OBSERVACIONES DEL LA PRACTICA

Lindsay Vanessa Pinto Morato Cod. 202023138
José Daniel Montero Cod. 202012732

CONTENIDO

LISTA DE TABLAS

LISTA DE ILUSTRACIONES

EQUIPO DE PRUEBAS

Table 1. Especificaciones computadores

	Máquina 1
Marca	Computador Huawei Matebook D14
Procesadores	AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10GHZ
Memoria RAM (GB)	8,00GB (6.94 utilizable)
Sistema Operativo	Windows 10 Home Single languaje
Tipo de sistema	Sistema operativo de 64bits, procesador x64

	Máquina 2
Marca	Personalizado
Procesadores	AMD Ryzen 5 2600X Hexa-core @ 3.6 GHz
Memoria RAM (GB)	8.00 GB @ 2133 MHz
Sistema Operativo	Windows 10 Pro
Tipo de sistema	Sistema operativo de 64 bits, procesador x64

CONTROLES EN LAS PRUEBAS Y RESULTADOS

Las pruebas se realizaron utilizando los mismos parámetros en cada uno de los equipos de esta forma:

- Se cerraron todas las aplicaciones a excepción de Visual Studio Code
- Se realizaron las pruebas con el cargador conectado
- **Para el requerimiento 1** se utilizaron las entradas de landing point 1: Redondo Beach, CA, United States y landing point 2: Vung Tau, Vietnam. Se aclara que se utilizó el nombre completo pues algunos landing points tienen el mismo nombre pero su ubicación es en diferente país.
- Para el **requerimiento 3** se utilizó como país 1 Colombia y país 2 Indonesia.
- Para el **requerimiento 5** se utilizó el landing point: Fortaleza, Brazil

COMPLEJIDAD

Table 2. Complejidad temporal requerimientos

Requerimiento		**O(n)**
1	Encontrar cantidad de clusteres dentro de la red de cables submarinos	$O(V + E)$ -> Lineal
2	Encontrar landing points que sirven como punto de interconexión a más cables	O(V + E) -> Lineal
3	Encontrar la ruta mínima para enviar información entre dos países	$E \log(v)$ -> Linearítmica

4	Identificar la infraestructura crítica para garantizar el mantenimiento preventivo	O(n^2) -> Cuadrática
5	Conocer el impacto que tendría el fallo de un determinado landing point	$o(n^2)$ -> Cuadrática

A continuación se muestra el análisis detallado de estas complejidades

REQUERIMIENTO 1: Encontrar cantidad de clusteres dentro de la red de cables submarinos

def findClusters(analyzer, landingPoint1, landingPoint2):
  scc = kos.KosarajuSCC(analyzer['connections'])
  vertexAId = me.getValue(mp.get(analyzer['landingPointNames'], landingPoint1))
  vertexBId = me.getValue(mp.get(analyzer['landingPointNames'], landingPoint2))
  lpA = me.getValue(mp.get(analyzer['landingPoints'], vertexAId))
  lpB = me.getValue(mp.get(analyzer['landingPoints'], vertexBId))
  vertexA = lt.getElement(lpA['vertices'], 1)
  vertexB = lt.getElement(lpB['vertices'], 1)
  return kos.connectedComponents(scc), kos.stronglyConnected(scc, vertexA, vertexB)
	O (V+E) -> lineal
	O(1)
	O(1)
	O(1)
	O(1)
	O(1)
	O(1)

Llamaremos a O(1) = C y se realizará suma de cada una de las complejidades $$O(V+E)+C+C+C+C+C+C$$ Resolviendo: $$O(V+E)+6C$$ Por complejidad asintótica se tiene que la complejidad del algorítmo es de $O(V+E)$ La cual es lineal	

REQUERIMIENTO 2: Encontrar landing points que sirven como interconexión a más cables

def findInterLandingPoints(analyzer):
  total = 0
  iterator = lli.newIterator(gr.vertices(analyzer["arches"]))
  vList = lt.newList()
 	
  while lli.hasNext(iterator):
  vertex = lli.next(iterator)
  inDeg = gr.indegree(analyzer["arches"], vertex)
  outDeg = gr.outdegree(analyzer["arches"], vertex)
  if inDeg >= 1 and outDeg >1:
  total += 1
  lt.addLast(vList, vertex)
 	
  final = lt.newList()
  nIterator = lli.newIterator(vList)
 	
  while lli.hasNext(nIterator):
  elt = lli.next(nIterator)
  cpl = mp.get(analyzer["countriesCodes"], elt)
  value = me.getValue(cpl)
	
