Instituto Politécnico Nacional
Escuela Superior de Cómputo

Análisis de Algoritmos

Semestre: 2022-1

Grupo: 3CV11

Práctica 7

Fecha: 17 de diciembre de 2021

Práctica 7: Verificación en tiempo polinomial.

Brayan Ramirez Benitez.
*bramirezb*1901@*alumno.ipn.mx*

Resumen: Para la teoría de grafos, existe una gran cantidad de algoritmos que resuelven problemas bastante interesantes como los algoritmos de Dijkstra, Bellman - Ford, Christofides, Floyd – Warshall, Johnson, Prim, Vecino más próximo, Kruskal, entre otros. Dentro de estos problemas existe uno en particular que tiene como objetivo determinar si un certificado es un ciclo hamiltoniano para un determinado grafo, este problema forma parte del conjunto de los problemas NP-completos. En esta práctica realizamos el análisis a priori y posteriori para un algoritmo propuesto que resuelve el problema de verificar en tiempo polinomial que un certificado es o no un ciclo hamiltoniano de un grafo. Para esto el algoritmo se implementó mediante el uso del lenguaje de programación en C, luego de la implementación se realizaron pruebas con distintos grafos y certificados, lo anterior con el objetivo de verificar el correcto funcionamiento del algoritmo. Posteriormente se llevó a cabo el análisis a priori para el algoritmo propuesto mediante segmentos de código, obteniendo como resultado una complejidad temporal cuadrática, además, se realizó el análisis a posteriori haciendo uso de una calculadora gráfica del sitio web Desmos, en donde fue posible corroborar satisfactoriamente los resultados obtenidos en el análisis a priori.

Palabras Clave: Hamiltoniano, Grafo, Polinomial, Certificado, Verificación.

1 Introducción

De manera general podemos definir que la teoría de grafos, es la ciencia matemática que se encarga de estudiar los grafos, estos últimos, grosso modo, son estructuras matemáticas que modelan relaciones entre objetos. Existen diferentes tipos de grafos y una enorme variedad de problemas para estos. Por ejemplo, en el año de 1959 aparece uno de los algoritmos más conocidos del problema de SSSP (Single Source Shortest Path) que no utiliza una cola de prioridad, es decir, el algoritmo de Dijkstra, este algoritmo tiene una complejidad temporal $O(n^2)$ y es bastante simple, por esta razón es muy utilizado en problemas prácticos, sin embargo, también existen problemas para los que es difícil determinar si tienen solución o si existe una mejor solución que las conocidas, uno de ellos el problema del viajero, en el cual se necesita saber si existe una ruta optima que pasa por absolutamente todos los nodos de un determinado grafo sin repetir los nodos, a esta clase de problemas se les conoce como NP-completos.

Un problema NP-completo, es un problema computacional para el cual todavía no se encuentra un algoritmo de solución eficiente. Existe una gran cantidad de problemas que pertenecen a esta clase, algunos de ellos son el problema del viajante y los problemas relacionados con grafos.

Normalmente los problemas pueden resolverse haciendo uso de algoritmos que poseen un tiempo de ejecución polinomial, esto quiere decir que, si tenemos una entrada de tamaño n, la cantidad de pasos necesarios para encontrar una de las soluciones es una función polinomial para la entrada n. Por el contrario, existen algoritmos para encontrar una solución a problemas intratables, sin embargo, requieren tiempos de ejecución que se representan como funciones exponenciales de tamaño n, por esto, se consideran ineficientes puesto que los tiempos de ejecución aumentan más rápido a medida que aumenta el tamaño del problema.

Todos los problemas NP tienen un algoritmo determinista en tiempo polinomial, estos algoritmos verifican posibles soluciones al problema. Por lo tanto, el desafío de los problemas de NP es encontrar la respuesta de manera eficiente, lo cual representa una forma efectiva de verificar la solución ya encontrada. Una de las preguntas abiertas más importantes en informática es: ¿Si estos problemas realmente no se pueden resolver en tiempo polinomial? Puesto que, si encontráramos un algoritmo de tiempo polinomial para al menos uno de ellos, existiría un algoritmo polinomial para todos los problemas en NP. Para los problemas NP-completos su estado se desconoce. Todavía no se ha descubierto ningún algoritmo de tiempo polinomial para ningún problema de NP completo, además, nadie ha podido demostrar que no exista ningún algoritmo de tiempo polinomial para ninguno de ellos. Lo interesante es que, si cualquiera de los problemas completos de NP se puede resolver en tiempo polinomial, entonces se pueden resolver todos.

