

# 1INF06 Data Structures and Methodical Programming

Loaiza Vasquez, Manuel Alejandro  
Oyarce Tocto, Elizabeth Patricia  
Saras Rivera, André Edgardo

November 11, 2020

Pontificia Universidad Católica del Perú  
Lima, Perú  
[manuel.loaiza@pucp.edu.pe](mailto:manuel.loaiza@pucp.edu.pe)  
[a20182778@pucp.edu.pe](mailto:a20182778@pucp.edu.pe)  
[andre.saras@pucp.edu.pe](mailto:andre.saras@pucp.edu.pe)

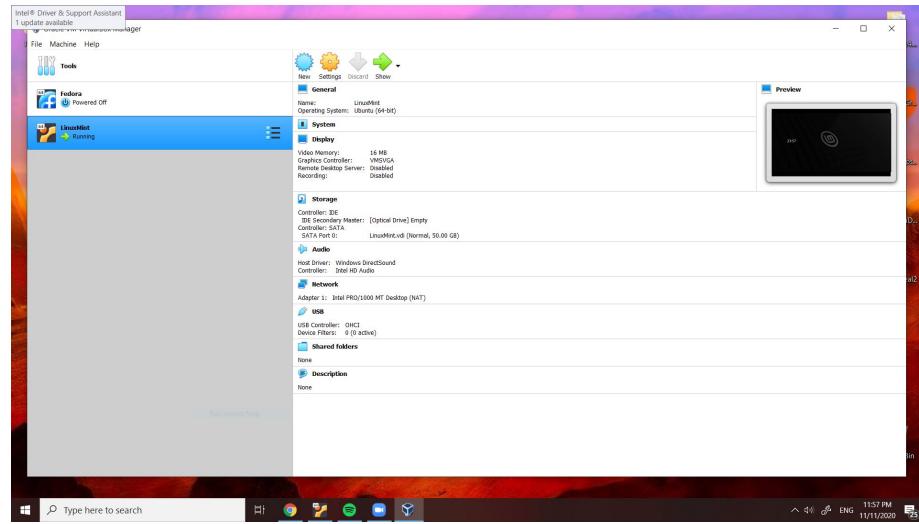
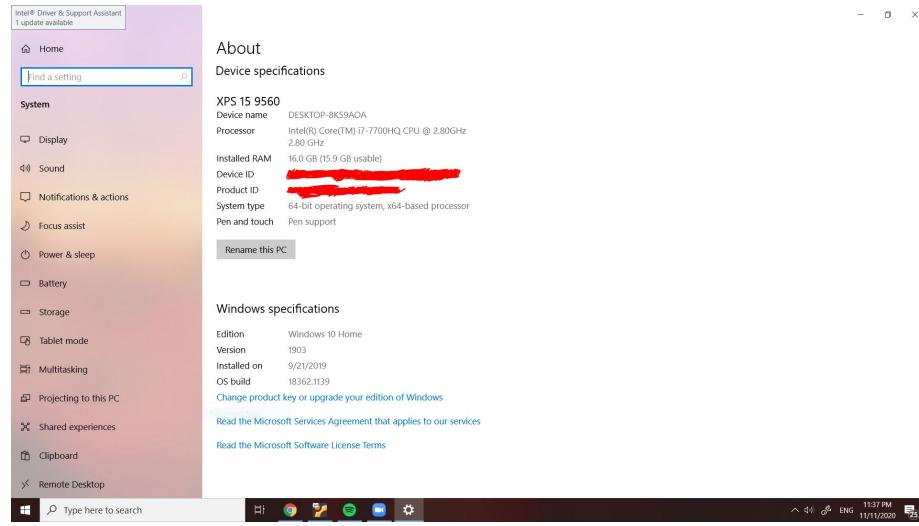
Fourth report of the course Data Structures and Methodical Programming taught at the Faculty of Science and Engineering at Pontificia Universidad Católica del Perú (PUCP) by Viktor Khlebnikov in the semester 2020-2.

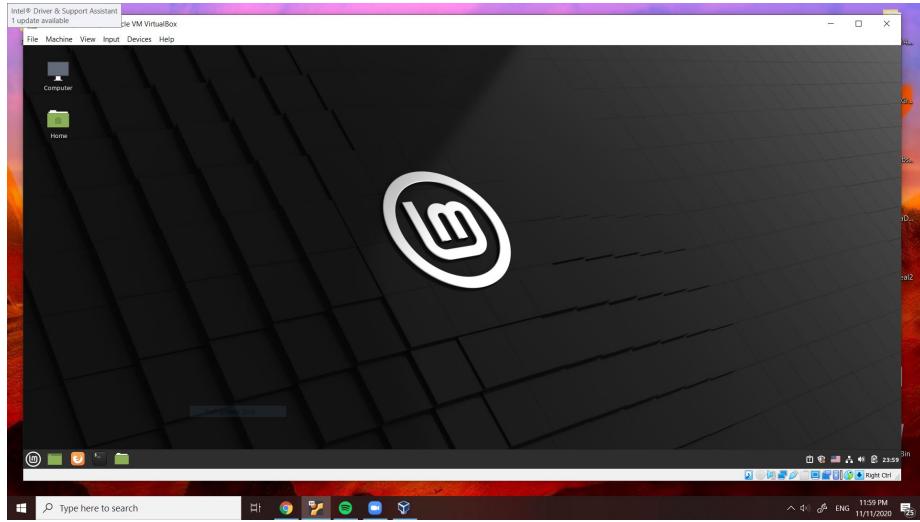
## Contents

<b>1 Loaiza Vasquez, Manuel Alejandro</b>	<b>2</b>
1.1 Resources and Oracle VM VirtualBox parameters . . . . .	2
1.2 Graph implementation using map . . . . .	3
1.3 Ruby example program . . . . .	6
<b>2 Oyarce Tocto, Elizabeth Patricia</b>	<b>11</b>
2.1 Resources and Oracle VM VirtualBox parameters . . . . .	11
2.2 Graph implementation using linked lists . . . . .	12
2.3 Ruby example program . . . . .	16
<b>3 Saras Rivera, André Edgardo</b>	<b>19</b>
3.1 Resources and Oracle VM VirtualBox parameters . . . . .	19
3.2 Graph implementation using adjacency matrix . . . . .	20
3.3 Ruby example program . . . . .	25

# 1 Loaiza Vasquez, Manuel Alejandro

## 1.1 Resources and Oracle VM VirtualBox parameters





## 1.2 Graph implementation using map

The class `Graph` represents a graph with the specified nodes and edges. I have created two more classes: `Node` and `Edges`.

`Node` definition has two attributes: `value` and `neighbors`. The first one is going to be an identifier for a node, for example, the student code of a PUCP student. The second one is a map (also known as dictionaries or `Hash` in Ruby) in which each element of the map stores a pointer to a `Node` instance and the key of that element is the identifier of the node pointed by the pointer.

`Edge` definition has two attributes: `from` and `to`, which represents the endpoints of a directed edge. If our graph is undirected, then we have to add the reversed edge too.

- Method: `Size`  
Usage: `size = graph.Size`  
Returns the number of nodes in the graph.
- Method: `IsEmpty`  
Usage: `if graph.IsEmpty . . .`  
Returns `true` if the graph is empty.
- Method: `AddNode`  
Usage: `graph.AddNode(node_id)`  
Add a node to the graph.
- Method: `RemoveNode`  
Usage: `graph.RemoveNode(node_id)`  
Removes a node from the graph by its identifier. It also remove all the edges that contains at least one endpoint with these node.

- Method: **GetNode**  
 Usage: `node_ptr = graph.GetNode(node_id)`  
 Looks up a node in the map attached to the graph and returns a pointer to that node.
- Method: **NodeExists**  
 Usage: `if graph.NodeExists(node_id) . . .`  
 Return `true` if a node with the give identifier exists
- Method: **AddEdge**  
 Usage: `graph.AddEdge(from, to)`  
 Adds an edge to the graph. If a node doesn't exist in the graph, then it is going to be added.
- Method: **RemoveEdge**  
 Usage: `graph.RemoveEdge(from, to)`  
 Removes an edge from the graph.
- Method: **IsConnected**  
 Usage: `if graph.IsConnected(u, v) . . .`  
 Return `true` if the graph contains an edge from `u` to `v`.
- Method: **GetNodeSet**  
 Usage: `nodes = graph.GetNodeSet`  
 Returns the set of all nodes in the graph.
- Method: **GetEdgeSet**  
 Usage: `edges = graph.GetEdgeSet`  
 Returns the set of all edges in the graph.
- Method: **GetNeighbors**  
 Usage: `neighbors = graph.GetNeighbors(node_id)`  
 Returns the set of nodes that are neighbors of the specified node.

The screenshot shows a Windows desktop environment. In the top-left corner, there's a system tray icon for 'Intel® Driver & Support Assistant' with a message '1 update available'. The taskbar has icons for File Explorer, Task View, Start, and other system applications. A terminal window titled 'File Machine View Input Devices Help' is open, displaying Ruby code for a graph data structure. Below it, a file explorer window titled 'manejodealgoritmos' shows a file named 'informe4d4' with a size of 1,1 KB.

```
File Edit View Search Terminal Help
manejodealgoritmos/manejodealgoritmos ~/linfile-estructura-de-datos-y-programacion-metodica-pcp/informe4d4
1 class Node
2   attr_accessor :value, :neighbors
3 
4   def initialize value
5     self.value = value
6     self.neighbors = Hash.new
7   end
8 end
9 
10 class Edge
11   attr_accessor :from, :to
12 
13   def initialize from, to
14     self.from = from
15     self.to = to
16   end
17 end
18 
19 class Graph
20   attr_accessor :nodes
21 
22   def initialize
23     self.nodes = Hash.new
24   end
25 
26   def Size
27     return self.nodes.size()
28   end
29 
30   def IsEmpty
31     return self.nodes.empty?()
32   end
33
```

Intel® Driver & Support Assistant  
1 update available  
File Machine View Input Devices Help

manuelloalcaz@manuelloalcaz: ~/linfo6-estructura-de-datos-y-programacion-metodica-puzz/informes/informe04

```
34     def Clear
35         self.nodes.clear()
36     end
37
38     def AddNode id
39         new_node = Node.new(id)
40         self.nodes[id] = new_node
41     end
42
43     def RemoveNode id
44         raise "Node doesn't exist" unless self.NodeExists(id)
45
46         self.nodes.delete(id);
47         self.nodes.each do |node_id, node_ptr|
48             if node_ptr.neighbors.key?(id)
49                 node_ptr.neighbors.delete(id)
50             end
51         end
52     end
53
54     def GetNode id
55         return self.nodes[id]
56     end
57
58     def NodeExists id
59         return self.nodes.key?(id);
60     end
61
62     def AddEdge from, to
63         if not self.NodeExists(from)
64             self.AddNode(from)
65         end
66     end
```

66,0-1 37%

```

Intel® Driver & Support Assistant
1 update available
File Machine View Input Devices Help
maneuvealize@maneuvealize: ~/linfo6-estructura-de-datos-y-programacion-metodica-pwp/informes/informe4
65   end
66
67   if not self.NodeExists(to)
68     self.AddNode(to)
69   end
70
71   self.nodes[from].neighbors[to] = nodes[to]
72 end
73
74 def RemoveEdge from, to
75   raise "Nodes don't exist" unless self.NodeExists(from) and self.NodeExists(to)
76   raise "Edge doesn't exist" unless self.nodes[from].neighbors.key?(to)
77
78   self.nodes[from].neighbors.delete(to)
79 end
80
81 def IsConnected from, to
82   if not self.NodeExists(from) or not self.NodeExists(to)
83     return false
84   end
85
86   return self.nodes[from].neighbors.key?(to)
87 end
88
89 def GetNodeSet
90   node_set = Set.new
91
92   self.nodes.each do |node_id, node_ptr|
93     node_set.add(node_id)
94   end
95
96   return node_set
97 end

