Instituto Tecnológico de Costa Rica

Unidad de Computación

“Primer Proyecto Programado Análisis de Algoritmos”

Alejandro Acuña Morera

Marco Antonio Espinoza Salas

Lester Alexander Trejos Bermúdez

Sede San Carlos

5/6/2020

**Introducción**

El propósito de este informe es presentar las experiencias obtenidas en la realización del proyecto, que consiste en el desarrollo de dos estructuras de datos conocidas como los son los árboles binarios y los grafos y de dos métodos (para cada uno) de diferentes formas de recorrer estas estructuras. Este proyecto se realizó utilizando herramientas informáticas, principalmente Java.

Ejemplificando el desarrollo del proyecto, para su creación se utilizó el lenguaje de programación Java, se comienza con la lectura de los requisitos del proyecto para su posterior desarrollo. Después de establecer los requisitos del proyecto, se comenzó con la codificación del programa, al final de la codificación se realiza una revisión y corrección de los errores presentes en el sistema, para que el proyecto sea fluido y no tenga complicaciones para la hora de usarlo; cabe resaltar que el usuario final no ingresa datos.

Un proyecto como este es muy necesario para la gente de hoy, ya que aprender a administrar sus viajes y sus vuelos a diferentes lugares les permitirá optimizar el tiempo y administrar mejor sus actividades.

El objetivo principal del proyecto es aplicar todas las habilidades aprendidas a lo largo del curso, así como el conocimiento obtenido a través de la investigación sobre varias características que tiene Java, que fueron útiles para la realización del proyecto anteriormente mencionado.

El análisis de un algoritmo es importante, ya que, permite demostrar cual algoritmo es mejor para ciertas situaciones específicas y cual puede tener un mejor desempeño y desenvolvimiento a la hora de resolver un problema planteado.

**Análisis del problema**

Requerimiento #1:

Como primer requerimiento, se solicita la creación de un árbol binario de forma automática en el cual el usuario final no debe ingresar datos. Los datos que se deben introducir en el árbol binario deben tener un tamaño de: 1000, 5000, 10.000, 100.000, 200.000 y 500.000 unidades.

Requerimiento #2:

Como segundo requerimiento, se solicita la creación de un grafo fuertemente conexo de forma automática en el cual como en el árbol binario, el usuario final no debe ingresa datos. Los datos que se deben introducir en el árbol binario deben tener un tamaño de: 1000, 5000, 10.000, 100.000, 200.000 y 500.000 vértices.

Requerimiento #3:

Como tercer requerimiento, se solicita implementar solo un recorrido en profundidad para árbol binario de manera recursiva. Estos son comúnmente llamados métodos de recorrido en Preorden, Inorden y Postorden.

Requerimiento #4:

Como cuarto requerimiento, se solicita implementar el recorrido en anchura para árbol binario de manera iterativa. Este recorrido trata de imprimir los nodos de un nivel, empezando desde el nivel 0. Por ejemplo: si tenemos un árbol binario de 4 niveles, primero, imprimiría los nodos del nivel 0, luego los nodos del nivel 1 y así, sucesivamente con los demás niveles hasta llegar al último nivel del árbol.

Requerimiento #5:

Como quinto requerimiento, se solicita implementar el recorrido en profundidad para el grafo de manera recursiva. Este algoritmo se posiciona sobre el primer vértice del grafo y, avanza al siguiente por medio de una “marca” que se le da al vértice, si la marca ya ha sido visitada, el algoritmo no recorrer ese vértice, por eso es primordial que, a la hora de insertar los vértices en el grafo, estos lleven su “marca” con un valor de tipo booleano igual a “false”.

Requerimiento#6:

Como sexto requerimiento, se solicita implementar el recorrido en amplitud para el grafo de manera iterativa. Este algoritmo se posiciona sobre el primer vértice del grafo, luego avanza hacia algún arco de este vértice y posteriormente, continúa avanzado entre los vértices, ejecutando lo antes mencionado al principio de la explicación de este algoritmo.

Requerimiento #7:

Como séptimo requerimiento, se solicita realizarles la medición empírica a todos los algoritmos de recorrido, en esta parte se omitirá la medición a los métodos de inserción solicitados en los requerimientos #1 y #2, anteriormente mencionados. Esta medición se debe efectuar, utilizando los tamaños establecidos por medio del documento del proyecto.