También, podemos afirmar que un problema pertenece a la clase NP si en todas las instancias, la respuesta puede ser certificada en tiempo polinomial. Por ejemplo, aun no se conocen algoritmos de complejidad polinomial que ofrezcan una solución para el problema de reconocimiento de grafos hamiltonianos. Pero, sabemos que todo grafo que es hamiltoniano tiene como certificado de su condición un orden de recorrido de los nodos. Para verificar la validez de ese certificado, basta con probar que el grafo contiene un camino que recorre los vértices en este orden, pero esta verificación debe hacerse en tiempo polinomial. Por lo tanto, este problema se encuentra en la clase NP, pero aún no se sabe si está en P.

El problema planteado para esta práctica tiene como objetivo verificar en un tiempo polinomial si un certificado es un ciclo hamiltoniano para un determinado grafo. Durante está práctica se presenta algoritmo propuesto y todo lo necesario para determinar la complejidad temporal mediante un análisis a priori y posteriori, además, las pruebas realizadas para corroborar el corrécto funcionamiento del algoritmo propuesto. Finalmente, se incluye la implementación del algoritmo en el lenguaje de programación en C, las conclusiones generales e individuales y las referencias utilizadas para la elaboración de la práctica.

2 Conceptos Básicos

En esta sección mostraremos todos los conceptos necesarios para el desarrollo de la práctica.

Definition 2.1. *La complejidad temporal o eficiencia en tiempo es el número de operaciones para resolver un problema con una entrada de tamaño* $\boldsymbol{n}$*. Se denota por* $\boldsymbol{T(n)}$*.*

Definition 2.2. *Dada una función* $\boldsymbol{g(n)}$*.* $O(\boldsymbol{g(n)})$ *denota el conjunto de funciones definidas como:*
$O(\boldsymbol{g(n)}) = \{\ \boldsymbol{f(n)}$*:* $\exists n\ C > 0$ *y* $n_0 > 0$ *constantes Tal que* $0 \leq \boldsymbol{f(n)} \leq C\ \boldsymbol{g(n)}\ \forall\ n \geq n_0\ \}$

Definition 2.3. *El análisis a priori consiste en determinar una expresión que nos indique el comportamiento para un algoritmo particular en función de los parámetros que contiene, es decir, determina una función que limita el tiempo de cálculo para un algoritmo en específico, por lo tanto, es un factor fundamental para el diseño de algoritmos.*

Definition 2.4. *El análisis a posteriori, es una expresión que proviene del latín, la cual significa posteriormente, es posible traducirla como "de lo posterior". De esta manera, se refiere a examinar un evento después de que ocurrió.*

Definition 2.5. *Los nodos son elementos cuyas relaciones se expresan mediante aristas.*

Definition 2.6. *Las aristas, son componentes las cuales se usan para representar relaciones entre varios nodos de un grafo.*

Definition 2.7. *Las aristas adyacentes, son aristas que comparten un vértice común.*

Definition 2.8. *Un camino, es una serie de vértices que tienen una propiedad en la cual cada vértice de la serie es adyacente al vértice contiguo. Una ruta que no repite vértices se llama ruta simple.*

Definition 2.9. *Un grafo, es una estructura de datos no lineal que consiste en nodos y aristas. Los nodos también se denominan vértices y las aristas son líneas que conectan dos nodos cualesquiera.*

Definition 2.10. *Un ciclo es una arista que une un nodo a sí mismo.*

Definition 2.11. *Un ciclo hamiltoniano, es un ciclo cerrado en un grafo donde cada nodo es visitado única y exclusivamente una sola vez. La única manera de determinar si un grafo tiene un ciclo hamiltoniano, es realizando una búsqueda exhaustiva, pasando por todas las opciones.*

2.1 Algoritmo propuesto

A continuación se muestra el algoritmo propuesto y la explicación correspondiente para su funcionamiento.

Es importante mencionar que para la implementación del algoritmo se utilizarón tres estructuras de datos, las cuales son Arista, Grafo y Certificado, también, hacemos uso de tres funciones adicionales, TamCtf devuelve el tamaño de un certificado, TamGrf devuelve el tamaño del grafo y LlenaArreglo que únicamente llena los valores de un arreglo con un determinado valor.

El **Algorithm 1 Estructura para una arista** corresponde al pseudo-código para la implementación de una arista con dos nodos (nd1 y nd2).

Algorithm 1 *Estructura para una Arista*

```
1: typedef struct Arista{
2: nd1, nd2
3: } Art
```

El **Algorithm 2 Estructura para un grafo** corresponde al pseudo-código para la implementación de un grafo, almacenando en un arreglo todos sus nodos y aristas correspondientes.

