Instituto Politécnico Nacional
Escuela Superior de Cómputo

Análisis de Algoritmos, Sem.: 2022-1, 3CV11, Práctica 7, 16 diciembre 2021

Práctica 7: Verificación en Tiempo Polinomial

Vanegas García Andrés, Vaquera Aguilera Ethan Emiliano.

*avanegasg*1601@*alumno.ipn.mx*, *evaqueraa*@*alumno.ipn.mx*

Resumen:
Para la practica en cuestión se realizo un algoritmo y su análisis para resolver un problema NP-Completo que consiste en verificar si un certificado es o no un ciclo Hamiltoniano de un grafo. Una vez presentado el algoritmo se realiza el análisis a priori donde por conceptos teóricos se determina que orden de complejidad se resuelve el problema, después en el análisis a posteriori se realiza una formulación del comportamiento con base a lo teorico.

Palabras Clave

C: C es un lenguaje de programación estructurado, usado por ya poco más de 52 años, esto brindándole un amplio repertorio de bibliotecas, además de un gran soporte de la comunidad que lo utiliza. C al ser un lenguaje de programación estructurado comparte similitudes con otros lenguajes desarrollados en la época además de que es un lenguaje de alto nivel, que de igual manera pude ser utilizado a bajo nivel para programar en ensamblador y rutinas de memoria.

Complejidad de un Algoritmo: Este es el punto en el que se mide la complejidad de resolución de un algoritmo por medio de una computadora, poniendo énfasis en el uso de la memoria de este sistema además de, el tiempo que este toma para desarrollarlo. Esta desde el inicio de la computación es una problemática que ha ido evolucionando a modo de que hoy en día se utiliza el análisis de algoritmos para so resolución y determinar la complejidad de un algoritmo en especifico.

Análisis a priori y posteriori: Según lo comprendido en clase. El análisis a priori en computación es analizar un algoritmo, de manera formalmente teórica y matemáticamente, primero haciendo un análisis de sus posibles

pseudocódigos y después en base a esto realizar una formulación del comportamiento de este en base a una función matemática.

Por otro lado el análisis a posterior, es la recolección de datos por medio de la experimentación, que en el caso de un algoritmo es la programación de este para así tomar una serie de pruebas en base a las veces ejecutadas y la cantidad de veces que este algoritmo tarda para terminar.

Iteración: Una iteración es la forma de hacer una serie de pasos repetidas veces hasta llegar al resultado esperado en un rango esperado sin sobrepasar este, las iteraciones pueden ir de cualquier manera en la vida real siendo como realizar un ejercicio matemático de una raíz, una potencia numérica o un logaritmo, y en computación serian las sentencias de while, for, do-while, etc.

Recursividad: La recursividad consiste mas que nada en la realización iterativa de un mismo lenguaje descriptivo, pero sin tener que mandar a llamar a una función iterativa. En palabras mas simples se trata de el desarrollo de una función que se llama a si misma, y después de llegar a un resultado esta regresa de vez en vez al inicio y finalmente otorgar el resultado final.
Grafo: Es un conjunto no vació de objetos (o vértices) que están unidos por una arista que pueden ser orientados o no. **P:** Es un problema que se puede resolver en un tiempo polinómico y es tratable. **NP:** Es un problema que se puede verificar en un tiempo polinómico, osea que se tenga un solución y se tenga un algoritmo para verificar en tiempo polinomial que esa es una solución al problema.

