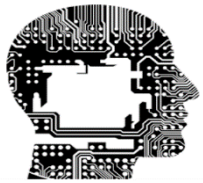
Université Constantine 2
جامعة قسنطينة 2



Foundation of Artificial Intelligence

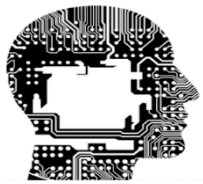
TP 01 (Suite)

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

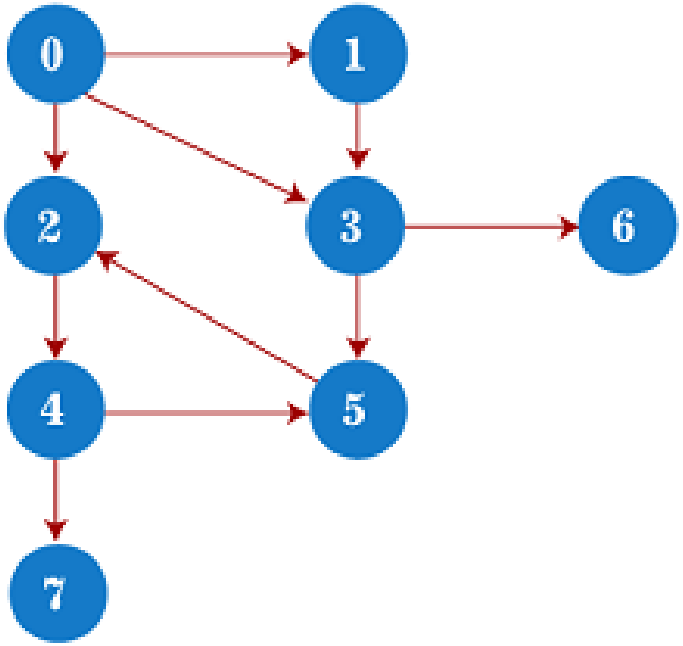
Khaled.necibi@univ-constantine2.dz

Etudiants concernés

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

Résolution de problème par recherche non informée

- Exercice 02
  - On considère un espace de recherche représenté par le graphe orienté suivant



  - Question : écrire un programme Java qui donne l'ordre de parcours des nœuds pour l'algorithme DFS, sachant que l'état initial est 0; l'état final est 6

- Exercice 02 : Solution
  - Représentation de la liste d'adjacence

```
class DFSTraversal {
  private LinkedList<Integer> adj[];
  private boolean visited[];
```

  - Création du graphe; le paramètre V représente le nombre de sommets dans le graphe

```
DFSTraversal(int V)
{
  adj = new LinkedList[V];
  visited = new boolean[V];

  for (int i = 0; i < V; i++)
    adj[i] = new LinkedList<Integer>();
}
```

- Exercice 02 : Solution
  - Ajouter un arc dans un graphe

```
void insertEdge(int src, int dest) {
  adj[src].add(dest);
}
```

  - Marquer le nœud courant comme visité et itérer sur les autres nœuds

```
void DFS(int vertex) {
  visited[vertex] = true;
  System.out.print(vertex + " ");

  Iterator<Integer> it = adj[vertex].listIterator();
  while (it.hasNext()) {
    int n = it.next();
    if (!visited[n])
      DFS(n);
  }
}
```

- Exercice 02 : Solution
  - Définir à la fin la méthode principale « main » tout en initialisant notre graphe de recherche comme indiqué dans l'exercice

