Algorithmique

TD4 : Entraînement

1 Analyse d'algorithmes

1. Donner la valeur asymptotique des sommes suivantes, en utilisant l'encadrement du cours reposant sur une intégrale.

   (a) $A_n \coloneqq \sum_{x=1}^{n} x^2$ (b) $B_n \coloneqq \sum_{x=1}^{n} \sqrt{x}$ (c) $C_n \coloneqq \sum_{x=1}^{n} \frac{1}{\sqrt{x}}$ (d) $D_n \coloneqq \sum_{x=1}^{n} \frac{1}{x}$

2. Pour chacun des algorithmes suivants, exprimer un invariant de boucle qui donne la valeur des variables internes. En déduire leur complexité (on pourra utiliser les calculs de la question précédente).

Algorithme 1 :

Entrées : n: entier
$i \leftarrow 0, j \leftarrow 0, k \leftarrow 1$
tant que $i \leq n$ **faire**
  `Afficher(i,j,k)`
  $j \leftarrow j + k$
  $i \leftarrow i + j$
  $k \leftarrow k + 2$
fin

Algorithme 2 :

Entrées : n: entier
$i \leftarrow 0, j \leftarrow 1$
tant que $i \leq n$ **faire**
  `Afficher(i,j)`
  $i \leftarrow i + \frac{1}{j}$
  $j \leftarrow j + 1$
fin

2 Recherche du maximum dans un tableau structuré

On considère un tableau `tab` de taille n qui représente une séquence unimodale — les valeurs de `tab` sont strictement croissantes jusqu'à une certaine position $i_0 \in [0, n-1]$ (on ne connaît pas la valeur de i_0), puis strictement décroissantes. On cherche à trouver la valeur de l'élément maximum dans `tab`.

1. Écrire un algorithme naïf de complexité $O(i_0)$ (à justifier) pour résoudre ce problème.
2. On veut maintenant un algorithme reposant sur une recherche dichotomique.
   (a) En fonction des valeurs des deux éléments les plus au centre dans `tab`, dans quelle partie du tableau doit-on continuer la recherche ?
   (b) Écrire un algorithme de complexité $O(\ln n)$ qui renvoie la valeur de l'élément maximum dans `tab`.

3 Un problème dans les tableaux

Le but de cet exercice est de résoudre le problème MAXSUBARRAY : étant donné un tableau `t` contenant n nombres (positifs et négatifs), retourner la plus grande somme (non vide) de valeurs contiguës au sein de `t`, à savoir

$$\max_{0 \leq i_0 \leq i_{\max} \leq n} \sum_{i=i_0}^{i_{\max}} \texttt{t[}i\texttt{]}.$$

1. Quelle est la solution au problème si tous les nombres dans `t` sont négatifs ? Et s'ils sont tous positifs ?

2. *Première approche naïve.* Supposons que l'on dispose de la fonction `sum(t,`i_0`,`$i_{\max}$`)` qui retourne la somme des éléments de `t` de la position i_0 à la position $i_{\max}$, de complexité $O(\max(1, i_{\max} - i_0))$. Écrire un algorithme faisant appel à `sum` pour résoudre le problème MAXSUBARRAY (sans utiliser d'opérations arithmétiques). Quelle est sa complexité (en fonction de n) ?

3. *Première amélioration.* Implanter une fonction `maxSumFrom(t,`i_0`)` qui retourne la plus grande somme de valeurs consécutives démarrant à la position i_0, de complexité linéaire (effectuer le calcul de complexité). Écrire un algorithme faisant appel à `maxSumFrom` pour résoudre le problème MAXSUBARRAY (sans utiliser d'opérations arithmétiques). Quelle est sa complexité (en fonction de n) ?

4. *Vers une complexité linéaire.* Nous allons maintenant utiliser des structures annexes qui vont nous permettre de proposer une solution au problème MAXSUBARRAY de complexité linéaire.

   (a) Nous allons construire un tableau `maxSumTo` qui contiendra en chaque position i la plus grande somme de valeurs consécutives (potentiellement vide) se terminant à la position i. Que vaut `maxSumTo[0]` ? Si on a déjà calculé `maxSumTo[`i`]`, comment peut-on calculer `maxSumTo[`$i+1$`]` en temps $O(1)$?

   (b) Supposons que l'on ait déjà calculé toutes les valeurs de `maxSumTo`. Comment peut-on obtenir le résultat au problème MAXSUBARRAY en temps $O(n)$?

   (c) En déduire un algorithme de complexité linéaire pour résoudre le problème MAXSUBARRAY.

4 Analyse de séquence

Un problème classique de bio-informatique est la détection d'occurrences de gènes au sein d'une séquence ADN. Cela se modélise par la recherche d'un certain mot au sein d'une chaîne de caractères à valeurs dans $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$. La séquence d'ADN peut subir des mutations, qui la modifient légèrement localement. Pour simplifier, nous considérerons uniquement les mutations qui consistent à insérer un caractère en une certaine position de la séquence.

Étant donné un mot `w` de taille k et une séquence `S`, on dit que `w` a une occurrence à la position i dans `S` si `w[`j`]` $=$ `S[`$i+j$`]` pour tout $j \in \{0, \ldots, k-1\}$. On dit que `w` a une occurrence t-défectueuse dans `S` s'il existe un mot `w'` obtenu en insérant au plus t caractères dans `w` tel que `w'` a une occurrence dans `S` (ou bien si on peut obtenir une séquence `S'` en retirant au plus t caractères de `S` telle que `w` a une occurrence dans `S'`). Un exemple est illustré dans la Table 1.

A	C	A	G	A	A	T	T	T	C	A	C	C	T	G	T
				A	A	T	•	T	C	•	C				

Table 1: Une occurrence 2-défectueuse du mot `AATTCC` dans la séquence `ACAGAATTTCACCTGT`.

Dans ce qui suit `w` est une chaîne de caractères de taille k qui représente un mot (un gène), `S` est une chaîne de caractère de taille $n \geq k$ qui représente une séquence (ADN), et $t \geq 0$ est un entier.

1. Implanter une fonction `occ(w,S)` qui renvoie un booléen qui indique si le mot `w` a une occurrence dans la séquence `S`. Quelle est sa complexité (en fonction de k et n) ?

2. Implanter une fonction `occDefect(w,S,t)` qui renvoie un booléen qui indique si le mot `w` a une occurrence t-défectueuse dans la séquence `S`. Quelle est sa complexité (en fonction de k, n, et t) ?

3. On dit que `w` est un *sous-mot* de `S` s'il a une occurrence t-défectueuse de `S` pour un certain $t \leq n$. En d'autres termes, il existe une suite strictement croissante d'indices $i_0, \ldots, i_{k-1}$ telle que `w` $=$ `S[`i_0`]` $\cdots$ `S[`i_{k-1}`]`.

   (a) Soit i_0 la première occurrence de la lettre `w[0]` dans `S`. Prouver que `w` est un sous-mot de `S` si et seulement si `w[1]` $\cdots$ `w[k-1]` est un sous-mot de `S[`i_0+1`]` $\cdots$ `S[n-1]`.

   (b) En déduire une implantation d'une fonction `EstSousMot(w,S)`, de complexité $O(n)$ (à justifier), qui renvoie un booléen qui détermine si le mot `w` est un sous-mot de `S`.