1 Introducción

Con el paso de los años y el surgimiento de nuevas necesidades, en la actualidad existen diversos problemas de varios ámbitos que se pueden resolver mediante un algoritmo que nos va a permitir encontrar una o mas soluciones para ese problema u otros del mismo tipo. Para algunos encontrar un algoritmo para un problema se volvió una tarea difícil, por lo que a lo largo de tiempo se fueron estudiando y clasificando los problemas con mayor relevancia, es así como surgieron los problemas P, NP y NP-Completos. Y a continuación en esta practica se propondrá un algoritmo que de solución a un problema NP-Completo. El problema consiste en verificar si un certificado es un ciclo Hamiltoniano de un grafo, para ello se tiene que verificar que el certificado parta y llegue al mismo vértice pasando por todos los demás y sin repetir alguno.
Este problema es NP-Completo debido a que, primero el algoritmo no determina el ciclo Hamiltoniano, sino verifica que una solución (certificado) es o no un ciclo Hamiltoniano en un tiempo polinomial, por lo tanto, es un problema NP. Y es un problema NP-Completo debido a que verifica si el certificado es un ciclo para cualquier grafo $G = (V, E)$.
Finalmente para el análisis a priori se determinara si el algoritmo propuesto tiene un orden polinomial con base a conceptos teóricos. Después se realizara el análisis a posteriori donde con la implementación del algoritmo en lenguaje C se recabaran datos y con ellos elaborar una gráfica que nos permitirá observar el tiempo y espacio de ejecución.

2 Conceptos Básicos

Θ: La sigla griega Teta (Θ) es la denotación que describe de manera clara, el comportamiento similar de las funciones matemáticas del peor y mejor caso, esto siendo una referencia clara para hacer distinguir de Ω y O, que tiene definiciones diferentes.

Definición formal: $\Theta(\mathrm{g(n)}) = \{f(n) : existe\, n, c_1, c_2, > 0\, y\, n_0 > 0 \,||\, 0 \leq c_1 g(n \leq f(n) \leq c_2 g(n)\, \forall\, n \geq n_0\}$

O: Big O es la denotación usada para el conjunto de funciones que se acotan por encima de f(n), describiendo de manera clara el comportamiento da la función del peor caso, y denota los puntos experimentales de esta también.

Definición Formal: $O(g(n)) = \{f(n) : existen, c > 0 \,\&\, n_0 > 0\, constantes \,||\, 0 \leq f(n) \leq cg(n)\, \forall\, n \geq n_0\}$

Ω: La denotación Omega (Ω) es la denotación usada para acotar por abajo a la función f(n), siendo que es usada para describir al conjunto de funciones que están por debajo de la función experimental f(n), y de igual manera sirve para describir los puntos experimentales de igual manera.

Definición Formal: $\Omega(g(n)) = \{f(n) : existen, c > 0 \,\&\, n_0 > 0 \, constantes \,||\, cg(n) \leq f(n) \,\forall\, n \geq n_0\}$

Recursividad: La recursividad es la serie de procesos definidos, en la cual se quiere la sucesión de varios factores y en vez de usar una función de la clase iterativa, se usan los mismos procesos ya definidos para crear esa sucesión de procesos, Imaginemos una muñeca de Matryoshkas (tipo de muñecas rusas), cada que abrimos una para sacar una nueva estas son mas pequeñas y además, todas y cada una de de ellas son iguales, usando ese ejemplo podemos definir un poco mejor la función recursiva, donde cada vez que mandamos a llamar a la función los valores de esta pueden ir disminuyendo o aumentando, dependiendo de lo que se quiera solucionar, hasta llegar a una condición de frontera, y de igual manera a pesar de que los valores cambien, lo que nunca se modifica son los procesos de la función recursiva que se mantiene de la misma manera.

Algorithm 1 Algoritmo Ejemplo Recursivo

```
function RECURSIVO(n)
    a ← n
    if a ≠ 0 then
        return Recursivo(a ← a − 1)
    else
        return a
    end if
end function
```

Problemas NP-Completos: Se dice que un problema L es NP-Completo si $L \in NP$, además se tiene que cumplir $L' \geq L$ para cada $L' \in NP$, en otras palabra, que el algoritmo para resolver el problema L pueda resolver en tiempo polinómico cualquier problema contenido en NP.

Ciclo Hamiltoniano: Si $G = (V, E)$ es un grafo con un numero de vértices mayor o igual a 3, osea $|V| \leq 3$, se dice que G contiene un ciclo Hamiltoniano si existe un ciclo en G que contenga cada vértice de V. Un camino Hamiltoniano es un camino de G que contiene todos los vértices.

