Como obtener la ruta menos costosa para llevar todos los recursos a la capital con el método de Goloso.

Alberto Rodríguez Zamorano
Alberto.rodriguez.z@usach.cl
https://github.com/Albertorz31

Algorithm design: greedy.

RESUMEN: El siguiente documento consiste en la resolución de un problema por medio de la implementación de Goloso. El problema que trata el documento es el poder encontrar la ruta de menor costo, para llevar todos los recursos de cada punto de abastecimiento hasta la capital. El documento trata el problema mediante la creación de un algoritmo que recibe de entrada un archivo, el cual contiene el número de puntos de abastecimientos adyacente a la capital, y las líneas siguientes muestra el costo de ir de un punto a otro (Todos los puntos son adyacentes con todos). Por esto se debe entregar un archivo de texto en el que muestre el mínimo costo de la ruta en recorrer todos los puntos, sin repetirse junto con la ruta que realiza. Para su solución, se crea un grafo en el cual cada nodo será un punto de abastecimiento, el nodo inicial es la capital y las aristas entre ellas sea el costo de ese camino. El criterio por utilizar es que siempre se irá por el camino de menor costo posible, esto quiere decir que no lo allá recorrido antes, realizará esto hasta recorrer todos los nodos y devolverse a la capital. El algoritmo es iterativo y posee una complejidad de O(n^2). Al ser la complejidad un polinomio se puede decir que el algoritmo es eficiente. Sin embargo, al utilizar el método Goloso no se garantiza obtener el caso óptimo, es decir, puede que no se obtenga la ruta óptima de costo mínimo. Se puede resolver de otra forma para obtener la ruta óptima, pero puede que la complejidad del algoritmo sea menos eficiente.

PALABRAS CLAVE: Lista, algoritmo, grafo, ruta, nodo, costo.

ABSTRACT: The following document consists in the resolution of a problem through the implementation of Greedy. The problem dealt with in the document is to find the lowest cost route, to take all the resources from each supply point to the capital. The document addresses the problem by creating an algorithm that receives input from a file, which contains the number of supply points adjacent to the capital, and the following lines show the cost of going from one point to another (All points they are adjacent to all). For this reason, a text file must be submitted in which it shows the minimum cost of the route in going through all the points, without repeating itself along with the route that it carries out. For its solution, a graph is created in which each node will be a point of supply, the initial node is the capital and the edges between them is the cost of that path. The criterion to be used is that you will always go down the path of least possible cost, this means that you have not traveled before, you will do this until you go through all the nodes and return to the capital. The algorithm is iterative and has a complexity of O (n ^ 2). Being the complexity a polynomial it can be said that the algorithm is efficient. However, when using the Greedy method, it is not guaranteed to obtain the optimal case, that is, the optimum minimum cost route may not be obtained. It can be resolved in another way to obtain the optimal route, but the complexity of the algorithm may be less efficient.

KEYWORDS: List, algorithm, graph, route, node, cost.

1 INTRODUCCIÓN

En la asignatura de Algoritmos Avanzados, la temática que se trata es la creación de algoritmos de poca complejidad para resolver problemas. El documento presente busca la resolución de un problema por medio de Goloso o Voraz, cuyo problema trata de buscar la ruta menor costosa al recorrer todos los puntos de abastecimiento y volver a la capital. Lo que se espera de la aplicación de Goloso es que sea poco compleja como solución, dejándola para posibles comparaciones en el futuro con otras formas de solución, de tal forma que se busque cual tiene un orden de complejidad menor.

2 DESCRIPCIÓN DEL PROBLEMA

El Reino *Clover* posee un encargado que debe llevar todos los recursos a la capital, estos se encuentran en distintos puntos de abastecimiento adyacentes a la capital, además el rey debe pagarle a este encargado por cada recorrido que haga. Dado esto el rey necesita una ruta que el encargado debe seguir para pasar por todos los puntos y que esta sea la ruta de menor costo posible. Para esto la entrada debe ser un archivo de texto que contenga el número de puntos de abastecimiento, y las líneas siguientes muestran el costo de ir de un punto a otro. (Agregar que ir desde la capital hasta cualquier punto, el costo es 1).

