Universidad Rey Juan Carlos

Práctica 3: Backtracking

Diseño y Análisis de Algoritmos

- Los códigos tendrán que probarse con el juez automático **DOMjudge**
  - gibson.escet.urjc.es
  - El nombre de usuario será “team-XXX”, donde XXX es un número único por cada alumno. Podéis ver qué número os corresponde en un documento subido al aula virtual que describe la relación entre el nombre de usuario y el nombre de un alumno.
  - La contraseña es vuestro DNI (incluida la letra final en mayúsculas).
- Además de probar vuestros códigos con DOMjudge debéis subir el ficheros fuente al aula virtual
- No se entregará una memoria
- Fecha límite: Se especificará en el campus virtual
- 10 % de la nota final

Índice

1. Subcolecciones del divisor (5 puntos) **2**

2. Problema del viajante (5 puntos) **4**

1. Subcolecciones del divisor (5 puntos)

1.1. Introducción

En este ejercicio se plantea un problema para que uséis la técnica de *backtracking*. Se recomienda utilizar los esquemas vistos en clase.

1.2. Enunciado del problema

Dada una colección C de n números enteros positivos, y un determinado número entero m, tal que $1 \leqslant m \leqslant n \leqslant 100$, se pide hallar cuántas subcolecciones de m números de C se pueden obtener de tal forma que el menor elemento de la subcolección sea un divisor de los $m-1$ enteros restantes (la división resultaría ser entera). Llamémosle a este resultado s.
Nota: C puede contener elementos repetidos.

1.2.1. Descripción de la entrada

La primera línea contiene n. La segunda contiene los n números enteros que componen la colección C, separados por espacios en blanco. La tercera línea contiene m y un salto de línea.

1.2.2. Descripción de la salida

La salida contiene el entero s, seguido de un salto de línea.

Ejemplo de entrada 1

```
5↵
2␣2␣5␣4␣7↵
2↵
```

Salida para el ejemplo de entrada 1

```
3↵
```

ACLARACIÓN: En este ejemplo las subcolecciones son:

1. {El primer 2, el segundo 2}. Se considera que la subcolección { El segundo 2, el primer 2} es idéntica, ya que está formada por los mismos elementos de C.
2. {El primer 2, el 4}
3. {El segundo 2, el 4}. Se considera que esta subcolección es distinta de la anterior, al estar formada por elementos diferentes de la colección inicial C.

Ejemplo de entrada 2

```
5
2 2 5 4 7
3
```

Salida para el ejemplo de entrada 2

```
1
```

Ejemplo de entrada 3

```
5
2 2 5 4 7
1
```

Salida para el ejemplo de entrada 3

```
5
```

Ejemplo de entrada 4

```
5
2 2 5 4 7
4
```

Salida para el ejemplo de entrada 4

```
0
```

Ejemplo de entrada 5

```
10
6 14 8 1 6 7 14 7 5 4
2
```

Salida para el ejemplo de entrada 5

```
17
```

2. Problema del viajante (5 puntos)

2.1. Introducción

Un comerciante debe realizar un recorrido por una serie de ciudades de tal manera que solo visite cada ciudad una vez, y termine en la ciudad de la que partió. Queriendo ahorrar tiempo y dinero en gasolina, el viajante decide calcular el recorrido más corto que cumpla las restricciones anteriores, utilizando *backtracking* (vuelta atrás).

2.2. Enunciado del problema

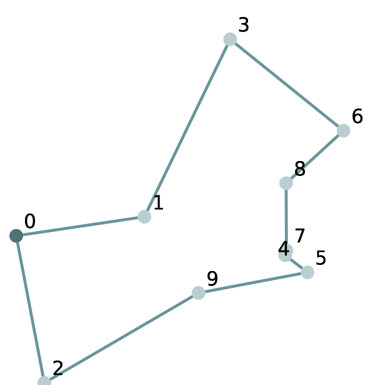
En este problema se dispone de las coordenadas en un plano (x_i, y_i) de las n ciudades, para $i = 0, \dots, n-1$. Se deberá construir una matriz simétrica $\mathbf{D}$ que contenga las distancias Euclídeas entre las ciudades. En concreto, los elementos de la matriz son:

$$d_{i,j} = d_{j,i} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Se debe implementar un método recursivo basado en la técnica de *backtracking* para calcular el camino de distancia mínima que empiece y termine en la ciudad 0, que pase por el resto de las ciudades una sola vez. La solución será una lista $\mathbf{s} = (0, s_1, \dots, s_{n-1})$, donde los términos s_i son índices de ciudades, de tal forma que se minimice:

$$\sum_{i=0}^{n-2} (d_{s_i, s_{i+1}}) \quad + \quad d_{s_{n-1}, 0}$$

y donde $s_i \neq s_j$.



El algoritmo deberá calcular tanto la longitud del camino óptimo, como la lista de ciudades que definen el camino. Además, como un camino se puede recorrer en dos sentidos, en las soluciones el índice de la primera ciudad debe ser menor que el de la última. Es decir, $s_1 < s_{n-1}$. Por ejemplo, para las 10 ciudades de la figura la solución es:

$$\mathbf{s} = [0, 1, 3, 6, 8, 7, 4, 5, 9, 2]$$

2.2.1. Descripción de la entrada

La primera línea de la entrada contendrá n. En las siguientes n líneas se especificarán las coordenadas (x_i, y_i), desde $i = 0$ hasta $i = n - 1$.

2.2.2. Descripción de la salida

La salida contendrá dos líneas. En la primera se escribirá la longitud del camino mínimo, con 4 decimales obligatoriamente. En la segunda línea se especificarán los índices de las ciudades que forman el camino óptimo ($\mathbf{s}$), separados por espacios en blanco. La segunda línea terminará con un salto de línea.

Ejemplo de entrada

```
10
0.07952670864170575 0.4798500567592401
0.41251388375062115 0.5293641341429488
0.1525191338187477 0.09798037149221062
0.6349500823733416 0.9901197701364185
0.7780004527656195 0.4293123673650461
0.8357955962085358 0.3850631742789996
0.9293653820182484 0.752829474604692
0.7811846332442332 0.44163376192797676
0.7803357169147921 0.616535309428859
0.5528779682466874 0.33149653436424664
```

Salida para el ejemplo de entrada

```
2.8290
0 1 3 6 8 7 4 5 9 2
```