UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Algoritmos Avanzados

Laboratorio N°2

Alumnos: Cristhofer Parada
José Toro

Sección: A-1

Profesor(a): Cristian Sepúlveda

Contents

1. Introducción

A los estudiantes de la asignatura de Algoritmos Avanzados se les solicitó implementar en código, utilizando el lenguaje de programación C, 2 problemas. El primero consiste en encontrar la suma mayor de un grupo de números, el cual dichos números están acompañados de una ponderación, por lo que en el primer caso se debe encontrar la suma mayor de números sin superar la suma de ponderación total dada por el nombre del archivo. El segundo problema consiste en realizar, en pocos términos, un recorrido de grafo y encontrar el camino que sea menor.

2. Método

Como bien fue dicho en la introducción, para esta entrega el algoritmo utilizado será el algoritmo de “Greedy”, utilizando esta implementación, si bien es cierto no se apunta a encontrar precisamente la mejor solución, se intenta llegar a la mejor solución, sin tener que sacrificar tiempo de ejecución, por medio del uso de un criterio, criterio el cual está definido por el programador. Comparado con la entrega anterior, donde se hacían todas las soluciones por medio de fuerza bruta, y luego de tener todas las soluciones buscamos la que nos sirve, por medio de un criterio nos enfocamos solo en una solución intentando adivinar que sea la mejor.

Ante esto, “Greedy” presenta las siguientes características como:

- Reducción significativa del tiempo de ejecución.
- Utilizados frecuentemente en problemas de optimización.
- Se toma decisiones en función de un criterio propuesto.
- No garantiza que la solución sea la óptima.

2.1 Método algoritmo N°1

Para la resolución de este problema se utilizó como criterio lo siguiente:

$$ponderacion = \frac{v}{p}$$

Donde v corresponde al valor de la izquierda del archivo a leer y p corresponde al valor de la derecha del valor. Cabe recordar que el propósito de este problema es maximizar el valor de v sin sobrepasar el valor de $pMax$, dado por el nombre del archivo, al momento de sumar el valor de p cada vez que sumamos v. Entonces, la lógica detrás de esta implementación consiste en que al momento de leer el archivo y guardar los datos dentro de una *struct*, generamos una matriz de 3xN en donde las dos primeras columnas corresponderán a los valores de v y p respectivamente y la tercera columna corresponderá a la $ponderacion$ entre v y p. Una vez leído el archivo, procederemos a ordenar esta matriz que se generó de mayor a menor utilizando el método de ordenamiento de *Quicksort.*

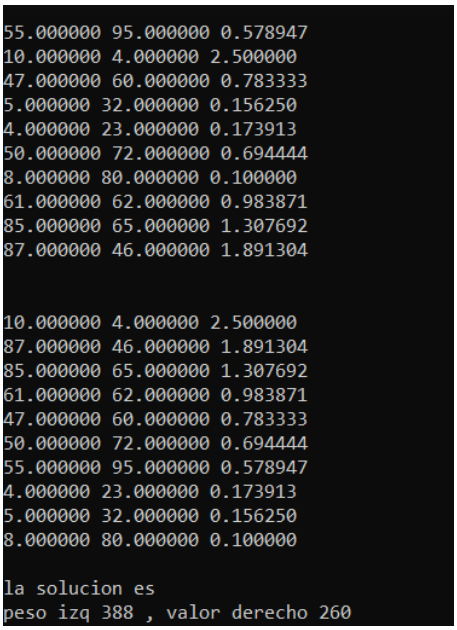

Figura 1: Matriz ordenada de mayor a menor a partir de su $ponderacion$.

Con esa matriz generada, entonces, buscamos seguir el camino desde el mayor al menor hasta que se llegue que p sea lo más alto posible sin sobrepasar el $pMax$ dado por el nombre del archivo de texto.

2.2 Método Algoritmo N°2

Para la implementación del algoritmo 2, debido a la naturaleza del problema, no se podía ordenar según un criterio, pero si es posible realizar una selección siguiendo un criterio, este criterio fue tomar el camino de menor peso dentro del grafo, es decir, se inicia en el nodo 0, se revisan todos los caminos que pueda tomar y se toma el de menor peso, luego desde ese nuevo nodo, se realiza la misma acción, para ello se utilizaron tres criterios:

- El primer criterio consta en que no se puede tomar un camino con peso negativo, ya que estos representan el nodo en el cual uno se encuentra.
- El segundo criterio consta en tomar el camino de menor peso.
- El tercer criterio consta en no tomar el camino de un nodo ya visitado anteriormente.