	O(1)
	O(1)
	
	O(1)
	O(1)
	O(V+E)
	O(V+E)
	O(1)
	
	O(1)
	
	O(1)
	O(1)
	
	O(1)
	O(1)
	O(1)
	O(1)

lt.addLast(final, value["id"]) lt.addLast(final, value["name"]) return total, final	O(1) O(1)

Llamaremos a O(1) = C y realizamos una suma de cada una de las complejidades:

$$C + C + C(C + O(V+E) + O(V+E) + C(C)) + C + C + C(C + C + C + C + C)$$

Resolviendo:

$$2C + C(C + 2(O(V+E)) + C^2) + 2C + C(5C)$$
$$4C + C^2 + C^3 + 2C(V+E) + 5C^2$$
$$4C + 6C^2 + C^3 + 2C(V+E)$$

Se devuelve C = 1:

$$4(1) + 6(1)^2 + (1)^3 + 2(V+E)$$
$$2(V+E)$$

Por complejidad asintótica, se tiene que la complejidad del algorítmo es de O(V+E), la cual es lineal.

REQUERIMIENTO 3: Encontrar la ruta mínima para enviar información entre dos países

def findShortestPath(analyzer, pais1, pais2):	
country1 = me.getValue(mp.get(analyzer['countries'], pais1))	O(1)
vertex1 = country1['vertex']	O(1)
country2 = me.getValue(mp.get(analyzer['countries'], pais2))	O(1)
vertex2 = country2['vertex']	O(1)
search = djk.Dijkstra(analyzer['connections'], vertex1)	E log (v)
dist = djk.distTo(search, vertex2)	O(1) -> De acuerdo al algoritmo en la libreria
path = djk.pathTo(search, vertex2)	O(1) -> De acuerdo al algoritmo en la libreria
return path, dist	

Llamaremos a O(1) = C y se realizará suma de cada una de las complejidades

$$C + C + C + C + E\log(v) + C + C$$

Resolviendo:

$$E\log(v) + 6C$$

Por complejidad asintótica se tiene que la complejidad del algorítmo es de $E\log(v)$ la cual es linearítmica.

REQUERIMIENTO 4: Identificar la infraestructura crítica para garantizar el mantenimiento preventivo

def criticalInfrastructure(analyzer):	
vertex = gr.numVertices(analyzer["connections"])	O(1)
tree = pr.PrimMST(analyzer["connections"])	O((V+E) log V) -> De acuerdo al algoritmo en la librería
weight = pr.weightMST(analyzer["connections"], tree)	O(1)
branch = pr.edgesMST(analyzer["connections"], tree)	O(1)
branch = branch["edgeTo"]["table"]["elements"]	O(1)
max = 0	
for i in range(len(branch)):	O(n)

value = branch[i]["value"] if (value != None) and (float(value["weight"]) > max): max = value["weight"] return vertex, weight, max	O(n) O(n) O(1)

Llamaremos a O(1) = C y se realizará la suma de cada una de las complejidades:

$$C + O((V+E) \log V) + C + C + C + n(n + n(C))$$

Resolviendo:

$$4C + O((V+E) \log V) + n(n + nC)$$
$$4C + O((V+E) \log V) + n^2 + n^2C$$

Se devuelve C = 1:

$$4(1) + O((V+E) \log V) + n^2 + n^2(1)$$
$$O((V+E) \log V) + 2n^2$$

Por complejidad asintótica, se tiene que la complejidad del algorítmo es de $O(n^2)$, la cual es cuadrática.

REQUERIMIENTO 5: Conocer el impacto que tendría el fallo de un determinado landing point

