TP 2

Judicaël Courant

26 septembre 2016

Note : il est interdit de toucher à l'ordinateur pendant les 30 premières minutes. Ne faites la deuxième partie que si vous avez complètement terminé la première.

1 Factorielle

1.1 Présentation du travail demandé

On veut écrire une fonction prenant en entrée un argument, censé être un entier naturel n et retournant $n!$. On propose ci-dessous trois algorithmes différents.

1. Vous écrirez les trois fonctions demandées sur papier.
2. Vous démontrerez rigoureusement leur correction.
3. Vous donnerez, pour chacun de ces algorithmes, une estimation asymptotique de l'espace nécessaire au calcul (vous justifierez votre estimation). En particulier, quel est l'espace de pile nécessaire pour chacun de ces algorithmes ? Cela risque-t-il de constituer une limitation pour certains de ces algorithmes ? Vous préciserez votre réponse.
4. Vous calculerez, pour chacun de ces algorithmes, le nombre de multiplications effectuées en fonction de n pour calculer $n!$.
5. Vous en déduirez une estimation asymptotique du temps de calcul de ces trois fonctions.
6. Vous les programmerez.
7. Vous les testerez sur quelques exemples, notamment pour les valeurs 0, 1, 2, 3, 4 et 10 (on rappelle que $10! = 3\,628\,800$) ; vous vérifierez en outre qu'elles donnent toutes les trois la même valeur pour 500.
8. Enfin, vous étudierez leurs performances (voir ci-dessous comment faire).

1.2 Les trois algorithmes considérés

Les trois versions demandées sont les suivantes :

Version itérative Vous appelerez votre fonction fact_it(n), elle devra calculer $n!$ de façon itérative.

Première version récursive Votre fonction s'appelera fact_rec(n) et elle utilisera les égalités $0! = 1$ et $n! = n \times (n-1)!$ pour tout n entier non nul.

Deuxième version récursive Vous écrirez une fonction récursive prod_range(a,b) prenant en argument deux entiers relatifs a et b et retournant la valeur $p(a,b)$ définie par

$$p(a,b) = \prod_{k \in [\![a,b[\![} k$$

Vous utiliserez pour cela les égalités suivantes :

$$p(a,b) = \begin{cases} 1 & \text{si } b \leq a; \\ a & \text{si } b = a+1; \\ p(a,m) \times p(m,b) \text{ où } m = \left\lfloor \frac{a+b}{2} \right\rfloor & \text{sinon.} \end{cases}$$

Vous en déduirez une fonction fact(n) faisant appel à prod_range pour calculer $n!$.

1.3 Étude des performances

Le but est de montrer les performances relatives des trois algorithmes. Vous calculerez donc le temps de calcul de $n!$ pour chacun des algorithmes pour n appartenant à **range**(a, b, c) où a, b et c sont à déterminer.

Pour calculer la durée d d'exécution d'une expression e, on peut utiliser le module timeit : timeit.timeit(**lambda** : e, number=1) retourne la durée recherchée.

Vous tracerez, à l'aide du module matplotlib.pylab (fonction pyplot) une courbe montrant les temps de calcul des trois fonctions que vous avez écrites.

Dans l'interprète interactif, les instructions n = fact(200000) et fact(200000) ont-elles des temps d'exécution significativement différents ? Expliquez. Vous justifierez votre réponse à l'aide de tests mettant en évidence le phénomène.

Le graphique que vous avez tracé est-il cohérent avec ce qui était attendu ? Expliquez et si besoin, raffinez votre modèle de calcul.

2 Permutations

On s'intéresse, dans cet exercice, à des permutations de $[\![0,n[\![$, où n est un entier. Une permutation σ sera représentée par un tableau Python de taille n contenant successivement les valeurs $\sigma(0)$, $\sigma(1)$, ... $\sigma(n)$.

1. Écrire une fonction identite(n) retournant le tableau de taille n représentant l'identité sur $[\![0,n[\![$.
2. Écrire une fonction compose(s_1, s_2) prenant en argument les représentations respectives s_1 et s_2 de deux permutations σ_1 et σ_2, supposées de même longueur, et retournant la représentation de $\sigma_1 \circ \sigma_2$. Quelle est la complexité de votre fonction ?
3. Écrire une fonction inverse(s) prenant en argument la représentation s d'une permutation σ et retournant la représentantion de σ^{-1}. On fera en sorte que la complexité de cette fonction soit en $O(n)$, où n est la taille du tableau s.