfkgklfngksdgmlerp"`, `c="gggk"` est une sous-séquence commune `a` et `b`. Notre objectif est de rechercher la sous-séquence commune à `a` et `b` de longueur maximale.

Notons `long_a` la longueur de la séquence `a`, `long_b` la longueur de la séquence `b`. Nous introduisons une fonction $S : (i,j) \in [\![0, \texttt{long\_a}]\!] \times [\![0, \texttt{long\_b}]\!] \to S(i,j)$ telle que pour tout couple (i,j) appartenant à $[\![0, \texttt{long\_a}]\!] \times [\![0, \texttt{long\_b}]\!]$, $S(i,j)$ contienne la plus longue sous-séquence commune à la sous-séquence $a_{0\dots i-1}$ et $b_{0\dots j-1}$.

1. (a) Déterminer $S(i,0)$ pour tout i appartenant à $[\![0, \texttt{long\_a}]\!]$.
   (b) Déterminer $S(0,j)$ pour tout j appartenant à $[\![0, \texttt{long\_b}]\!]$.
   (c) Soit (i,j) appartenant à $[\![1, \texttt{long\_a}]\!] \times [\![1, \texttt{long\_b}]\!]$.
   Déterminer une expression de $S(i,j)$ en fonction de $S(k,l)$, avec (k,l) appartenant à $\mathbb{N}^2$ tel que $(k,l) < (i,j)$. L'expression proposée doit être expliquée.
   (d) Expliquer où se trouve le résultat cherché.
2. Écrire en OCaml une fonction utilisant le paradigme de programmation dynamique

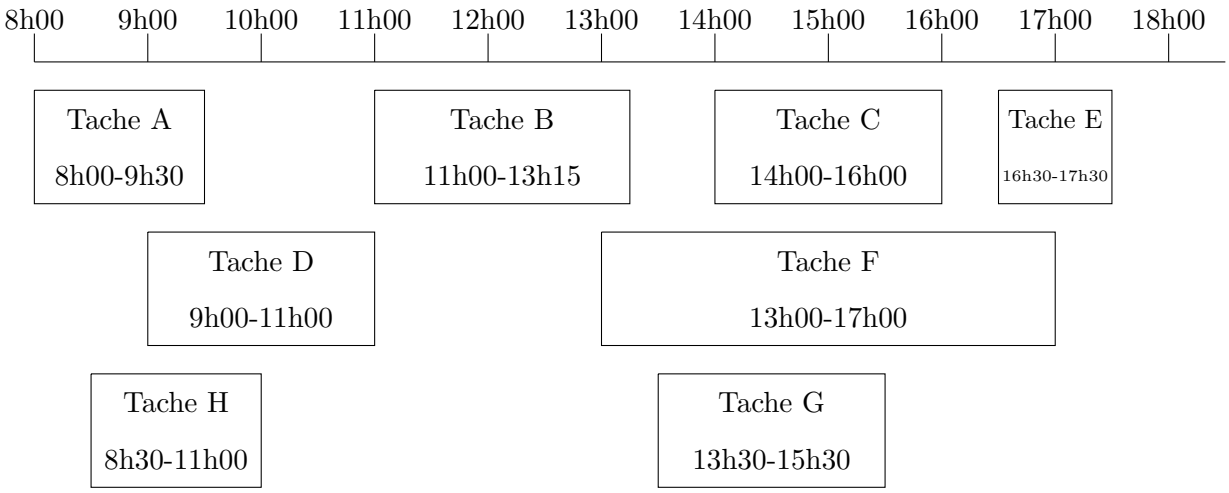
   `plus_longue_sc a b`

   qui, étant donnés deux mots `a` et `b`, détermine la plus longue séquence commune aux mots `a` et `b`.