def failImpact(analyzer, landingPoint):	
vertexId = me.getValue(mp.get(analyzer['landingPointNames'], landingPoint))	O(1)
lp = me.getValue(mp.get(analyzer['landingPoints'], vertexId))	O(1)
countryName = lp['country']	O(1)
country = me.getValue(mp.get(analyzer['countries'], countryName))	O(1)
countries = mp.newMap(numelements=300, maptype='PROBING')	O(1)
for vertex in lt.iterator(lp['vertices']):	O(n)
adjVertices = gr.adjacents(analyzer['connections'], vertex)	O(1)
for adjVertex in lt.iterator(adjVertices):	O(n)
if adjVertex == country['vertex']:	O(1)
if not mp.contains(countries, countryName):	O(1)
ctryEdge = gr.getEdge(analyzer['connections'], vertex, country['vertex'])	O(1)
mp.put (countries, countryName, ctryEdge['weight'])	O(1)
else:	
adjVertexId = adjVertex.split('-')[0]	O(1)
adjLP = me.getValue(mp.get(analyzer['landingPoints'], adjVertexId))	O(1)
if not mp.contains(countries, adjLP['country']):	O(1)
adjEdge = gr.getEdge(analyzer['connections'], vertex, adjVertex)	O(1)
mp.put (countries, adjLP['country'], adjEdge['weight'])	O(1)
countriesList = mp.keySet(countries)	O(1)
result = om.newMap('BST')	O(log(n))
for ctry in lt.iterator(countriesList):	O(n)
distance = me.getValue(mp.get(countries, ctry))	O(1)
if om.contains(result, distance):	O(1)
ctryList = me.getValue(om.get(result, distance))	O(1)
lt.addLast(ctryList, ctry)	O(1)
else:	
newList = lt.newList('SINGLE_LINKED', compareCountries)	O(1)

lt.addLast(newList, ctry) om.put(result, distance, newList) return mp.size(countries), result	O(1) O(1)

Llamaremos a O(1) = C y se realizará suma de cada una de las complejidades

$$C + C + C + C + C + n\left(C + n\left(C\left(C\left(C + C\right)\right)\right)\right) + C + O\left(\log(n)\right) + n\left(C + C\left(C + C + C + C + C\right)\right)$$

Resolviendo:

$$5C + n\left(C + n\left(C\left(C\left(C + C\right)\right)\right)\right) + C + O\left(\log(n)\right) + n\left(C + C\left(5C\right)\right)$$

$$5C + n\left(C + n\left(C\left(C^2 + C^2\right)\right)\right) + C + O\left(\log(n)\right) + n\left(C + 5C^2\right)$$

$$5C + n\left(C + n\left(C^3 + C^3\right)\right) + C + O\left(\log(n)\right) + nC + 5\,C^2 n$$

$$5C + n\left(C + nC^3 + nC^3\right) + C + O\left(\log(n)\right) + nC + 5\,C^2 n$$

$$5C + nC + n^2C^2 + n^2C^2 + C + O\left(\log(n)\right) + nC + 5\,C^2 n$$

Devolviendo C = 1

$$5(1) + n(1) + n^2(1)^2 + n^2(1)^2 + (1) + O\left(\log(n)\right) + n(1) + 5\,(1)^2 n$$

$$n + n^2 + n^2 + O\left(\log(n)\right) + n + n$$

$$3n + 2n^2 + O\left(\log(n)\right)$$

Por complejidad asintótica se tiene que la complejidad del algorítmo es de $o(n^2)$La cual es cuadratica.

ANALISIS DE CONSUMO DE MEMORIA Y TIEMPO

MAQUINA 1

Resultados

Table 3. Comparación de consumo de datos y tiempo de ejecución para cada uno de los requerimientos en la Maquina 1.

REQUERIMIENTO	CONSUMO DE DATOS [KB]	TIEMPO DE EJECUCIÓN [MS]
Carga	1046,019	510,608
1	117,632	4042,227
2	397,475	106,489
3	21,044	1221,357
4	18,773	1528,92
5	22,134	6,348

MAQUINA 2

Resultados

Table 4. Comparación de consumo de datos y tiempo de ejecución para cada uno de los requerimientos en la Maquina 2.

REQUERIMIENTO	CONSUMO DE DATOS [KB]	TIEMPO DE EJECUCIÓN [MS]

Carga	4626,8756	865,4718
1	121,2606	3682,378
2	587,539	69,505
3	27,0874	1114,013
4	19,6922	1390,254
5	22,9188	2,536

