.UBA fiuba
FACULTAD DE INGENIERÍA

Generar y probar: Permutaciones y el problema del viajante

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

Generar y probar: Permutaciones

- **Espacio de soluciones**
  - Trabajaremos sobre la determinación de un ordenamiento de n elementos
  - Los elementos son únicos e indivisibles.
  - Todos los elementos deben ser seleccionados.
- **Función generativa**
  - Generación de las permutaciones de los elementos
  - Corresponderá a las restricciones explicitas del problema

Ejemplo práctico: Problema del viajante de comercio

- **Contamos con un conjunto de “n” ciudades a visitar.**
  - Existen caminos que unen pares de ciudades.
- **Cada camino**
  - inicia en una ciudad “x” y finaliza en la ciudad “y”
  - tiene asociado un costo de tránsito de wx,y.
- **Partiendo desde una ciudad inicial y finalizando en la misma se quiere**
  - construir un circuito que visite cada ciudad una y solo una vez minimizando el costo total.

Ejemplo práctico: Problema del viajante de comercio

El circuito parte de una ciudad inicial

la nombraremos “0”

Y debe finalizar en la misma ciudad

Cada ciudad

Recibe un identificador numérico entre 1 y n.

Para representar el orden de visita a las ciudades

Usaremos un vector C de n posición

En la primer posición la ciudad que iremos desde “0”

En la última posicion la ciudad desda la que volveremos a “0”

2	3	1	5	6	7	4

0 → 2 → 3 → 1 → 5 → 6 → 7 → 4 → 0

Ejemplo práctico: Problema del viajante de comercio

Para un circuito “C”

- Podemos determinar su factibilidad
- Podemos calcular su costo

Un circuito es factible

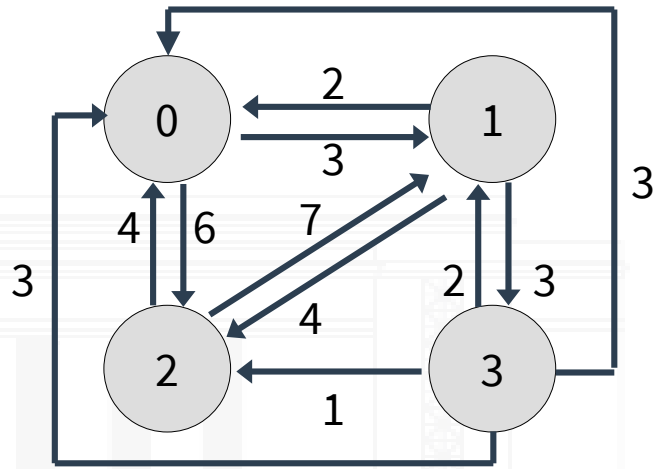
- Si existe un camino que une a cada par de ciudades en él

El costo de circuito

- Es la suma de los costo de los caminos entre ciudades en él

Existen n! posibles circuitos

- Correspondientes a generar las n permutaciones de las ciudades



2	3	1	✗
2	1	3	✓

Costo: 6 + 7 + 3 + 3 = 19

Generación de las permutaciones

- **Se generarán en orden lexicográfico.**
  - Se comenzará con un vector con los elementos ordenados de forma creciente
- **Generar la próxima permutación**
  - Se realiza en un proceso de 3 pasos
- **Finaliza el proceso**
  - cuando el vector esta ordenado de forma decreciente

Con n=4 → 4! = 24

1	2	3	4
1	2	4	3
1	3	2	4
1	3	4	2
1	4	2	3
1	4	3	2
2	1	3	4
...			
4	3	2	1

Generación de permutaciones: Inicialización

Para inicializar:

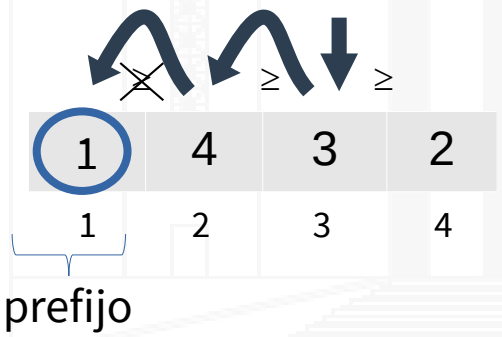
Establecer en cada posición C[i] el elemento i ordenado lexicográficamente

Corresponde a realizar el circuito

Complejidad temporal: O(n)

```
Sea C un vector representando un orden de
visita de las m ciudades internas


Inicializar C:
     Desde x=1 a n
          C[x]=x
```

