Universidad de Santiago de Chile

Facultad de Ingeniería

Departamento de Ingeniería Informática

Laboratorio N°3: Metodología Ramificación y acotamiento

Integrante: Matías Cortés.

Ignacio Villarroel.

Profesor: Cristián Sepúlveda.

Ayudante: Cesar Rivera.

Asignatura: Algoritmos Avanzados.

Tabla de contenidos

Introducción…… 3

Método…… 4

a. Herramientas……..... 4

b. Desarrollo Experimental…… 4

Resultado y Análisis……..... 5

Discusión…… 7

Conclusión……... 7

Apéndice…….. 8

Referencia…… 8

Introducción

En el desarrollo de softwares, la resolución de problemas mediante la implementación de algoritmos es muy común, desde el momento donde se analiza el problema pasando por el modelado de las posibles soluciones hasta la misma construcción del algoritmo permite entender que existen diversas respuestas para una misma problemática. Existen diversos algoritmos que están enfocados a resolver los mismos problemas, sin embargo, hay algunos que son mejores que otros, en términos del tiempo de ejecución, es decir, en lo que estos se demoran al encontrar la solución.

En el presente informe se enfocará la implementación de un algoritmo específico; los algoritmos hacen referencia a un conjunto de instrucciones que encuentran una solución a un problema particular. La base del algoritmo a utilizar es la metodología por ramificación y acotamiento, esta metodología se enfoca en encontrar el óptimo a partir de la recursividad, es decir, ir construyendo una posible solución por medio del llamado del algoritmo a si mismo, de esta forma se pretende ir eliminando las soluciones que no cumplen el requisito de óptimo, posterior a esto se analizarán los resultados con diferentes valores entregados, además de graficar el tiempo de ejecución empleado a nivel computacional para el problema. Como objetivos principales, se tiene:

- Analizar el algoritmo creado en base a sus tiempos de ejecución.
- Obtener la complejidad.
- Identificar si lo obtenido está acorde con lo esperado en la experiencia.
- Ver si los algoritmos creados son mejorables.
- Comparar el uso de la tecnología de Ramificación y acotamiento con el método greedy y el de enumeración exhaustiva utilizados en las anteriores experiencias.

Por último, se observará la complejidad del algoritmo en relación al pseudo-código creado, de esta forma se concluirá que tan eficiente es el algoritmo, viendo si es factible su implementación para otros problemas computacionales en términos de la eficiencia y eficacia.

Método

a. Herramientas

Para el desarrollo óptimo de la experiencia se utilizaron diversos implementos de tipo software, los cuales fueron:

- Editor de texto ATOM para la programación en C. y uso de la terminal Windows.
- Archivos .txt los cuales fueron obtenidos mediante la plataforma uvirtual para las pruebas de código.
- Google drive para el traspaso de información para el desarrollo de trabajo grupal.

Respecto al hardware utilizado para la compilación y aplicación del código, el equipo que se utilizo fue un computador con:

- Procesador: AMD Ryzen 7 4800H CPU 2.90 GHz - 8 núcleos.
- Ram: 16 Gb.
- Sistema: Windows 10x64.

b. Desarrollo Experimental

Bajo el concepto de un algoritmo de ramificación y acotamiento, donde se calculan todas las posibles soluciones y entre todas ellas se buscan la más óptima, en ambos problemas

```
busquedaSolucion (array X, array Y, array Camino, int N, int ValorCompleto, int
PonderacionMaxima, int PonderacionTotal ): vacío

        IF ValorCompleto THEN
                tomarSolucion(X, Camino, N)
        ELSE
                i <- 0
                FOR i to N
                        IF CaminoExiste(X, i, N) = 0 THEN
                                IF X->PonderacionMaxima >= X->PonderacionTotal + Y->Pond[i]
                                        add(X, Y, i, N)
                                        CompletarSolucion(X, Y, N, 1, Camino)
                                        delete(X, Y, i, N)
                                ELSE
                                        CompletarSolucion(X, Y, N, 1, Camino)
```

Pseudocódigo 1.

se realizaron bajo esta técnica, es por eso que se desarrolló el pseudocódigo para cada problemática.

