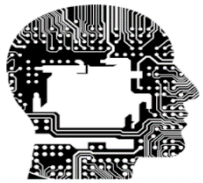
Université Constantine 2
جامعة قسنطينة 2

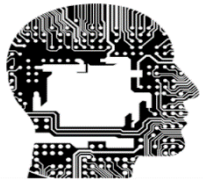


Foundation of Artificial Intelligence

TP 02

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 informée -

Dr. NECIBI Khaled

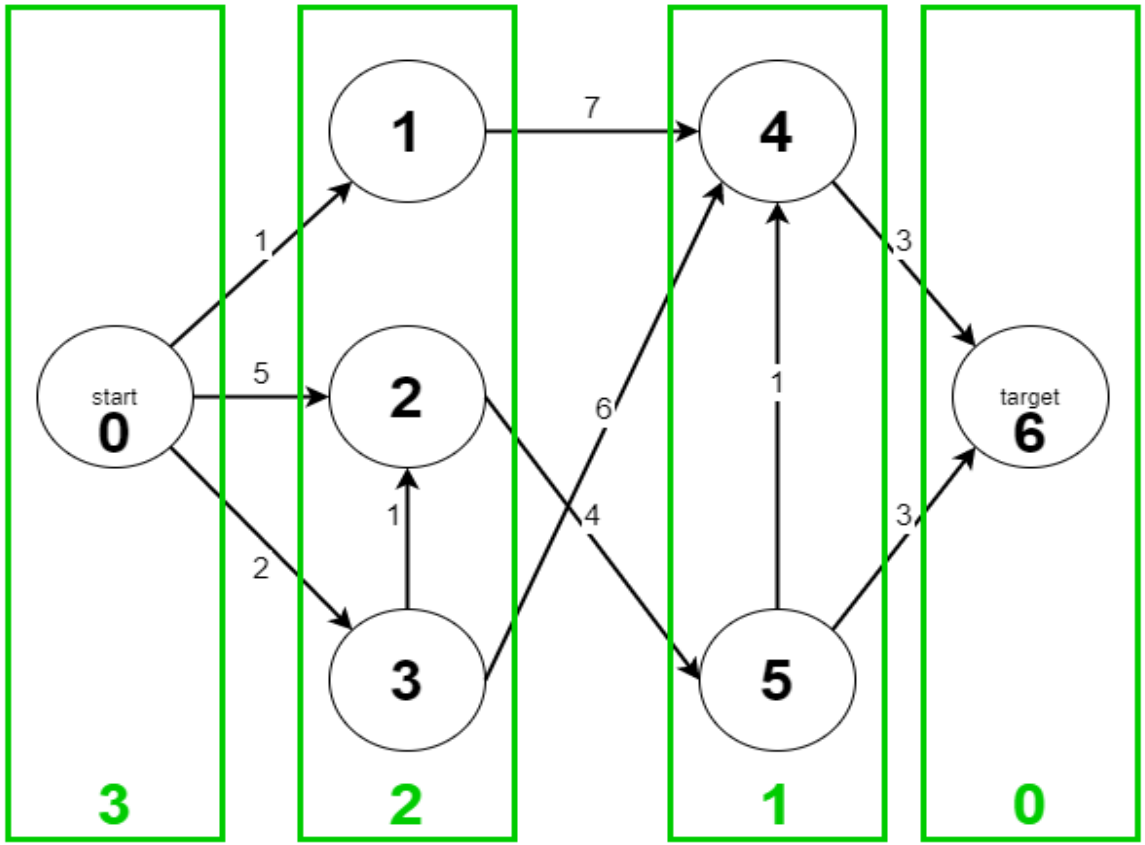
Faculté des nouvelles technologies

Khaled.necibi@univ-constantine2.dz

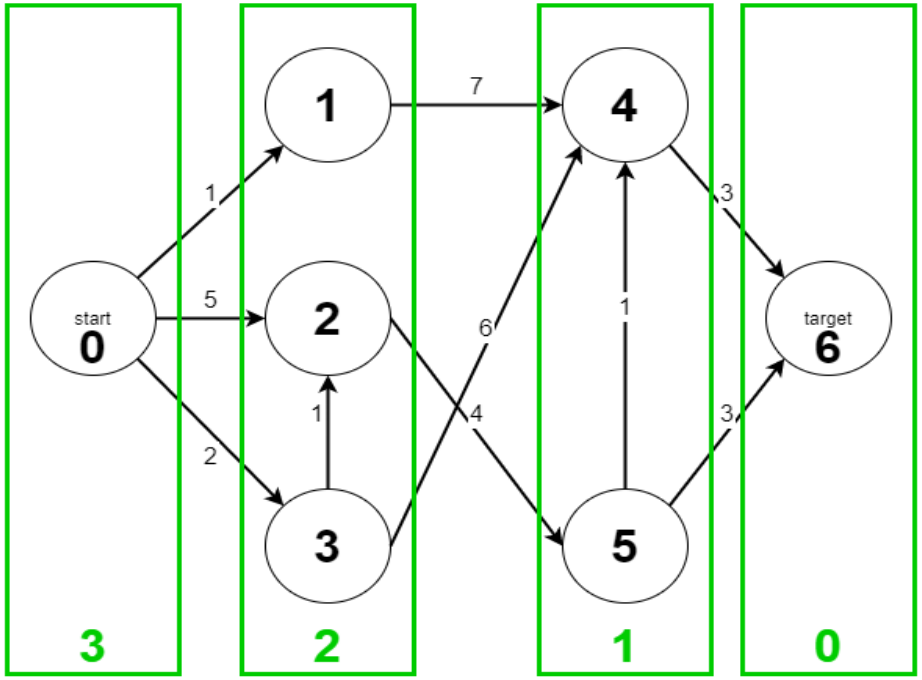
Etudiants concernés

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

- Exercice 01
  - On considère un espace de recherche représenté par le graphe orienté suivant



- Exercice 01
  - Question : Ecrirer un programme Java pour implemnter l'algorithme A*
  - L'euristique proposé traite chaque profondeur de l'espace de recherche par d'étape (classe) jusqu'à l"état final



- Exercice 01 : Solution
  - On commence par la création d'une classe Node pour représenter les nœuds du graphe de recherche

Node.java > Node > Node(double)

```java
import java.util.*;

public class Node implements Comparable<Node> {
    // Id for readability of result purposes
    private static int idCounter = 0;
    public int id;

    // Parent in the path
    public Node parent = null;

    public List<Edge> neighbors;

    // Evaluation functions
    public double f = Double.MAX_VALUE;
    public double g = Double.MAX_VALUE;
    // Hardcoded heuristic
    public double h;
```

- Exercice 01 : Solution
  - On commence par la création d'une classe Node pour représenter les nœuds du graphe de recherche

Node.java > Node

```
18
19      Node(double h){
20          this.h = h;
21          this.id = idCounter++;
22          this.neighbors = new ArrayList<>();
23      }
24
25      @Override
26      public int compareTo(Node n) {
27          return Double.compare(this.f, n.f);
28      }
29
30      public static class Edge {
31          Edge(int weight, Node node){
32              this.weight = weight;
33              this.node = node;
34          }
35
36          public int weight;
37          public Node node;
38      }
```

- Exercice 01 : Solution
  - On commence par la création d'une classe Node pour représenter les nœuds du graphe de recherche

Node.java > Node

```
39
40      public void addBranch(int weight, Node node){
41          Edge newEdge = new Edge(weight, node);
42          neighbors.add(newEdge);
43      }
44
45      public double calculateHeuristic(Node target){
46          return this.h;
47      }
48
49      public static Node aStar(Node start, Node target){
