Luiss
Libera Università Internazionale degli Studi Sociali Guido Carli

Algorithms A.Y. 2022/2023

Lab – Graphs exercises

Irene Finocchi, Flavio Giorgi, Bardh Prenkaj
finocchi@luiss.it, fgiorgi@luiss.it, bprenkaj@luiss.it©

2 May 2023

LUISS

Dipartimento di Impresa e Management

Graphs Exercises

Given a **non-direct Graph** *G=(V, E)*, a **node v** ∈ V and an **integer k** count how many nodes are at a distance smaller or equal than **k from the source node v**. Note that v is at distance 0 from itself!

Graphs Exercises

To solve the exercise we can exploit an algorithm used to explore graphs…

Graphs Exercises

To solve the exercise we can exploit an algorithm used to explore graphs…

The BFS algorithm

Graphs Exercises

```
BFS (G, s)

    let Q be queue.
    Q.enqueue( s )
    mark s as visited.

    while ( Q is not empty)

        v  =  Q.dequeue( )

        for all neighbours w of v in Graph G

            if w is not visited

                Q.enqueue( w )
                mark w as visited
```

Graphs Exercises

```
BFS (G, s)
    let Q be queue.
    Q.enqueue( s )
    mark s as visited.

    while ( Q is not empty)

        v  =  Q.dequeue( )

        for all neighbours w of v in Graph G

            if w is not visited

                Q.enqueue( w )
                mark w as visited
```

Queue initialization

Graphs Exercises

```
BFS (G, s)
    let Q be queue.
    Q.enqueue( s )
    mark s as visited.

    while ( Q is not empty)
        v = Q.dequeue( )
        for all neighbours w of v in Graph G
            if w is not visited
                Q.enqueue( w )
                mark w as visited
```

Source node first element in the queue

Graphs Exercises

```
BFS (G, s)
    let Q be queue.
    Q.enqueue( s )
    mark s as visited.
    while ( Q is not empty)
        v  =  Q.dequeue( )
        for all neighbours w of v in Graph G
            if w is not visited
                Q.enqueue( w )
                mark w as visited
```

Source node set as visited

Graphs Exercises

```
BFS (G, s)
    let Q be queue.
    Q.enqueue( s )
    mark s as visited.

    while ( Q is not empty)
        v  =  Q.dequeue( )
        for all neighbours w of v in Graph G
            if w is not visited
                Q.enqueue( w )
                mark w as visited
```

While loop to explore all the nodes

Graphs Exercises

```
BFS (G, s)
    let Q be queue.
    Q.enqueue( s )
    mark s as visited.

    while ( Q is not empty)
        v  =  Q.dequeue( )
        for all neighbours w of v in Graph G
            if w is not visited
                Q.enqueue( w )
                mark w as visited
```

Take the first node of the queue out

Graphs Exercises

```
BFS (G, s)

    let Q be queue.
    Q.enqueue( s )
    mark s as visited.

    while ( Q is not empty)

        v  =  Q.dequeue( )

        for all neighbours w of v in Graph G      ← Explore all the neighborhoods of v

            if w is not visited

                Q.enqueue( w )
                mark w as visited
```

Graphs Exercises

```
BFS (G, s)

    let Q be queue.
    Q.enqueue( s )
    mark s as visited.

    while ( Q is not empty)

        v  =  Q.dequeue( )

        for all neighbours w of v in Graph G

            if w is not visited

                Q.enqueue( w )
                mark w as visited
```

If the node has not been visited

Graphs Exercises

```
BFS (G, s)
    let Q be queue.
    Q.enqueue( s )
    mark s as visited.

    while ( Q is not empty)
        v  =  Q.dequeue( )
        for all neighbours w of v in Graph G
            if w is not visited
                Q.enqueue( w )
                mark w as visited
```

