Algorithmes et Pensée Computationnelle

Algorithmes de recherche

Le but de cette séance est de se familiariser avec les algorithmes de recherche. Dans la série d'exercices, nous manipulerons des listes et collections en Java et Python. Nous reviendrons sur la notion de récursivité et découvrirons les arbres de recherche. Au terme de cette séance, l'étudiant sera en mesure d'effectuer des recherches de façon efficiente sur un ensemble de données.
Le code présenté dans les énoncés se trouve sur Moodle, dans le dossier **Ressources**.

1 Introduction et architecture des ordinateurs

1.1 Introduction/Résumé

1.2 Exercices

Question 1: (*Durée*) Exercice 1

Conseil

Solution

Langage utilisé :

2 Logiciels système

2.1 Introduction/Résumé

2.2 Exercices

Question 2: (*Durée*) Exercice 1

Conseil

Solution

Langage utilisé :

3 Programmation de base

3.1 Introduction/Résumé

3.2 Exercices

Question 3: (*Durée 5*)

1. Convertir $52_{(10)}$ en base 2 sur 8 bits.
2. Convertir $100_{(10)}$ en base 2 sur 8 bits.
3. Calculer en base 2 la soustraction de la question de l'exercice 1.2 par 1.1.

4. Déterminer au complément à deux l'opposé (multiplication par -1 en base 10) de l'exercice 3.3.

Conseil

— Se référer aux techniques apprises dans la série 1 et la série 3
— Faire un tableau des puissances de 2 sur 8 bits.

Solution

— $52_{(10)} = 32_{(10)} + 16_{(10)} + 4_{(10)} = 00110100_{(2)}$
— $100_{(10)} = 64_{(10)} + 32_{(10)} + 4_{(10)} = 01100100_{(2)}$
— $01100100_{(2)}$ - $00110100_{(2)} = 0110000_{(2)}$
— **Complément à 1 :** not($0110000_{(2)}$) = $1001111_{(2)}$
— **Complément à 2 :** Complément à 1 + 1 = $1001111_{(2)} + 1_{(2)} = 1010000_{(2)}$

4 Itération et récursivité

4.1 Introduction/Résumé

4.2 Exercices

Question 4: (*Durée*) Exercice 1

Conseil

Solution

Langage utilisé :

5 Algorithmes et compléxité

5.1 Introduction/Résumé

5.2 Exercices

Question 5: (*15 minutes*) **Tri fusion (Merge Sort)**

1. Ecrire un fonction "merge" qui prend deux listes triées comme argument et retourne une liste fusionnée triée.
2. Quelle est le nombre d'opérations effectuées ? Déterminer ensuite la complexité de la fonction, en posant n = longueur de la liste fusionnée

Pour les tests utilisez les listes suivantes :
l1 [3,10,12] et **l2** [5,7,14,15]

Conseil

— Cette fonction est une des deux parties de l'algorithme de tri fusion
— Revenez à l'exercice 11 de la série 5.
— Revenez à la visualisation de l'algorithme dans les diapositives 83 à 111 pour comprendre comment marche concrètement le tri fusion.

Solution

Python :

```python
def merge(partie_gauche, partie_droite):
    # créer la liste qui sera retournée à la fin
    liste_fusionnee = []

    # définir un compteur pour l'index de la liste de gauche
    compteur_gauche = 0
    # pareil pour la liste de droite
    compteur_droite = 0

    longueur_gauche = len(partie_gauche)
    longueur_droite = len(partie_droite)

    # continuer jusqu'à ce que l'un des index (ou les deux) atteigne l'une des longueurs (ou les deux)
    while compteur_gauche < longueur_gauche and compteur_droite < longueur_droite:
        # comparer les éléments actuels, ajouter le plus petit à la liste fusionnée
        # et augmenter le compteur de cette liste
        if partie_gauche[compteur_gauche] < partie_droite[compteur_droite]:
            liste_fusionnee.append(partie_gauche[compteur_gauche])
            compteur_gauche += 1
        else:
            liste_fusionnee.append(partie_droite[compteur_droite])
            compteur_droite += 1

    # s'il y a encore des éléments dans les listes, il faut les ajouter à la liste fusionnée
    liste_fusionnee += partie_gauche[compteur_gauche:longueur_gauche]
    liste_fusionnee += partie_droite[compteur_droite:longueur_droite]

    return liste_fusionnee  # retourner la liste fusionnée



if __name__ == "__main__":
    l1 = [3, 10, 12]
    l2 = [5, 7, 14, 15]
    print(merge(l1,l2))
```

Java :

```java
import com.sun.tools.corba.se.idl.constExpr.Not;

public class question_conso1_mergesort {
    // Fusionne 2 sous-listes de arr[].
    // Première sous-liste est arr[l..m]
    // Deuxième sous-liste est arr[m+1..r]
    public static int[] merge(int[] arr1, int[] arr2) {
        // Trouver la taille des deux sous-listes à fusionner
        int n1 = arr1.length;
        int n2 = arr2.length;

        /* Créer des listes temporaires */
        int L[] = new int[n1];
        int R[] = new int[n2];
        int F[] = new int[n1 + n2];

        /*Copier les données dans les sous-listes temporaires */
        for (int i = 0; i < n1; ++i) {
            L[i] = arr1[i];

        }
        for (int j = 0; j < n2; ++j) {
            R[j] = arr2[j];
        }
        /* Fusionner les sous-listes temporaires */
        // Indexes initiaux de la première et seconde sous-liste
        int i = 0, j = 0;

        // Index initial de la sous-liste fusionnée
        int k = 0;
        // Boucle qui fusionne L et R de manière ordonnée
        while (i < n1 && j < n2) {

            if (L[i] <= R[j]) {
                F[k] = L[i];
                i++;
                ++k;
            } else {
```

Solution

— **En python il y a :** n = 3+ 4 = 7 itérations. Les opérations avant et après la boucle sont des opérations simples, donc la complexité ”pire cas” est calculé grâce à la boucle while et donc en fonction de la taille de la liste finale.

— **En java il y a :** n = 3 + 2 itérations effectuées dans la première boucle while.
Dans la deuxième boucle while il y a uniquement m = 2 itérations effectuées. Les boucles n'étant pas imbriquées les complexités s'additionnent.

— Donc la complexité est O (n + m) car cela va dépendre de la grandeur des listes rentrées en argument.

Question 6: (*Durée*) Exercice 1

Conseil

Solution

Langage utilisé :

6 Algorithmes de recherche

6.1 Introduction/Résumé

6.2 Exercices

Question 7: (*Durée*) Exercice 1

Conseil

Solution

Langage utilisé :