Universidad de Santiago de Chile

Facultad de Ingeniería

Departamento de Ingeniería Informática


# Laboratorio N°2: Greedy

Integrante: Matías Cortés.

Ignacio Villarroel.

Profesor: Cristián Sepúlveda.

Ayudante: Cesar Rivera.

Asignatura: Algoritmos Avanzados.

# Tabla de contenidos

# 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 "greedy" o goloso, esta metodología se enfoca en encontrar el óptimo pasando por todas las posibles soluciones del problema, se evaluará en el contexto de los problemas 1 y 2 si se encuentra la solución óptima, además de considerar los recursos utilizados por el algoritmo en términos de almacenamiento, además de analizar el tiempo de ejecución empleado a nivel computacional por cada problema.

- Analizar los algoritmos creados 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.

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 goloso, donde se calculan todas las posibles soluciones y entre todas ellas se buscan la más óptima, en ambos problemas se realizaron bajo esta técnica, es por eso que se desarrolló el pseudocódigo para cada problemática.

```
busquedaMaxima(array X, array Y, num PonderacionMaxima, num Cantidad): num
        Num valor <- 0
        Num ponderación <- 0
        Array posicionArreglo
        Para Num i <- 1 hacia Cantidad
                valor <- Y[i]
                ponderación <- X[i]
                posicionArreglo[i] <- valor/ponderación
        AlgoritmoDeOrdenamiento para ordenar los arreglos de mayor a menor
        Num PTotal <- 0
        Array PosiblesSumandos
        Num j <- 0
        Mientras j <= Cantidad y PTotal < PonderacionMaxima hacer
                Si PTotal + Y[j] <= PonderacionMaxima
                        PosiblesSumandos[j] <- X[j]
                        PTotal <- PTotal + Y[j]
                j <- j+1
        Num Suma <- 0
        Para Num k <- 0 hacia Cantidad
                Suma <- Suma + PosiblesSumandos
        Retornar Suma
```

*Pseudocódigo 1.*

En el primer problema, se necesita encontrar las posibles combinaciones dado un subconjunto de elementos dado un ponderado, la idea es encontrar la suma de términos que sea la más cercana a este, es decir encontrar un óptimo que maximice la suma acercándose al ponderado solicitado. Para esto se necesita los números ordenados de manera descendente de modo que a partir del primer término de la sucesión se utilice para la búsqueda viendo si es un valor óptimo para la solución, así sucesivamente hasta recorrer todas las soluciones, el orden de complejidad obtenido es de $O(n\ log(n))$ .

```
caminoCorto(matriz M, Num Dimension): Num
        Num origen <- minimoDeMatriz(M, Dimension)
        Num contador <- 0
        Array arreglosVistos
        Array solución
        arreglosVistos[origen] <- 1
        solución[contador] <- origen
        contador <- contador + 1
        Mientras vistos(arreglosVistos, Dimension) <- 0 hacer
                Num PMin <- mínimo(M[origen], arreglosVistos,
Dimension)
                solución[contador] <- PMin
                arreglosVistos[PMin] <- 1
                origen <- PMin
                contador <- contador + 1
        Num resultado <- suma(M, solución, dimensión)
        Retornar resultado

minimoDeMatriz(matriz M, Num D): Num
        Num FMin
        Num FPosicion <- 0
        Para Num i <- 0 hacia
                Num suma <- 0
                Para Num j <- 0 hacia D
                        Si M[i][j] <> -1
                                Suma <- suma + M[i][j]
                Si suma < FMin
                        FMin <- suma
                        FPosicion <- i
        Retornar FPosicion

Mínimo(array F, array A, Num D): Num
        Num min
        Num PMin <- 0
        Para Num i <- 0 hacia D
                Si F[i] <> -1 y F[i] < min y A[i] = 0
                        Min <- F[i]
                        PMin <- i
        Retornar PMin
```

*Pseudocódigo 2.*

Bajo el mismo concepto del problema 1, se aborda el problema 2 con la diferencia que el elemento a manipular para el recorrido es una matriz, para obtener el/los mínimo(s)

camino(s), para esto el algoritmos a partir de un vértice recorre el grafo, obteniendo todos los resultado posibles, para luego ver cuál es el mínimo, la única condición para el desarrollo es que el grafo debe ser conexo, vale decir que esta metodología que se toma el primer vertice como óptimo para buscar la solución. El orden de complejidad obtenido corresponde al de un algoritmo no óptimo el cual es $O(n^2)$ que según lo visto en la *Imagen 1* (ver anexo) indica que es un algoritmo deficiente.

Respecto a las limitaciones que existieron en laboratorio anterior, estas no se presentaron ya que el método greddy permite la rápida ejecución del programa al no tener que recorrer todas las soluciones.

