

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
a20182778@pucp.edu.pe
andre.saras@pucp.edu.pe

Fifth 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.

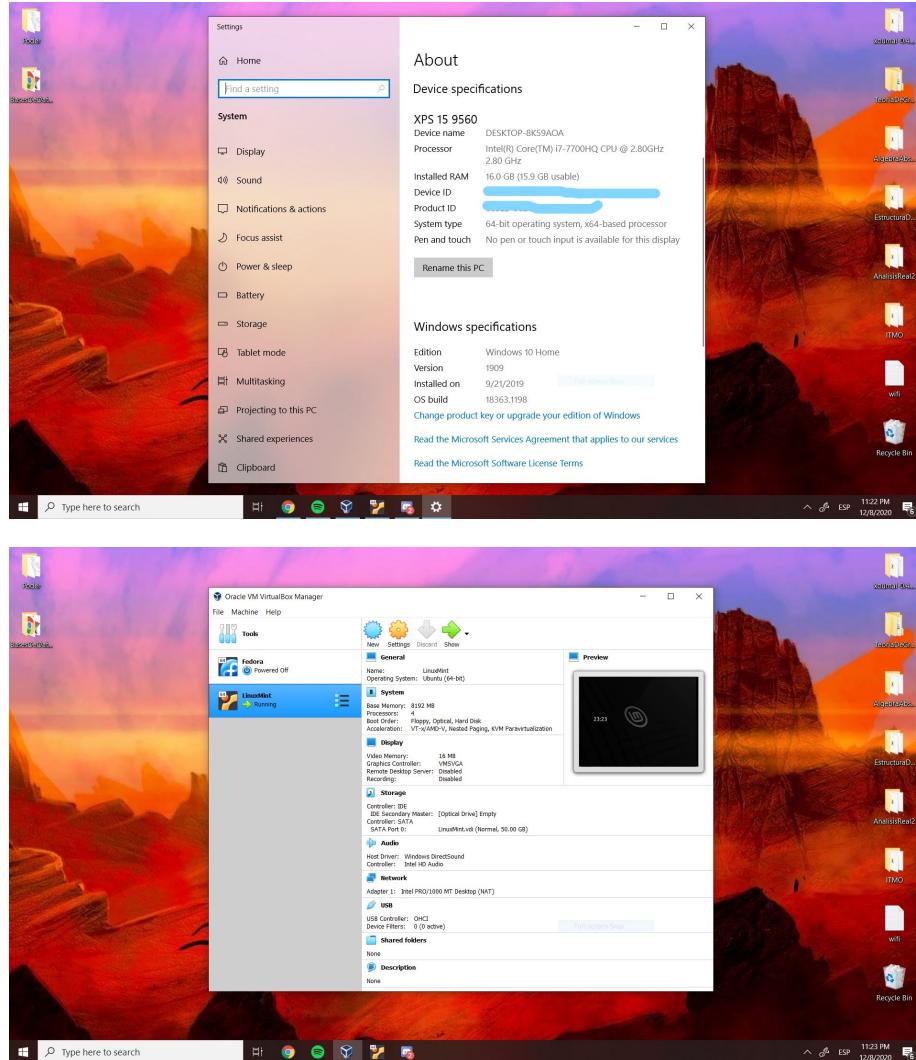
Contents

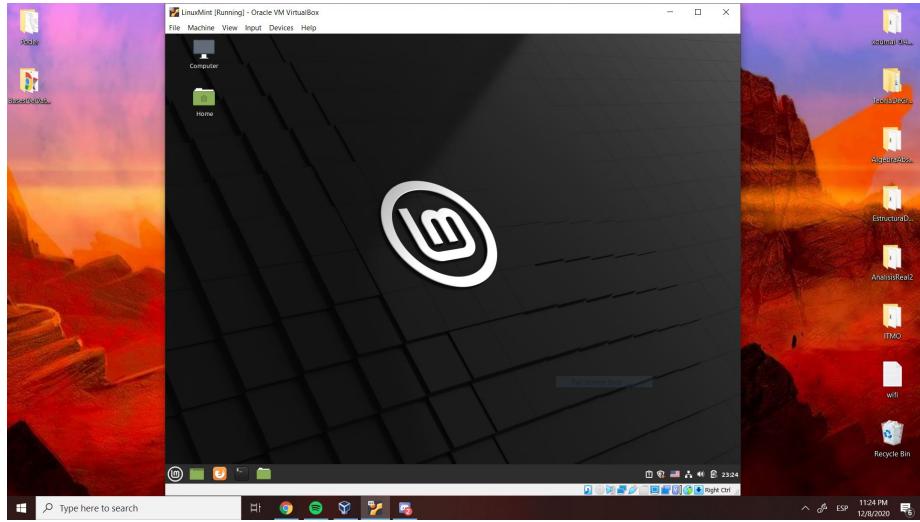
1 Loaiza Vasquez, Manuel Alejandro	2
1.1 Resources and Oracle VM VirtualBox parameters	2
2 Oyarce Tocto, Elizabeth Patricia	3
2.1 Resources and Oracle VM VirtualBox parameters	3
3 Saras Rivera, André Edgardo	4
3.1 Resources and Oracle VM VirtualBox parameters	5
4 Single Source Shortest Path	6
4.1 Single Source Shortest Path Problem	6
4.2 Main Algorithm	7
4.3 Implementation	7
4.3.1 Graph	7
4.3.2 Min-Heap	9
4.3.3 Dijkstra's Algorithm	12
4.3.4 Testing	13