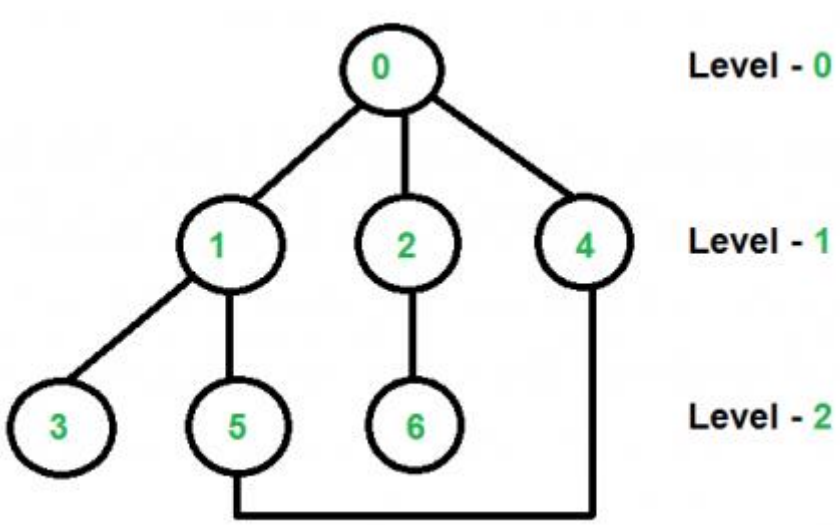
```
Run | Debug
public static void main(String args[]) {
    DFSTraversal graph = new DFSTraversal(V:8);

        graph.insertEdge(src:0, dest:1);
        graph.insertEdge(src:0, dest:2);
        graph.insertEdge(src:0, dest:3);
        graph.insertEdge(src:1, dest:3);
        graph.insertEdge(src:2, dest:4);
        graph.insertEdge(src:3, dest:5);
        graph.insertEdge(src:3, dest:6);
        graph.insertEdge(src:4, dest:7);
        graph.insertEdge(src:4, dest:5);
        graph.insertEdge(src:5, dest:2);

        System.out.println(x:"La recherche en profindeur dabird pour le graphe est:");
        graph.DFS(vertex:0);
}
```

Résolution de problème par recherche non informée

- Exercice 02 : Solution
  - Le résultat d'exécution du programme DFS sera:

```
PS 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_743eeb7
a\bin' 'DFSTraversal'
La recherche en profindeur dabird pour le graphe est:
0 1 3 5 2 4 7 6
PS C:\Users\KHALED\Strategies>
```

- Exercice 03
  - On considère un espace de recherche représenté par le graphe orienté suivant



  - Question : écrire un programme Java pour implémenté l'algorithme IDS.
    - Le programme doit avoir trois entrés : l'état initial; l'état final et la valeur maximale de la profondeur.
    - Le programme doit retourner un message indiquant si l'état final est atteint ou non

Exercice 03 : Solution

- Commencer par créer une classe qui implémente un graphe en utilisant une liste d'adjacence avec un certain nombre de nœud V;

```
class Graph {
int V;

LinkedList<Integer> adj[];
```

- Créer ensuite une fonction récursive DFS; si la profondeur maximale est atteinte l'appel récursive s'arrêtera; sinon réitéré le même processus avec les nœuds adjacents

```
boolean DLS(int v, int target, int limit)
{
    if (v == target)
    return true;
    if (limit <= 0)
    return false;
    for (int i : adj[v])
    if (DLS(i, target, limit - 1))
        return true;
    return false;
}
```

- Exercice 03 : Solution
  - Créer une fonction pour l'initialisation du graphe

```
public Graph(int v)
{
    V = v;
    adj = new LinkedList[v];
    for (int i = 0; i < v; ++i)
    adj[i] = new LinkedList();
}
```

  - Créer ensuite une fonction pour ajouter un nouvel arc dans le graphe

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

- Exercice 03 : Solution
  - Créer ensuite une fonction avec trois paramètres : l'état initial, l'état final et la profondeur maximale pour vérifier si la solution appartient à la profondeur maximale imposée ou non

