Lebanese University faculty of sciences

I3341

Implementation of Dijkstra algorithm using priority queue

Mahdi Mahdi – Ali Tawbi

Dijkstra's algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

priority queue

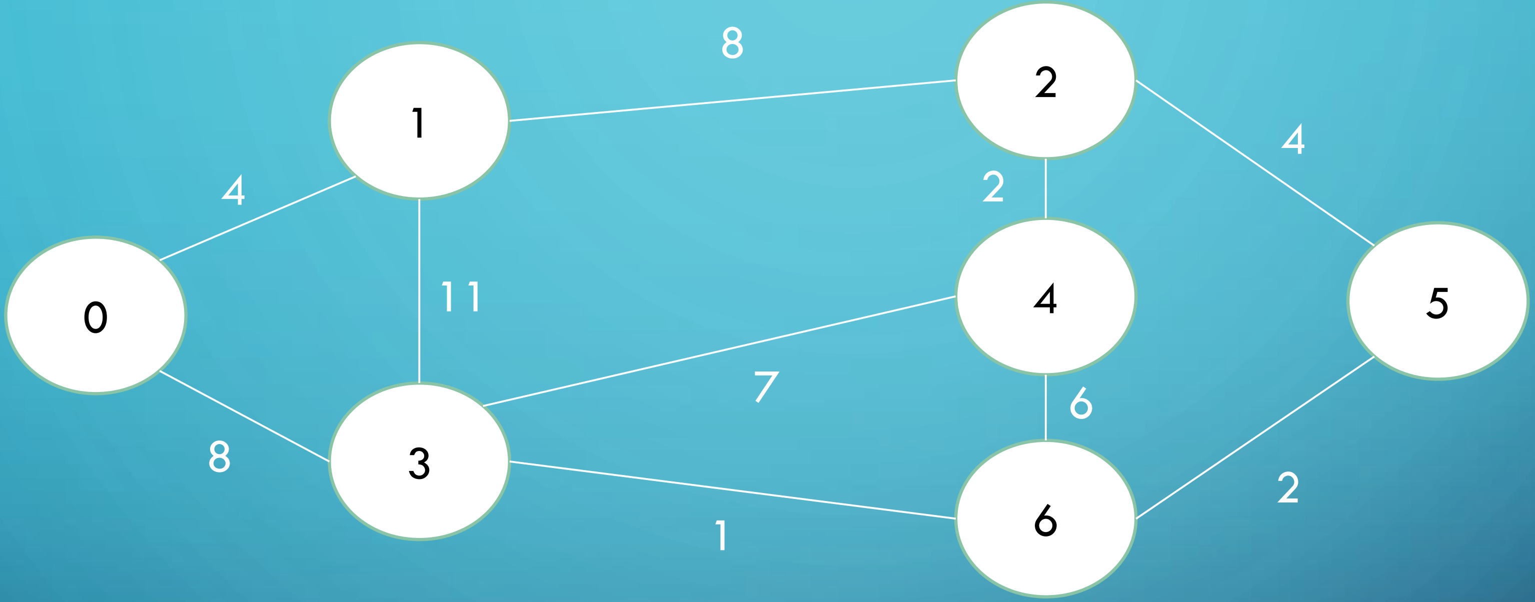
A **priority queue** is an abstract data type similar to a regular queue or stack data structure in which each element additionally has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority

Dijkstra algorithm using priority queue

We will use the priority queue to implement the **Dijkstra algorithm .**
First we take the source node of the graph and put in the priority queue the neighbors nodes of the source with cost of each one.
We pop from the queue the minimum cost ,if this cost is not already exist in the results , we obtain the minimum cost from the source and the popped node.
The popped destination will be the source and we will repeat the same operation from the begin but will add the popped result cost to the inserted elements in the queue in next step
We will stop when we have all minimum cost between the source and every node

Example:

We will take the source 0 and will find the shortest path from 0 to all nodes

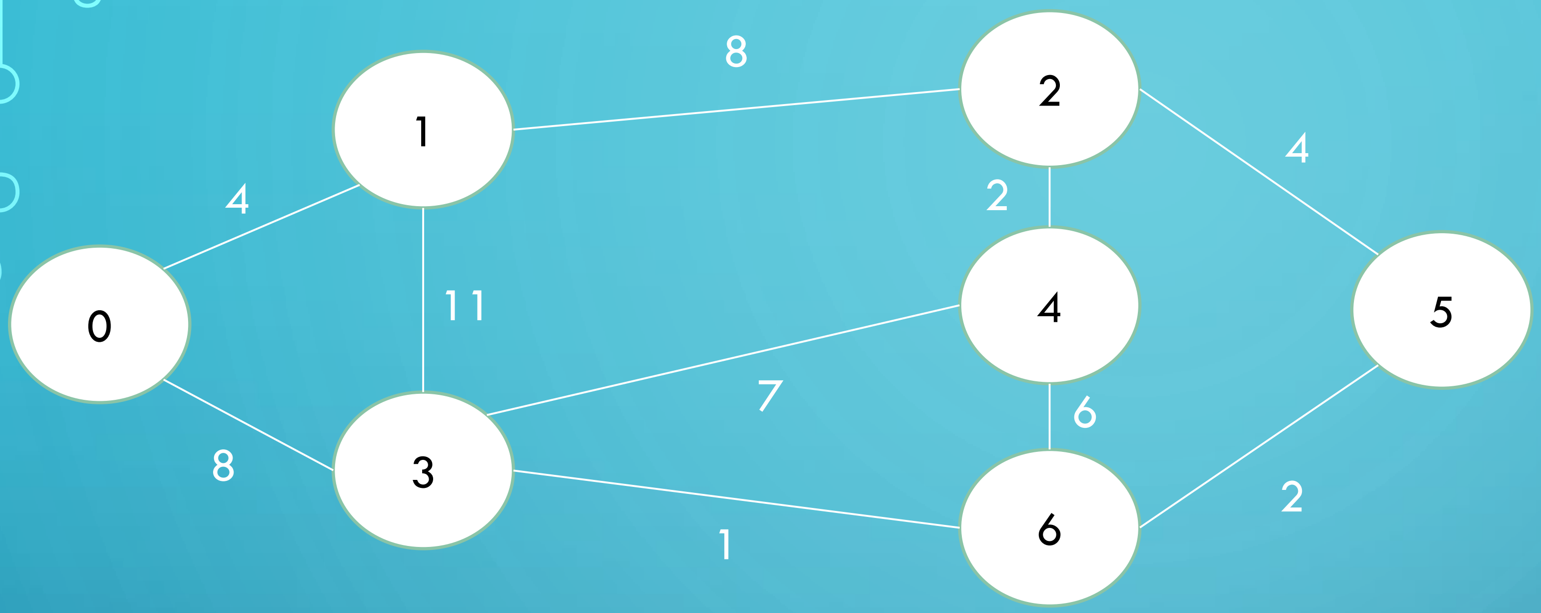


Priority Queue Structure:

- Node From
- Node To
- Cost

Algorithm:

