Algoritmica
Practica 4. Backtracking
2018-2019

UNIVERSIDAD
DE GRANADA

Félix Ramírez García
Raúl Del Pozo Moreno
Cristian Piles Ruiz
Oleksandr Kudryavtsev
Sixto Coca Cruz

INDICE

1. Introducción

El objetivo de la práctica es resolver mediante Backtracking el problema ‘cena de gala’ y realizar un estudio empírico de la eficiencia. El problema es el siguiente:

Se va a celebrar una cena de gala a la que asistirán n invitados. Todos se van a sentar alrededor de una única gran mesa rectangular, de forma que cada invitado tendrá sentados junto a él a otros dos comensales (uno a su izquierda y otro a su derecha). En función de las características de cada invitado (por ejemplo categoría o puesto, lugar de procedencia,...) existen unas normas de protocolo que indican el nivel de conveniencia de que dos invitados se sienten en lugares contiguos (supondremos que dicho nivel es un número entero entre 0 y 100). El nivel de conveniencia total de una asignación de invitados a su puesto en la mesa es la suma de todos los niveles de conveniencia de cada invitado con cada uno de los dos invitados sentados a su lado. Se desea sentar a los invitados de forma que el nivel de conveniencia global sea lo mayor posible. Diseñar e implementar un algoritmo vuelta atrás para resolver este problema. Realizar un estudio empírico de su eficiencia.

2. Planteamiento

Primero creamos una matriz M[i][j] simétrica para almacenar la conveniencia entre los invitados i y j . Los valores asignados entre los invitados (número aleatorio entre 0 y 100) es simétrica ya que hemos asumido que entre un invitado i y un invitado j hay un nivel de conveniencia mutuo . Los valores de la diagonal son ceros, ya que un invitado no tiene conveniencia consigo mismo.

Cogemos una solución inicial (un recorrido de nuestro árbol hasta un nodo hoja) sin tener en cuenta la conveniencia, luego usando backtracking nos vamos hasta el nodo vivo (estado del problema que ya ha sido generado, pero aún no se han generado todos su hijos) utilizando nuestro criterio de poda, que la cota actual de conveniencia mejore respecto a la que está guardada, hasta llegar a otro nodo hijo, comparamos si el nivel de conveniencia total en ese nodo hijo con el nivel de conveniencia total de la solución parcial, elegimos la mejor y nos quedamos con ella. Repetimos hasta que no quede ningún nodo vivo y obtendremos nuestra solución.

3. Elementos de la solución del problema

Los elementos de la solución del problema son los siguientes:

a. Representación del problema

El problema se ha representado usando una Matriz simétrica rellenada con valores aleatorios y cuya diagonal son ceros.

b. Representación de la solución

La solución es un vector de tamaño (N) en la que almacena el orden de los invitados en la mesa. En cada celda del vector se almacena el invitado que se sienta en la posición i (i < N).

c. Restricciones explícitas

Los valores que toma la solución, enteros de 1 hasta N.

d. Restricciones implícitas

Un comensal sólo puede estar sentado en un único asiento y cada asiento solo puede estar ocupado por un único comensal.

e. Cotas

Definimos la cota local como la conveniencia acumulada en cada nodo. En caso de ser nodo hoja , si la conveniencia es mayor que la máxima , actualizamos la cota global con el valor de la cota local.

4. Pseudocodigo

Antes de mostrar el pseudocódigo de la función que realiza el backtracking vamos a exponer la estructura de la implementación de la solución del problema.

El código contiene las siguientes funciones :

```
1 ) int **rellenarMatriz(int n)
```

Rellena la matriz con valores aleatorios, diagonal de ceros y la devuelve

```
2 ) void imprimirMatriz(int **m, int n)
```

Imprime por pantalla el resultado de una matriz.

```
3 ) void imprimirSolucion(int *solucion, int n)
```

Imprime por pantalla la solución , almacenada en un vector de tamaño N.

```
4 ) int convenienciaActual(int n, int solucion[], vector<bool> &sentados, int
**matriz)
```

Esta función devuelve la conveniencia que tiene una solución parcial del problema.

```
5 ) void algoritmo_backtracking (int n_invitados, int nivel, int invitado, int
&conveniencia_actual, int solucion_actual[],vector<bool> &sentados, int **m, int
solucion_final[])
```

A continuación se muestran los parámetros de la función de backtracking :

n_invitados	entero que almacena el número de invitados
nivel	entero que almacena el nivel de profundidad del árbol
invitado	entero que almacena el índice del invitado actual
conveniencia_actual	entero que almacena la conveniencia de una solución
solucion_actual	vector de enteros que almacena la lista de los invitados sentados en una solución parcial.
sentados	vector de booleanos que almacena si un invitado está sentado en una solución parcial
m	matriz de enteros simetrica
solucion_final	vector de enteros que almacena la lista de los invitados sentados en la solución parcial con mayor conveniencia hasta el momento.