Put it in the queue

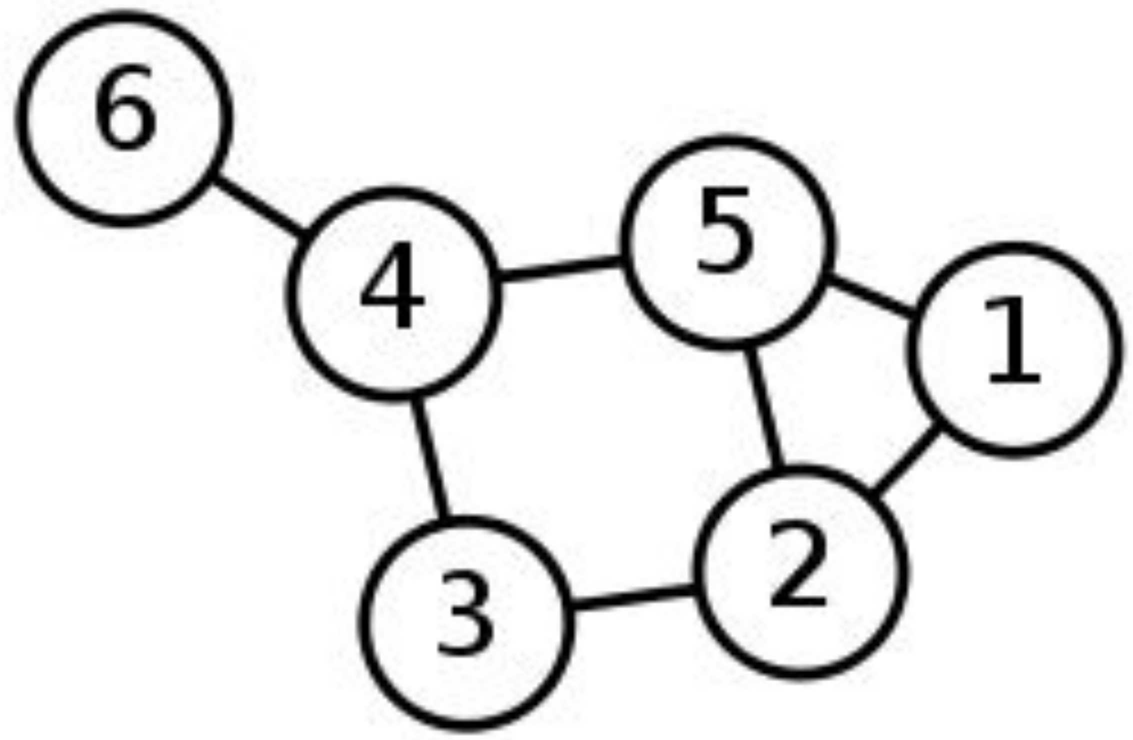
Graphs Exercises

```
BFS (G, s)

    let Q be queue.
    Q.enqueue( s )
    mark s as visited.

    while ( Q is not empty)

        v  =  Q.dequeue( )

        for all neighbours w of v in Graph G

            if w is not visited

                Q.enqueue( w )
                mark w as visited
```

