Laboratorio 1: Mayor cantidad de sucursales con Fuerza Bruta

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

Algorithm design: Explicit enumeration.

RESUMEN: El siguiente documento consiste en la resolución de un problema por medio de la implementación de Fuerza Bruta. El problema que trata de que una empresa quiere poner la mayor cantidad de sucursales en la ciudad, cumpliendo las condiciones de que no debe existir más de una sucursal en una conexión entre cables horizontales, verticales y diagonales. El documento trata el problema mediante la creación de un algoritmo que recibe de entrada un archivo, el cual contiene el ancho y largo de la matriz, las líneas siguientes y las posiciones de las sucursales que deben estar si o sí. Este debe entregar un archivo de texto en el que muestre la cantidad de sucursales, una lista con las coordenadas de las sucursales y una representación grafica del mapa de la ciudad. Para su solución, se tomó como la idea la creación de una lista enlazada que guarde las posiciones de todas las sucursales de la matriz final, esta lista se irá modificando a medida que el algoritmo encuentre un camino mejor que el actual. El algoritmo al ser recursivo y se descompone en un subproblema y el tiempo para definir el subproblema es de O(n^2), el algoritmo tiene una complejidad de O(n^3). Al ser la complejidad un polinomio se puede decir que el algoritmo es eficiente, por lo tanto, la idea es viable y funcional. Sin embargo, no se sabe si puede que haya otra forma de resolver el problema con tal que sea más efectiva.

PALABRAS CLAVE: Lista, algoritmo, sucursales, posiciones, matriz.

ABSTRACT: The following document consists in the resolution of a problem through the implementation of Brute Force. The problem is that a company wants to put as many branches in the city, fulfilling the conditions that there should be no more than one branch in a connection between horizontal, vertical and diagonal cables. The document addresses the problem by creating an algorithm that receives input from a file, which contains the width and length of the matrix, the following lines and the positions of the branches that should be yes or yes. It must submit a text file showing the number of branches, a list with the coordinates of the branches and a graphic representation of the city map. For its solution, it was taken as the idea to create a linked list that stores the positions of all the branches of the final matrix, this list will be modified as the algorithm finds a better way than the current one. The algorithm being recursive and decomposing into a subproblem and the time to define the subproblem is O (n ^ 2), the algorithm has a complexity of O (n ^ 3). Being the complexity a polynomial we can say that the algorithm is efficient, therefore, the idea is viable and functional. However, it is not known if there may be another way to solve the problem as long as it is more effective.

KEYWORDS: List, algorithm, Branch offices, positions, matrix.

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 la Fuerza Bruta, cuyo problema trata de buscar la

mayor cantidad de sucursales posible que se puede obtener en una matriz de un determinado tamaño que cumpla con las restricciones iniciales (no debe existir conexión entre los cables verticales, horizontales y diagonales). Lo que se espera de la aplicación de Fuerza Bruta 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

La empresa de Courier internacional *kaioken express* tiene una variedad de sucursales en diferentes partes del mundo, por eso llega a la ciudad de Novigrad a ingresar la mayor cantidad de sucursales en la ciudad, pero por las conexiones de la ciudad no pueden haber mas de dos sucursales en una conexión de cable, las conexiones son verticales, horizontales y diagonales. Para esto la entrada debe ser un archivo de texto que contenga el ancho y largo de la matriz, la cantidad de líneas siguientes y las posiciones de las sucursales que deben estar sí o sí.

Entrada1.in

```
10 10
2
1 1
5 8
```

Figura 1: ejemplo de entrada.

Y la salida debe ser un archivo de texto que contenga la cantidad de sucursales, una lista con las coordenadas de las sucursales y una representación gráfica del mapa de la ciudad.

Salida1.out

```
9
0-4 || 1-1 || 2-3 || 3-0 || 4-2 || 5-8 || 7-9 || 8-7 || 9-5 ||
_____x_____
_x________
___x______
x_________
__x_______
________x_
__________
_________x
_______x__
_____x____
```

Figura 2: ejemplo de salida.