50        PriorityQueue<Node> closedList = new PriorityQueue<>();
51        PriorityQueue<Node> openList = new PriorityQueue<>();
52
53        start.f = start.g + start.calculateHeuristic(target);
54        openList.add(start);
55
```

- Exercice 01 : Solution
  - On commence par la création d'une classe Node pour représenter les nœuds du graphe de recherche

Node.java > Node

```
55
56          while(!openList.isEmpty()){
57              Node n = openList.peek();
58              if(n == target){
59                  return n;
60              }
61
62              for(Node.Edge edge : n.neighbors){
63                  Node m = edge.node;
64                  double totalWeight = n.g + edge.weight;
65
66                  if(!openList.contains(m) && !closedList.contains(m)){
67                      m.parent = n;
68                      m.g = totalWeight;
69                      m.f = m.g + m.calculateHeuristic(target);
70                      openList.add(m);
71                  } else {
72                      if(totalWeight < m.g){
73                          m.parent = n;
74                          m.g = totalWeight;
75                          m.f = m.g + m.calculateHeuristic(target);
```

- Exercice 01 : Solution
  - On commence par la création d'une classe Node pour représenter les nœuds du graphe de recherche

Node.java > Node

```
77                                  if(closedList.contains(m)){
78                                      closedList.remove(m);
79                                      openList.add(m);
80                                  }
81                              }
82                          }
83                      }
84
85                      openList.remove(n);
86                      closedList.add(n);
87                  }
88                  return null;
89              }
90
```

- Exercice 01 : Solution
  - On commence par la création d'une classe Node pour représenter les nœuds du graphe de recherche

Node.java > Node

```java
    public static void printPath(Node target){
        Node n = target;

        if(n==null)
            return;

        List<Integer> ids = new ArrayList<>();

        while(n.parent != null){
            ids.add(n.id);
            n = n.parent;
        }
        ids.add(n.id);
        Collections.reverse(ids);

        for(int id : ids){
            System.out.print(id + " ");
        }
        System.out.println(x:"");
    }
}
```

- Exercice 01 : Solution
  - Ensuite on procède à l'implémentation de l'algorithme A*

Star.java > ...