GRÁFICAS

COMPARACIÓN TIEMPOS DE RENDIEMIENTO

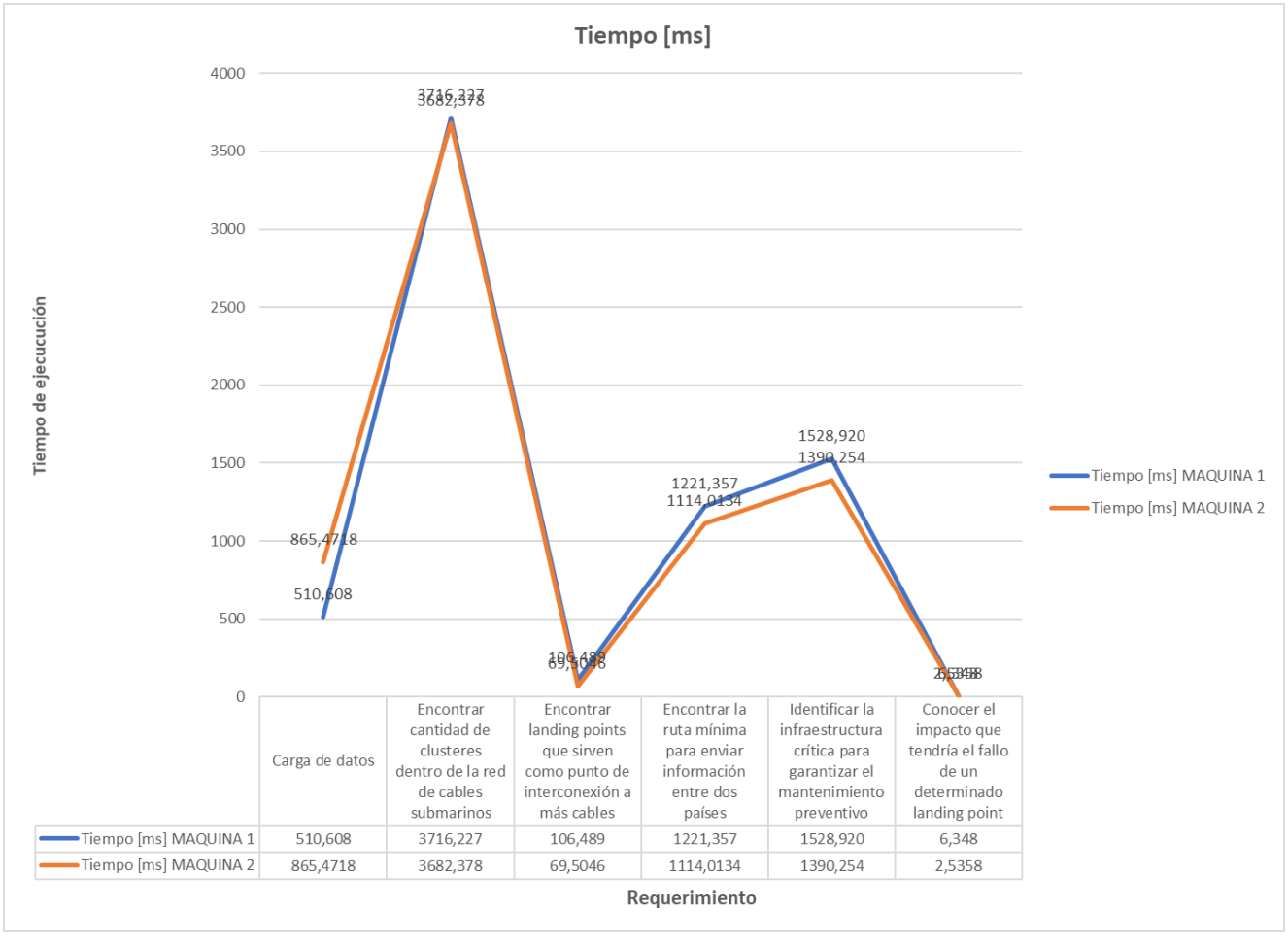

Ilustración 1. Comparación tiempos de ejecución maquina 1 y 2

En la gráfica anterior se muestra la comparación del desempeño en tiempos de ejecución en las dos maquinas de prueba. Se puede evidenciar que aunque en la carga el tiempo fue menor para la

máquina 1, los demás requerimientos tuvieron comportamiento similar sin diferencias significativamente marcadas entre los dos equipos.El requerimiento que presentó un tiempo menor fue el número 5.