Con estos tres criterios si realizamos una pequeña traza al primer grafo se tendrá lo siguiente:

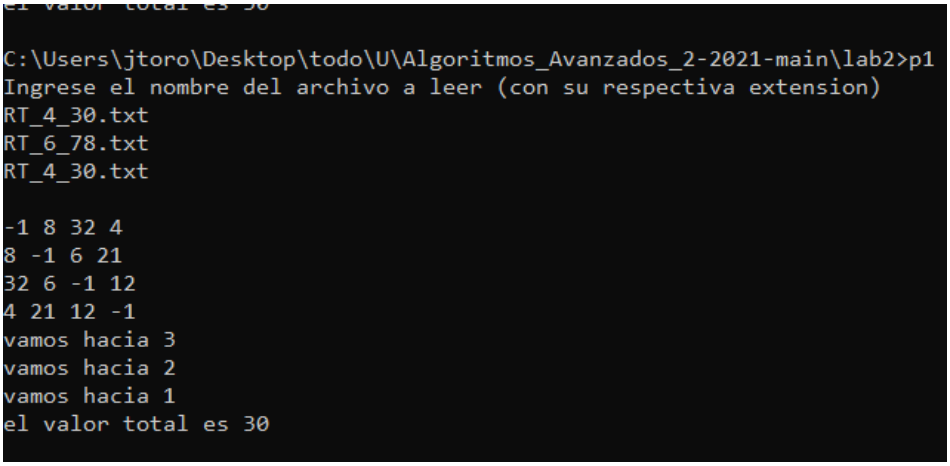

Figura 2: Grafo de algoritmo N°2 con el archivo “RT_4_30.txt”.

Iniciamos en el nodo 0, los cuales tiene los siguientes valores en la fila [-1 8 32 4], de estos valores el que cumple con los tres criterios es el número 4, el cual corresponde al nodo número 3.

El nodo 3 tiene [4 21 12 -1], como el valor 4 representa un nodo ya visitado, el único que cumple los tres criterios es el valor 12, representando al nodo 2.

El nodo 2 tiene [32 6 -1 12], de estos el que cumple los requisitos es el número 6 que es el nodo 1

El nodo 1 tiene [8 -1 6 21] de estos el menor es el valor 8 que representa el nodo 0, terminando así la búsqueda con el siguiente recorrido:

$$inicio\ (0) \rightarrow nodo3 \rightarrow nodo2 \rightarrow nodo1 \rightarrow nodo0\ (inicio)$$

Lo cual al sumar sus pesos da 4+12+6+8 resultando un peso total de 30, lo cual en este caso corresponde con el mejor resultado según los parámetros establecidos en el nombre.

3. Discusión de la solución

En el siguiente capítulo, se discutirá acerca de las soluciones encontradas en el archivo y su fidelidad con los resultados esperados.

3.1 Resultados Algoritmo N°1

Antes de comenzar a discutir sobre los resultados obtenidos en el algoritmo N°1, se mostrarán cuales fueron los archivos utilizados como lectura:

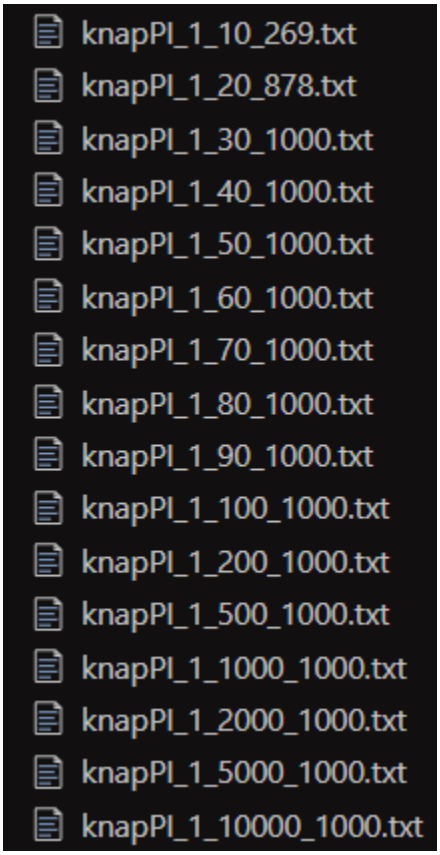

Figura 3: Archivos de entrada para el algoritmo N°1.

Los nombres de los archivos siguen el siguiente patrón $knapPI\_1\_N\_pMax.txt$ donde N corresponde al numero de filas de dicho archivo y $pMax$ corresponde a la ponderación máxima aceptable.

