TP4 Informatique
Boucle `while` et degrés
J1 MI 1003, groupe B3, Université Bordeaux

Toujours le chapitre 2

Exercice 1 Finir le chapitre 2 (sauf la partie 2.4 si vous êtes en retard).

Quelques exercices utilisant la boucle `while`

Exercice 2 Écrire une fonction `logarithme(a, n)` qui retourne le plus grand entier k tel que $a^k \leq n$ (on supposera $a > 1$).

Exercice 3 Écrire une fonction `racinecarree(n)` qui renvoie la partie entière de la racine carrée de n, et ceci **sans utiliser la fonction `sqrt` du module `math`**.

Degrés

Exercice 4 Faire les exercices 3.1.2 et 3.1.3 du fascicule.

Exercice 5 Ecrire le programme `nbAretes(G)` de l'exercice 3.3.7 du fascicule.

Exercice 6 Utiliser dans l'interprête la fonction `nbSommetsDegre(G,d)` de l'exercice 3.1.3 pour calculer le nombre de sommets de degré 1,2,3,4 dans les graphes renvoyés par les fonctions `construireGrille(m,n)` et `construireTriangle(n)` (en testant pour plusieurs valeurs de `m` et `n`), et vérifier ainsi la formule des poignées de mains.

Exercice 7 Ecrire le programme `cubique(G)` de l'exercice 3.3.2 du fascicule.

Somme des chiffres

Exercice 8 Ecrire une fonction `sommechiffres(n)` qui renvoie la somme des chiffres de l'entier n en base 10.

Exercice 9 Ecrire une fonction `chiffreassocie(n)` qui répète la fonction précédente `sommechiffres(n)` jusqu'à obtenir un chiffre compris entre 1 et 9.
Par exemple, calculons `chiffreassocie(8643)`. La somme des chiffres de 8643 vaut $8 + 6 + 4 + 3 = 21$. 21 n'est pas réduit à un seul chiffre, on continue donc l'opération : $2 + 1 = 3$. Le nombre 3 étant un chiffre compris entre 1 et 9, c'est le résultat souhaité. L'exécution de `chiffreassocie(8643)` doit donc renvoyer 3.
Vérifier sur plusieurs exemples que

$$\texttt{chiffreassocie}(\texttt{chiffreassocie}(a) \times \texttt{chiffreassocie}(b)) = \texttt{chiffreassocie}(a \times b).$$

Il s'agit de la preuve par 9.

Et enfin

Exercice 10 Écrire un programme `verifiedegres(G,l)` qui renvoie `True` si le nombre des degrés des sommets de `G` sont listés dans la liste `l` (c'est-à-dire `l[0]` correspond au nombre de sommets de degré 0, `l[1]` au nombre de sommets de degré 1, etc...) ; `False` sinon.

Exercice 11 Écrire un programme `listedegres(G)` qui renvoie la liste des degrés des sommets de `G` (c'est-à-dire le premier élément de la liste correspond au nombre de sommets de degré 0, le second au nombre de sommets de degré 1, etc...). Le dernier élément de la liste doit correspondre au nombre de sommets de degré maximal.
Entre autres, vérifier que `verifiedegres(G,listedegres(G))` renvoie toue le temps `True`.

Exercice 12 Utiliser la fonction `listedegres(G)` de l'exercice précédent sur les graphes `Europe`, `dodecaedre`, `isocaedre`, `construireArbre(d,h)` et `construireTriangle(n)` (en testant pour plusieurs valeurs de `d`, `h` et `n`).

Exercice 13 L'objectif est d'écrire un *test de primalité*, c'est-à-dire une fonction `premier(n)` qui retourne `True` si l'entier naturel $n > 1$ est premier et `False` sinon. Pour cela on parcourt les nombres supérieurs à 2 pour chercher un diviseur d de n : dès que l'on en trouve un on arrête les calculs, n n'est pas premier.

1. Si on ne trouve pas de diviseur $d \leq \sqrt{n}$, on est sûr que n est premier : pourquoi ? Comment effectuer le test $d \leq \sqrt{n}$ sans utiliser de fonction « racine carrée » ?
2. Écrire la fonction `premier(n)`.
3. Tester cette fonction en affichant tous les nombres premiers inférieurs à 1000.
4. Améliorer `premier(n)` en traitant à part le cas où n est pair ; dans le cas où n est impair, il suffit ensuite de chercher un diviseur d impair.