COMPARACIÓN CONSUMO DE MEMORIA

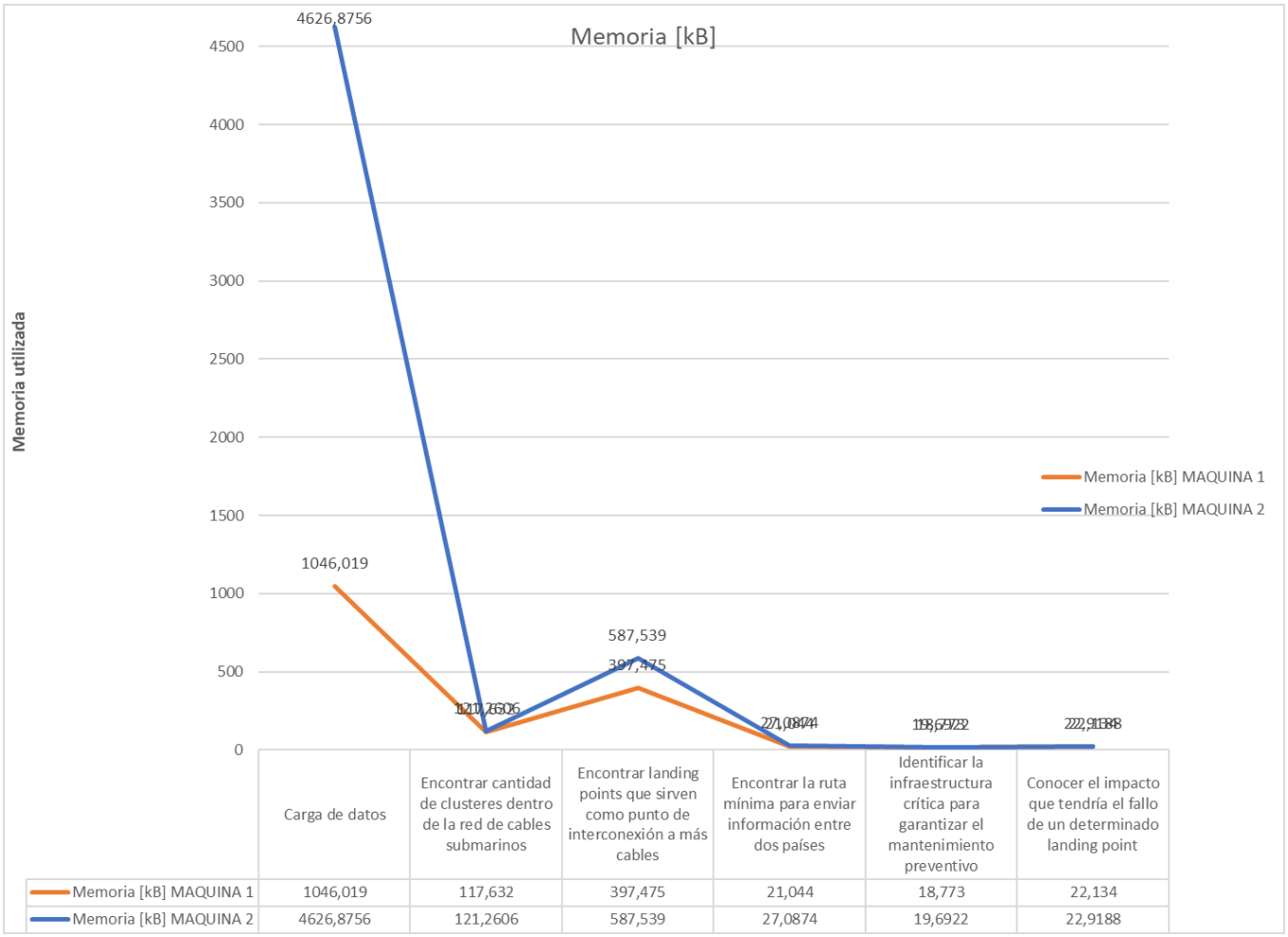

Ilustración 2. Comparación consumos de memoría para las maquinas 1 y 2

En esta gráfica se muestra la diferencia en el consumo de memoria para cada uno de los requerimientos. Se presenta una marcada diferencia en el consumo de memoria para la carga en la maquina 2, sin embargo, en los demás requerimientos el comportamiento es similar para las dos maquinas con leves diferencias en cada uno.

CONCLUSIONES

La utilización de algorítmos propios de los grafos pueden representar una marcada diferencia en la ejecución de requerimientos propios de los mismos al presentar complejidades lineales o linearítmicas en la mayoría de los casos.

Aunque algunos de los requerimientos presenten complejidad $O(n^2)$, el tiempo de ejecución puede verse disminuido por el tipo de recorrido presentado en el algorítmo. Este es el caso del requerimiento 5, por ejemplo, el cual pese a que su complejidad es cuadrática presenta tiempos de ejecución mínimos (comparados a los demás) dada la instancia del problema. Pues cuando encuentra los vértices adyacentes detienen la ejecución.