Al utilizar como archivo de entrada el llamado “knapPI_1_10_269.txt” se muestra por pantalla lo siguiente:

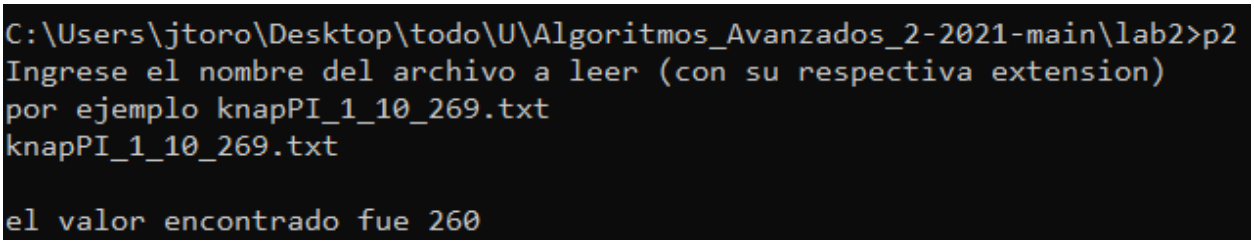

Figura 4: Resultado al ejecutar el algoritmo N°1 con el archivo “knapPI_1_10_269.txt”.

Ante esto, y considerando el valor esperando al que se debe llegar en este caso (295), podemos decir que no se llegó al valor esperado, pero está bastante cercano al que se esperaba y aun más, podemos ver que al ejecutar los siguientes archivos:

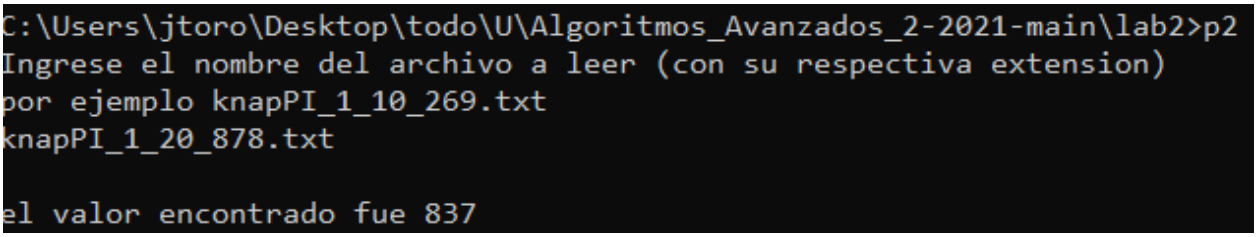

Figura 5: Resultado al ejecutar el algoritmo N°1 con el archivo “knapPI_1_20_878.txt”.

El resultado obtenido muestra que el mejor valor sin sobrepasarse de la ponderación es 837, lo cual se mantiene dentro de lo aceptable para el resultado esperado (1024).

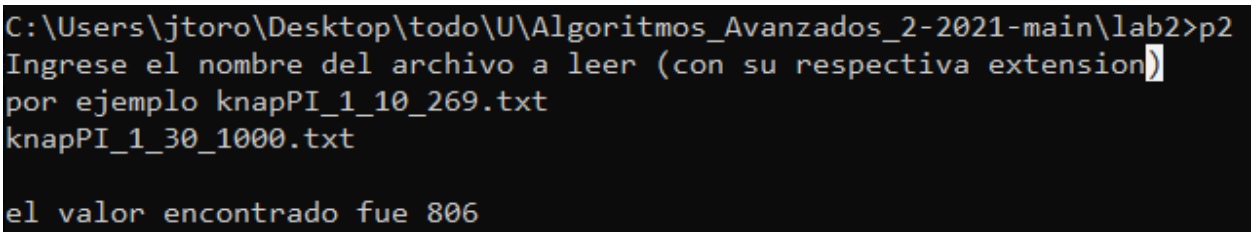

Figura 6: Resultado al ejecutar el programa con el archivo “knapPI_1_30_1000.txt”.

Se puede ver que, en este caso, existe un amplio margen entre la solución obtenida por el algoritmo en relación con los resultados que se esperaban (1500). Ante esto, se puede llegar a pensar que el método “Greedy” pierde su efectividad a medida que la cantidad de posibles soluciones crece.

3.2 Resultados Algoritmo N°2

A diferencia de la anterior entrega, al hacer un análisis de los resultados obtenidos por el algoritmo numero 2 utilizando greedy no encontramos tan buenos resultados como se esperaba, incluso cuando para el primer documento de prueba (RT_4_30.txt) se obtiene la mejor solución posible.