1 Loaiza Vasquez, Manuel Alejandro

This member has done 4.1, 4.2, 4.3.1, 4.3.3 and 4.3.4.

1.1 Resources and Oracle VM VirtualBox parameters





2 Oyarce Tocto, Elizabeth Patricia

This member has done 4.2, 4.3.1. and 4.3.2.

2.1 Resources and Oracle VM VirtualBox parameters

Sistema

Ver información básica acerca del equipo

Edición de Windows

Windows 10 Home Single Language
© 2019 Microsoft Corporation. Todos los derechos reservados.

Sistema

Procesador: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.21 GHz
Memoria instalada (RAM): 8.00 GB / 7.88 GB utilizadas
Tipo de sistema: Sistema operativo de 64 bits, procesador x64
Lápiz y entrada táctil: La entrada táctil o manuática no está disponible para esta pantalla

Windows 10

Lenovo

Información de soporte técnico

Cambiar configuración

Configuración de nombre, dominio y grupo de trabajo del equipo

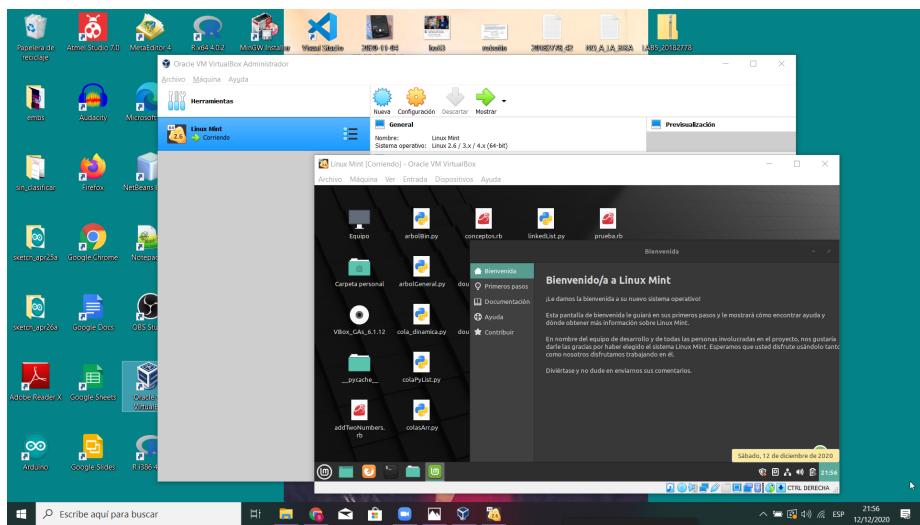
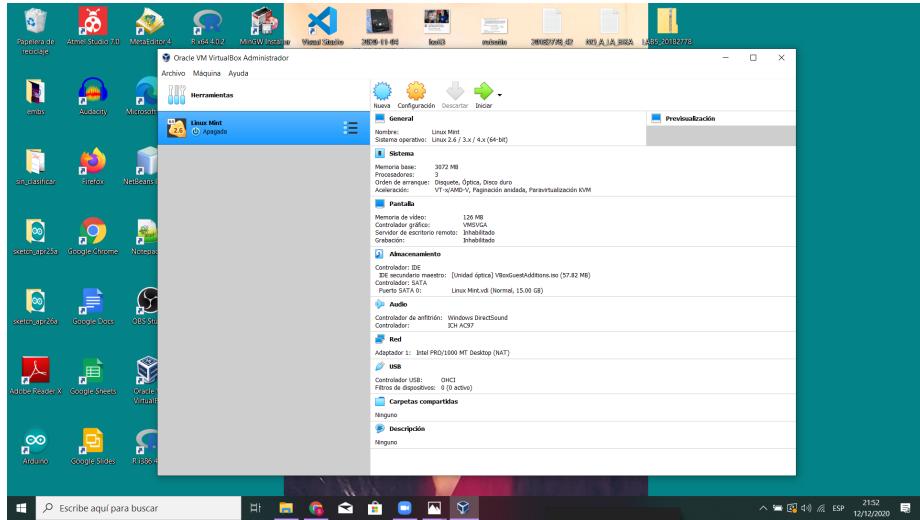
Nombre de equipo: LAPTOP-FSCAO6
Nombre completo de equipo:
Descripción del equipo:
Grupo de trabajo: WORKGROUP

