TD 14 – Fourres-y tout

Exercice 1. *Diviser pour régner*

Dans cet exercice, on s'intéresse à deux problèmes de sélections d'éléments dans un tableau.

1. Étant donné un tableau trié d'entiers deux à deux distincts $A[1,\dots,n]$, on veut décider s'il existe un entier i tel que $A[i] = i$. Donner un algorithme en temps $\mathcal{O}(\log n)$ pour ce problème.
2. Soit $A[1,\dots,n]$ un tableau d'entiers triés dont on ne connaît pas la longueur (n est inconnu). On peut demander la valeur de $A[i]$ pour n'importe quelle valeur entière de i : si $i \leq n$, on reçoit la valeur de $A[i]$, sinon on reçoit ∞. Donner un algorithme en temps $\mathcal{O}(\log n)$ qui prend en entrée un entier x et trouve l'indice du tableau où se trouve x (s'il en existe un).

Exercice 2. *Programmation dynamique*

Une fanfare est composée de n musiciens de tailles $t_1, t_2, \dots, t_n$. Pour les jours de fête l'orchestre dispose de m uniformes ($m \geq n$) de tailles $u_1, u_2, \dots, u_m$. Chaque année certains musiciens s'en vont et sont remplacés par d'autres, il faut alors ré-attribuer à chaque musicien le costume qui lui sied le mieux.

René, le percussionniste, pense qu'il faut chercher à minimiser la différence moyenne entre la taille d'un musicien et celle de son costume :

$$\frac{1}{n}\sum_{i=1}^{n} |t_i - u_{\alpha(i)}|$$

où $\alpha(i)$ est l'indice du costume attribué au musicien de taille t_i. Il propose pour cela un algorithme glouton qui consiste à chercher i et j minimisant $|t_i - u_j|$. On attribue alors l'uniforme j au musicien i et on itère jusqu'à ce que tout le monde ait reçu un uniforme.

1. L'algorithme de René est-il optimal ?

Anne, la corniste, trouve plus équitable de chercher à minimiser le carré moyen des écarts :

$$\frac{1}{n}\sum_{i=1}^{n} (t_i - u_{\alpha(i)})^2$$

2. Montrez par un exemple l'avantage de cette fonction objective par rapport à la précédente. L'algorithme glouton est-il optimal pour la nouvelle fonction objective ?
3. Montrer que si il y a autant de costumes que de musiciens, l'algorithme consistant à classer les musiciens et le costumes par taille croissante et à attribuer au musicien i le costume i est optimal pour la seconde fonction objective.
4. Donner un algorithme donnant une solution optimale dans le cas $m \geq n$.

Exercice 3. *Analyse Amortie*

On considère une pile munie des opérations suivantes :
– $PUSH(S, x)$: empile un objet x sur la pile S
– $POP(S)$: dépile le sommet de la pile S et retourne l'objet dépilé
– $MULTIPOP(S, k)$: dépile au plus k objets de la pile S

Algorithm 1: $MULTIPOP(S, k)$

```
début
    tant que S ≠ ∅ et k ≠ 0 faire
        POP(S);
        k ← k − 1;
```

1. Quelle est la complexité de chacune des 3 opérations ? En déduire avec la méthode globale (méthode de l'agrégat) le coût amorti pour une suite de n opérations $PUSH$, POP et $MULTIPOP$ sur une pile initialement vide.
2. Même question avec la méthode des acomptes.
3. Même question avec la méthode des potentiels.
4. On souhaite implémenter une file à l'aide de deux piles, de telle façon que le coût amorti des opérations $ENQUEUE$ et $DEQUEUE$ soit $O(1)$. Comment peut-on faire ?

Exercice 4. NP-*complétude (cours)*

Soient P_1 et P_2 deux problèmes de décision, et supposons qu'on connaisse une transformation polynomiale (une réduction) de P_1 en P_2. Répondre aux sept questions suivantes avec un maximum de deux lignes de justification par question.

1. Si $P_1 \in \mathcal{P}$, a-t-on $P_2 \in \mathcal{P}$?
2. Si $P_2 \in \mathcal{P}$, a-t-on $P_1 \in \mathcal{P}$?
3. Si P_1 est NP-complet, P_2 est-il NP-complet ?
4. Si P_2 est NP-complet, P_1 est-il NP-complet ?
5. Si on connaît une transformation polynomiale de P_2 en P_1, P_1 et P_2 sont-ils NP-complets ?
6. Si P_1 et P_2 sont NP-complets, existe-t-il une transformation polynomiale de P_2 en P_1 ?
7. Si $P_1 \in \mathcal{NP}$, P_2 est-il NP-complet ?

