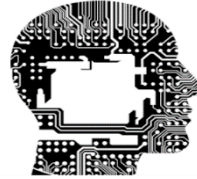




Université Constantine 2
جامعة قسنطينة 2



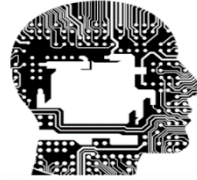
Foundation of Artificial Intelligence

TP 01

Dr. NECIBI Khaled
Faculté des nouvelles technologies
Khaled.necibi@univ-constantine2.dz



Université Constantine 2
جامعة قسنطينة 2



Foundation of Artificial Intelligence

- TP Résolution de problèmes par recherche non informée -

Dr. NECIBI Khaled

Faculté des nouvelles technologies

Khaled.necibi@univ-constantine2.dz

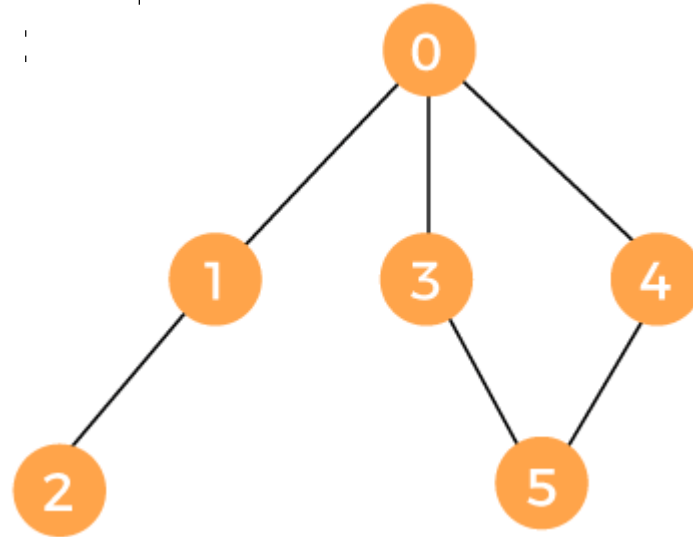
Etudiants concernés

| Faculté/Institut | Département | Niveau | Spécialité |
|------------------------|-------------|----------|------------|
| Nouvelles technologies | IFA | Master 1 | SDIA |

Résolution de problème par recherche non informée

- Exercice 01

- On considère un espace de recherche représenté par le graphe suivant :

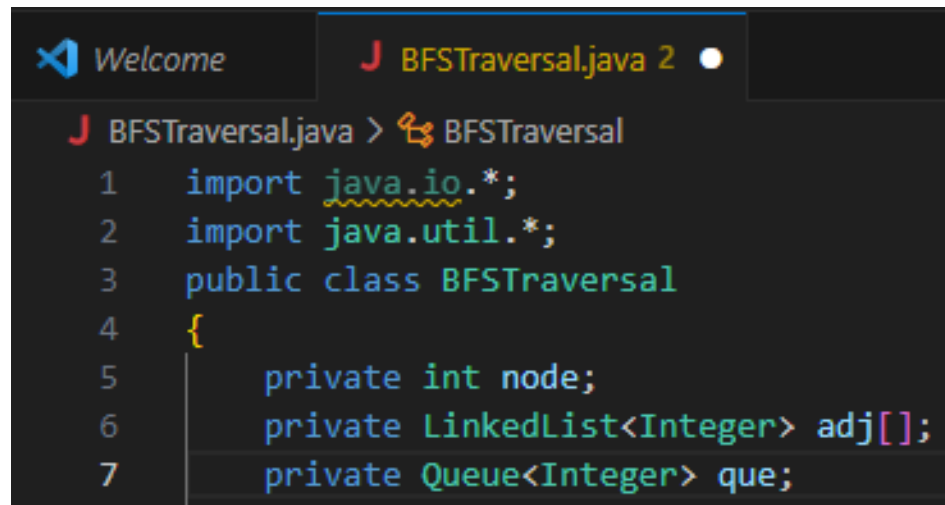


- On considère que l'état initial est représenté par le nœud 0; l'état final est représenté par le nœud 5;
- Question : Ecrire un programme Java qui donne l'ordre du parcours des nœuds pour l'algorithme BFS.