3 Experimentación y Resultados

Para la abstracción del problema en cuestión se empezó por el análisis de como funciona un grafo y como se conecta este a través de los arcos para así conectar los nodos a través del sistema. Un arco de un grafo seria la conexión que une a dos nodos del grafo, siendo que podemos darle valores o números de la ruta que usa el sistema, esto para así tener el numero de ruta y que tan óptima es esta. Ahora un nodo es la parte donde se conecta el sistema, donde podemos ver las rutas, aquí es donde entra el algoritmo de ciclos Hamilton, donde este sistema se trata de comportar a modo que el sistema busca el camino mas óptimo por medio de un árbol de búsqueda, ya que le sistema empieza en un nodo inicial y para completar un ciclo se debe de llegar al mismo nodo inicial, sin que el sistema se cruce por un mismo nodo mas de dos veces.
Para la implementación se busco de primera instancia el uso de arboles binarios para la construcción de un grafo, pero al parecer si noto que el sistema no necesita el uso de grafos tan elaborados, por lo que de primeras se hizo uso de una n cantidad de nodos por medio de un array, y en otro array se colocaron las posibles conexiones del grafo, eso para reducir el trabajo del uso de grafos creados por arboles binarios, en el array doble se le colocan los arcos tenemos que un 1 representa el uso de una conexión existente entre un nodo a otros siendo que tenemos n nodos donde puede tener n-1 conexiones, asiendo referir que el sistema puede estar conectado a todos los nodos del grafo, de ahí buscaremos el posible ciclo Hamiltoniano.
Ahora el ciclo debe de estar dado de la siguiente manera ya que el sistema debe de analizar nodo por nodo del sistema esto creando una breve tabla con un algoritmo greedy para así poder encontrar una solución, esta no sera la mas óptima de todo el sistema.
Para saber si el arco es permitido para la conexión del nodo se manda a llamar a una función que tiene orden cuadrática donde se ejecuta una comparativa y si el arco esta inicializado en 1 podemos intentar incluirlo a la posible solución del sistema, recordemos que la solución que vamos a encontrar solo es una solución que no es la mas óptima.

Y por ultimo se hace una llamada recursiva al sistema en donde para así poder saber el siguiente arco y de ahí poder y analizando los datos de los sistemas.

Al final el sistema consigue tener el orden polinomial, ya que el sistema es sencillo de razonar pero es difícil de implementar ya que al ser un problema N-P completo es mas complicado de implementar se implemento de maneras mas sencilla ya que le sistema no tiene una solución mas exacta para ese sistema. Podremos ver en la figura 1 el análisis a priori del sistema.