En el primer problema, se necesita encontrar las posibles combinaciones dado un subconjunto de elementos dado un ponderado, esto se aborda en el mismo estilo que el problema de la mochila mostrado en clases, puesto a que tiene una implementación similar, esto se hará de tal forma que se llama la función principal a si misma (recursión), donde en cada repetición se generará un camino como solución, a su vez se irán eliminando los peores caminos, así se irá cubriendo los caminos posibles y los que sean peores, se irán eliminando, repitiendo el mismo proceso hasta encontrar el camino óptimo que cumpla con el requisito de la ponderación. El orden de complejidad calculado para el problema es de $O(n!)$ mostrando ser un pseudocódigo en términos de eficiencia es bastante lento, por lo que esto se tendrá que ver reflejado en la ejecución del código.

Respecto a las limitaciones que existieron en la experiencia, estas se presentaron a nivel de ejecución puesto a que debido a la alta demora donde el código encuentre la solución no se lograron implementar todos los archivos.

Resultado y análisis

Al haber desarrollado el programa, mediante la utilización de time.h, se tabulan los resultados de las soluciones con sus respectivos tiempos además de la comparación de las soluciones obtenidas con las soluciones óptimas para cada caso.

Cantidad - Ponderación	Tiempo (s)	Solución encontrada	Solución óptima
6_500	0	1461	1461
8_500	0	1271	1271
10_500	0,001	2046	2046
12_500	0,001	1952	1952
14_500	0,001	1682	1682
16_500	0,001	2144	2144
18_500	0,001	1956	1956
20_500	0,001	2268	2268
24_1000	0,180	10627	10627
28_1000	1,296	11741	11741
32_1000	220,428	17573	17573

Tabla 1. Problema 1.

A continuación de grafica lo obtenido para poder analizarlo con la imagen ubicada en el *Apéndice*.

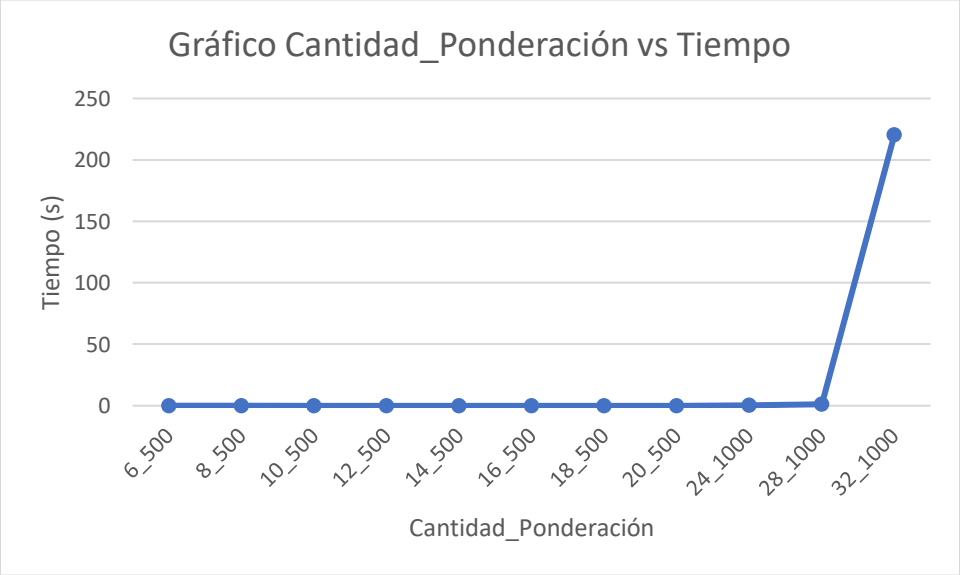

Gráfico 1. Problema 1.