El pseudocódigo de la función recursiva es el siguiente:

```
Función backtracking
    Se sienta un invitado
    Para cada invitado restante:

        Si  no esta sentado
            Calcular cota local (conveniencia actual)

            Si no se ha recorrido la primera rama completa (solucion base)
                Inicializar cotas por nivel

            Si se ha recorrido

                Si la cota no mejora
                    Se indica que se poda
                Si mejora
                    Se actualiza la cota

            Si no se poda
                Función recursiva backtracking aumentando nivel en 1 (hijo)

                Si se esta en una hoja
                    Se indica que se ha recorrido una rama completa (solucion base)
                        Si la conveniencia mejora
                            Actualizar solucion con la nueva solucion
                            Actualizar cota con nuevas cotas

            Si se ha podado
                Se indica que no se poda para la siguiente

            Se levanta al invitado
```

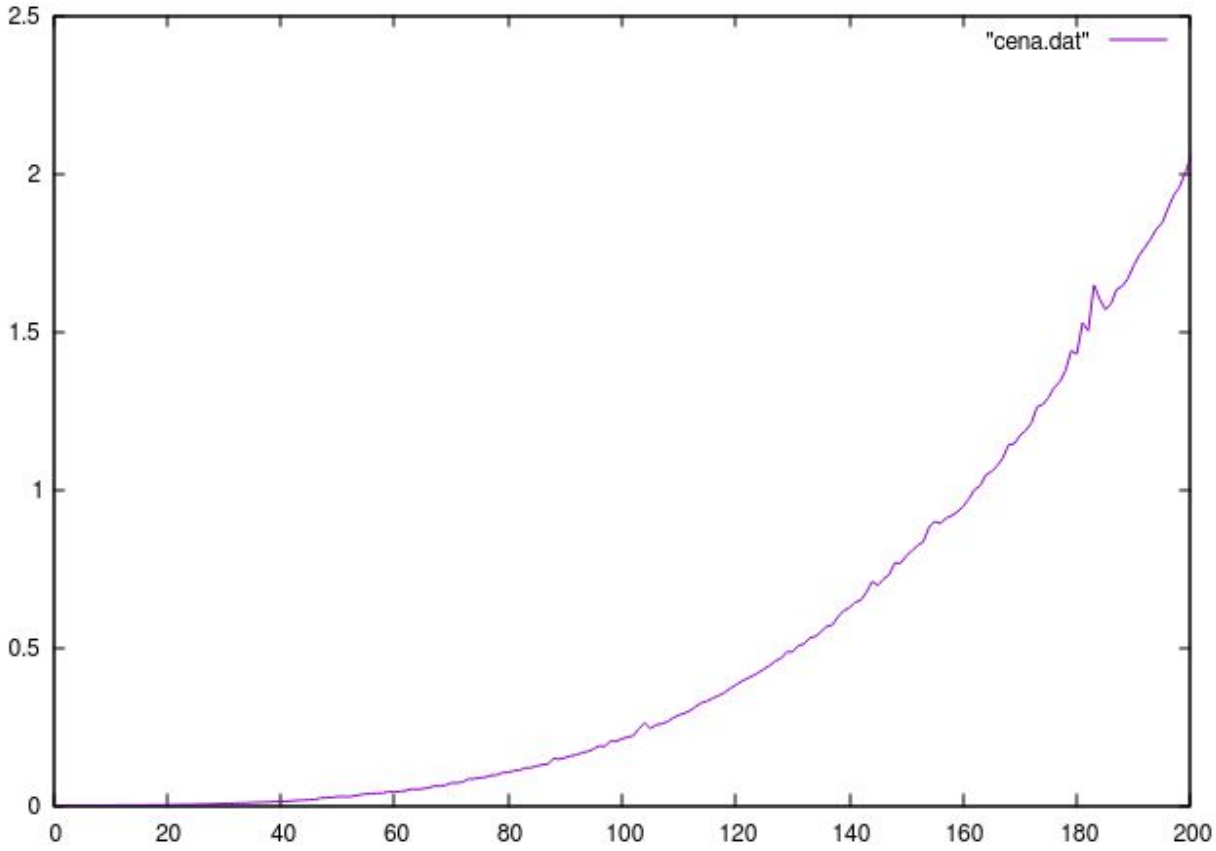
5. Eficiencia

A continuación vamos a medir la eficiencia híbrida obtenida tras la ejecución del programa con diferentes tamaños de entrada.

a. Salidas de ejecución

Aquí se muestra una pequeña porción de los datos generados.