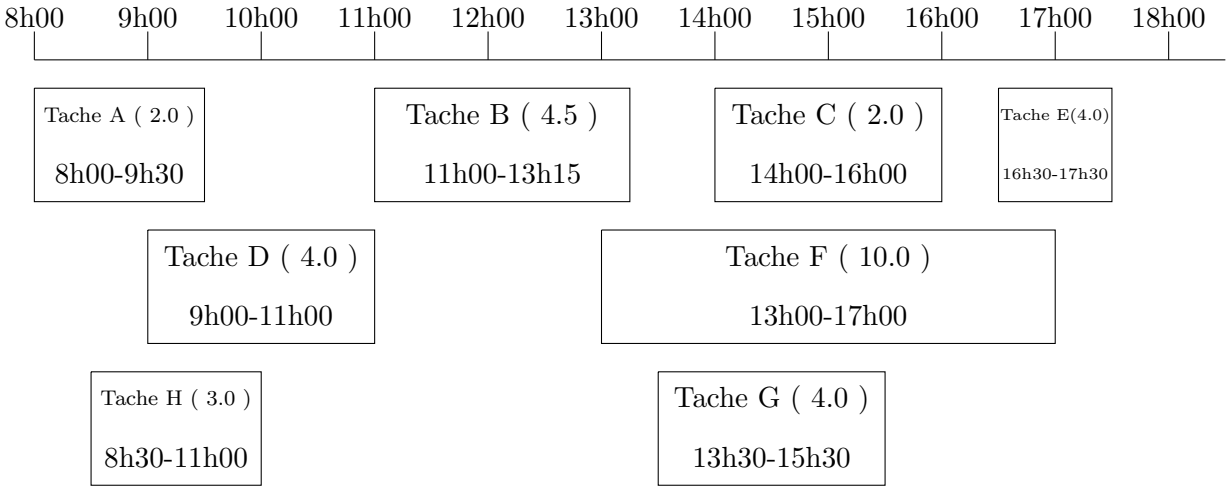
Le weighted interval scheduling

Nous allons nous intéresser maintenant au problème d'ordonnancement de taches pondérées. Nous appellerons **planning**, un ensemble de taches pouvant chacune s'exécuter dans un intervalle de temps déterminé (intervalles qui peuvent se recouvrir les uns les autres). Le problème d'**ordonnancement des taches pondérées** est le suivant : chaque tache ayant chacune une importance donnée, trouver un sous-ensemble de taches compatibles, c'est-à-dire dont les intervalles de temps ne s'intersectent pas, qui maximise la somme des importances.

Par exemple, le planning d'occupation d'une salle de laboratoire est le suivant :



Mais il nous est impossible de se tenir à cet emploi du temps car certains événements se chevauchent, nous allons devoir faire des choix. Il s'avère que tous les événements de l'emploi du temps précédent ne sont pas aussi importants les uns que les autres. Nous pouvons étiqueter chaque événement par son importance comme suit :



Nous remarquons que, sur le planning précédent, nous pouvons obtenir un emploi du temps d'importance totale 12.5 en choisissant les taches A, B, C, E. Mais nous pouvons aussi obtenir une importance totale 13 en choisissant les taches H, F.

L'objectif est de trouver l'emploi du temps d'importance totale maximale.

Nous décidons d'implémenter un planning par une liste de taches, chaque tache étant un triplet `(debut,fin,importance)` où `debut` est l'instant de début de la tache, `fin` son instant de fin et `importance` son importance, ces trois valeurs seront des flottants.

1. **Algorithme glouton.**
   Nous appelons **qualité d'une tache**, le rapport `importance/(fin-debut)`. Ainsi, la qualité d'une tache est proportionnelle a son importance et inversement proportionnelle à sa durée.

Nous proposons de résoudre le problème de façon approchée à l'aide d'un algorithme glouton. L'idée est de sélectionner systématiquement la tache ayant une qualité maximale, puis de recommencer tant que c'est possible sur les taches compatibles restantes.

(a) Implémenter cet algorithme en OCaml.

(b) Que donne cet algorithme sur l'exemple ? Est-ce effectivement la solution optimale ?

(c) Quelle est la complexité de l'algorithme ?

2. **Algorithme naïf**.
Nous proposons de résoudre le problème de façon exacte à l'aide d'un algorithme récursif naïf.

(a) Proposer un tel algorithme et l'implémenter en OCaml.

(b) Quelle est la complexité de l'algorithme ?

3. **Programmation dynamique.**
Nous proposons de résoudre le problème de façon exacte à l'aide d'un algorithme de programmation dynamique.

Nous décidons d'implémenter un planning par un vecteur de taches, chaque tache étant un triplet `(debut,fin,importance)` où `debut` est l'instant de début de la tache, `fin` son instant de fin et `importance` son importance, ces trois valeurs seront des flottants.

Nous supposons les taches du planning $\texttt{t}_0, \ldots, \texttt{t}_{\texttt{n-1}}$, `n` appartenant à $\mathbb{N}^*$, triées par ordre croissant d'instant de fin des taches. Nous pourrons noter, par commodité, pour tout entier naturel `i` compris entre `0` et `n-1`, $\texttt{d}_\texttt{i}$ le début de la tache $\texttt{t}_\texttt{i}$, $\texttt{f}_\texttt{i}$ la fin de la tache $\texttt{t}_\texttt{i}$ et $\texttt{w}_\texttt{i}$ l'importance de la tache $\texttt{t}_\texttt{i}$.

(a) Écrire une fonction OCaml qui prend comme argument le tableau trié des taches et donne comme résultat un tableau `p` tel que :

$$\forall \texttt{j} \in [\![1, \texttt{n-1}]\!] \text{, } \texttt{p.(j) = max\{i } \in [\![0, \texttt{j-1}]\!] \texttt{ , } \texttt{f}_\texttt{i} \leq \texttt{d}_\texttt{j}\}.$$

Autrement dit, pour tout entier naturel `j` compris entre `1` et `n-1`, `p.(j)` est le plus grand des entiers `i` strictement inférieurs à `j` tels que $\texttt{t}_\texttt{i}$ est compatible avec $\texttt{t}_\texttt{j}$.

Quelle est sa complexité ?

(b) Notons pour tout entier naturel `j` compris entre `0` et `n`, `c.(j)` l'importance maximale qu'on peut obtenir avec les seules taches $\texttt{t}_0$, ... , $\texttt{t}_{\texttt{j-1}}$. Nous cherchons donc `c.(n)` et les autres `c.(j)`, `j` entier compris entre `0` et `n-1`, sont nos sous-problèmes.

Donner `c.(0)` puis une relation de récurrence vérifiée par les `c.(j)`, `j` entier compris entre `1` et `n`.

(c) En déduire un algorithme de programmation dynamique permettant de calculer `c.(n)` et le coder en OCaml.

(d) Donner la complexité totale de la résolution du problème.

(e) Adapter l'algorithme précédent pour qu'il calcule non seulement `c.(n)`, mais aussi la (une) liste de taches compatibles qui permet d'obtenir une importance totale de `c.(n)` et le coder en OCaml.