```

65,1 71% 65,1 71%

```

Intel® Driver & Support Assistant
1 update available
File Machine View Input Devices Help
maneuvealize@maneuvealize: ~/linfo6-estructura-de-datos-y-programacion-metodica-pwp/informes/informe4
90   node_set = Set.new
91
92   self.nodes.each do |node_id, node_ptr|
93     node_set.add(node_id)
94   end
95
96   return node_set
97 end
98
99 def GetEdgeSet
100   edge_set = Set.new
101
102   self.nodes.each do |from_id, from_ptr|
103     from_ptr.neighbors.each do |to_id, to_ptr|
104       new_edge = Edge.new(from_id, to_id)
105       edge_set.add(new_edge)
106     end
107   end
108
109   return edge_set
110 end
111
112 def GetNeighbors id
113   raise "Node doesn't exist" unless self.nodes.key?(id)
114
115   neighbors_set = Set.new
116   self.nodes[id].neighbors.each do |to_id, to_ptr|
117     neighbors_set.add(to_id)
118   end
119
120   return neighbors_set
121 end
122 end

```

122,1 80% 122,1 80%

### 1.3 Ruby example program

The following program has the goal to test if the methods implemented in the class `Graph` are correct. The first header is for using the `Set` data structure in which I am going to store the partial results of my analysis to print them on screen to see how my graph has changed after deleting a vertex o an edge. The second header tells Ruby I want to bring in my `Graph` class.

```
Intel® Driver & Support Assistant 1 update available [x] VM VirtualBox
File Machine View Input Devices Help manuelalao@manuelalao: ~/linfo6-estructura-de-datos-y-programacion-metodica-pwp/informes/informe4
manuelalao@manuelalao:~/linfo6-estructura-de-datos-y-programacion-metodica-pwp/informes/informe4
1 require "set"
2 require_relative "./graph.rb"
3
4 def PrintNodes
5   print "Nodes"
6   nodes.each do |node|
7     print " #{node}"
8   end
9   puts ""
10 end
11
12 def PrintEdges edges
13   print "Edges"
14   edges.each do |edge|
15     print " (#edge.from), (#edge.to))"
16   end
17   puts ""
18 end
19
20 graph = Graph.new
21 raise "Error in Size" unless graph.Size() == 0
22 raise "Error in IsEmpty" unless graph.IsEmpty()
23
24 graph.AddNode()
25 graph.AddNode()
26 graph.AddNode(2)
27 graph.AddNode(3)
28 graph.AddNode(4)
29 graph.AddNode(5)
30
31 raise "Error in Size" unless graph.Size() == 6
32
33 # 0 --> 1      2
34 # |           |
35 # |           v
36 # v           v
37 # 3 <-- 4      5 <-
38 #
39 #      ...
40 graph.AddEdge(0, 1)
41 graph.AddEdge(0, 3)
42 graph.AddEdge(2, 4)
43 graph.AddEdge(2, 5)
44 graph.AddEdge(3, 1)
45 graph.AddEdge(3, 4)
46 graph.AddEdge(4, 5)
47
48 raise "Error in AddEdge" unless graph.IsConnected(0, 1)
49 raise "Error in AddEdge" unless graph.IsConnected(0, 2)
50 raise "Error in AddEdge" unless graph.IsConnected(2, 4)
51 raise "Error in AddEdge" unless graph.IsConnected(2, 5)
52 raise "Error in AddEdge" unless graph.IsConnected(3, 1)
53 raise "Error in AddEdge" unless graph.IsConnected(4, 3)
54 raise "Error in AddEdge" unless graph.IsConnected(4, 5)
55
56 raise "Error in AddEdge" unless not graph.IsConnected(0, 2)
57 raise "Error in AddEdge" unless not graph.IsConnected(1, 2)
58 raise "Error in AddEdge" unless not graph.IsConnected(1, 3)
59 raise "Error in AddEdge" unless not graph.IsConnected(2, 3)
60 raise "Error in AddEdge" unless not graph.IsConnected(2, 5)
61
62 PrintNodes(graph.GetNodeSet)
63 PrintEdges(graph.GetEdgeSet)
64
65
```

```
Intel® Driver & Support Assistant 1 update available [x] VM VirtualBox
File Machine View Input Devices Help manuelalao@manuelalao: ~/linfo6-estructura-de-datos-y-programacion-metodica-pwp/informes/informe4
manuelalao@manuelalao:~/linfo6-estructura-de-datos-y-programacion-metodica-pwp/informes/informe4
33 # 0 --> 1      2
34 # |           |
35 # |           v
36 # v           v
37 # 3 <-- 4      5 <-
38 #
39 #      ...
40 graph.AddEdge(0, 1)
41 graph.AddEdge(0, 3)
42 graph.AddEdge(2, 4)
43 graph.AddEdge(2, 5)
44 graph.AddEdge(3, 1)
45 graph.AddEdge(3, 4)
46 graph.AddEdge(4, 5)
47
48 raise "Error in AddEdge" unless graph.IsConnected(0, 1)
49 raise "Error in AddEdge" unless graph.IsConnected(0, 2)
50 raise "Error in AddEdge" unless graph.IsConnected(2, 4)
51 raise "Error in AddEdge" unless graph.IsConnected(2, 5)
52 raise "Error in AddEdge" unless graph.IsConnected(3, 1)
53 raise "Error in AddEdge" unless graph.IsConnected(4, 3)
54 raise "Error in AddEdge" unless graph.IsConnected(4, 5)
55
56 raise "Error in AddEdge" unless not graph.IsConnected(0, 2)
57 raise "Error in AddEdge" unless not graph.IsConnected(1, 2)
58 raise "Error in AddEdge" unless not graph.IsConnected(1, 3)
59 raise "Error in AddEdge" unless not graph.IsConnected(2, 3)
60 raise "Error in AddEdge" unless not graph.IsConnected(2, 5)
61
62 PrintNodes(graph.GetNodeSet)
63 PrintEdges(graph.GetEdgeSet)
64
65
```

```

Intel® Driver & Support Assistant
1 update available
File Machine View Input Devices Help
manuelloaiza@manuelloaiza: ~/linfo6-estructura-de-datos-y-programacion-metodica-pucp/informes/informe04
File Edit View Search Terminal Help
manuelloaiza@manuelloaiza:~/linfo6-estructura-de-datos-y-programacion-metodica-pucp/informes/informe04
57 raise "Error in AddEdge" unless not graph.isConnected(1, 2)
58 raise "Error in AddEdge" unless not graph.isConnected(2, 3)
59 raise "Error in AddEdge" unless not graph.isConnected(3, 4)
60 raise "Error in AddEdge" unless not graph.isConnected(4, 5)
61
62 PrintNodes(graph.GetNodeSet)
63 PrintEdges(graph.GetEdgeSet)
64
65
66 graph.RemoveNode(4)
67 # 0 --> 1      2
68 # |           |
69 # |           |
70 # |           v
71 # 3           5 <-
72 # |           |
73 # |           ...
74 PrintNodes(graph.GetNodeSet)
75 PrintEdges(graph.GetEdgeSet)
76
77 graph.RemoveEdge(5, 1)
78 # 0 --> 1      2
79 # |           |
80 # |           |
81 # |           v
82 # 3           5
83 PrintNodes(graph.GetNodeSet)
84 PrintEdges(graph.GetEdgeSet)
85
86 graph.Clear()
87 raise "Error in Clear" unless graph.IsEmpty
88
89 puts "All tests are OK"
89,1 Bot

```

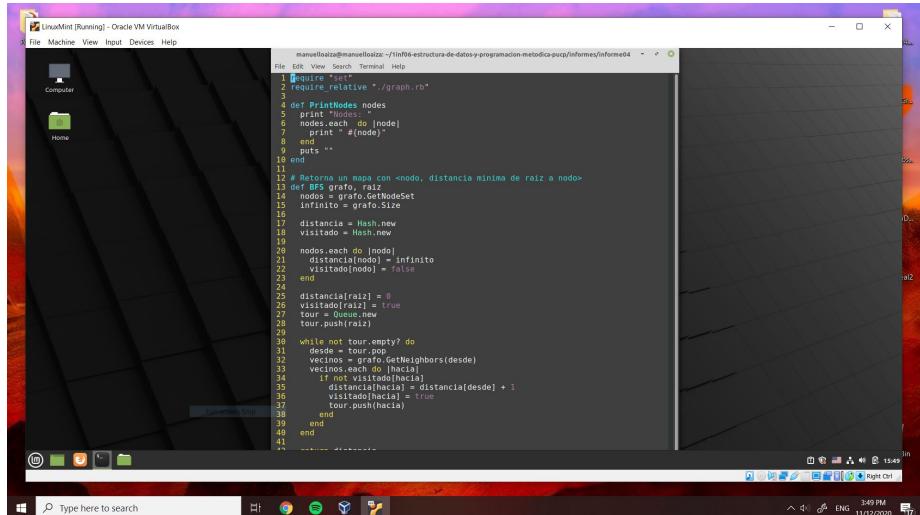
  

```

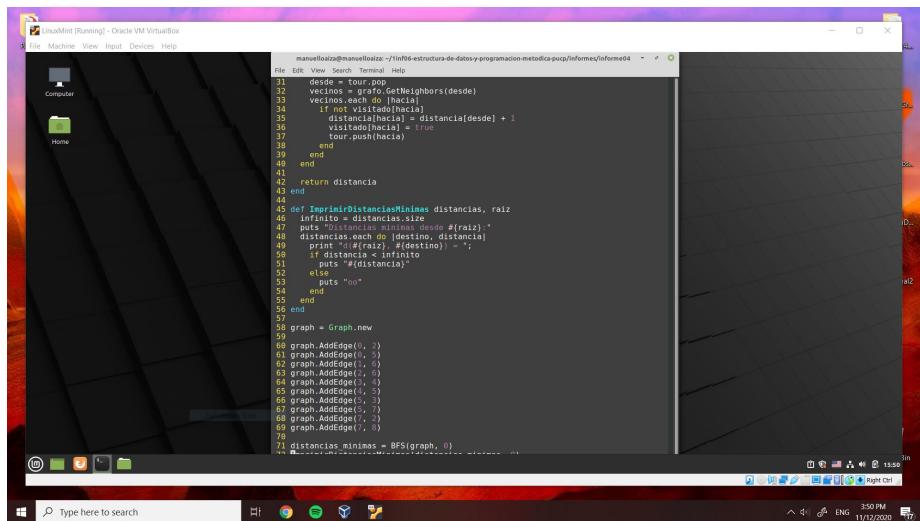
Intel® Driver & Support Assistant
1 update available
File Machine View Input Devices Help
manuelloaiza@manuelloaiza: ~/linfo6-estructura-de-datos-y-programacion-metodica-pucp/informes/informe04
File Edit View Search Terminal Help
manuelloaiza@manuelloaiza:~/linfo6-estructura-de-datos-y-programacion-metodica-pucp/info
rmes/informe04$ ls -l
total 12
-rw-rw-r-- 1 manuelloaiza manuelloaiza 2163 Nov 11 09:37 graph.rb
-rw-rw-r-- 1 manuelloaiza manuelloaiza 2002 Nov 11 10:39 main.rb
manuelloaiza@manuelloaiza:~/linfo6-estructura-de-datos-y-programacion-metodica-pucp/info
rmes/informe04$ vim graph.rb
manuelloaiza@manuelloaiza:~/linfo6-estructura-de-datos-y-programacion-metodica-pucp/info
rmes/informe04$ vim main.rb
manuelloaiza@manuelloaiza:~/linfo6-estructura-de-datos-y-programacion-metodica-pucp/info
rmes/informe04$ ruby main.rb
Nodes: 0 1 2 3 5
Edges: (0, 1) (0, 3) (2, 4) (2, 5) (3, 1) (4, 3) (5, 5)
Nodes: 0 1 2 3 5
Edges: (0, 1) (0, 3) (2, 5) (3, 1) (5, 5)
Nodes: 0 1 2 3 5
Edges: (0, 1) (0, 3) (2, 5) (3, 1)
All tests are OK
manuelloaiza@manuelloaiza:~/linfo6-estructura-de-datos-y-programacion-metodica-pucp/info
rmes/informe04$ 