Requerimiento #8:

Como octavo requerimiento, se solicita determinarles el factor de crecimiento a todos los algoritmos de recorrido, en esta parte se omitirá determinarle el factor de crecimiento a los métodos de inserción solicitados en los requerimientos #1 y #2, anteriormente mencionados.

Requerimiento #9:

Como noveno requerimiento, se solicita realizarles la medición analítica a todos los algoritmos de recorrido, en esta parte se omitirá la medición a los métodos de inserción solicitados en los requerimientos #1 y #2, anteriormente mencionados.

Requerimiento #10:

Como décimo requerimiento, se solicita realizar la clasificación en notación O grande según las asignaciones, comparaciones, tiempo de ejecución y líneas ejecutadas a cada uno de los algoritmos de recorrido.

**Metodología**

En este apartado se van a abarcar los puntos que van a dar solución a los objetivos planteados en el proyecto. Se elaborarán algoritmos recursivos e iterativos para solventar el problema planteado de grafos fuertemente conexos y arboles binarios con sus respectivos recorridos. Posterior a ello, se analizarán los algoritmos mediante mediciones empíricas y analíticas para recopilar información de los recorridos de los algoritmos, en los cuales para el árbol binario se utilizará una inserción de 1000, 5000, 10000, 100000, 200000 y 500000 nodos. Para el grafo se utilizarán 1000, 1500, 2500, 3500, 4500 y 5000 vértices, esta cantidad es menor debido a que si es muy grande el computador tiende a quedarse trabado. Cuando los datos de entrada sean recopilados de los algoritmos de recorrido, serán evaluados respecto a su desempeño y mostrados en graficas para comparar sus comportamientos.

**Solución del Problema**

**Solución al requerimiento #1:**

Para la creación del árbol binario de forma automática se implementó una clase en el proyecto la cuál posee las características que necesita el árbol (izq, der, num), además de su constructor e instancia para ser utilizada en todo el proyecto. También la generación automática se realizó por medio de una función que va insertando nodo a nodo, con un valor random de 1 entre 1000000 de datos posibles con verificación de valores repetidos. Este último método es utilizado con un “for” en la clase “main” correspondiente a la cantidad de nodos que se quiere insertar.

**Solución al requerimiento #2:**

Para la creación de un grafo fuertemente conexo que se llene automáticamente y conecte todos los vértices entre sí, se implementaron dos clases, una con las características del vértice y otra con la conexión del arco. La automatización se logra con un “for” anidado que, el primero sitúa el vértice en el inicio y el segundo conecta ese vértice con el resto de vértice por medio de los arcos, generando el peso del arco automáticamente y los vértices generándose de “1” al tamaño definido máximo. Este método de insertar vértices y crear el grafo fuertemente conexo se utiliza en el “main” recibiendo la cantidad de vértices que se quieran insertar.

**Solución al requerimiento #3:**

Para desarrollar el método de recorrido por profundidad en un árbol binario, primero se debió establecer cuál era la condición de parada, ya que, este método utiliza la recursividad como medio para recorrer por todos los nodos del árbol. Esta condición de parada es preguntar si el valor que estamos ingresando que es de tipo “Árbol” está vació o tiene un valor nulo, luego, sino se cumple esta condición, se imprime el contenido del nodo, y después, se deben preguntar por los valores tanto de la izquierda como de la derecha (recursividad). Se utilizó un método muy conocido para la solución de este requisito, que es el método en “Preorden” en árboles binarios (Soriano Grande, 2019).

**Solución al requerimiento #4:**

Para desarrollar el método de recorrido por anchura en un árbol binario se tuvo que hacer una investigación en la red, en búsqueda de un método que pudiera cumplir con los requisitos del algoritmo. Se descubrió un método iterativo que utiliza la “colas” para resolver este ejercicio. Consiste en guardar en cola el primer valor del árbol binario (primer nodo), luego compara si la cola no está vacía como condición de parada, si esta no lo está, entonces saca el valor de la cola y posteriormente imprime el valor del nodo (Dave, 2020). Después comprueba si la izquierda está vacía, sino lo está, agrega el valor a la cola, a continuación, ejecuta lo mismo por el valor derecho, sino está vacío entonces, lo agrega a la cola. ("Level Order Tree Traversal - GeeksforGeeks", s.f.) Después, de realizar todos esos procesos, se regresa a la condición de parada, preguntando si la cola no está vacía, este método termina cuando la cola se queda sin valores, lo cuál representa que ya se recorrió el árbol binario por completo.

