Análisis y Diseño de Algoritmos.

Sesión 11. 4 de Noviembre de 2015.

Maestría en Sistemas Computacionales.

Por: Hugo Iván Piza Dávila.

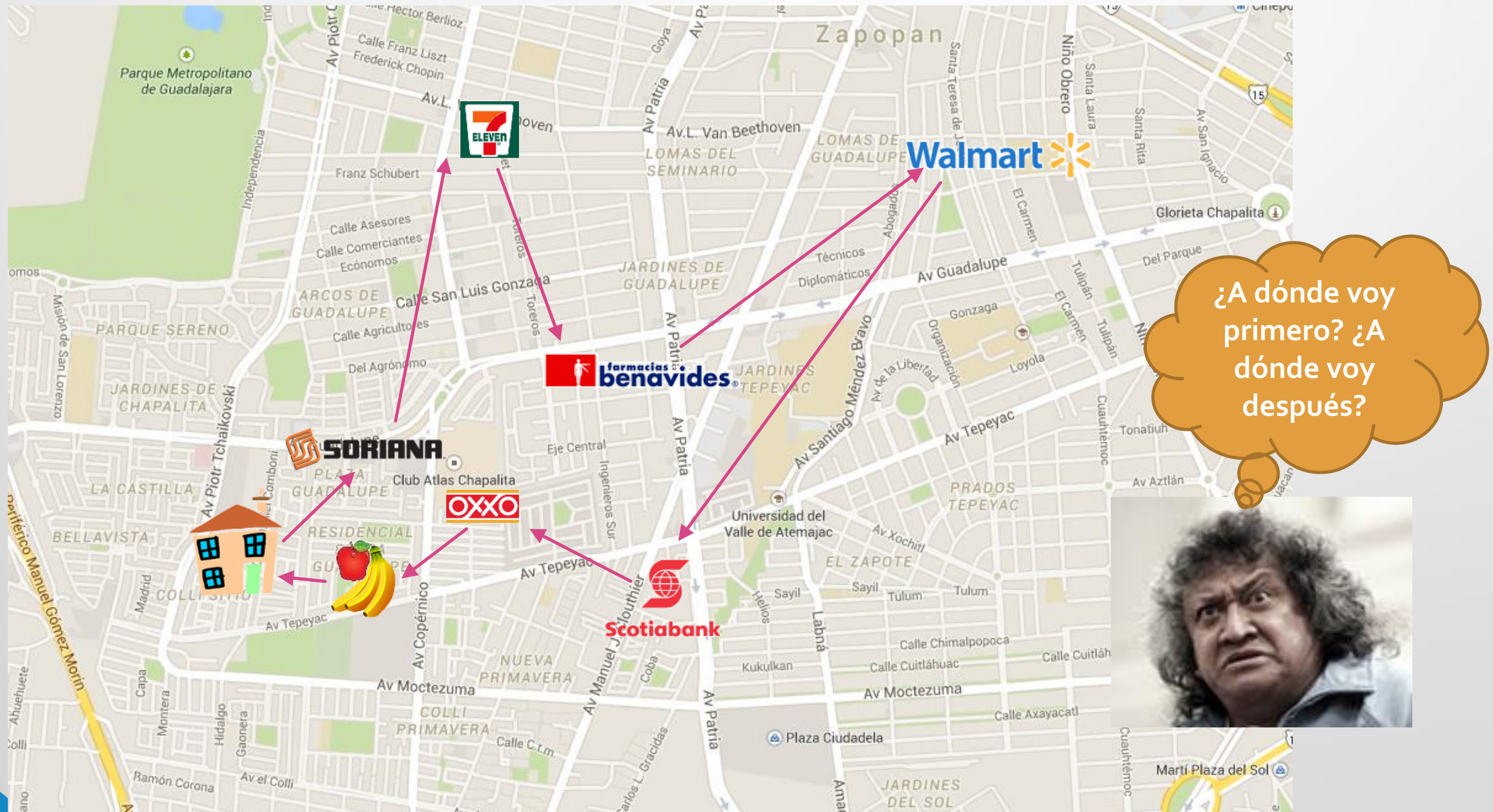
¿Qué veremos hoy?

- El Problema del ~~Agente~~ Mandilón Viajero.
- Optimización combinatoria.
- Búsqueda exhaustiva.
- Backtracking para el Mandilón Viajero.
- El problema de las 8 Reinas.
- Backtracking para el Problema de las N Reinas.

