TD #19 — Paradigmes & texte

Exercice 1. *Stratégie de révision*

Repris d'un exercice de Bruno Grenet

Un·e élève prévoit son programme de révision avec la stratégie suivante : chaque jour peut être un jour de travail *tranquille*, un jour de travail *soutenu*, ou un jour de *repos*, avec la contrainte qu'un jour *soutenu* doit être précédé d'un jour de *repos*. L'élève sait estimer pour chaque jour i le nombre t_i de points que permettra d'obtenir un travail *tranquille* et le nombre s_i de points d'un travail *soutenu*. Un jour de repos ne lui fait gagner aucun point.

Par exemple, sur la période suivante, une stratégie optimale est d'être en repos les jours 1 et 4, en travail tranquille le jour 3, et en travail soutenu les jours 2 et 5, ce qui rapportera 2,5 points.

Jour	1	2	3	4	5
t	0,3	0,5	0,4	0,6	0,2
s	0,5	1,2	1	0,8	0,9

1. Montrez que l'algorithme (à tendance gloutonne) suivant n'est pas optimal (avec n le nombre de jours de la période de révision).

```
i ← 1
Tant que i ≤ n :
    Si i ≤ n − 1 et s_{i+1} > t_i + t_{i+1} :
        Repos le jour i et travail soutenu le jour i + 1
        i ← i + 2
    Sinon :
        Travail tranquille le jour i
        i ← i + 1
```

2. Proposez une formule récursive pour calculer p_i le nombre maximum de points obtenu en travaillant jusqu'au jour i.
3. Écrivez en OCaml un algorithme (itératif) de programmation dynamique pour calculer p_n et analysez sa complexité en temps et en espace. On suppose que les valeurs t_i et s_i sont respectivement stockées dans un tuple `(float * float) array` comme `fst a.(i)` et `snd a.(i)`, et que les indices i commencent désormais à 0.
4. Pouvez-vous (éventuellement) réduire la coût spatial de votre algorithme précédent ?
5. Modifiez votre algorithme précédent pour qu'il renvoie maintenant une stratégie optimale, en plus de la seule quantité de points qu'elle rapporte.

Exercice 2. *Plus longue sous-suite croissante*

Étant donnée une suite finie d'entier (s_i) de longueur n, le problème de la *plus longue sous-suite croissante* consiste à en trouver une sous-suite maximale dont les termes sont tous croissants (pour un ordre non strict $\leq$).

1. Esquissez un algorithme qui résout ce problème par une recherche exhaustive. Quel est son coût en fonction de n ? (On demande une évaluation *assez précise* du coût, qui explicite les éventuelles hypothèse faites.)

Pour résoudre ce problème par programmation dynamique, on définit (les solutions à) la famille de sous-problèmes intermédiaires ℓ_k indexée par $k \in [\![n]\!]$ consistant à trouver la **longueur** de la plus longue sous-suite croissante *terminant à l'indice* k.

2. Que vaut ℓ_0 ?
3. Donnez une expression récursive (en k) de ℓ_k.
4. Pourquoi une implémentation naïve de l'expression récursive ci-dessus serait elle inefficace ?
5. Proposez un algorithme **itératif** efficace pour calculer l'ensemble des solutions ℓ_k, et **analysez son coût**.
6. Adaptez votre algorithme pour qu'il permette pour tout $k \in [\![n]\!]$ de reconstruire une plus-longue sous-suite croissante terminant en k.
7. Donnez un algorithme qui résout le problème initial.

Exercice 3. *Sous-mots*

On rappelle qu'un sous-mot u d'un mot v (sur un alphabet quelconque) est une sous-suite de v : $\forall i \in [\![i, |u|]\!]$, $u_i = v_{\varphi(i)}$ pour une certaine fonction strictement croissante $\varphi : [\![1, |v|]\!] \to [\![1, |v|]\!]$

1. Écrivez une fonction C de signature :
   ```
   bool is_subw(char *u, char *v)
   ```
   qui renvoie `true` si la chaîne de caractères `u` représente un sous-mot du mot représenté par la chaîne de caractère `v`, et `false` sinon. Cette fonction devra être de coût linéaire en la taille de ses arguments.

Exercice 4. *Deux propriétés sur les préfixes & suffixes*

Dans tout ce qui suit, on fixe un alphabet Σ arbitraire.

1. Montrez le lemme « de Levi » :
   Soit $u, v, u', v' \in \Sigma^*$ tels que $uv = u'v'$, alors il existe un unique mot t tel que :
   — $u = u't \wedge v' = tv$, ou symétriquement
   — $u' = ut \wedge v = tv'$
2. Montrez que si u, u' sont tous deux préfixes (resp. suffixes) d'un même mot v, alors u est préfixe (resp. suffixe) de u' ou u' est préfixe (resp. suffixe) de u.