Entrada.in

```
4
1 2 5
1 3 7
1 4 8
2 3 10
2 4 2
3 4 9
```

Figura 1: ejemplo de entrada.

Y la salida debe ser un archivo de texto que contenga el costo de la ruta mínimo y el camino que realizó punto por punto hasta devolverse a la capital.

Salida.out

```
18
0-1-2-4-3-0
```

Figura 2: ejemplo de salida (el 0 es la capital).

Además, el algoritmo debe incluir las funcionalidades greedy() y printCurrent(). La función greedy() debe ser una función que genere el recorrido punto por punto que se debe hacer y al mismo tiempo encuentre la solución. La función printCurrent() debe ser una función que imprima la ruta actual que se tiene, es decir, los puntos que ya se han visitado y el costo total que se lleva actualmente en cada iteración:

```
printCurrent(...) {

        #ifdef DEBUG
        printf("enter para continuar...\n")
        while(getchar() != '\n');
        /*
        en esta parte debe escribir su código para imprimir
        lo que sea necesario para mostrar el estado actual
        del nodo.
        */
        #endif
    }
```

Figura 3: Parte del algoritmo printCurrent()

3 MARCO TEÓRICO

Para comprender los conceptos utilizados en el presente documento, se entiende lo siguiente: *Goloso*, en informática, consiste en un método de búsqueda en el cual elige la opción optima en cada paso local, con la esperanza de llegar a una solución general optima. Es considerado un algoritmo heurístico, ya que, es una técnica diseñada para revolver un problema más rápidamente cuando otros métodos son demasiado lentos, o para encontrar una

solución aproximada, que no necesariamente sea la óptima. La construcción del algoritmo es bastante más sencilla en diseñar, en comparación con otros métodos, además la comprobación de su funcionamiento es más sencilla, ya que, al pasar por un estado solo una vez sin retroceder (Por eso el nombre Goloso o Voraz, es como que se "come" ese camino y no lo vuelve a ver) a ese estado después, como en el caso de *backtracking.*

La estructura de datos utilizada en este laboratorio donde se aplicará Goloso es un grafo. Un grafo es un conjunto de nodos (tambien llamados vértices) y un conjunto de aristas que establecen las relaciones entre los nodos. Un grafo se puede definir como G=(V, A), siendo V el conjunto de vértices del grafo y A cuyos elementos son aristas, estas aristas puede ser dependiendo si el grafo es dirigido o no. Para este laboratorio se usará un grafo no dirigido, ya que, todos los puntos están unidos con todos, entonces puede haber aristas para ambas direcciones.

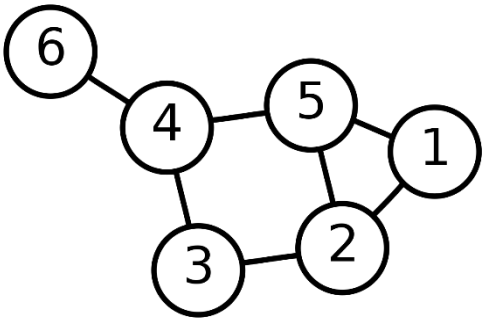


Figura 4: Ejemplo de grafo no dirigido.

Existen muchas formas de representar o implementar un grafo, la usada en este trabajo es una matriz de adyacencia.

Una matriz, en informática, es una estructura de datos que almacena conjuntos de variables, donde cada conjunto es representado con un índice (filas) y cada valor tiene asociado un índice (columnas). En este caso, se utiliza una matriz de adyacencia, que consiste en una matriz cuadrada que se asocia cada fila y cada columna a cada nodo del grafo, siendo los elementos de la matriz la relación entre los mismos. Esta relación puede representar el costo de ir de un vértice a otro, o simplemente se puede representar como 0 o 1 si es que existe una relación o no entre esos vértices.

M	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

Figura 5: Ejemplo de matriz de adyacencia.