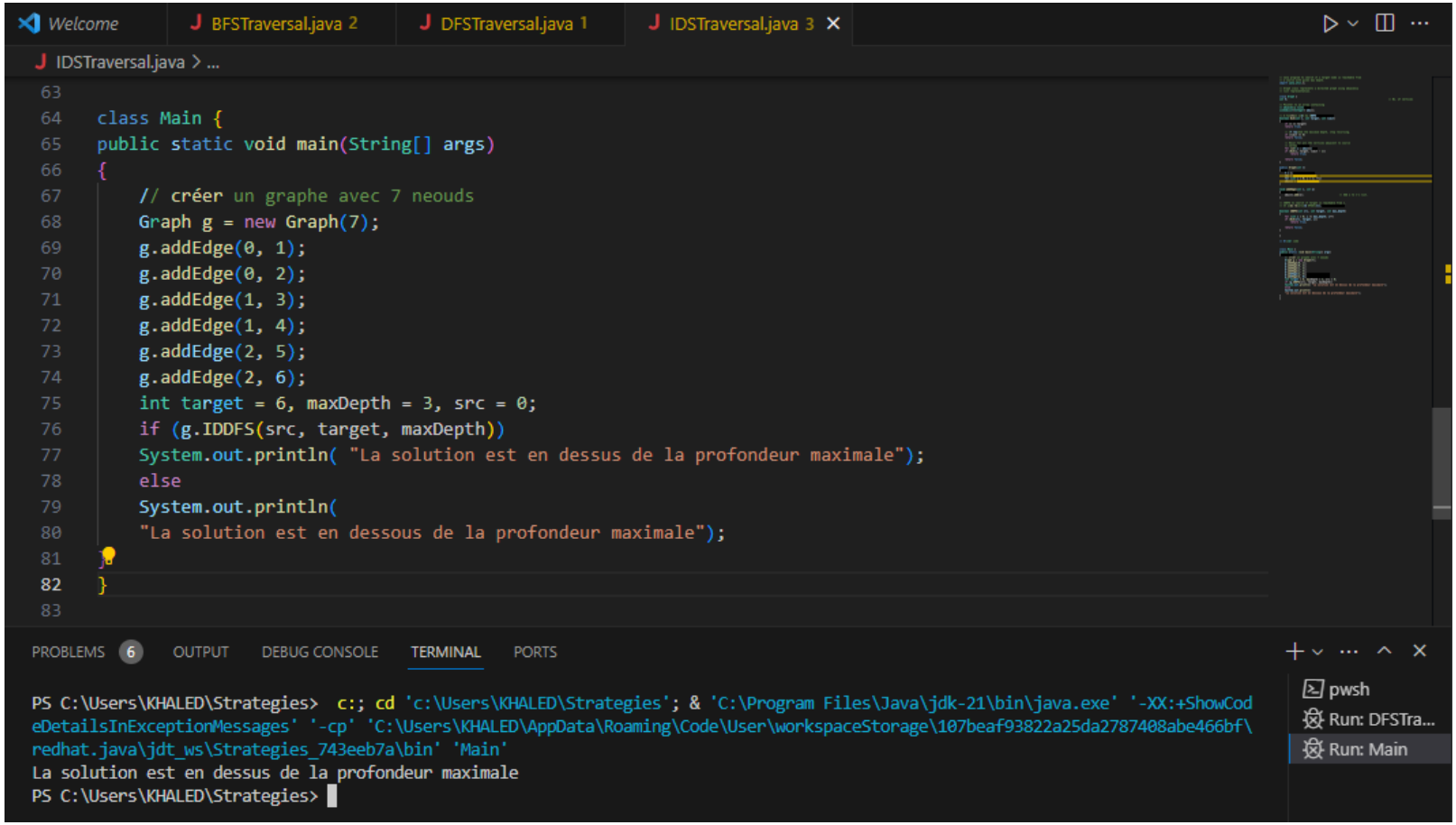
```
boolean IDDFS(int src, int target, int max_depth)
{
    for (int i = 0; i <= max_depth; i++)
    if (DLS(src, target, i))
        return true;

    return false;
}
```

