TP4

Algorithmes dichotomiques

I Recherche d'un élément dans une liste triée

Nous avons déjà vu un algorithme permettant de rechercher une valeur dans une liste.
Le principe consiste à parcourir chaque élément de la liste et de les comparer avec l'élément recherché. Cet algorithme possède un coût en temps proportionnel à n (où n désigne la longueur de la liste) car n'implique qu'une seule boucle `for` de l'ordre de n itération. On dit que cet algorithme a un coût linéaire ou une complexité en $O(n)$.
Nous allons mettre en place ici un algorithme plus efficace dans le cas où la liste est triée.

Principe de la recherche dichotomique

Supposons que la liste soit triée dans l'ordre croissant.Pour rechercher la valeur `v`, on la compare avec la valeur située au milieu de la liste :

- Si `v` est égale à la valeur du milieu de la liste, notée `liste[m]`, la valeur `v` est trouvée et on renvoie sa position
- Sinon si `v < liste[m]`, on recommence la recherche dans la première moitié de la liste.
- Sinon, on recommence la recherche dans la seconde moitié de la liste.

Exercice 1

La fonction `recherche_dichotomique(liste, v)` où `liste` est supposée triée par ordre croissant retournant la position de `v` si elle présente et la valeur `None` sinon.

1. Combien de valeurs sont examinées lors d'un appel à `recherche_dichotomique([0, 1, 1, 2, 3, 5, 8, 13, 21], 7)`
2. Donner un exemple d'exécution de `recherche_dichotomique` où le nombre de valeurs examinées est exactement 4.
3. Quel est le nombre maximal de valeurs à examiner pour une liste de 100 valeurs, 1000 valeurs, n valeurs.
4. Écrire cette fonction.
   Indication : On utilisera une boucle `while` ainsi que deux variables `g` et `d` contenant les deux indices entre lesquels il est encore possible de trouver `v` dans `liste`.

Exercice 2

Écrire un programme qui permette à l'ordinateur de jouer à *Devine un nombre* contre l'utilisateur. L'utilisateur choisit un nombre entre 0 et 100 et l'ordinateur doit le trouver le plus efficacement possible. A chaque proposition faite par l'ordinateur, l'utilisateur doit donner une réponse sous la forme d'une chaîne de caractère parmi `"plus grand"`, `"plus petit"` ou `"bravo"`.

Exercice 3

Ecrire un programme qui triche au jeu précédent : ce programme plutôt que de fixer un nombre mystère, devra faire durer la partie aussi longtemps que possible mais sans jamais donner un indice qui contredirait les indices précédents.

Exercice 4

Ecrire une fonction insertion qui prend en paramètres une liste de nombres triée et un nombre x et insère le nombre x dans la liste de manière à ce que la liste restre triée. La fonction utilise un algorithme dichotomique pour trouver la bonne place et ne renvoie rien.

II Exponentiation rapide

L'objectif de cette partie est de calculer efficacement la puissance entière d'un flottant uniquement à l'aide de multiplications, sans utiliser l'opérateur préexistant `**`.

1. Écrire une fonction `puissance(x, k)` prenant en argument un nombre réel x (un flottant), un entier positif k et renvoyant la valeur x^k au moyen de k multiplications successives.

2. Compléter la fonction précédente en incluant le cas ou k est négatif.
   On pourra utiliser l'instruction `assert x != 0` dans ce cas pour vérifier que x est non nul et interrompre la fonction avec une erreur sinon.
3. Quel est le coût de la fonction puissance ?
4. Déterminer une méthode permettant de déterminer x^{16} à l'aide de seulement 4 multiplications. En combien de multiplications peut-on obtenir x^{15} ?
5. La méthode repose donc sur les remarques suivantes :
   - Si n est pair, $x^n = (x^2)^{n//2}$
   - Si n est impair, $x^n = x(x^2)^{n//2}$
   - puis on reproduit ce raisonnement pour calculer $y^{n//2}$ où $y = x^2$.

   En déduire un algorithme puis écrire la fonction `puissance_rapide(x, n)` permettant d'appliquer cette méthode.
6. Décrire les valeurs successives des variables pour obtenir le résultat de `puissance_rapide(3, 7)`
7. Écrire une fonction `nombre_mult(x, n)` renvoyant le nombre de multiplications utilisées par cette méthode.
   En prenant $x = 1$, déterminer le plus petit entier k tel que le calcul de x^k par `puissance_rapide` nécessite au moins 40 multiplications.

III Diviser pour régner

L'algorithme de recherche d'une valeur dans une liste triée et celui d'exponentiation rapide consistent à diviser un problème en deux sous-problèmes. Ce principe est connu en informatique sous le nom de *diviser pour régner* et il est appliqué dans de nombreux algorithmes. On peut le décomposer en 3 étapes :

- Diviser : découper un problème initial en sous-problèmes ;
- Régner : résoudre les sous-problèmes ;
- Combiner : calculer une solution au problème initial à partir des solutions des sous-problèmes

Nous allons l'appliquer à nouveau très prochainement, notamment dans de nouveaux algorithmes de tris (tri fusion, tri rapide).