IUT de Montpellier
TD1 bis : prendre ou ne pas prendre ? Là n'est pas la question, faisons les deux !
M3103 Algorithmique avancée

1 Echauffement sur des variantes de recherche de sous suites croissantes

Toutes les méthodes sont à écrire récursivement.

Exercice 1. Sous suite croissante consécutive

On souhaite écrire un algorithme qui, étant donné un tableau d'entiers t, calcule la longueur de la plus grande sous suite consécutive croissante. Plus formellement, une sous suite consécutive croissante est une séquence d'indices du tableau $(i, i+1, .., i+k)$ telle que $t[i] \leq t[i+1] \leq \cdots \leq t[i+k]$, et la longueur d'une telle suite est $k+1$. Par exemple, pour $t = [5, 6, 2, 5, 3, 7, 8, 9, 1, 11]$, la plus longue sous suite consécutive croissante est $(4, 5, 6, 7)$ (car $t[4] \leq t[5] \leq t[6] \leq t[7]$), et est donc de longueur 4.

Question 1.1.
Considérons la spécification naturelle `int sousSuiteConsAux(int []t, int i)` qui, étant donné un tableau d'entiers positifs t, et un i avec $0 \leq i \leq t.length$, calcule la longueur de la plus longue sous suite consécutive croissante du sous tableau $t[i..(t.length-1)]$. Imaginons que `sousSuiteConsAux(t,i+1)` retourne m. Quelles sont les valeurs possibles de `sousSuiteConsAux(t,i)` ? Pourquoi ne peut on pas "facilement" déterminer la bonne valeur de `sousSuiteConsAux(t,i)` ?

La question précédente montre donc que la spécification la plus naïve est peu pratique pour le raisonnement par récurrence, puisque en fait `sousSuiteConsAux` a besoin d'une fonction auxiliaire, que nous allons écrire maintenant.

Question 1.2.
Ecrire la fonction `int maxDebFixe(int []t, int i)` qui, étant donné un i avec $0 \leq i < t.length$, calcule la longueur de la plus grande sous suite consécutive croissante démarrant en i. Par exemple, avec t le tableau précédent, maxDebFixe(t,6) retourne 2, maxDebFixe(t,7) retourne 1, et maxDebFixe(t,8) retourne 2.

Question 1.3.
En utilisant des appels à maxDebFixe, écrire la fonction `int sousSuiteCons(int []t, int i)` de la question 1.1. En déduire l'algorithme `int sousSuiteCons(int []t)` qui répond à l'objectif initial!

Exercice 2. Sous suite croissante non consécutive

On souhaite écrire un algorithme qui, étant donné un tableau d'entiers t, calcule la longueur de la plus grande sous suite croissante. Plus formellement, une sous suite croissante est une séquence d'indices du tableau $(i_1, i_2, .., i_k)$ (avec $i_l < i_{l+1}$) telle que $t[i_l] \leq t[i_{l+1}]$ pour tout l. La longueur d'une telle suite est k. Par exemple, pour $t = [5, 6, 2, 5, 3, 7, 8, 9, 1, 11]$, la plus longue sous suite croissante est $(2, 4, 5, 6, 7, 9)$, et est donc de longueur 6 [1].

Supposons que l'on veuille calculer la plus grande sous suite croissante de $t[i..(t.legnth-1)]$. Le problème est qu'a priori on ne sait pas si la meilleur sous suite contient la case i ou non ... on va donc essayer les deux ! Pour ce faire, on va donc dire que

[1] Remarquez que contrairement au cas consécutif, résoudre ce problème en itératif (en temps de calcul raisonnable!) requiert un peu de réflexion.

- soit on prend la case i dans notre sous suite, et il faudra donc "indiquer" à l'appel récursif que les éléments suivants devront être supérieurs à $t[i]$ (pour que $t[i]$ concaténé à la suite trouvée récursivement soit bien croissante)
- soit on ne prend pas la case i, et l'on cherchera alors la meilleure sous suite à partir de $i+1$

Pour pouvoir "donner" à l'appel récursif les informations mentionnées ci-dessus, il faut donc enrichir la spécification, d'où l'algorithme suivant.

Question 2.1.
Ecrire l'algorithme `int sousSuiteNonCons(int []t, int i, int x)` ayant les spécifications suivantes :

- prérequis : $0 \leq i \leq t.length$
- action : calcule la longueur de la plus grande sous suite croissante du sous tableau $t[i..(t.length-1)]$ dont tous les éléments (les $t[i_l]$) sont plus grands que x.

Par exemple, avec $t = [-1, 3, 10, 5, 3, 4, 8, 6, 7]$ le tableau précédent, sousSuiteNonCons(t,2,0) retourne 4, et sousSuiteNonCons(t,2,4) retourne 3.

Question 2.2.
En déduire l'algorithme `int sousSuiteNonCons(int []t)` qui répond à l'objectif initial!.

2 Une remarque importante

Les algorithmes tels que sousSuiteNonCons sont inutilisables en pratique car ils font typiquement de l'ordre de 2^n opérations sur un tableau de taille n. Cependant, il existe une technique générale (appelée programmation dynamique, hors programme!) qui permet en modifiant quelques lignes (toujours les mêmes) de rendre ces algorithmes très efficaces. Même si nous n'abordons pas cette technique ici, il est donc bon de s'exercer à trouver d'abord les algorithmes récursifs tels ceux ci-dessus !

3 Application à d'autres problèmes

On va voir que pour beaucoup de problèmes ayant l'air difficiles (pour lesquels typiquement on n'a pas de critère "facile" pour déterminer, en arrivant sur la case i, si l'on doit utiliser cette case ou non) peuvent se résoudre avec un raisonnement semblable à celui utilisé pour sousSuiteNonCons : on va essayer les deux "débuts de solution" (prendre ou pas la case i), et informer correctement l'appel récursif.

Exercice 3. Meilleure sous somme
On souhaite écrire un algorithme qui, étant donné un tableau d'entiers t, calcule la plus grande somme d'éléments consécutifs. Par exemple, pour $t = [3, -4, 5, -1, 3, -2, -3, 6, -10, 4, 2]$, la plus grande somme d'éléments consécutifs est 8 ($8 = t[2] + t[3] + t[4] + t[5] + t[6] + t[7]$).
On applique la technique précédente :

- soit on prend la case i dans notre somme d'éléments consécutifs, et il faudra donc "indiquer" à l'appel récursif que la sous somme est "en cours" de calcul (pour ne pas avoir de trous)
- soit on ne prend pas la case i, et l'on cherchera alors la meilleure sous somme $i+1$

Question 3.1.
En vous inspirant de l'idée précédente, **spécifiez** puis écrivez un algorithme répondant au problème.

Exercice 4. Retour sur sous suite croissante consécutive Revenons sur le premier problème de plus longue sous suite croissante consécutive. Même si cela est artificiel (ce problème est trivial en itératif!), on applique la technique précédente :

- soit on prend la case i dans notre sous suite, et il faudra donc "indiquer" à l'appel récursif
  - que l'on a déjà démarré notre sous suite (pour pas qu'il n'y ait de trous)
  - que les éléments devront être supérieurs à $t[i]$ (pour que $t[i]$ concaténé à la suite trouvée récursivement soit bien croissante)
- soit on ne prend pas la case i, et l'on cherchera alors la meilleure sous suite à partir de $i + 1$

Question 4.1.
En vous inspirant de l'idée précédente, **spécifiez** puis écrivez un algorithme répondant au problème.