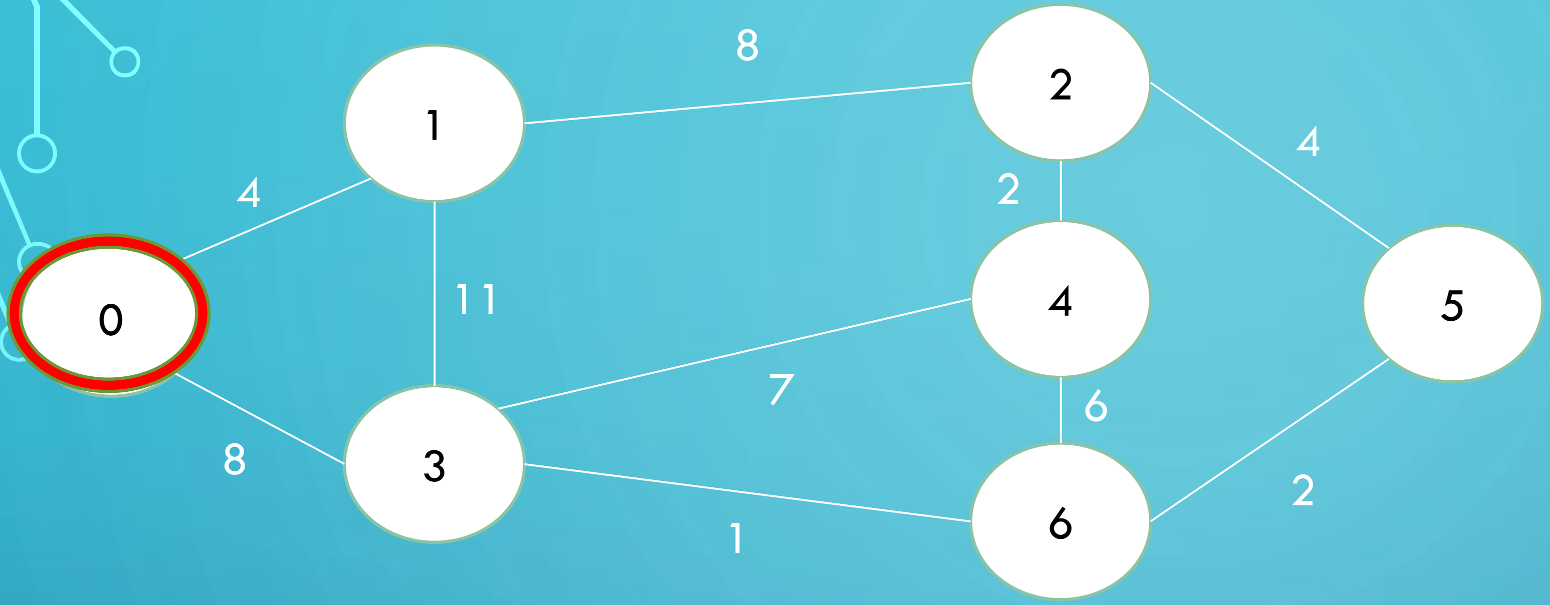
```
Source = Graph.source
Cost = 0
Result.add(from:Source,to:Source,minCost:0)
nbOfRes = 1
While (nbOfRes < Graph.nbOfNodes)
{
    foreach (neighbor node of Source)
    {
        PriorityQ.Add (from:Source,to:neighbor , cost(Source,neighbor) + Cost)
    }
    Element = PriorityQ.popElement()
    if Result.exist(from:Graph.source,to:Element.to):
        Result.add(from:Graph.source,to:Element.to,Element.cost)
    Source = Element.to
    Cost = Element.cost
}
```



From 0 To	Min Cost
0	0
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞

Priority Queue

First we choose the source of the graph , we choose 0 ,and we set the cost from 0 to all other nodes equal infinite and equal 0 to node 0,the priority queue is empty



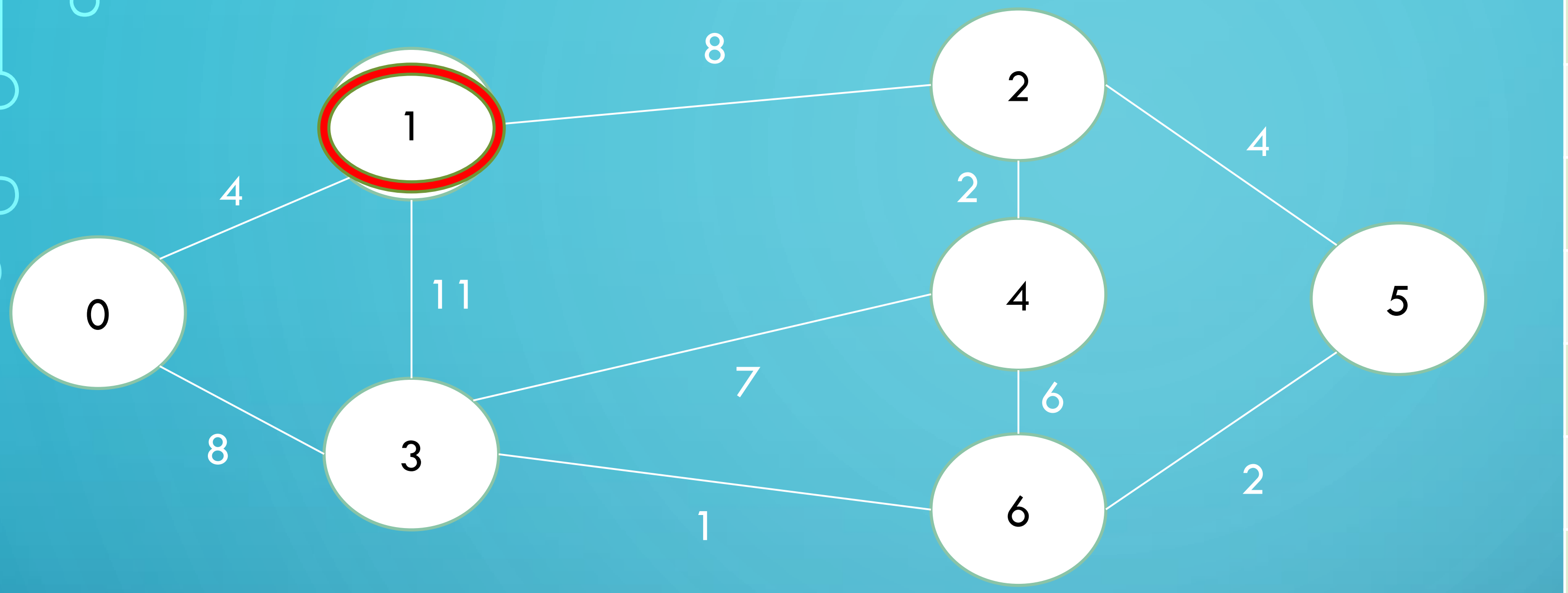
From 0 To	Min Cost
0	0
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞

Priority Queue

From 0	From 0								
To 1	To 3								
Cost 4	Cost 8								

We are at 0

We insert in the priority queue the cost to all neighbors of 0



From 0 To	Min Cost
0	0
1	4
2	∞
3	∞
4	∞
5	∞
6	∞

Priority Queue

From 0	From 1	From 1							
To 3	To 2	To 3							
Cost 8	Cost 12	Cost 15							

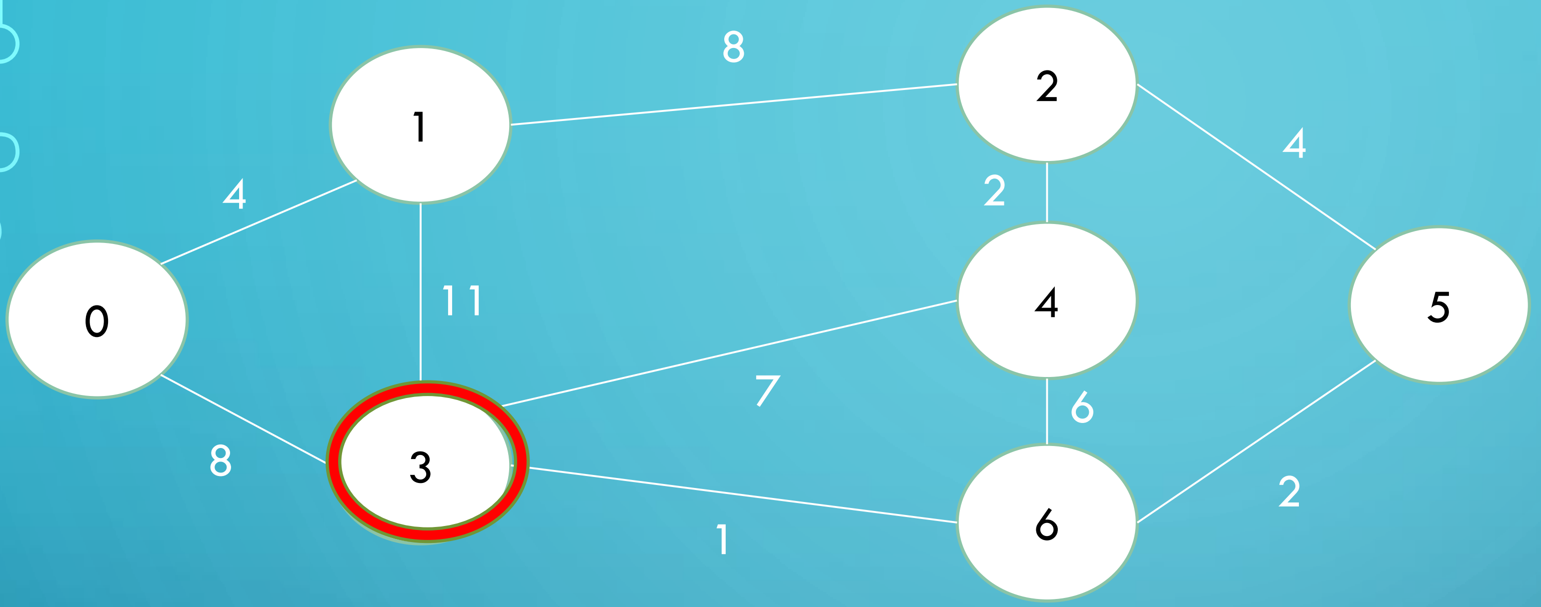
Popped From the queue

From 0
To 1
Cost 4

We pop from the queue and insert the value in the min costs table

We are now at 1

We insert in the priority queue the cost to all neighbors of 1 and we add cost from 0 to 1 to the cost, and we ignore the node 0 Because has already inserted