El problema del Mandilón Viajero

- Un sábado cualquiera en la tarde, a un marido cualquiera se le ha encomendado una misión típica:
  1. Comprar el cereal en Walmart porque sólo ahí lo venden.
  2. Comprar la carne en Soriana porque sabe mejor que la de Walmart
  3. Comprar las manzanas y los plátanos en la frutería del jardín porque están más frescas.
  4. Comprar los pañales del bebé en las Farmacias Benavides.
  5. Comprar la botana en el Oxxo y el café en e 7-Eleven.
  6. Sacar dinero del cajero para el gasto de la próxima semana.
- Pero tiene que regresar a la casa lo más pronto posible porque … ¡¡ya va a empezar el fútbol!!

El problema del Mandilón Viajero

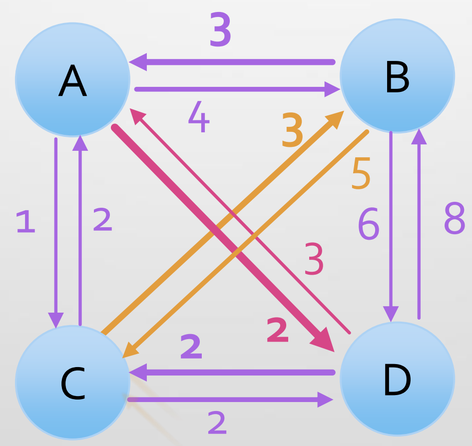


El problema del Mandilón Viajero

- En este problema, es claro que desde cualquier lugar J puedes llegar a cualquier otro lugar K con un tiempo mínimo T(J, K).
- Considerando que hay calles de un sentido, vueltas a la izquierda con semáforos, vueltas a la derecha continuas, rutas de camiones, topes, es muy probable que: $T(J, K) \neq T(K, J)$.
- Por tanto, el escenario se puede modelar mediante un grafo completo dirigido y ponderado.
- El problema se puede describir de la siguiente manera:
  - Encontrar un camino que parta de un nodo N, visite a todos los otros nodos del grafo y regrese a N con la menor duración: la suma de los pesos de los arcos visitados sea mínima.

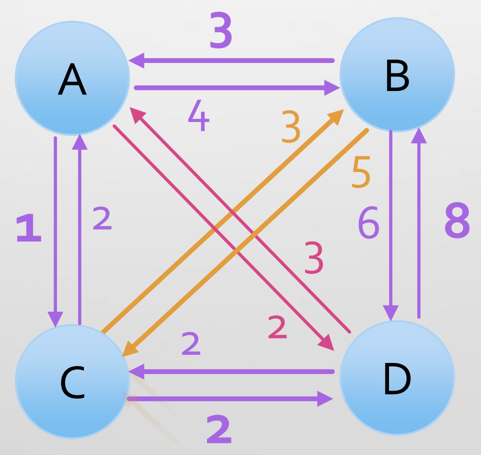
El problema del Mandilón Viajero

- En el grafo, si N = A, el mínimo camino es ADCBA.
- El tiempo total del mínimo camino es: 2 + 2 + 3 + 3 = 10.



El problema del Mandilón Viajero

- Obsérvese que elegir el arco que sale del nodo actual con el menor peso no nos garantiza la solución óptima.
  - La solución sería: ACDBA. Tiempo total = 1 + 2 + 8 + 3 = **14**
  - Un algoritmo voraz no daría la solución correcta.