Además, el algoritmo debe incluir las funcionalidades bruteForce() y printCurrent(). La función bruteForce() debe ser una función que genera todas las combinaciones y al mismo tiempo encuentre la solución. La función printCurrent() debe ser una función que imprima el estado actual de un nodo, es decir, las sucursales distribuidas en el mapa en ese instante. Para esta función, se tiene parte de su algoritmo realizado:

```
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: *Fuerza Bruta*, en informática, consiste en la generación de todas las posibles soluciones para un problema y luego busca la mejor solución que satisfaga el problema.

Una *Lista Enlazada* es una estructura de datos fundamental que puede incorporar otras estructuras de datos, la cual consiste en una secuencia de nodos, donde cada uno puede poseer datos arbitrarios y a su vez tiene

punteros (enlaces) que puede ir hacia un nodo anterior y hacia un nodo posterior. Estas listas permiten la inserción, eliminación, búsqueda y obtención de datos. Una representación de una lista enlazada es la siguiente:

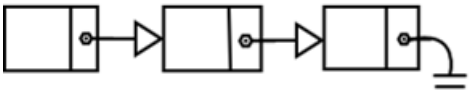


Figura 4: Ejemplo de lista enlazada.

4 DESCRIPCIÓN DE LA SOLUCIÓN

4.1 IDEA

Para resolver este problema se piensa en la utilización de la estructura de listas enlazadas y de matrices NxM. Primero se deben considerar las posiciones donde deben ir sucursales si o si (si es que las contiene), después aplicaremos el método fuerza bruta para obtener todos los posibles lugares donde pueden ir sucursales cumpliendo con las condiciones iniciales, lo explicado sobre las condiciones verticales, horizontales y diagonales. Como es una matriz se tiene los ejes i y j. Entonces se deben ver todas las posibilidades que cumplan con la posición de la sucursal[i][j], entonces no deben haber sucursales en las posiciones donde el i y el j sean iguales a la posición indicada y tampoco que sea diagonal, es decir, donde i y j de sucursal sean igual a:

- i – n y j - n
- i + n y j + n
- i + n y j – n
- i - n y j +n

Donde n es un número desde 0 hasta el largo de la matriz (vertical o horizontal). Mientras se aplica fuerza bruta para cada caso se va guardando en cada nodo de la lista enlazada la posición de cada una de las sucursales que se tiene en cada caso, obteniendo un largo determinado (cantidad de sucursales), si se encuentra un caso donde el largo de la lista es mayor actual, la nueva lista máxima será la del largo mayor.

4.2 REPRESENTACIÓN Y ESTRCUTURA

Los datos del archivo de texto de la entrada inicialmente son representados como strings, pero luego son transformados a enteros para se trabajados. Las listas enlazadas van a poseer 3 tipos de datos para almacenar; los primeros son de tipo entero llamados *fila* y *columna* los cuales contiene la posición de una sucursal y otra llamada *siguiente* para ver el siguiente nodo en la lista. Cuando se crean las combinaciones, donde cada combinación en una lista enlazada con las posiciones de cada sucursal, esta lista se reemplaza si se encuentra que la lista de la combinación actual tiene mayor largo que la máxima. Cuando se tiene la lista definitiva, se calcular el largo de la lista para ingresarlo en el archivo final, junto con las posiciones de cada nodo y al final una matriz que muestras las coordenadas de las sucursales de la lista final.

4.3 PSEUDOCÓDIGO

Para el tipo de dato lista, posee alguna de sus funcionalidades básicas de Crear_lista(lista, entero, entero), obtener(lista, entero), insertar(lista, lista, entero), copiar_lista(lista) y Largo(lista). La lectura del archivo se hace en la función principal, incluye la función de crear matriz(entero, entero, lista), la cual entrega la lista final. Pero la función principal es la que realiza el método Fuerza Bruta, la cual en el código se divide en dos partes, ambas hacer fuerza bruta pero la primera entrega el resultado y llama a la segunda. Mientras la segunda realiza la recursión para crear la lista final.

Para el pseudocódigo se mostrarán ambas partes, pero de una forma más general y reducida que el código original:

```
BruteForce(filas, columnas, inicial: Lista):
   Lista aux
   Lista final
   aux=inicial

   Para i desde 0 hasta filas-1:
    Para j desde 0 hasta columnas-1:
     Mientras(aux!=NULO):
       Si(i y j no cumplen con condiciones):
          No se considera
       aux= aux.siguiente

     Si(i y j cumplen con condiciones):
       final= comenzarArbol(i,j,aux)

   matriz= crearMatriz(final)
   devolver matriz
```

comenzarArbol es la parte dos de la fuerza bruta;

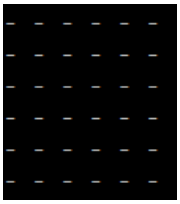
```
comenzarArbol(inicio: Lista,i,j,fila,columna):
  Lista nuevoNodo
  Lista aux
  aux=inicio
  nuevoNodo.fila=i
  nuevoNodo.columna=j
  nuevoNodo.siguiente=NULO
  INSERTAR(inicio,nuevoNodo,ultima posición)