From 0 To	Min Cost
0	0
1	4
2	∞
3	8
4	∞
5	∞
6	∞

Priority Queue

From 3	From 1	From 1	From 3					
To 6	To 2	To 3	To 4					
Cost 9	Cost 12	Cost 15	Cost 15					

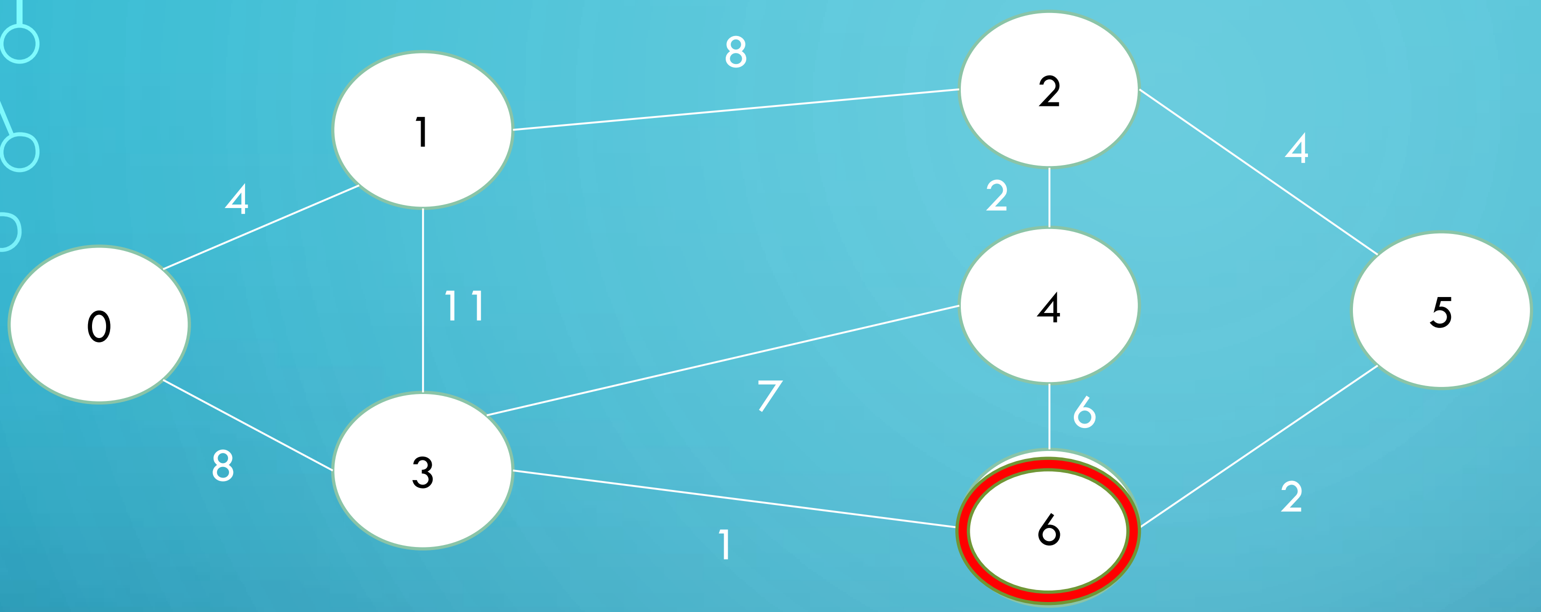
Popped From the queue

From 0
To 4
Cost 8

We pop from the queue and insert the value in the min costs table

We are now at 3

We insert in the priority queue the cost to all neighbors of 3 and we add cost from 0 to 3 to the cost , and we ignore the nodes 0 and 1 Because they are already inserted



From 0 To	Min Cost
0	0
1	4
2	∞
3	8
4	∞
5	∞
6	9

Priority Queue

From 6	From 1	From 1	From 3	From 6			
To 5	To 2	To 3	To 4	To 4			
Cost 11	Cost 12	Cost 15	Cost 15	Cost 15			

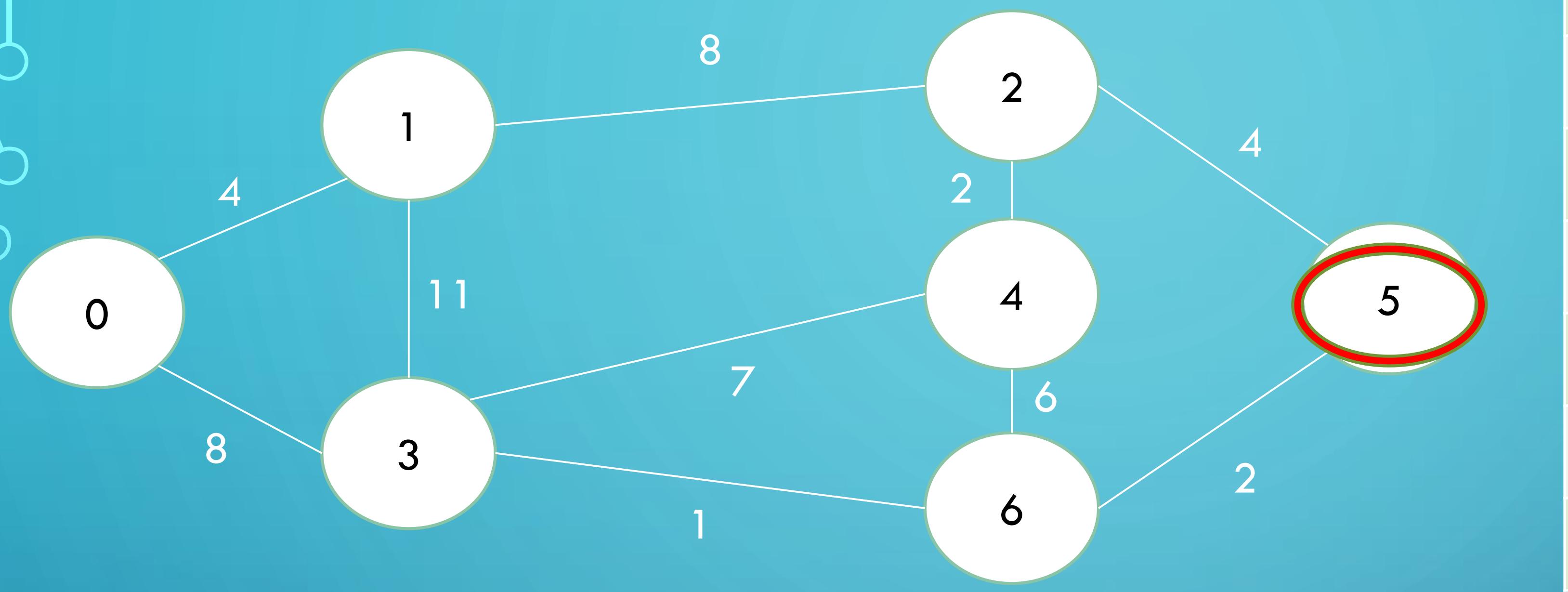
Popped From the queue

From 3
To 6
Cost 8

We pop from the queue and insert the value in the min costs table

We are now at 6

We insert in the priority queue the cost to all neighbors of 6 and we add cost from 0 to 6 to the cost , and we ignore the node 3 Because 3 has already inserted