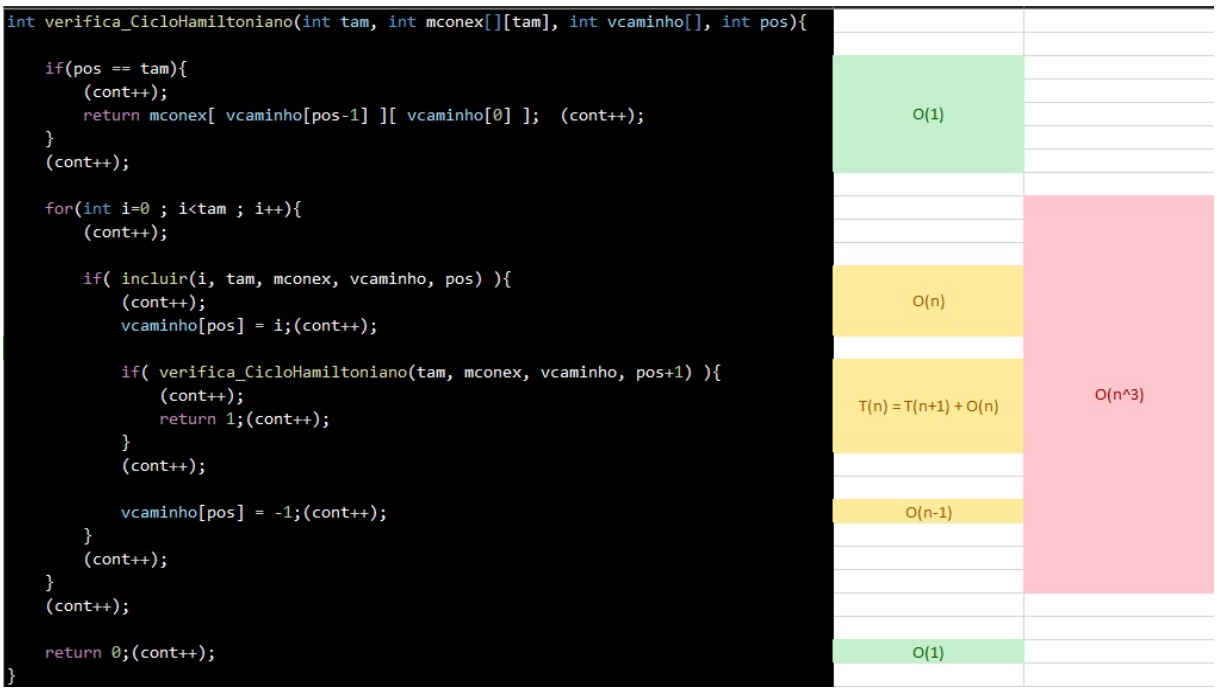


Figura 1. Análisis a priori del Algoritmo de ciclos Hamiltoniano

Para el análisis a posteriori se hizo el use de ir aumentando la cantidad de nodos del sistema, junto con sus respectivos arcos que se desean e iterando los valores de los arcos entre 0 y 1 para así poder ir modificando los valores de las conexiones entre los nodos. En la figura 2 podemos ver el análisis a posterior, y como podremos ver los puntos dispersos en el plano son los punto experimentales del análisis y las curvas serian el comportamiento teórico de la función.

