TP n°5

Récursivité

Pour rappel, voici un exemple de programme récursif, calculant factorielle.

```
static int fact(int n){
   if (n<=0) return 1; // cas de base
   else return n*fact(n-1); // recursion
}
```

Si on appelle `fact(n)`, la méthode appelle `fact(n-1)`, qui à son tour appelle `fact(n-2)`, etc jusqu'à `fact(0)`. La condition du `if` est alors vérifiée, donc `fact(0)` renvoie `1`, autrement dit la récursivité se termine.

Pour rappel encore, en écrivant une fonction récursive, il faut toujours :
– définir un cas de base, qui renvoie un résultat sans appel récursif,
– définir le cas récursif de façon à être certain d'atteindre le cas de base.
Sans cela, le programme ne termine pas.

Exercice 1 Toutes les méthodes suivantes doivent être récursives, il n'y a aucune boucle. N'oubliez pas le cas de base (cas d'arrêt) pour chaque méthode.

1. Quelle est la définition récursive de la fonction $S(n) = \Sigma_{i=1}^{n} i^2$? Écrivez une méthode récursive calculant $S(n)$ pour n donné en argument.
2. Étant donné deux entiers a et b, on sait que :
   – $\text{pgcd}(a, 0) = a$,
   – si $a < b$ alors $\text{pgcd}(a, b) = \text{pgcd}(b, a)$,
   – si $a = b.q + r$ alors $\text{pgcd}(a, b) = \text{pgcd}(b, r)$.
   En utilisant cette propriété, écrivez une fonction récursive `pgcd(a,b)` qui calcule le pgcd de deux entiers `a` et `b`.
3. Reprenez l'exercice 3 du TD4 (recherche d'un élément dans un tableau) en utilisant des méthodes récursives :
   (a) Écrivez une méthode qui retourne la position d'un entier `n` dans un tableau `tab`. Utilisez une méthode auxiliaire récursive `int rechercheAux(int[] tab, int n, int j)` qui retourne la position de `n` dans le sous-tableau de `tab` situé entre les indices `0` et `j`. Elle retourne `-1` si l'entier n'y est pas.
   (b) Écrivez une méthode de recherche dichotomique d'un entier `n` dans un tableau trié `tab`. Utilisez une méthode auxiliaire récursive
   `int rechercheDichoAux(int[] tab, int n, int i, int j)`
   qui retourne la position de `n` dans le sous-tableau de `tab` situé entre les indices `i` et `j`. Elle retourne `-1` si l'entier n'y est pas.

Exercice 2 (Tours de Hanoi)

Il s'agit d'un jeu, consistant à déplacer des cylindres empilés. Les règles sont les suivantes :
– Il y a exactement trois piles, que nous appellerons `1`, `2` et `3`.
– Dans chaque pile, les cylindres sont empilés du plus grand, en bas, au plus petit, au sommet. C'est vrai au début, et ça doit le rester à chaque étape.
– Au début, tous les cylindres sont empilés sur la pile `1`.
– On ne peut déplacer qu'un cylindre à la fois, du sommet d'une pile vers celui d'une autre pile.
– À la fin, les piles `1` et `2` sont vides, et tous les cyclindres ont été déplacés sur la pile `3`.

Le but de cet exercice est d'écrire une méthode affichant la solution pour déplacer une tour de `k` cylindres de la pile initiale `1` à la pile finale `3` (pour un entier `k` quelconque). Vous écrirez les solutions sous la forme : `i1->j1; i2->j2;`..., où `i->j` signifie : je déplace le cylindre du sommet de la pile `i` vers le sommet de la pile `j`. Par exemple, pour une tour avec `1` cylindre, la solution est `1->3`; et pour une tour avec `2` cylindres, la solution est `1->2; 1->3; 2->3;`.

1. Quelle est la solution pour une tour avec 3 cylindres ?
2. Connaissant la solution pour `k` cylindres, comment peut-on trouver celle pour `k+1` cylindres ? Faites un schéma.
3. En vous appuyant sur la question 2, écrivez une méthode récursive prenant en entrée un entier `k` et le nom de trois piles `a`, `b` et `c`, et affichant la solution pour transférer `k` cylindres de la pile `a` vers la pile `c`. Par exemple `hanoi(2, "1", "2", "3")` affichera `1->2; 1->3; 2->3;`. N'oubliez pas le cas de base.

Exercice 3 (Tri fusion)

Pour trier un tableau de taille n, on peut commencer par trier les $n/2$ premières cases, puis trier les $n/2$ dernières, pour enfin fusionner ces deux sous-tableaux triés.

1. Écrivez une méthode `int[] copie(int[] tab, int i, int j)`, qui renvoie le sous-tableau de `tab` situé entre les indices `i` et `j`.
2. Écrivez une méthode `void fusion(int[] tab, int i, int j)`. Cette méthode prend en argument le tableau `tab` dont les sous-tableaux situés entre les indices `i` et `(i+j)/2` d'une part, et entre les indices `(i+j)/2+1` et `j` d'autre part, sont triés. Elle modifie `tab` en fusionnant ces deux sous-tableaux pour que le sous-tableau de `tab` situé entre les indices `i` et `j` soit trié. Vous pourrez commencer par copier le premier sous-tableau dans un autre tableau par la méthode précédente.
3. Écrivez une méthode récursive auxiliaire `void trifusionAux(int[] tab, int i, int j)` qui tri le sous-tableau de `tab` situé entre les indices `i` et `j`. Finalement, écrivez une méthode `void trifusion(int[] tab)` qui tri le tableau `tab` en appliquant le tri fusion.