```

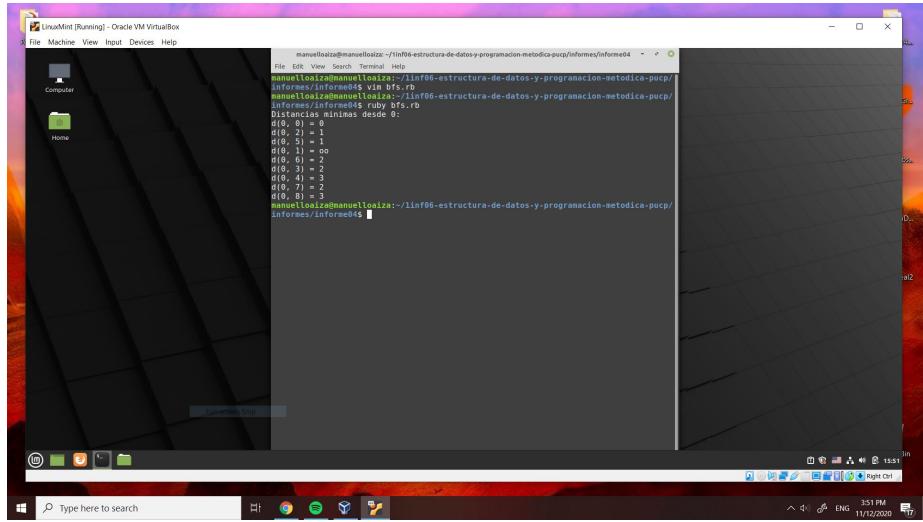
Finally, I am going to apply an algorithm called Breadth First Search to find the minimum distance in an unweighted graph from a specific vertex to the other ones in  $O(V + E)$ . If a node is not reachable from that vertex, then the minimum distance is going to be  $\infty$ . If we want to calculate the minimum distance from a specific vertex to all the other vertices in a weighted graph, we have to add weights to the edges and this algorithm is not going to work.



```
manuelalizademanuelaliza -/fin96-estructura-de-datos-y-programacion-metodica-pucp/informes/informe04
File Edit View Search Terminal Help
File Machine View Input Devices Help
Computer
Home
manuelalizademanuelaliza -/fin96-estructura-de-datos-y-programacion-metodica-pucp/informes/informe04
File Edit View Search Terminal Help
1 require "set"
2 require_relative "./graph.rb"
3
4 def PrintNodes nodes
5   print "Nodes: "
6   nodes.each do |node|
7     print "#{$node}"
8   end
9   puts ""
10 end
11
12 # Retorna un mapa con <nodo>, distancia minima de raiz a nodo
13 def BFS grafo, raiz
14   nodos = grafo.GetNodeSet
15   infinito = grafo.Size
16   distancia = Hash.new(infinito)
17   visitado = Hash.new(false)
18   tour = Queue.new
19   nodos.each do |nodo|
20     distancia[nodo] = infinito
21     visitado[nodo] = false
22   end
23
24   distancia[raiz] = 0
25   visitado[raiz] = true
26   tour.push(raiz)
27   tour.push(true)
28   tour.push(true)
29
30   while not tour.empty? do
31     desde = tour.pop
32     vecinos = grafo.GetNeighbors(desde)
33     vecinos.each do |hacia|
34       if not visitado[hacia]
35         distancia[hacia] = distancia[desde] + 1
36         visitado[hacia] = true
37         tour.push(hacia)
38       end
39     end
40   end
41
42 return distancia
43 end
44
45 def ImprimirDistanciasMinimas distancias, raiz
46   infinito = distancias.size
47   distancias.each do |destino, distancia|
48     print "#{raiz}#{destino}#{distancia}\n"
49   end
50   puts "-----"
51   exit(0)
52 end
53
54 puts "oo"
55 end
56 end
57
58 graph = Graph.new
59
60 graph.AddEdge(0, 2)
61 graph.AddEdge(0, 3)
62 graph.AddEdge(1, 0)
63 graph.AddEdge(1, 6)
64 graph.AddEdge(2, 1)
65 graph.AddEdge(2, 4)
66 graph.AddEdge(3, 0)
67 graph.AddEdge(3, 5)
68 graph.AddEdge(4, 2)
69 graph.AddEdge(4, 5)
70 graph.AddEdge(5, 3)
71 graph.AddEdge(5, 6)
72 graph.AddEdge(6, 1)
73 graph.AddEdge(6, 4)
74
75 distancias minimas = BFS(graph, 0)
```



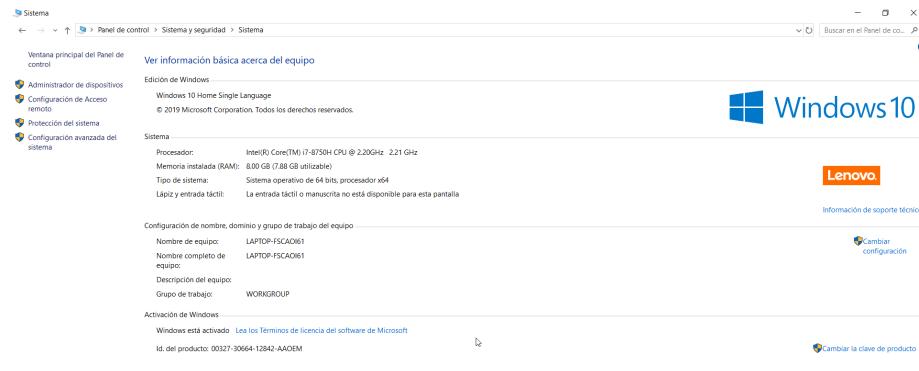
```
manuelalizademanuelaliza -/fin96-estructura-de-datos-y-programacion-metodica-pucp/informes/informe04
File Edit View Search Terminal Help
File Machine View Input Devices Help
Computer
Home
manuelalizademanuelaliza -/fin96-estructura-de-datos-y-programacion-metodica-pucp/informes/informe04
File Edit View Search Terminal Help
1 require "set"
2 require_relative "./graph.rb"
3
4 def PrintNodes nodes
5   print "Nodes: "
6   nodes.each do |node|
7     print "#{$node}"
8   end
9   puts ""
10 end
11
12 # Retorna un mapa con <nodo>, distancia minima de raiz a nodo
13 def BFS grafo, raiz
14   nodos = grafo.GetNodeSet
15   infinito = grafo.Size
16   distancia = Hash.new(infinito)
17   visitado = Hash.new(false)
18   tour = Queue.new
19   nodos.each do |nodo|
20     distancia[nodo] = infinito
21     visitado[nodo] = false
22   end
23
24   distancia[raiz] = 0
25   visitado[raiz] = true
26   tour.push(raiz)
27   tour.push(true)
28   tour.push(true)
29
30   while not tour.empty? do
31     desde = tour.pop
32     vecinos = grafo.GetNeighbors(desde)
33     vecinos.each do |hacia|
34       if not visitado[hacia]
35         distancia[hacia] = distancia[desde] + 1
36         visitado[hacia] = true
37         tour.push(hacia)
38       end
39     end
40   end
41
42 return distancia
43 end
44
45 def ImprimirDistanciasMinimas distancias, raiz
46   infinito = distancias.size
47   distancias.each do |destino, distancia|
48     print "#{raiz}#{destino}#{distancia}\n"
49   end
50   puts "-----"
51   exit(0)
52 end
53
54 puts "oo"
55 end
56 end
57
58 graph = Graph.new
59
60 graph.AddEdge(0, 2)
61 graph.AddEdge(0, 3)
62 graph.AddEdge(1, 0)
63 graph.AddEdge(1, 6)
64 graph.AddEdge(2, 1)
65 graph.AddEdge(2, 4)
66 graph.AddEdge(3, 0)
67 graph.AddEdge(3, 5)
68 graph.AddEdge(4, 2)
69 graph.AddEdge(4, 5)
70 graph.AddEdge(5, 3)
71 graph.AddEdge(5, 6)
72 graph.AddEdge(6, 1)
73 graph.AddEdge(6, 4)
74
75 distancias minimas = BFS(graph, 0)
```



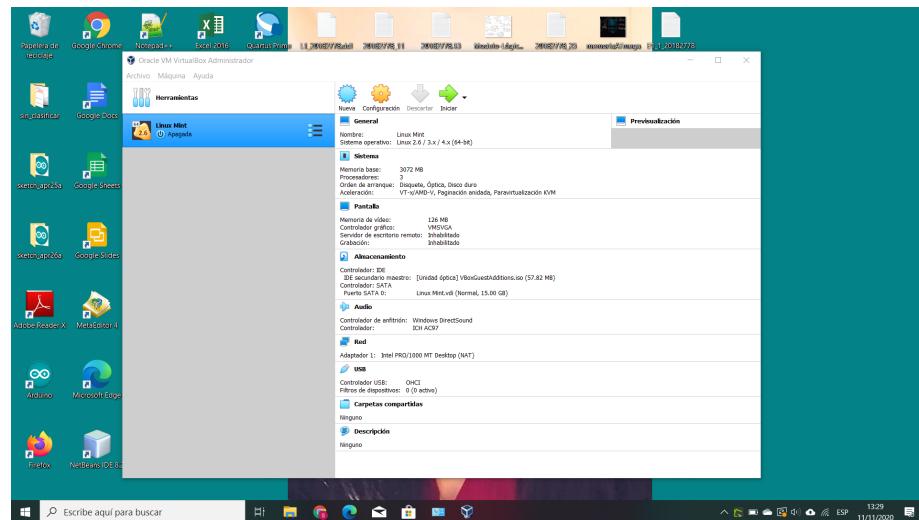
## 2 Oyarce Tocco, Elizabeth Patricia

### 2.1 Resources and Oracle VM VirtualBox parameters

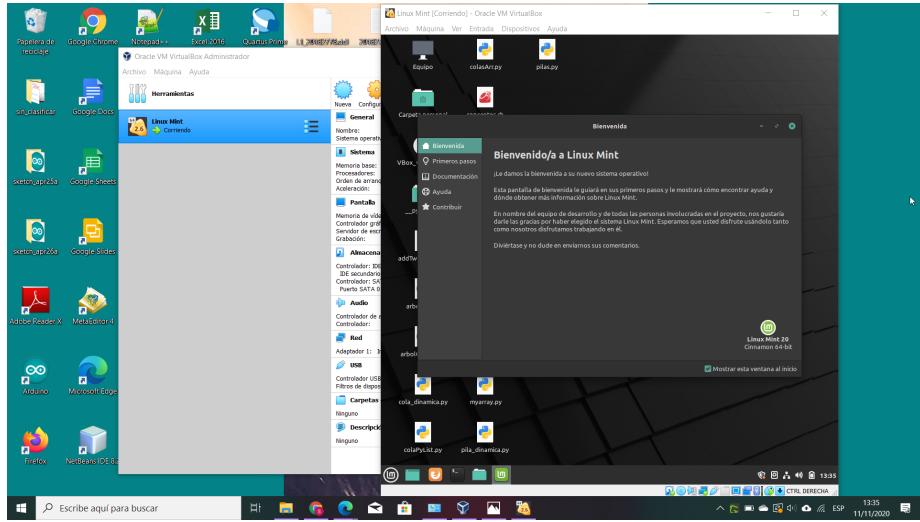
#### Computer resources



VirtualBox parameters



Linux Mint desktop



## 2.2 Graph implementation using linked lists

### LinkedList program

The following images are of the linked list code which is seen to implement the graph. This TAD was explained in detail in the previous report.

```

class LinkedList
    attr_accessor :head, :tail, :length

    def initialize
        @head = nil
        @tail = nil
        @length = 0
    end

    # Pushes element before the head
    def addBefore(value)
        node = Node.new value
        if self.length == 0
            self.head = node
            self.tail = node
        else
            node.next = self.head
            self.head = node
        end
        self.length += 1
    end

    # Removes the tail node
    def removeLast
        raise "Empty list" unless self.length > 0
        if self.length == 1
            self.head = self.tail = nil
        else
            current = self.head
            while current.next != self.tail
                current = current.next
            end
            self.tail = current
            current.next = nil
        end
        self.length -= 1
    end

    # Removes the head node
    def removeFirst
        raise "Empty list" unless self.length > 0
        if self.length == 1
            self.head = self.tail = nil
        else
            self.head = self.head.next
        end
        self.length -= 1
    end

    # Removes a node with value target in the list. If our target is in our head,
    # then we are going to first being aware of not having a empty list.
    # Then we are going to link the previous node to the next node of our
    # target and then, we are going to link this one and the one that is immediately
    # after our target.
    def remove(target)
        raise "Empty list" unless self.length > 0
        if self.length == 1
            self.head = self.tail = nil
        else
            current = self.head
            while current.next != self.tail
                if current.value == target
                    if current == self.head
                        self.head = current.next
                    else
                        current.next = current.next.next
                    end
                    self.length -= 1
                    return
                end
                current = current.next
            end
        end
    end
end

