TD
Structures de données

Exercice 1

Soit une suite de clés dans une liste L de N entiers tous différents et appartenant à l'intervalle [1..N]. Par exemple pour N= 8, L = [5, 8, 2, 6, 3, 4, 1, 7]

1. Ecrire la fonction verifierListe qui vérifie que les éléments de la liste appartiennent à l'intervalle [1..N] et sont tous différents.
2. Evaluer en ordre de grandeur, le coût temporel de cette fonction

Exercice 2

L'évaluation d'expressions arithmétiques est essentielle dans le processus de compilation et d'interprétation des programmes. Lorsque des expressions infixées sont rencontrées, elles sont souvent converties en expressions postfixes (opérateurs suivant les opérandes) pour simplifier l'évaluation. Une expression arithmétique infixe est un texte composé d'opérateurs (+, -, * et /), d'opérandes (constantes entières et/ou réelles) et éventuellement des parenthèses (pour forcer des priorités). Dans ce cadre, on se propose d'explorer cette conversion et d'implémenter l'évaluation d'expression postfixe. Exemple d'expression arithmétique infixée : "3.14 + 5 * (2.718 - 8)" son équivalent postfixe est "3.14 5 2.718 8 - * +"

1. Implémenter une fonction tokenize(expression) qui prend une expression arithmétique infixée en entrée et renvoie une liste de tokens (opérandes et opérateurs) sans utiliser d'expressions régulières. Utiliser une boucle pour parcourir la chaîne de caractères et construire la liste de manière itérative. tokenize("3.14 + 5 * (2.718 - 8)") doit produire : ['3.14', '+', '5', '*', '(', '2.718', '-', '8', ')']
2. Implémenter une classe Pile avec des méthodes empiler, depiler, et estVide. Cette pile sera utilisée pour gérer les opérateurs lors de la conversion de l'infixe en postfixe.
3. Implémenter une fonction infixeToPostfixe(expression_infixe) qui prend une expression arithmétique infixée en entrée et renvoie l'expression postfixée correspondante sous la forme d'une liste en suivant les étapes suivantes :
   a. Initialiser une pile pour les opérateurs.
   b. Initialiser une liste de sortie postfixe
   c. Parcourir chaque token de la liste obtenue après la tokenization et distinguer les cas suivants :
   d. Si le token est un opérande (entier ou réel), ajouter le directement à la liste de sortie postfixe.
   e. Si le token est un opérateur, dépiler tous les opérateurs de la pile ayant une priorité supérieure ou égale à celui-ci, et ajouter les dans l'ordre à la liste de sortie postfixe. Empiler le nouvel opérateur sur la pile.

f. Cas Parenthèse : Pour une parenthèse ouvrante, empiler la directement. Pour une parenthèse fermante, dépiler les opérateurs jusqu'à trouver la parenthèse ouvrante correspondante et ajouter les à la liste de sortie postfixe.
g. Une fois le parcours des tokens terminé, dépiler tous les opérateurs restants de la pile et ajouter les à la liste de sortie postfixe.

4. Implémenter une fonction evaluaerPostfixe (liste_postfixe) qui prend une liste expression postfixe en entrée et qui renvoie le résultat de l'évaluation.
5. Donner un code de test pour tester votre implémentation en utilisant une expression arithmétique infixée avec des réels.

Exercice 3

Implement the stack ADT: push(element), pop(), and size() using the Queue ADT: enqueue(element), dequeue(), and size(). Derive complexities for each method in your stack ADT.

Exercice 4

Analyze the complexity (in big-Oh terms) of the following selection-sort routine. The routine picks out the largest element in the current list and exchanges it with the last element. It continues until the list has only one element:

```
for (i = n-1; i > 0; i--) {
    MaxPosition = i;
    for (j = 0; j < i; j++) {
        if (a[j] > a[MaxPosition])
            MaxPosition = j;
    }
    exchange(i, MaxPosition);
}
```

Exercice 5

1. Give a Theta(n) algorithm for computing a^n, given a and n.
2. Give a Theta(log n) algorithm for computing a^n, given that n is a power of 2.

Exercice 6

You are given a data structure stack with the following ADT:

- push(element)
- pop()
- size()

Each of these operations runs in O(1) time.

Use pseudocode to implement the following Queue ADT using the stack ADT only. What is the complexity of each of your methods?

1. enqueue(element)
2. dequeue()
3. size()

Note: you do not have to implement a linked list or an array. Instead, you must use only the three methods in the stack ADT.