Sujet TD 1, Licence 2, module I31

Dominique Michelucci, Université de Dijon

12 septembre 2012

1. Vérifiez que vous saurez programmer en TP l'algorithme d'Euclide étendu (pgcd et coefficients de Bézout).

Idem pour le crible d'Eratosthène.

2. Dessinez la courbe $y = 1/x$ pour $x \in [1/2, 5]$. En déduire :

$$1.083 \approx 13/12 = 1/2 + 1/3 + 1/4 < \log 4 < 1 + 1/2 + 1/3 = 11/6 \approx 1.833$$

Déduisez en une formule pour encadrer $\log x$.

Comment peut-on améliorer la précision ?

Rappel (vu en cours) : $\log x = \int_{t=1}^{t=x} \frac{1}{t} dt$ est l'aire sous l'arc de l'hyperbole $(y = 1/x)$. $H(k) = 1 + 1/2 + 1/3 + \ldots 1/k$ est la série harmonique, tronquée au terme k. On en déduit que pour un grand entier n, $\log n \approx H_n$. En fait $\gamma = \lim_{n\to\infty} H_n - \log n \approx 0.577$ est appelée constante d'Euler.

3. D'après le petit théorème de Fermat, si p est un entier premier, alors pour tout $k \in \mathbb{N}$, on a : $1 = k^{p-1}$ modulo p. Un test probabiliste calcule k^{p-1} modulo p pour un certain nombre (disons de l'ordre de $\log p$) de valeurs entières aléatoires de k (dans $[1, p-1]$) ; si pour tous les tests, $1 = k^{p-1}$ modulo p le nombre p est "probablement" premier. Sinon (pour un des tests, $1 \neq k^{p-1}$ modulo p),le nombre p n'est sûrement pas premier. Comment pouvez vous optimiser le calcul de k^{p-1} ?

Remarque : $a^d = a^{d \bmod \phi(p)}$ modulo p pour tout entier (premier ou non premier), où $\phi(p)$ est l'indicatrice d'Euler ; $\phi(p)$ est le nombre d'entiers dans $[1, p-1]$ et premiers avec p (leur PGCD avec p vaut 1). Si p est premier, $\phi(p) = p - 1$.

Remarque : les nombres de Carmichael passent tous ces tests, mais ne sont pas premiers. Il vaut mieux utiliser le test probabiliste de primalité de Solovay-Strassen, qui utilise aussi l'exponentiation rapide, ou le test de Miller-Rabin.

4. Ecrire une classe ListEntier en Java. Null est la liste vide. Une liste non vide a une tête, qui est un entier. Elle a aussi une queue, qui est une liste (éventuellement vide). hd(l) retourne la tête (ou bien l.hd() si vous préférez). tl(l) retourne la queue de la liste l (ou bien l.tl() si vous préférez). L'appel cons(t : un entier, q : une liste) retourne une nouvelle liste, de tête t, de queue q. Remarquez que cette interface ne permet pas de modifier une liste.

5. Ecrire une fonction (ou une méthode...) pour tester si une liste contient un entier donné.

6. Ecrire une fonction (ou une méthode...) pour créer une liste égale à une liste donnée, mais où la première occurence d'un entier e (passé en paramètre) est supprimée. Noter que les queues des 2 listes peuvent être partagées.

7. Ecrire une fonction (ou une méthode...) qui rende le k ième élément (un entier) d'une liste donnée. Par convention le premier est le 0 ième. Ne pas perdre de temps avec le traitement des erreurs.

8. Ecrire (en récursif et/ou itératif) une fonction rendant la longueur d'une liste : 0 pour NULL, 1 + lgr(tl(l)) pour l non vide.

9. Ecrire une fonction (méthode...) rendant le minimum d'une liste.

10. Ecrire le trif naïf d'une liste l. Si la liste est vide, elle est triée. Sinon soit m le minimum de l, et soit l'= suppression(l, m). Alors la liste triée est cons(m, trinaif(l')).

11. Ecrire le tri rapide d'une liste l. Vous prendrez comme pivôt le premier (pas le plus petit) élément de la liste .

12. Ecrire le tri rapide d'un tableau. Le tableau sera modifié.

13. Ecrire le tri par tas (ou heapsort) d'un tableau. Le tableau sera modifié.

S'il reste du temps :

14. Trouvez une méthode pour calculer le nombre de façons de choisir k objets parmi n (les coefficients binomiaux).

15. Trouvez une méthode (récursive) pour calculer tous les sous ensembles d'un ensemble donné d'entiers (qu'on supposera tous distincts). Les ensembles seront représentés par des listes.