Résolution de problème par recherche non informée

● Exercice 01 : Solution

- Tout d'abord on commence par la création de la classe BFSTraversal
- On aura besoin de déclarer trois variables essentielles pour maintenir notre solution sous forme d'ordre de parcours de nœud :
 - node: représente le nombre total de nœuds dans le graph
 - adj: représente une liste d'adjacence
 - que: une file d'attente qui doit être maintenue au fur et à mesure



```

Welcome
J BFSTraversal.java 2
J BFSTraversal.java > BFSTraversal
1  import java.io.*;
2  import java.util.*;
3  public class BFSTraversal
4  {
5      private int node;
6      private LinkedList<Integer> adj[];
7      private Queue<Integer> que;
```

Résolution de problème par recherche non informée

● Exercice 01 : Solution

- Ensuite; le constructeur de la classe doit être implémenté en initialisant l'espace de recherche par le nœud initial

```
BFSTraversal(int v)
{
    node = v;
    adj = new LinkedList[node];
    for (int i=0; i<v; i++)
    {
        adj[i] = new LinkedList<>();
    }
    que = new LinkedList<Integer>();
}
```

- Après, il faut ajouter un arc en utilisant une méthode « insertEdge » à la liste d'adjacence. À noter que les arcs dans cet exercice sont bidirectionnels

```
void insertEdge(int v,int w)
{
    adj[v].add(w);
}
```

● Exercice 01 : Solution

- Maintenant, il ne reste qu'à définir le comportement de l'ordre de parcourir pour atteindre le nœud de destination. Pour cela, une méthode « BFS » doit être déclarée dont les fonctionnalités sont :
 - Initialiser un array booléen pour maintenir les données
 - Ajouter le nœud qui correspond à l'état initial au début de la file d'attente « que »
 - Traitement répétitif; si le nœud courant ne correspond pas à l'état final; consistant à :
 - Retirer le premier « node » de la file d'attente « que »
 - Afficher le premier élément de la file d'attente « que »
 - Itérer sur la liste d'adjacence « adj » ensuite empiler tous les nœuds voisins à la fin de la file d'attente « que »
 - Insérer dans la file d'attente « que » que les nœuds qui ne sont pas encore explorés

● Exercice 01 : Solution

- Initialiser un array booléen pour maintenir les données

```
void BFS(int n)
{
    boolean nodes[] = new boolean[node];
    int a = 0;
    nodes[n]=true;
```

- Ajouter le nœud qui correspond à l'état initial au début de la file d'attente « que »

```
que.add(n);
```

- Traitement répétitif; si le nœud courant ne correspond pas à l'état final; consistant à :
 - Retirer le premier « node » de la file d'attente « que »

```
while (que.size() != 0)
{
    n = que.poll();
```

● Exercice 01 : Solution

- Traitement répétitif; si le nœud courant ne correspond pas à l'état final; consistant à :
 - Retirer le premier « node » de la file d'attente « que »
 - Afficher le premier élément de la file d'attente « que »

```
while (que.size() != 0)
{
    n = que.poll();
    System.out.print(n+" ");
}
```

- Itérer sur la liste d'adjacence « adj » ensuite empiler tous les nœuds voisins à la fin de la file d'attente « que »
- Insérer dans la file d'attente « que » que les nœuds qui ne sont pas encore explorés

```
for (int i = 0; i < adj[n].size(); i++)
{
    a = adj[n].get(i);
    if (!nodes[a])
    {
        nodes[a] = true;
        que.add(a);
    }
}
```


Résolution de problème par recherche non informée

● Exercice 01 : Solution

- Définir à la fin la méthode principale « main » tout en initialisant notre graphe de recherche comme indiqué dans l'exercice

```
public static void main(String args[])
{
    BFSTraversal graph = new BFSTraversal(v:6);
    graph.insertEdge(v:0, w:1);
    graph.insertEdge(v:0, w:3);
    graph.insertEdge(v:0, w:4);
    graph.insertEdge(v:4, w:5);
    graph.insertEdge(v:3, w:5);
    graph.insertEdge(v:1, w:2);
    graph.insertEdge(v:1, w:0);
    graph.insertEdge(v:2, w:1);
    graph.insertEdge(v:4, w:1);
    graph.insertEdge(v:3, w:1);
    graph.insertEdge(v:5, w:4);
    graph.insertEdge(v:5, w:3);
    System.out.println(x:"La recherche en largeur d'abord pour le graphe est:");
    graph.BFS(n:0);
}
```

- Le résultat d'exécution du programme BFS sera:

```
PS C:\Users\KHALED\Strategies> c:; cd 'c:\Users\KHALED\Strategies'; & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\KHALED\AppData\Roaming\Code\User\workspaceStorage\107beaf93822a25da2787408abe466bf\redhat.java\jdt_ws\Strategies_743eeb7a\bin' 'BFSTraversal'
La recherche en largeur d'abord pour le graphe est:
0 1 3 4 2 5
```