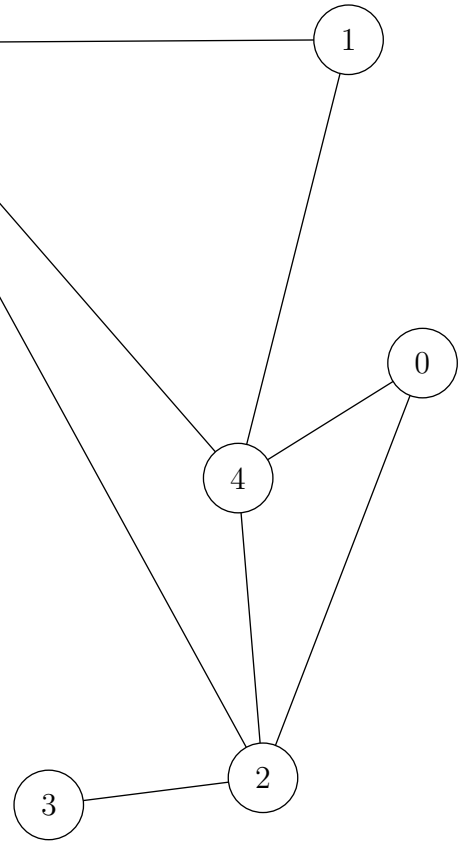
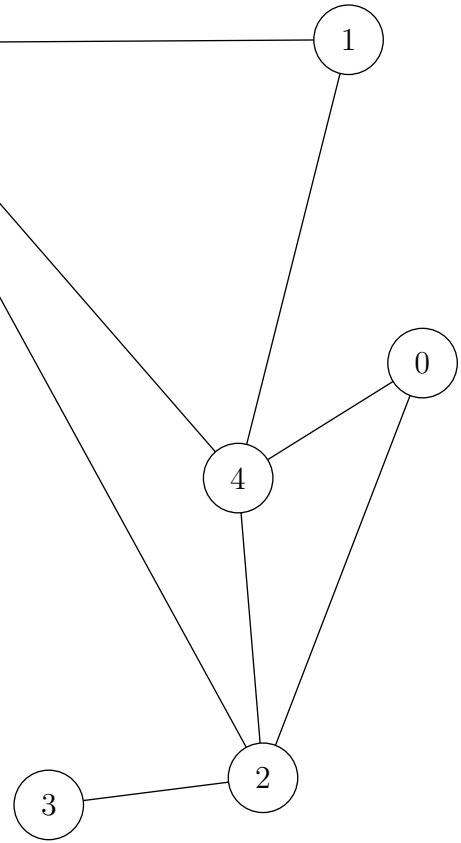
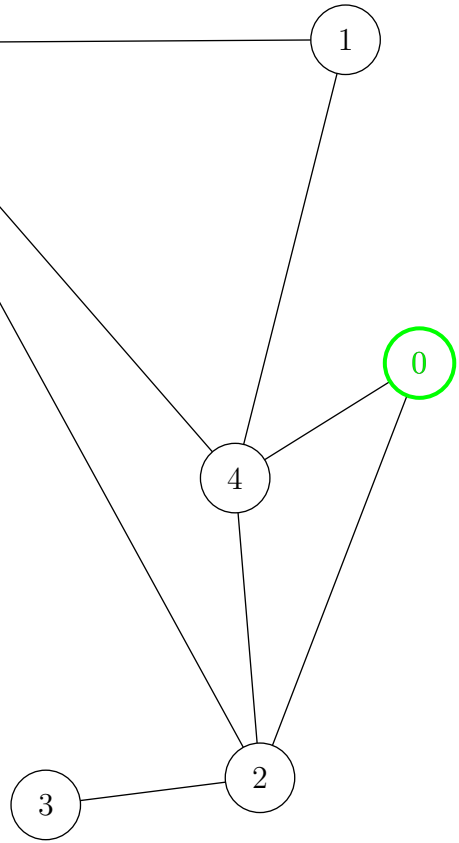
```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

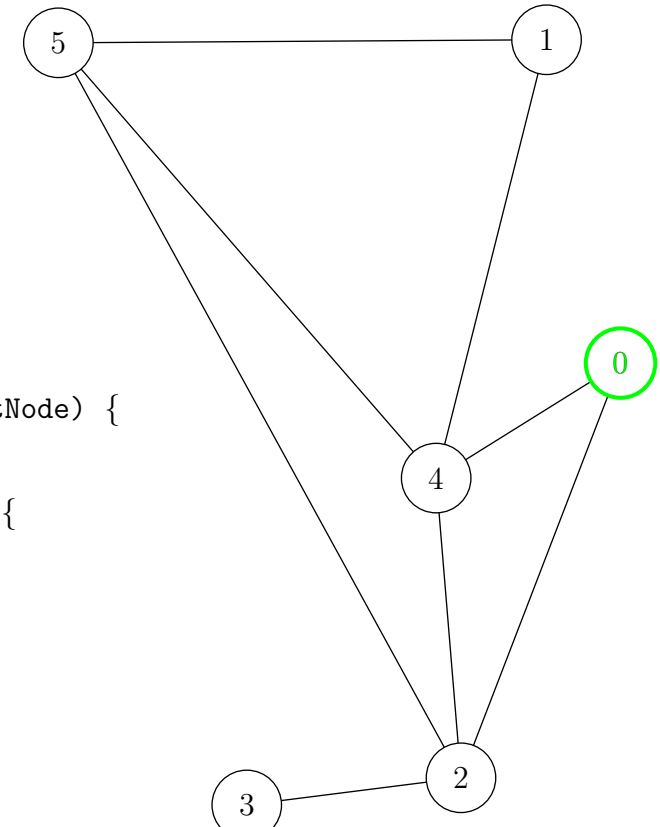


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

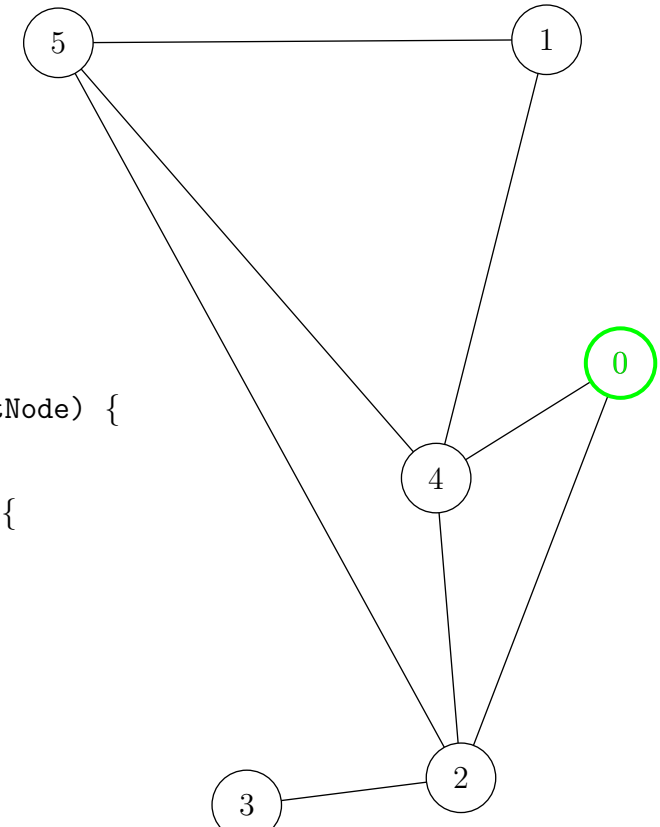


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

q=



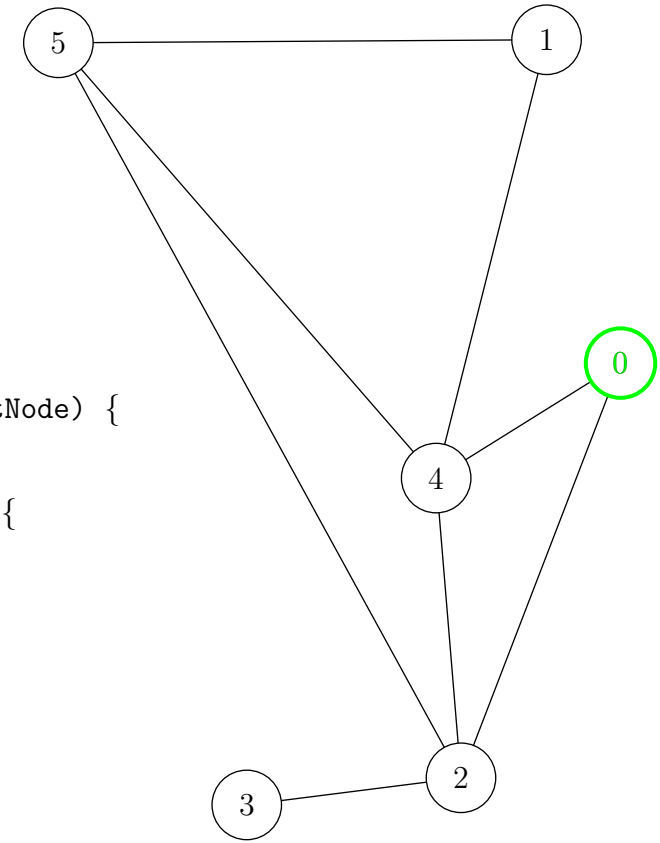
```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```



q=

0					

```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

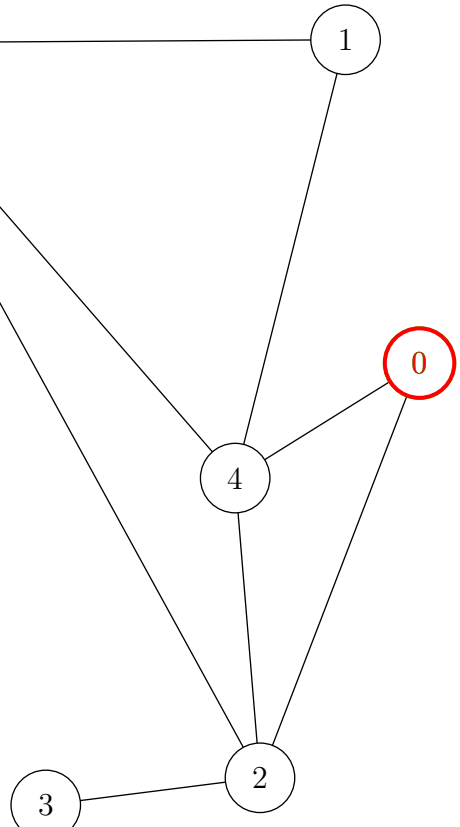


q= | 0 | | | | | |