  Para i desde 0 hasta filas-1:
    Para j desde 0 hasta columnas-1:
     Mientras(aux!=NULO):
       Si(i y j no cumplen con condiciones):
          No se considera
       aux= aux.siguiente

     Si(i y j cumplen con condiciones):
       final= comenzarArbol(i,j,aux)

  largoMaximo= largo(final)
  Si(largo(inicio) > largoMaximo):
    largoMaximo= largo(inicio)
    final = inicio

  Devolver final
```

La ultima parte verifica si el largo de la lista actual el superior a la lista máxima, si es mayor se reemplaza la lista máxima que sería lista *final*.

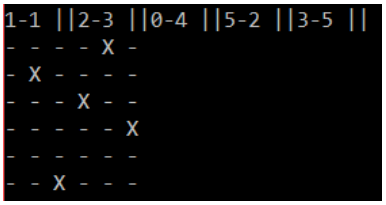
4.4 TRAZA

Para la traza de dará un caso ejemplo, en el cual se ingresa una matriz de 6x6 y tiene dos sucursales iniciales en las posiciones [1][1] y [2][3]. En esta sección mostraremos el funcionamiento de la función printCurrent, la cual va mostrando la matriz final que se tiene mientras se va ejecutando cada rama del árbol.

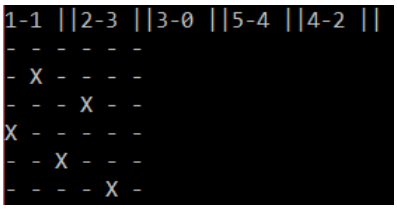
- Matriz inicial: primera mente no hay nada porque no se completado la primera rama del árbol, es decir, desde la raíz hasta una hoja.



- Primer caso: después se obtiene el primer mejor caso que obtiene el algoritmo.



- Segundo caso: el algoritmo puede iterar muchas veces, hasta encontrar uno mejor al anterior. En este caso no es mejor pero al ser de la mismo largo lo reemplaza. No encuentra uno mas largo después, así que se tiene uno final.



4.5 ORDEN DE COMPLEJIDAD

Anteriormente en el resumen se explicó que el algoritmo es recursivo, ya que, utiliza el método de fuerza bruta se descompone en un subproblema de tamaño n-1, el tiempo para definir el subproblema es de n ^2, entonces el tiempo es de:

$$T(n) = T(n-1) + O(n^2)$$

Entonces si luego se utiliza la fórmula para algoritmos recursivos por sustracción. El algoritmo finalmente tiene un orden de :

$$\sim O(n^3)$$

Para todas las funciones básicas de la listas enlazadas 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 si el archivo ingresado es correcto con los requerimientos. Esto quiere decir que el algoritmo se detiene cuando entrega el resultado operando con los parámetros correctos, que seria un archivo de texto que contenga los datos necesarios. Por el momento, se puede considerar como un algoritmo eficiente pues entrega lo que se desea en un tiempo creciente como un polinomio. Puede que el algoritmo fuera más eficiente si se conocieran más funcionalidades disponibles que reduzcan el tiempo y complejidad, como también otros métodos para elaborar algoritmos. Otro método que tambien hubiese funcionado es trabajando con pilas y colas, debido a que algunos casos, el agregar elementos pueda reducirse de O(n) a O(1), lo que podría reducir la complejidad de todo el algoritmo.

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\Lab1>gcc fuerzaBruta.c -o ejecutable
```

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

```
Avanzados\Lab1>gcc fuerzaBruta.c -DDEBUG -o ejecutable
```

Por ultimo, para ambos casos de compilación, la ejecución es la siguiente:

```
Avanzados\Lab1>ejecutable
```

6 CONCLUSIONES

Tras el desarrollo y análisis del algoritmo, se puede extraer que tiene una buena implementación por medio de Fuerza Bruta, consiguiendo el resultado esperado. Sin embargo, siendo Fuerza Bruta un método donde la generación de posibles soluciones no se basa en restricciones para filtrar esas mismas, en este algoritmo recorre las mismas posiciones de la matriz muchas veces, sin saber que esas estaban restringidas, esto puede causar retrasos en la entrega de una solución, por lo que se deben buscar mejores métodos para el desarrollo de algoritmos para resolver este mismo problema de forma mas eficiente. Al ejecutar, el archivo de entrada no debe tener líneas vacías ni espacios sobrantes.

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