**Solución al requerimiento #5:**

Para desarrollar el método de recorrido por profundidad en un grafo fuertemente conexo, primero se tomó en cuenta que este método precisamente (el de profundidad) ya se había planteado y resuelto en el curso de “Estructuras de Datos”, entonces por medio de este método previamente creado, se logró modificarlo y hacerle los cambios pertinentes para que funcionara con el grafo del proyecto.

**Solución al requerimiento #6:**

Para desarrollar el método de recorrido por amplitud en un grafo fuertemente conexo, al igual que en la solución del **requerimiento #5,** este método ya se había planteado y resuelto en el curso de “Estructuras de Datos”, entonces por medio de este algoritmo previamente creado, se logró modificarlo y hacerle los cambios pertinentes para que funcionara con el grafo del proyecto.

**Solución al requerimiento #7:**

La medición empírica de todos los algoritmos se realizó por medio de la recolección de datos que brindaba cada algoritmo en específico con su respectivo valor de entrada (cantidad de datos), los datos se anotaron en una tabla que engloba la separación de las operaciones (Asignaciones, Comparaciones…).

**Solución al requerimiento #8:**

El factor de crecimiento se realizó mediante la división de los valores de entrada de cada algoritmo y la división de las operaciones generados en asignaciones, comparaciones… respectivamente.

**Solución al requerimiento #9:**

Para la medición analítica se realizó el conteo de pasos línea por línea, de cada algoritmo, después se efectuó la suma que otorga dicha medición para generar un total, además con ello se clasifica en notación O Grande respectivamente cada algoritmo. Se tomó en cuenta lo que explica Muñiz, en el año 2015 para la realización de la medición analítica.

**Solución al requerimiento #10:**

Para la clasificación de las operaciones (asignaciones, comparaciones, tiempo, y líneas de ejecución) en notación O grande se tomó en cuenta las divisiones que generan el factor de talla, factor de asignaciones, factor de comparaciones, de tiempo y de líneas de ejecución respectivamente de los datos generados por cada algoritmo.

**Análisis y Resultados**

Nombre del algoritmo #1: recorrido en Profundidad de un árbol binario

| Operaciones | Cantidad de datos de entrada por defecto | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| 1000 | 5000 | 10000 | 100000 | 200000 | 500000 |
| Asignaciones | 2000 | 10000 | 20000 | 199994 | 399988 | 999970 |
| Comparaciones | 2001 | 10001 | 20001 | 199995 | 399989 | 999971 |
| Cantidad de líneas ejecutadas | 6002 | 30002 | 60002 | 599984 | 1199966 | 2999912 |
| Tiempo de ejecución | 0.052 | 0.224 | 0.507 | 4.971 | 9.836 | 24.371 |
| Cantidad de líneas del código | 7 | | | | | |

Tabla 1= Medición empírica recorrido en Profundidad de un árbol binario.

Nombre del algoritmo #1: recorrido en Profundidad de un árbol binario

|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código del método de resolver el recorrido en profundidad del árbol binario. | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
| public void recorridoProfundidad(Arbol aux) {  if (aux == null) {  return;  }  System.out.println("Nodo: " + aux.num);  recorridoProfundidad(aux.izq);  recorridoProfundidad(aux.der);  } | 2n      n  n |
| Total (la suma de todos los pasos) | 4n |
| **Clasificación en notación O Grande** | O(n) |

Tabla #2: Medición analítica recorrido en Profundidad de un árbol binario.