```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=0

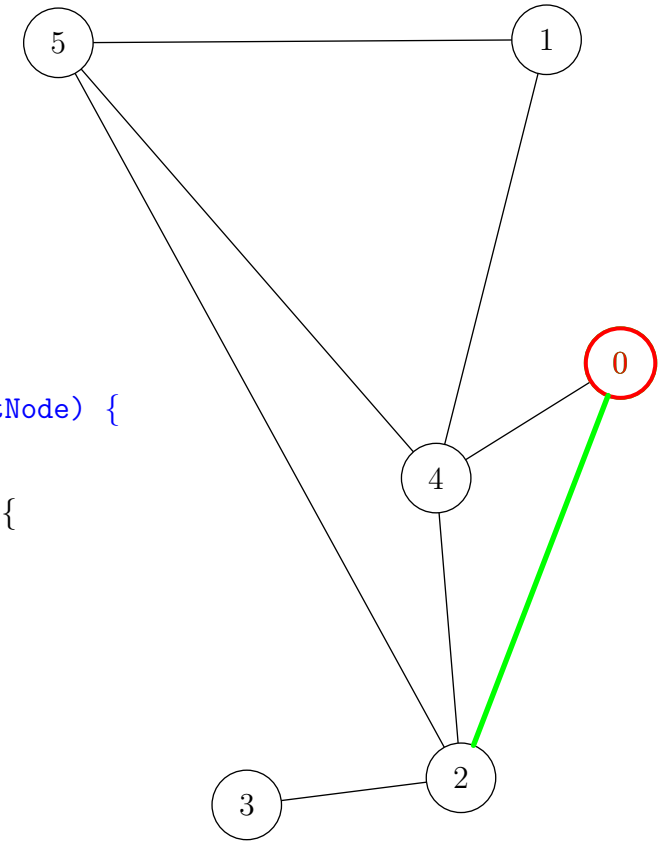
q=



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=0 neighbour=2

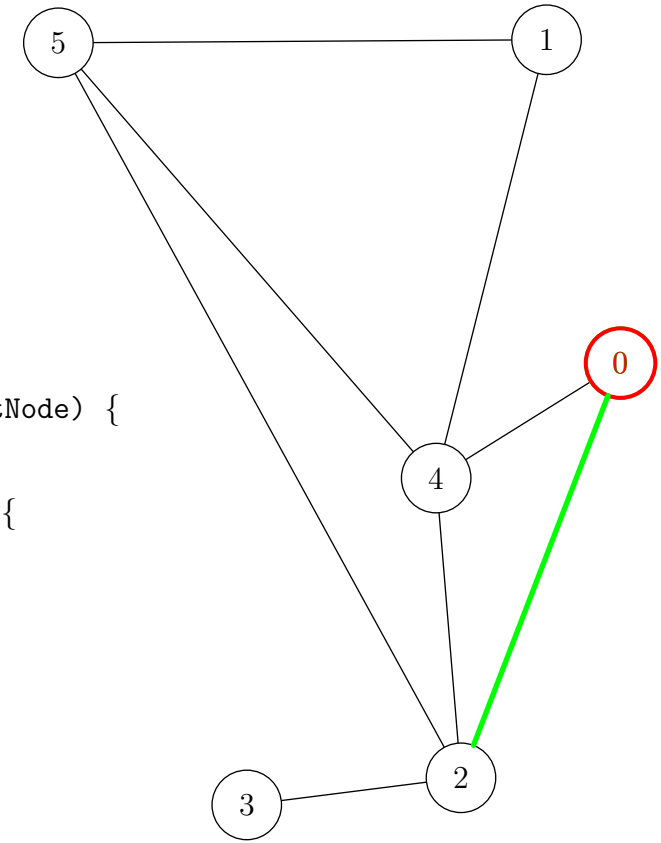
q=



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=0 neighbour=2

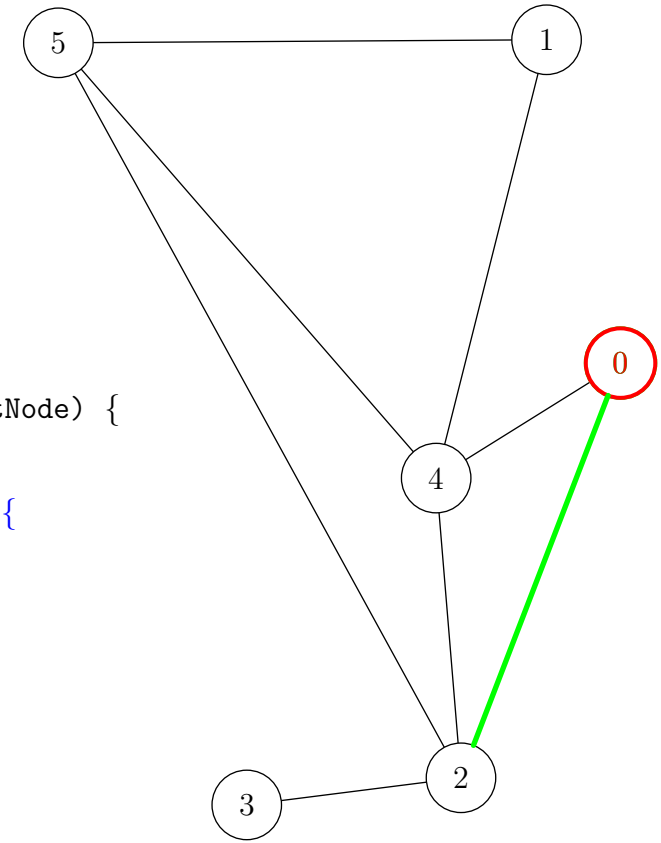
q=



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=0 neighbour=2

q=

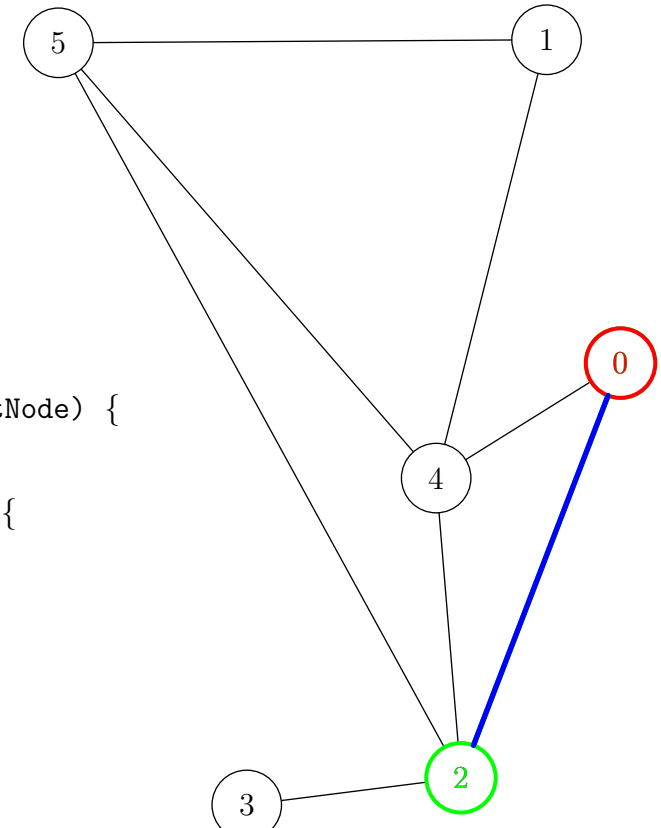


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=0 neighbour=2

q=

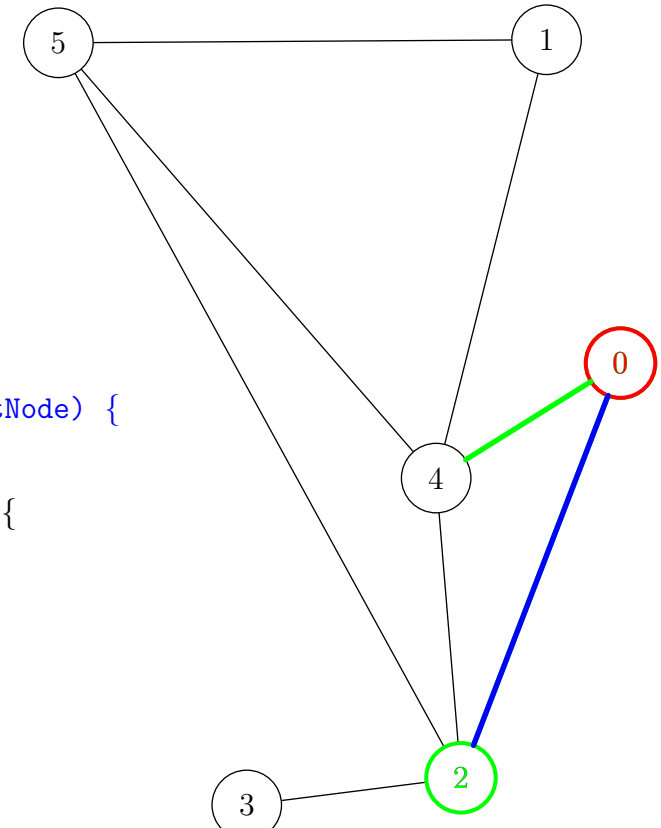
2					



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=0 neighbour=4

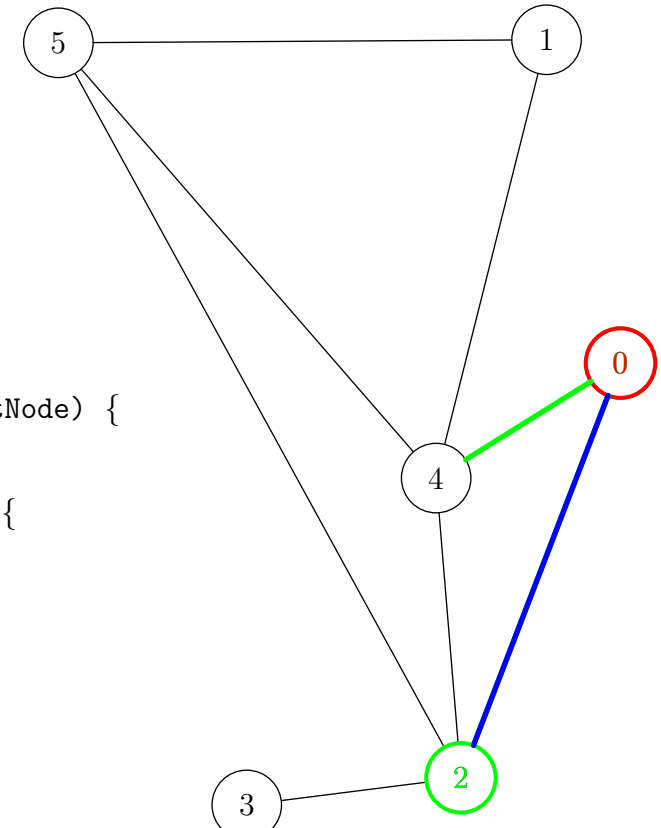
q= | 2 | | | | | |



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=0 neighbour=4