Activación de Windows

Windows está activado | Leer los Términos de licencia del software de Microsoft
Id. del producto: 00327-30664-12842-AAOEM | Cambiar la clave de producto

Vea también

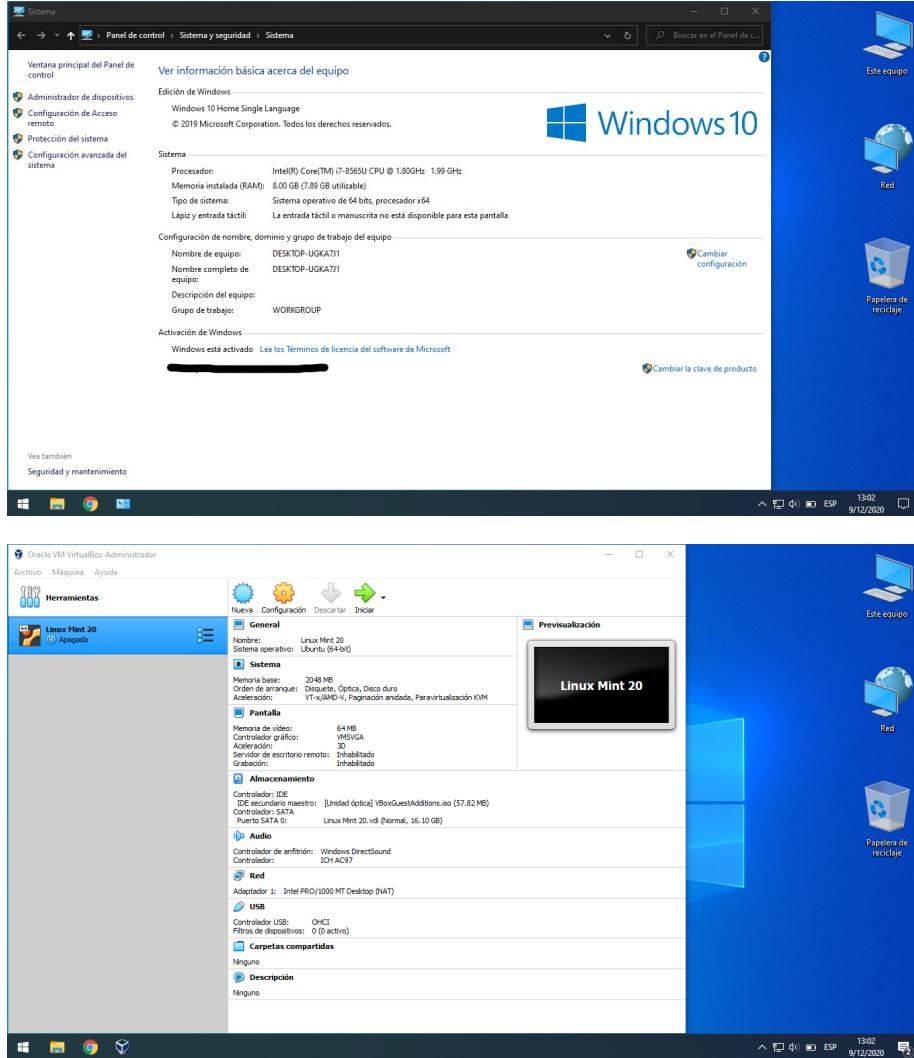
Seguridad y mantenimiento

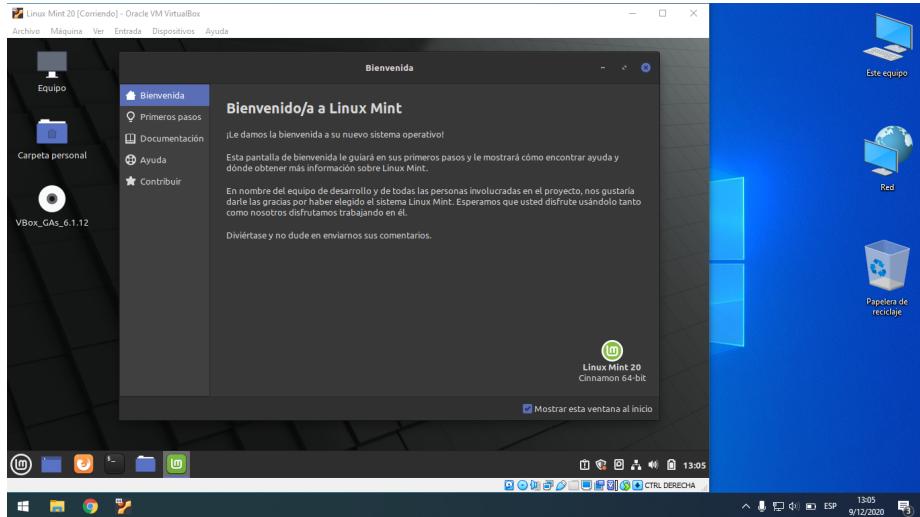


3 Saras Rivera, André Edgardo

This member has done 4.2, 4.3.1. and 4.3.2.

3.1 Resources and Oracle VM VirtualBox parameters





4 Single Source Shortest Path

4.1 Single Source Shortest Path Problem

Finding a shortest path between two nodes of a graph is an important problem that has many practical applications. For example, a natural problem related to a road network is to calculate the shortest possible length of a route between two cities, given the lengths of the roads.

In an unweighted graph, the length of a path equals the number of its edges and we can simply use a breadth-first search to find a shortest path as we did in our previous report. However, in this report we focus on weighted graphs where more sophisticated algorithms are needed for finding shortest paths.

Given a directed graph $G(V, E)$, with weight function $w : E \rightarrow R$. The weight $w(p)$ of a path $p = \langle v_0, v_1, \dots, v_k \rangle$ is the sum of the weights of its edges

$$w(p) = \sum_{i=1}^k w(v_{i-1}, v_i).$$

We define the shortest path weight $\delta(u, v)$ from u to v by

$$\delta(u, v) = \begin{cases} \min\{w(p) \mid \forall p = \langle u, \dots, v \rangle\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise.} \end{cases}$$

A shortest path from vertex u to v is then defined as any path p with weight $w(p) = \delta(u, v)$.

4.2 Main Algorithm

Dijkstra's algorithm finds shortest paths from the starting node to all nodes of the graph. The algorithm requires that there are no negative weight edges in the graph.