From 0 To	Min Cost
0	0
1	4
2	∞
3	8
4	∞
5	11
6	9

Priority Queue

From 1	From 1	From 3	From 6	From 5			
To 2	To 3	To 4	To 4	To 2			
Cost 12	Cost 15	Cost 15	Cost 15	Cost 15			

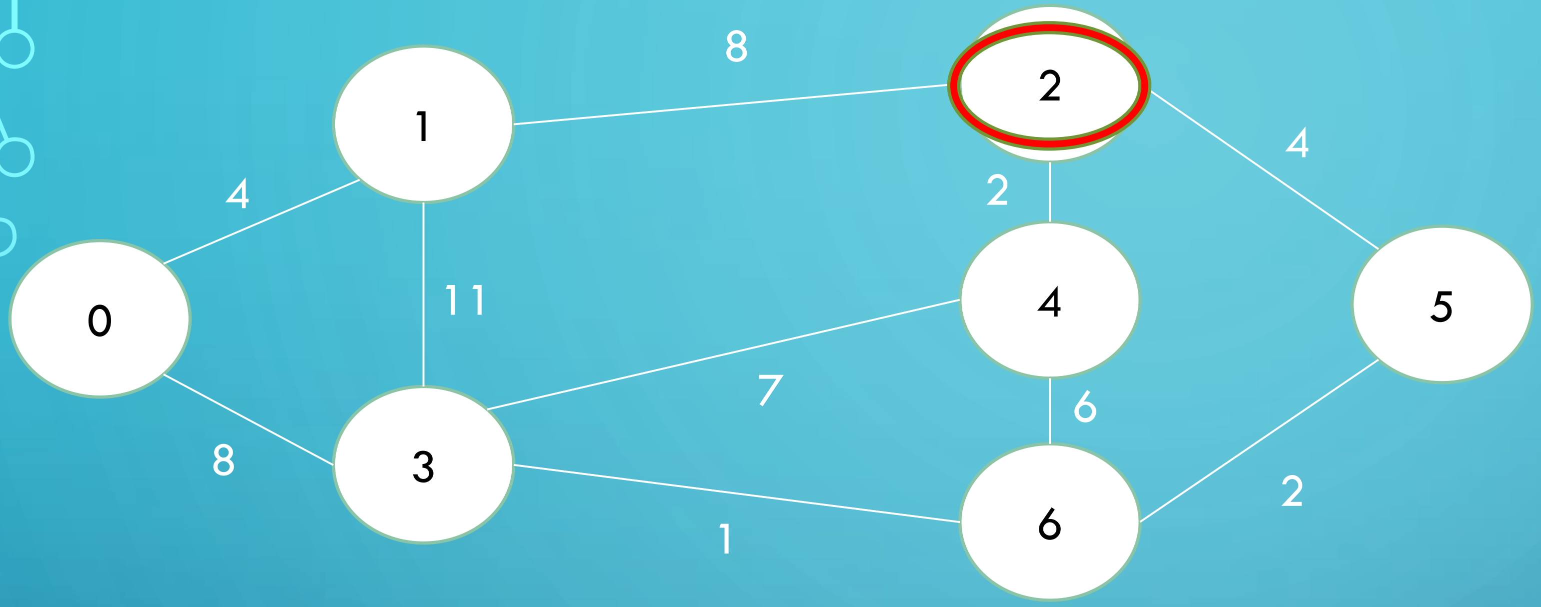
Popped From the queue

From 6
To 5
Cost

We pop from the queue and insert the value in the min costs table

We are now at 5

We insert in the priority queue the cost to all neighbors of 5 and we add cost from 0 to 5 to the cost , and we ignore the node 6 Because 6 has already inserted



From 0 To	Min Cost
0	0
1	4
2	12
3	8
4	∞
5	11
6	9

Priority Queue

From 2	From 1	From 3	From 6	From 5			
To 4	To 3	To 4	To 4	To 2			
Cost 14	Cost 15	Cost 15	Cost 15	Cost 15			

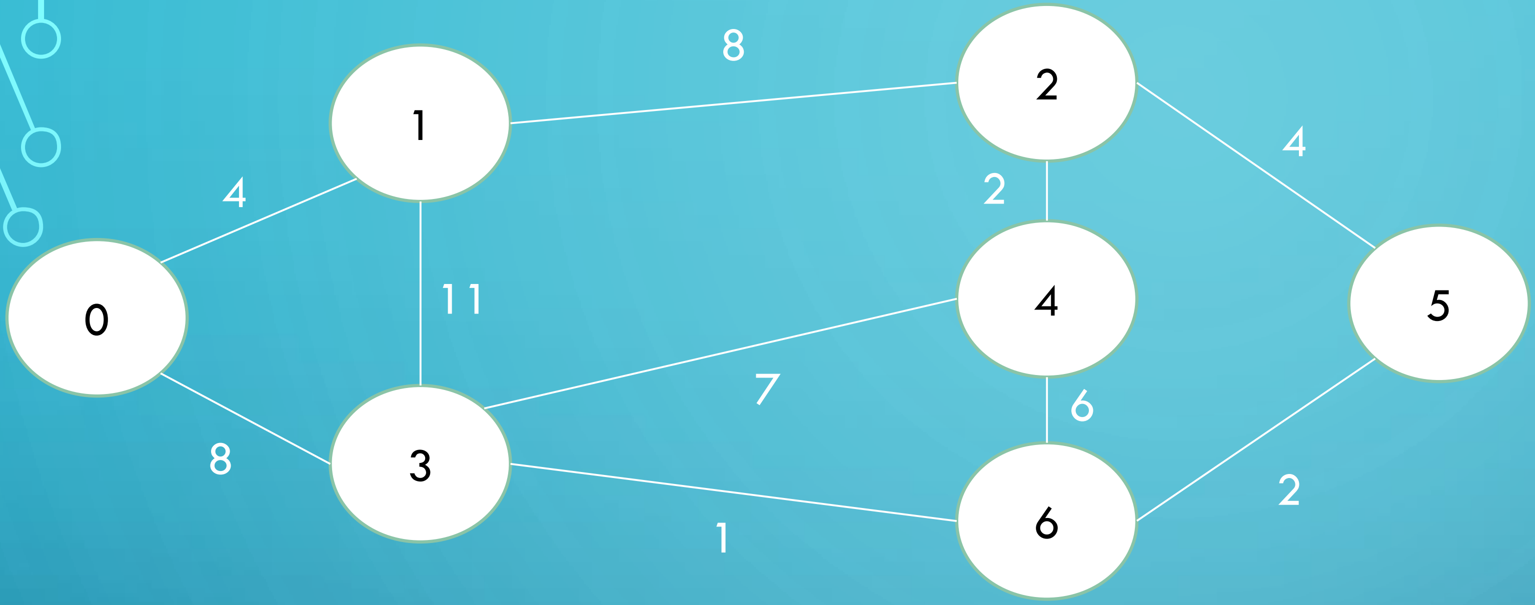
Popped From the queue

From 1
To 2
Cost 12

We pop from the queue and insert the value in the min costs table

We are now at 2

We insert in the priority queue the cost to all neighbors of 2 and we add cost from 0 to 2 to the cost , and we ignore the nodes 5,1 Because they are already inserted



From 0 To	Min Cost
0	0
1	4
2	12
3	8
4	14
5	11
6	9

Priority Queue

From 1	From 3	From 6	From 5			
To 3	To 4	To 4	To 2			
Cost 15	Cost 15	Cost 15	Cost 15			