Algorithm 2 *Estructura para un Grafo*

```
1: typedef struct Grafo{
2: int nds[]
3: Art art[]
4: } Grafo
```

El **Algorithm 3 Estructura para un certificado** corresponde al pseudo-código para la implementación de un certificado, almacenado en un arreglo de nodos.

Algorithm 3 *Estructura para un Certificado*

```
1: typedef struct Certificado{
2: nd[]
3: } certificado
```

El **Algorithm 4 TamCtf** corresponde al pseudo-código para la implementación de una función que permite obtener el tamaño de un certificado.

Algorithm 4 *TamCtf*

Require: certificado C
Ensure: Tamaño de un certificado
1: int i
2: **for** i = 0 ;$C.nd[i]! = 0$; i++ **do**
3: **end for**
4: **return** i

El **Algorithm 5 TamGrf** corresponde al pseudo-código para la implementación de una función que permite obtener el tamaño de un Grafo.

Algorithm 5 *TamGrf*

Require: Grafo G
Ensure: Tamaño de un grafo
1: int i
2: **for** i = 0 ;$G.nds[i]! = 0$; i++ **do**
3: **end for**
4: **return** i

El **Algorithm 6 LlenaArreglo** corresponde al pseudo-código para la implementación de una función que permite asignar un valor (num) a cada una de las posiciones de un arreglo.

Algorithm 6 *LlenaArreglo*

Require: int *A, int num
Ensure: Arreglo lleno de num
1: int i
2: **for** i = 0 to 100 **do**
3: A[i] = num
4: **end for**

Para el algoritmo propuesto, como lo solicita la práctica, se le asignó el nombre de **VerificacionHamilton**, también, recibe dos parámetros un Grafo G y un certificado C, además, regresa un valor entero que indica si el certificado C es o no un ciclo hamiltoniano para el grafo G, el número uno indica que si es un ciclo hamiltoniano, por el contrario, el número cero indica que no es un ciclo hamiltoniano. Comenzamos declarando cinco variables y un arreglo, las variables j e i nos servirán como iteradores para los ciclos que utilizaremos, la variable marcador no servirá para identificar si existe un camino, las variables tamcr y tamg que nos servirán para almacenar el tamaño del certificado y del grafo respectivamente y el arreglo pruebas que nos servirá para colocar una prueba de haber pasado por un determinado nodo. Asignamos a las variables el tamaño del certificado y grafo respectivamente mediante las funciones antes mencionadas (TamCtf y TamGrf), luego llenamos el arreglo de las pruebas con un valor cero qlo cual indica que no se ha recorrido ese nodo, de manera similar utilizando la función LlenaArreglo. Posteriormente nos aseguramos que el certificado cumpla con las condiciones necesarias para ser un ciclo hamiltoniano, entonces, confirmamos que tiene un tamaño adecuado y que no contenga un único elemento, lo anterior mediante una condición if, si cualquiera de estas dos condiciones no se cumplen devolvemos un cero, recordando que este valor indica que no es un ciclo hamiltoniano. Una vez verificado lo anterior iniciamos un ciclo for para verificar todo el certificado, asignando nuestro marcador en cero, luego iniciamos otro ciclo for para verificar si existe un camino, lo anterior mediante dos if, si cualquiera de estas condiciones se cumple asignamos a la variable marcador un número uno y termina el ciclo, estó nos indicara que si existe un camino. Ahora, revisamos la variable marcador para comprobar que existe un camino, si no existe devolvemos un cero. Una vez confirmado lo anterior verificamos los nodos y colocamos pruebas de haber pasado por los nodos, sin embargo, si pasamos por un mismo nodo devolvemos un cero, puesto que este no es candidato a ser hamiltoniano. Finalmente, después de todo lo anterior mediante otro ciclo for verificamos si pasamos por todos los nodos, mediante un if preguntamos si algún valor del arreglo pruebas contiene un cero dentro del rango del tamaño del grafo, si esto se cumple implica que no es hamiltoniano y regresa un cero, por el contrario, si esto no se cumple para ningún valor del arreglo terminamos el ciclo y devolvemos un 1, lo cual indica que el certificado si es un ciclo hamiltoniano.