Mark it as visited

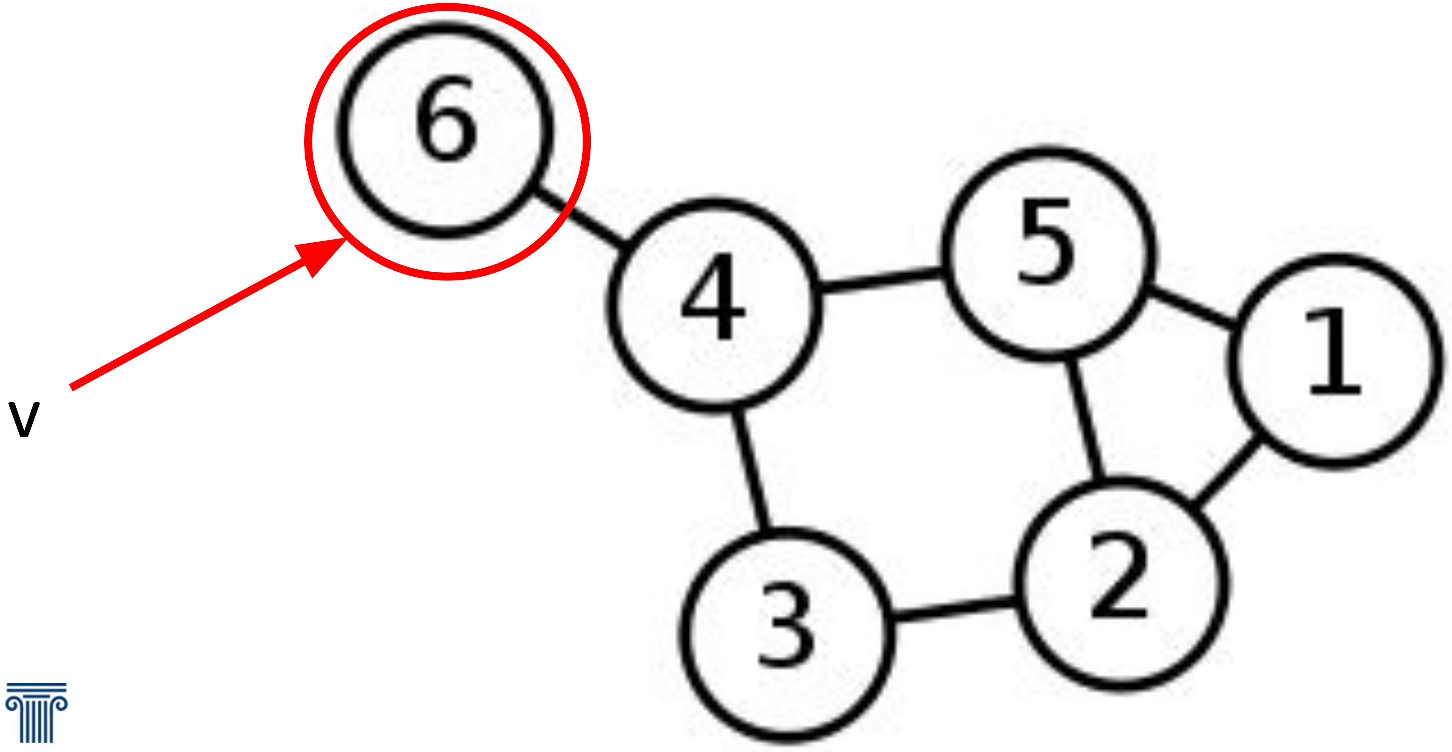
Graphs Exercises

Given a **non-direct Graph** *G=(V, E)*, a **node v** ∈ V and an **integer k** count how many nodes are at a distance smaller or equal than **k from the source node v**. <u>Note that v is at distance 0 from itself!</u>



