TP 2

M1201 - Mathématiques Discrètes

Jérôme HILDENBRAND

1. Fin du TP 1

Correction rapide. Prenez des notes sur votre fichier TP1.py.

2. TP 2 - Division Euclidienne

Nous entrons dans de l'algorithmique proche des cours vus en TD. Ce TP constitue un complément de cours, orienté évidemment vers l'informatique. Les algorithmes vus sont célèbres et font partie d'une culture informatique élémentaire. Les écrire vous permettra également d'augmenter vos capacités algorithmiques. Comme d'habitude, n'hésitez pas à commenter vos fonctions, de manière à en retirer des informations lors de votre relecture. CE TP SERA AUGMENTÉ LES SEMAINES SUIVANTES, CAR NOUS LE POURSUIVRONS SUR PLUSIEURS SÉANCES. CONSERVEZ DONC BIEN VOTRE FEUILLE DE CALCUL QUI SERA RÉUTILISÉE.

Première partie : manipulation des symboles / et %

1. **dernierChiffre(n)** Renvoie le dernier chiffre de l'entier n.

2. **deuxDerniersChiffres(n)** Renvoie les deux derniers chiffres de n. Exemple : $deuxDerniersChiffres(2756)$ renvoie 56.

3. **derniersChiffres(n,nbre)** Renvoie les nbre derniers chiffres de n. Exemple : $derniersChiffres(276553, 4)$ renvoie 6553.

4. **nombreChiffres(n)** Renvoie le nombre de chiffres de n.

5. **premierChiffre(n)** Renvoie le premier chiffre de n.

6. **premiersChiffres(n,nbre)** Renvoie les $nbre$ premiers chiffres de n. Exemple : $premiersChiffres(23456, 3)$ renvoie 234.

7. chiffres(n,debut,fin) Renvoie le bon intervalle de chiffres de n. Exemple : $chiffres(456345234, 3, 6)$ renvoie 6345.

8. Amélioration (bonus) La fonction $chiffres(n, debut, fin)$ doit en outre détecter si l'intervalle n'a pas de sens ($b < a$), et agir intelligemment si l'intervalle "sort" du nombre n. Exemple : $chiffres(456345234, 7, 28)$ répondra 234.

9. Un petit problème graphique On dispose d'une fenêtre graphique sur laquelle figure un quadrillage de 5867 cases par 34567 cases. Combien y a-t'il de cases ? Les cases sont numérotées de 1 (en haut à gauche) à ... (en base à droite). Sur quel ligne et quelle colonne se trouve la case 3456234 ?

Deuxième partie : programmation de la division euclidienne Bien entendu, dans cette partie l'utilisation des symboles / et % est rigoureusement interdite...

1. quotient(a,b) Programmer cette fonction qui prend en argument deux entiers positifs et renvoie le quotient dans la division euclidienne de a par b. Un message d'erreur sera imprimé en cas de division par 0.

2. reste(a,b) Même chose que la question 1, pour le reste cette fois-ci.

3. div(a,b) Même chose, mais renvoie q, r.

4. quotient2(a,b), reste2(a,b), div2(a,b) Ces fonctions améliorent les précédentes en gérant en plus les nombres négatifs.

Troisième partie : exponentiation modulaire Python étant un langage éminemment mathématique, beaucoup moins attaché à la rigueur informatique au d'autres langages plus proches de la machine, dispose de certains moyens préprogrammés pour effectuer des grandes opérations, qui dépassent normalement les capacités du type *int*. En particulier, le type *long int* est automatiquement utilisé dès que les choses deviennent violentes en terme de nombre de chiffres. L'opérateur %, qui normalement effectue un simple algorithme de division euclidienne, est donc (bien) programmée pour gérer des cas classiques de calculs énormes tels que $3^{23456}\%10$. Bien entendu, l'algorithme utilisé n'a RIEN À VOIR avec une simple division euclidienne (succession de soustractions), sans quoi cela prendrait des années de calcul.

Commençons par analyser les capacités de %. Un très brêve recherche sur Google, que dans ma bonté j'ai fait pour vous, permet de découvrir comment mesurer le temps d'exécution d'un programme. Voici le lien :

$http://bonjourdata.fr/afficher-le-temps-dexecution-dun-script-python/$

1. Tester des calculs de puissances importantes du genre (ab)%c.** Vous êtes dans une démarche d'éxpérimentation : faites varier a, b, c pour voir les limites de la fonction.

1bis. Tester de même le temps d'execution de reste(a^b,c) qui effectue le même calcul. Noter qu'elle est très vite larguée. Beaucoup plus vite que %, mais attention, % triche en utilisant ce qui vient dans la suite du TP !

2. puissance(a,b,c) - Algorithme d'exponentiation linéaire Cette fonction calcule a^b[de manière linéaire : exactement comme quand nous calculons les premières puissances de 7 modulo 10 en cours par exemple. On multiplie à chaque étape par a, et dès qu'on dépasse c, on se ramène en dessous grâces au propriétés des congruences. Ainsi les multiplications restent de taille limitée par c.

3. puissance2(a,b,c) - Algorithme d'exponentiation rapide Cette fonction est dite récursive (elle s'appelle elle-même). L'idée est de couper l'exposant en deux partie égales ou presque égales ("diviser pour régner"), qui seront ainsi plus faciles à calculer. Cette opération va se prolonger jusqu'à arriver à un exposant nul ou égal à 1. PRENDRE DES NOTES SUR LE PLAN DE BASE D'UNE FONCTION RÉCURSIVE (AU TABLEAU).

4. Comparer les performances de puissance et puissance2.

5. puissance3(a,b,c) - Algorithme itératif utilisant les cycles Comment faisons nous en cours pour calculer les grandes puissances à la main ? Faites de même avec la machine. Puis comparer les performances avec les deux précédentes fonctions.