Algorithm 7 *VerificacionHamilton*

Require: Grafo G, certificado C
Ensure: Decide si el certificado es un ciclo hamiltoniano

```
1: Int i
2: Int j
3: Int pruebas[100]
4: Int marcador
5: Int tamcr = TamCtf(C)
6: Int tamg = TamGrf(G)
7: LlenaArreglo(pruebas, 0)
8: if tamcr != (tamg+1) || C.nd[0] != C.nd[tamg] then
9:     return 0
10: end if
11: for j = 0 to tamcr-1 do
12:     marcador = 0
13:     for i = 0 to 100 do
14:         if G.art[i].nd2 == C.nd[j + 1]&&G.art[i].nd1 == C.nd[j] then
15:             marcador = 1
16:             break
17:         end if
18:         if G.art[i].nd2 == C.nd[j]&&G.art[i].nd1 == C.nd[j + 1] then
19:             marcador = 1
20:             break
21:         end if
22:     end for
23:     if marcador == 0 then
24:         return 0
25:     end if
26:     if pruebas[C.nd[j] − 1] == 0 then
27:         pruebas[C.nd[j]-1] = 1
28:     else
29:         return 0
30:     end if
31: end for
32: for i = 0 to tamg do
33:     if pruebas[i] == 0 then
34:         return 0
35:     end if
36: end for
37: return 1
```

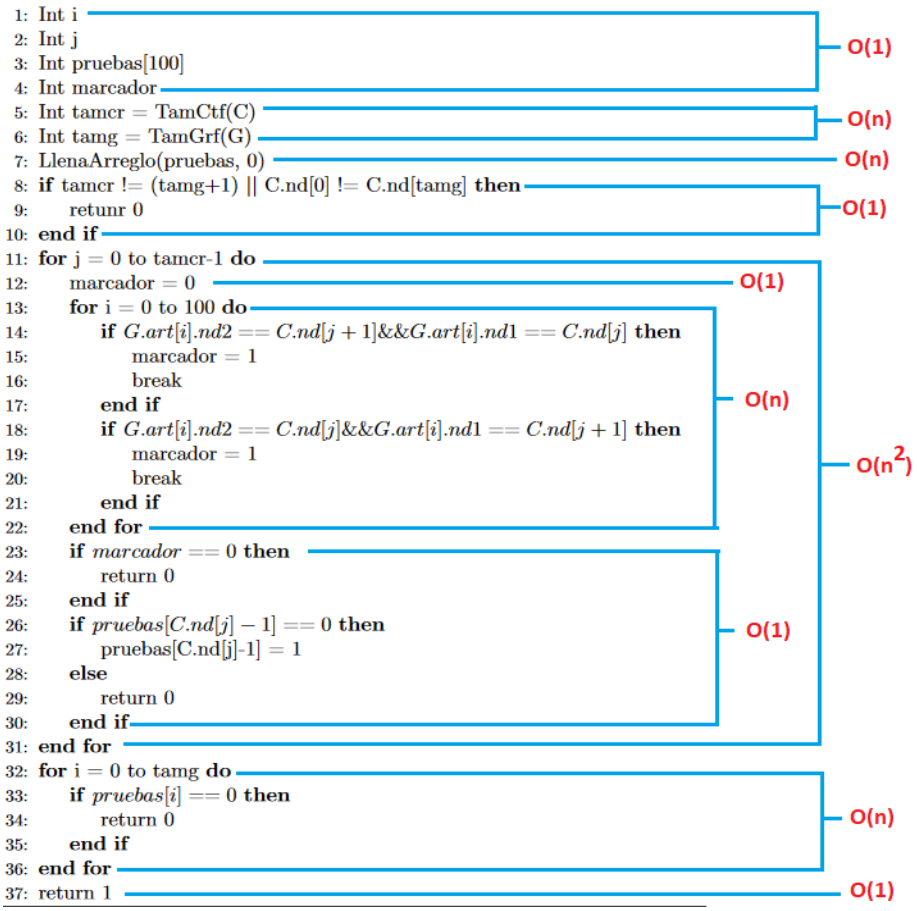
3 Experimentación y Resultados

En esta sección mostraremos los resultados de los análisis a priori y posteriori, además las pruebas del funcionamiento para el algoritmo propuesto.

3.1 Análisis a priori para el algoritmo propuesto.

Para el algoritmo **Algorithm 7** ***VerficacionHamilton*** llevaremos a cabo el análisis mediante segmentos de código. Si observamos el pseudo-código.