Dijkstra's algorithm maintains distances to the nodes and reduces them during the search. At each step, Dijkstra's algorithm selects a node that has not been processed yet and whose distance is as small as possible. Then, the algorithm goes through all edges that start at the node and reduces the distances using them. Dijkstra's algorithm is efficient because it only processes each edge in the graph once, using the fact that there are no negative edges.

4.3 Implementation

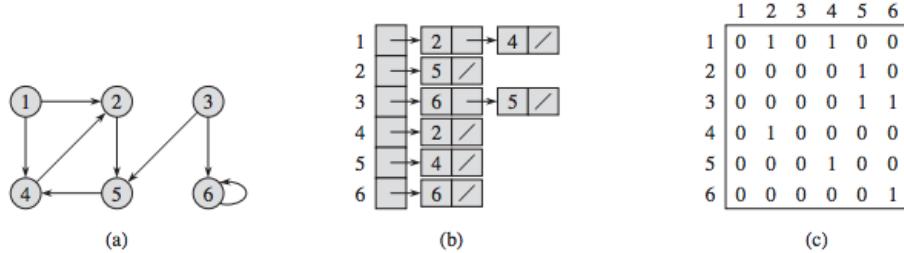
An efficient implementation of Dijkstra's algorithm requires that we can efficiently find the minimum distance to a node that has not been processed. An appropriate data structure for this is min-heap that contains the remaining nodes ordered by their distances. Using a min-heap, the next node to be processed can be retrieved in logarithmic time. Due to not being able to modify the value in the min-heap, we are going to add a new instance of a node to the min-heap always when its distance changes.

Our implementation of Dijkstra's algorithm calculates the minimum distances from a specific node to all other nodes of the graph. The graph is stored as adjacency lists so that `adj[u]` contains a pair (v, w) always when there is an edge from node u to v with weight w . The min-heap q contains pairs of the form (d, x) meaning that the current distance to node x is d . The array `distance` contains the distance to each node and the array `parent` the last node that has made each node to reduce its distance.

The implementation is as follows:

4.3.1 Graph

The structures we have to research and implement in this report are graphs. There are two common ways to represent a graph: using an adjacency matrix or using an adjacency list. Given a graph $G = (V, E)$, the adjacency matrix representation is easier because we are going to create an array of $|V|^2$ units of memory and a_{ij} will represent if an edge between the nodes i and j exists. However, we need a lot of memory to represent the graph and, also, there will be too much unused space. As a consequence, the adjacency list is the most efficient, in terms of space complexity, way to represent a graph because we are going to create nodes on each list that represent each existing edge if necessary.



