MPCI

Programmation 2 — Examen Terminal

Mercredi 24 février 2021

On rappelle qu'aucun document ni équipement électronique n'est autorisé. L'usage d'une gopro hero 9 à des fins de diffusion en direct sur twitch est néanmoins tolérée.

1 programmation objet

1.1 listes python

1. Explicitez les différences entre les les 2 instructions suivantes, si `l` est une liste :
   - `l = l + [1]`
   - `l.append(1)`
2. Quel instruction est-il préférable d'utiliser ?
3. Que rend la méthode de liste `append` ?

1.2 namespaces

Expliquez comment fonctionne l'instruction `x = 1` en python.

1.3 modélisation

1.3.1 classe `UndoRedo`

Donnez le code python d'une classe `UndoRedo` dont l'initialisation prend deux paramètres : une liste `l` et un objet `x`. Cette classe doit contenir deux méthodes :

- `supprime()` qui ne prend **pas** de paramètre et supprime le dernier élément de `l` s'il vaut `x`
- `remet()` qui ne prend **pas** de paramètre et place `x` comme dernier élément de la liste.

Vous pourrez utiliser le fait que si `l` est une liste en python :

- `l.pop()` supprime le dernier élément de l et le rend.
- `l[-1]` rend le dernier élément de la liste `l`.

1.3.2 utilisation de `UndoRedo`

En sachant que `randint(1, 10)` rend un entier aléatoire entre 1 et 10 détaillez ce que fait le code suivant :

```
from random import randint

l = []
for x in range(10):
  l.append(randint(1, 10))
print(l)

sauve = []
while len(l) > 0:
  undo_redo = UndoRedo(l, l[-1])
  undo_redo.supprime()
  sauve.append(undo_redo)

print(l)

while len(sauve) > 0:
  undo_redo = sauve.pop()
  undo_redo.remet()

print(l)
```

Ci-après, un résultat possible du code précédent :

```
[10, 3, 3, 1, 8, 1, 10, 3, 4, 4]
[]
[10, 3, 3, 1, 8, 1, 10, 3, 4, 4]
```

2 Algorithme glouton

On suppose que pour mener à bien un projet, on doit réaliser n tâches où chaque tâche t_i a :

- une durée de réalisation : d_i,
- un temps de fin conseillé : f_i.

Si on note s_i le début de la réalisation de la tâche t_i on définit son retard par :

$$r_i = \max(0, s_i + d_i - f_i)$$

Si $r_i > 0$ la tâche t_i est en retard. Le but du problème est de trouver un algorithme glouton qui affecte à chaque tache son début et qui minimise le retard maximum :

$$R = \max_{1 \leq i \leq n} r_i$$

Comme on a qu'un seul ouvrier pour réaliser les tâches, on ne peut créer qu'une tâche à la fois.

2.1 Première propriété

1. Montrez que le retard d'une solution ne peut pas augmenter si l'on supprime les temps d'inactivité de celle-ci : l'ouvrier enchaîne les tâches sans s'arrêter jusqu'à ce que toutes les tâches aient été réalisées.

2. En déduire que la solution de notre problème est un ordonnancement des tâches selon un ordre particulier : c'est un algorithme glouton sans étape de choix (toutes les tâches sont dans la solution).

2.2 Algorithme

On suppose que les tâches $(t_i)_{1 \leq i \leq n}$ sont rangées dans un certain ordre. Écrivez l'algorithme qui calcule le retard maximum pour cet ordre. Quel est sa complexité ?

2.3 mauvais ordres

Montrez que les ordres suivants ne sont pas optimaux :

1. Les tâches triées par durée décroissante.

2. Les tâches triées par durée croissante.

2.4 ordre optimal

1. Montrez que si une solution possède deux tâches successives t_i et t_{i+1} telles que $f_i > f_{i+1}$, les échanger n'augmente pas le retard.

2. Montrez que si une solution ne possède aucunes tâches successives t_i et t_{i+1} telles que $f_i > f_{i+1}$, alors les tâches sont rangées par temps de fin conseillé croissante.

3. En déduire que l'ordre optimal est réalisé pour l'ordre des temps de fin conseillé croissante.

3 Tris

```
algorithme selection(tab):
  de i allant de 0 à la longueur de tab - 2:
      min_index = i
      de j allant de i+1 à la la longueur de tab - 1:
          si tab[j] < tab[min_index] alors : min_index = j
      échanger tab[i] et tab[min_index]
```

1. Décrivez le fonctionnement de l'algorithme ci-dessus, nommé *tri par sélection*.
2. Après avoir donné les définitions des complexités maximale, minimale et en moyenne vous en donnerez les valeurs pour le tri par sélection.
3. Qu'est-ce que la *complexité du tri* ? Donnez sa valeur (inutile d'en donner la preuve).