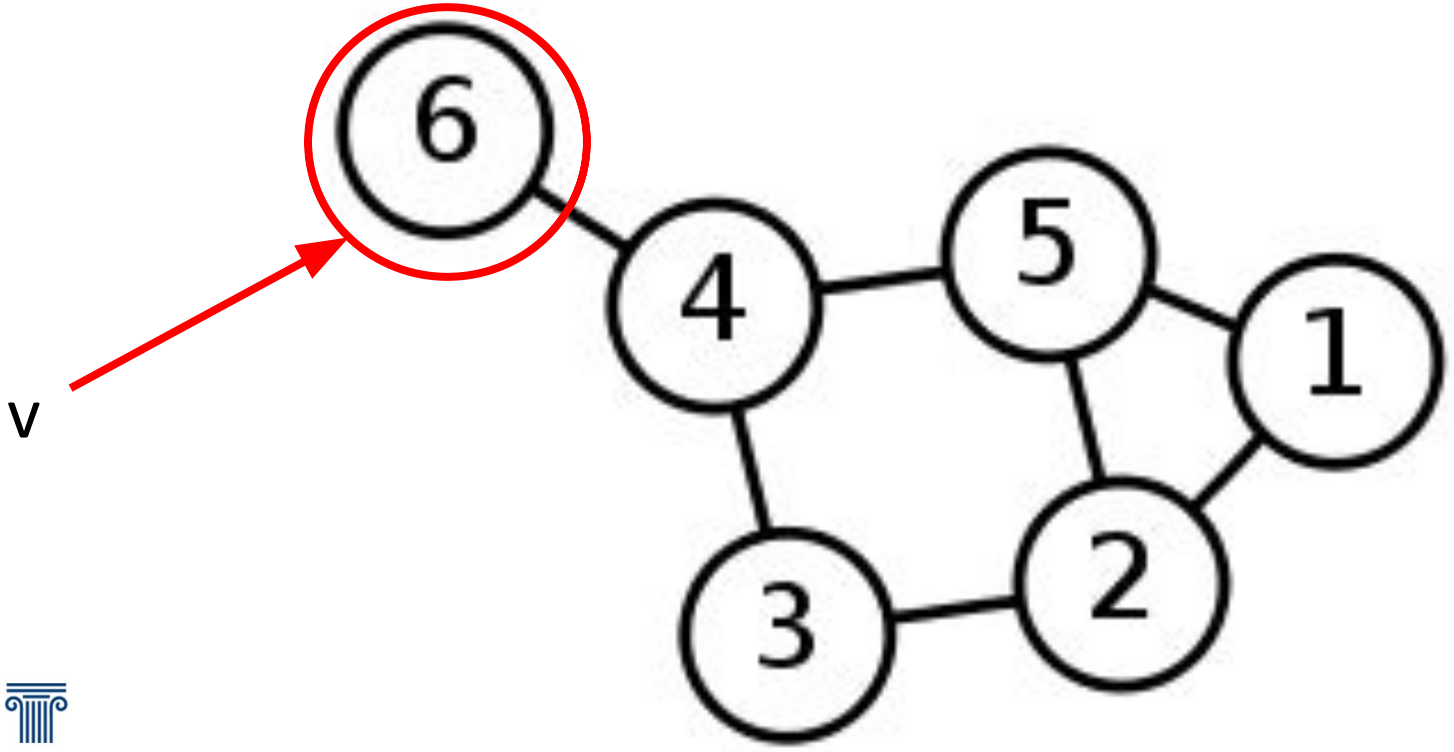
Graphs Exercises

Given a **non-direct Graph** *G=(V, E)*, a **node v** ∈ V and an **integer k** count how many nodes are at a distance smaller or equal than **k from the source node v**. Note that v is at distance 0 from itself!



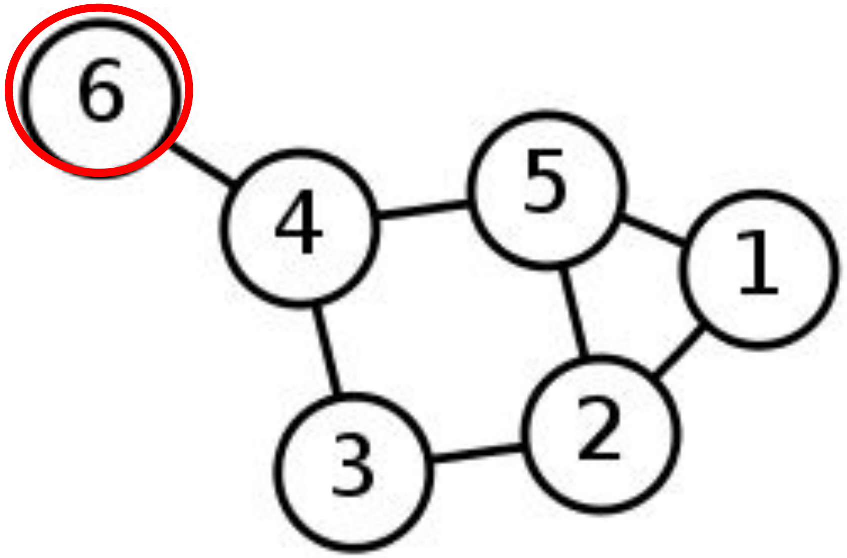
Graphs Exercises

Given a **non-direct Graph** *G=(V, E)*, a **node v** ∈ V and an **integer k** count how many nodes are at a distance smaller or equal than **k from the source node v**. <u>Note that v is at distance 0 from itself!</u>