Exercice 03 : Solution

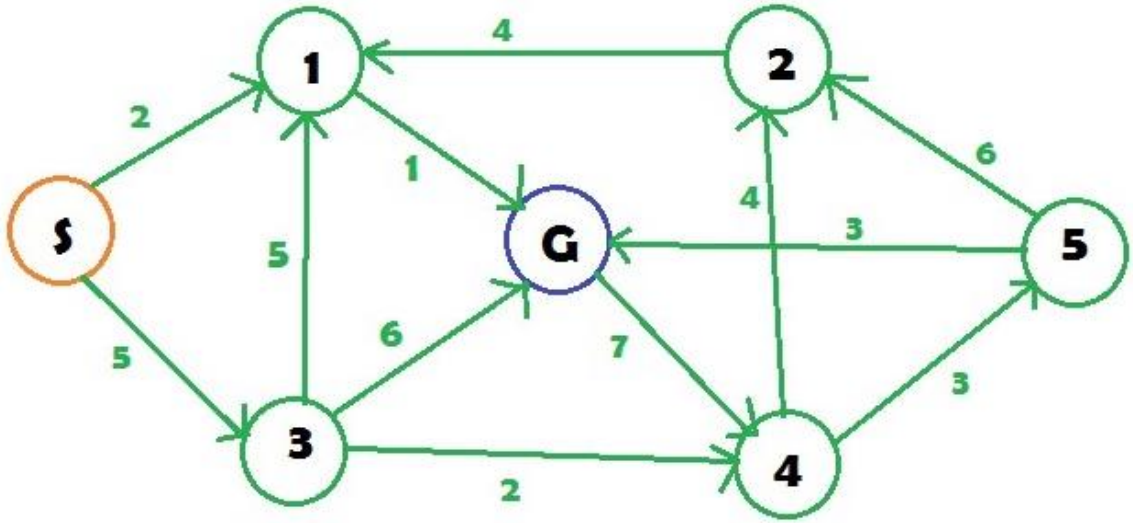
- À la fin, il faut créer le programme principal pour représenter notre graphe de recherche comme indiqué dans l'exercice
- Faut appeler ensuite la méthode IDDFS pour voir si l'état final appartient, ou non, à la profondeur maximale imposée

```
class Main {
public static void main(String[] args)
{
    // créer un graphe avec 7 neouds
    Graph g = new Graph(7);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 5);
    g.addEdge(2, 6);
    int target = 6, maxDepth = 3, src = 0;
    if (g.IDDFS(src, target, maxDepth))
    System.out.println( "La solution est en dessus de la profondeur maximale");
    else
    System.out.println(
    "La solution est en dessous de la profondeur maximale");
}
}
```

- Exercice 03 : Solution
  - Résultat de l'exécution



- Exercice 04
  - On considère un espace de recherche représenté par le graphe orienté suivant



  - L'état initial et final sont respectivement S et G.
    - Ecrire un programme Java pour implémenter l'algorithme UCS
    - Le programme doit retourner un message indiquant le chemin le moins coûteux

- Exercice 04 : Solution
  - Commencer par créer une classe UCSTraversal qui implémente un graphe en utilisant un ArrayList de type entier

```
public class UCSTraversal {

    static List<List<Integer>> graph = new ArrayList<List<Integer>>();
```

  - Créer ensuite un HashMap afin de sauvegarder le score de chaque arc dans le graphe

```
static HashMap<List<Integer>, Integer> cost = new HashMap<List<Integer>, Integer>();
```

- Exercice 04 : Solution
  - Maintenant il faut créer une fonction uniform_cost_search qui prend comme paramètres l'état initial et l'état final

```
static List<Integer> uniform_cost_search(List<Integer> goal, int start) {
// Le cout minimmum jusqu'a létat final partant de l'état initial
    List<Integer> answer = new ArrayList<Integer>();
    // créaation de la frontière
    List<Tuple<Integer, Integer>> queue = new ArrayList<Tuple<Integer, Integer>>();
    // définir le vecteur de réponse à la valeur maximale
    for (int i = 0; i < goal.size(); i++)
        answer.add(Integer.MAX_VALUE);
    // inserer l'état initial
    queue.add(new Tuple<Integer, Integer>(0, start));
    // maper le score pour le noeud visité
    HashMap<Integer, Integer> visited = new HashMap<Integer, Integer>();
    // compteur
    int count = 0;
```

Exercice 04 : Solution

- Maintenant il faut créer une fonction uniform_cost_search qui prend comme paramètres l'état initial et l'état final

```
// tant que la list queue (frontière) n'est pas vide
while (!queue.isEmpty()) {
// récuperer le premier élement de la liste d'attent
    Tuple<Integer, Integer> q = queue.get(0);
    Tuple<Integer, Integer> p = new Tuple<Integer, Integer>(-q.x, q.y);
    // retirer l'element
    queue.remove(0);
    if (goal.contains(p.y)) {
        // récuperer la position
        int index = goal.indexOf(p.y);
        // si une noindex:uvelle destination est atteinte
        if (answer.get(index) == Integer.MAX_VALUE)
            count++;
        // si le cout est inferieur
        if (answer.get(index) > p.x)
            answer.set(index, p.x);
        // retirer l'élement
        queue.remove(0);
        // si tout les destination sont atteintes
        if (count == goal.size())
            return answer;
    }
```