As we can see, we are going to represent a graph like the one in (a) using an array of linked lists in (b). In Ruby, we can use `Hash`, which is the implementation of a hash table, instead of a linked list and the time complexity is not going to be affected.

The class `Graph` represents a graph with the specified nodes and edges. I have created a class `Edge`. This is a simplified version of the implementation from the previous report because we are not going to use all the methods and we are going to focus on the edges instead of the nodes.

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

- Method: `initialize`
Usage: `graph = Graph.new(n)`
Creates a graph with n nodes.
- Method: `Size`
Usage: `size = graph.Size`
Returns the number of nodes in the graph.
- Method: `AddEdge`
Usage: `graph.AddEdge(from, to, weight)`
Adds an oriented weighted edge to the graph.
- Method: `GetNeighbors`
Usage: `neighbors = graph.GetNeighbors(node_id)`
Returns the set of edges that start at the specified node.

The image shows a dual-monitor setup. The left monitor displays a Linux Mint desktop environment with a terminal window open. The terminal window title is "LinuxMint (Running) - Oracle VM VirtualBox". The terminal content is as follows:

```

1 require "set"
2
3 class Edge
4   attr_accessor :to, :weight
5
6   def initialize to, weight
7     self.to = to
8     self.weight = weight
9   end
10 end
11
12 class Graph
13   attr_accessor :adj, :nodes
14
15   def initialize n
16     self.adj = Hash.new
17     self.nodes = n
18     for u in 0 .. n - 1
19       self.adj[u] = Set.new
20     end
21   end
22
23   def AddEdge from, to, weight
24     self.adj[from] << Edge.new(to, weight)
25   end

```

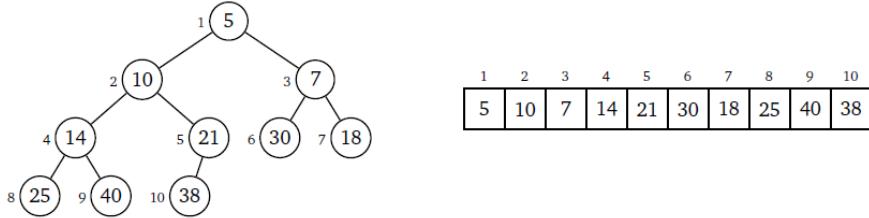
The right monitor displays a Windows 10 desktop environment with a file explorer window open. The file explorer window shows the following files:

- AnálisisReal2
- EstructuraD...
- AgostoReal...
- BasesDeDatos...
- Recycle Bin
- wifi
- IMO

Both monitors have a sunset landscape wallpaper.

4.3.2 Min-Heap

A Min-Heap is a complete binary tree where the element in the root is less than all elements in the left and right subtrees, and both children satisfy this property too. It is usually represented by an arrangement where a node stored at position i has its left child at position $2i$ and its right child in position $2i + 1$



We need the class **Path** to be able to work. A path of length k from a vertex u to a vertex v in a graph $G = (V, E)$ is a sequence $\langle v_0, v_1, v_2, \dots, v_k \rangle$ of distinct vertices such that $u = v_0$ and $v_k = v$, and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \dots, k$. The length of the path is the number of edges in the path. There is always a 0-length path from u to u . In particular, since all the paths used in the algorithm start at the same initial vertex, it is only necessary to put where it ends and the weight of the path to define the class.

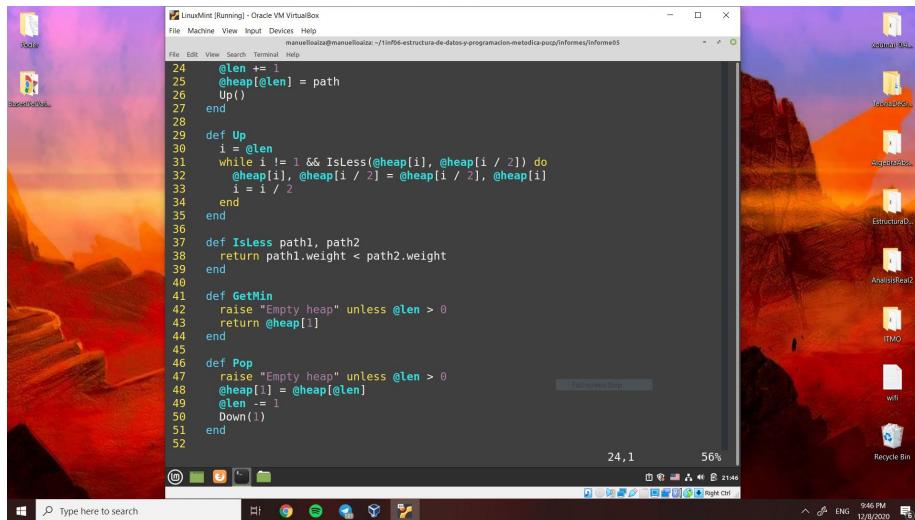
```

1 class Path
2     def initialize weight, node
3         @weight = weight
4         @node = node
5     end
6     attr_reader :weight, :node
7 end
8
9 class MinHeap
10    MAX = 100
11
12    def initialize
13        @heap = Array.new(MAX + 1)
14        @len = 0
15    end
16    attr_reader :len, :heap
17
18    def IsEmpty?
19        return @len == 0
20    end
21
22    def Push path
23        raise "Insufficient space" unless @len < MAX
24        @len += 1
25        @heap[@len] = path
26        Up()
27    end
28
29    def Up
"min_heap.rb" 70L, 1187C
1,1           Top