Se tiene que



De lo anterior podemos concluir que

$\therefore$ ***VerficacionHamilton*** $\in O(n^2)$

3.2 Análisis a posteriori para el algoritmo propuesto.

Para realizar las pruebas comenzamos implementando el algoritmo como una función llamada ***VerificacionHamilton*** con base en el pseudo-código del **Algorithm 7**, mediante el Lenguaje de Programación en C, además, se implementaron las estructuras y funciones mostradas en el **Algoritmo propuesto**.

```c
int VerificacionHamilton(Grafo G, certificado C, int *ct){//Verifica si el certificado es hamiltoniano
    int i, j, marcador, pruebas[100];
    int tamcr = TamCtf(C);
    int tamg = TamGrf(G);

    LlenaArreglo(pruebas, 0);//Colocamos las pruebas en 0

    if(tamcr != (tamg + 1) || C.nd[0] != C.nd[tamg])//Revisamos si cumple para ser hamiltoniano
        return 0;

    for(j = 0 ;j < tamcr-1; j++){//Revisamos el certifcado
        marcador = 0;

        for(i = 0 ;i < 100; i++){//Revisamos si existe un camino
            if(G.art[i].nd2 == C.nd[j+1] && G.art[i].nd1 == C.nd[j]){
                marcador = 1;
                break;
            }
            if(G.art[i].nd2 == C.nd[j] && G.art[i].nd1 == C.nd[j+1]){
                marcador = 1;
                break;
            }
        }

        if(marcador == 0)//Confirmamos si existe un camino
            return 0;

        if(pruebas[C.nd[j]-1] == 0)//Colocamos pruebas de haber pasado por un nodo
            pruebas[C.nd[j]-1] = 1;
        else//En caso de pasar por un mismo nodo ya no es un ciclo hamiltoniano
            return 0;
    }

    for(i = 0 ;i < tamg; i++){//Revisamos si pasamos por todos los nodos
        if(pruebas[i] == 0)
            return 0;
    }
    return 1;
}
```

Figura 1. Código para el **algoritmo propuesto** en lenguaje C.

Además, utilizaremos la función **Principal** la cual nos servirá para ejecutar el algoritmo y análizar los resultados obtenidos.

Para realizar el análisis a posteriori la función **VerificacionHamilton (Figura 1)** debe recibir un parámetro por referencia **(ct)** que obtendrá el número ejecuciones para cada línea de código.

```
int VerificacionHamilton(Grafo G, certificado C, int *ct){
    int i, j, pruebas[100], marcador;(*ct)++;(*ct)++;(*ct)++;(*ct)++;
    int tamcr = TamCtf(C);(*ct)++;
    int tamg = TamGrf(G);(*ct)++;

    LlenaArreglo(pruebas, 0);(*ct)++;

    if(tamcr != (tamg + 1) || C.nd[0] != C.nd[tamg]){
        (*ct)++;
        return 0;
    }(*ct)++;

    (*ct)++;
    for(j = 0 ;j < tamcr-1; j++){
        (*ct)++;
        marcador = 0;
        for(i = 0 ;i < 100; i++){
            (*ct)++;
            if(G.art[i].nd2 == C.nd[j+1] && G.art[i].nd1 == C.nd[j] ){
                (*ct)++;
                (*ct)++;
                marcador = 1;
                break;
            }
            if(G.art[i].nd2 == C.nd[j] && G.art[i].nd1 == C.nd[j+1]){
                (*ct)++;
                (*ct)++;
                marcador = 1;
                break;
            }
        }
        (*ct)++;
        if(marcador == 0){
            (*ct)++;
            (*ct)++;
            return 0;
        }
        if(pruebas[C.nd[j]-1] == 0){
            (*ct)++;
            (*ct)++;
            pruebas[C.nd[j]-1] = 1;
        }else{
            (*ct)++;
            (*ct)++;
            return 0;
        }
    }
    (*ct)++;

    (*ct)++;
    for(i = 0 ;i < tamg; i++){
        (*ct)++;
        if(pruebas[i] == 0){
            (*ct)++;
            (*ct)++;
            return 0;
        }
    }
    (*ct)++;
    return 1;(*ct)++;
}
```

Figura 2. Código para el **algoritmo propuesto** con el parámetro **ct**.

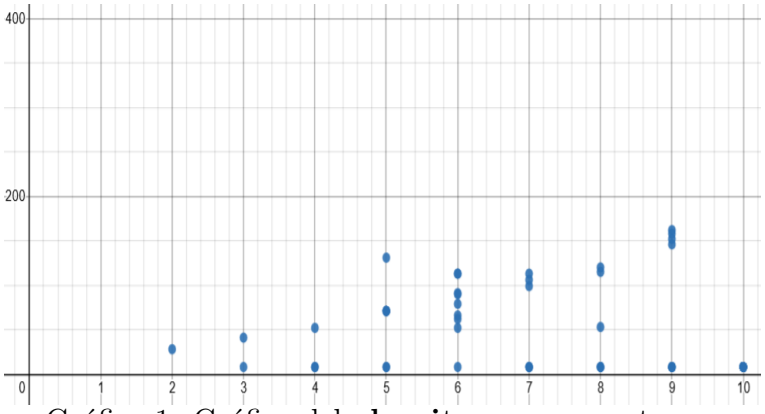
Para obtener los puntos de muestra para el análisis a posteriori, alimentamos el algoritmo con distintos grafos y certificados, en donde algunos de ellos si corresponden a un ciclo hamiltoniano y otros que no son ciclos hamiltonianos. El resultado de esta ejecución arroja los siguientes valores.

n	Ct
2	28
3	41
4	52
5	71
5	71
6	91
7	113
7	99
8	120
9	152
9	146

Tabla 1. Valores de la ejecución para el **algoritmo propuesto**.

