Universidad de Santiago de Chile
Departamento ingeniería informática
Ingeniería civil informática
Algoritmos Avanzados
Laboratorio

2-2019

Aplicación del método Backtracking para la problemática de la empresa Kaioken.

Felipe Villalobos Padilla
e-mail: felipe.villalobos.p@usach.cl

Application of the Backtracking method for the Kaioken company problem.

RESUMEN: *En el siguiente documento se analizará el laboratorio n°1 de la asignatura de algoritmos avanzados donde se tocará la problemática de la empresa de courier internacional kaioken express para expandirse a la ciudad de Novigrad instalando la mayor cantidad de sucursales en dicha ciudad, este problema se tratara utilizando backtracking dada ciertas restricciones a la hora de poner las sucursales. Dada la naturaleza del problema, esta es una solución es eficiente dado el* $O(n^4)$.

PALABRAS CLAVE: Backtracking, sucursales, matriz dinámica, NP.

1 INTRODUCCIÓN

En el presente laboratorio del curso Algoritmos Avanzados, se abordará la problemática de la empresa de courier internacional kaioken express para expandirse a la ciudad de Novigrad y aprender de la complejidad que se puede generar al dar una solución a esto y además resolver esto mediante la implementación de backtracking.

Este informe tiene por objetivo explicar en profundidad la solución propuesta al problema planteado por dicha empresa. El documento deberá esclarecer toda duda acerca de la solución propuesta y su funcionamiento, haciendo énfasis en el método de resolución.

2 DESCRIPCIÓN DEL PROBLEMA

El problema que se plantea en el presente laboratorio corresponde a una situación que acompleja a la empresa internacional kaioken express que busca expandirse a la ciudad de Novigrad, la cual posee una distribución de calles de la siguiente forma:

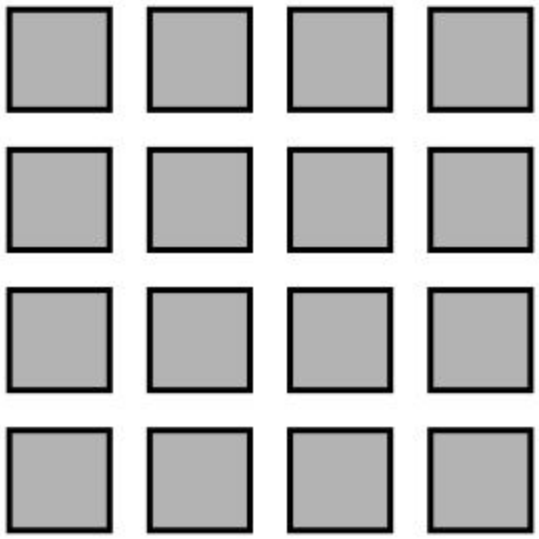

Figura 1: Distribución de calles y cuadras.

Para realizar esta acción busca instalar la mayor cantidad de sucursales posibles en la ciudad, pero al hacer esto conlleva un problema con el recurso del internet y como lamentablemente la conexión que posee la

ciudad es aún precaria y las sucursales consumen una gran cantidad de intenet, 2 o más sucursales no podrán estar conectadas al mismo cable de alimentación de internet. Estos cables de alimentación están distribuidos de la siguiente forma por la ciudad:

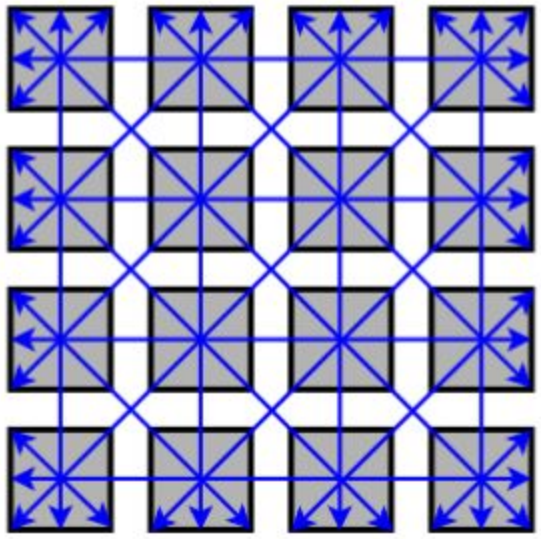

Figura 2: Red de cables de internet entre cuadras.