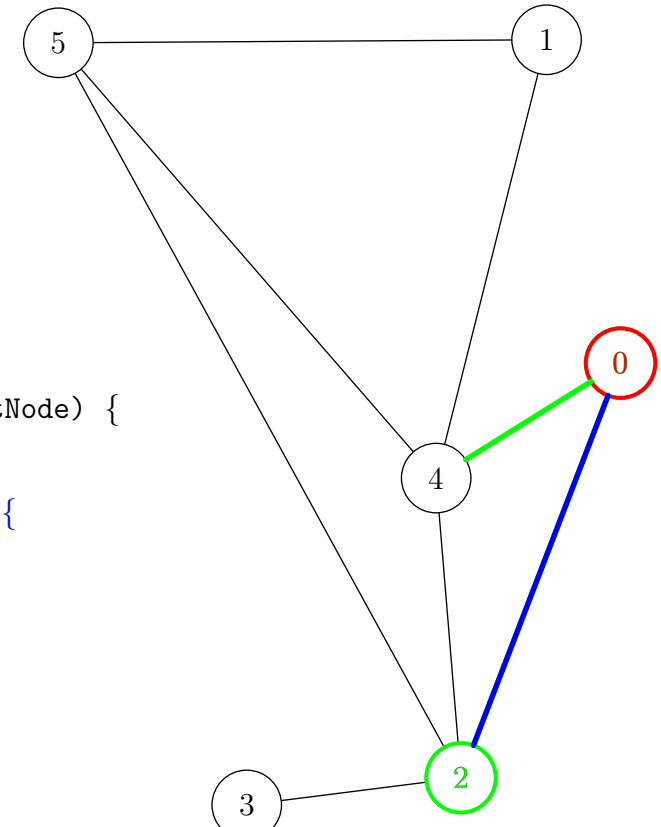
q= | 2 | | | | | |



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=0 neighbour=4

q= | 2 | | | | | |

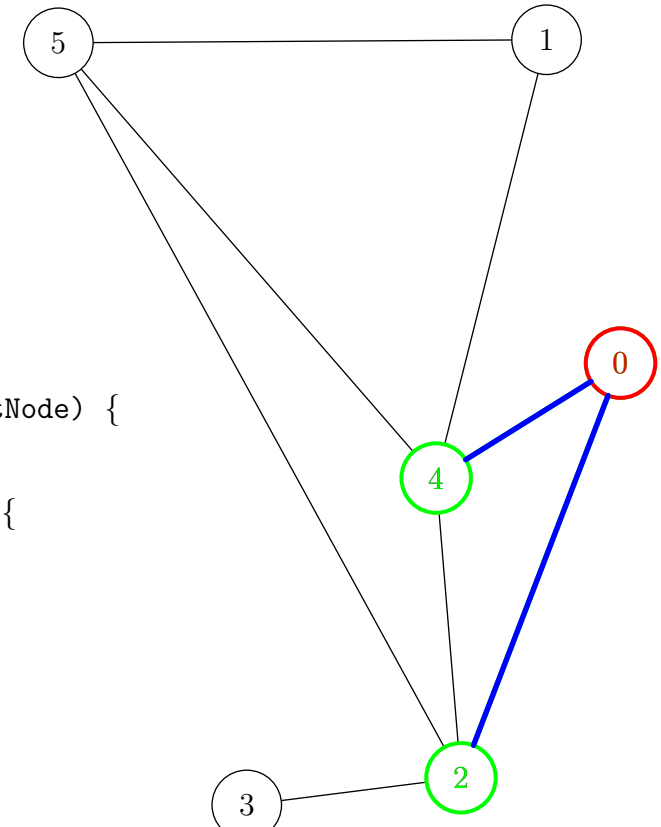


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=0 neighbour=4

q=

2	4				

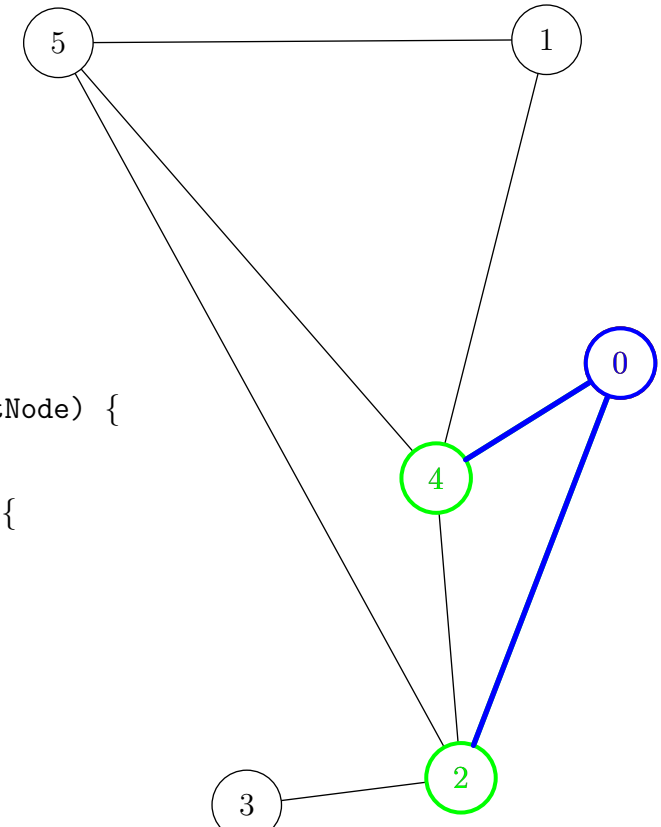


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

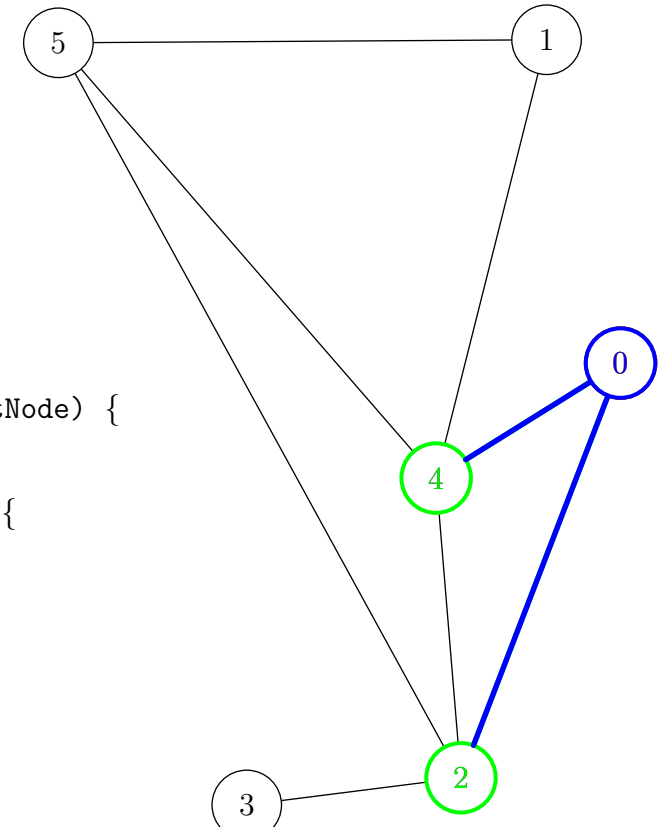
currentNode=0

q=

2	4				



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```



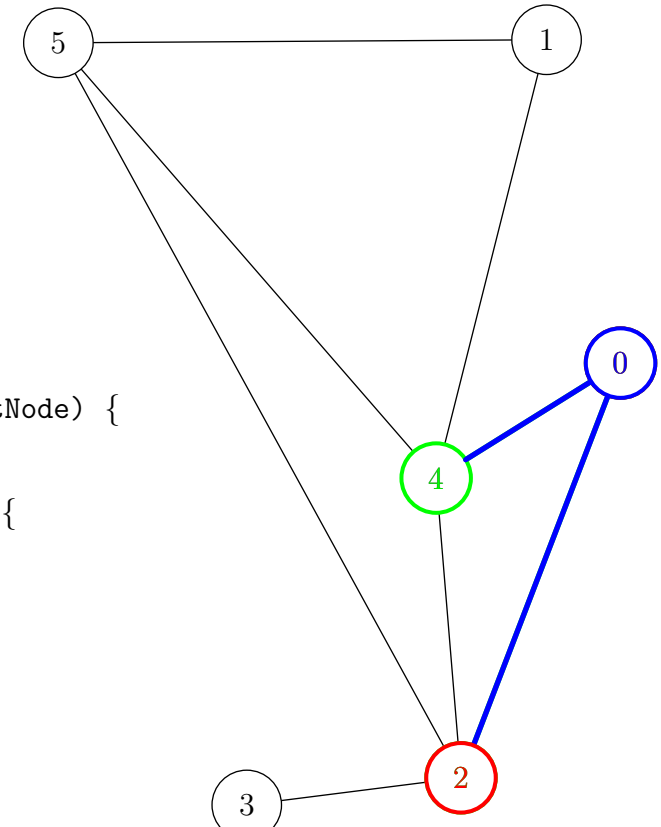
q=

2	4				

```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=2

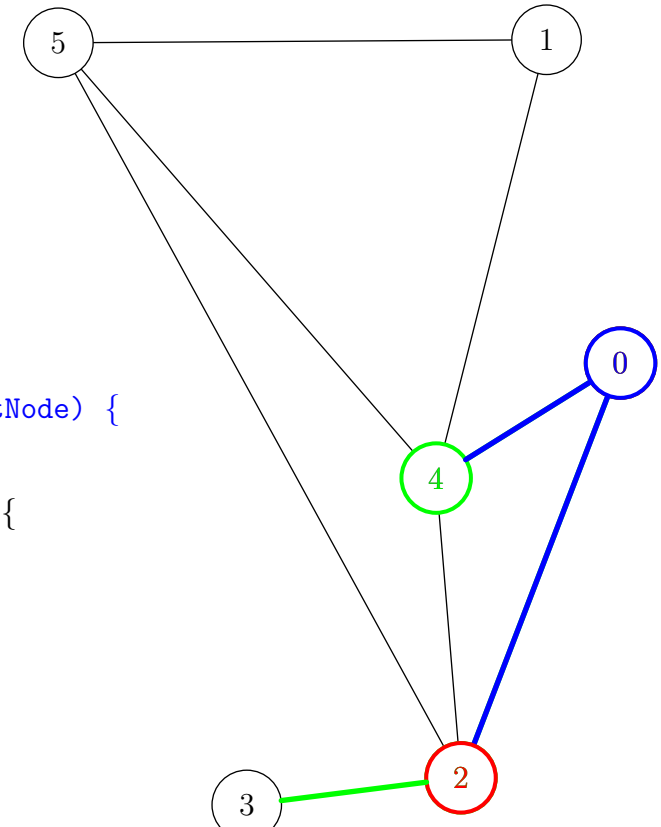
q= | 4 | | | | | |



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=2 neighbour=3