Popped From the queue

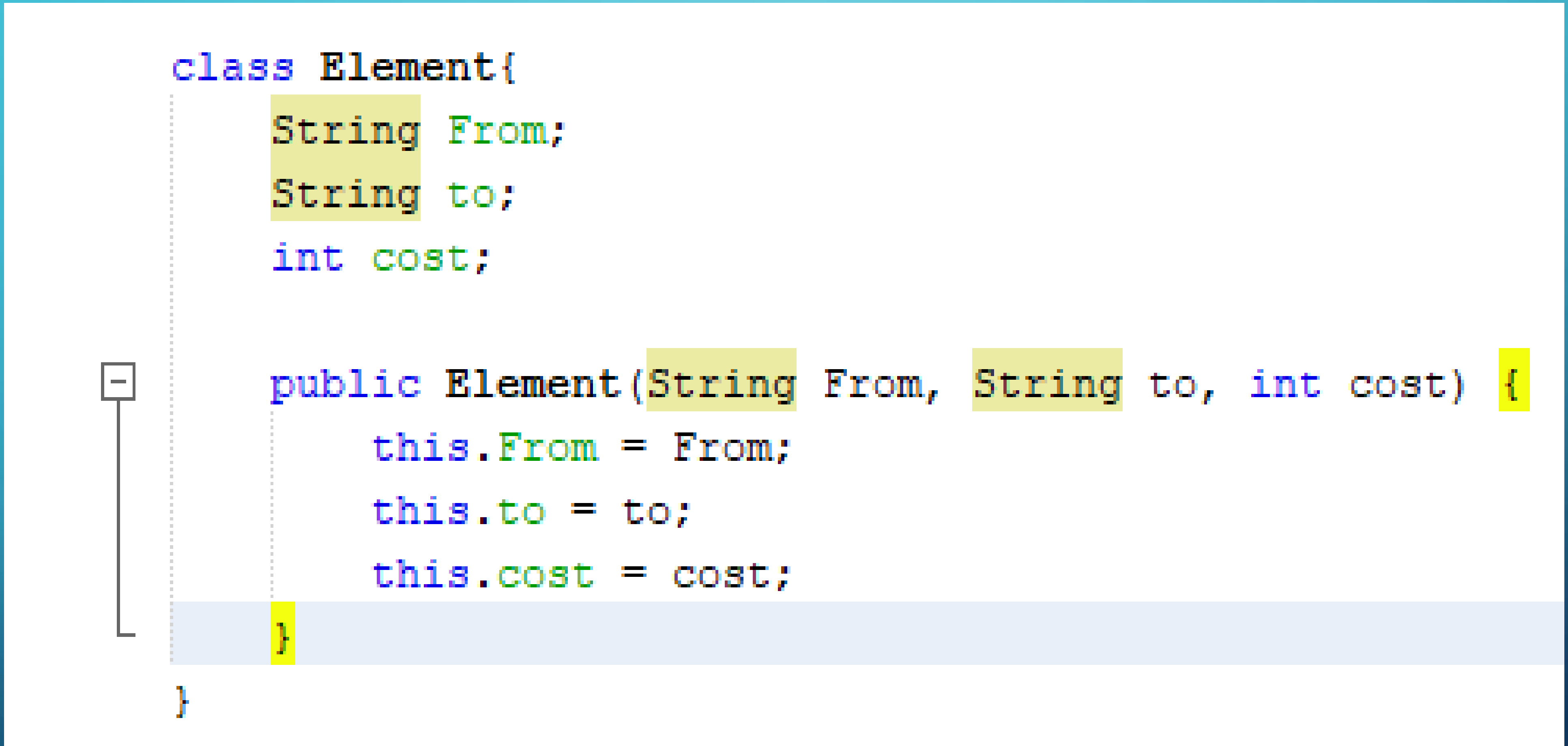
From 2
To 4
Cost 12

We pop from the queue and insert the value in the min costs table

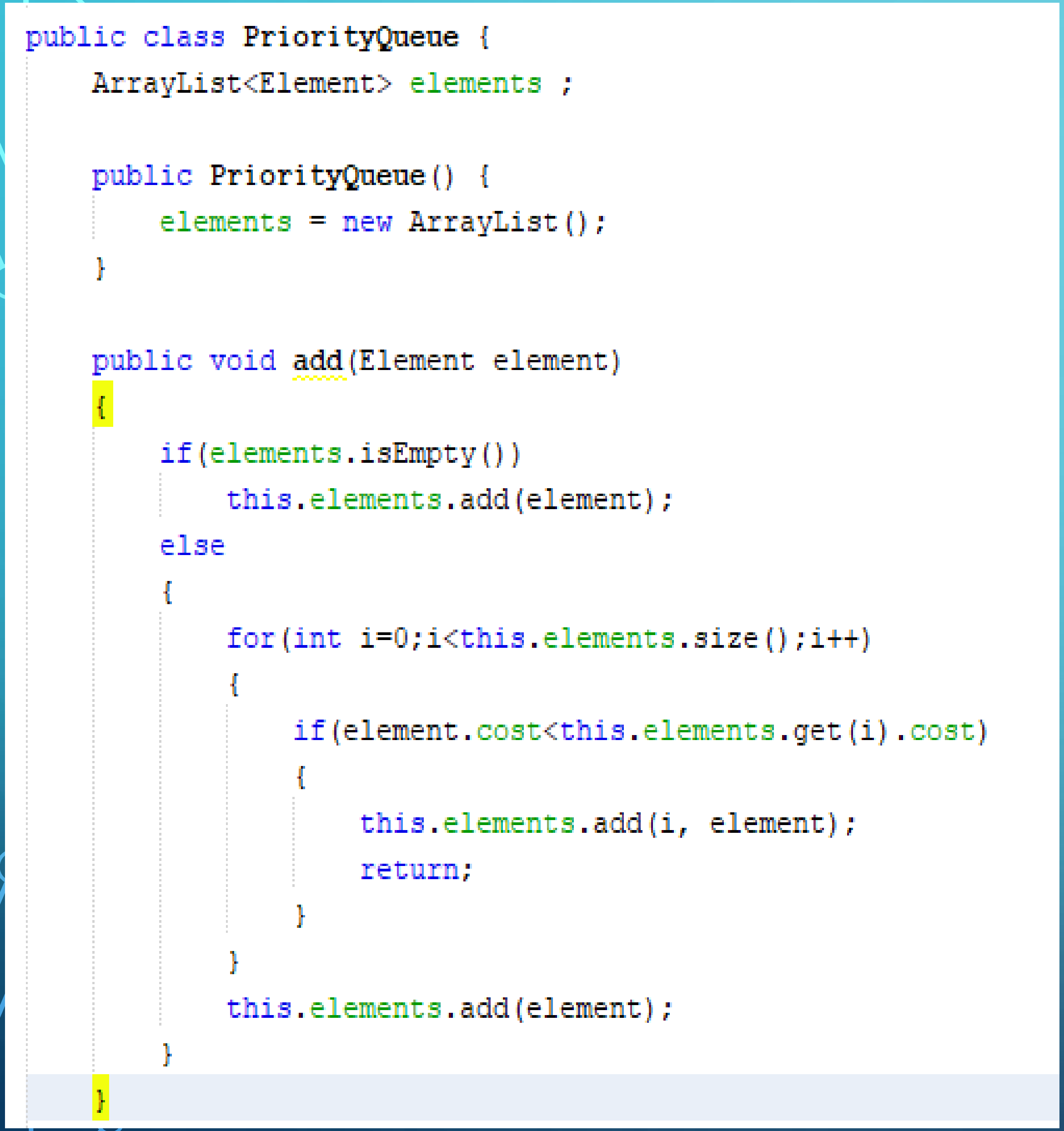
We ignore the rest in the queue because we finish