Nombre del algoritmo #1: recorrido en Profundidad de un árbol binario

| Talla | | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas | Factor Tiempo de ejecución |
| --- | --- | --- | --- | --- | --- | --- |
| De 1000 a 5000 |  | 5 |  |  |  |  |
|  | 5 | 4.998 | 4.998 | 4.307 |
|  |  |  |  |  |
| De 5000 a 10000 |  | 2 |  |  |  |  |
|  | 2 | 1.999 | 1.999 | 2.263 |
|  |  |  |  |  |
| De 10000 a 100000 |  | 10 |  |  |  |  |
|  | 9.999 | 9.999 | 9.999 | 9.804 |
|  |  |  |  |  |
| De 100000 a 200000 |  | 2 |  |  |  |  |
|  | 2 | 1.999 | 1.999 | 1.978 |
|  |  |  |  |  |
| De 200000 a 500000 |  | 2.500 |  |  |  |  |
|  | 2.500 | 2.499 | 2.499 | 2.477 |
|  |  |  |  |  |
| De 1000 a 500000 |  | 500 |  |  |  |  |
|  | 499.985 | 499.735 | 499.818 | 468.673 |
|  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** | O(n) |  |
| Clasificación del comportamiento de las **comparaciones** | O(n) |
| Clasificación del comportamiento de las **líneas ejecutadas** | O(n) |
| Clasificación del comportamiento en el **tiempo de ejecución** | O(n) |

Tabla #3: Determinar factor de crecimiento del recorrido en Profundidad de un árbol binario.

Nombre del algoritmo #2: recorrido anchura de un árbol binario

| Operaciones | Cantidad de datos de entrada por defecto | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1000 | 5000 | 10000 | 100000 | 200000 | 500000 |
| Asignaciones | 1999 | 9999 | 19999 | 199995 | 399993 | 999981 |
| Comparaciones | 3001 | 15001 | 30001 | 299995 | 599992 | 1499974 |
| Cantidad de líneas ejecutadas | 5002 | 25002 | 50002 | 499992 | 999987 | 2499957 |
| Tiempo de ejecución | 0.048 | 0.365 | 0.926 | 9.171 | 18.669 | 46.798 |
| Cantidad de líneas del código | 15 | | | | | | |

Tabla 4= Medición empírica recorrido anchura de un árbol binario.

Nombre del algoritmo #2: recorrido anchura de un árbol binario

|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código del método de resolver el rompecabezas. | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
| public void recorridoAnchura(Arbol aux) {  Queue<Arbol> queue = new LinkedList<>();  queue.add(aux);  while (!queue.isEmpty())  {  Arbol tempNode = queue.poll();  System.out.println("Nodo: " + tempNode.num);  if (tempNode.izq != null) {  queue.add(tempNode.izq);  }  if (tempNode.der != null) {  queue.add(tempNode.der);  }  } } | 1  1  n+1  n  n  n  n  n |
| Total (la suma de todos los pasos) | 6n+3 |
| **Clasificación en notación O Grande** | O(n) |

Tabla #5: Medición analítica recorrido anchura de un árbol binario.

Nombre del algoritmo #2: recorrido anchura de un árbol binario

| Talla | | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas | Factor Tiempo de ejecución |
| --- | --- | --- | --- | --- | --- | --- |
| De 1000 a 5000 |  | 5 |  |  |  |  |
|  | 5.002 | 4.998 | 4.998 | 7.604 |
|  |  |  |  |  |
| De 5000 a 10000 |  | 2 |  |  |  |  |
|  | 2.000 | 1.999 | 1.999 | 2.53 |
|  |  |  |  |  |
| De 10000 a 100000 |  | 10 |  |  |  |  |
|  | 10.000 | 9.999 | 9.999 | 9.903 |
|  |  |  |  |  |
| De 100000 a 200000 |  | 2 |  |  |  |  |
|  | 2.000 | 2.000 | 2.000 | 2.035 |
|  |  |  |  |  |
| De 200000 a 500000 |  | 2.5 |  |  |  |  |
|  | 2.499 | 2.499 | 2.499 | 2.506 |
|  |  |  |  |  |
| De 1000 a 500000 |  | 500 |  |  |  |  |
|  | 500.240 | 499.824 | 499.791 | 974.958 |
|  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** | **O(n)** |  |
| Clasificación del comportamiento de las **comparaciones** | O(n) |
| Clasificación del comportamiento de las **líneas ejecutadas** | O(n) |
| Clasificación del comportamiento en el **tiempo de ejecución** | O(n) |

Tabla #6: Medición analítica recorrido anchura de un árbol binario.