4 DESCRIPCIÓN DE LA SOLUCIÓN

4.1 IDEA

Para resolver este problema por medio de Goloso, se tiene pensado el uso de listas enlazadas, grafos y matrices de adyacencia. En el archivo de entrada, la primera línea que representa el número de puntos de abastecimiento, estos se utilizarán para crear el tamaño del grafo, es decir cuántos nodos tendrá (se incluye tambien el nodo de la capital), los datos de las siguientes líneas son los costos de ir de un nodo a otro, para esto se guardan dentro de la matriz de adyacencia los costos de cada arista. Como es un grafo no dirigido en la posición opuesta de cada arista (es decir si el la posición (x,y), entonces la opuesta es (y,x)) se guardará el mismo costo de su opuesto. La capital será representada como el nodo 0 y todas sus aristas tendrán valor de 1. Después de crear el grafo y de llenar la matriz de adyacencia se lleva a cabo el método Goloso.

Para ejecutar Goloso, primero se debe marcar los nodos, es decir, indicar cuales nodos han sido recorridos y cuáles no. Inicialmente se marca a todos como no recorridos, excepto el

nodo inicial el cual es la capital. Después mediante una función random, el algoritmo escogerá el primer camino al cual irse desde la capital, el primer camino es al azar, ya que, la distancia de todos los nodos con el inicial es 1, así que no influye su elección. Ya elegido un camino marcará este nodo destino como recorrido, y pasará a ser el nodo inicial. Entonces ahí comenzará a aplicar el criterio de ver que nodos son adyacentes a este (que no esté marcado) y elegirá el que tenga el menor costo (si hay 2 con el mismo costo mínimo, elegirá al primero que le salga), entonces este pasará a ser el nuevo nodo inicial y realizará lo mismo mencionado anteriormente, hasta que este todos los nodos marcados y entonces volverá a la capital (sumando el costo de este tambien).

La ruta final se irá almacenando en una lista, a la cual se le van sumando los nodos uno por uno, hasta tenerlos todos. Siempre el ultimo nodo en agregar a la lista será el de la capital, indicando la vuelta del encargado a la capital con los recursos.

4.2 REPRESENTACIÓN Y ESTRUCTURA

Los datos del archivo de texto de la entrada inicialmente son representados como strings, pero luego son transformados a enteros para ser trabajados. Se van a tener 3 estructuras distintas. La primera es la estructura Nodo la cual va a poseer 2 enteros, uno muestra el número de ese nodo, y el otro indica si ese nodo fue marcado o no (puede ser 0 o 1). Otra estructura es Grafo la cual va a poseer 3 tipos de datos, una es vértice, en el cual se incluyen todos los nodos de este (la estructura de este es la anteriormente mencionada), otra es un entero indicando la cantidad de nodos que tiene y otra es una matriz de adyacencia donde se guardan los costos entre los nodos.

La última estructura es una lista enlazada, la cual tiene 3 tipos de datos, los cuales son 2 enteros, los cuales son nodo y costo que representa el costo de ir a ese nodo del nodo anterior (Si el anterior es la capital el costo seria 1), y un puntero al nodo siguiente. Esta será la ruta final que se entrega al rey, se van sumando los costos de cada nodo de la lista para obtener el costo final, junto con el orden en que los nodos fueron recorridos.

4.3 PSEUDOCÓDIGO

Para el tipo de dato lista, posee alguna de sus funcionalidades básicas de Crear_lista(lista, entero, entero), obtener(lista, entero) y insertar(lista, lista, entero). La lectura del archivo se hace en la función principal, la cual también crea el grafo y le va insertando los nodos junto con sus aristas y al final entrega la lista final con la ruta. Pero la función más importante es la que realiza el método de Goloso, la cual es la que realiza el recorrido del grafo, usando el criterio de ir por el camino mínimo.

Para el pseudocódigo se mostrará cómo se crea el grafo, junto con cómo se insertan los nodos y como se crea la matriz de adyacencia. También se mostrará cómo funciona de forma general el método Goloso, como irá marcando los nodos recorridos, mientras va recorriendo el grafo.

```
FuncionPrincipal:
    grafo G
    leerArchivo(archivoEntrada)
    G.nodos=cantidadNodos #lee del archivo
    Para i desde 0 hasta G.nodos:
       G.vertice=i
       #nodos no marcados
       G.vertice.marcardo=0

    #Se crea matriz de adjatencia
    Para i desde 0 hasta G.nodos
     Para j desde 0 hasta G.nodos:
         #Si i o j es 0 esta en la capital
         Si(i=0 o j=0):
            G.adj[i][j]=1 #Costo de la ruta
         Sino:
            #nodoO y D sacados del archivo
            G.adj[nodoO][nodoD]= costo
    listaFinal= Greedy(G)
    Devolver listaFinal
```