Considerando lo descrito con anterioridad y la distribución del cableado, las sucursales no podrán estar en la misma fila, columna y diagonales. Considerando que el problema tiene una gran similitud con el problema de las n reinas, con una variante en cuanto a la dimensiones del tablero, ya que este originalmente está planteado para un tablero de nxn, en cambio la ciudad de Novigrad puede tener una distribución nxm como también nxn. Considerando esta similitud se puede esperar que este problema sea tipo NP, por lo tanto no se espera una solución con tiempo polinomial.

Dado el problema mencionado se requiere diseñar un programa en C que dada la forma similar de Novigrad entregue una forma de colocar la mayor cantidad de sucursales en la ciudad.

3 DESCRIPCIÓN DEL MÉTODO

El algoritmo utilizado para la resolución del problema descrito en el capítulo anterior se conoce como volver atrás o Backtracking, es una estrategia de resolución de problemas con alguna restricción, donde el orden de la solución no importa. Se dice que este es un algoritmo de búsqueda en profundidad, debido a que en su funcionamiento durante la búsqueda de la solución avanza lo máximo posible siempre que se cumplan las restricciones, si se encuentra con una alternativa incorrecta, la búsqueda retrocede hasta el paso anterior y toma la siguiente alternativa.

4 DESCRIPCIÓN DE LA SOLUCIÓN

La solución corresponde a un programa diseñado en lenguaje C, el cual consta con diferentes funciones algunas ya implementadas tales como malloc, free, fopen, fclose, etc. y otras implementadas para este programa.
Dado el método descrito con anterioridad se deben considerar las restricciones que se deben aplicar, para que cuando estas se cumplan se active el retroceso e ir en búsqueda de otra solución. Estas restricciones son:

- La sucursal no debe estar en la misma fila que otras sucursales.
- La sucursal no debe estar en la misma columna que otras sucursales.
- La sucursal no debe estar en las misma diagonales que otras sucursales.

Dada estas restricciones, también se debe considerar que el máximo posible de sucursales que se pueden colocar en la ciudad dependerá del tamaño mínimo que puede existir entre el alto y el ancho (min{alto,ancho}), donde para n>3 existirá al

menos una solución donde se pueden colocar este máximo de soluciones, siempre y cuando no se coloquen sucursales de forma obligatoria por medio del archivo de entrada, en dicho caso, no se puede asegurar que se pueden posicionar la máxima cantidad de sucursales descritas con anterioridad. Dado todo esto, una condición de salida ocurría cuando la cantidad de máxima posible de sucursales estén puestas en la matriz, en caso contrario seguirá operando el backtracking.
Según lo descrito con anterioridad un pseudocódigo es:

```
void<- backtracking()

si las sucursales colocadas != sucursales máximas hacer:
      para i->0 hasta n:
            para j->0 hasta m
                  si en sucursal en i,j cumple las restriccion
            sucursales += {sucursal}
      sí cant sucursales > sucMaximas

copiar(sucursales,sucMaximas)
      fin si
            backtracking();
Sino
      copiar(sucursales,sucMaximas)
      salir
Fin si
```

4.1 REPRESENTACIÓN Y ESTRUCTURA

Considerando que los datos que ingresan al programa serán mediante un archivo ingresado por argumento, y la estructura que contiene, se utilizan matrices dinámicas para almacenar dicha información.Las matrices dinámicas son estructuras utilizadas para almacenar información generadas durante ejecución, a diferencia de una matriz estática la cual es generada durante compilación.
Se utilizaran 2 de estas matrices dinámicas:

- La primera de estas será utilizada para realizar todas las operaciones de movimientos en ella en la operación de backtracking, tales como el posicionamiento de sucursales y el retroceso en el a la hora de operar.
- La segunda matriz irá almacenando la representación de la ciudad con mayor cantidad de sucursales durante el proceso de backtracking, esta será modificada cada vez que se encuentre un tablero con más sucursales puestas en el, de los que hubo con anterioridad.

