.UBAfiuba
FACULTAD DE INGENIERÍA

Backtracking: Problema del viajante de comercio

Teoría de Algoritmos I (75.29 / 95.06)

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

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.

Representación del circuito

El circuito parte de una ciudad inicial

la nombraremos “0”

Y debe finalizar en la misma ciudad

2	3	1	5	6	7	4

Cada ciudad

Recibe un identificador numérico entre 1 y n.

Para representar el orden de visita a las ciudades

0 → 2 → 3 → 1 → 5 → 6 → 7 → 4 → 0

Usaremos un vector C de n posición

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

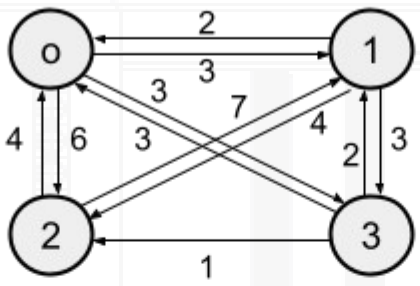
En la última posición la ciudad desde la que volveremos a “0”

Árbol de estados del problema

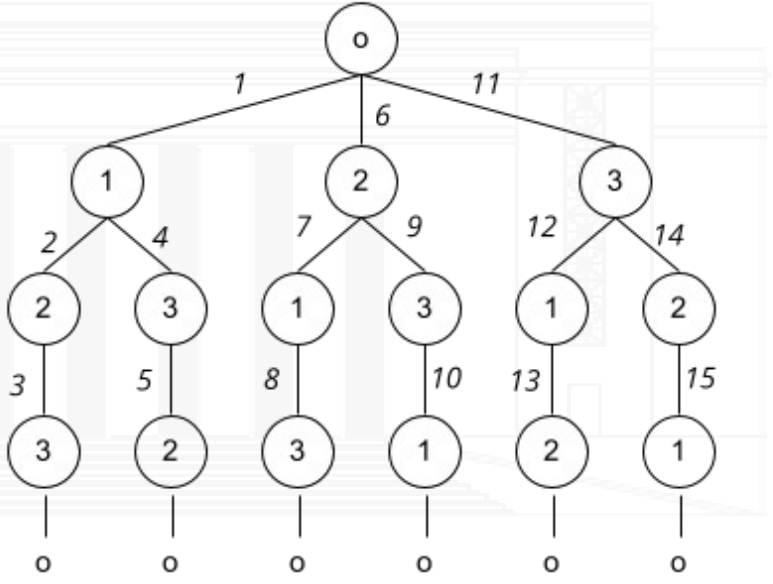
- **La raíz representa el comienzo del viaje partiendo de la ciudad inicial**
- **Cada nodo representa la inclusión de una ciudad en el recorrido que no fue previamente visitada**
  - Tiene como posibles estados descendientes las posibles elecciones de próximas ciudades (restringidas por las previamente visitadas).
  - Al elegir una opción se debe sumar el costo del traslado
- **El camino desde la raíz hasta el nodo representa el recorrido realizado desde la ciudad inicial hasta el momento**
  - El costo del recorrido es la suma de los costos de los trayectos entre las ciudades realizadas
- **El árbol para n ciudades tiene un total de n! estados solución**

Ejemplo: Árbol de estados del problema

Supongamos la siguiente instancia del problema:



Los estados se pueden representar como un árbol de permutaciones:



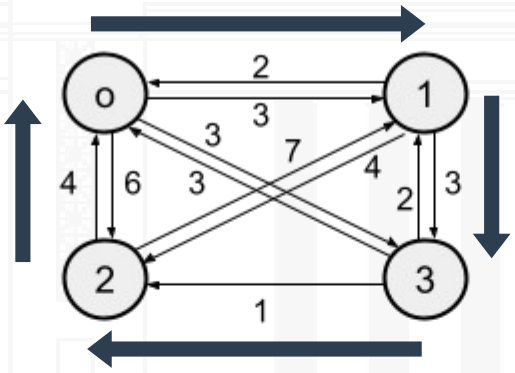
Tenemos:

3! = 6 estados solución (permutaciones de 3 ciudades)

$1+\sum_{i=1}^{n}\prod_{j=1}^{i}(n+1)-i$ estados del problema

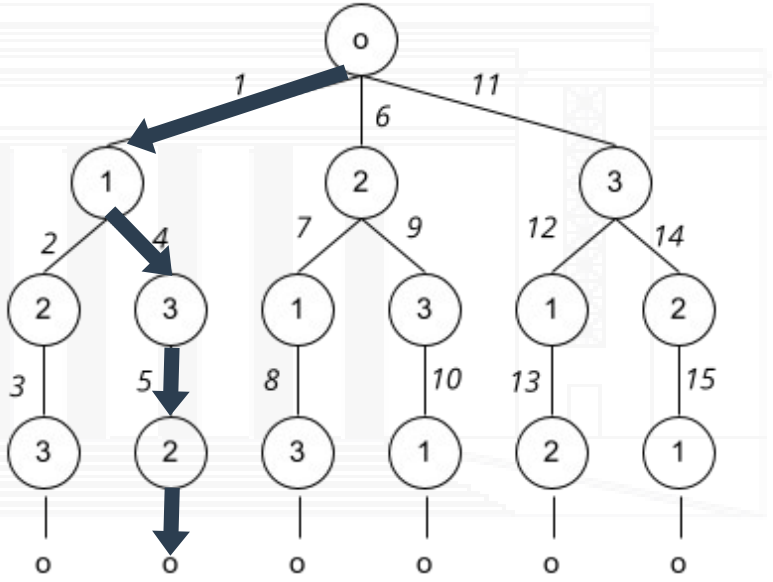
Ejemplo: Árbol de estados del problema

Posible circuito:



Costo: 11

Equivale al recorrido en el árbol:



Poda del árbol

No todos los caminos son posibles

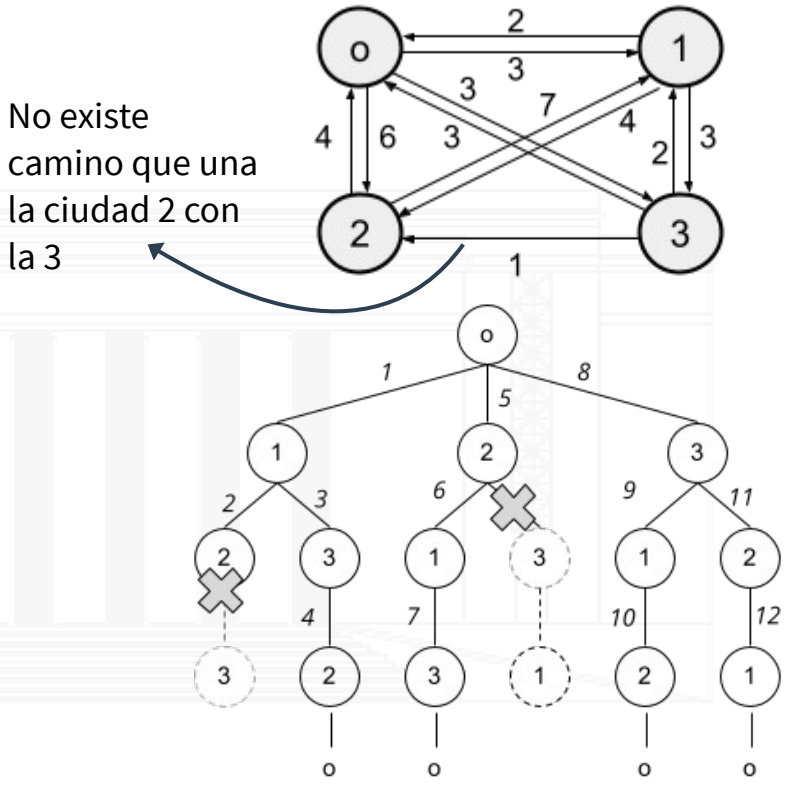
Si no existe un camino que une dos ciudades a y b podemos podar cualquier circuito que lo utilice

La función límite verifica la existencia de un camino que una a las ciudades

En caso de ausencia, poda el nodo y desestima la inspección de todas las ramas del árbol que las contiene

Si el grafo tiene una cantidad importante de nodos no comunicados

La poda reduce considerablemente el árbol de estados.



Backtracking - Pseudocódigo

```
Sea C[x,y] el costo de ir de la ciudad x a la y
Sea A[y] la lista de ciudades adyacentes de la ciudad y.
Sea camino el recorrido realizada hasta el momento
Sea minimoCosto el costo del camino minimo encontrado
Sea minimoCamino el circuito de menor costo encontrado

camino={o}.
minimoCosto=infinito
minimoCamino={}

Backtrack(camino)
```

Backtracking – Pseudocódigo (II)

```
Backtrack (camino):
     Sea x la ultima ciudad visitada.
     Si longitud del camino es n-1
          Si la ciudad o se encuentra en A[x]
               Agregar al final de camino la ciudad o
               Sea costoCamino la suma de los costos de camino
               Si costoCamino < minimoCosto
                    minimoCosto = costoCamino
                    minimoCamino=camino
               Remover o de camino
     Sino
          Por cada ciudad y en A[x] no visitada previamente excepto “o”
               Agregar y a camino
               Backtrack(camino)
               Quitar  y de camino
```

Complejidad Temporal

En el peor de los casos no es posible podar nodos del arbol

Debemos recorrer cada uno de los nodos del árbol con una complejidad $O\left(\sum_{i=1}^{n}\prod_{j=1}^{i}(n+1)-i\right)$

En los nodos en el peor de los casos debemos hacer un trabajo O(n)

Para obtener los siguientes nodos posibles de visitar

Para calcular el costo del circuito

La multiplicación de ambas complejidades nos brinda la complejidad temporal del algoritmo.

Complejidad Espacial

Si utilizamos una implementación recursiva

Por cada llamado en profundidad en el árbol incluimos el consumo de memoria adicional

La memoria utilizada

Es proporcional a la profundidad máxima de la recursión generada

Para el árbol de permutaciones la profundidad máxima es “n”.

En cada nivel de profundidad

Se agrega una nueva ciudad al vector

Se realizan cálculos que requieren O(1) de almacenamiento

Por lo que la complejidad espacial es O(n)