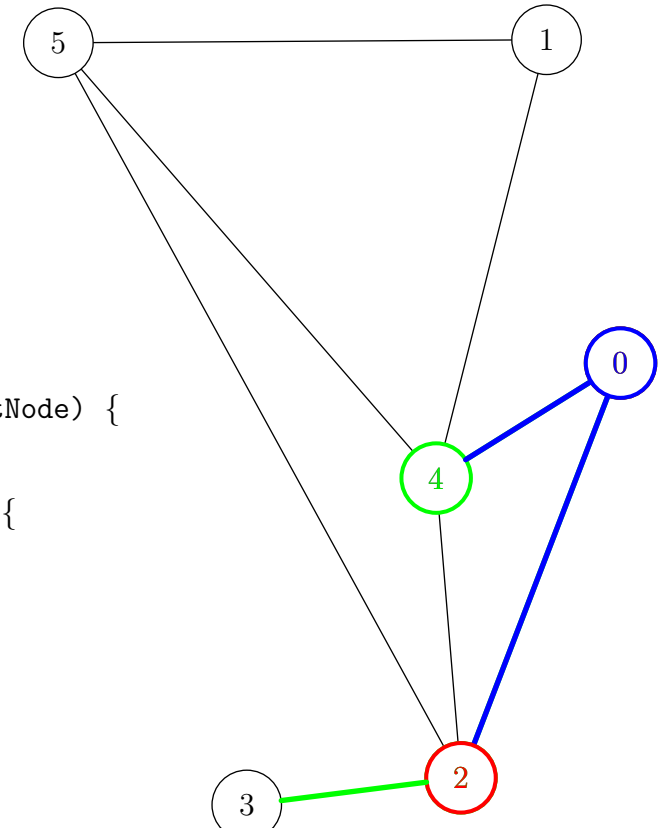
q= | 4 | | | | | |



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=2 neighbour=3

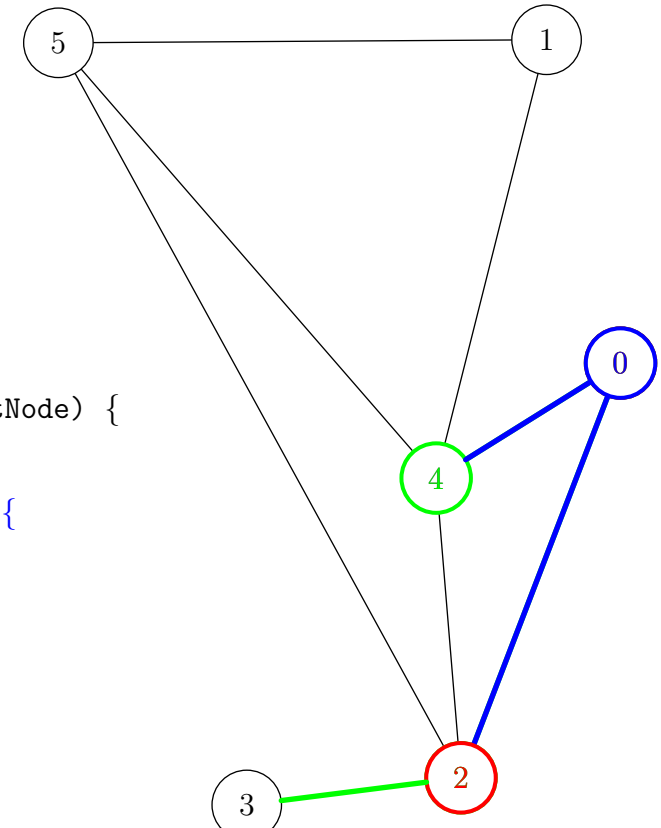
q= | 4 | | | | | |



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=2 neighbour=3

q= | 4 | | | | | |

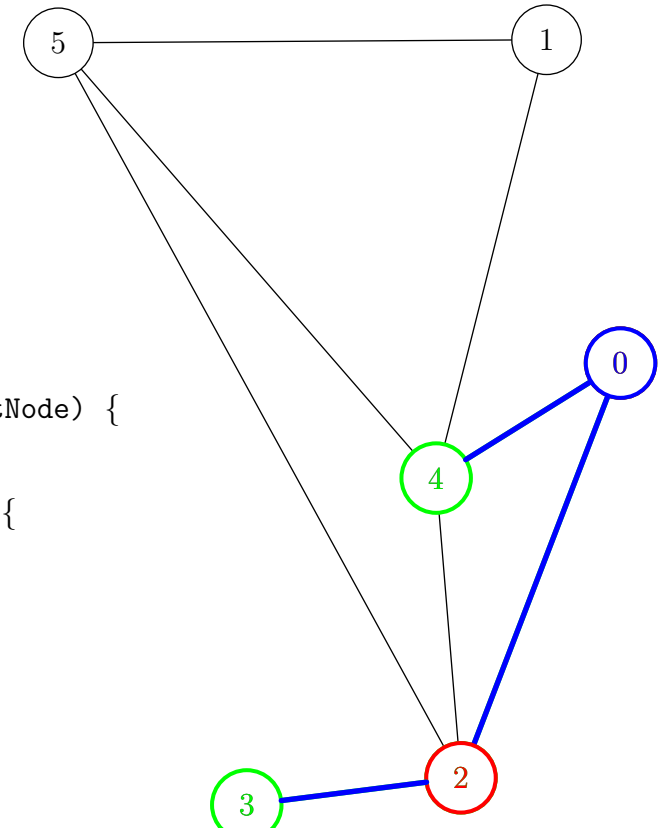


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=2 neighbour=3

q=

4	3				

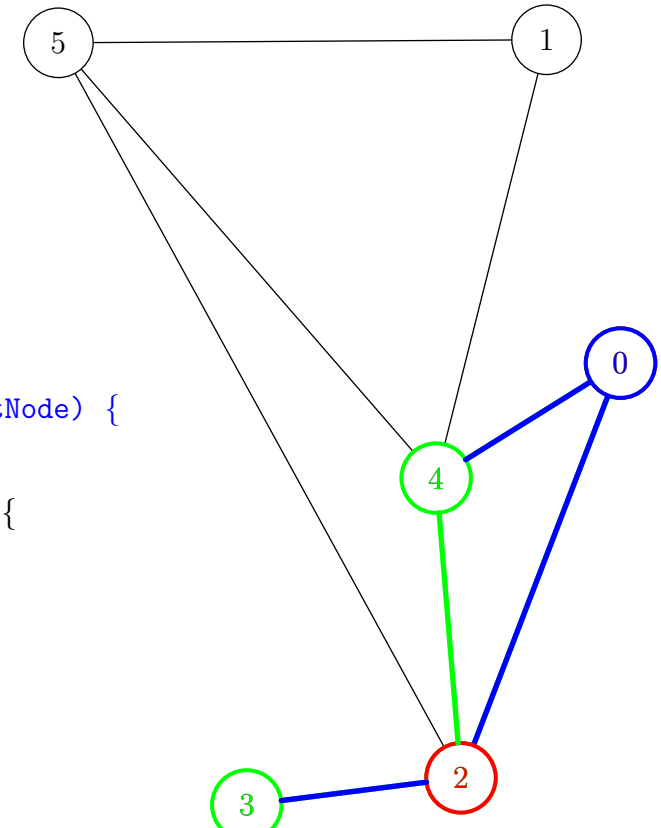


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=2 neighbour=4

q=

4	3				

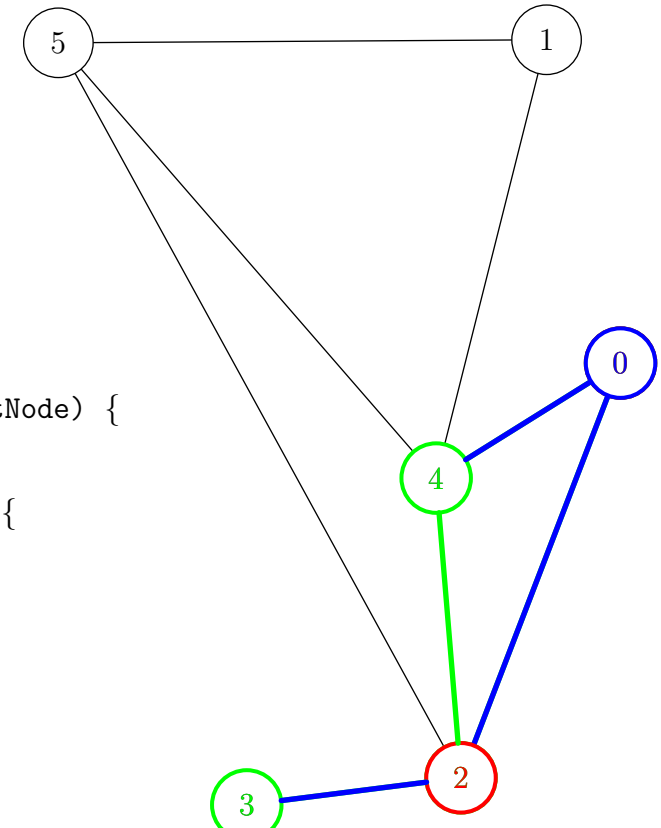


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=2 neighbour=4

q=

4	3				

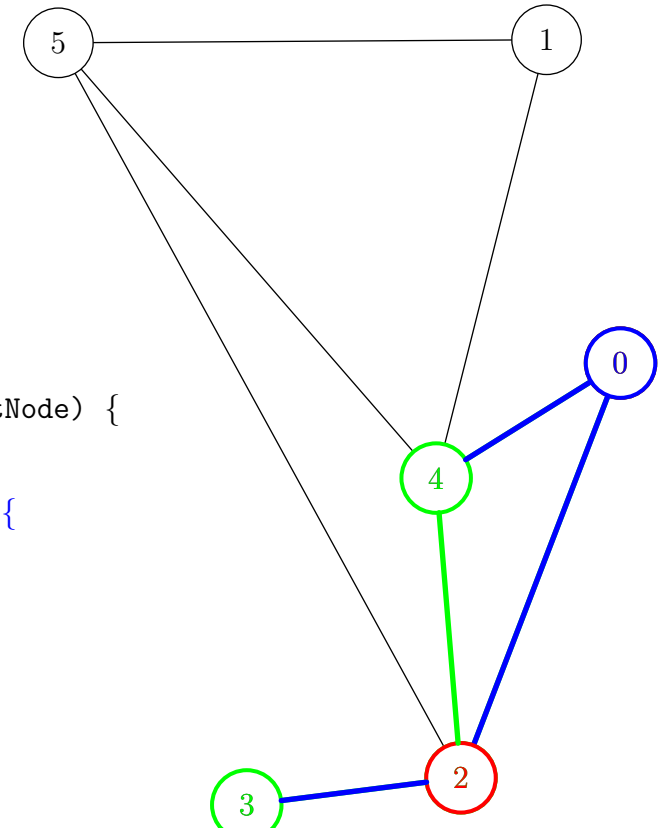


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=2 neighbour=4

q=

4	3				