```

```

1 # target and then, we are going to link this one and the one that is immediately
2 # after it.
3 def RemoveTarget
4   if self.length > 0
5     if target == current.value
6       if self.head == self.tail == nil
7         self.head = self.tail = nil
8       else
9         self.head = self.tail = nil
10        self.head = self.head.next
11      end
12    end
13  end
14
15  # Prints the list from head to tail. The variable cnt counts the number of
16  # characters we have analyzed to print the characters that represent the links correctly.
17  def Print
18    puts "Empty list"
19    return
20  end
21
22  current = self.head
23  current = nil
24  current = current.next
25  current = current.next
26  current = current.next
27  current = current.next
28  current = current.next
29  current = current.next
30  current = current.next
31  current = current.next
32  current = current.next
33  current = current.next
34  current = current.next
35  current = current.next
36  current = current.next
37  current = current.next
38  current = current.next
39  current = current.next
40
41  puts " "
42  puts " "
43  puts " "
44  puts " "
45  puts " "
46  puts " "
47  puts " "
48  puts " "
49  puts " "
50  puts " "
51  puts " "
52  puts " "
53  puts " "
54  puts " "
55  puts " "
56  puts " "
57  puts " "
58  puts " "
59  puts " "
60  puts " "
61  puts " "
62  puts " "
63  puts " "
64  puts " "
65  puts " "
66  puts " "
67  puts " "
68  puts " "
69  puts " "
70  puts " "
71  puts " "
72  puts " "
73  puts " "
74  puts " "
75  puts " "
76  puts " "
77  puts " "
78  puts " "
79  puts " "
80  puts " "
81  puts " "
82  puts " "
83  puts " "
84  puts " "
85  puts " "
86  puts " "
87  puts " "
88  puts " "
89  puts " "
90  puts " "
91  puts " "
92  puts " "
93  puts " "
94  puts " "
95  puts " "
96  puts " "
97  puts " "
98  puts " "
99  puts " "
100 puts " "
101 puts " "
102 puts " "
103 puts " "
104 puts " "
105 puts " "
106 puts " "
107 puts " "
108 puts " "
109 puts " "
110 puts " "
111 puts " "
112 puts " "
113 puts " "
114 puts " "
115 puts " "
116 puts " "
117 puts " "
118 puts " "
119 puts " "
120 puts " "
121 puts " "
122 puts " "
123 puts " "
124 puts " "
125 puts " "
126 puts " "
127 puts " "
128 puts " "
129 puts " "
130 puts " "
131 puts " "
132 puts " "
133 puts " "
134 puts " "
135 puts " "
136 puts " "
137 puts " "
138 puts " "
139 puts " "

```

## Graph implementation

To define the **Graph** class, we will use two helper classes: the **NodeG** class and the **Edge** class. In addition, the linked list implemented previously will be imported

The **NodeG** definition has three attributes: **value**, **next** and **neighbors**. The first contains the value or identifier of one of the nodes of the graph. The second is a pointer that will be used to link it to the next node in the graph. The third is a linked list that contains all the neighbors that the node is related to.

The definition of **Edge** has two attributes: **from** and **to**, which represents the end points of a directed edge. If our graph is not directed then we have to add the inverted edge as well.

```

1 require_relative "./linkedList.rb"
2
3 class NodeG
4   attr_accessor :value, :neighbors, :next
5
6   def initialize(value)
7     self.value = value
8     self.next = nil
9     self.neighbors = LinkedList.new
10  end
11
12  end
13
14  class Edge
15    attr_accessor :from, :to
16    def initialize(from, to)
17      self.from = from
18      self.to = to
19    end
20  end
21
22  class Graph
23    attr_accessor :head, :tail, :length
24
25    def initialize
26      self.head = nil
27      self.tail = nil
28      self.length = 0
29    end
30
31    def Size
32      return self.length
33    end
34
35    def IsEmpty
36      return self.length == 0
37    end
38
39    def Clear
40      self.head = nil
41      self.tail = nil
42      self.length = 0
43    end
44  end

```

- Method: **Size**  
Returns the number of nodes in the graph.
  - Method: **IsEmpty**  
Returns `true` if the graph is empty, otherwise returns `false`.

Linux Mint [Comiendo] - Oracle VM VirtualBox

Acerca Mostrar Ver Entradas Desplazarse Ayuda

File Edit Selection Find View Goto Tools Project Preferences Help

graph.c Sublime Text (UNREGISTERED)

```
43
44     def AddNode value
45         node = Node.new value
46         if self.head == nil
47             self.head = node
48             self.tail = node
49         else
50             self.tail.next = node
51             self.tail = node
52         end
53         self.length += 1
54     end
55
56     def RemoveNodes target
57         raise "Empty graph" unless self.length > 0
58         raise "Node doesn't exist" unless self.NodeExists(target)
59         if target == self.head.value
60             if self.head == self.tail
61                 self.head = self.tail = nil
62             else
63                 self.head = self.head.next
64             end
65             self.length -= 1
66         else
67             current = self.head
68             while current != nil && current.next.value != target
69                 current = current.next
70             end
71             if current != nil
72                 current.next = current.next.next
73                 self.length -= 1
74                 if current.next == nil
75                     self.tail = current
76                 end
77             end
78         end
79         current = self.head
80         while current != nil
81             current.neighbors.Remove target
82             current = current.next
83         end
84     end

```

Line 76, Column 16

Tab Stop: 4 Policy

- Method: **AddNode**  
Add a node to the graph.
  - Method: **RemoveNode**  
Removes a node from the graph by its value. It also remove all the edges that contains at least one endpoint with these node.
  - Method: **Clear**  
Deletes all the information from the graph, leave it empty.

```

graph.rb
1 current = self.head
2 while current != nil do
3   if current.value == target
4     return current
5   end
6 end
7
8 def GetNode(target)
9   raise "Node doesn't exist" unless self.NodeExists target
10  current = self.head
11  while current != nil do
12    if current.value == target
13      return current
14    end
15  end
16
17 def NodeExists target
18   current = self.head
19   while current != nil
20     if current.value == target
21       return true
22     end
23   end
24   return false
25 end
26
27 def AddEdge from,to
28   if not self.NodeExists from
29     self.AddNode from
30   end
31   if not self.NodeExists to
32     self.AddNode to
33   end
34   current = self.GetNode from
35   current.neighbors.AddRight to
36 end
37
38 def RemoveEdge from,to
39   raise "Nodes don't exist" unless self.NodeExists from and self.NodeExists to
40   current = self.GetNode from
41   current.neighbors.Remove to
42 end
43
44 def IsConnected from,to
45   if not self.IsConnected from or not self.IsConnected to
46     return false
47   end
48   current = self.GetNode(from).neighbors.head
49   while current != nil
50     if current.value == to
51       return true
52     end
53   end
54   return false
55 end
56
57 def GetNodeGet
58   nodos = LinkedList.new
59   current = self.head
60   while current != nil
61     nodos.AddRight(current.value)
62     current = current.next
63   end
64   return nodos
65 end
66
67 def GetNeighbors target
68   vecinos = LinkedList.new
69   current = self.GetNode(target).neighbors.head
70   while current != nil
71     vecinos.AddRight(current.value)
72     current = current.next
73   end
74   return vecinos
75 end
76
77 def GetEdgeSet
78   puentes = LinkedList.new
79   current = self.head
80   ...

```

- Method: **GetNode**

Looks up a node in the neighbors list to the graph and returns a pointer to that node.

- Method: **NodeExists**

Return **true** if a node with the give value exists, otherwise returns false.

- Method: **AddEdge**

Add a border to the graphic. If a node does not exist in the chart, it will be added previously.

- Method: **RemoveEdge**

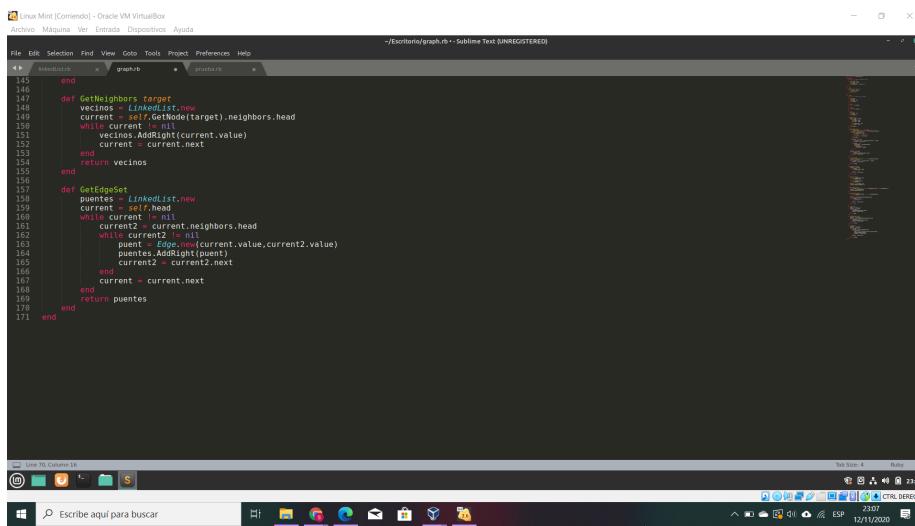
Removes an edge from the graph.

```

graph.rb
118 raise "Nodes don't exist" unless self.NodeExists from and self.NodeExists to
119 current = self.GetNode from
120 current.neighbors.Remove to
121 end
122
123 def IsConnected from,to
124   if not self.IsConnected from or not self.IsConnected to
125     return false
126   end
127   current = self.GetNode(from).neighbors.head
128   while current != nil
129     if current.value == to
130       return true
131     end
132   end
133   return false
134 end
135
136 def GetNodeGet
137   nodos = LinkedList.new
138   current = self.head
139   while current != nil
140     nodos.AddRight(current.value)
141     current = current.next
142   end
143   return nodos
144 end
145
146 def GetNeighbors target
147   vecinos = LinkedList.new
148   current = self.GetNode(target).neighbors.head
149   while current != nil
150     vecinos.AddRight(current.value)
151     current = current.next
152   end
153   return vecinos
154 end
155
156 def GetEdgeSet
157   puentes = LinkedList.new
158   current = self.head
159   ...