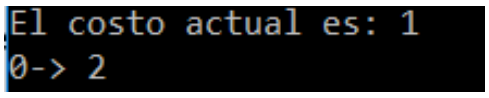
```
greedy(grafo G): lista
  listaFinal:  lista
  #Se marca el nodo de la capital
  nodoInicial.marcado = 1
  agregarLista(lista,nodoInicial)

  newNodo = funcionRandom(1 a G.nodos-1)
  newNodo = nodoInicial
  agregarLista(lista,newNodo)
  costoRuta = costoRuta + costo(newNodo)
  newNodo.marcado=1

  #Se aplica Goloso
  Mientras(nodosMarcados< G.nodos):
   Para j desde 0 hasta G.nodos-1:
     Si(G.adj[newNodo][j] y G.newNodo!=1):
       Si(G.adj[newNodo][j]< minimoArista):
         minimo= G.adj[newNodo][j]

     nodoInicial = newNodo
     agregarLista(lista,nodoInicial)
     nodosMarcado++

  devolver ListaFinal
```

Se quedará en el bucle del mientras, hasta que haya marcado todos los nodos, lo sabrá cuando los nodos marcados sean iguales a la cantidad de nodos en el grafo.

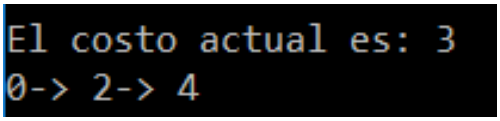
4.4 TRAZA

Para la traza de dará un caso ejemplo, en cómo se recorre el grafo nodo por nodo, y el costo que se va acumulando a medida que más recorre (en cual se mostrará en la siguiente imagen). En esta sección mostraremos el funcionamiento de la función printCurrent, la cual va mostrando como se crea la ruta, junto con el costo de esta.

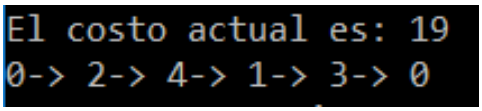
- Ruta inicial: Se tiene inicialmente el nodo 0 que representa la capital y a cuál nodo se fue aleatoriamente, junto con el costo de irse a ella.



- Primer caso: después ya se aplica el criterio de Goloso, el cuál selecciona el nodo que tiene camino mínimo de los que no han sido marcados.



- Caso final: el algoritmo puede iterar muchas veces, hasta recorrer todos los nodos del grafo y se devuelve a la capital junto con el costo final, como muestra en la imagen.



4.5 ORDEN DE COMPLEJIDAD

Anteriormente en el resumen se explicó que el algoritmo es iterativo, ya que, utiliza el método de Goloso, entonces es un algoritmo polinómico y su tiempo es:

$$T(n) = c \cdot n^2 + c \cdot n + c$$

Entonces si luego se calcula la complejidad para el peor caso, se obtiene una complejidad de orden:

$$\sim O(n^2)$$

Para todas las funciones básicas de la lista enlazada tiene complejidad de orden O(n), debido a que n corresponde a la cantidad de veces que se recorre la lista.

Para el algoritmo de printCurrent su complejidad tambien es O(n) debido a que imprime todos los elementos de una lista.

5 ANÁLISIS DE LA SOLUCIÓN

5.1 ANÁLISIS DE IMPLEMENTACIÓN

Se puede comprobar que el algoritmo se detiene, entregando el resultado correcto, si el archivo ingresado es correcto con los requerimientos, por ejemplo, los nodos deben ser enteros, no pueden ser strings. Esto quiere decir que el algoritmo se detiene cuando entrega el resultado operando con los parámetros correctos, que sería un archivo de texto que contenga los datos necesarios.

El algoritmo es eficiente ya que la complejidad es un polinomio, al ser un algoritmo iterativo no se generan estados pendientes o requiere más memoria. Puede que el algoritmo fuera más eficiente si se conocieran más funcionalidades disponibles que reduzcan el tiempo y complejidad.