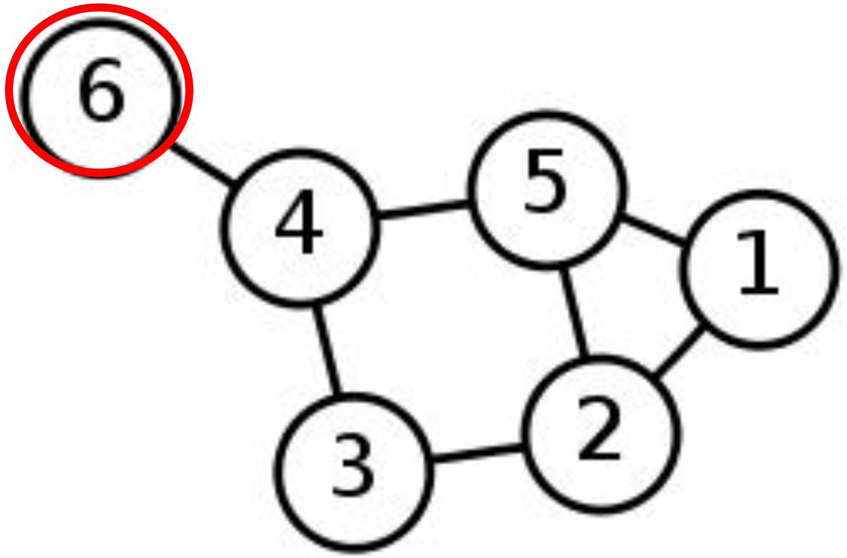
k = 2

Graphs Exercises



k = 2

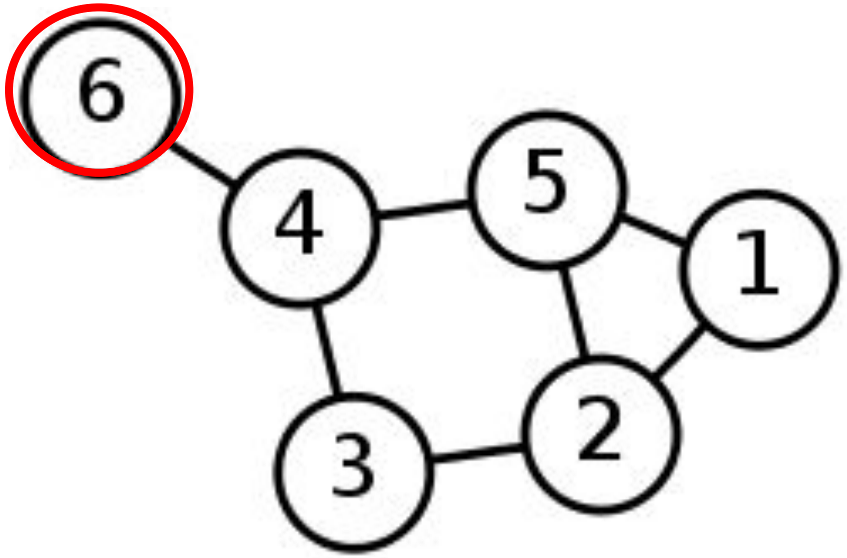
Graphs Exercises



k = 2

Level 0

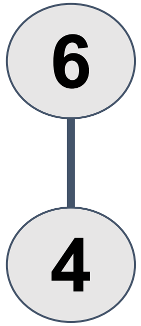
Graphs Exercises



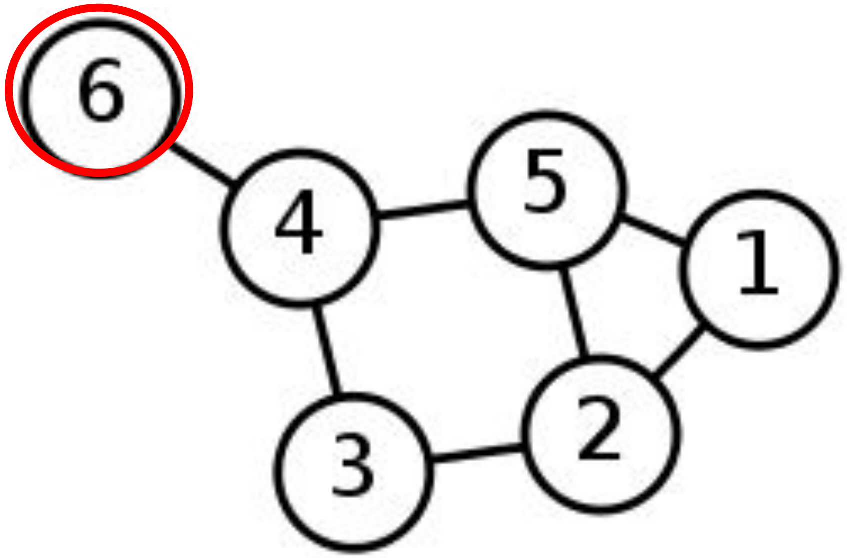
k = 2

Level 0

Level 1



Graphs Exercises



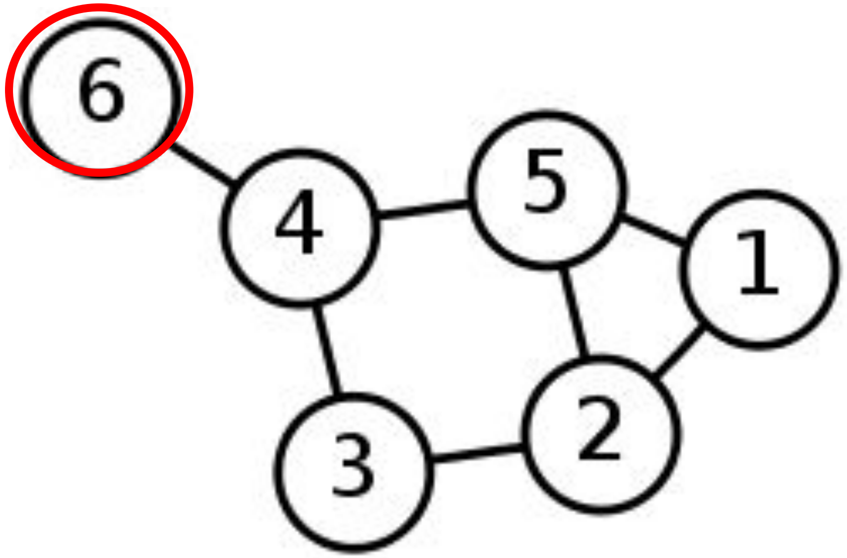
k = 2

Level 0

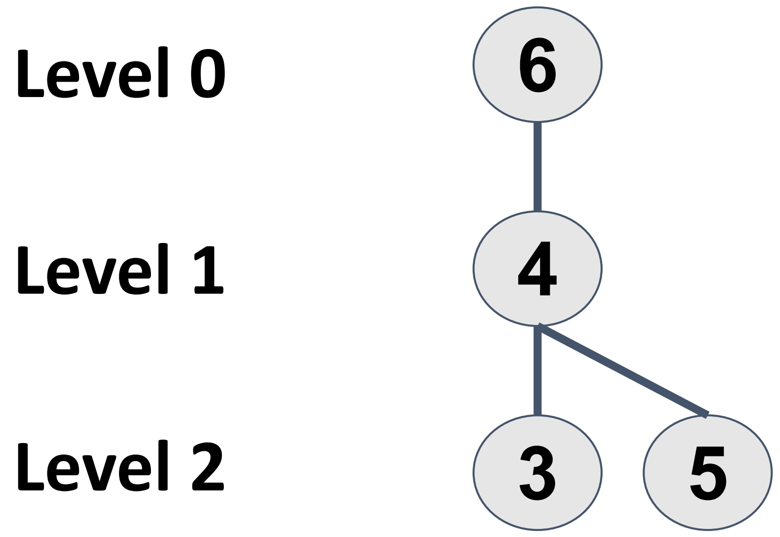
Level 1

Level 2

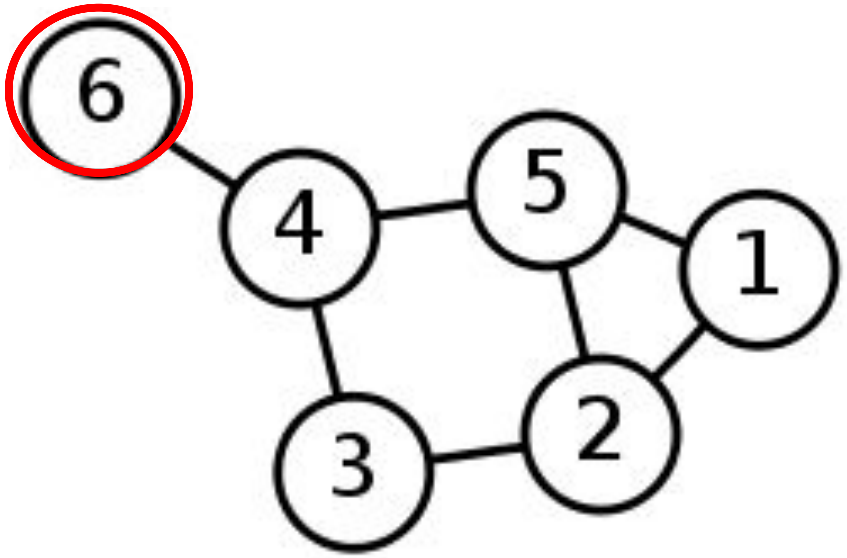
Graphs Exercises



k = 2



Graphs Exercises

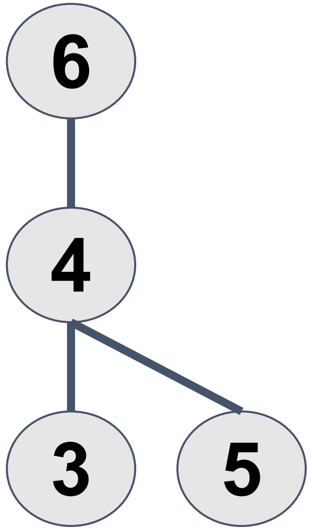


k = 2

Level 0