```

- Method: **initialize**
Usage: `minHeap = MinHeap.new`
Creates a min-heap using an array with initial length 0.
- Method: **IsEmpty?**
Usage: `minHeap.IsEmpty?`
Returns a boolean value true if it is empty and false otherwise.
- Method: **Push**
Usage: `minHeap.Push(path)`
Places the new path at the end of the arrangement and then by means of the *Up* function makes it float to its proper position.

- Method: **Up**
Usage: `Up()`
Swaps the path with its parent until it is less than or equal, or the root is reached.
- Method: **IsLess**
Usage: `var = IsLess(path1,path2)`
Compare two paths by their weight attribute.
- Method: **GetMin**
Usage: `min = minHeap.GetMin`
Returns the minimum element of the arrangement.
- Method: **Pop**
Usage: `minHeap.Pop`
Swaps the first path with the last path and then by means of the *Down* function makes it fall to its proper position.
- Method: **Down**
Usage: `Down(id)`
Swaps the path with its minimum son until it is higher than or equal, or a leaf is reached.

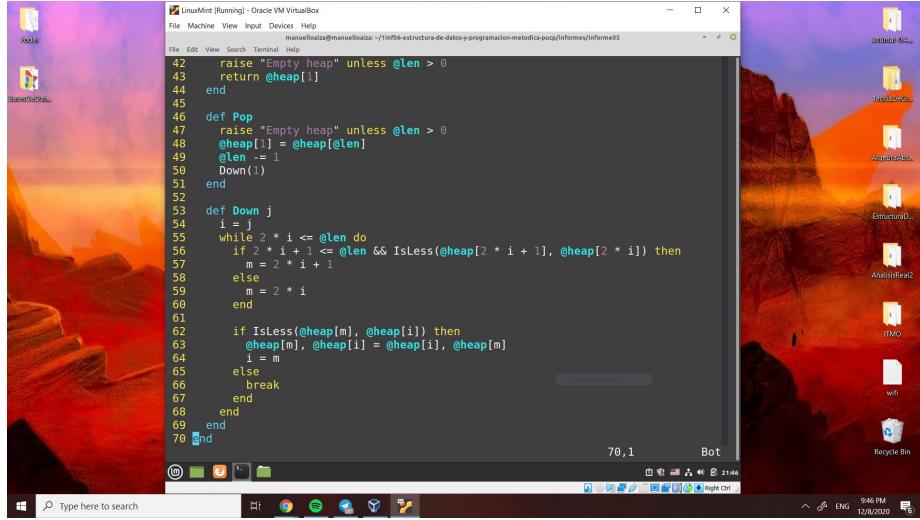


The screenshot shows a Linux Mint desktop environment. On the left, a terminal window titled "LinuxMint [Running] - Oracle VM VirtualBox" displays Python code for a heap data structure. The code includes methods for Up, IsLess, GetMin, and Pop. On the right, a file explorer window shows various files and folders on a desktop with a desert landscape background. The desktop icons include "Documentos", "Mantenimiento", "Recyclers", "Estructura1", "Analisis1", "ITMO", "Veff", and "Recycle bin". The taskbar at the bottom shows the system tray and some application icons.

```

24 @len += 1
25 @heap[@len] = path
26 Up()
27 end
28
29 def Up
30     i = @len
31     while i != 1 && IsLess(@heap[i], @heap[i / 2]) do
32         @heap[i], @heap[i / 2] = @heap[i / 2], @heap[i]
33         i = i / 2
34     end
35 end
36
37 def IsLess path1, path2
38     return path1.weight < path2.weight
39 end
40
41 def GetMin
42     raise "Empty heap" unless @len > 0
43     return @heap[1]
44 end
45
46 def Pop
47     raise "Empty heap" unless @len > 0
48     @heap[1] = @heap[@len]
49     @len -= 1
50     Down(1)
51 end
52