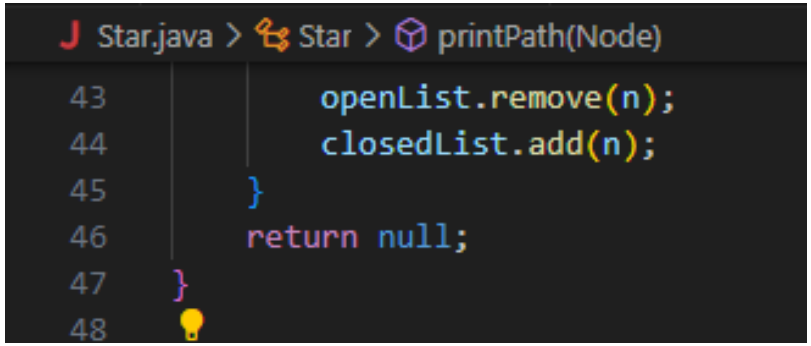
```java
import java.util.*;

public class Star {

 static Node aStar(Node start, Node target){
    PriorityQueue<Node> closedList = new PriorityQueue<>();
    PriorityQueue<Node> openList = new PriorityQueue<>();

    start.f = start.g + start.calculateHeuristic(target);
    openList.add(start);

    while(!openList.isEmpty()){
        Node n = openList.peek();
        if(n == target){
            return n;
        }

        for(Node.Edge edge : n.neighbors){
            Node m = edge.node;
            double totalWeight = n.g + edge.weight;

```

- Exercice 01 : Solution
  - Ensuite on procède à l'implémentation de l'algorithme A*

Star.java > Star > aStar(Node, Node)

```
19              Node m = edge.node;
20              double totalWeight = n.g + edge.weight;
21
22              if(!openList.contains(m) && !closedList.contains(m)){
23                  m.parent = n;
24                  m.g = totalWeight;
25                  m.f = m.g + m.calculateHeuristic(target);
26                  openList.add(m);
27              } else {
28                  if(totalWeight < m.g){
29                      m.parent = n;
30                      m.g = totalWeight;
31                      m.f = m.g + m.calculateHeuristic(target);
32
33                      if(closedList.contains(m)){
34                          closedList.remove(m);
35                          openList.add(m);
36                      }
37                  }
38              }
39          }
```

- Exercice 01 : Solution
  - Ensuite on procède à l'implémentation de l'algorithme A*



- Exercice 01 : Solution
  - Ensuite on procède à l'implémentation de l'algorithme A*

Star.java > Star > printPath(Node)

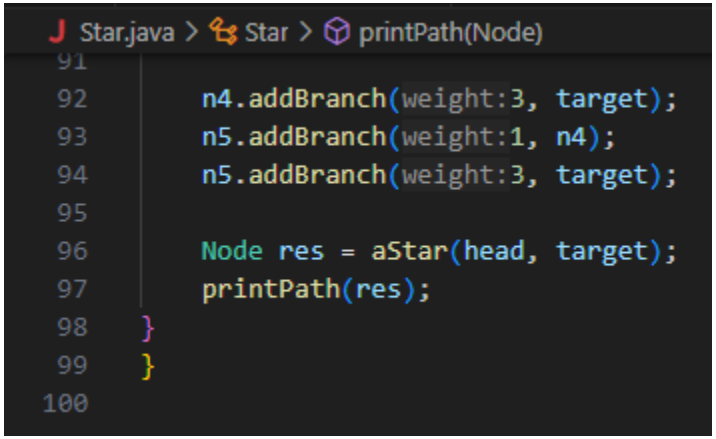
```java
public static void printPath(Node target){
    Node n = target;

    if(n==null)
        return;

    List<Integer> ids = new ArrayList<>();

    while(n.parent != null){
        ids.add(n.id);
        n = n.parent;
    }
    ids.add(n.id);
    Collections.reverse(ids);

    for(int id : ids){
        System.out.print(id + " ");
    }
    System.out.println(x:"");
}
```

- Exercice 01 : Solution
  - Maintenant nous allons procéder à la creation du graphe et appler les méthodes nécessaires

Star.java > Star > printPath(Node)

```
Run | Debug
71  public static void main(String[] args) {
72      Node head = new Node(h:3);
73      head.g = 0;
74
75      Node n1 = new Node(h:2);
76      Node n2 = new Node(h:2);
77      Node n3 = new Node(h:2);
78
79      head.addBranch(weight:1, n1);
80      head.addBranch(weight:5, n2);
81      head.addBranch(weight:2, n3);
82      n3.addBranch(weight:1, n2);
83
84      Node n4 = new Node(h:1);
85      Node n5 = new Node(h:1);
86      Node target = new Node(h:0);
87
88      n1.addBranch(weight:7, n4);
89      n2.addBranch(weight:4, n5);
90      n3.addBranch(weight:6, n4);
```

- Exercice 01 : Solution
  - Maintenant nous allons procéder à la creation du graphe et appler les méthodes nécessaires



Résolution de problème par recherche non informée

- Exercice 01 : Solution
  - L'exécution du programme principal doit retourner l'ordre de parcours des nœuds suivant :

```
PROBLEMS 10   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:+ShowCod
eDetailsInExceptionMessages' '-cp' 'C:\Users\KHALED\AppData\Roaming\Code\User\workspaceStorage\107beaf93822a25da2787408abe466bf\
redhat.java\jdt_ws\Strategies_743eeb7a\bin' 'Star'
0 3 2 5 6
PS C:\Users\KHALED\Strategies>
```