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

2-2018

Backtracking

Catalina Andrea Morales Rojas
e-mail: catalina.morales.r@usach.cl
https://github.com/CatalinaMoralesR

Backtracking

RESUMEN: *Para lograr resolver problemas se cuenta con una gran variedad de algoritmos, uno de los más conocidos corresponde a backtracking. Este documento presenta el análisis, implementación y solución para el problema del camino mínimo con un grafo completo utilizando el algoritmo de backtracking.*

PALABRAS CLAVE: Grafo, nodo, costo.

ABSTRACT: *To solve problems, we have a great variety of algorithms, one of the best known is backtracking. This document presents the analysis, implementation and solution for the minimum path problem with a complete graph using the backtracking algorithm.*

KEYWORDS: Graph, node, cost.

1 INTRODUCCIÓN

Backtracking es un algoritmo sencillo de implementar y que entrega la solución óptima (siempre que esta exista). Es por esto que a continuación se expone la implementación realizada a un problema que debe entregar como resultado el camino mínimo que se obtiene al recorrer todos los nodos de un grafo completo.

El presente documento busca analizar las ventajas y desventajas que presenta este algoritmo, junto con el desarrollo realizado en la resolución del problema.

2 DESCRIPCIÓN DEL PROBLEMA

El problema a resolver consiste en encontrar el camino mínimo que debe recorrer un vendedor al visitar todas las ciudades presentes en un grafo completo, es decir, en donde cada nodo (ciudad) se encuentra directamente conectado con el resto (como se muestra en la imagen), con la restricción de que solo se debe pasar por cada ciudad una sola vez.

Para esto, cada una de los caminos entre las ciudades cuenta con un costo asociado.

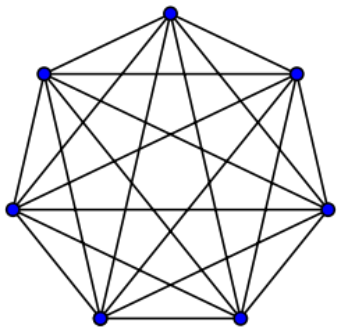

Figura 1: ejemplo de grafo completo.

El problema debe ser implementado mediante el uso del algoritmo de backtracking, el cual permite encontrar el camino óptimo.

3 MARCO TEÓRICO

3.1 Grafo

Un grafo corresponde a una representación gráfica de nodos, los cuales se encuentran unidos por líneas llamadas aristas.

3.2 Nodo

Un nodo corresponde a un punto de intersección entre varios elementos que confluyen en un mismo lugar.

3.3 Backtracking

El principio de backtracking es generar una solución mediante búsqueda en profundidad. A continuación, se siguen buscando soluciones, pero si se encuentra una que no cumpla con los criterios establecidos o que el costo sea mayor a encontrada con anterioridad, se vuelve un paso atrás y se elige la siguiente ruta.
Este algoritmo a diferencia de fuerza bruta no genera todas las soluciones, solo aquellas que satisfacen lo requerido.

4 DESCRIPCIÓN DE LA SOLUCIÓN

4.1 IDEA

La idea para resolver el problema consiste en encontrar una solución inicial, para luego volver atrás y comparar si el resto de las posibles soluciones corresponden a un mejor resultado.
Por ejemplo, si se cuenta con los nodos:
“A,B Y C”
Junto con sus costos asociados:
A-B: 1
B-C: 5
A-C: 3
El algoritmo en primera instancia encuentra el camino A - B - C, con el costo: 6.
Luego, volverá atrás, tomará como inicial:
A-C: 3, y como es menor al costo anterior (6) continuará.
Por último añade A-C-B, Con costo: 8, como es mayor al camino óptimo, se descarta.
El algoritmo termina cuando se han recorrido todos los nodos.

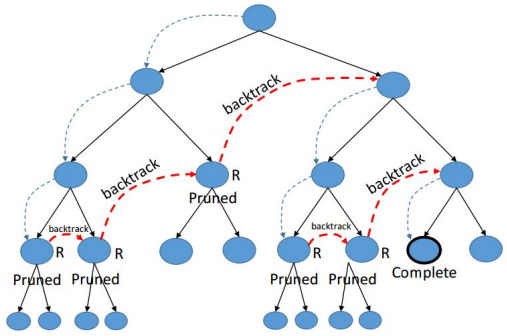

Figura 2: Traza de backtracking.

4.2 REPRESENTACIÓN Y ESTRUCTURA

Los datos son tomados desde el archivo de entrada, el cual cuenta con la cantidad de ciudades y el costo que tiene llegar desde una a otra.
El costo asociado entre cada ciudad se ingresa a una matriz dinámica, la cual tiene un tamaño de MXM, donde M corresponde al número de ciudades.
El orden de los caminos que se recorren, el resultado final y los caminos que ya fueron visitados se manejan cada uno en un arreglo dinámico, en donde el tamaño corresponde al número de ciudades, y todos se inicializan con valor -1.