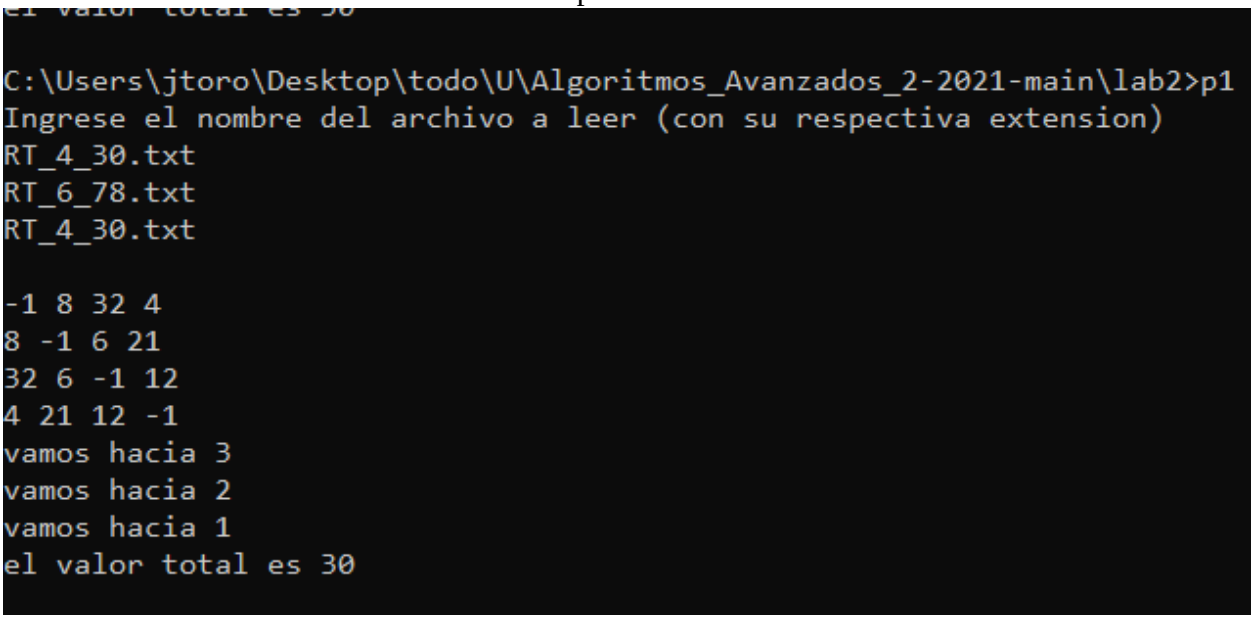

Figura 7: Resultados de la ejecución del algoritmo N°2 con el archivo “RT_4_30.txt”.

Para el segundo documento de prueba (RT_6_78.txt), los resultados obtenidos no son constantes, es decir, entrega distintas soluciones dependiendo de cuantas veces se ejecute, lo cual es completamente contradictorio al uso de “Greedy” debido a que este tipo de algoritmos genera una sola solución, la cual siempre debe seguir el mismo criterio, por ende, con la misma entrada debe entregar el mismo resultado, lo cual por razones que no logramos descifrar no fue así.

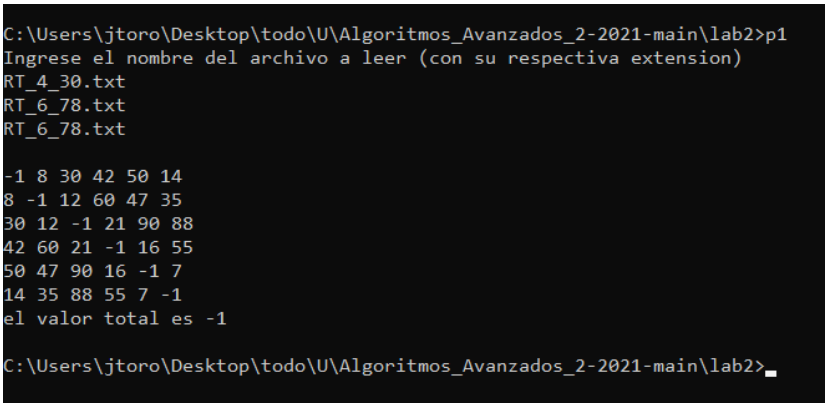

Figura 8: Resultado obtenido al ejecutar el algoritmo N°2 con el archivo “RT_6_78.txt”.

4. Conclusión

Al igual que en la anterior entrega, los problemas entre si no tienen mucho en común, sin embargos, ambos pueden ser resueltos con algoritmos muy similares, ya sea enumeración explicita o “Greedy”, si bien utilizando enumeración explicita obteníamos los resultados exactos, debíamos esperar mucho más tiempo para obtenerlos. En cambio, con “Greedy”, se obtiene una solución aproximada pero en un tiempo muchísimo menor, un gran ejemplo de esto es comparar el tiempo de ejecución del algoritmo 1 utilizando enumeración explicita y utilizando “Greedy”, si bien el algoritmo para el problema numero 2 utilizando “Greedy” entrega el mejor resultado para el primer archivo, esto no siempre será así, ya que dependerá de los criterios utilizados, esto queda en evidencia en el algoritmo numero 1 utilizando “Greedy”, donde el valor obtenido es 260, un poco alejado del mejor valor (295) obtenido con enumeración explicita, sin embargo, esta pérdida de precisión es compensada con creces con la reducción del tiempo de ejecución del algoritmo.