Programmation et Algorithmique I

Test coté

20 novembre 2009

Consignes

La durée du test est de 4 heures (strictes) et il comporte 4 problèmes, ce qui représente approximativement 1 heure par problème. Pour chaque problème une spécification et des consignes de résolution sont données. Ensuite, une série de questions sont posées.

Veuillez respecter scrupuleusement les consignes suivantes :
- répondez à chaque problème **sur des feuilles séparées** et utilisez uniquement les feuilles de réponses qui vous sont données ;
- indiquez votre nom, prénom et section sur chaque feuille de réponses ;
- écrivez lisiblement ;
- les notes de cours et personnelles **ne sont pas autorisées** ;
- gérez votre temps, ne vous laissez pas impressionner par la longueur des énoncés : leur compréhension et leur résolution dans les temps fait partie du test ;
- lisez (et relisez) attentivement l'énoncé d'un problème entièrement avant de commencer à répondre aux questions ;
- si ce n'est pas explicitement demandé dans l'énoncé, **vous ne devez pas tester que les préconditions sont respectées**.

Problème 1 : affichage récursif

Pour cette question, il vous est demandé d'implémenter des fonctions récursives. Vous **ne pouvez pas** utiliser de boucle. Vous **ne pouvez pas** utiliser l'opérateur * sur les chaînes de caractères. Pour la résolution d'une fonction, vous pouvez utiliser les autres fonctions déjà écrites.

Questions

Donnez le code Python des fonctions suivantes :

a) Une fonction *récursive* `multiply(s, nb)` qui prend en entrée `s` une chaîne de caractères et `nb` un nombre entier positif (précondition). Son rôle est de retourner une chaîne de caractères qui est la concaténation de `nb` fois la chaîne `s`.

Par exemple, `multiply('a', 5)` retourne la chaîne `aaaaa`.

b) Une fonction *récursive* `print_from_middle(s, nb = 1)` qui prend en entrée `s` une chaîne de caractères de taille impaire et `nb` un nombre entier positif impair (1 par défaut). Ce sont des préconditions. Son rôle est d'afficher à l'écran sur une ligne les `nb` caractères au milieu de la chaîne `s`, sur une autre ligne, les (`nb`+2) caractères au milieu de la chaîne `s`, sur la ligne suivante les (`nb`+4) caractères au milieu de la chaîne `s`, et ainsi de suite, jusqu'à afficher la chaîne `s` entière.

Par exemple, `print_from_middle('recursivite')` affiche ceci à l'écran :

```
s
rsi
ursiv
cursivi
ecursivit
recursivite
```

c) Une fonction *récursive* `print_pyramid(s, nb = 1)` qui prend en entrée `s` une chaîne de caractères de taille impaire et `nb` un nombre entier positif impair (1 par défaut). Ce sont des préconditions. Son rôle est d'afficher une sorte de pyramide avec les caractères de la chaîne. Sur la première ligne, elle affiche les `nb` caractères au milieu de la chaîne `s` avec, de part et d'autre, un certain nombre de fois le caractère '-'. Sur la deuxième ligne, elle affiche les (`nb`+2) caractères au milieu de la chaîne `s` avec, de part et d'autre, un certain nombre de fois le caractère '-'. Sur la ligne suivante, elle affiche les (`nb`+4) caractères au milieu de la chaîne `s` avec, de part et d'autre, un certain nombre de fois le caractère '-', et ainsi de suite, jusqu'à afficher la chaîne `s` entière (sans aucun caractère '-'). Le nombre de caractères '-' à afficher par ligne doit être choisi afin de permettre de garder un affichage centré (voir exemple).

Par exemple, `print_pyramid('recursivite')` affiche ceci à l'écran :

```
-----s-----
----rsi----
---ursiv---
--cursivi--
-ecursivit-
recursivite
```

Problème 2 : approximation de l'intégrale d'une fonction

Soient :
- $X \subseteq \mathbb{R}$;
- $f : X \to \mathbb{R}^+$ une fonction continue et apériodique ; et
- $a, b \in X$ avec $a < b$.