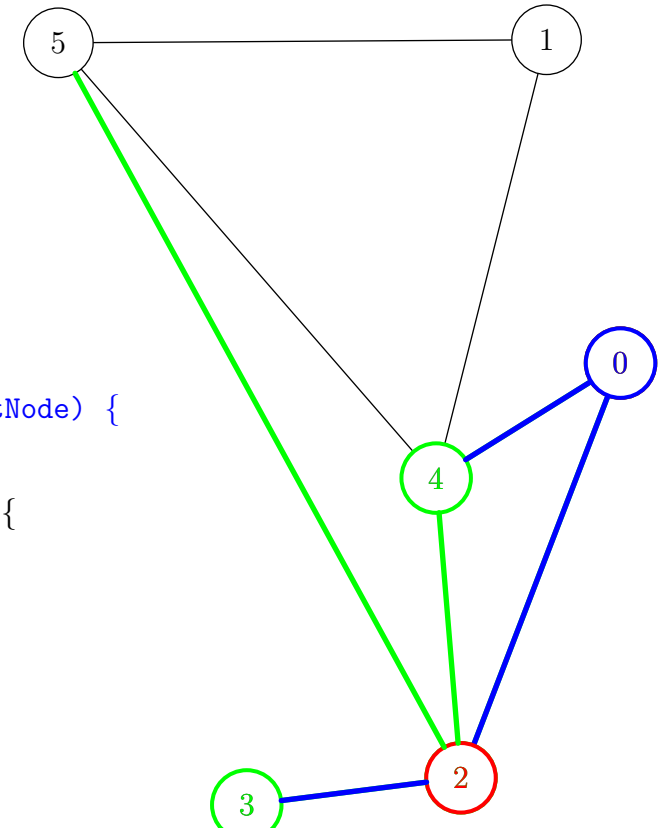


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=2 neighbour=5

q=

4	3				

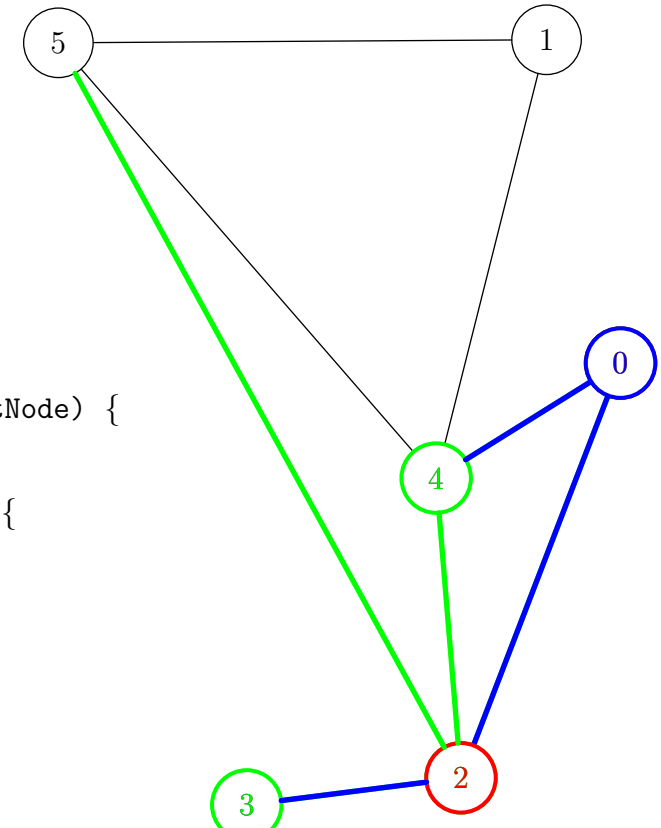


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=2 neighbour=5

q=

4	3				

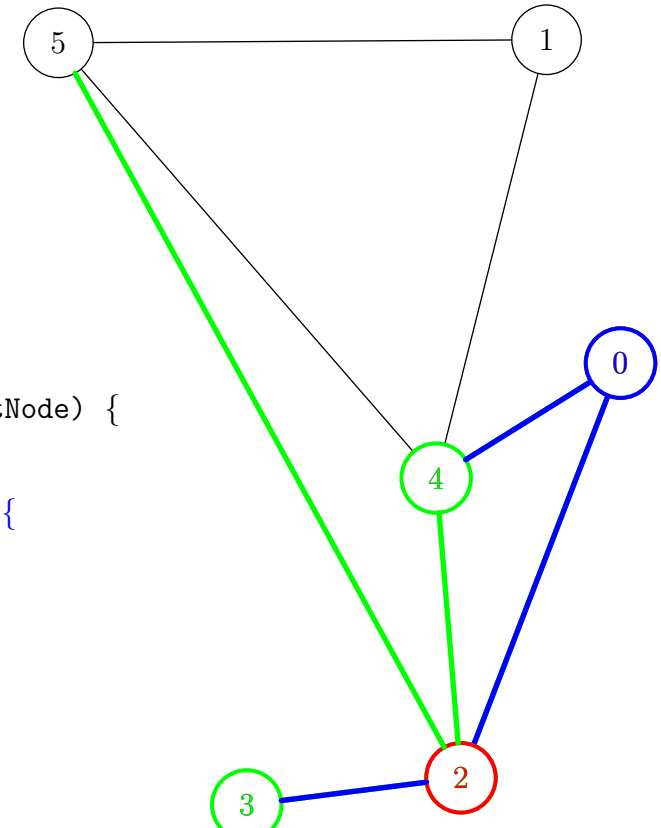


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=2 neighbour=5

q=

4	3				

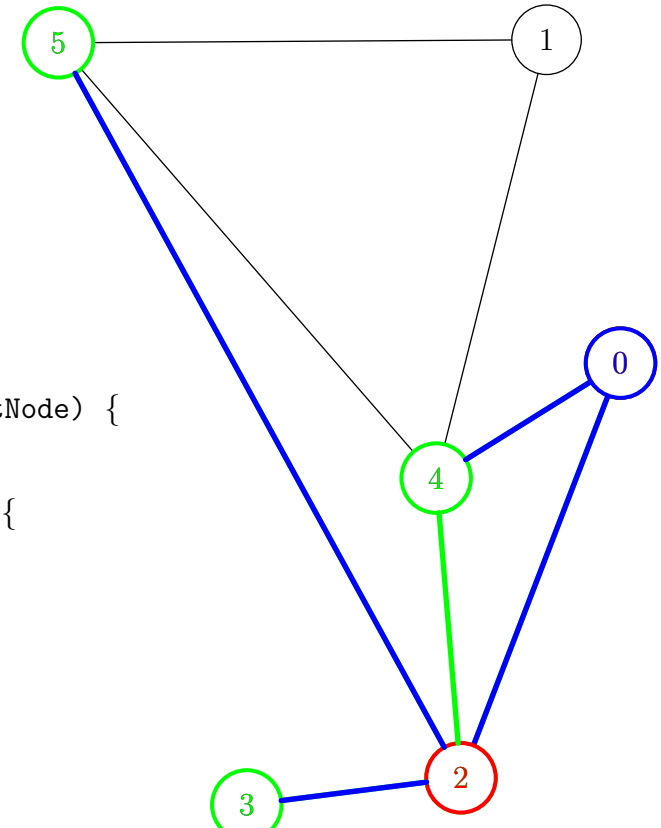


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=2 neighbour=5

q=

4	3	5			

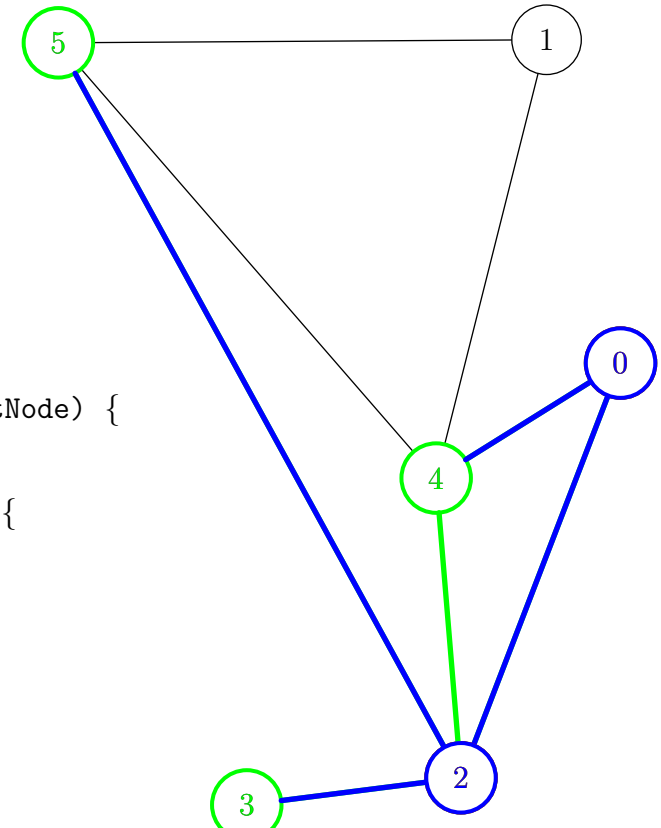


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

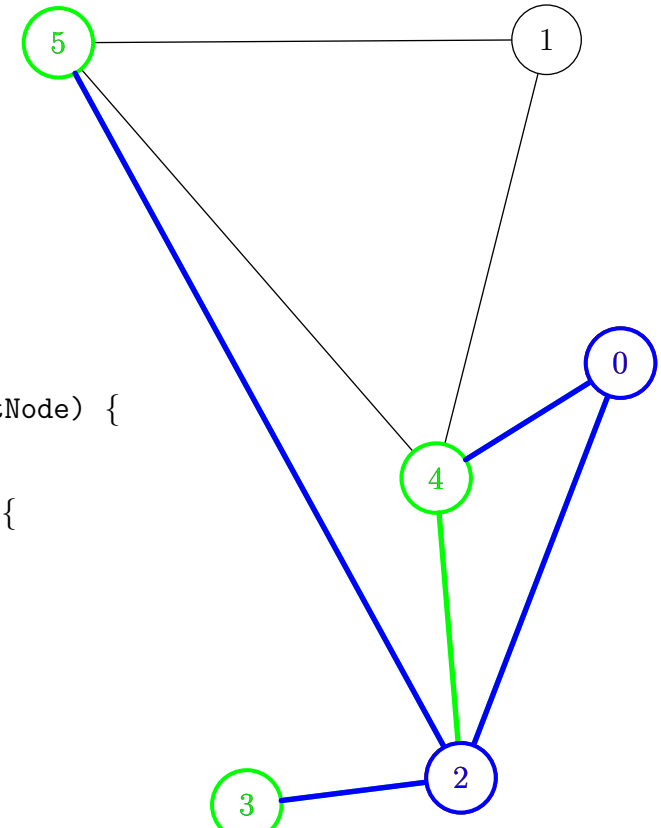
currentNode=2

q=

4	3	5			



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```



q=

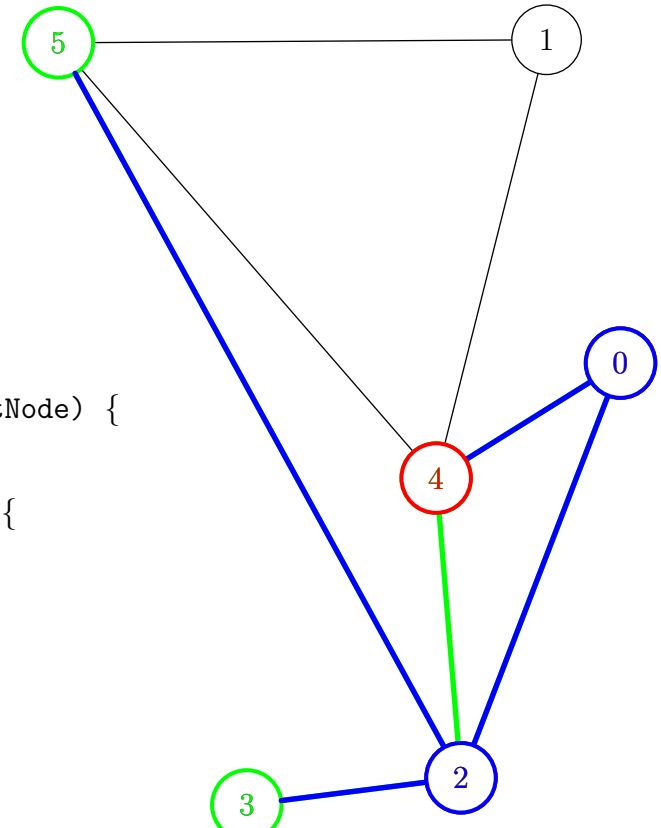
4	3	5			