N	Tiempo (seg)
1	5.96e-07
25	0.0032621
50	0.0275547
75	0.0871117
100	0.210825
125	0.431553
150	0.789624
175	1.29055
200	2.05619

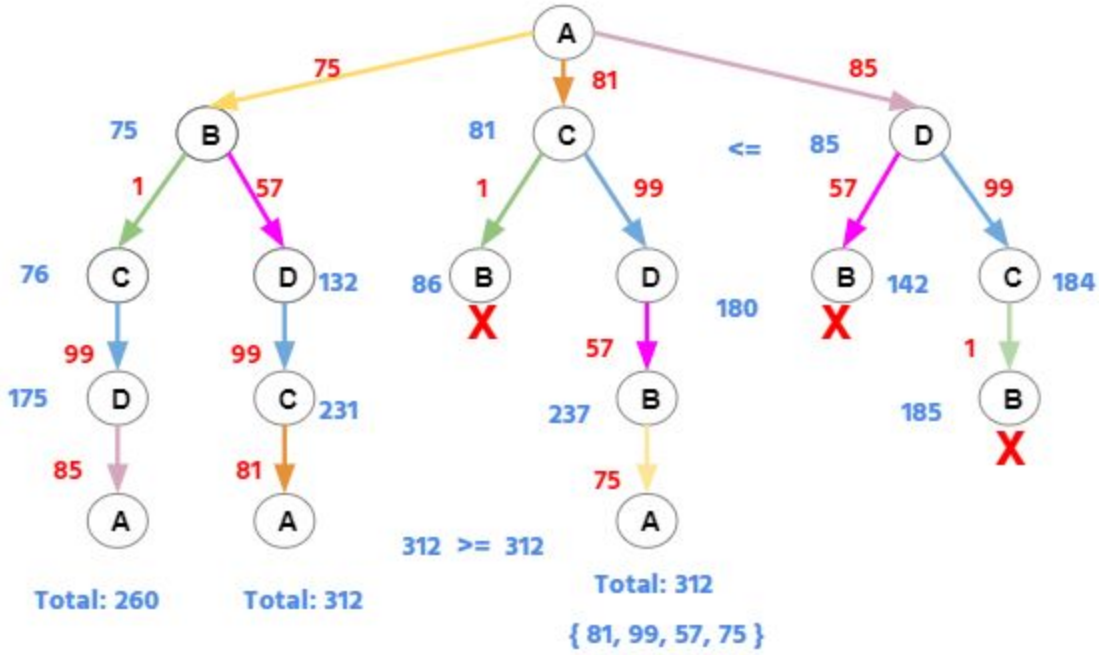

"cena.dat"
2.5
2
1.5
1
0.5
0
0
20
40
60
80
100
120
140
160
180
200

6. Ejemplo de árbol recorrido

A partir de la siguiente tabla de valores de conveniencia:

	A	B	C	D
A	0	75	81	85
B	75	0	1	57
C	81	1	0	99
D	85	57	99	0

Se recorre un árbol que tiene la siguiente estructura (no almacenada)



Como se puede observar, se ha recorrido las dos primeras ramas de la izquierda al completo, la rama {A - B - C - D - A} se recorre si o si ya que es la solución base, ya que la primera rama tiene que hacerla al completo.

Cuando termina con esa rama, vuelve al nodo B y sigue por D, en este caso, va comparando las cotas de cada nivel (en azul) con las cotas de la rama solución, como todas las cotas mejoran, se consigue terminar la rama y esta pasa a ser la nueva rama solución, actualizando sus cotas.

A continuación vuelve al nodo A y sigue por el nodo C, aquí continúa con el mismo proceso de comparación de cotas y en el momento en que alguna cota no mejore, se poda la rama en ese lado (marcada con una X roja) y se pasa a la siguiente rama.

Asi, se han obtenido 3 soluciones:

1. A - B - C - D - A (por defecto) con 260 puntos de conveniencia
2. A - B - D - C - A con 312 puntos, siendo una solución mejor que la ya obtenida
3. A - C - D - B - A con 312 puntos, esta solución es la misma que la última encontrada, por lo que esta pasa a ser la nueva solución