IFT2015-A25: TP1

September 23, 2025

Instructions générales

- **Date de remise : Lundi 6 octobre à 23h59.**
- Ce devoir est à compléter seul(e) ou en équipe de deux.
- Aucun plagiat ne sera accepté.
- Il est possible que des précisions ou modifications aux consignes soient envoyées par courriel et affichées sur Studium.
- Chaque jour de retard dans la remise entraînera une pénalité de 5 points.
- Tout le code doit être écrit en Java.
- Vous devez soumettre les fichiers Java requis (`Node.java`, `Stack.java`, `Q4.java`) ainsi qu'**un PDF** contenant les réponses écrites.
- Le PDF peut inclure des photos/scans de réponses manuscrites. **Assurez-vous que ce soit lisible!**
- **Si vous travaillez en équipe :** une personne soumet les fichiers nécessaires et l'**autre** soumet seulement un fichier **.txt** avec le nom de son/sa partenaire.
- L'évaluation du code pourra être automatisée; assurez-vous d'avoir **les mêmes signatures de fonctions**.
- Vous pouvez créer autant de fonctions auxiliaires que vous le souhaitez.
- **N'oubliez pas d'ajouter des docstrings et des commentaires à vos implémentations. Le code sans documentation sera pénalisé.**
- Pour toute question concernant le devoir 1, veuillez les poster sur le forum "TP1 Q&A" sur Studium.
- Bonne chance !

Problème 1 (5pts)

Pour chacun des algorithmes ci-dessous, déterminez la complexité temporelle et spatiale de son exécution (Big O).

(a)
```
int i, j, k = 0;
for (i = n / 8; i <= n; i++) {
    for (j = 1; j <= n; j = j * 2) {
        k = k + n / 4;
    }
}
```

(b)
```
int i, j, k, value = 0;
for (i = 1; i < N; i++) {
    for (j = 1; j < N; j = j * 2) {
        value += j;
    }
    for (k = 1; k < N * N; k++) {
        value += k;
    }
}
return value;
```

(c)
```
for (int i = 1; i < n; i++) {
    i *= k;
    for (int j = 1; j < n; j++) {
        j *= k;
    }
}
```

(d)
```
for(int i = 0; i < N; i++) {
    for(int j = 0; j < N / 2; j++) {
        ... // certaines opérations en O(N)
    }
}
```

(e)
```
int i, j, k, N = 0;
for(i = 0; i < n; i++) {
    for(j = 0; j < n^(1/2); j++) {
        for (k = 0; k < 100000000; k++) {
            N += k;
        }
    }
}
```

Problème 2 (5pts)

Dans cette partie, vous implémentez une pile simple, que vous utiliserez dans le problème 4. Il est essentiel d'ajouter des commentaires clairs à votre code. Complétez les fonctions suivantes dans le fichier `Stack.java`.

Signature de la fonction	**Description**
`public void push(String s)`	Ajoute un String à la pile.
`public String peek()`	Retourne le dernier String dans la pile.
`public String pop()`	Retire et retourne le dernier String dans la pile.
`public int size()`	Retourne la taille de la pile.
`public boolean isEmpty()`	Vérifie si la pile est vide.

Problème 3 (5pts)

Dans cette partie, vous implémentez une liste chaînée simple, que vous utiliserez dans le prochain problème. Complétez les fonctions suivantes dans le fichier `Node.java`. Les fonctions décrites comme "itératives" doivent être réalisées avec au moins une boucle (`for` ou `while`).

Signature de la fonction	**Description**
`public Node(String s)`	Crée un nœud de valeur chaîne s (0,5)
`public Node(String s, Node next)`	Crée un nœud de valeur s et relie au nœud suivant (0,5)
`public void add_iter(String s)`	Ajoute un élément à la fin de la liste, itératif (1)
`public void add_rec(String s)`	Ajoute un élément à la fin de la liste, récursif (1)
`public int length_iter()`	Retourne la longueur de la liste, itératif (1)
`public int length_rec()`	Retourne la longueur de la liste, récursif (1)

Problème 4 (5pts)

Lors de la journée d'accueil, Kevin veut créer une atmosphère conviviale avec un jeu. Il y a N personnes alignées, représentées par une liste chaînée simple (utilisez votre implémentation du Problème 3). À chaque tour, Kevin enlève une personne de la file et lui demande de se présenter à l'avant. Votre tâche est d'aider Kevin à suivre qui a été retiré et à conserver cette information dans une pile (utilisez votre implémentation du Problème 2). Si le retrait ne peut pas être effectué, inscrivez "**???**" dans la pile. Pour toutes les questions qui suivent, **l'index commence à 1**. Vous pouvez ajouter des variables et des fonctions utilitaires, mais vous **ne pouvez pas** utiliser les méthodes intégrées de la classe List de Java. Commencez avec le code fourni dans `Q4.java`.

1. Complétez la fonction `rm_mid(Node head)` qui retire la personne à la position centrale $\lceil N/2 \rceil$, place son nom dans la pile, et retourne le début de la liste.

2. Complétez la fonction **itérative** `rm_k_end_iter(Node head, int k)` qui, étant donné la tête de la file et un entier K, retire la Kième personne en comptant à partir de la **fin** de la file, place son nom dans la pile, et retourne le début de la liste.

3. (2pts) Complétez la fonction **récursive** `rm_k_end_rec(Node head, int k)` qui, étant donné la tête de la file et un entier K, retire la Kième personne en comptant à partir de la **fin** de la file, place son nom dans la pile, et retourne le début de la liste.

4. Complétez la fonction `rm_after_k(Node head, int k)` qui retire la personne tout après la Kième personne en comptant à partir du début, place son nom dans la pile, et retourne le début de la liste.

Exemples:

```
--- start ---
head: Alice->Bob->Claude->Duke->Elaine->
Stack is empty.

--- after rm_mid(head) ---
head: Alice->Bob->Duke->Elaine->
Stack:
---------
Claude
---------

--- after rm_k_end_iter(head, 4) ---
head: Bob->Duke->Elaine->
Stack:
---------
Alice
---------
Claude
---------

--- after rm_k_end_rec(head, 4) ---
head: Bob->Duke->Elaine->
Stack:
---------
???
---------
Alice
---------
Claude
---------

--- after rm_after_k(head, 1) ---
head: Bob->Elaine->
```

```
Stack:
---------
Duke
---------
???
---------
Alice
---------
Claude
---------
```