IMPLEMENT THE ALGORITHM IN JAVA

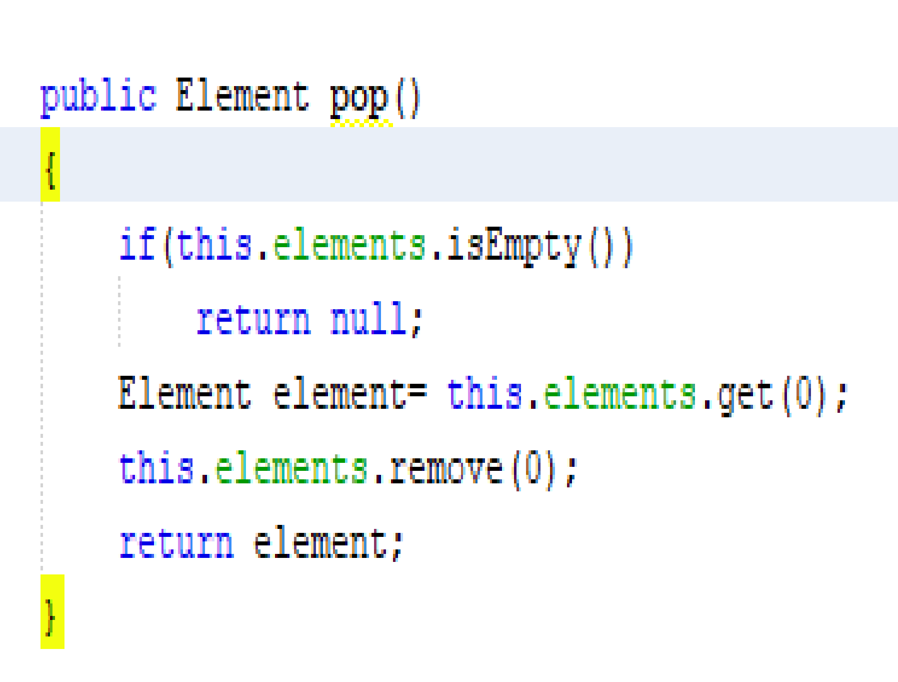
This class is for nodes of the queue and nodes of the graph



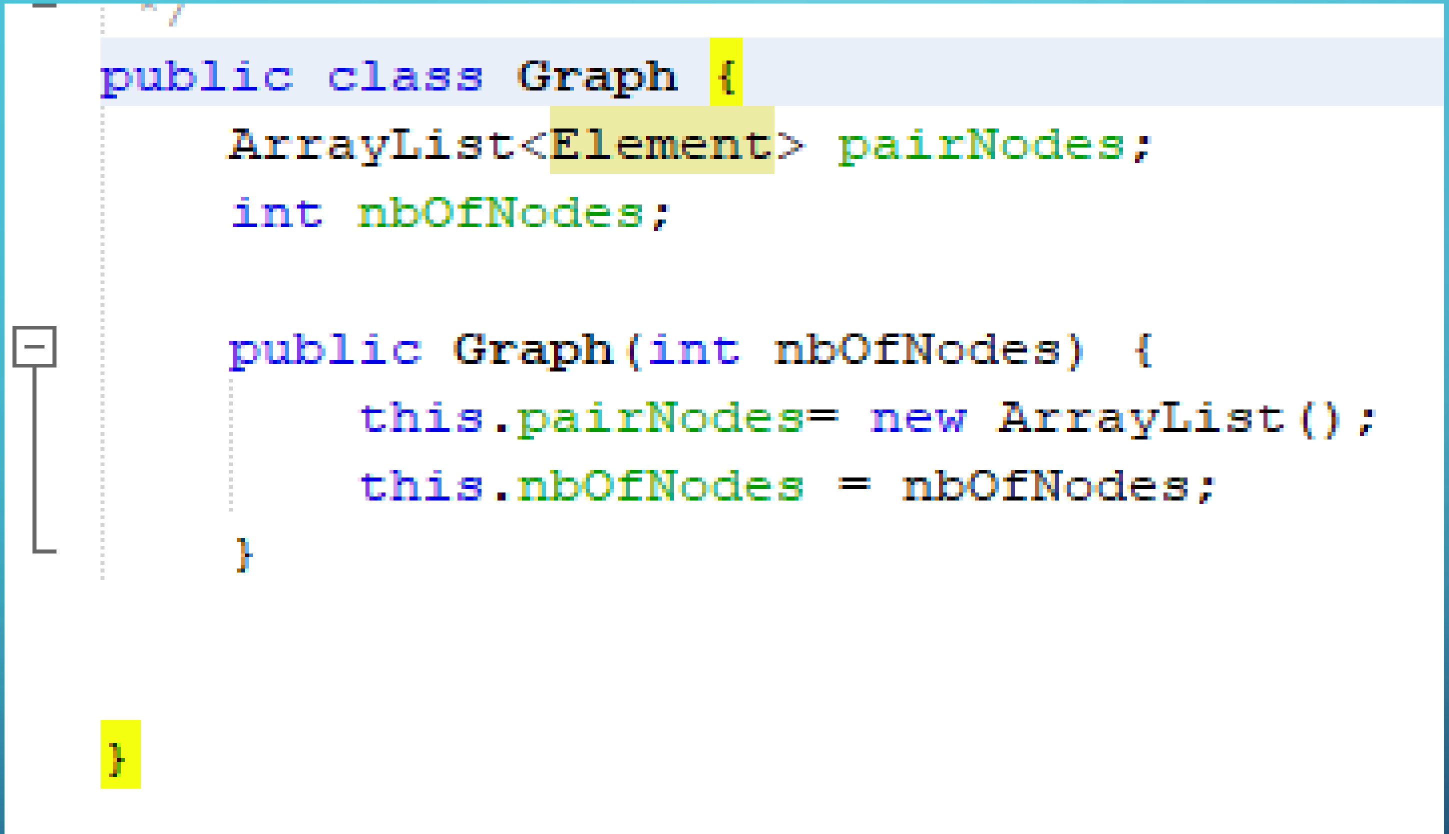
```java
class Element{
    String From;
    String to;
    int cost;

    public Element(String From, String to, int cost) {
        this.From = From;
        this.to = to;
        this.cost = cost;
    }
}
```



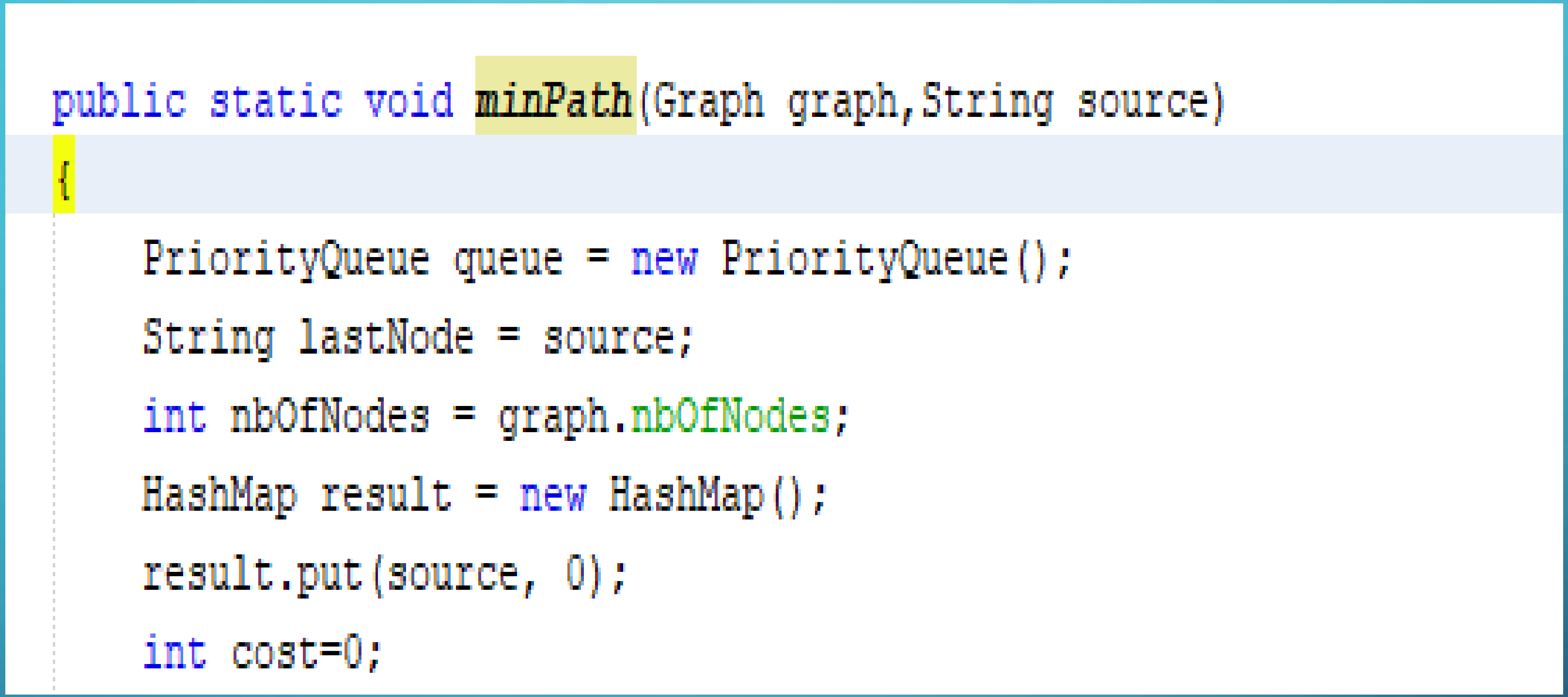
```
public class PriorityQueue {
    ArrayList<Element> elements ;

    public PriorityQueue() {
        elements = new ArrayList();
    }

    public void add(Element element)
    {
        if(elements.isEmpty())
            this.elements.add(element);
        else
        {
            for(int i=0;i<this.elements.size();i++)
            {
                if(element.cost<this.elements.get(i).cost)
                {
                    this.elements.add(i, element);
                    return;
                }
            }
            this.elements.add(element);
        }
    }
```



