Option Informatique

TD $n°08$: ordonnancement de tâches pondérées

Il s'agit d'un algorithme de programmation dynamique.

Le problème : On appelle planning un ensemble de tâches.

- Chaque tâche a une importance donnée.
- Chaque tâche peut s'exécuter dans un intervalle de temps déterminé (intervalles qui peuvent se recouvrir les uns les autres), déterminé par son horaire de début et son horaire de fin.
- Deux tâches sont dites compatibles si l'exécution de l'une n'empêche pas l'exécution de l'autre, c'est-à-dire si l'horaire de fin de l'une des deux est ultérieur à l'horaire de début de l'autre.

Le problème *d'ordonnancement des tâches pondérées* est le problème d'optimisation suivant : étant donné un planning, trouver un sous-ensemble de tâches compatibles qui **maximise** la somme des importances. On dira qu'un tel sous-ensemble de tâches est optimal.

Implémentation : On implémente le planning par trois tableaux de longueur n, où n est le nombre de tâches :

- Un tableau `deb` dont les éléments sont les horaires de début de chaque tâche.
- Un tableau `fin` dont les éléments sont les horaires de fin de chaque tâche.
- Un tableau `imp` dont les éléments sont les importances de chaque tâche.

On supposera enfin que les tâches sont triées par ordre croissant de leurs horaires de fin, c'est-à-dire que le tableau `fin` est trié. Bien sûr, si tel n'était pas le cas, un prétraitement permettrait de le faire en temps $O\big(n\lg(n)\big)$.

On notera que les tâches sont numérotées de 0 à $n-1$. Ainsi, la tâche 0 est celle qui se termine en premier alors que la tâche $n-1$ est celle qui se termine le plus tard, sans préjuger de leurs horaires de début.

Exercice 1. *Algorithme naïf de l'enfer*

1. Décrire en français l'algorithme le plus naïf possible permettant de résoudre ce problème. **Ne pas l'implémenter !**
2. Déterminer sa complexité.

Exercice 2. *Algorithme glouton*

On propose de résoudre le problème de façon approchée à l'aide d'un algorithme glouton. On appelle **qualité** d'une tâche le rapport $\mathit{importance} \div (\mathit{fin} - \mathit{debut})$. Ainsi, la qualité d'une tâche est proportionnelle a son importance et inversement proportionnelle à sa durée. L'idée de l'algorithme glouton est de sélectionner systématiquement la tâche ayant une qualité maximale, puis de recommencer tant que c'est possible sur les tâches compatibles restantes.

3. Quelle est la complexité de l'algorithme ?
4. Implémenter cet algorithme en Caml. Votre fonction devra rendre une liste de tâches compatibles, *i. e.* une liste d'entiers $i_1 < \ldots < i_r$ tels qu'on ait toujours `fin.(`i_k`)<deb.(`i_{k+1}`)`. Cette liste de tâches ne sera pas nécessairement optimale mais correspondra à ce que l'on obtient à l'aide de l'algorithme glouton.

Exercice 3. *Les sous-problèmes et la relation de récurrence*

On propose les sous-problèmes suivants : étant donné un entier $j \leqslant n-1$, déterminer le sous-ensemble de tâches optimal parmi les tâches 0, 1, ..., j. On notera $\mathcal{M}_j$ l'importance totale d'un tel sous-ensemble optimal, c'est-à-dire la somme des importances la plus grande qu'on puisse obtenir avec des tâches compatibles parmi les tâches 0 à j.

On cherche donc $\mathcal{M}_{n-1}$.

On note `p.(j)` le plus grand entier $i < j$ tel que les tâches i et j soient compatibles, en convenant de poser `p.(j)` $= -1$ si un tel entier n'existe pas.

5. Écrire une fonction OCaml qui prend comme argument les tableaux `deb`, `fin` et `imp` (on rappelle que `fin` est trié) et retourne comme résultat le tableau p défini ci-dessus. Quelle et sa complexité ?
6. Déterminer $\mathcal{M}_{-1}$ et la relation de récurrence vérifiée par $\mathcal{M}_j$.
   Cette relation devrait faire intervenir $\mathcal{M}_{j-1}$, `imp.(j)` et $\mathcal{M}_{\texttt{p.(j)}}$.

Exercice 4. *Programmation dynamique.*

On zappe l'étape de la version récursive enveloppée, qui n'est jamais très intéressante !

7. Déduire des questions précédentes un algorithme de programmation dynamique permettant de calculer $\mathcal{M}_n$. (Le décrire en français.) Déterminer sa complexité.
8. L'implémenter.
9. L'adapter pour qu'il calcule non seulement $\mathcal{M}_n$, mais aussi une liste de tâches compatibles optimale.