4.2 ORDEN DE COMPLEJIDAD

Tomando en cuenta las siguientes funciones utilizadas en la solución son, considerando el modo debug activado:

- guardar: $T(n^2)$ -> $O(n^2)$
- comprobar: $T(n^2+n)$ -> $O(n^2)$
- backtracking:

$$T(n) = \begin{cases} C & n=0 \\ n^4 + T(n-1) & n>0 \end{cases}$$

->de donde $O((n^4)!)$

- printCurrent: $T(n^2+C)$ -> $O(n^2)$

Dado el caso que el modo debug no esté activado la función backtracking cambia a:

- backtracking:

$$T(n) = \begin{cases} C & n=0 \\ n^2 + T(n-1) & n>0 \end{cases}$$

-> de donde $O((n^2)!)$

Considerando las operaciones y llamados de funciones el T(n) de la solución con el modo debug activado es:

$$T(n) = n + n * m + (n^4)!$$

En donde el n representa el ancho de la matriz y en donde m representa el alto de la matriz.

De dicha función de T(n) de la solución con modo debug activado se obtiene que el orden de complejidad es: $O((n^4)!)$.

En caso de que el modo debug no esté activado cambia a:

$$T(n) = n + n * m + (n^2)!$$

De dicha función de T(n) de la solución sin el modo debug activado se obtiene que el orden de complejidad es: $O((n^2)!)$.

5 ANÁLISIS DE LA SOLUCIÓN

Probando la solución con distintos archivos de diferentes tamaños utilizando la función clock() de la biblioteca time.h para medir el tiempo de ejecución de estos ensayos, se puede apreciar en el siguiente gráfico:

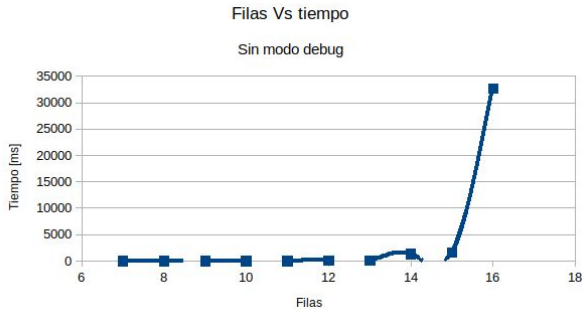

Figura 3: Gráfico Tamaño vs Tiempo[ms]

El cual tiene una apariencia similar a la función de la Figura 4.

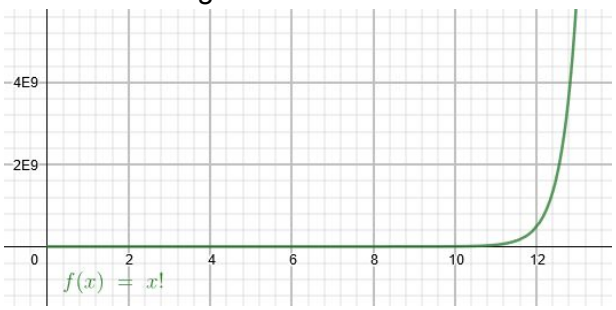

Figura 4: Gráfico de f(x)= $(x)!$.

Cabe señalar que estos tiempos dependen del equipo donde se realicen las pruebas.

Considerando que el peor de los casos se tomará una forma $(n^2)!$ para el modo sin debug, pero esto no sucede de forma constante, por lo que en promedio se nota una curva menos ceñida, haciendo que este tome una forma cercana a $(n)!$, esto debido a que a medida que aumenta el orden de la matriz, la cantidad de soluciones posibles de encontrar también aumenta en gran medida pero se sigue haciendo una gran cantidad de operaciones para llegar a ella.
A medida que el tamaño de la entrada crezca, el tiempo estimado para la solución crecerá en forma $(n)!$ según lo estimado, el algoritmo seguirá funcionando, pero para entradas demasiada grandes no será conveniente aplicar esta solución debido a su alto costo en tiempo.

5.1 ANÁLISIS DE IMPLEMENTACIÓN

Este algoritmo tiene un fin, el cual es al revisar todo el árbol de solución. Cuando revise todo este árbol, el algoritmo se detendrá, o antes en caso de encontrar la solución ideal. Debido a su naturaleza recursiva y alto costo de revisar las restricciones tiene una complejidad de $O((n^4)!)$ con el modo debug activado. Además se puede mejorar implementando aplicando otra forma de revisar las restricciones, el cual actualmente tiene un orden de $O(n^2)$.