Nombre del algoritmo #3: recorrido en Profundidad de un Grafo

| Operaciones | Cantidad de datos de entrada por defecto | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| 1000 | 1500 | 2500 | 3500 | 4500 | 5000 |
| Asignaciones | 2002000 | 4503000 | 12505000 | 24507000 | 40509000 | 50010000 |
| Comparaciones | 2000001 | 4500001 | 12500001 | 24500001 | 40500001 | 50000001 |
| Cantidad de líneas ejecutadas | 4003002 | 9004502 | 25007502 | 61257002 | 81013502 | 125010002 |
| Tiempo de ejecución | 0.091 | 0.157 | 0.308 | 1.63 | 3.144 | 3.321 |
| Cantidad de líneas del código | 14 | | | | | |

Tabla 7= Medición empírica recorrido en Profundidad de un árbol binario.

Nombre del algoritmo #3: recorrido en Profundidad de un Grafo

|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código del método de resolver el recorrido en profundidad del árbol binario. | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
| public void profundidadGrafo(Vertice grafo){  if ((grafo == null) | (grafo.visitado == true)) {  return;  } else {  grafo.visitado = true;  Arco aux = grafo.subListaArcos;    while (aux != null) {    profundidadGrafo(aux.destino);  aux = aux.sigA;  }  }  } | 1 1  1  1  n+1  nn  n |
| Total (la suma de todos los pasos) | 2n + nn + 5 |
| **Clasificación en notación O Grande** | O(nn) |

Tabla #8: Medición analítica recorrido en Profundidad de un Grafo.

Nombre del algoritmo #3: recorrido en Profundidad de un Grafo

| Talla | | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas | Factor Tiempo de ejecución |
| --- | --- | --- | --- | --- | --- | --- |
| De 1000 a 1500 |  | 1.5 |  |  |  |  |
|  | 2.24 | 2.24 | 2.24 | 1.72 |
|  |  |  |  |  |
| De 1500 a 2500 |  | 1.6 |  |  |  |  |
|  | 2.77 | 2.77 | 2.77 | 1.97 |
|  |  |  |  |  |
| De 2500 a 3500 |  | 1.4 |  |  |  |  |
|  | 1.95 | 1.95 | 2.44 | 5.29 |
|  |  |  |  |  |
| De 3500 a 4500 |  | 1.28 |  |  |  |  |
|  | 1.65 | 1.65 | 1.32 | 1.92 |
|  |  |  |  |  |
| De 4500 a 5000 |  | 1.11 |  |  |  |  |
|  | 1.23 | 1.23 | 1.54 | 1.04 |
|  |  |  |  |  |
| De 1000 a 5000 |  | 5 |  |  |  |  |
|  | 24.9 | 24.99 | 31.22 | 36.9 |
|  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** | O(n2) |  |
| Clasificación del comportamiento de las **comparaciones** | O(n2) |
| Clasificación del comportamiento de las **líneas ejecutadas** | O(n2) |
| Clasificación del comportamiento en el **tiempo de ejecución** | O(n2) |

Tabla #9: Determinar factor de crecimiento del recorrido en Profundidad de un Grafo.

Nombre del algoritmo #4: recorrido amplitud de un grafo

| Operaciones | Cantidad de datos de entrada por defecto | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| 1000 | 1500 | 2500 | 3500 | 4500 | 5000 |
| Asignaciones | 1002001 | 2253001 | 6255001 | 12257001 | 20259001 | 25010001 |
| Comparaciones | 1001001 | 2251501 | 6252501 | 12253501 | 20254501 | 25005001 |
| Cantidad de líneas ejecutadas | 1004003 | 2256003 | 6260003 | 12264003 | 20268003 | 25020003 |
| Tiempo de ejecución | 0.086 | 0.103 | 0.165 | 0.273 | 0.390 | 0.494 |
| Cantidad de líneas del código | 18 | | | | | |

Tabla 10= Medición empírica recorrido amplitud de un grafo

Nombre del algoritmo #4: recorrido amplitud de un grafo

|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código del método de resolver el rompecabezas. | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
| public void amplitudGrafo(Vertice grafo) {  if (grafo == null) {  System.out.println("No hay grafo");  } else {  Vertice temp = grafo;  while (temp != null)  {  System.out.println("Vertice: " + temp.num);  Arco aux = temp.subListaArcos;  while (aux != null)  {  aux = aux.sigA;  }  temp = temp.sigV;  }  } } | 1  1  n+1  n  n2+n  n2  n |
| Total (la suma de todos los pasos) | 2n2+4n+3 |
| **Clasificación en notación O Grande** | O(n2) |

Tabla #11: Medición analítica recorrido amplitud de un grafo.

