TP 4 Graphes et langages

1 Une FILE « fabrication maison » vs une FILE `ArrayList`

L'implémentation du parcours en largeur repose sur une FILE Il est possible de programmer de manière générique avec de telles structures de données. La fiche Wikipedia affirme que la « programmation générique » permet de concevoir

> *des algorithmes identiques opérant sur des données de types différents. On définit de cette façon des procédures ou des types entiers génériques. On pourrait ainsi programmer une pile, ou une procédure qui prend l'élément supérieur de la pile, indépendamment du type de données contenues.*

Le code ci-dessous permet d'enfiler 300000 fois *la valeur entière 5* dans une file d'entiers. Ensuite, ces entiers sont récupérés (défilés) un par un pour les additionner dans la boucle `while`.

```
int n = 300000;
ArrayList<Integer> file= new ArrayList();
for(int i=0;i<n;i++)
    file.add(5);
int sum = 0;
while(!file.isEmpty()){
    sum = sum + file.get(0);
    file.remove(0);
}
System.out.println("sum="+sum);
```

Exercice 1 Faites varier le nombre d'éléments n pour trouver la plus petite valeur de n pour laquelle le programme a besoin de plus de 3 secondes. À partir de la valeur trouvée, noter les temps d'exécution pour $2n$, $3n$, $4n$ et $5n$.

- Pouvez vous observer si la croissance du temps de calcul est linéaire, quadratique, exponentielle, etc. ?
- Utiliser un programme de gestion de tâches du système d'exploitation et noter la mémoire demandée pour $5n$;

Exercice 2 Modifier le code ci-dessus et utiliser la structure `LinkedList` pour faire les même opérations. Il va falloir remplacer `file.get(0)` avec `file.getFirst()` et `file.remove(0)` avec `file.pop()` Réaliser la même consigne de l'exercice précédent sur la nouvelle structure de données, c. à. d., trouver la nouvelle valeur de n qui demande plus de 3 secondes et noter les temps de calcul pour $2n,\ldots,5n$. Vous obtenez une croissance toujours quadratique ?

Exercice 3 Modifier le code ci-dessus pour ne plus utiliser de classe standard de file. On utilise la structure de FILE rappelée ci-dessous. Trouver la nouvelle valeur de n qui demande plus de 3 secondes et noter les temps de calcul pour les mêmes valeurs de $n, 2n, 3n, 4n, 5n$.

```
static int[] q ;
static int premier=0;
static int dernier=-1;
static void initFile(int taille){
    q = new int[taille];
}
static boolean fileVide(){
    return(premier>dernier);
}
static void fileDefiler(){
    premier++;
}
static void fileEnfiler(int x){
    dernier++;
    q[dernier]=x;
}
static int fileTete(){
    return q[premier];
}
```

Exercice 4 Rédiger un document texte (fichier `.txt`) de maximum 100 lignes avec les conclusions de cette comparaison. Quel est le classement de vitesse de ces trois structures. Quelle méthode demande le plus de mémoire ?

Télécharger le fichier `cedric.cnam.fr/~porumbed/graphes/Parcours.java`

Vous allez observer que ce fichier contient une matrice d'adjacence écrite « en dur » dans le code. Cette matrice d'adjacence de taille 9×9 représente un graphe ; l'instance est donc codée par cette matrice d'adjacence

Exercice 5 Modifier le fichier téléchargé `Parcours.java` en ajoutant une fonction d'en-tête

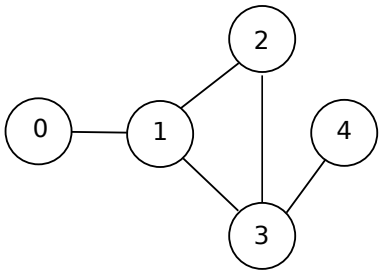
```
public static int noVoisins(int sommet)
```

qui renvoie le degré du sommet `sommet` passé en paramètre.

Exercice 6 Ajouter au programme précédent une fonction `noMaxVoisins()` qui renvoie le degré maximum (c. à. d, le nombre max de voisins).

Exercice 7 Ajouter une fonction `afficherSommetsDenses(...)` qui permet d'afficher tous les sommets qui ont un degré supérieur au nombre de lettres de votre nom.

Il est important de se (re-)familiariser avec les parcours pour continuer : n'hésitez pas à consulter la vidéo déposée à `cedric.cnam.fr/~porumbed/graphes/`



Exercice 8 Dérouller à la main l'algorithme à droite sur le graphe ci-dessus avec `sommetDépart`=0.

1. `enfiler(sommetDépart)`
2. `sommetsOuverts`← init. tableau de n booléens : mettre `sommetDépart` à `true` et les autres à `false`
3. **while**(`!fileVide()`)
   (a) x =`fileTête()`
   (b) `println("Vu ":"+`x`)`
   (c) `fileDefiler();`
   (d) **pour chaque** voisin non ouverts v de x :
      – `enfiler(`v`);`
      – `sommetsOuverts[`v`]` ← `true`
   (e) Afficher les éléments dans la file

Exercice 9 Modifier le fichier téléchargé `Parcours.java` pour réaliser un *parcours en largeur*. Le sommet de départ doit être saisi par l'utilisateur. Il faut afficher l'ordre de parcours. N'hésitez pas à utiliser le code à droite.

Ajouter l'analyse de l'exercice 4 comme commentaire au début du fichier `Parcours.java`. Déposer le fichier `Parcours.java` qui résulte après avoir tout fini à :

`cedric.cnam.fr/~porumbed/graphes/tp4/`

ATTENTION : le non respect de cette consigne est pénalisé de 10% si c'est complètement injustifié.