Document d'accompagnement

TP 1-2

1 Suites récurrentes

Ici, la récursivité s'utilise pour le calcul de valeurs numériques, en suivant la définition par récurrence des suites.

Exemple : fonction récursive calculant la factorielle d'un nombre n (produit des entiers de 1 à n), définie par **n! = (n-1)! * n**, avec **0! = 1**

En java :

```
public static int facto(int n){
   if(n==0) // Cas de base
      return 1;
   else // Cas général de la récursivité
      return facto(n-1) * n;
}
```

En python :

```
def facto(n):
   if n==0: # Cas de base
      return 1
   else: # Cas général de la récursivité
      return facto(n-1) * n
```

A. Pour la vérification, on utilisera la fonction Math.pow(a,b) en java, ou l'opérateur ** en python.

B. Pour c_n, conjecturer une formule (pas besoin de calcul « compliqué »).

C. 100, 300, 1100, … Que se passe-t-il à partir de la 13ème année ?

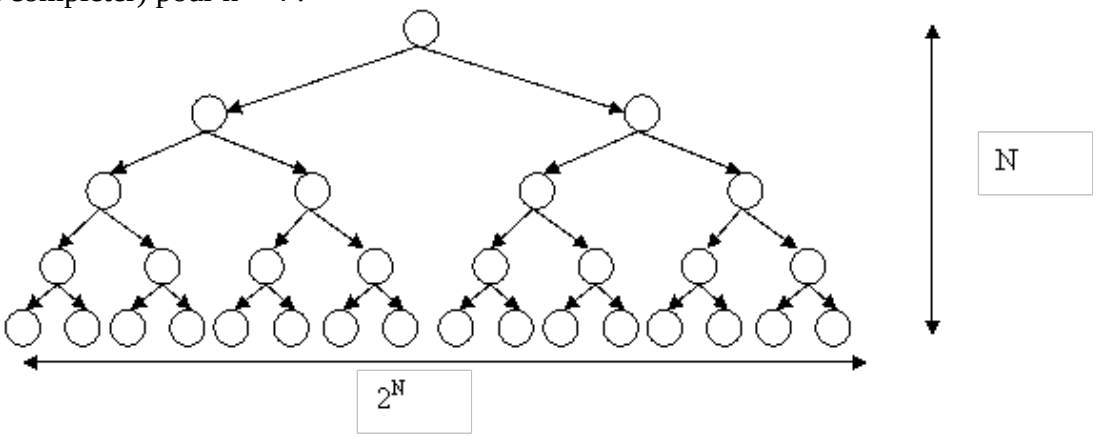
D. 1000, 1550, 2127.5, …

E. La racine carrée s'obtient avec la fonction sqrt (il faut importer la bibliothèque mathématique de Python avec l'instruction « from math import sqrt » en début de fichier). L'entier le plus proche s'obtient avec la fonction round.

2 Séquences binaires

Ici la récursivité nous permet de construire des séquences binaires (mots constitués de 0 et de 1). Dans ce cas, le résultat est construit lors du parcours des branches et est atteint quand on parvient aux feuilles.

Exemple (à compléter) pour n = 4 :



On peut stocker les séquences dans des tableaux (ou listes), ou dans des chaînes de caractères, en concaténant les 0 et les 1, à chaque appel récursif.

La fonction construisant les séquences peut retourner le résultat, ou bien simplement l'afficher (on affiche quand on parvient aux feuilles (en bas) contenant les séquences complètes.

A. Modèle pour la suite des exercices.

Exemple de code si on stocke les séquences dans des chaînes (sans les retourner) :

En java :

```
public static void bin(int n,String chaine){
   if(n==0) System.out.println(chaine);
   else {
      bin(n-1,chaine+"0");
      bin(n-1,chaine+"1");
   }
}
```

En python :

```
def bin(n, chaine):
   if(n==0):
      print(chaine, end=" ")
   else:
      bin(n-1,chaine+"0")
      bin(n-1,chaine+"1")
```

B. Plus dur. A faire après avoir fait B, C, D, E.

Technique : à partir de l'arbre de génération des séquences binaires, modifier certaines concaténations afin d'obtenir l'ordre du code de Gray. Modifier le code en conséquence.
C. Erreur dans la formule, lire F_{n-2} au lieu de F_{2-1}.

3. Combinaisons
Erreur dans les conditions, lire 'si k = 0' au lieu de 'si n = 0', et lire 'si n ≥ k > 0' au lieu de 'si n >= 1'.
Même principe, mais on concatène les chiffres 1, 2, 3, … au lieu des 0 et 1.

TP 3-4
1. Triangle de Pascal
Version récursive : coder la fonction calculant les $c_{i,j}$ en suivant la définition, puis appelez cette fonction pour calculer et afficher les coefficients binomiaux du triangle au sein d'une double boucle dans le main.
Version itérative : utiliser un tableau à deux dimensions pour stocker et afficher toutes les valeurs du triangle (sans utiliser la fonction récursive).

2. Compositions
Utiliser une boucle pour les appels récursifs dans la fonction.

TP 4
1. Nombre de Stirling
Fonction calculatoire. Vérifier avec les valeurs des Nombre de Stirling de seconde espèce sur https://fr.wikipedia.org/wiki/Nombre_de_Stirling.

2. Exponentiation rapide
Il s'agit d'implémenter l'algorithme le plus efficace calculant la puissance n-ième d'un nombre.
Suivre la définition (sans utiliser la fonction Math.pow !).
Adapter ensuite la fonction pour qu'elle ne comporte plus qu'**un seul** appel récursif.

3. Exponentiation et nombre de Fibonacci
Adapter l'algorithme précédent pour qu'il calcule la puissance n-ième d'une matrice 2x2.
L'utiliser ensuite pour effectuer le calcul donné (donc de la manière la plus efficace possible).