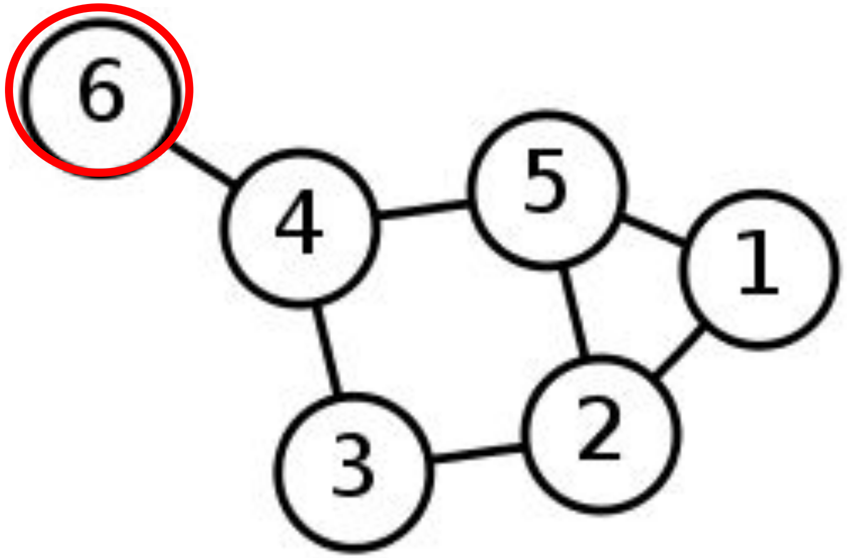
Level 1

Level 2



The answer is?

Graphs Exercises

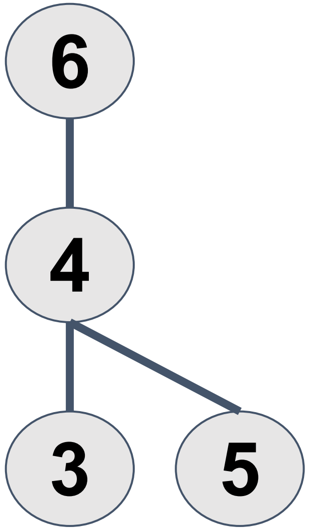


k = 2

Level 0

Level 1

Level 2



The answer is? 4

Graphs Exercises

```
BFS (G, s)
    let Q be queue.
    Q.enqueue( s )
    mark s as visited.

    while ( Q is not empty)

        v  =  Q.dequeue( )

        for all neighbours w of v in Graph G

            if w is not visited

                Q.enqueue( w )
                mark w as visited
```

We have to modify the pseudocode to make it works! How can we do that?

Graphs Exercises

```
BFS (G, s)
    node_count = 1
    let Q be queue.
    Q.enqueue( (s, 0) )
    mark s as visited.

    while ( Q is not empty)
        v, level  =  Q.dequeue( )

        if level > k
            break

       for all neighbours w of v in Graph G

           if w is not visited

               Q.enqueue( (w, level+1) )
               mark w as visited
               node_count += 1
```

We have to modify the pseudocode to make it works! How can we do that?

LUISS

Graphs Exercises

What about the BFS using the adjacency matrix?