4 Le $k$ème plus petit élément d'une liste

Pour une liste l de n entiers, son $k$ème plus petit élément x est tel que l'on peut séparer les $n-1$ autres éléments de l en deux ensembles : l'un de $k-1$ éléments plus petit ou égal à x et l'autre de $n-k$ éléments plus grand ou égal à x.

4.1 sélection modifié

1. Modifiez l'algorithme du tri par sélection (partie 3) pour qu'il rende le $k$ème plus petit élément d'une liste l où k et l sont les paramètres d'entrée de l'algorithme.
2. Quel est la complexité de l'algorithme ?
3. En déduire la complexité de cet algorithme lorsque l'on utilise celui-ci pour la recherche de la médiane d'une liste (le $\frac{len(l)}{2}$ème plus petit élément d'une liste l).

4.2 avec un tri

1. Donnez l'algorithme ainsi que sa complexité de la recherche du $k$ème plus petit élément d'une liste d'entiers triée par ordre croissant.
2. Si l'on doit au préalable trier la liste quelle est la complexité totale de l'algorithme ?
3. Est-ce mieux pour trouver la médiane (le $\frac{len(l)}{2}$ème plus petit élément de l) d'une liste ?

4.3 avec un pivot

4.3.1 algorithme du pivot

Écrivez un algorithme nommé `pivot(l)` à partir d'une liste l de n éléments, rend deux listes $l1$ et $l2$ telles que :

- $l1 = \{l[i] \mid i \geq 1, l[i] \leq l[0]\}$
- $l2 = \{l[i] \mid i \geq 1, l[i] > l[0]\}$

Vous veillerez à ce que sa complexité soit en $\mathcal{O}(n)$.

4.3.2 algorithme récursif

1. Utilisez l'algorithme `pivot(l)` précédent pour écrire un algorithme récursif permettant de trouver le $k$ème plus petit élément d'une liste d'entier. La récursion se fera en rappelant l'algorithme sur $l1$ ou $l2$ selon les cas.
2. Donnez en sa complexité minimale et maximale pour une liste l et un entier k donné.

4.3.3 équation de récurrence de la complexité

1. Montrez que la complexité $C(n, k)$ de l'algorithme récursif de recherche du $k$ème élément d'une liste l à n éléments peut s'écrire :

$$C(n, k) = \mathcal{O}(n) + C(n', k')$$

   Où vous expliciterez n' et k' en fonction de n, de k et du pivot $l[0]$ que l'on supposera être le $p$ème plus petit élément de l.

2. Si les entiers de l sont tous différents et dans un ordre aléatoire, quelle est la probabilité que $l[0]$ soit le $p$ème plus petit élément de l ?
3. En déduire la taille *moyenne* de n'.

4.3.4 complexité en moyenne

On cherche la complexité en moyenne de l'algorithme récursif. On va calculer cette complexité en ne prenant pas en compte k (ce qui est possible puisque $k \leq n$) : pour une taille de liste donnée n, cette complexité vaut alors : $\mathcal{C}(n)$. L'équation de récurrence de la complexité en moyenne s'écrit alors $\mathcal{C}(n) = \mathcal{O}(n) + \mathcal{C}(n')$ où n' est la taille moyenne de la liste lors de la récursion (que vous avez déterminée à la question précédente). En remplaçant (sans perte de généralité) $\mathcal{O}(n)$ par n, cela revient à résoudre l'équation de récurrence suivante (remplacez n' par sa valeur) :

$$\mathcal{C}(n) = n + \mathcal{C}(n')$$

En remarquant que $0 < \sum_{i=1}^{m} \frac{1}{2^i} \leq \sum_{i=1}^{\infty} \frac{1}{2^i} = 2$ (quelque soit m positif), montrez que :

1. $\mathcal{C}(n) = \mathcal{O}(n)$.

2. En déduire qu'il existe un algorithme de calcul de la médiane d'une liste de n éléments en $\mathcal{O}(n)$ opérations en moyenne.

4.3.5 pour ne pas conclure

On peut montrer — mais ce sera pour une autre histoire — qu'en choisissant judicieusement le pivot on peut modifier l'algorithme récursif pour qu'il ait une complexité maximale de $\mathcal{O}(n)$ opérations.
En déduire :

1. La complexité du problème de la recherche du $k$ème plus petit élément d'une liste.

2. La complexité du problème de la recherche de la médiane d'une liste.