4.3 PSEUDOCÓDIGO

Entradas:
i: Ciudad inicial
Camino: Lista en donde se almacenarán los caminos recorridos
matriz: Matriz con los costos de ir desde una ciudad a otra.
cantCiudades: Número de ciudades en el archivo
visitados: Lista con las ciudades que han sido visitadas.
resultado: Lista que contendrá el resultado con el menor costo.
CostoMejorCamino: corresponde a una variable global que contiene el costo asociado al mejor camino, se va modificando a medida que disminuye el valor del camino.

```
 backTracking( i, camino, matriz, cantCiudades, visitados,
resultado){
_
| _
| | Si (validar(ciudades,camino) = verdadero)
| |    costo = calcularCamino(camino,matriz,cantCiudades)
| |    _
| |    |Si Costo < CostoMejorCamino
| |    |   CostoMejorCamino = costo
| |    |   Para i = 0 hasta CantCiudades
| |    |      resultado[i] = camino[i]
| |    |_  Retornar nuevo resultado
| |    _
| |    |Si no
| |    |_  Retornar resultado
| |__
|
|    _
| |Si no
| |    visitados[i] = 1;
| |    _
| |    |  Para j = 0 hasta CantCiudades
| |    |
| |    |    _
| |    |    |Si visitados[k] ==-1
| |    |    |   camino = agregarCamino(j,camino,cantCiudades)
| |    |    |
| |    |    |Costo2=calcularCamino(camino,matriz,cantCiudades)
| |    |    | _
| |    |    | |Si costo2 < CostoMejorCamino
| |    |    | |
| |    |    | |       resultado=
| |    |    | |      backTracking(k,camino,matriz,
| |    |    | |_           cantCiudades,visitados,resultado)
| |    |    |
| |    |    |_   camino = eliminarCamino(k,camino,cantCiudades)
| |    |_
| |_    visitados[i] = -1
|     Retornar resultado
|_
```

validar = corresponde a una función que retorna como resultado verdadero si todas las ciudades fueron visitadas

```
validar(visitados, cantCiudades)
            i = 0;
            Mientras i<cantCiudades
                Si visitados[i]==-1
                   Retornar falso
                i++;
            Retornar verdadero.
```

calcularCamino = corresponde a una función que en base a las ciudades agregadas, calcula el costo de llegar desde la primera a la última.

```
calcularCamino(camino,matriz,cantCiudades)
            cantCiudades = cantCiudades - 1
            costo = 0
            Para i = 0 hasta cantCiudades
                Si camino[i] != -1 y  camino[i+1]!=-1
            costo = costo + matriz[camino[i]][camino[i+1]]
            Retornar costo
```

agregarCamino = agrega la ciudad indicada a la lista de caminos.

```
agregarCamino( k,camino, cantCiudades)
            Para i = 0 hasta cantCiudades
                Si camino[i] == -1
                    camino[i] = k
                   Retornar camino
            Retornar camino
```

EliminarCamino = elimina la ciudad indicada de la lista de caminos.

```
eliminarCamino(k, camino, cantCiudades)
            Para i = 0 hasta cantCiudades
                    Si camino[i] == k
                       camino[i] = -1
                       Retornar camino
            Retornar camino
```

4.4 TRAZA

La traza se encuentra en el apartado 8.1

4.5 ORDEN DE COMPLEJIDAD

Para backtracking, generar todos los posibles caminos en un grafo completo implica que a medida que se agrega un nuevo nodo, para el nodo siguiente quedan n-1 nodos que visitar, para el subsiguiente n-2 y así sucesivamente. Por lo tanto, la complejidad de establecer caminos que cuenten con todas las ciudades corresponde a:

$$\sim O(n!)$$

5 ANÁLISIS DE LA SOLUCIÓN

5.1 ANÁLISIS DE IMPLEMENTACIÓN

1-¿Se detiene?
El algoritmo se detiene cuando recorre todos los caminos.

2-¿Entrega solución?

Entrega como resultado la combinación de caminos que permiten visitar todas las ciudades al menor costo.

3-Eficiencia

La complejidad del algoritmo es de orden ~ $O(n!)$, la cual no es eficiente, ya que no es polinómica.

4-¿Se puede mejorar?

El algoritmo se puede mejorar si al momento de recorrer nuevas ramas, se descartan aquellas que fueron eliminadas con anterioridad por tener un costo mayor al costo mínimo. De esta manera, se descartan ramas que no entregarán una solución óptima.

5-¿Otra idea?

Se puede realizar el mismo algoritmo mediante fuerza bruta, pero este no es eficiente, ya que no descarta las soluciones, aunque si entrega el óptimo.

.

5.2 EJECUCIÓN

Para la ejecución del programa se debe tener instalado el compilador GNU gcc.