Graphs Exercises

```
def BFS(self, start):
    visited = [False] * self.v
    q = [start]
    visited[start] = True
    while q:
        vis = q[0]
        q.pop(0)
        for i in range(self.v):
            if (Graph.adj[vis][i] == 1 and
                (not visited[i])):
                    q.append(i)
                    visited[i] = True
```

Build a list of length |V| with every entry equal to False

Graphs Exercises

```
def BFS(self, start):
    visited = [False] * self.v
    q = [start]
    visited[start] = True
    while q:
        vis = q[0]
        q.pop(0)
        for i in range(self.v):
            if (Graph.adj[vis][i] == 1 and
                (not visited[i])):
                    q.append(i)
                    visited[i] = True
```

Put the starting node in the queue

Graphs Exercises

```
def BFS(self, start):
    visited = [False] * self.v
    q = [start]
    visited[start] = True
    while q:
        vis = q[0]
        q.pop(0)
        for i in range(self.v):
            if (Graph.adj[vis][i] == 1 and
                (not visited[i])):
                    q.append(i)
                    visited[i] = True
```

We set the list at the position "start" equal to true

Graphs Exercises

```
def BFS(self, start):
    visited = [False] * self.v
    q = [start]
    visited[start] = True
    while q:
        vis = q[0]
        q.pop(0)
        for i in range(self.v):
            if (Graph.adj[vis][i] == 1 and
                (not visited[i])):
                    q.append(i)
                    visited[i] = True
```

Then we start exploring the nodes in the queue

Graphs Exercises

```
def BFS(self, start):
    visited = [False] * self.v
    q = [start]
    visited[start] = True
    while q:
        vis = q[0]
        q.pop(0)
        for i in range(self.v):
            if (Graph.adj[vis][i] == 1 and
                (not visited[i])):
                    q.append(i)
                    visited[i] = True
```

We extract the first node in the queue (in this case an integer)

Graphs Exercises

```
def BFS(self, start):
    visited = [False] * self.v
    q = [start]
    visited[start] = True
    while q:
        vis = q[0]
        q.pop(0)
        for i in range(self.v):
            if (Graph.adj[vis][i] == 1 and
                (not visited[i])):
                    q.append(i)
                    visited[i] = True
```

Then we delete the element

Graphs Exercises

```
def BFS(self, start):
    visited = [False] * self.v
    q = [start]
    visited[start] = True
    while q:
        vis = q[0]
        q.pop(0)
        for i in range(self.v):
            if (Graph.adj[vis][i] == 1 and
                (not visited[i])):
                    q.append(i)
                    visited[i] = True
```

For each node in the graph

Graphs Exercises

```
def BFS(self, start):
    visited = [False] * self.v
    q = [start]
    visited[start] = True
    while q:
        vis = q[0]
        q.pop(0)
        for i in range(self.v):
            if (Graph.adj[vis][i] == 1 and
                (not visited[i])):
                    q.append(i)
                    visited[i] = True
```

If the node i is adjacent to the current node (vis)

Graphs Exercises

```
def BFS(self, start):
    visited = [False] * self.v
    q = [start]
    visited[start] = True
    while q:
        vis = q[0]
        q.pop(0)
        for i in range(self.v):
            if (Graph.adj[vis][i] == 1 and
                (not visited[i])):
                    q.append(i)
                    visited[i] = True
```

And it is not visited

Graphs Exercises

```
def BFS(self, start):
    visited = [False] * self.v
    q = [start]
    visited[start] = True
    while q:
        vis = q[0]
        q.pop(0)
        for i in range(self.v):
            if (Graph.adj[vis][i] == 1 and
                (not visited[i])):
                    q.append(i)
                    visited[i] = True
```

Append the node to the queue

Graphs Exercises

```
def BFS(self, start):
    visited = [False] * self.v
    q = [start]
    visited[start] = True
    while q:
        vis = q[0]
        q.pop(0)
        for i in range(self.v):
            if (Graph.adj[vis][i] == 1 and
                (not visited[i])):
                    q.append(i)
                    visited[i] = True
```

Set the node "i" as visited

Graphs Exercises

```
def BFS(self, start):
    visited = [False] * self.v
    q = [start]
    visited[start] = True
    while q:
        vis = q[0]
        q.pop(0)
        for i in range(self.v):
            if (Graph.adj[vis][i] == 1 and
             (not visited[i])):
                q.append(i)
                visited[i] = True
```

In this way we explore the adjacency matrix

BFS

Complexity:

$O(|E| + |V|)$

What if the graph is a complete graph?

BFS

Complexity:

$O(|E| + |V|)$

What if the graph is a complete graph?

$O(|V|^2)$