```



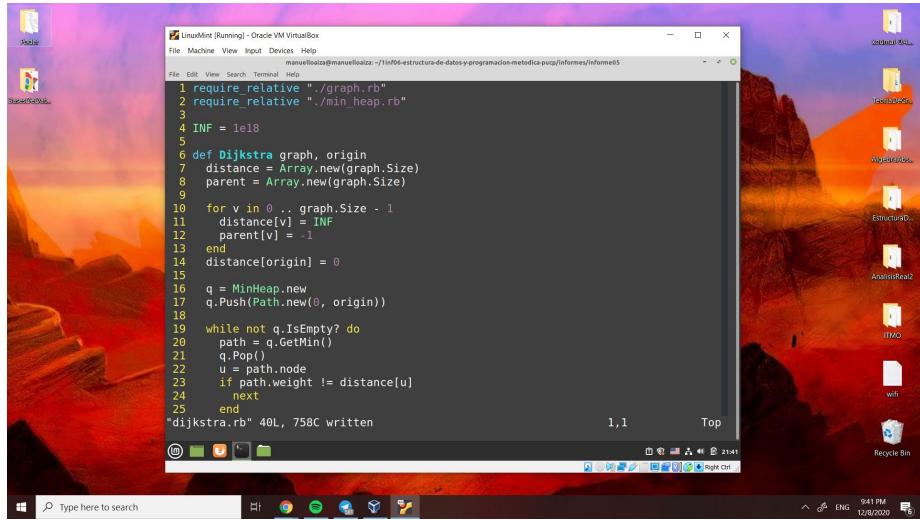
```

LinuxMint [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
manuelloalza@manuelloalza:~/finfd-estructura-de-datos-y-programacion-metodica-pusp/informes/informe05
File Edit View Search Terminal Help
42      raise "Empty heap" unless @len > 0
43      return @heap[1]
44    end
45
46    def Pop
47      raise "Empty heap" unless @len > 0
48      @heap[1] = @heap[@len]
49      @len -= 1
50      Down(1)
51    end
52
53    def Down j
54      i = j
55      while 2 * i <= @len do
56        if 2 * i + 1 <= @len && IsLess(@heap[2 * i + 1], @heap[2 * i]) then
57          m = 2 * i + 1
58        else
59          m = 2 * i
60        end
61
62        if IsLess(@heap[m], @heap[i]) then
63          @heap[m], @heap[i] = @heap[i], @heap[m]
64          i = m
65        else
66          break
67        end
68      end
69    end
70  end

```

4.3.3 Dijkstra's Algorithm

The time complexity of this implementation is $O((V + E) \log_2 E)$ because the algorithm goes through all nodes of the graph and adds for each edge at most one distance to the min-heap. The efficiency of Dijkstra's algorithm is based on the fact that the graph does not have negative edges. However, if the graph has a negative edge, the algorithm may give incorrect results.



```

LinuxMint [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
manuelloalza@manuelloalza:~/finfd-estructura-de-datos-y-programacion-metodica-pusp/informes/informe05
File Edit View Search Terminal Help
1 require_relative "./graph.rb"
2 require_relative "./min_heap.rb"
3
4 INF = 1e18
5
6 def Dijkstra graph, origin
7   distance = Array.new(graph.Size)
8   parent = Array.new(graph.Size)
9
10  for v in 0 .. graph.Size - 1
11    distance[v] = INF
12    parent[v] = -1
13  end
14  distance[origin] = 0
15
16  q = MinHeap.new
17  q.Push(Path.new(0, origin))
18
19  while not q.IsEmpty? do
20    path = q.getMin()
21    q.Pop()
22    u = path.node
23    if path.weight != distance[u]
24      next
25    end