## Resultado y análisis

a. Problema 1

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 | 1396 | 1461 |
| 8_500 | 0 | 1271 | 1271 |
| 10_500 | 0,001 | 2046 | 2046 |
| 12_500 | 0,001 | 1952 | 1952 |
| 14_500 | 0,001 | 1598 | 1682 |
| 16_500 | 0,001 | 2144 | 2144 |
| 18_500 | 0,001 | 1956 | 1956 |
| 20_500 | 0,001 | 2268 | 2268 |
| 24_1000 | 0,001 | 10517 | 10627 |
| 28_1000 | 0,001 | 11741 | 11741 |
| 32_1000 | 0,001 | 17195 | 17573 |
| 60_1000 | 0,001 | 18989 | 18989 |
| 130_1000 | 0,001 | 44980 | 45128 |
| 300_1000 | 0,001 | 96117 | 96237 |
| 1000_1000 | 0,002 | 24019 | 54503 |
| 5000_1000 | 0,004 | 54333 | 276457 |
| 10000_1000 | 0,007 | 75928 | 563647 |

*Tabla 1. Problema 1.*

b. Problema 2

Para el segundo problema, se obtuvo los siguientes tiempos con la solución encontrada por el algoritmo comparada en conjunto con la solución óptima al problema:

| Mapa | Tiempo (s) | Solución encontrada | Solución óptima |
|---|---|---|---|
| 4_30 | 0 | 30 | 30 |
| 6_78 | 0,001 | 78 | 78 |
| 8_249 | 0,001 | 275 | 249 |
| 10_601 | 0,002 | 713 | 601 |
| 12_677 | 0,002 | 853 | 677 |
| 16_1220 | 0,001 | 1409 | 1220 |
| 20_1547 | 0,002 | 2049 | 1547 |
| 24_2526 | 0,001 | 3113 | 2526 |
| 30_5039 | 0,001 | 5713 | 5039 |
| 40_5422 | 0,001 | 6758 | 5422 |

*Tabla 2. Problema 2.*

# Discusión

Al analizar los resultados obtenidos, en primer lugar, se nota que los tiempo de ejecución para tanto el problema 1 como el problema 2 son muy bajos independiente de la cantidad números para la ponderación y vértices respectivamente. Esto haciendo una comparación con el laboratorio anterior de enumeración exhaustiva y fuerza bruta hace una gran diferencia en lo que respecta a los tiempos de ejecución, la diferencia radica en que el método goloso solo se procura de buscar el primer buen resultado, esto se puede observar en la *Tabla 1* y *Tabla 2* en el comparativo de solución encontrada con la óptima, se ve que generalmente en los primeros casos encuentra la solución, sin embargo, mientras mas grande es la cantidad de números en el caso del problema 1 y mientras mas sea la cantidad de nodos en el problema 2, mas alejado estará del óptimo la solución. Lo anterior nos indica que la metodología “greedy” es imprecisa, ya que, a pesar de encontrar una solución, esta no es la óptima para el problema.

Viendo singularmenete para el problema 1, se ve que el tiempo de ejecución se mantiene a medida que se aumenta la cantidad, con una leve excepción al final al aumento

de 0,007 segundos, estos tiempos indican que sigue una línea de tener un algoritmo de muy baja complejidad.

Para el problema 2, las obtenciones de la solución no están tan alejadas de las óptimas como en el problema 1, sin embargo, al no ser la soluciones óptimas se considera que el método para esta clase de problemas no es la ideal. Por último, no se ve una diferencia en los tiempos de ejecución por lo que el orden de complejidad calculado para la implementación del pseudo-código esté erróneo o la cantidad de vértices todavía no es tan alta como para notar diferencias.

## Conclusión

A partir de la implementación de la metodología “greedy” se puede deducir que en términos de velocidad de ejecución es excelente, sin embargo, en la búsqueda del óptimo no es eficiente, puesto a que no encuentra lo solicitado, además de que el resultado obtenido a partir de esta metodología se aleja mas del óptimo entre mas cantidad de términos haya. Es por esto que se infiere que la implementación de goloso es para cierto tipo de contexto, en particular para situaciones donde la cantidad de términos sea muy baja, de esta forma encontrará el óptimo tardándose en tiempos muy bajos. Además, en lo que respecto al gasto de recursos de almacenamiento es bajísimo eso puede explicar los bajos tiempos de ejecución.

A modo sobre el desarrollo de la experiencia se indica que los objetivos propuestos se cumplieron a cabalidad, donde se logra tabular los tiempos de los archivos de los cuales se pudo calcular la solución, además de poder hacer una leve comparación de los ordenes de complejidad obtenidos.

Finalmente se considera a nivel grupal que el método goloso o “greedy” es un método que a nivel de cálculo de solución es ineficiente pero que a nivel de velocidad es muy rápido. Este método es ideal para problemas donde la cantidad de términos es muy baja.

# Apéndice

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

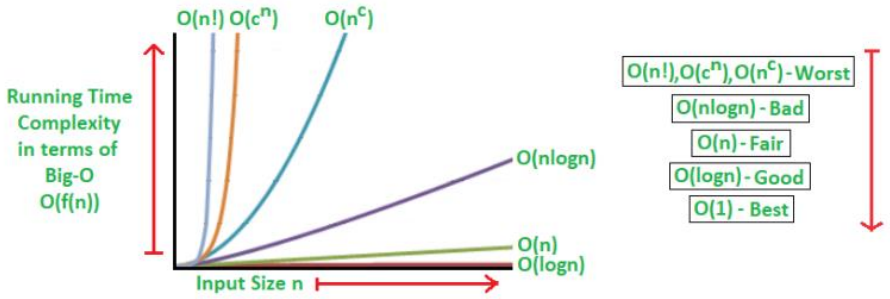


*Imagen 1. Complejidades*

# Referencias


- GeeksforGeeks. (2022, 19 abril). *Huffman Coding | Greedy Algo-3*. Recuperado 9 de mayo de 2022, de https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/
- Universidad de Wisconsin. (1993, 30 octubre). *Greedy codes*. Greedy Codes. Recuperado 9 de mayo de 2022, de https://www.sciencedirect.com/science/article/pii/009731659390085M