Maths pour l'info, TP3 : Génération et dénombrement

Ce TP est à réaliser en langage Java, avec les outils dont vous avez l'habitude.

Consigne universelle : à chaque fois que vous écrivez une fonction, testez-la sur plusieurs exemples ! Et aussi : si vous avez plusieurs méthodes pour calculer la même chose, testez sur des exemples qu'elles renvoient bien le même résultat.

Exercice 1 (Relations symétriques)**.** Dans cet exercice, on s'intéresse à la génération et au dénombrement des relations symétriques d'un ensemble. Vous compléterez le squelette de fichier disponible à l'adresse suivante :
`http://www.lri.fr/~blsk/Tp3RelSym.java`
Ce fichier contient déjà deux méthodes qui peuvent vous être utiles pour tester votre code :

- `bool estSym(int[][])` renvoie `true` si la matrice passée en argument représente une relation symétrique ;
- `void affiche(int[][])` affiche la matrice passée en argument.

Questions

a) Combien y a-t-il de relations symétriques sur un ensemble de cardinal n ? Établissez une formule, puis écrivez une méthode `public static int nbSym(int)` qui calcule et renvoie ce nombre.

b) Écrivez une méthode `public static boolean ithBit(int n, int i)` qui renvoie `true` si le bit d'indice i de n vaut 1. Point sur l'écriture binaire : l'écriture binaire du nombre 6 est 110. Le bit d'indice 0 et le bit le plus à droite (0), le bit d'indice 1 est le bit du milieu (1), le bit d'indice 2 est le bit suivant en allant vers la gauche (c'est-à-dire 1, le bit de gauche).

c) Écrivez une méthode `public static boolean[][] kthRel(int n, int k)` qui prend un entiers k et qui renvoie une matrice de côté n dans laquelle la case d'indice (i, j) est donnée par le bit d'indice $n * i + j$ de k.

d) En vous servant des méthodes précédentes, écrivez une méthode `public static int enumSym(int n)` qui compte les relations symétriques sur l'ensemble $[0..(n-1)]$ en les énumérant toutes. Comparez le résultat avec celui donné par la formule de la première question.

e) Écrivez une méthode `public static int[][] genSym(int)` qui génère de manière aléatoire une relation symétrique sur l'ensemble $[0..(n-1)]$.

Exercice 2 (Rugby)**.** Au rugby, différentes situations permettent de marquer différents nombres de points :

- Marquer un essai donne 5 points, et donne droit à tenter une transformation, qui apporte 2 points supplémentaires si elle est réussie.
- Un tir entre les poteaux donne 3 points.
- Une pénalité donne 3 points si elle est réussie.

L'objectif de cet exercice est d'écrire différentes méthodes qui, étant donné un nombre de points total, dit de combien de manières différentes ce score peut être réalisé dans un matche de rugby (la fonction renverra donc 0 si le score n'est pas réalisable).

Version récursive. Pour l'instant, on donne de l'importance à l'ordre des différents marques.

- Question préparatoire : soit un score S supérieur ou égal à 7. Combien y a-t-il de manières d'atteindre ce score en terminant par un essai transformé ?
- De même, combien y a-t-il de manières d'atteindre le score S en terminant par un essai non transformé ? Par un tir ? Par une pénalité ?
- Des deux points précédents, déduisez une formule récursive donnant le nombre de séquences de marques atteignant un score S donné. Écrivez une méthode `public static int nbSeq(int S)` effectuant ce calcul.

Version dynamique. La version récursive précédente est peu efficace, car elle recalcule plusieurs fois les mêmes résultats. On propose donc l'astuce suivante : pour calculer le nombre de séquences atteignant le score S, créer un tableau t de taille $S+1$, dans lequel $t[i]$ est destiné à contenir le nombre de séquences atteignant le score i. Remplir le tableau à partir de l'indice 0 et jusqu'à l'indice S, en suivant la formule récursive, mais en allant chercher dans le tableau les résultats précédents déjà calculés plutôt qu'en faisant un appel récursif.

- Écrire une méthode `public static int nbSeqDynamique(int S)` qui met en œuvre cette technique.

Version énumérative. Cette fois, on ne veut plus compter le nombre de séquences arrivant à un score S, mais le nombre de quadruplets (et, ent, t, p) donnant un score S, où et désigne le nombre d'essais transformés, ent le nombre d'essais non transformés, t le nombre de tirs, et p le nombre de pénalités. Par rapport aux questions précédentes, cela revient à ne pas tenir compte de l'ordre.

- Question préparatoire. Écrivez une méthode `public static int nbQuadPrep(int S)` qui énumère tous les quadruplets d'entiers compris entre 0 et S, et qui compte ceux pour lesquels la somme des quatre éléments vaut S.
- En vous inspirant de la méthode précédente, écrivez une méthode `public static int nbQuadEnum(int S)` qui compte le nombre de quadruplets (et, ent, t, p) donnant un score S.

Version D. Comme dans la version précédente, on s'intéresse maintenant seulement aux quadruplets, et plus à l'ordre des marques.

- Écrivez une méthode `public static int nbPaires(int S)` qui compte le nombre de paires d'entiers dont la somme vaut S, puis écrivez une méthode `public static int nbTriplets(int S)` qui utilise la méthode précédente et compte le nombre de triplets d'entiers dont la somme vaut S.
- En vous inspirant de la question précédente, écrivez une méthode `public static int nbQuadD(int S)` qui compte le nombre quadruplets (et, ent, t, p) donnant un score S.