El problema de usar el método Goloso es que no entrega la solución óptima, al ser un algoritmo heurístico, se prioriza el tiempo de ejecución y la eficiencia ante lo exacto de la solución. Puede que exista una mejora en su exactitud en cambiar la primera parte del algoritmo, en vez de seleccionar al azar el primer camino desde la capital a un nodo, se puede crear un nuevo criterio que si se tiene costos mínimos iguales entre las opciones posibles, se verifica para cada uno de estos si el costo mínimo de estos con otro nodo adyacente, es el menor de todos de los demás mínimos, entonces escogerá ese camino, esa podría ser una opción para mejorar la exactitud del método, mejoraría porque, al seleccionar un camino inicial al azar puede cambiar notoriamente el resultado final del recorrido.

Otro método que también hubiese funcionado es utilizar Backtracking, ya que, al ir generando estados posibles, va a retroceder hasta encontrar la solución óptima de la ruta, aunque puede que la complejidad del algoritmo sea peor y puede empeorar la eficiencia.

Se realizaron varias pruebas en el programa, se utilizó un grafo con 4 nodos (sin contar la capital) y un grado con 6 nodos (sin contar la capital), en el cual para el primer caso, el algoritmo se demoró 0,4 milisegundos en entregar el archivo de salida. Mientras que en el segundo caso de demoró aproximadamente 0,6 milisegundos. Con esto se podría decir que por cada nodo que tenga el grafo se puede demorar aproximadamente 0,1 milisegundo

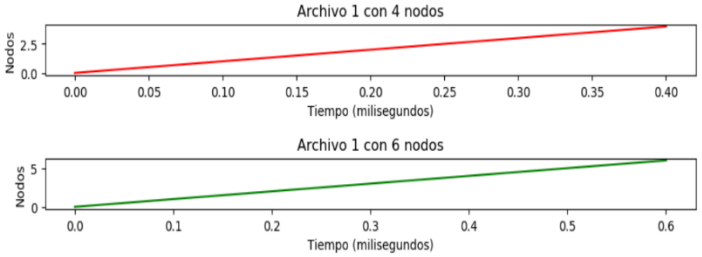


Figura 6: Gráfico de nodo vs tiempo.

5.2 EJECUCIÓN

El algoritmo fue implementado en el lenguaje de programación C. Además, se trabajó en el sistema operativo Windows 10, por lo tanto, para la compilación del programa se debe agregar en la consola lo siguiente:

```
Avanzados\Lab3>gcc goloso.c -o ejecutable.exe
```

En el caso de que se quiere compilar implementando el DEBUG, se agrega lo siguiente:

```
Avanzados\Lab3>gcc goloso.c -DDEBUG -o ejecutable.exe
```

Por ultimo, para ambos casos de compilación, la ejecución es la siguiente, en la cual se deben ingresar el archivo de entrada y el nombre del archivo de salida:

```
Avanzados\Lab2>ejecutable.exe entrada.in solucion.txt
```

6 CONCLUSIONES

Tras el desarrollo y análisis del algoritmo, se puede extraer que fue fácil de implementar el código, y que se logra de buena forma el método de Goloso. Sin embargo, siendo Goloso un algoritmo heurístico, puede no entregar necesariamente la solución óptima al problema, además como se explicó en el análisis al seleccionar inicialmente un camino al azar, cambiar totalmente la solución final del algoritmo. Por lo tanto se puede concluir que en temas de eficiencia el algoritmo es bueno, pero si se quiere priorizar la exactitud del algoritmo, o se realiza la mejora mencionada anteriormente o se busca otro método para resolverlo.
En conclusión, se puede decir que el método de Goloso es recomendable para tipos de problemas donde se priorice la rapidez del programa antes de la exactitud de este. Puede ser utilizado cuando se tenga algoritmos lentos para resolver un problema y las soluciones aproximada no necesariamente optimas sean la prioridad.

7 REFERENCIAS

Enlaces de donde se ha basado para el desarrollo del documento. El formato debe ser

- “Algoritmos: Teoría y aplicaciones” , Mónica Villanueva ,2002 , disponible en: http://www.udesantiagovirtual.cl/moodle2/pluginfile.php?file=%2F117134%2Fmod_resource%2Fcontent%2F1%2FAlgoritmos-apuntes.pdf