Algorithmique

DM

Merci d'indiquer votre numéro de groupe de TD sur votre copie. Tous les algorithmes sont à écrire en pseudo-code selon les modèles vus en cours (éviter les notations type C).

1 Décomposition en facteurs premiers

1. Écrire un algorithme `FacteursPremiers` prenant en entrée un entier n, et qui retourne la décomposition de n en facteurs premiers, sous la forme d'une liste $[(p_1, a_1), \ldots, (p_r, a_r)]$, de sorte que $n = \prod_{i=1}^{r} p_i^{a_i}$. 2 pts

2. Dans le pire cas, quelle est la complexité de l'algorithme `FacteursPremiers`, et quelle propriété a alors n ? Même question pour le meilleur cas. 1 pt

3. Même question, mais cette-fois-ci on suppose que l'on a accès à un tableau `Premier` de taille N tel que `Premier[i]` est le i-ème nombre premier (on a donc `Premier[0]` $= 2$, `Premier[1]` $= 3$, `Premier[2]` $= 5$, etc), et que l'on a $n \leq N^2$. 1.5 pt

4. Exprimer la complexité dans le pire cas de l'algorithme à l'aide de la fonction $x \mapsto \pi(x)$ qui compte les nombres premiers inférieurs (ou égaux) à x. Utiliser le théorème des nombres premiers pour exprimer cette complexité à l'aide de fonctions usuelles. 1 pt

5. Modifier l'implémentation naïve du Crible d'Eratosthène afin qu'il renvoie un tableau `Fact` de taille $n + 1$ tel que `Fact[i]` contient la liste des facteurs premiers de i (sans leur multiplicité), pour tout $1 \leq i \leq n$. 2 pts

6. Exprimer la complexité (dans le pire cas) de cet algorithme, d'abord comme une fonction de n, puis comme une fonction de la taille n' de la sortie. En déduire le nombre moyen de facteurs premiers distincts d'un nombre i compris entre 1 et n. 1.5 pt

7. (Bonus) Écrire un algorithme `Décompositions` prenant en entrée un entier n, et qui retourne un tableau `decomp` de taille $n + 1$ tel que `decomp[i]` est le résultat de `FacteursPremiers(i)`. Sa complexité devra être strictement inférieure à celle de n appels à `FacteursPremiers`, et convenablement calculée. 2 pts

2 Tableaux et recherche

Dans cet exercice, on explore les possibilités de recherche dichotomique dans un tableau trié.

1. Écrire un algorithme `PointFixe` de complexité logarithmique qui prend en entrée un tableau `tab` d'entiers **distincts** triés par ordre croissant, et retourne une position i telle que `tab[i]=i`, ou -1 s'il n'existe pas de telle position. 1.5 pt

2. On considère un tableau `t` de taille n, qui contient des entiers distincts triés par ordre croissant, mais qui a subi un shift circulaire : il existe une position $j \geq 0$ telle que `t[j], t[j+1], ..., t[n-1], t[0], ..., t[j-1]` est une suite strictement croissante. Écrire un algorithme `TrouverDébut` de complexité logarithmique qui prend en entrée un tel tableau, et qui retrouve la position j à partir de laquelle il est strictement croissant. 1.5 pt

3. On veut faire une recherche d'un élément x au sein d'une matrice M de dimensions $n \times m$, dont chaque ligne et chaque colonne est croissante.

   (a) En fonction du résultat de la comparaison de x avec l'élement en haut à droite de la matrice M, dans quelle sous-matrice de M faut-il continuer la recherche ? 1 pt

(b) Combien de fois au maximum doit-on répéter l'opération précédente avant d'atteindre un cas de base ? 0.5 pt

(c) En s'appuyant sur le principe décrit précédemment, écrire un algorithme `EstPrésent` qui prend en entrée la matrice M et l'élément x, et qui retourne `Vrai` si x apparaît dans M, et `Faux` sinon. Quelle est sa complexité ? 1.5 pts

3 Nombres binomiaux

1. Écrire une fonction récursive `FactorielleTronquée` qui prend en entrée deux entiers $n \geq k \geq 0$, et qui renvoie la valeur de $\frac{n!}{k!}$. 1 pt

2. Écrire un algorithme `Binomial` qui prend en entrée deux entiers $n \geq k \geq 0$, et qui renvoie la valeur de $\binom{n}{k}$, en faisant appel à `FactorielleTronquée`. Quelle est sa complexité ? Quelles sont les limitations de cet algorithme (par exemple, est-il capable de calculer correctement $\binom{100}{98}$) ? 1.5 pt

3. Pour tout $n \geq k > 0$, comment obtient-on $\binom{n}{k}$ à partir de $\binom{n-1}{k-1}$? Écrire une fonction récursive `Binomial2` qui utilise cette propriété afin de calculer $\binom{n}{k}$. En quoi `Binomial2` est-elle meilleure que `Binomial`? 1.5 pt