```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=4

q=

3	5				

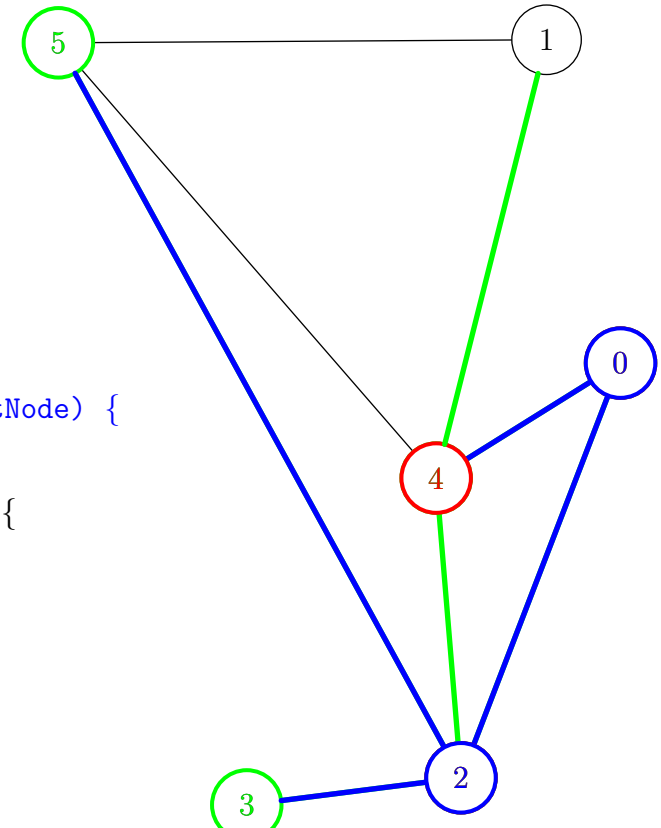


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=4 neighbour=1

q=

3	5				

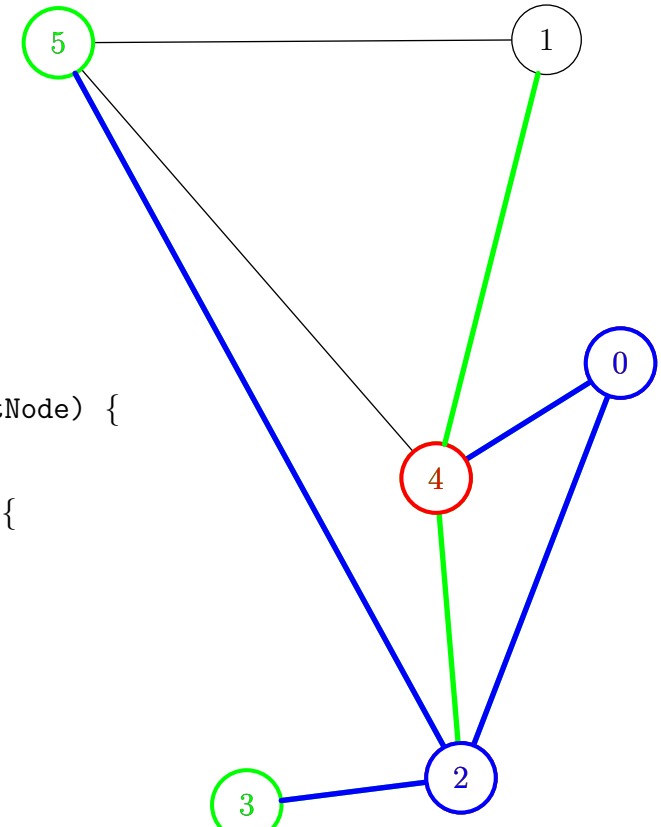


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=4 neighbour=1

q=

3	5				

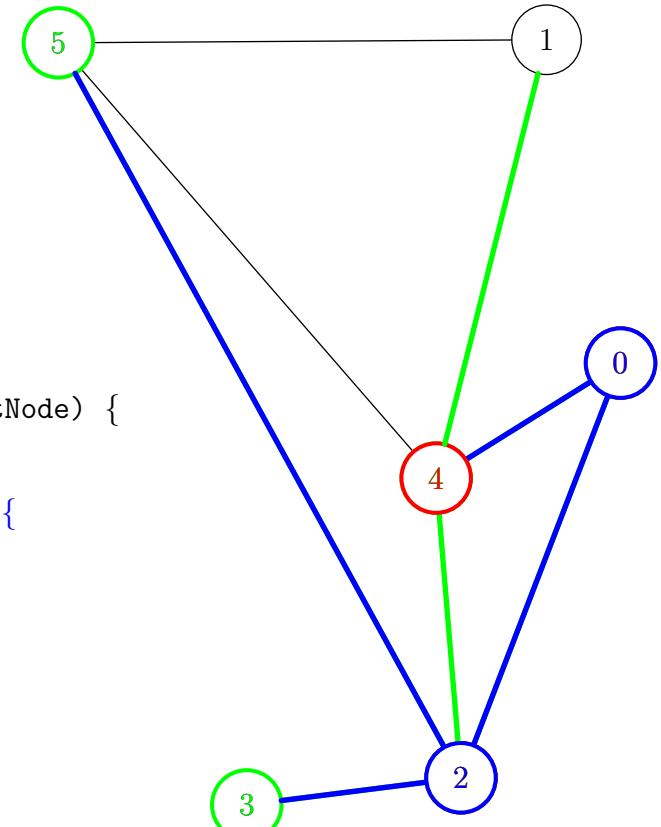


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=4 neighbour=1

q=

3	5				

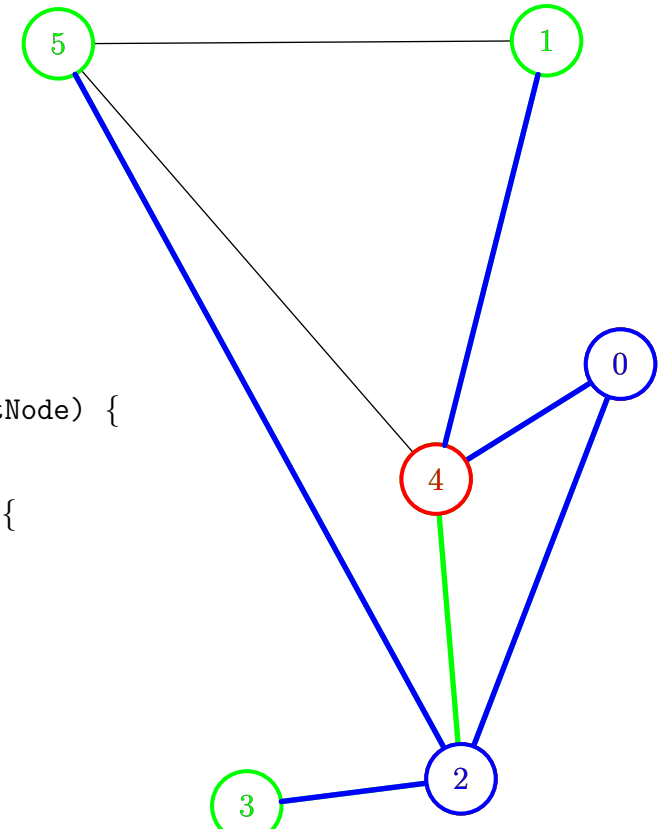


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=4 neighbour=1

q=

3	5	1			

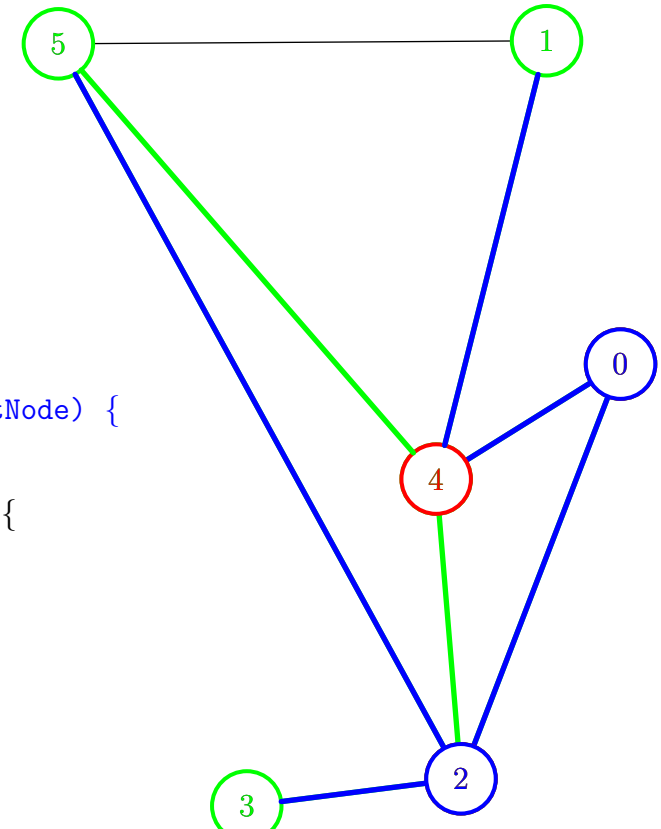


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=4 neighbour=5

q=

3	5	1			

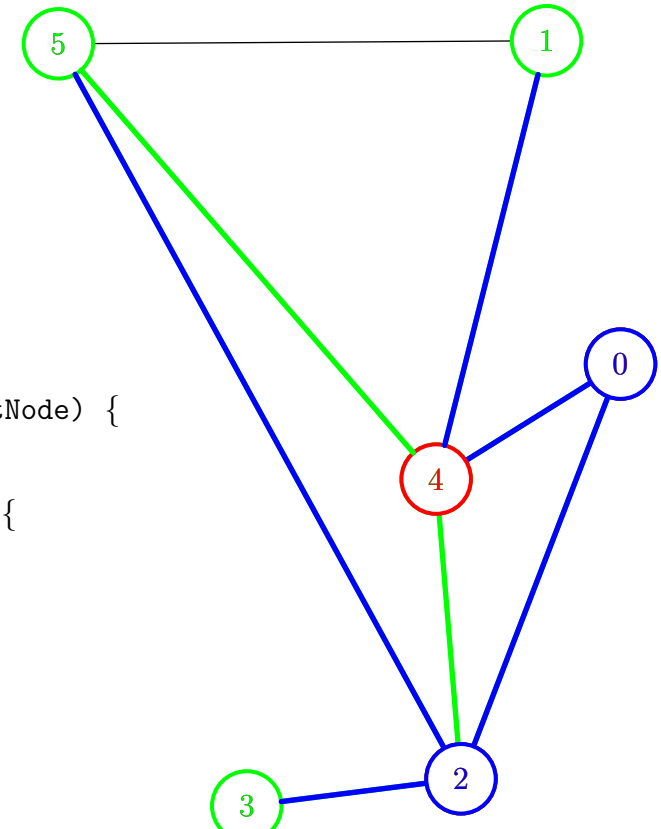


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=4 neighbour=5