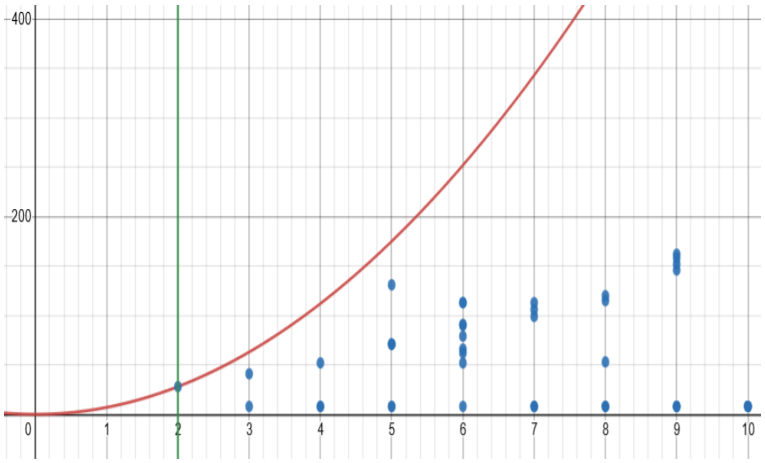
En la tabla es posible observar los resultados de la ejecuión en donde n representa la cantidad de nodos que contiene un grafo y ct el número de pasos ejecutados por el algoritmo propuesto. Es importante mencionar que algunos puntos se repiten, esto ocurre debido a que algunos certificados no cumplen con las condiciones necesarias para ser un ciclo hamiltoniano, entonces, son descartados por el algoritmo propuesto.

Luego de revisar los resultados de la ejecución del algoritmo, copiamos los valores que obtuvimos y los graficamos mediante la calculadora Gráfica del sitio web Desmos, observamos que efectivamente tiene una complejidad temporal cuadrática $O(n^2)$.



Gráfica 1. Gráfica del **algoritmo propuesto**.

Ahora proponemos la función $\mathbf{g(n)} = 7(n^2) \in O(n^2)$ con $n_o = 2$ para la cota superior.



Gráfica 2. Gráfica del **algoritmo propuesto** con la función g(n).

En color azul podemos observar los valores de la ejecución del algoritmo propuesto, en color rojo la función propuesta **g(n)** y en color verde el n_o.

3.3 Características del equipo de cómputo

- Procesador: Ryzen 5 1400
- Tarjeta gráfica: GTX 1050 ti de la marca PNY
- Tarjeta Madre: Asus A320M-K
- Memoria RAM: 8 Gb
- Disco duro: Disco Duro Interno 1tb Seagate

3.4 Ejemplos del funcionamiento del algoritmo propuesto

En esta sección se mostrarán los resultados de las pruebas ejecutadas para el algoritmo propuesto. Para todas las pruebas se utilizaron archivos de texto, los cuales contienen distintos grafos y certificados.

Primera prueba: En esta prueba verificamos 6 certificados para un grafo. Para cada prueba se agregaron certificados que si son ciclos hamiltonianos para el grafo y otros que no son ciclos hamiltonianos. La **Figura 3** corresponde a un grafo utilizado para la primera prueba.

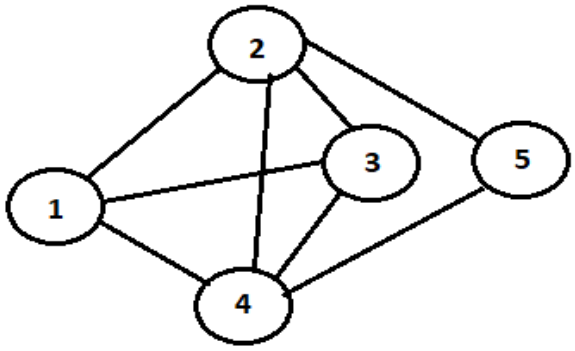


Figura 3. Grafo para la primera prueba.

Para el grafo anterior se probaron los siguientes certificados, en donde es posible corroborar cuales si corresponden a un ciclo hamiltoniano.

```
El certificado: 1 2 4 1
No es un ciclo hamiltoniano.
Numero de pasos: 8

El certificado: 5 4 2 3 4 5 2 3
No es un ciclo hamiltoniano.
Numero de pasos: 8

El certificado: 1 2 5 4 3 1
Si es un ciclo hamiltoniano.
Numero de pasos: 71
```

Figura 4. Resultados de la ejecución para tres certificados.