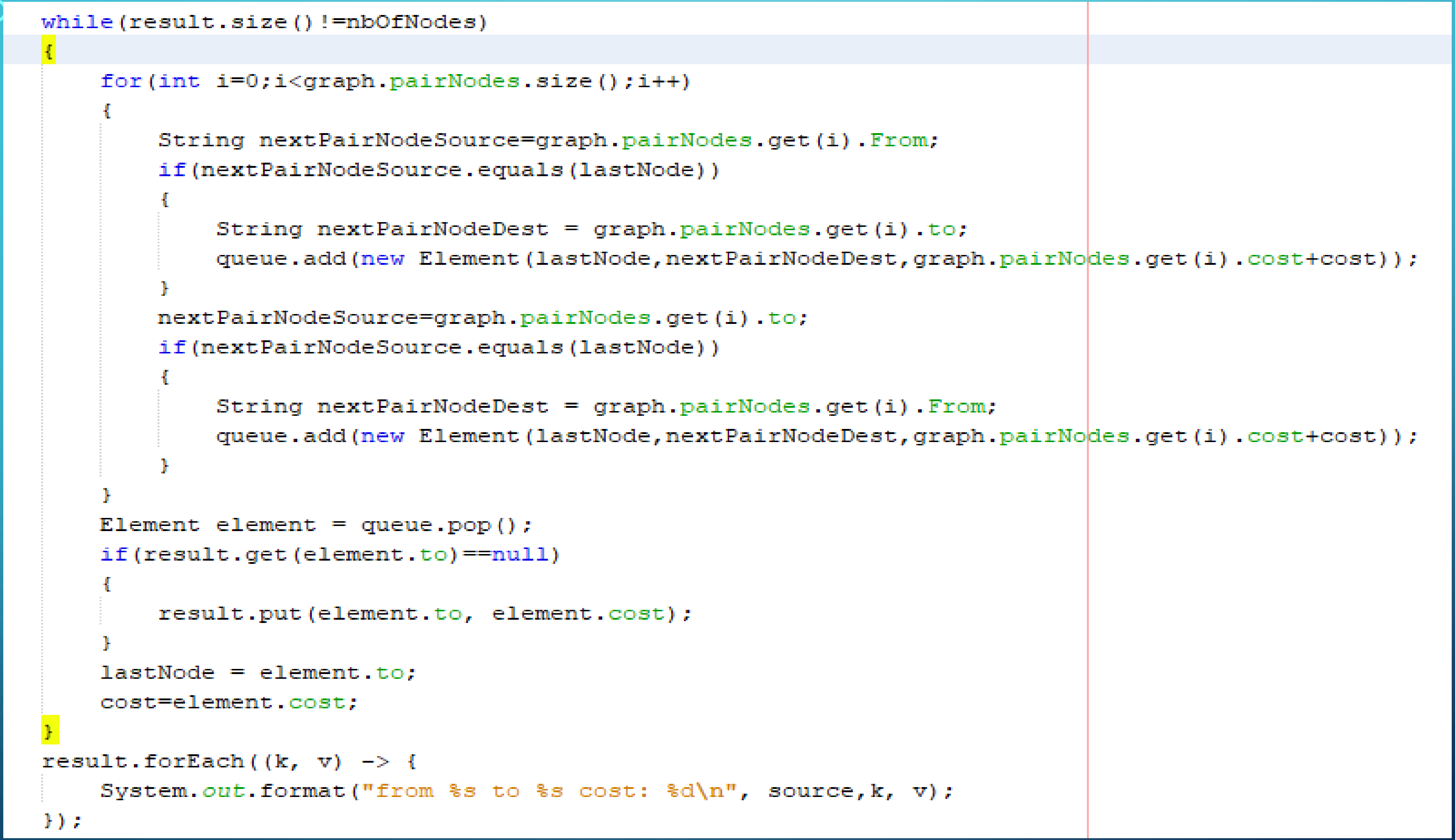
```
public Element pop()
{
    if(this.elements.isEmpty())
        return null;
    Element element= this.elements.get(0);
    this.elements.remove(0);
    return element;
}
```



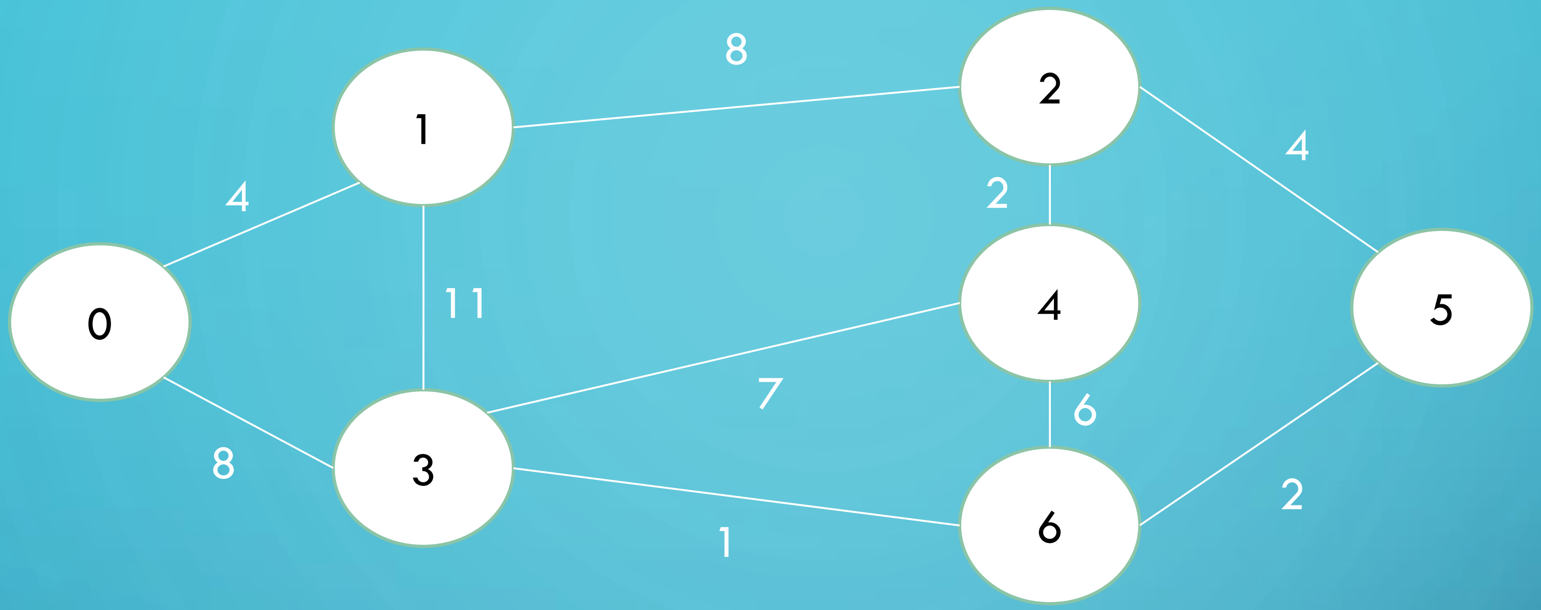
```
*/
public class Graph {
    ArrayList<Element> pairNodes;
    int nbOfNodes;

    public Graph(int nbOfNodes) {
        this.pairNodes= new ArrayList();
        this.nbOfNodes = nbOfNodes;
    }


}
```