Y otro método totalmente distinto seria la aplicación programación paralela, para realizar distintas operaciones al mismo tiempo y optimizar tiempos, junto también un sistema de predicción de nodos, y así eliminar nodos que son poco prometedores y que no llegaran a una solución óptima, esta última técnica es llamada branch and bound.

5.2 EJECUCIÓN

Para la ejecución de este programa es necesario un archivo con la siguiente estructura:

Entrada1.in

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

Figura 5: ejemplo de archivo de entrada.

donde la primera línea contiene dos números que representan el ancho y alto de la matriz respectivamente. En la siguiente línea irá un número entero N el cual representa las líneas siguientes que pueden haber. En las N líneas siguientes habrá un par de números por línea los cuales representan las posiciones de las sucursales (partiendo del 0) que deben estar si o si.

Dado lo necesario para su funcionamiento, ahora se hablará de la compilación y ejecución, tanto para su forma normal, como su modo debug.

- Para compilar el programa:

```
$ make
```

- para ejecutar el programa:

```
$ ./lab.out archivo_entrada
```

o

```
> lab.exe archivo_entrada
```

Para la utilización del modo DEBUG y poder apreciar cada movimiento realizado se debe ejecutar de la siguiente manera:

ejemplo:

- Para compilar el programa en modo DEBUG:

```
$ make debug
```

- para ejecutar el programa:

```
$ ./lab_debug.out archivo_entrada
```

o

```
> lab_debug.exe archivo_entrada
```

Dado la ejecución del programa, el archivo de salida para ambos casos cumple la siguiente estructura:

```
10
4-0 || 1-1 || 7-2 || 9-3 || 6-4 || 3-5 || 0-6 || 8-7 || 5-8 || 2-9 ||
_ _ _ X _ _ _ _ _ _
_ X _ _ _ _ _ _ _ _
_ _ _ _ _ _ X _ _ _
_ _ _ _ _ _ _ _ X _
_ _ _ _ _ X _ _ _ _
_ _ _ X _ _ _ _ _ _
X _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ X _ _
_ _ _ _ X _ _ _ _ _
_ _ X _ _ _ _ _ _ _
```

Figura 6: ejemplo de archivo de salida.

Donde la primera línea indica la cantidad de sucursales, la segunda línea muestra una lista con las coordenadas de las sucursales y las líneas siguientes una representación gráfica del mapa de la ciudad.

6 CONCLUSIONES

El algoritmo desarrollado para resolver el problema planteado debido a su naturaleza recursiva es relativamente fácil de implementar, debido a que sus restricciones son ampliamente conocidas y sencillas de dilucidar, debido a esto la implementación mediante backtracking para soluciones de bajo orden es recomendable. Pero debido a su alto orden de complejidad, para matrices de gran tamaño esto no es aconsejable debido a su gran tardanza en encontrar una solución. Pero también se debe considerar que este problema es considerado un problema NP, por lo cual dar una solución rápida para grandes tamaños sigue siendo una tarea de gran esfuerzo y de alto costo en cuanto al tiempo. Por esta razón realizar otra implementar usando otro tipo de metodologías o podas más eficientes, haciendo que el orden baje, o el tiempo promedio disminuya, la utilización de estas soluciones podría llegar a ser viables.

7 REFERENCIAS

- "El esquema algorítmico del backtracking", Carlos Medina, desconocido, disponible en : www.it-docs.net/ddata/3619.pdf
- "Backtracking", Jorge Baier Aranda, desconocido, disponible en : http://jabaier.sitios.ing.uc.cl/iic2552/backtracking.pdf
- "Backtracking", Andrés Becerra Sandoval, 2007, disponible en: http://cic.puj.edu.co/wiki/lib/exe/fetch.php?id=materias%3Alenguajes3%3Alenguajes_iii&cache=cache&media=materias:lenguajes3:backtracking.pdf
- "El problema de las N reinas", Andreas Spading, 2005, disponible en: https://www.cs.buap.mx/~zacarias/FZF/nreinas3.pdf