```

- Method: **IsConnected**  
Returns `true` if the graph contains an edge from  $u$  to  $v$ , otherwise returns `false`.
- Method: **GetNodeSet**  
Returns the list of all nodes in the graph.
- Method: **GetNeighbors**  
Returns the list of nodes that are neighbors of the specified node.



The screenshot shows a Linux Mint desktop environment with a Sublime Text window open. The window title is "graph.rb - Sublime Text (UNREGISTERED)". The code in the editor is:

```

145     end
146
147     def GetNeighbors target
148       vecinos = LinkedList.new
149       current = self.GetNode(target).neighbors.head
150       while current != nil
151         vecinos.AddRight(current.value)
152         current = current.next
153     end
154     return vecinos
155   end
156
157   def GetEdgeSet
158     puentes = LinkedList.new
159     current = self.head
160     while current != nil
161       next = current.neighbors.head
162       while next != nil
163         puentes.AddRight([current.value,next.value])
164         next = next.next
165       end
166       current = current.next
167     end
168   end
169 end
170 end
171 end

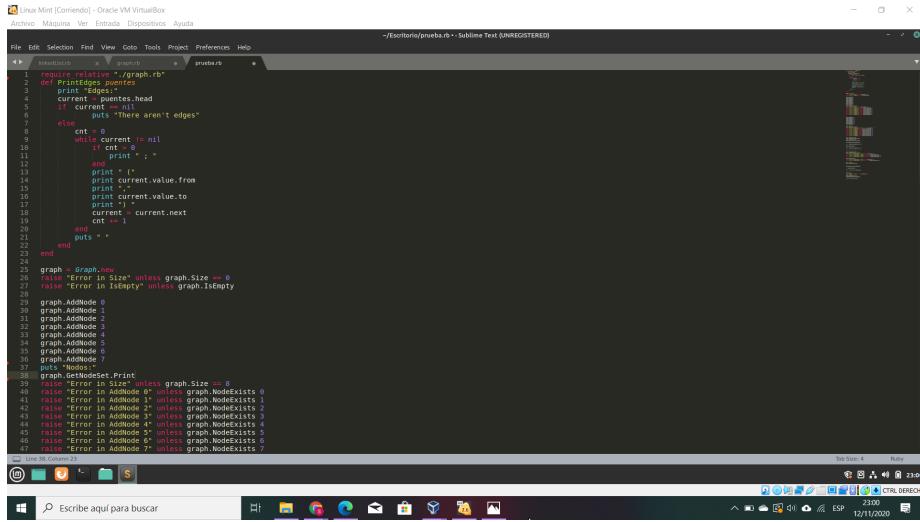
```

The status bar at the bottom of the screen shows "Sublime Text 4 Beta" and the date "12/11/2020".

- Method: **GetEdgeSet**  
Returns the list of all edges (tuple form) in the graph.

## 2.3 Ruby example program

To test that the implementation of the graph is correct, asserts will be used (in the case of Ruby it is `raise`) throughout the entire program. First, a function has been created that allows me to print the list of edges that I get when calling the `GetNeighbors` method, and that when the graph does not have edges it still prints `There aren't edges`. We start the program by creating a new graph and check that it is empty with the `Size` and `IsEmpty` methods. Then we add 8 nodes and print the list of nodes that is obtained with the `GetNodeSet` method, we proceed to verify that these nodes exist in the graph by calling `Node Exists`.

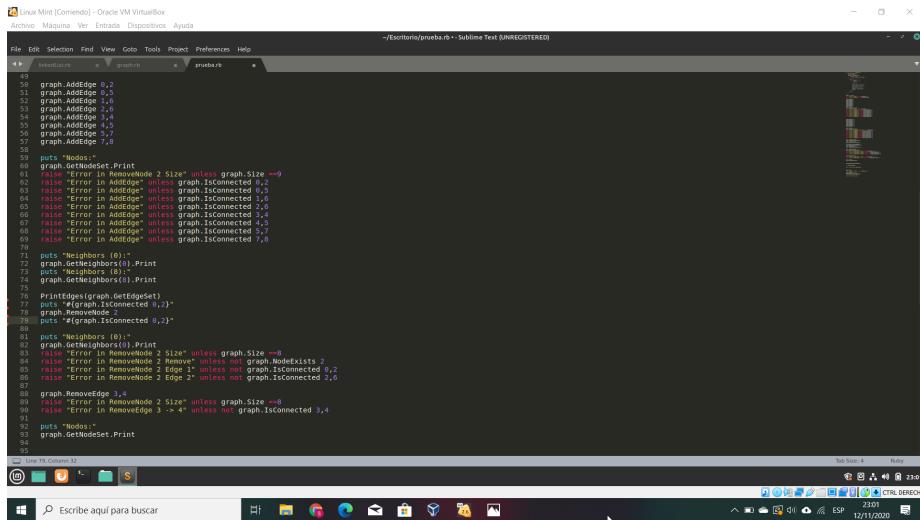


```

1  require relative "./graph.rb"
2  def PrintEdges(puentes)
3    edges = []
4    current = puentes.head
5    while current != nil
6      puts current.value
7      edges <= current.value.edges
8      current = current.next
9    end
10   puts "There aren't edges"
11 end
12
13 graph = Graph.new
14 if graph.size == 0
15   raise "Error in isEmpty" unless graph.isEmpty
16 graph.AddNode 0
17 graph.AddNode 1
18 graph.AddNode 2
19 graph.AddNode 3
20 graph.AddNode 4
21 graph.AddNode 5
22 graph.AddNode 6
23 graph.AddNode 7
24 puts "Nodes"
25 graph.GetNodes.Print
26 if graph.size == 0
27   raise "Error in size" unless graph.size == 0
28 graph.AddEdge 0,1
29 graph.AddEdge 0,2
30 graph.AddEdge 1,3
31 graph.AddEdge 2,3
32 graph.AddEdge 3,4
33 graph.AddEdge 4,5
34 graph.AddEdge 5,6
35 graph.AddEdge 6,7
36 puts "Edges"
37 graph.GetEdges.Print
38 if graph.size == 0
39   raise "Error in size" unless graph.size == 0
40 graph.AddEdge 0,1
41 raise "Error in AddEdge 1" unless graph.nodeExists 1
42 graph.AddEdge 0,2
43 raise "Error in AddEdge 2" unless graph.nodeExists 2
44 graph.AddEdge 1,3
45 raise "Error in AddEdge 3" unless graph.nodeExists 3
46 graph.AddEdge 2,3
47 raise "Error in AddEdge 4" unless graph.nodeExists 4
48 graph.AddEdge 3,4
49 raise "Error in AddEdge 5" unless graph.nodeExists 5
50 graph.AddEdge 4,5
51 raise "Error in AddEdge 6" unless graph.nodeExists 6
52 graph.AddEdge 5,6
53 raise "Error in AddEdge 7" unless graph.nodeExists 7
54
55 puts "IsConnected"
56 graph.GetNodes.Print
57 if graph.size == 0
58   raise "Error in RemoveNode 2 Size" unless graph.size == 0
59 raise "Error in RemoveNode 2" unless graph.isConnected 0,2
60 graph.RemoveNode 0
61 raise "Error in RemoveNode 2" unless graph.size == 0
62 raise "Error in RemoveNode 2" unless graph.isConnected 0,2
63 graph.GetNodes.Print
64 graph.GetEdges.Print
65 if graph.size == 0
66   raise "Error in RemoveEdge 2" unless graph.size == 0
67 raise "Error in RemoveEdge 2" unless graph.isConnected 0,2
68 graph.RemoveEdge 0,1
69 raise "Error in RemoveEdge 2 Size" unless graph.size == 0
70 raise "Error in RemoveEdge 2" unless graph.isConnected 0,2
71 graph.GetNodes.Print
72 graph.GetEdges.Print
73 puts "Neighbors"
74 graph.GetNeighbors(0).Print
75 graph.GetNeighbors(1).Print
76 graph.GetNeighbors(2).Print
77 graph.GetNeighbors(3).Print
78 graph.GetNeighbors(4).Print
79 graph.GetNeighbors(5).Print
80 graph.GetNeighbors(6).Print
81 graph.GetNeighbors(7).Print
82 puts "IsConnected"
83 graph.GetNodes.Print
84 if graph.size == 0
85   raise "Error in RemoveEdge 2" unless not graph.nodeExists 2
86 raise "Error in RemoveEdge 2 Edge 1" unless not graph.edgeExists 0,2
87 raise "Error in RemoveEdge 2 Edge 2" unless not graph.edgeExists 1,2
88 graph.RemoveEdge 3,4
89 raise "Error in RemoveEdge 2 Size" unless graph.size == 0
90 raise "Error in RemoveEdge 3 -> 4" unless not graph.isConnected 3,4
91 graph.GetNodes.Print
92 puts "Nodes"
93 graph.GetNodes.Print

```

Next we add edges even though those nodes do not exist previously, we print the nodes again and check the added edges with the `IsConnected` method. We will also call `GetNeighbors` to print the neighbors of some specific nodes, in addition we will print the existing edges. Now we will try `RemoveNode` to remove a node and we will make sure that the edges that were leaving or going to the node are also removed. Then we will remove an edge by calling `RemoveEdge` and we will check that those nodes are no longer connected.

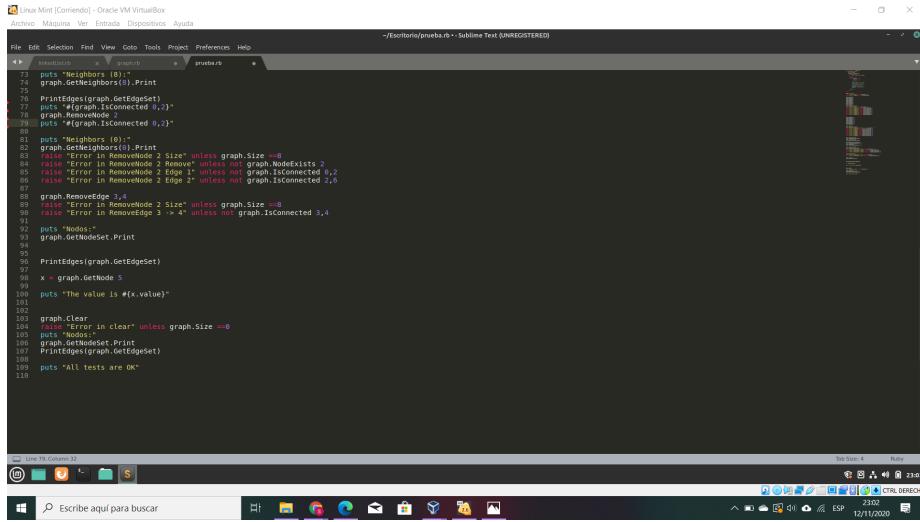


```

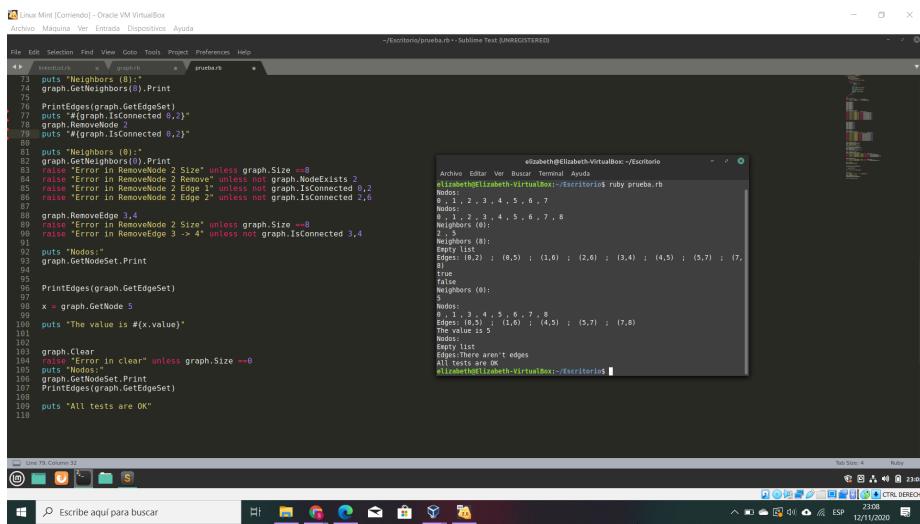
49 graph.AddEdge 0,3
50 graph.AddEdge 0,4
51 graph.AddEdge 0,5
52 graph.AddEdge 0,6
53 graph.AddEdge 0,7
54 graph.AddEdge 1,4
55 graph.AddEdge 1,5
56 graph.AddEdge 1,6
57 graph.AddEdge 1,7
58 graph.AddEdge 2,5
59 graph.AddEdge 2,6
60 graph.AddEdge 2,7
61 graph.AddEdge 3,6
62 graph.AddEdge 3,7
63 graph.AddEdge 4,6
64 graph.AddEdge 4,7
65 graph.AddEdge 5,7
66 graph.AddEdge 6,7
67 puts "IsConnected"
68 graph.GetNodes.Print
69 if graph.size == 0
70   raise "Error in RemoveNode 2 Size" unless graph.size == 0
71 raise "Error in RemoveNode 2" unless graph.isConnected 0,2
72 graph.RemoveNode 0
73 raise "Error in RemoveNode 2" unless graph.size == 0
74 raise "Error in RemoveNode 2" unless graph.isConnected 0,2
75 graph.GetNodes.Print
76 graph.GetEdges.Print
77 if graph.size == 0
78   raise "Error in RemoveEdge 2" unless graph.size == 0
79 raise "Error in RemoveEdge 2" unless graph.isConnected 0,2
80 graph.RemoveEdge 0,1
81 raise "Error in RemoveEdge 2 Size" unless graph.size == 0
82 raise "Error in RemoveEdge 2" unless graph.isConnected 0,2
83 graph.GetNodes.Print
84 puts "Neighbors"
85 graph.GetNeighbors(0).Print
86 graph.GetNeighbors(1).Print
87 graph.GetNeighbors(2).Print
88 graph.GetNeighbors(3).Print
89 graph.GetNeighbors(4).Print
90 graph.GetNeighbors(5).Print
91 graph.GetNeighbors(6).Print
92 graph.GetNeighbors(7).Print
93 puts "IsConnected"
94 graph.GetNodes.Print