Il vous est demandé d'implémenter un algorithme qui permet d'approximer $\int_a^b f(x)\,\mathrm{d}x$ par la méthode des rectangles inférieurs. Le principe est de partager l'intervalle $[a, b]$ en n intervalles ($n \in \mathbb{N}_0$) de même largeur Δn : $[x_0, x_1]; [x_1, x_2]; \ldots; [x_{n-1}, x_n]$ avec $x_0 = a$ et $x_n = b$ (voir Figure 1).

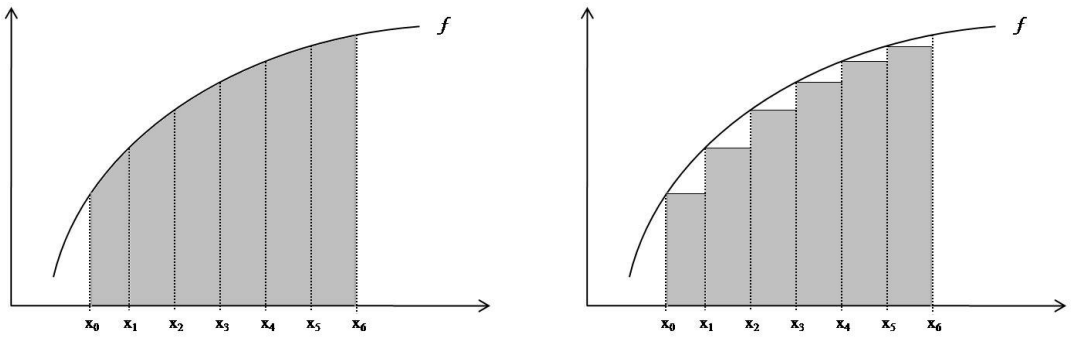

FIGURE 1 – Exemple d'intervalles (avec $n = 6$) et représentation des rectangles inférieurs

On a que :

$$\int_a^b f(x)\,\mathrm{d}x = \int_{x_0}^{x_1} f(x)\,\mathrm{d}x + \int_{x_1}^{x_2} f(x)\,\mathrm{d}x + \cdots + \int_{x_{n-1}}^{x_n} f(x)\,\mathrm{d}x.$$

La méthode des rectangles inférieurs consiste à approximer l'intégrale sur un intervalle $[x_i, x_{i+1}]$ par l'intégrale d'une fonction constante $f(x_i)$, et de calculer la somme de ces approximations :

$$\int_{x_0}^{x_1} f(x_0)\,\mathrm{d}x + \int_{x_1}^{x_2} f(x_1)\,\mathrm{d}x + \cdots + \int_{x_{n-1}}^{x_n} f(x_{n-1})\,\mathrm{d}x.$$

Cette expression peut être calculée comme la somme des aires de chacun des n rectangles (de largeur Δn et de longueur $f(x_i)$). Ainsi, l'approximation de $\int_a^b f(x)\,\mathrm{d}x$ devient :

$$\sum_{i=0}^{n-1} \Delta n * f(x_i).$$

On vous demande d'évaluer l'approximation de $\int_a^b f(x)\,\mathrm{d}x$ par cette méthode. Pour cela, vous devez faire croître la valeur de n (multiplier n par 2 à chaque étape) jusqu'à ce que la différence des résultats obtenus pour deux étapes consécutives soit inférieure à une certaine valeur $\varepsilon \in \mathbb{R}_0^+$ donnée (condition d'arrêt). Commencez toujours avec $n = 1$.

Questions

Donnez le code Python de la fonction suivante :

a) Une fonction `approx_int(f,a,b,epsilon)` qui retourne l'approximation, par la méthode des rectangles inférieurs, de l'intégrale d'une fonction `f` entre les bornes `a` et `b`, à `epsilon` près (voir condition d'arrêt ci-dessus). Cette fonction a pour préconditions que les paramètres entrés respectent les hypothéses sur f, a et b (f continue, $a < b$,...).

Problème 3 : ordonnanceur de tâches

Un ordonnanceur de tâches a pour objectif de trouver un ordre dans lequel différentes tâches numérotées peuvent être effectuées tout en respectant certaines contraintes de précédence. Une contrainte de précédence (i, j) stipule que la tâche numéro j doit être réalisée *avant* d'effectuer la tâche numéro i.