Nombre del algoritmo #4: recorrido amplitud de un grafo

| Talla | | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas | Factor Tiempo de ejecución |
| --- | --- | --- | --- | --- | --- | --- |
| De 1000 a 1500 |  | 3/2 |  |  |  |  |
|  | 2.248 | 2.249 | 2.247 | 1.197 |
|  |  |  |  |  |
| De 1500 a 2500 |  | 5/3 |  |  |  |  |
|  | 2.776 | 2.771 | 2.774 | 0.624 |
|  |  |  |  |  |
| De 2500 a 3500 |  | 7/5 |  |  |  |  |
|  | 1.959 | 1.959 | 1.959 | 1.654 |
|  |  |  |  |  |
| De 3500 a 4500 |  | 9/7 |  |  |  |  |
|  | 1.652 | 1.652 | 1.652 | 1.428 |
|  |  |  |  |  |
| De 4500 a 5000 |  | 10/9 |  |  |  |  |
|  | 1.234 | 1.234 | 1.234 | 1.266 |
|  |  |  |  |  |
| De 1000 a 5000 |  | 5 |  |  |  |  |
|  | 24.960 | 24.979 | 24.920 | 5.744 |
|  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** | O(n2) |  |
| Clasificación del comportamiento de las **comparaciones** | O(n2) |
| Clasificación del comportamiento de las **líneas ejecutadas** | O(n2) |
| Clasificación del comportamiento en el **tiempo de ejecución** | O(n2) |

Tabla #12: Medición analítica recorrido amplitud de un grafo

**Bitácoras**

**Bitácora #1:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Integrantes** | **Fecha y Hora** | **Lugar o medio** | **Participantes** | **Asuntos por tratar** | **Acuerdos** | **Asuntos pendientes** |
| **Alejandro Acuña Morera** | **5/8/2020**  **A las 9:46 pm.** | **Digital vía Discord.** | **El grupo completo.** | **Creación del repositorio para el proyecto.** | **El proyecto se va a manejar por medio un controlador de versiones para evitar errores en la creación de los algoritmos.**  **División del trabajo.** | **Empezar con el desarrollo del proyecto, así como, la documentación externa de este.** |
| **Marco Antonio Espinoza Salas** |
| **Lester Alexander Trejos Bermúdez** |

**Bitácora#2:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Integrantes** | **Fecha y Hora** | **Lugar o medio** | **Participantes** | **Asuntos por tratar** | **Acuerdos** | **Asuntos pendientes** |
| **Alejandro Acuña Morera** | **5/27/2020**  **A las 11:01 pm.** | **Digital vía Discord.** | **Marco Espinoza.** | **Creación de los algoritmos de recorrido de árboles binarios, tanto por profundidad como de anchura.** | **Los algoritmos satisfacen con los requerimientos principales del proyecto.** | **Empezar con el desarrollo de los algoritmos de recorrido de grafos, tanto por amplitud como por profundidad.** |
| **Marco Antonio Espinoza Salas** |
| **Lester Alexander Trejos Bermúdez** |

**Bitácora#3:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Integrantes** | **Fecha y Hora** | **Lugar o medio** | **Participantes** | **Asuntos por tratar** | **Acuerdos** | **Asuntos pendientes** |
| **Alejandro Acuña Morera** | **5/29/2020**  **A las 1:19 pm.** | **Digital vía Discord.** | **Lester Trejos.** | **Creación de los algoritmos de recorrido de árboles binarios, tanto por profundidad como de anchura.** | **Los algoritmos satisfacen con los requerimientos principales del proyecto.** | **Empezar con el desarrollo de los algoritmos de recorrido de grafos, tanto por amplitud como por profundidad.** |
| **Marco Espinoza** |
| **Lester Alexander Trejos Bermúdez** |

**Bitácora#4:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Integrantes** | **Fecha y Hora** | **Lugar o medio** | **Participantes** | **Asuntos por tratar** | **Acuerdos** | **Asuntos pendientes** |
| **Alejandro Acuña Morera** | **5/29/2020**  **A las 1:19 pm.** | **Digital vía Discord.** | **Alejandro Acuña.** | **Creación de los algoritmos de recorrido de árboles binarios, tanto por profundidad como de anchura.** | **Los algoritmos satisfacen con los requerimientos principales del proyecto.** | **Empezar con el desarrollo de los algoritmos de recorrido de grafos, tanto por amplitud como por profundidad.** |
| **Marco Antonio Espinoza Salas** |
| **Lester Alexander Trejos Bermúdez** |