Generación de permutaciones: Obtener siguiente - Paso 1

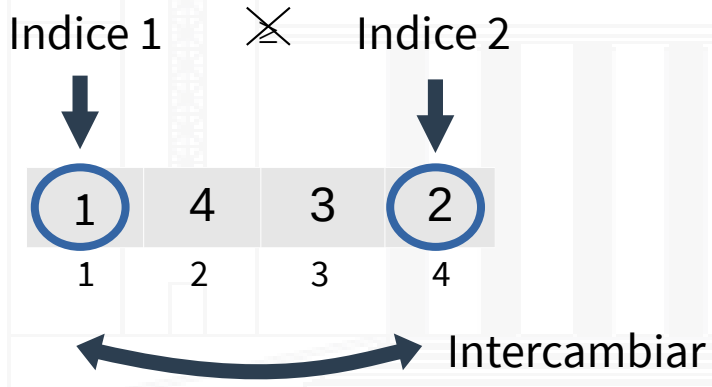


Sea C un vector representando un orden de visita de las m ciudades internas

```
permutar C:
     Sea indice1=n-1
     Mientras C[indice1]>=C[indice1+1]
          indice1 -= 1

     Si indice1 == 0
          retornar 'fin'

     ...
```

Generación de permutaciones: Obtener siguiente - Paso 2

Reemplazar el último elemento del prefijo con el elemento más pequeño a su izquierda que sea superior a este



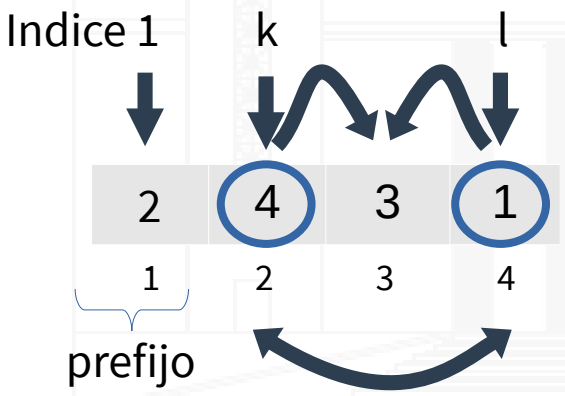
Complejidad temporal (en el peor de los casos): O(n)

```
...

Sea indice2=n
Mientras C[indice1]>=C[indice2]
     indice2 -= 1

Intercambiar C[indice1] y C[indice2]

...
```

Generación de permutaciones: Obtener siguiente - Paso 3

Ordenar de menor a mayor los elementos por fuera del prefijo

Invertir el vector fuera del prefijo



Complejidad temporal (en el peor de los casos): O(n)

```
...

Sea k = indice1 + 1
Sea l = n

Mientras k < l
    Intercambiar C[k] y C[l]
    k += 1
    l -= 1

retornar C
```

Generación de permutaciones: Obtener siguiente - Final

La generación finaliza cuando los elementos quedan ordenados de mayor a menor

(equivalente a la ausencia de prefijo)

Complejidad temporal (en el peor de los casos): O(n)

```
Sea C un vector representando un orden de
visita de las m ciudades internas


permutar C:
     Sea indice1=n-1
     Mientras C[indice1]>=C[indice1+1]
          indice1 -= 1

     Si indice1 == 0
          retornar 'fin'

     ...
```

Verificación de la solución

Costo:

Evaluar la suma de los costos de los caminos que lo integran

Opción: Considerar costo infinito si no existe un camino que une a dos ciudades en el circuito

Complejidad temporal: O(n)

```
Sea o la ciudad de inicio y finalización
Sea w[x,y] el costo de transitar por el
camino que une la ciudad x a la y.

Costo C:
     Sea costoTotal = w[o,C[1]]
     Desde i=1 hasta m
          costoTotal += w[C[i],C[i+1]]
     costoTotal += w[C[m],o]

     retornar costoTotal
```

Generar y probar: Problema del viajante de comercio

Unificando:

Comenzar con el primer circuito posible

Mientras quedan circuitos posibles obtener el próximo y evaluar si es superior al mejor previamente encontrado.

Complejidad temporal: O((n+1)!)

```
Inicializar C

Sea minimoCosto = Costo C
Sea soluciónинima = C

Mientras permutar C <> 'fin'
     Si minimoCosto  > Costo C
          minimoCosto = Costo C
          soluciónинima = C

retornar soluciónинima y minimoCosto
```