Si observamos los resultados podemos notar que para los certificados que no cumplen con las condiciones para ser ciclo hamiltoniano, solo toma 8 pasos determinar que no es un ciclo hamiltoniano. Por el contrario, para un certificado que si es hamiltoniano resultó en 71 pasos.

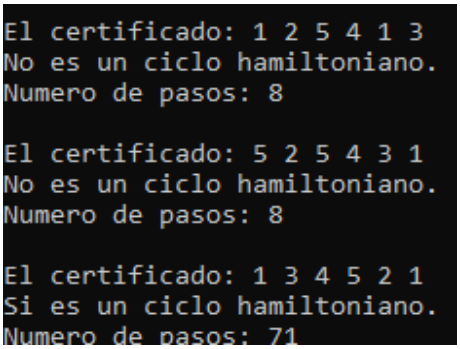


Figura 5. Otros resultados de la ejecución para tres certificados.

De manera similar, para esta ejecución podemos observar que los certificados no cumplen con las condiciones para ser hamiltoniano, puesto que no terminan en el nodo que comenzaron.

Segunda prueba: En esta prueba verificamos 3 certificados para un grafo. En esta prueba para los tres certificados solo uno de ellos corresponde a un ciclo hamiltoniano. La **Figura 6** corresponde a un grafo utilizado para la segunda prueba.

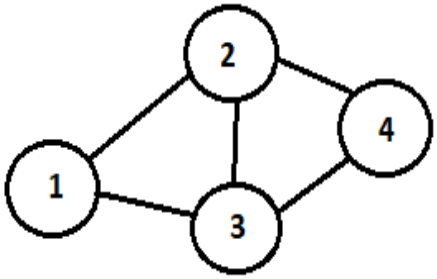


Figura 6. Grafo para la segunda prueba.

Para el grafo anterior se probaron los siguientes certificados, en donde es posible corroborar que solo un certificado corresponde a un ciclo hamiltoniano.

```
El certificado: 1 2 4 3 1
Si es un ciclo hamiltoniano.
Numero de pasos: 52

El certificado: 2 3 4 2
No es un ciclo hamiltoniano.
Numero de pasos: 8

El certificado: 1 2 3 4 2
No es un ciclo hamiltoniano.
Numero de pasos: 8
```

Figura 7. Resultados de la ejecución para los tres certificados.

Tercera prueba: En esta prueba, de igual manera verificamos 3 certificados para un grafo. Para los tres certificados solo uno de ellos corresponde a un ciclo hamiltoniano. La **Figura 8** corresponde a un grafo utilizado para la tercera prueba.

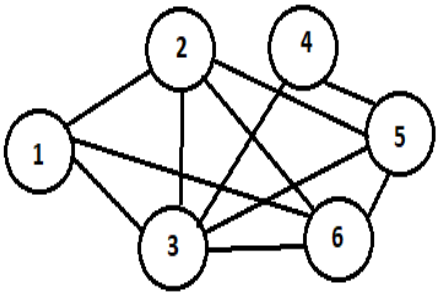


Figura 8. Grafo para la tercera prueba.

Para el grafo anterior se muestran los resultados de la ejecución para los certificados.

```
El certificado: 1 4 6 2 3 5 1
No es un ciclo hamiltoniano.
Numero de pasos: 113

El certificado: 6 5 4 3 1 2 6
Si es un ciclo hamiltoniano.
Numero de pasos: 91

El certificado: 6 2 3 5 2 3 6
No es un ciclo hamiltoniano.
Numero de pasos: 66
```

Figura 9. Resultados de la ejecución para los certificados.

Cuarta prueba: En esta prueba, de igual manera verificamos 3 certificados para un grafo, el cual contiene 7 nodos. La **Figura 10** corresponde a un grafo utilizado para la cuarta prueba.

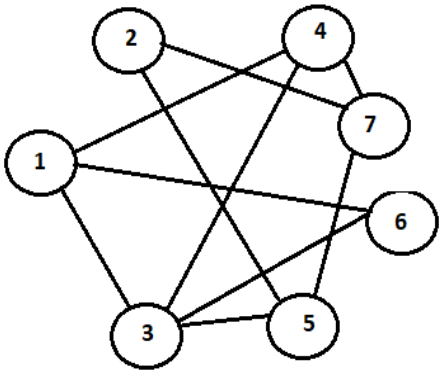


Figura 10. Grafo para la cuarta prueba.