**Bitácora#5:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Integrantes** | **Fecha y Hora** | **Lugar o medio** | **Participantes** | **Asuntos por tratar** | **Acuerdos** | **Asuntos pendientes** |
| **Alejandro Acuña Morera** | **5/29/2020**  **A las 10:57 pm.** | **Digital vía Discord.** | **El grupo completo.** | **Terminar la parte programada del proyecto.** | **Todos los algoritmos, tanto los de inserción como los de recorridos, satisfacen las necesidades y requerimientos del proyecto.**  **Se empezó con la realización de la documentación interna.** | **Empezar con el desarrollo de la documentación externa del proyecto.**  **Terminar con la documentación interna.** |
| **Marco Antonio Espinoza Salas** |
| **Lester Alexander Trejos Bermúdez** |

**Bitácora#6:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Integrantes** | **Fecha y Hora** | **Lugar o medio** | **Participantes** | **Asuntos por tratar** | **Acuerdos** | **Asuntos pendientes** |
| **Alejandro Acuña Morera** | **5/30/2020**  **A las 3:30 pm.** | **Digital vía Discord.** | **El grupo completo.** | **Terminar con la toma de datos y con la documentación externa del proyecto.** | **División del trabajo en cuanto a las mediciones empíricas, analíticas y establecer el factor de crecimiento de los algoritmos.** | **Terminar con la documentación interna y externa.** |
| **Marco Antonio Espinoza Salas** |
| **Lester Alexander Trejos Bermúdez** |

**Recomendaciones:**

* Gestione su proyecto con un controlador de versión, ya que compartir el código es más fácil y también corrige errores.
* Dividir el trabajo en un proyecto informático es una buena práctica para su ejecución e implementación.
* El manejo de los tamaños del grafo en el proyecto es muy importante, ya que, al usar datos o tamaños muy elevados, el sistema puede tardar mucho tiempo, tiempo que los usuarios finales no disponen; se recomienda utilizar tamaños no mayores a 10.000 unidades.
* Coordinar adecuadamente con los integrantes del equipo de trabajo para la adecuada distribución de tareas, estableciendo medios de retroalimentación por parte de los integrantes, para la corrección inmediata de posibles anomalías en el desarrollo del proyecto.

**Conclusiones**

* Para cantidades o tamaños más elevados, se recomienda el uso de algoritmos iterativos, ya que su tiempo de ejecución es más rápidos que los algoritmos que utilicen métodos recursivos.
* Con base en los resultados obtenidos por medio del análisis de los algoritmos, se recomienda el uso del método de recorrido por anchura para un árbol binario cuando se están usando una cantidad elevada de datos, ya que su tiempo de ejecución es menor que la del recorrido por profundidad.
* Con base en los resultados obtenidos por medio del análisis de los algoritmos, se recomienda el uso del método de recorrido por amplitud para un grafo cuando se están usando una cantidad elevada de datos, ya que su tiempo de ejecución es menor que la del recorrido por profundidad.

**Referencias Bibliográficas**

Soriano Grande, D. (2019, febrero 13). Código de Java - Arbol Binario.

Recuperado de <https://www.lawebdelprogramador.com/codigo/Java/5111-Arbol-Binario.html>

Level Order Tree Traversal - GeeksforGeeks. Recuperado de <https://www.geeksforgeeks.org/level-order-tree-traversal/>

Dave, G. (2020). Queue poll() method in Java - GeeksforGeeks. Recuperado de <https://www.geeksforgeeks.org/queue-poll-method-in-java/>

Velasquez, F. (2014). Medir el tiempo de ejecución en Java ~ Blog Un Estudiante de Informática. Recuperado de <http://unestudiantedeinformatica.blogspot.com/2014/07/medir-el-tiempo-de-ejecucion-en-java.html>

Muñiz, F. (2015). Material de apoyo Unidad 1 Estructura de datos - ppt descargar. Recuperado de <https://slideplayer.es/slide/3390457/>