```

Finally, print out the remaining edges before using the `Clear` method to leave the graph empty again.

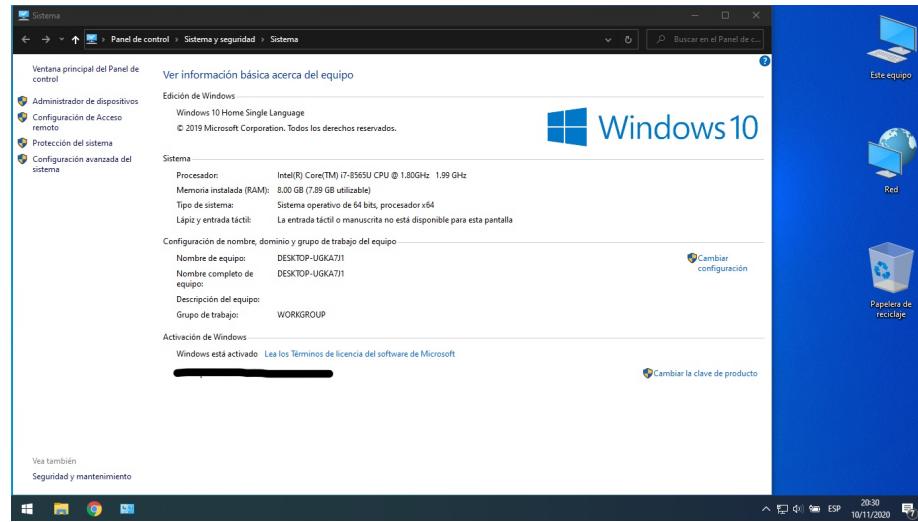


The following image is the execution of the example program.

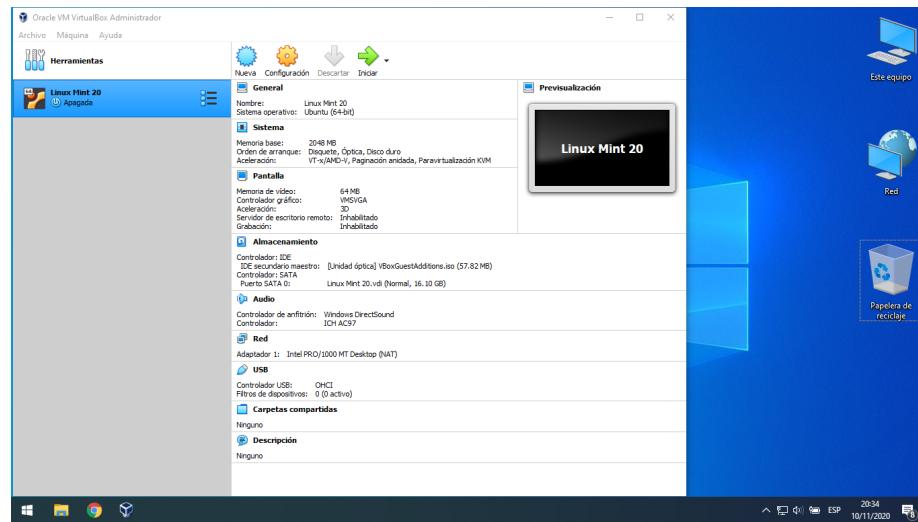


### 3 Saras Rivera, André Edgardo

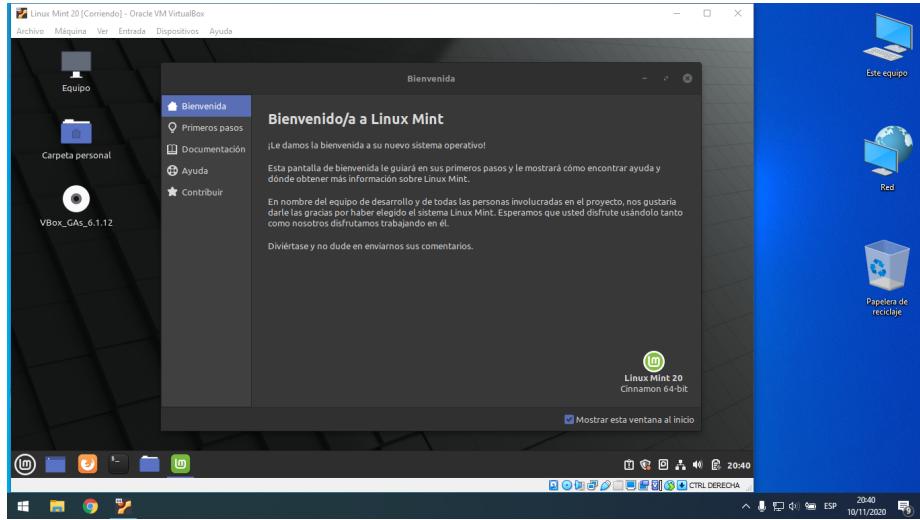
#### 3.1 Resources and Oracle VM VirtualBox parameters



Computer resources.



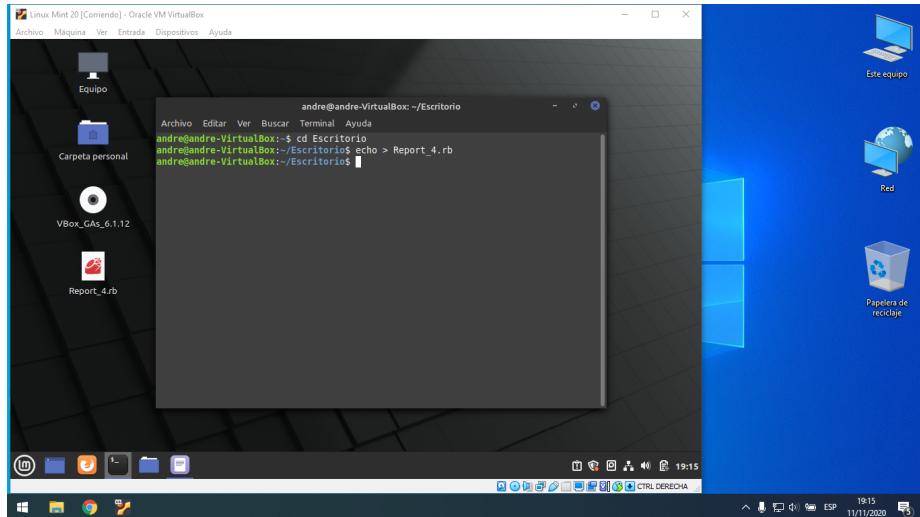
VirtualBox parameters.



Linux Mint desktop.

### 3.2 Graph implementation using adjacency matrix

To start working create the file where the TAD will be encoded.



Within the file .rb the header is created and then the elements of the graph (edge and vertex) are implemented.

```

Linux Mint 20 [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Report_4.rb (-/Escritorio)
Archivo Editar Ver Buscar Herramientas Documentos Ayuda
Report_4.rb x
begin
  File.open('graph.h')
  puts "Note: we have a graph implemented in Ruby language using adjacency matrices."
end

# Implementation of an edge with initial and end vertex, and weight.
class Edge
  def initialize(starVertex, endVertex, weight)
    @starVertex = starVertex
    @endVertex = endVertex
    @weight = weight
  end
  attr_reader :starVertex, :endVertex, :weight
end

# Implementation of a labeled vertex.
class Vertex
  def initialize(value, label)
    @value = value
    @label = label
  end
  attr_reader :value, :label
end

# Implementation of a graph with its adjacency and weight matrix. Additionally implementation of support...

```

Within the implementation of the graph the main thing is to initialize and create the methods to form the adjacency matrix and the weight matrix (which uses the incidences), consequently we will create the support methods.

```

Report_4.rb x
# Implementation of a graph with its adjacency and weight matrix. Additionally implementation of support...
class Graph
  # Initialization of graph with a parameter that says if it is directed or not.
  def initialize(vertices, edges, isDirected)
    @vertices = vertices
    @edges = edges
    @isDirected = isDirected
  end
  attr_reader :vertices, :edges, :isDirected

  # Create the adjacency matrix.
  def adjacencyMatrix
    matrix = Array.new(@vertices.length) { Array.new(@vertices.length, 0) }
    v = 0
    for vertex in @vertices
      adjacents = adjacentVertices(vertex)
      for adjacent in adjacents
        matrix[v][adjacent - 1] += 1
      end
      v += 1
    end
    return matrix
  end

  # Function that returns all vertices adjacent to the given vertex.
  def adjacentVertices(vertex)
    edges.select { |edge| edge.start == vertex }
  end

```

Now, the function that will be in the main will be created and it will ask for the data and fill the adjacency matrix. It is important to mention that the vertices will have a label that is a letter, if the number of letters is finished it will continue with vertices without a label, but you can place any other.