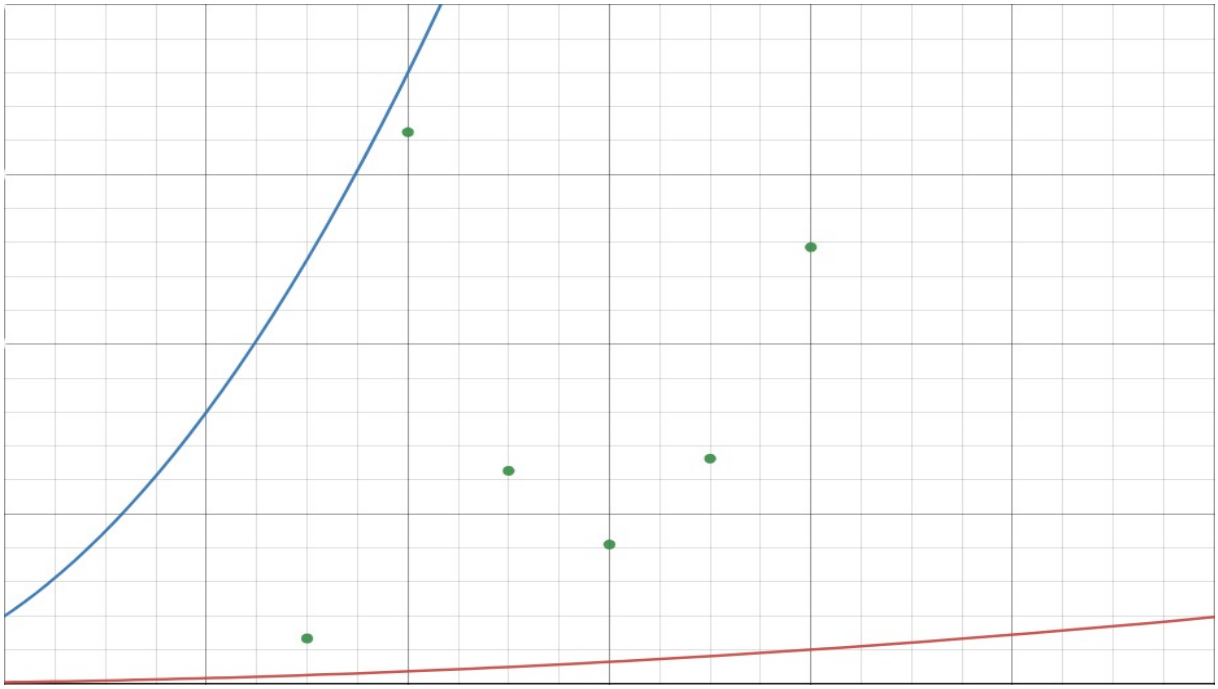


Figura 2. Análisis a posteriori del Algoritmo de ciclos Hamiltoniano

Para finalizar esto el análisis del programa, podemos ver que el código funciona de la manera correcta siendo que el sistema es mas sencillo pero al genera el mismo procedimiento que el ciclo Hamiltoniano del sistema.

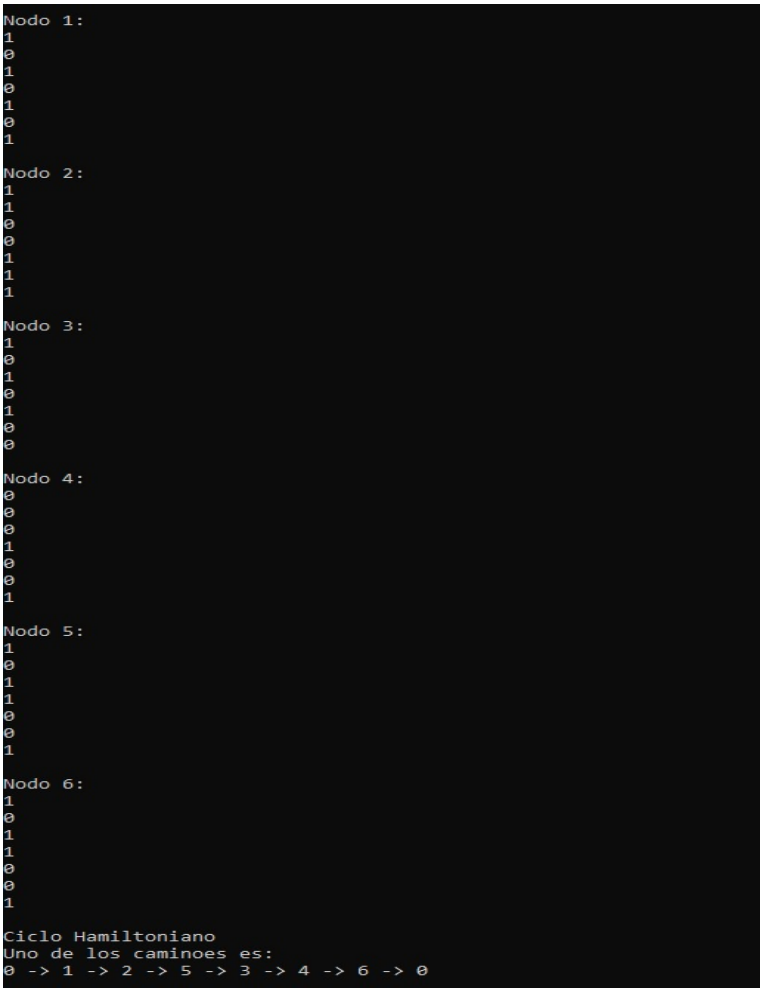


Figura 3. Ejecución del Algoritmo de Hamilton

Características PC Venegas Garcia Andres
-Procesador Intel i5-10400f a 2.9 Ghz six core
-16 Gb de Ram
-Nvidia Geforce GTX 970