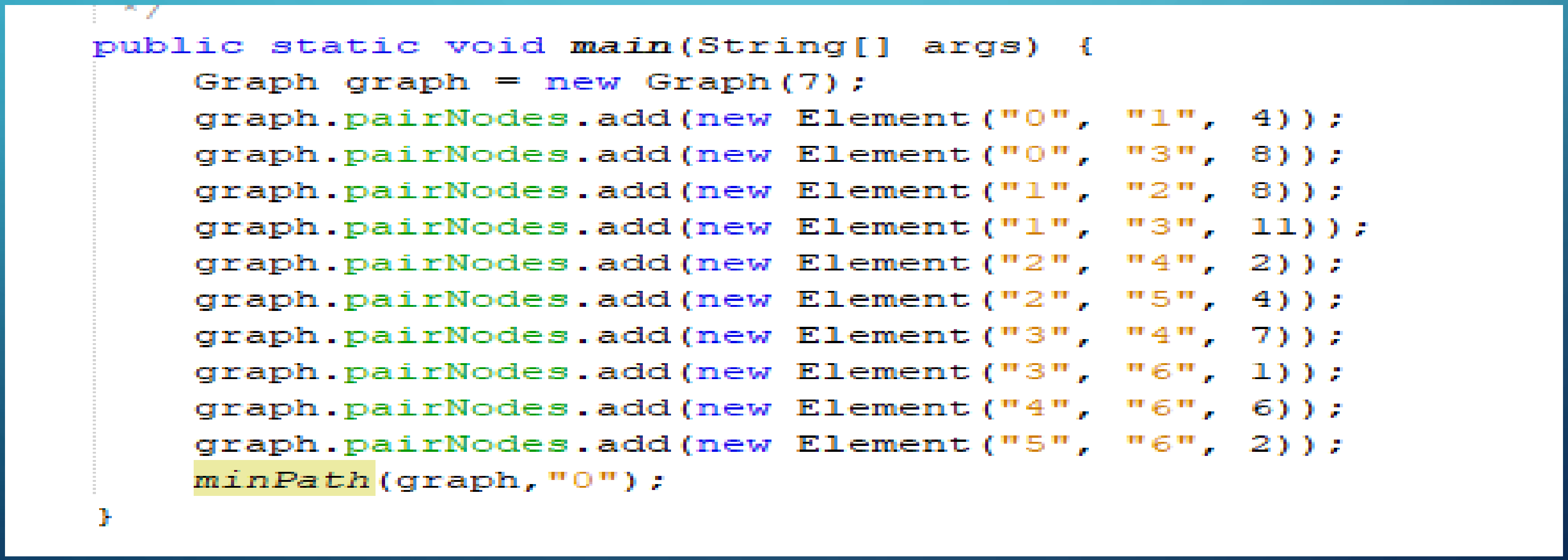


```java
public static void minPath(Graph graph,String source)
{
    PriorityQueue queue = new PriorityQueue();
    String lastNode = source;
    int nbOfNodes = graph.nbOfNodes;
    HashMap result = new HashMap();
    result.put(source, 0);
    int cost=0;
```

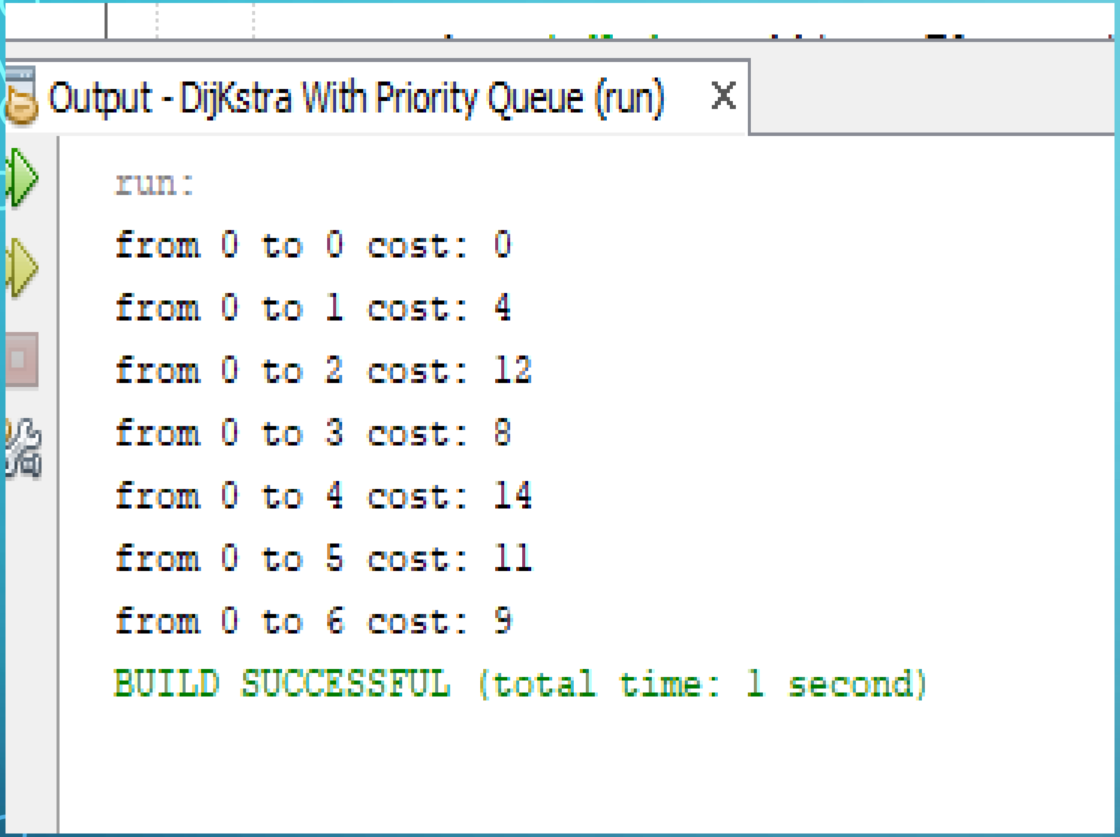


```java
while(result.size()!=nbOfNodes)
{
    for(int i=0;i<graph.pairNodes.size();i++)
    {
        String nextPairNodeSource=graph.pairNodes.get(i).From;
        if(nextPairNodeSource.equals(lastNode))
        {
            String nextPairNodeDest = graph.pairNodes.get(i).to;
            queue.add(new Element(lastNode,nextPairNodeDest,graph.pairNodes.get(i).cost+cost));
        }
        nextPairNodeSource=graph.pairNodes.get(i).to;
        if(nextPairNodeSource.equals(lastNode))
        {
            String nextPairNodeDest = graph.pairNodes.get(i).From;
            queue.add(new Element(lastNode,nextPairNodeDest,graph.pairNodes.get(i).cost+cost));
        }
    }
    Element element = queue.pop();
    if(result.get(element.to)==null)
    {
        result.put(element.to, element.cost);
    }
    lastNode = element.to;
    cost=element.cost;
}
result.forEach((k, v) -> {
    System.out.format("from %s to %s cost: %d\n", source,k, v);
});
```





```
  */
public static void main(String[] args) {
    Graph graph = new Graph(7);
    graph.pairNodes.add(new Element("0", "1", 4));
    graph.pairNodes.add(new Element("0", "3", 8));
    graph.pairNodes.add(new Element("1", "2", 8));
    graph.pairNodes.add(new Element("1", "3", 11));
    graph.pairNodes.add(new Element("2", "4", 2));
    graph.pairNodes.add(new Element("2", "5", 4));
    graph.pairNodes.add(new Element("3", "4", 7));
    graph.pairNodes.add(new Element("3", "6", 1));
    graph.pairNodes.add(new Element("4", "6", 6));
    graph.pairNodes.add(new Element("5", "6", 2));
    minPath(graph,"0");
}
```



```
run:
from 0 to 0 cost: 0
from 0 to 1 cost: 4
from 0 to 2 cost: 12
from 0 to 3 cost: 8
from 0 to 4 cost: 14
from 0 to 5 cost: 11
from 0 to 6 cost: 9
BUILD SUCCESSFUL (total time: 1 second)
```

From 0 To	Min Cost
0	0
1	4
2	12
3	8
4	14
5	11
6	9

END