Exercice 5. NP-*complétude (exo)*

Dans un graphe orienté, on dit que $\{x_1, \ldots, x_k\}$ est une chaîne transitive de longueur k si et seulement si pour tout $1 \leq i < j \leq k$, $(x_i, x_j) \in E$. Montrer que le problème suivant est $\mathcal{NP}$-complet.

Sous-chaîne transitive :

Instance : Un graphe orienté $D = (V, E)$.

Question : D contient-il une sous-chaîne transitive de longueur au moins $\left\lfloor \frac{|V|}{2} \right\rfloor$?

Indication : Vous pouvez par exemple effectuer une réduction à partir de 3-SAT. Si on se donne un ensemble de clauses $C_1, ..., C_k$, avec $C_i = (x_i^1 \vee x_i^2 \vee x_i^3)$, on peut construire une instance du problème de sous-chaîne transitive en posant $V = \{C_0\} \bigcup_{1 \leq i \leq k} \{C_i, x_i^1, x_i^2, x_i^3\}$ et en choisissant avec soin les arêtes à mettre dans E.

Exercice 6. *Algorithmes d'approximation*

Dans un TD précédent, on s'est intéressé au problème de décision SUBSET-SUM consistant à savoir s'il existe un sous-ensemble d'entiers de S dont la somme vaut exactement t. Le problème d'optimisation qui lui est associé prend aussi en entrée un ensemble d'entiers strictement positifs S et un entier t, il consiste à trouver un sous-ensemble de S dont la somme est la plus grande possible sans dépasser t (cette somme qui doit donc approcher le plus possible t sera appelée *somme optimale*).

On suppose que $S = \{x_1, x_2, \ldots, x_n\}$ et que les ensembles sont manipulés sous forme de listes triées par ordre croissant. Pour une liste d'entiers L et un entier x, on note $L + x$ la liste d'entiers obtenue en ajoutant x à chaque entier de L. Pour les listes L et L', on note $\textbf{Fusion}(L, L')$ la liste contenant l'union des éléments des deux listes.

Premier algorithme

Algorithm 2: Somme(S, t)

début
  $n \leftarrow |S|$;
  $L_0 \leftarrow \{0\}$;
  pour i **de** 1 **à** n **faire**
    $L_i \leftarrow$ **Fusion**$(L_{i-1}, L_{i-1} + x_i)$;
    Supprimer de L_i tout élément supérieur à t;
  retourner *le plus grand élément de* L_n;

1. Quelle est la distance entre la valeur retournée par cet algorithme et la somme optimale ?
2. Quelle est la complexité de cet algorithme dans le cas général ? Et si pour un entier $k \geq 1$ fixé, on ne considère que les entrées telles que $t = \mathcal{O}(|S|^k)$, quelle est la complexité de cet algorithme ?

Deuxième algorithme Cet algorithme prend en entrée un paramètre ϵ en plus, où ϵ est un réel vérifiant $0 < \epsilon < 1$.

Algorithm 3: Somme-avec-seuillage(S, t, ϵ)

début
  $n \leftarrow |S|$;
  $L_0 \leftarrow \{0\}$;
  pour i **de** 1 **à** n **faire**
    $L_i \leftarrow$ **Fusion**$(L_{i-1}, L_{i-1} + x_i)$;
    $L_i \leftarrow$ **Seuiller**$(L_i, \epsilon/2n)$;
    Supprimer de L_i tout élément supérieur à t;
  retourner *le plus grand élément de* L_n;

L'opération **Seuiller** décrite ci-dessous réduit une liste $L =< y_0, y_1, \ldots, y_m >$ (supposée triée par ordre croissant) avec le seuil δ :

Algorithm 4: Seuiller(L, δ)

début
  $m \leftarrow |L|$;
  $L' \leftarrow < y_0 >$;
  $dernier \leftarrow y_0$;
  pour i **de** 1 **à** m **faire**
    si $y_i > (1 + \delta) dernier$ **alors**
      Insérer y_i à la fin de L';
      $dernier \leftarrow y_i$;
  retourner L';

3. Évaluer le nombre d'éléments dans L_i à la fin de la boucle. En déduire la complexité totale de l'algorithme. Pour donner la qualité de l'approximation fournie par cet algorithme, borner le ratio valeur retournée/somme optimale.