```

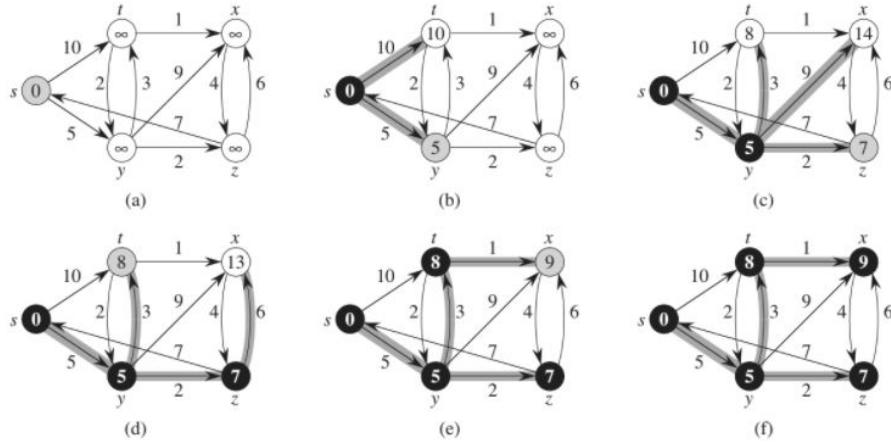
```

LinuxMint [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
manuellopez@manuellopez:~/finf06-estructura-de-datos-y-programacion-metodica-poo/informes/informe05
16 q = MinHeap.new
17 q.Push(Path.new(0, origin))
18
19 while not q.IsEmpty? do
20   path = q.GetMin()
21   q.Pop()
22   u = path.node
23   if path.weight != distance[u]
24     next
25   end
26
27   neighbors = graph.GetNeighbors(u)
28   neighbors.each do |edge|
29     edge.to
30     w = edge.weight
31     if distance[u] + w < distance[v]
32       distance[v] = distance[u] + w
33       parent[v] = u
34       q.Push(Path.new(distance[v], v))
35     end
36   end
37 end
38
39 return distance, parent
40 end

```

4.3.4 Testing

We are going to test our implementation with the following graph printing the weight and the path of the shortest path from s to each node. The nodes 0, 1, 2, 3, 4 represent s, t, x, y, z , respectively.



Linux Mint [Running] - Oracle VM VirtualBox

```
File Machine View Input Devices Help
manuelrealiza@manuelrealiza:~/fin06-estructura-de-datos-y-programacion-metodica-pusp/informes/informe05

1 require_relative "./dijkstra.rb"
2 require_relative "./graph.rb"
3 require_relative "./min_heap.rb"
4
5 def GetPath u, parent
6   path = []
7   while u != -1
8     path.push(u)
9     u = parent[u]
10 end
11 len = path.length
12 for i in 0 .. len / 2 - 1
13   path[i], path[len - 1 - i] = path[len - 1 - i], path[i]
14 end
15 return path
16 end
17
18 def PrintPath path
19   len = path.length
20   for i in 0 .. len - 1
21     if i == 0
22       print " "
23     end
24     print "#{path[i]}"
25   end
26   puts ""
27 end
28
29 def main
30   puts "Report 5 - Dijkstra's algorithm"
31   puts ""
32
33 graph = Graph.new(5)
34 graph.AddEdge(0, 1, 10)
35 graph.AddEdge(0, 3, 5)
36 graph.AddEdge(1, 3, 2)
37 graph.AddEdge(1, 2, 1)
38 graph.AddEdge(2, 4, 4)
39 graph.AddEdge(1, 3, 3)
40 graph.AddEdge(3, 2, 9)
41 graph.AddEdge(3, 4, 1)
42 graph.AddEdge(4, 0, 7)
43 graph.AddEdge(4, 2, 6)
44
45 distance, parent = Dijkstra(graph, 0)
46
47 for i in 0 .. graph.Size - 1
48   puts "Shortest path from 0 to #{i}: #{distance[i]}"
49   print "Path: "
50   PrintPath(GetPath(i, parent))
51   puts ""
52 end
53
54 main
```

1,1 Top

55,1 Bot

Type here to search

9:44 PM 12/8/2020

Linux Mint [Running] - Oracle VM VirtualBox

```
File Machine View Input Devices Help
manuelrealiza@manuelrealiza:~/fin06-estructura-de-datos-y-programacion-metodica-pusp/informes/informe05

27 end
28
29 def main
30   puts "Report 5 - Dijkstra's algorithm"
31   puts ""
32
33 graph = Graph.new(5)
34 graph.AddEdge(0, 1, 10)
35 graph.AddEdge(0, 3, 5)
36 graph.AddEdge(1, 3, 2)
37 graph.AddEdge(1, 2, 1)
38 graph.AddEdge(2, 4, 4)
39 graph.AddEdge(1, 3, 3)
40 graph.AddEdge(3, 2, 9)
41 graph.AddEdge(3, 4, 1)
42 graph.AddEdge(4, 0, 7)
43 graph.AddEdge(4, 2, 6)
44
45 distance, parent = Dijkstra(graph, 0)
46
47 for i in 0 .. graph.Size - 1
48   puts "Shortest path from 0 to #{i}: #{distance[i]}"
49   print "Path: "
50   PrintPath(GetPath(i, parent))
51   puts ""
52 end
53
54 main
```

55,1 Bot

Type here to search

9:45 PM 12/8/2020