- Para compilar el programa:

```
$ gcc lab_avanzados2.c funciones.c -o lab
```

- para ejecutar el programa:

```
$ ./lab
```

El programa solicitará ingresar el nombre del archivo que contiene la entrada, junto con su extensión:

```
$ Ingrese el nombre del archivo: entrada.txt
```

Para poder entrar al modo debug:

```
$ gcc lab_avanzados2.c funciones.c -DDEBUG -o lab
```

6 CONCLUSIONES

A partir del trabajo desarrollado es posible concluir que el algoritmo de backtracking siempre encuentra el resultado óptimo si es que existe alguno, además su implementación es relativamente sencilla.
Por otro lado, este algoritmo hace uso de una gran cantidad de memoria, ya que debe almacenar los resultados de los llamados recursivos, por lo tanto, si se desea implementar, se debe contar con un buen almacenamiento. Su tiempo de ejecución corresponde a $O(n!)$, por lo cual no es una buena estrategia para cuando se necesite eficiencia en la resolución de un problema, debido a que el orden no es polinomial.

Al comparar backtracking con fuerza bruta, se puede apreciar que el primero sólo genera aquellas combinaciones que cumplen los requisitos requeridos por el problema, y una vez encontrada una solución válida, construye solo las que tienen un menor costo, por lo cual, en relación a fuerza bruta, su eficiencia es mayor. Con respecto al código, la dificultad de backtracking es mayor, debido a que se debe implementar mediante el uso de recursión y descartar caminos a medida que son generados.

7 REFERENCIAS

- "Apuntes de grafos" , Desconocido ,2005 , disponible en : www.infor.uva.es/~cvaca/asigs/estr0506apg.pdf?fbclid=IwAR1Ajm3r1PSteh5U6JUFmkQejbCZBilEWVQn0ia_y1XSVHBZfeGIh9LX7yI

8 APARTADO

8.1 TRAZA

Archivo de entrada:

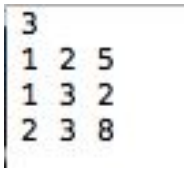

Figura 3: Archivo de entrada.

Entradas función:
i = 1
camino = [-1,-1,-1]
matriz = [[-1,5,2],
[5,-1,8],
[2,8,-1]]]
cantCiudades = 3
visitados = [-1,-1,-1]
resultado = [-1,-1,-1]
costoMejorCamino = 100000
backTracking(i, camino, matriz, cantCiudades, visitados, resultado){

Marcar como visitado: 1
Agregar camino: 2
Camino : [1, 2]
costo2: 5
costo2 < costoMejorCamino = 5 < 100000
Continuar
backtracking(2,camino,matriz,cantCiudades,visitados,resultado)
Marcar como visitado: 2
Agregar camino: 3
Camino : [1, 2, 3]
costo2: 13
costo2 < costoMejorCamino = 13 < 100000
Continuar
backtracking(3,camino,matriz,cantCiudades,visitados,resultado)
costo < costoMejorCamino = 13 < 13

Nuevo costo: 13
Retornar Mejor Camino: [1, 2, 3]
Eliminar camino: 2
Eliminar camino: 1
Agregar camino: 3
Camino : [1, 3]
costo2: 2
costo2 < costoMejorCamino = 2 < 13
Continuar
backtracking(3,camino,matriz,cantCiudades,visitados,resultado)
Marcar como visitado: 3
Agregar camino: 2
Camino : [1, 3, 2]
costo2: 10
costo2 < costoMejorCamino = 10 < 13
Continuar
backtracking(2,camino,matriz,cantCiudades,visitados,resultado)
costo < costoMejorCamino = 10 < 10
Nuevo costo: 10
Retornar Mejor Camino: [1, 3, 2]
Eliminar camino: 1
Eliminar camino: 2
Marcar como visitado: 2
Agregar camino: 1
Camino : [2, 1]
costo2: 5
costo2 < costoMejorCamino = 5 < 10
Continuar
backtracking(1,camino,matriz,cantCiudades,visitados,resultado)
Marcar como visitado: 1
Agregar camino: 3
Camino : [2, 1, 3]
costo2: 7
costo2 < costoMejorCamino = 7 < 10
Continuar
backtracking(3,camino,matriz,cantCiudades,visitados,resultado)
costo < costoMejorCamino = 7 < 7
Nuevo costo: 7
Retornar Mejor Camino: [2, 1, 3]
Eliminar camino: 2
Eliminar camino: 0
Agregar camino: 3
Camino : [2, 3]
costo2: 8
costo2 > costoMejorCamino = 8 < 7

Descartar Camino
Eliminar camino: 2
Marcar como visitado: 3
Agregar camino: 1
Camino : [3, 1]
costo2: 2
costo2 < costoMejorCamino = 2 < 7
Continuar
backtracking(1,camino,matriz,cantCiudades,visitados,resultado)
Marcar como visitado: 1
Agregar camino: 2
Camino : [3, 1, 2]
costo2: 7
costo2 > costoMejorCamino = 7 < 7
Descartar Camino
Eliminar camino: 1
Eliminar camino: 0
Agregar camino: 2
Camino : [3, 2,]
costo2: 8
costo2 > costoMejorCamino = 8 < 7
Descartar Camino
Eliminar camino: 1

Retornar Mejor Camino [2,1,3] costo:7