q=

3	5	1			

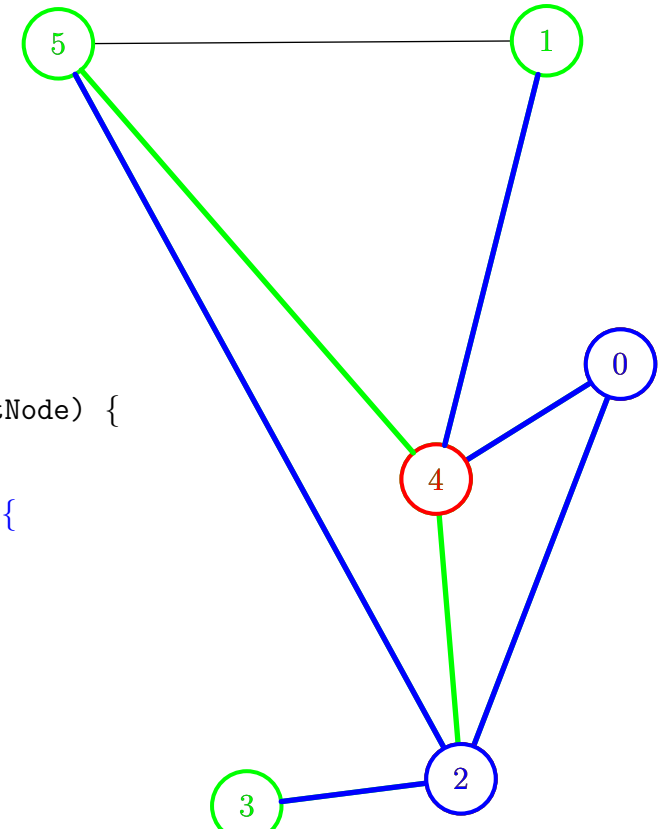


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=4 neighbour=5

q=

3	5	1			

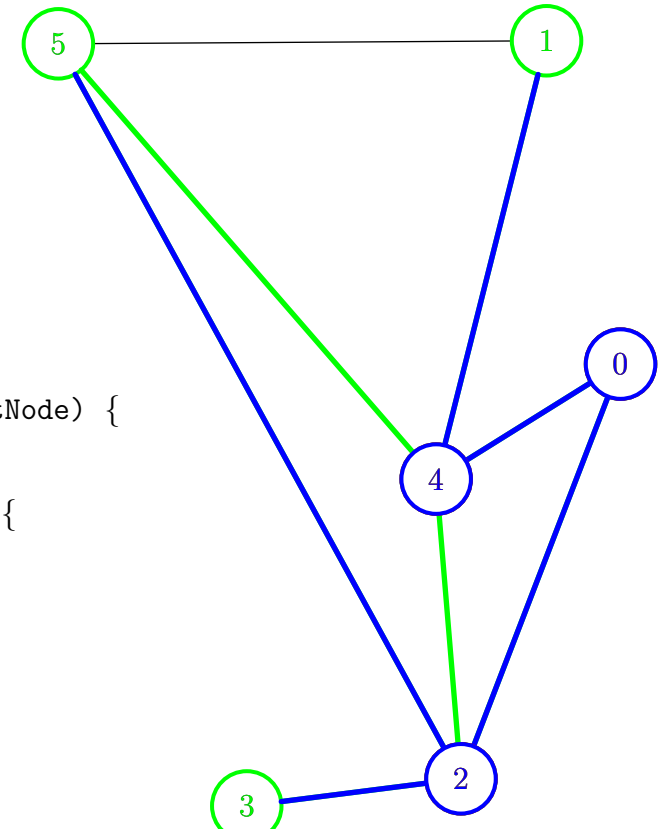


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

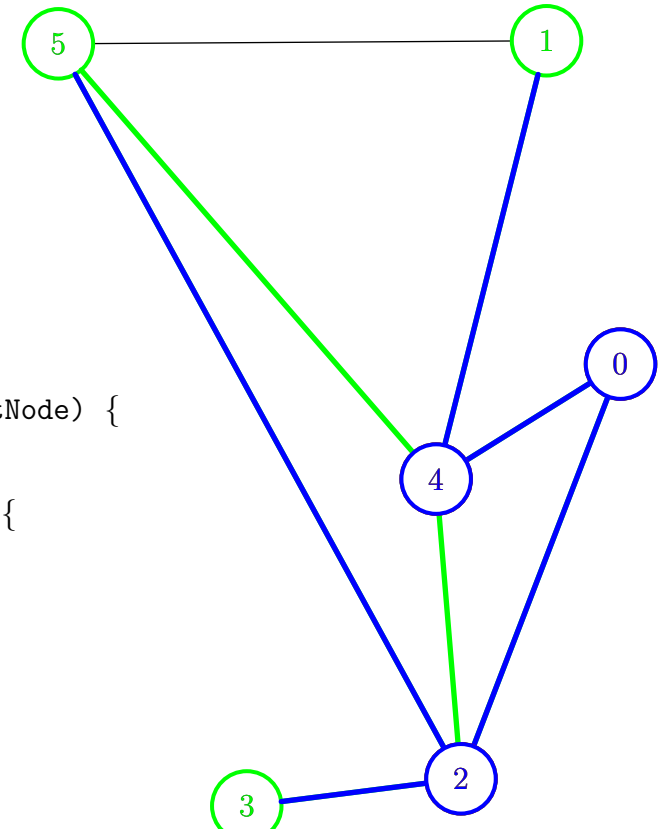
currentNode=4

q=

3	5	1			



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```



q=

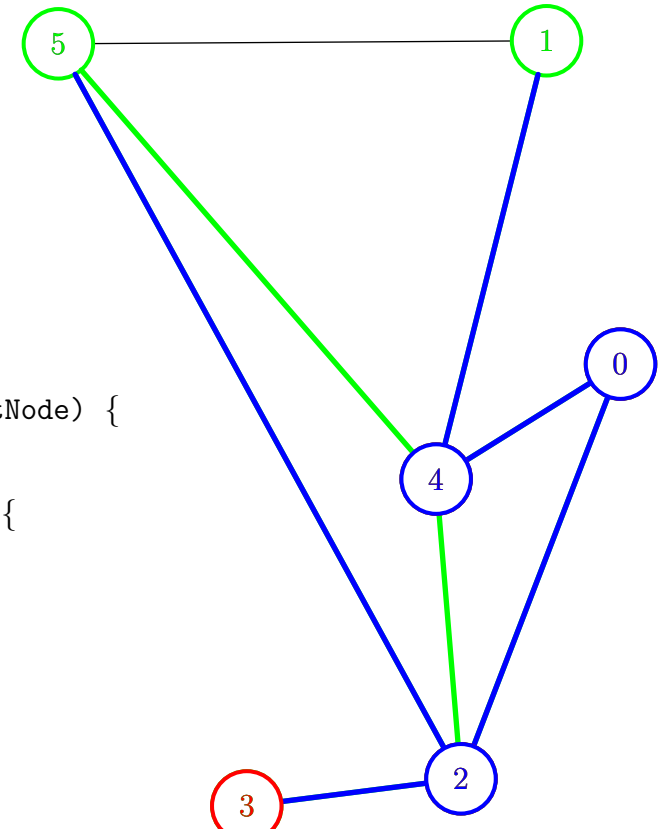
3	5	1			

```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=3

q=

5	1				

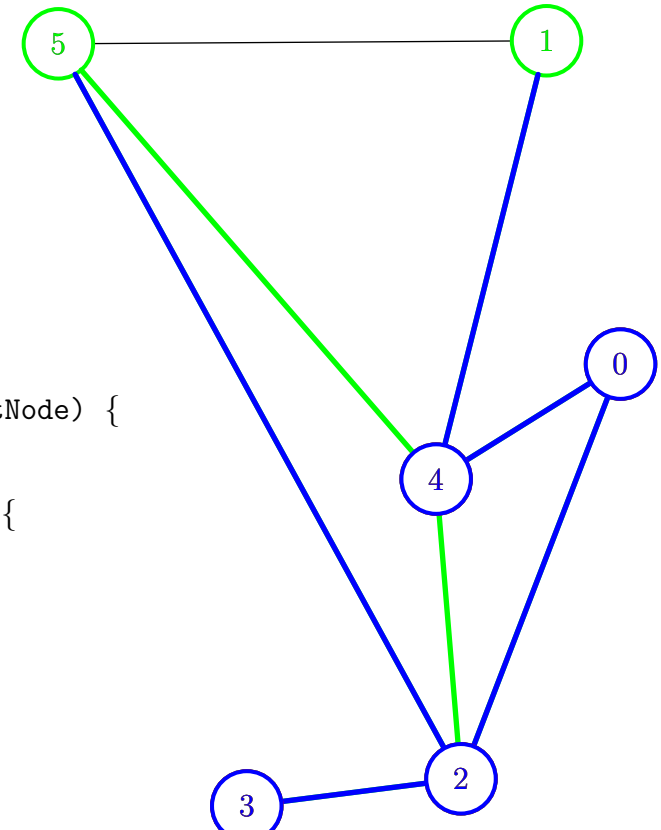


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

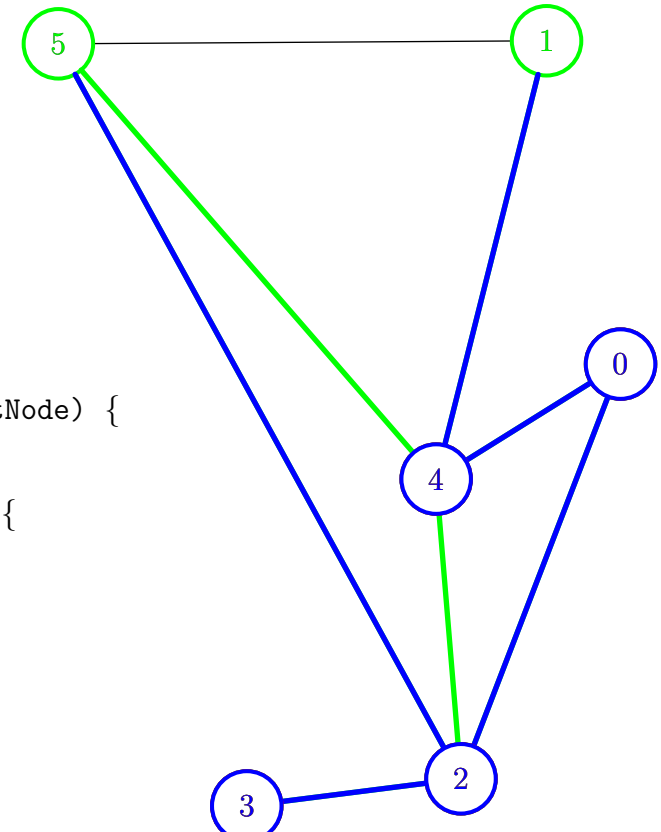
currentNode=3

q=

5	1				



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```



q=

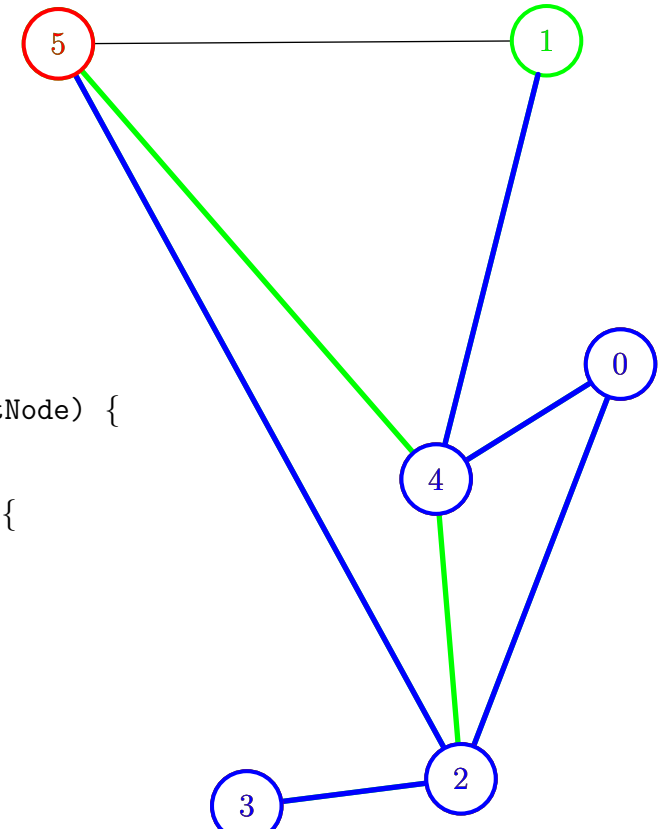
5	1				