- Exercice 04 : Solution
  - Maintenant il faut créer une fonction uniform_cost_search qui prend comme paramètres l'état initial et l'état final

```
        if (!visited.containsKey(p.y))
            for (int i = 0; i < graph.get(p.y).size(); i++) {
            // valeuur multiplier par-1 pour que la priorité soit au top
                queue.add(new Tuple<Integer, Integer>((p.x +
                (cost.containsKey(Arrays.asList(p.y, graph.get(p.y).get(i))) ?
                cost.get(Arrays.asList(p.y, graph.get(p.y).get(i))) : 0)) * -1,
                        graph.get(p.y).get(i)));
            }
        // marqué comme visité d
        visited.put(p.y, 1);
    }
    return answer;
}
```

- Exercice 04 : Solution
  - Maintenant il faut créer le programme principal en initialisant le graphe comme indiqué dans l'exercice

```
    // main function
    public static void main(String[] args) {
        // Créer le graphe
        graph = new ArrayList<List<Integer>>();
Run | Debug

        for (int i = 0; i < 7; i++) {
            graph.add(new ArrayList<Integer>());
        }

        // l'ajout des acrs
        graph.get(index:0).add(e:1);
        graph.get(index:0).add(e:3);
        graph.get(index:3).add(e:1);
        graph.get(index:3).add(e:6);
        graph.get(index:3).add(e:4);
        graph.get(index:1).add(e:6);
        graph.get(index:4).add(e:2);
        graph.get(index:4).add(e:5);
        graph.get(index:2).add(e:1);
        graph.get(index:5).add(e:2);
        graph.get(index:5).add(e:6);
        graph.get(index:6).add(e:4);
```

Exercice 04 : Solution

- Ensuite il faut rajouter les coûts sur chaque arc; spécifier l'état final et afficher les résultats

```
        // ajouter les cout
        cost.put(Arrays.asList(...a:0, 1), value:2);
        cost.put(Arrays.asList(...a:0, 3), value:5);
        cost.put(Arrays.asList(...a:1, 6), value:1);
        cost.put(Arrays.asList(...a:3, 1), value:5);
        cost.put(Arrays.asList(...a:3, 6), value:6);
        cost.put(Arrays.asList(...a:3, 4), value:2);
        cost.put(Arrays.asList(...a:2, 1), value:4);
        cost.put(Arrays.asList(...a:4, 2), value:4);
        cost.put(Arrays.asList(...a:4, 5), value:3);
        cost.put(Arrays.asList(...a:5, 2), value:6);
        cost.put(Arrays.asList(...a:5, 6), value:3);
        cost.put(Arrays.asList(...a:6, 4), value:7);

        // l'état final
        List<Integer> goal = new ArrayList<Integer>();
        goal.add(e:6);

        List<Integer> answer = uniform_cost_search(goal, start:0);

        // afficher les résultats
        System.out.print("Minimum cost from 0(S) to 6(G) is = " + answer.get(index:0));
    }
}
```

- Exercice 04 : Solution
  - Création de la classe Tuple

```java
class Tuple<X, Y> {
    public final X x;
    public final Y y;

    public Tuple(X x, Y y) {
        this.x = x;
        this.y = y;
    }
}
```

  - Résultat de l'exécution

```
PROBLEMS 6  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\KHALED\Strategies>  c:; cd 'c:\Users\KHALED\Strategies'; & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExc
eptionMessages' '-cp' 'C:\Users\KHALED\AppData\Roaming\Code\User\workspaceStorage\107beaf93822a25da2787408abe466bf\redhat.java\jdt_ws\Strateg
ies_743eeb7a\bin' 'UCSTraversal'
Minimum cost from 0(S) to 6(G) is = 3
PS C:\Users\KHALED\Strategies>
```