Discusión

Al analizar los resultados obtenidos, en primer lugar, se hace notar que los tiempo de ejecución para tanto el problema son bajos para una baja cantidad de términos pero luego aumenta de manera abismal, donde no se logró el tiempo del siguiente archivo, esto hace notar que la cantidad de tiempo necesario es demasiado alta, sin embargo, en los que logró realizar muestra que encuentra el óptimo en el 100% de los casos. Esto es muy diferente comparado al laboratorio N°2 donde ejecutaba todos los archivos propuestos, sin embargo no encontraba el óptimo en la mayoría de los casos.

Lo anterior hace dar cuenta que, entre mayor utilización de recursos a nivel de memoria, mayor será la precisión en la búsqueda del óptimo, sin embargo, la utilización de la memoria implica una mayor cantidad de tiempo empleado haciendo el algoritmo poco eficiente, pero si, eficaz a diferencia de la metodología "Greedy".

A nivel gráfico podemos notar que en un principio el tiempo de ejecución bordea tiempos excesivamente ínfimos, hasta que se dispara con tiempo cercano a los 3 minutos y medio, esto se compara gráficamente con la *Imagen 1* encontrada en el sector de *Apéndice*, donde se nota que el gráfico creado a partir de la tabulación de los datos es muy parecido a la curva donde el orden de complejidad es de $O(n!)$, por lo que se cumple que se mantiene la complejidad calculada a nivel de pseudocódigo.

Conclusión

A partir de la implementación de la metodología de ramificación y acotamiento se indica que no se lograron resolver todos los casos de prueba que se brindaron para el desarrollo de la experiencia, esto de debe a que los tiempo de ejecución aumenta de manera factorial dependiendo de la cantidad de términos para el cálculo de la ponderación mas cercana a la deseada. Esto hace entender que el algoritmo no es el ideal para este problema específico, puesto que no vale la pena la utilización de recursos para tiempos tan extensos.

A nivel comparativo en el tema de algoritmos, se ve una mejora en la búsqueda de un óptimo, puesto que, ramificación y acotamiento encuentra el óptimo mientras que existen

situaciones donde la metodología “greedy” no la encuentra, al igual que en enumeración exhaustiva.

Respecto a los objetivos, se indica que se cumplieron en su totalidad, donde se logró efectivamente el desarrollo del algoritmo, además de obtener la complejidad del algoritmo por medio de la gráfico y los tiempos de ejecución y estos datos obtenidos fueron comparados exitosamente con los datos logrados en las experiencias anteriores.

Finalmente, a modo de inferencia se dice que el código realizado es ineficiente, donde es uno de los peores a nivel en la complejidad, sin embargo, es mejorable implementando mejores técnicas para una recursividad más efectiva.

Apéndice

Se muestra la gráfica de complejidades en apoyo a la comparativa de los gráficos obtenidos.

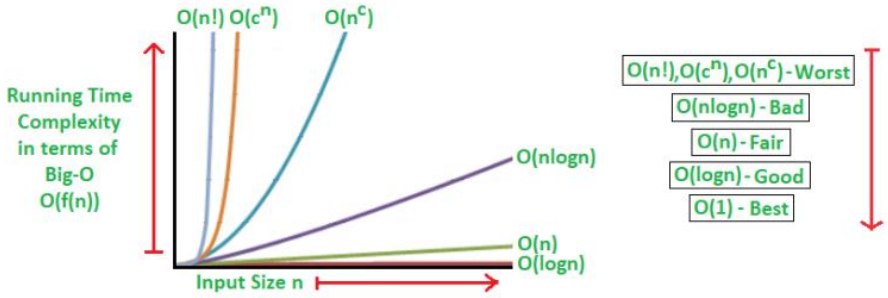

Imagen 1. Complejidades

Referencias

- GeeksforGeeks. (2021, 26 mayo). *Backtracking | Introduction*. Recuperado 26 de junio de 2022, de https://www.geeksforgeeks.org/backtracking-introduction/
- Jiménez, E. T. (2019, 13 octubre). *Técnica del backtracking o vuelta atrás*. Técnica del backtracking o vuelta atrás. Recuperado 25 de junio de 2022, de https://es.slideshare.net/PaolaTeran/tcnica-del-backtracking-o-vuelta-atrs