```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=5

q=

1					

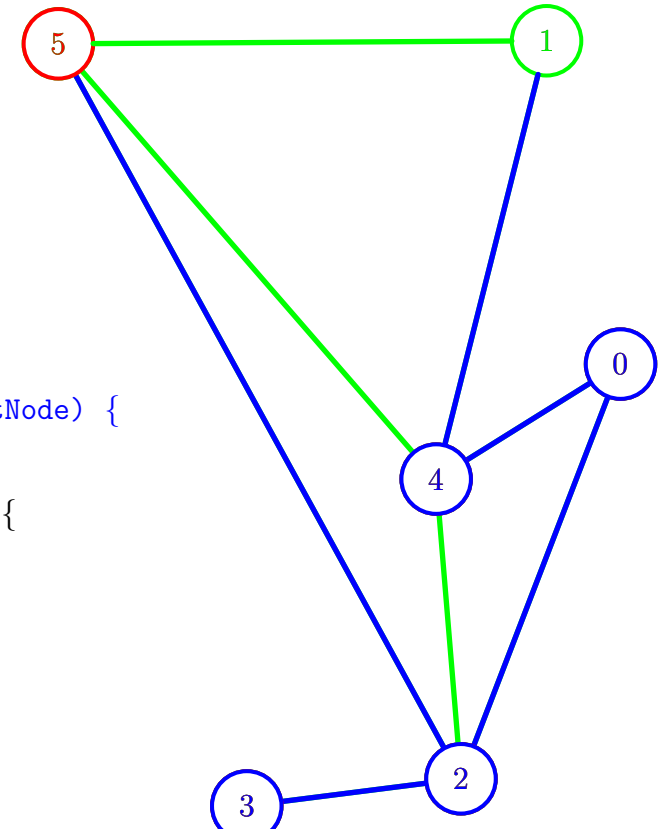


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=5 neighbour=1

q=

1					

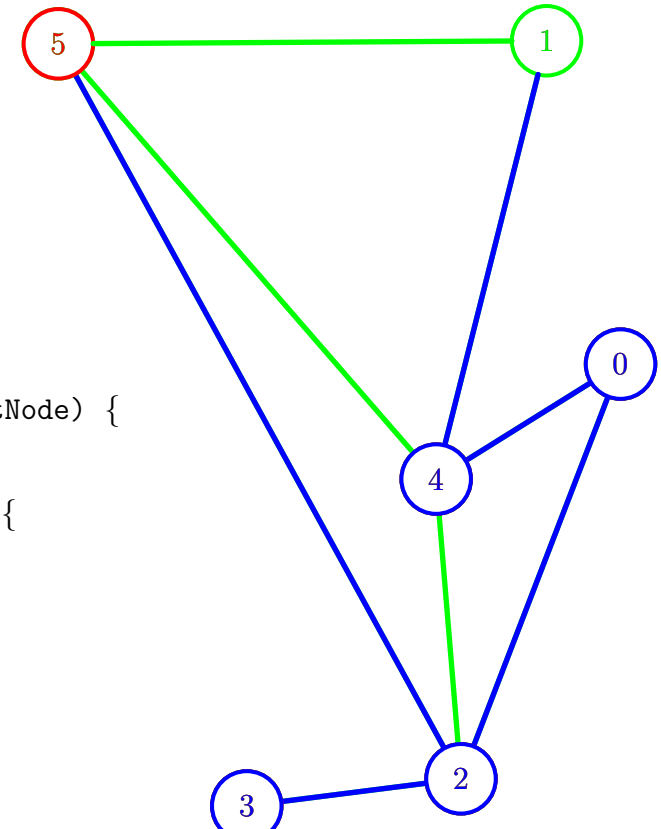


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=5 neighbour=1

q=

1					

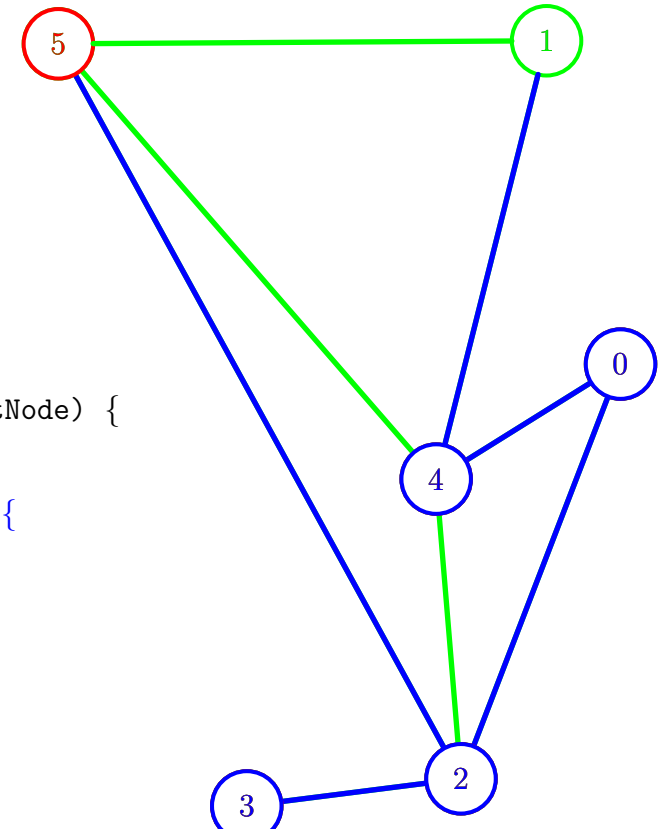


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=5 neighbour=1

q=

1					

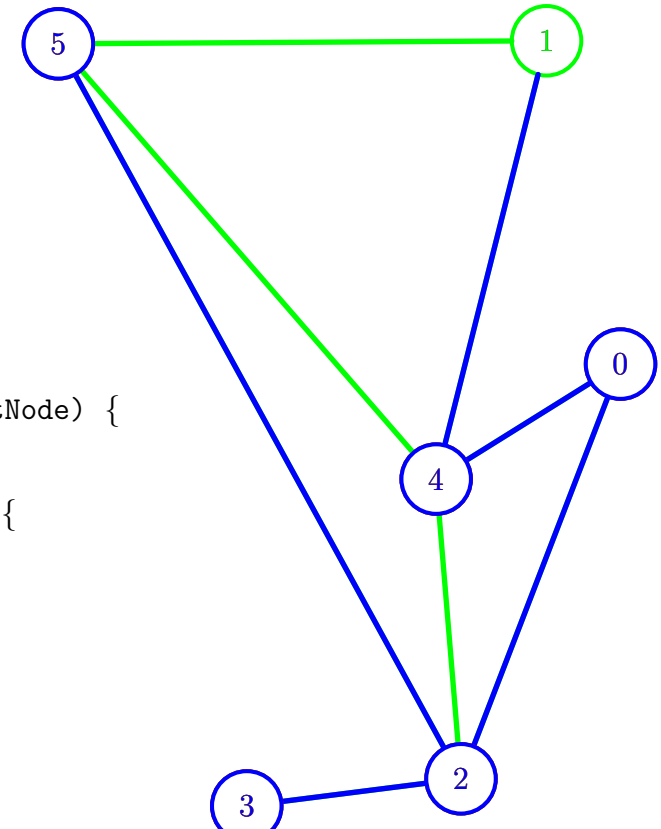


```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=5

q=

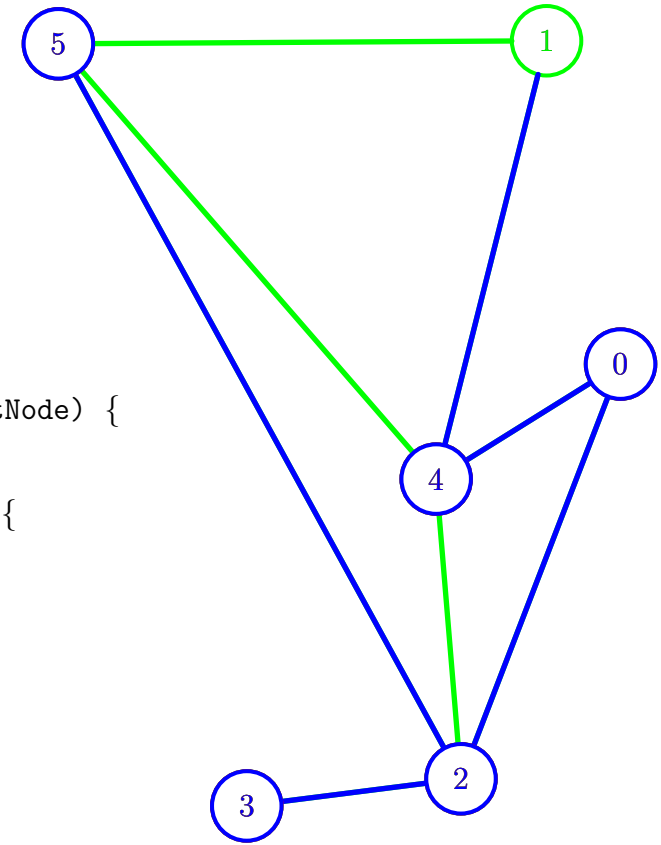
1					



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

q=

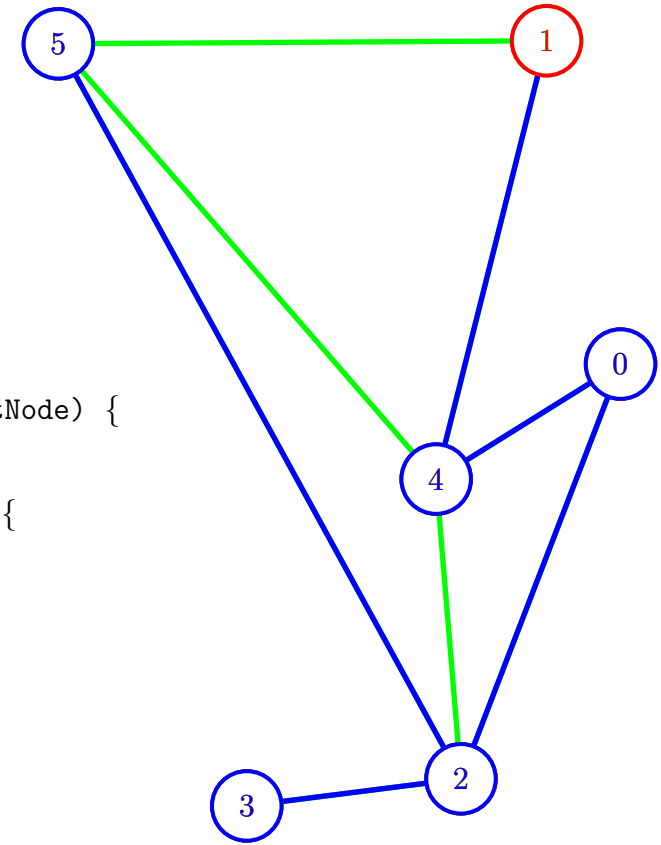
1					



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=1

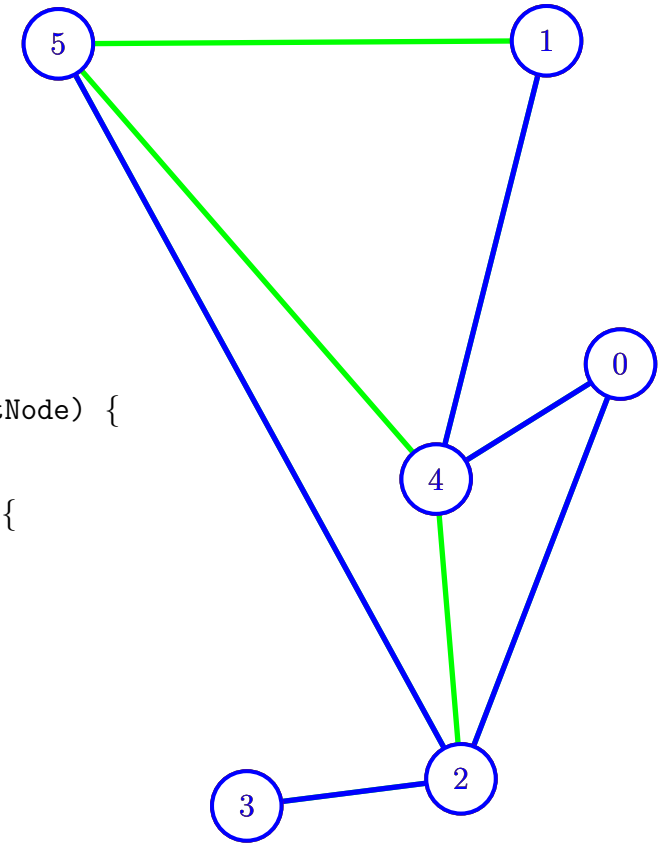
q=



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

currentNode=1

q=



```
bfs(graph, node) {
  List state(graph.noNodes, "undiscovered")
  List parent(graph.noNodes)
  state[node] ← "discovered"
  parent[node] ← nil
  Queue q
  q.enqueue(node)
  while (!q.isEmpty()) {
    currentNode ← q.dequeue()
    processVertexEarly(currentNode)
    state[currentNode] ← "processed"
    foreach neighbour ∈ Neighbourhood(currentNode) {
      if (state[neighbour] ≠ "processed")
        processEdge(currentNode, neighbour)
      if (state[neighbour] = "undiscovered") {
        state[neighbour] ← "discovered"
        parent[neighbour] ← currentNode
        q.enqueue(neighbour)
      }
    }
    processVertexLate(currentNode)
  }
}
```

q=