A continuación, se muestran los resultados de la ejecución para los certificados.

```
El certificado: 1 3 4 1
No es un ciclo hamiltoniano.
Numero de pasos: 8

El certificado: 1 2 3 4 5 6 7 1
No es un ciclo hamiltoniano.
Numero de pasos: 113

El certificado: 1 4 7 2 5 3 6 1
Si es un ciclo hamiltoniano.
Numero de pasos: 99
```

Figura 11. Resultados de la ejecución para los certificados.

Podemos observar que en está prueba, para uno de los certificados le tomó 113 pasos para determinar que no es un ciclo hamiltoniano, lo anterior ocurrió debido a que el certificado si cumple con las condiciones necesarias para ser un ciclo hamiltoniano, entonces, el algoritmo ejecutará todas las instrucciones para determinar que no es un ciclo hamiltoniano. Por otro lado, para el certificado que si es un ciclo hamiltoniano le tomó 99 pasos.

4 Conclusiones

En está sección se incluyen las conclusiones generales e individuales para la práctica.

Conclusiones generales: Durante el desarrollo de está práctica se ha presentado un algoritmo para verificar en tiempo polinomial si un certificado es un ciclo hamiltoniano de un grafo implementado mediante el uso del lenguaje de programación en C para el cual, se le asignó el nombre de ***VerificacionHamilton***.

Para esté algoritmo se llevó a cabo un análisis a priori y posteriori, además, de varias pruebas con el objetivo de corroborar un correcto funcionamiento del algoritmo propuesto. También, se graficarón los valores obtenidos de las ejecuciones del algoritmo, análizando estás gráficas es posible confirmar los resultados obtenidos en el análisis a priori, ya que si observamos en la Gráfica 2 al proponer una función cuadrática g(n) todos los valores obtenidos son acotados por g(n), lo cual implica que efectivamente el algoritmo tiene un orden de complejidad cuadrático.

Durante todas las pruebas es posible observar que el algoritmo funciona correctamente, ya que determina cuales de los certificados si corresponden a un ciclo hamiltoniano. Para algunos certificados que no corresponden a un ciclo hamiltoniano podemos notar que utiliza más pasos para determinarlo, esto se debe a que el certificado efectivamente cumple con las condiciones para ser un candidato a ciclo hamiltoniano pero en algún momento esté vuelve a pasar por un mismo nodo, con lo cual el algoritmo determina que no es un ciclo hamiltoniano para el grafo correspondiente.

Conclusiones individuales (Brayan Ramirez Benitez): Para está práctica fue un poco complicado implementar el algoritmo propuesto, ya que está requeria de 3 estructuras para funcionar, anteriormente no había implementado ninguna estructura similar a la estructura grafos, sin embargo, con una comprensión básica de la teoría de grafos es posible implementarla. Por otro lado, la implementación de algunas funciones que nos servirán para el algoritmo propuesto fue relativamente sencilla.

5 Bibliografía

References

[1] Aguilar, I. (2019) *Introducción al análisis de algoritmos.* PDF. Recuperado de http://ri.uaemex.mx/bitstream /handle/20.500.11799/105198/Libro%20Complejidad.pdf? sequence=1&isAllowed=y

[2] Cormen, E. A. (2009) *Introduction to Algorithms.* 3Rd Ed. (3.a ed., Vol. 3). Phi.

[3] UDG. (2021) *.4.2 PASEOS Y CIRCUITOS HAMILTONIANOS. (s. f.). UDG. Recuperado 4 de diciembre de 2021, de http://mate.cucei.udg.mx/matdis/5gra/5gra42.htm*

[4] Formella, A. (2010). *Algoritmia Avanzada: Teoría de Grafos. Recuperado 3 de diciembre de 2021, de https://www.geeksforgeeks.org/np-completeness-set-1/*

[5] *GeeksforGeeks. (2021, 29 noviembre). NP-Completeness — Set 1 (Introduction). Recuperado 3 de diciembre de 2021, de https://www.geeksforgeeks.org/np-completeness-set-1/*

[6] *L. Hdez, L. C. (2016). Problemas NP Completos. Scribd. Recuperado 4 de diciembre de 2021, de https://es.scribd.com/document/248373617/Problemas-NP-Completos*

[7] *Kalsang, T. (2018). NP-complete problem — Definition, Examples, & Facts. Encyclopedia Britannica. Recuperado 3 de diciembre de 2021, de https://www.britannica.com/science/NP-complete-problem*

[8] *LUDA UAM-Azc. (2016). UAM. Recuperado 4 de diciembre de 2021, de http://aniei.org.mx/paginas/uam/CursoAA/curso_aa_30.html*