The screenshot shows a Linux Mint 20 desktop environment with a blue theme. A window titled "Report\_4.rb (-/Escritorio)" is open in the foreground, displaying Ruby code for graph operations. The code includes methods for getting adjacent vertices and creating a weight matrix. The desktop background features the standard Windows logo. On the right side, there are several icons: "Este equipo" (This PC), "Red", and "Papelera de reciclaje" (Recycle Bin). The bottom taskbar has icons for various applications like a browser, file manager, and terminal. The system tray shows network status, battery level (19:26), and system notifications.

```
# Function that returns all vertices adjacent to the given vertex.
def adjacentVertices(vertex)
  adjacentVertices = []
  for edge
    if (vertex.value == edge.starVertex.value)
      adjacentVertices.push(edge.endVertex.value)
    elsif (!edge.bidirected && vertex.value == edge.endVertex.value)
      adjacentVertices.push(edge.starVertex.value)
    end
  end
  adjacentVertices.sort()
  return adjacentVertices
end

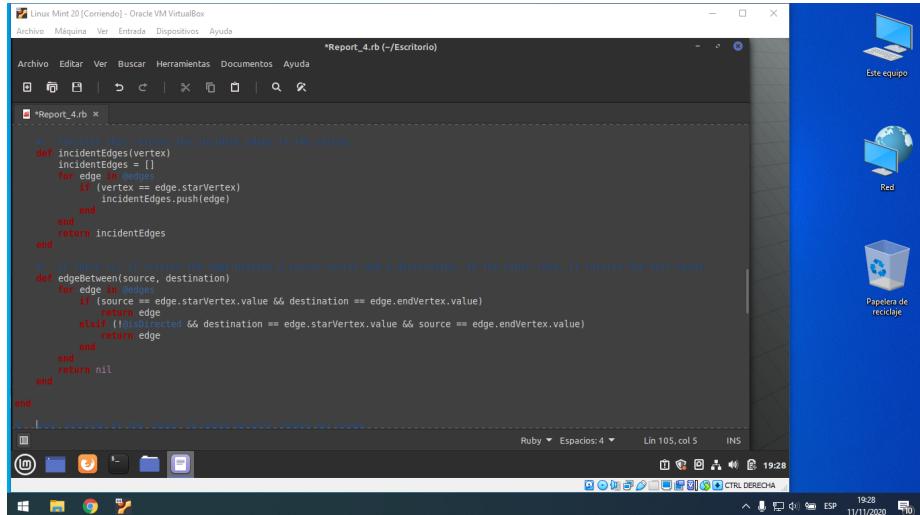
# Create the weight matrix.
def weightMatrix
  matrix = Array.new(vertices.length, []) { Float::INFINITY }
  v = 0
  for vertex in vertices
    incidents = incidentEdges(vertex)
    for incident in incidents
      matrix[v, incident.endVertex.value - 1] = incident.weight
    end
    v += 1
  end
  return matrix
end
```

The screenshot shows a dual-boot or virtual machine setup. The left side displays a Linux Mint 20 desktop environment with a dark theme. A terminal window titled 'Report\_4.rb (-/Escritorio)' is open, displaying Ruby code related to graph theory. The code defines a weight matrix and incident edges for vertices. The right side shows a Windows 10 desktop background with icons for 'Este equipo' (This PC), 'Red', and 'Papelera de reciclaje' (Recycle Bin). The taskbar at the bottom includes icons for the terminal, file browser, and system tray.

```
# Create the weight matrix.
def weightMatrix
  matrix = Array.new(@vertices.length, [])
  v = 0
  for vertex in @vertices
    incidentEdges(vertex)
    for incident in incidentEdges(vertex)
      matrix[v][incident.endVertex.value - 1] = incident.weight
    end
    v += 1
  end
  return matrix
end

# Function that returns the incident edges to the vertex.
def incidentEdges(vertex)
  incidentEdges = []
  for edge in @edges
    if (vertex == edge.starVertex)
      incidentEdges.push(edge)
    end
  end
  return incidentEdges
end

# Shows the total weight of the edges between two courses.
def totalWeight(courses)
  weight = 0
  for course in courses
    for vertex in course
      weight += weightMatrix[vertex.value][course[0].value - 1]
    end
  end
  return weight
end
```



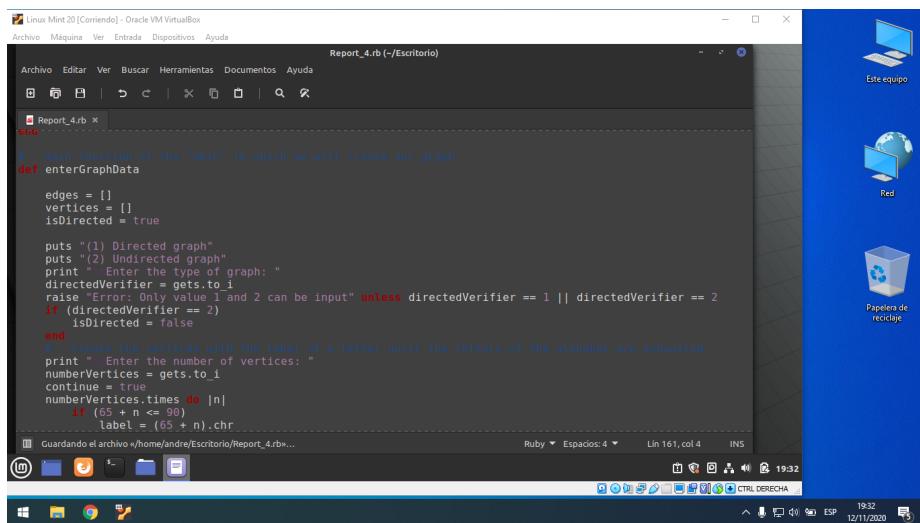
```
*Report_4.rb (~/Escritorio)
Archivo Editar Ver Buscar Herramientas Documentos Ayuda
Report_4.rb x
#<function> that returns the incident edges to the vertex
def incidentEdges(vertex)
    incidentEdges = []
    for edge in edges
        if (vertex == edge.starVertex)
            incidentEdges.push(edge)
        end
    end
    return incidentEdges
end

# If there is, it returns the edge between a source vertex and a destination. In the other case, it returns the null value.
def edgeBetween(source, destination)
    for edge in edges
        if (source == edge.starVertex.value && destination == edge.endVertex.value)
            return edge
        elsif (!isDirected && destination == edge.starVertex.value && source == edge.endVertex.value)
            return edge
        end
    end
    return nil
end

#<function> of the class to which we will create our graph
class Graph
end

Este equipo
Red
Papelera de reciclaje
```

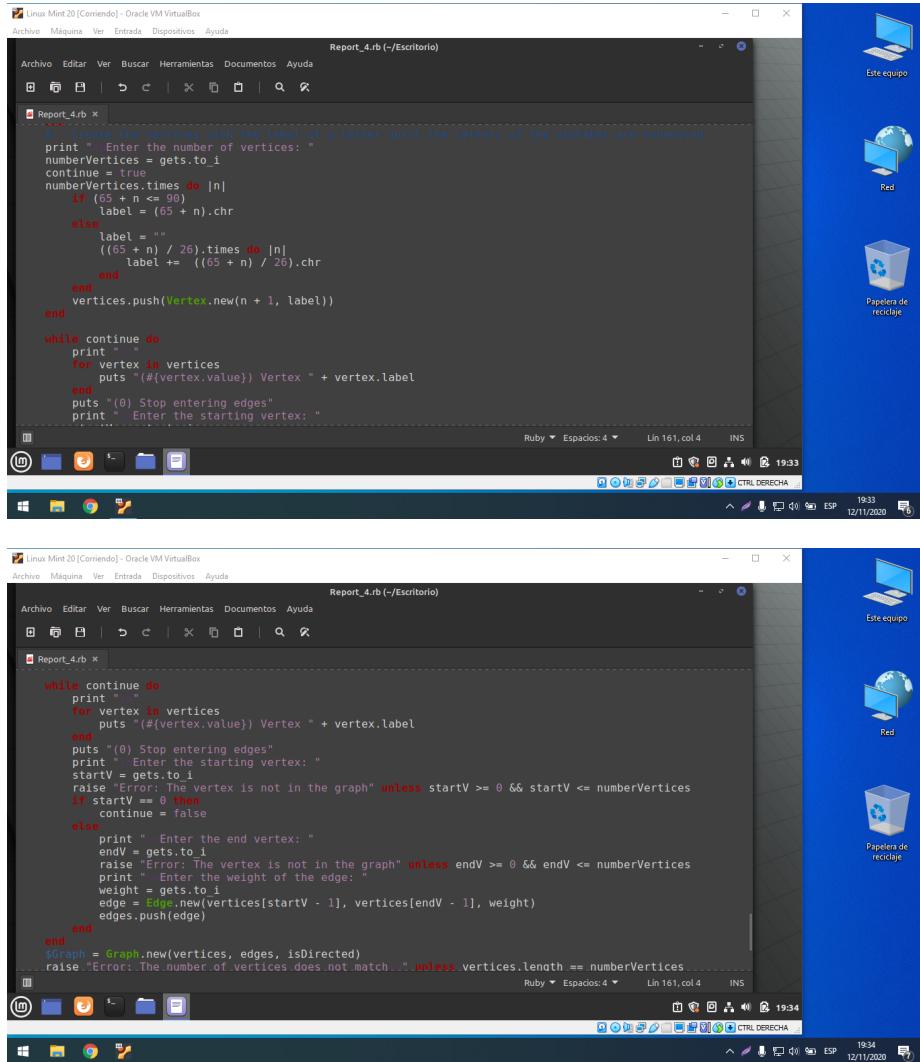
This screenshot shows a terminal window on a Linux Mint 20 desktop. The window title is "Report\_4.rb (~/Escritorio)". The code in the terminal defines two methods: `incidentEdges` and `edgeBetween`. The `incidentEdges` method takes a vertex and returns an array of edges incident to it. The `edgeBetween` method takes two vertices and returns the edge between them if it exists, or `nil` otherwise. It also handles the case for undirected graphs where the edge can be traversed in both directions.



```
Report_4.rb (~/Escritorio)
Archivo Editar Ver Buscar Herramientas Documentos Ayuda
Report_4.rb x
#<function> of the "main" in which we will create our graph
def enterGraphData
    edges = []
    vertices = []
    isDirected = true

    puts "(1) Directed graph"
    puts "(2) Undirected graph"
    print "Enter the type of graph: "
    directedVerifier = gets.to_i
    raise "Error: Only value 1 and 2 can be input" unless directedVerifier == 1 || directedVerifier == 2
    if (directedVerifier == 2)
        isDirected = false
    end
    # Create the vertices with the label of a letter until the letters of the alphabet are exhausted
    print "Enter the number of vertices: "
    numberVertices = gets.to_i
    continue = true
    numberVertices.times do |n|
        if (65 + n <= 90)
            label = (65 + n).chr
        end
    end
    Guardando el archivo ~/home/andres/Escritorio/Report_4.rb...
Ruby Espacios: 4 Lin 161, col 4 INS
Este equipo
Red
Papelera de reciclaje
```

This screenshot shows the beginning of a Ruby script named `Report\_4.rb`. The script starts with a function `enterGraphData` that prompts the user to choose between a directed or undirected graph. It then initializes arrays for edges and vertices, and a boolean `isDirected`. The script uses `gets` to read the user's input and `raise` to handle invalid inputs. It then enters a loop to create vertices labeled with letters from A to Z. Finally, it saves the file to the desktop.



```

Linux Mint 20 [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Report_4.rb (~/Escritorio)
Archivo Editar Ver Buscar Herramientas Documentos Ayuda
Report_4.rb x
--> Create the vertices with the label of a letter until the letters of the alphabet are exhausted
print " Enter the number of vertices: "
numberVertices = gets.to_i
continue = true
numberVertices.times do |n|
  if (65 + n <= 90)
    label = (65 + n).chr
  else
    label = ""
    ((65 + n) / 26).times do |n|
      label += ((65 + n) / 26).chr
    end
  end
  vertices.push(Vertex.new(n + 1, label))
end

while continue do
  print "\n"
  for vertex in vertices
    puts "#{vertex.value}) Vertex " + vertex.label
  end
  puts "(0) Stop entering edges"
  print " Enter the starting vertex: "