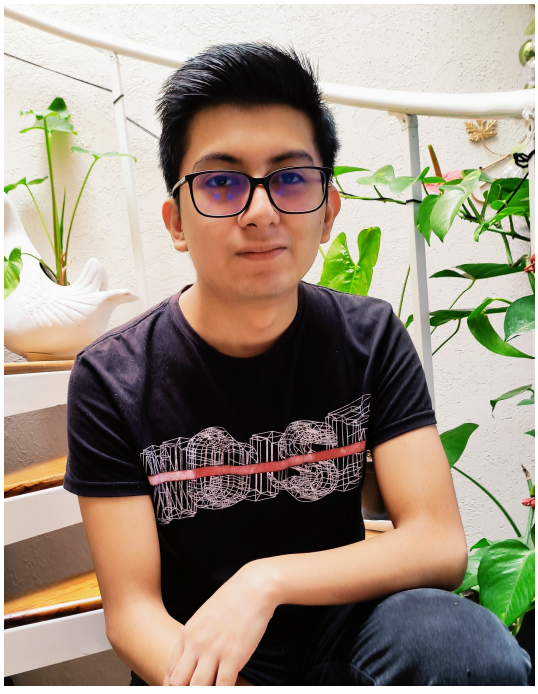
Características PC Vaquera Aguilera Ethan Emiliano
-Procesador ryzen 7-2700 a 3.2 Ghz octa core
-16 Gb de Ram
-AMD Radeon RX 5600XT

4 Conclusiones

Conclusión General: El realizar un algoritmo que solucione un problema NP, no es tarea sencilla, aunque el algoritmo presentado realiza su cometido el uso de algún paradigma o técnica se puede obtener tal vez un mejor resultado, aun asi lo implementado en este algoritmo fue sumamente impresionante, además el análisis y comprensión del problema fue un verdadero reto y se aplicaron la mayoría de conocimientos que hemos adquirido.

Conclusión Individual Vanegas García Andrés: Sin duda alguna realizar el análisis de este problemas fue una tarea difícil, además desarrollar el algoritmo tampoco fue una tarea fácil, aun así se logro que el algoritmo tuviera un orden de complejidad polinomial y fuera eficiente en su tarea, aunque se puede mejor su eficiencia implementando alguna otra técnica se logra obtener una solución aunque no sea una solución óptima. Para finalizar, creo que el trabajar en un problema difícil hace que pensemos en todas las herramientas que hemos aprendido en el curso para poder encontrar una solución y realizar dicho análisis.

Figura 4. Vanegas García Andrés



Conclusión Individual Vaquera Aguilera Ethan Emiliano: Para concluir el sistema es mas eficiente pero al mismo tiempo algo no me termina de cuadrar, ya que le sistema es muy eficiente como podemos ver en la gráfica que el sistema se comporta tal cual el comportamiento del orden de la función en ciertos intervalos de la función, siendo que el sistema es mas sencillo pero al mismo tiempo quisiera haber hecho el uso de grafos por arboles binarios pero el uso de estos sistemas es complicado pero mas geniales al mismo tiempo, siendo que le sistema es complicado de implementarlo de esta manera es sumamente impresionante, por lo que hace referencia además de como puede encontrar la ruta mas exacta sin ser la mas óptima.

Figura 5. Vaquera Aguilera Ethan Emiliano

4.1 Anexos

5 Bibliografía

programmerclick.com. (2021). Complejidad temporal y complejidad espacial. 2021, de programmerclick.com Sitio web: https://programmerclick.com/article/89671544092/
Agrawal, Manindra; Kayal, Neeraj; Saxena, Nitin: "PRIMES is in P". Annals of Mathematics 160 (2004), no. 2, pp. 781–793.
J.M.+Gimeno+y+J.L.+González. (2021). Recursividad. 2021, de web Sitio

web: http://ocw.udl.cat/enginyeria-i-arquitectura/programacio-2/continguts-1/2-recursividad.pdf
-. (2021). Algoritmo Greedy. 2021, de Universidad de Granada Sitio web: https://elvex.ugr.es/decsai/algorithms/slides/4

Maurette, M., Ojea, I. (2006). Programación Dinámica. Universidad de Buenos Aires Sitio Web: http://cms. dm. uba. ar/materias/1ercuat2009/optimizacion/Maurette$_{O}je$

Ralph P. Grimaldi. (1994).*Matemáticas Discretas y Combinatoria* . Massachusetts: Addison-Wesley Iberoamericana.p.578