A titre d'exemple, considérez un système informatique comprenant 4 tâches devant être réalisées, numérotées de 0 à 3. Les contraintes de précédence de ce système sont les suivantes : $(2, 0), (2, 1), (0, 1), (3, 2)$. Un ordonnanceur de tâches trouvera l'ordre de réalisation des tâches suivant : $1, 0, 2, 3$.

Un système de tâches à ordonnancer peut être encodé sous la forme d'une matrice carrée de taille $n \times n$ où n est le nombre de tâches à ordonnancer. Dans cette matrice, l'élément en position (i, j) a la valeur $True$ si une contrainte de précédence (i, j) a été spécifiée, $False$ sinon. Ainsi, dans notre exemple, le système serait encodé sous la forme de la matrice 4×4 suivante :

$$\begin{pmatrix} False & True & False & False \\ False & False & False & False \\ True & True & False & False \\ False & False & True & False \end{pmatrix}$$

Il n'est pas toujours possible de trouver un ordre de réalisation des tâches. Par exemple, dans un système à deux tâches, si les contraintes de précédence sont $(0, 1), (1, 0)$, il n'existe aucun ordre permettant de réaliser les tâches tout en respectant les contraintes.

Questions

Donnez le code Python des fonctions suivantes :

a) Une fonction `creer_matrice(n_taches, contraintes)` qui, étant donnés un nombre `n_taches` de tâches et une liste `contraintes` de couples représentant la liste des contraintes d'un système, retourne la matrice du système correspondante. Cette fonction a pour précondition que `n_taches` ≥ 1.

b) Une fonction `ordonnanceur(matrice)` qui, étant donnée une matrice encodant le système à ordonnancer, retourne une liste d'entiers représentant un ordre dans lequel les tâches peuvent être réalisées. Dans notre exemple, cette fonction retournerait `[1, 0, 2, 3]`. Cette fonction a pour précondition que la matrice fournie en paramètre encode un système pour lequel il existe toujours un ordre de réalisation des tâches. Notez que la fonction `ordonnanceur` ne doit pas avoir d'effet de bord.

Problème 4 : recherche dichotomique

Etant donné une séquence triée, la recherche dichotomique permet de chercher si un élément x est présent dans la séquence selon le principe suivant. On calcule l'indice du milieu de la séquence m et on regarde l'élément y correspondant :

- si x est égal à y, alors on l'a trouvé et son indice est m ;
- si x est plus petit que y, alors on doit le chercher dans la première partie de la séquence (avant m) ;
- sinon, x doit se trouver dans la deuxième partie de la séquence (après m).

En répétant ce processus, on réduit la taille des sous-séquences à explorer tant qu'on ne trouve pas x. Si finalement on explore une sous-séquence de taille 0 ou 1 qui ne contient pas x, c'est qu'il ne se trouve pas dans la séquence.

La fonction suivante implémente la recherche dichotomique et retourne l'indice de x s'il est présent, ou *None* sinon. Cette fonction a pour préconditions que la liste t est triée (et que les opérateurs de comparaisons peuvent être appliqués sur ses éléments).

```
1  def recherche(t, x):
2      start = 0
3      end = len(t) - 1
4      mid = (end - start) / 2
5      while (end - start > 0) and x != t[mid]:
6          if x < t[mid]:
7              end = mid - 1
8          else:
9              start = mid + 1
10         mid = start + (end - start) / 2
11     if len(t) > 0 and x == t[mid]:
12         return mid
13     else:
14         return None
```

Questions

Soit n le nombre d'éléments dans la liste.

a) Prouvez que la fonction `recherche` s'arrête toujours si $n \geq 0$.

b) Prouvez que l'assertion suivante est un invariant de la boucle `while` :
Si x est dans t, alors son indice est compris entre start et end.

c) Utilisez l'assertion précédente pour prouver que la fonction `recherche` est exacte (considérez les deux cas où x est présent dans t et où il ne l'est pas).

d) Donnez et prouvez la complexité dans le pire des cas de la fonction `recherche`, en fonction de n.