```

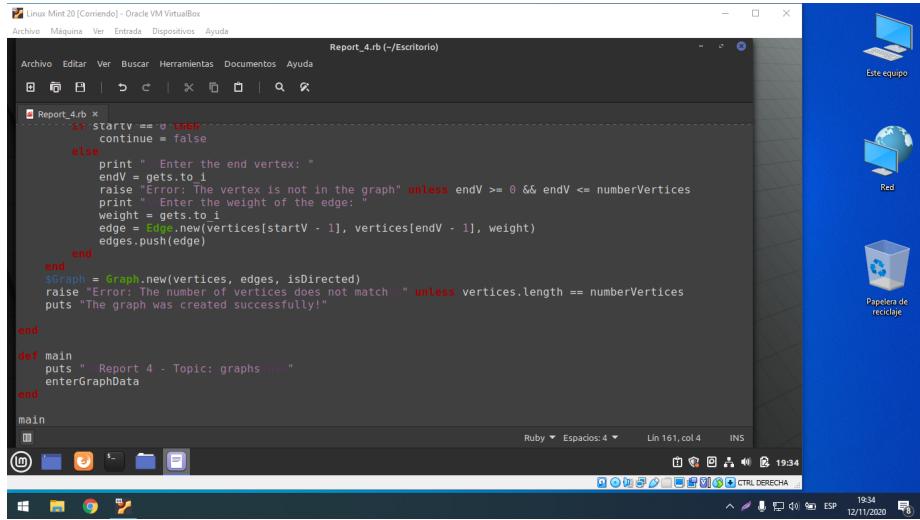
  

```

Linux Mint 20 [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Report_4.rb (~/Escritorio)
Archivo Editar Ver Buscar Herramientas Documentos Ayuda
Report_4.rb x
while continue do
  print "\n"
  for vertex in vertices
    puts "#{vertex.value}) Vertex " + vertex.label
  end
  puts "(0) Stop entering edges"
  print "\n Enter the starting vertex: "
  startV = gets.to_i
  raise "Error: The vertex is not in the graph" unless startV >= 0 && startV <= numberVertices
  if startV == 0 then
    continue = false
  else
    print "\n Enter the end vertex: "
    endV = gets.to_i
    raise "Error: The vertex is not in the graph" unless endV >= 0 && endV <= numberVertices
    print "\n Enter the weight of the edge: "
    weight = gets.to_i
    edge = Edge.new(vertices[startV - 1], vertices[endV - 1], weight)
    edges.push(edge)
  end
end
soraph = Graph.new(vertices, edges, isDirected)
raise "Error: The number of vertices does not match" unless vertices.length == numberVertices

```

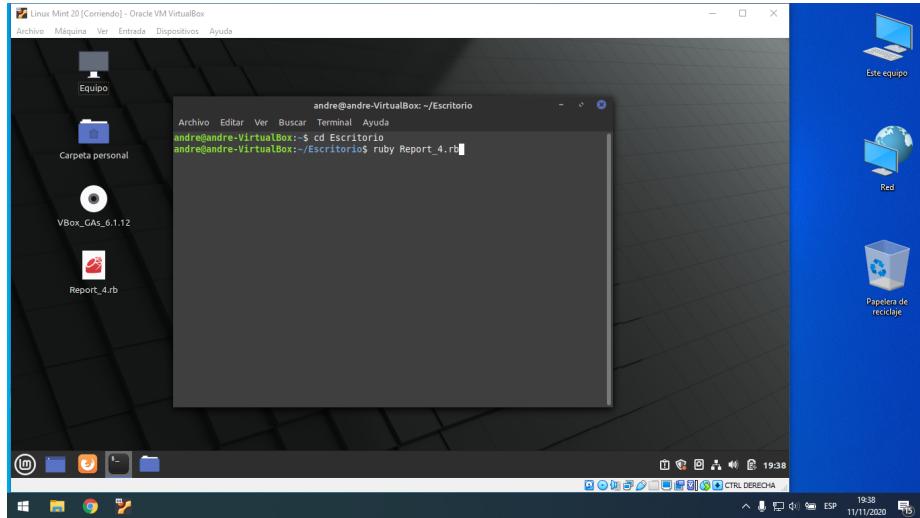
Finally, the main function is created.



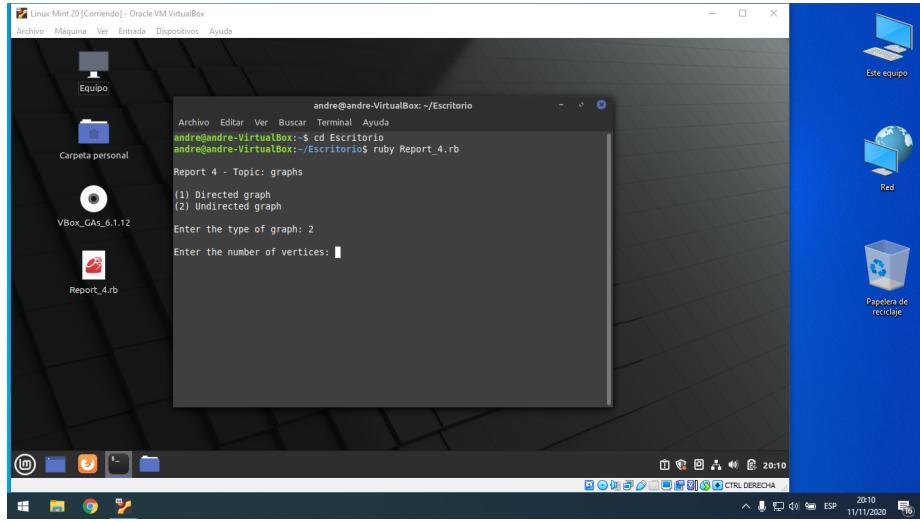
```
Linux Mint 20 [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Report_4.rb (~/Escritorio)
Archivo Editar Ver Buscar Herramientas Documentos Ayuda
Report_4.rb x
startV == v ? true : 
continua = false
else
print " Enter the end vertex: "
endV = gets.to_i
raise "Error: The vertex is not in the graph" unless endV >= 0 && endV <= numberVertices
print " Enter the weight of the edge: "
weight = gets.to_i
edge = Edge.new(vertices[startV - 1], vertices[endV - 1], weight)
edges.push(edge)
end
graph = Graph.new(vertices, edges, isDirected)
raise "Error: The number of vertices does not match!" unless vertices.length == numberVertices
puts "The graph was created successfully!"
end
def main
puts "\nReport 4 - Topic: graphs\n"
enterGraphData
end
main
Ruby Espacios:4 Lin 161, col.4 INS
Ruby 19:34
12/11/2020
```

### 3.3 Ruby example program

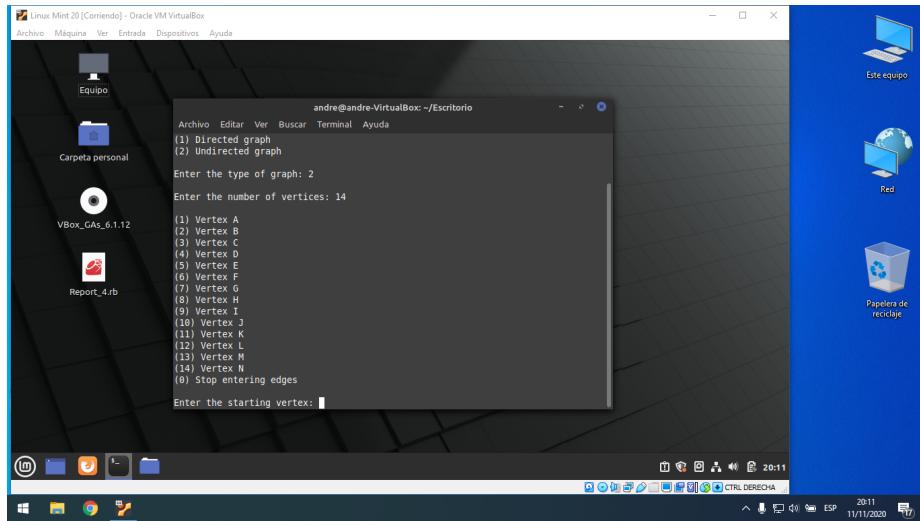
Let's test the program. First of all, we have to indicate whether it is a directed graph or not

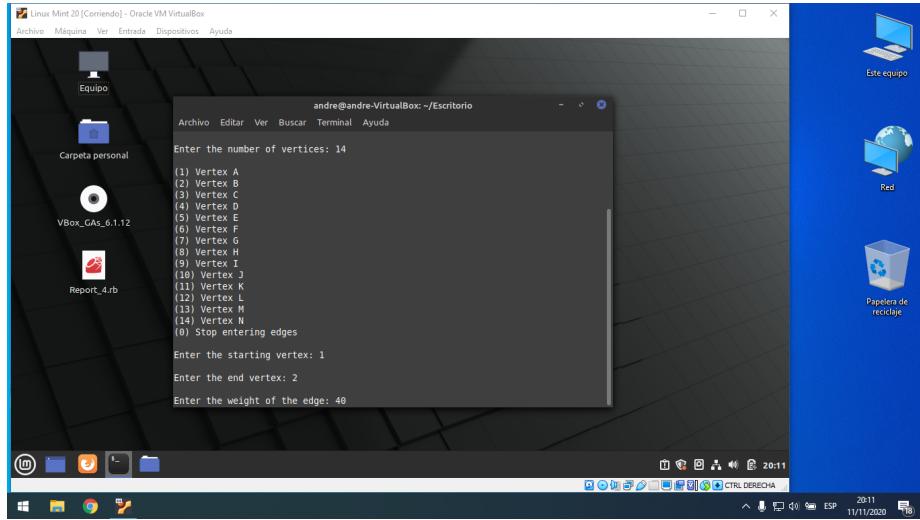


```
Linux Mint 20 [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Equipo
Carpeta personal
VBox_GAs_6.1.12
Report_4.rb
andre@andre-VirtualBox: ~/Escritorio
Terminal
andre@andre-VirtualBox:~$ cd Escritorio
andre@andre-VirtualBox:~/Escritorio$ ruby Report_4.rb
```

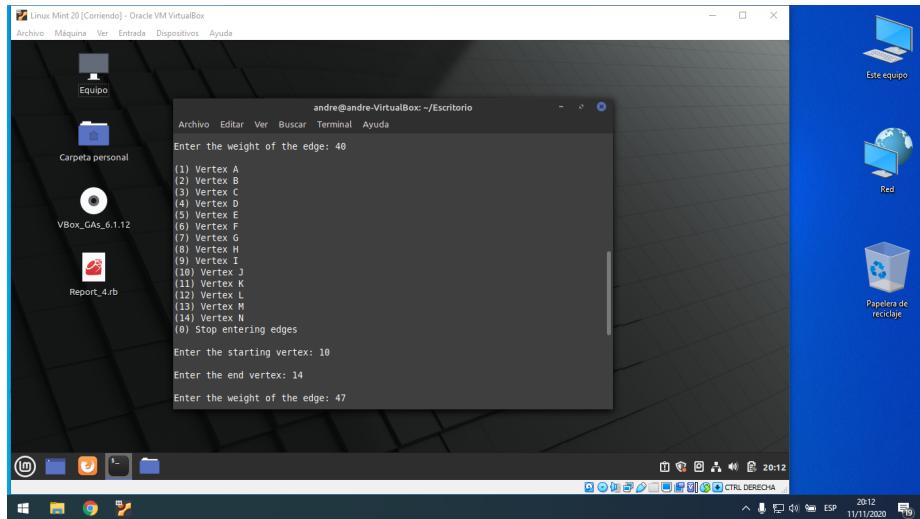


When entering the number of vertices, these are created automatically with their respective label.





Then, we enter the edges one by one, placing their beginning, end and weight vertex (it can have 0 or negative weight without any problem).



To finish the process, we enter the value of 0 as the initial node. If everything has been correct, then the message of successful creation of the graph will appear on screen.