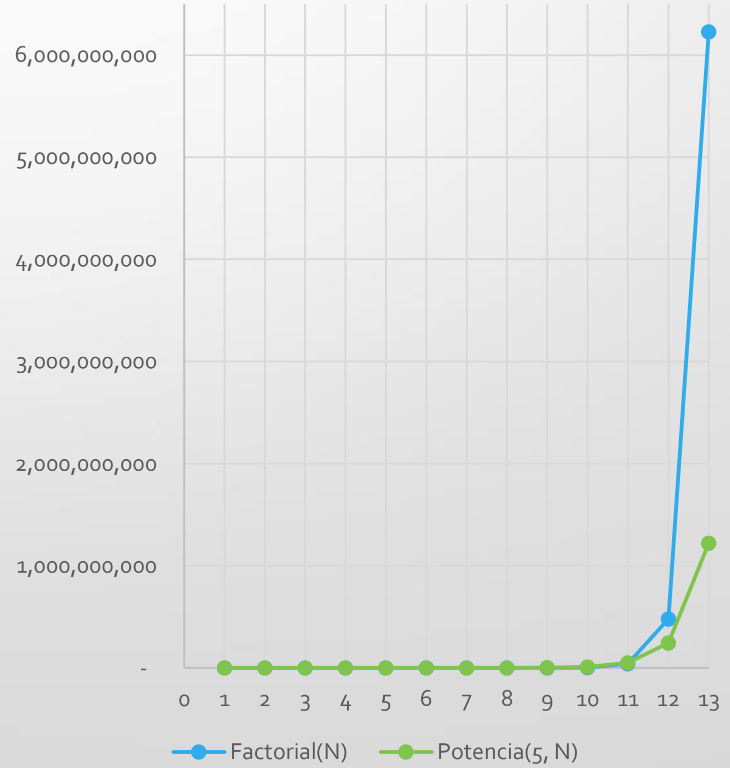
Optimización combinatoria

- Por lo anterior, se tienen que probar todas las posibilidades.
  - Cada posibilidad es una permutación de N – 1 elementos.
- Este problema pertenece al campo de la optimización combinatoria.
  - Encontrar una permutación que minimice el tiempo total.
- ¿Cuántas permutaciones existen en el grafo anterior?
  - ABCDA, ABDCA, ACBDA, ACDBA, ADBCA, ADCBA
- Para grafos no dirigidos se reduce a la mitad:
  - ABCDA = ADCBA, ABDCA = ACDBA, ACBDA = ADBCA

Optimización combinatoria

- Sea N el número de lugares por visitar (nodos del grafo – 1), el número de caminos posibles es: N!
  - *Número de permutaciones sin repetición.*

N	Caminos
1	1
2	2
3	6
4	24
5	120
	…
12	479,001,600
13	1,220’703,125



Optimización combinatoria

- El problema es claramente del tipo No Polinomial (NP).
  - Su complejidad no se puede explicar con una ecuación explícita en dónde N participe sólo en sumas, restas, multiplicaciones y/o logaritmos.
    - $N * f(N - 1)$ es implícita porque depende de otro valor para N.
    - Una aproximación explícita: $N! \approx N^{0.7N}$
    - Ecuación de tendencia de Excel: $0.019e^{1.926N}$
- Al crecimiento acelerado en el número de caminos posibles conforme se le denomina explosión combinatoria y es inherente a los problemas de optimización combinatoria.
- Un problema NP-Completo es aquél para el cual lo más seguro es que no exista una solución con complejidad Polinomial.

¿Cómo atacaremos este problema?

1. Algoritmo de búsqueda exhaustiva
   - Construye y analiza todos los caminos posibles.
   - Recursivo en su forma natural.
2. Algoritmo de búsqueda más inteligente
   - Backtracking.
   - Construye sólo caminos potenciales.
   - Si un camino no tiene futuro, ya no se explora por ahí.
   - Más eficiente que el anterior. Pero sigue siendo exponencial.
3. Heurística
   - Método evolutivo de la optimización computacional.
   - Genera una buena solución en un tiempo polinomial.
   - No ofrece garantías de encontrar la mejor.

Búsqueda Exhaustiva

- Sea N el número de nodos del grafo.
- Se crea un *bosque* de soluciones tal que los nodos internos almacenan caminos parciales y las hojas almacenan caminos completos.
- Una solución consta de:
  - Un arreglo de N – 1 índices de los nodos involucrados en el camino.
  - Un arreglo de N nodos involucrados en el camino como *booleans*.
- Sea 0 el índice del nodo inicial. Se construyen tres soluciones al inicio:

T,T,F,F	T,F,T,F	T,F,F,T
1, ?, ?	2, ?, ?	3, ?, ?

Búsqueda Exhaustiva

- Cada solución parcial con K elementos conocidos da lugar a la construcción de N – K nuevas soluciones.
  - Se debieron crear 6 arreglos en total, no 15.

T,T,F,F
1, ?, ?

T,F,T,F
2, ?, ?

T,F,F,T
3, ?, ?

T,T,T,F
1, 2, ?

T,T,F,T
1, 3, ?

T,T,T,F
2, 1, ?

T,F,T,T
2, 3, ?

T,T,F,T
3, 1, ?

T,F,T,T
3, 2, ?

T,T,T,T
1, 2, 3

T,T,T,T
1, 3, 2
Calcular tiempo del camino 02310

T,T,T,T
2, 1, 3
Calcular tiempo del camino 02130

T,T,T,T
2, 3, 1
Calcular tiempo del camino 02310

T,T,T,T
3, 1, 2
Calcular tiempo del camino 03120

T,T,T,T
3, 2, 1
Calcular tiempo del camino 03210

Ejercicio

- Definir de manera global las variables que almacenen la mejor solución encontrada y el tiempo correspondiente.
- Crear un método `calculateMinTime` que construya las N – 1 raíces:
  1. Recibe el índice del nodo inicial y el grafo como matriz de enteros.
  2. Crea los dos arreglos con la solución en común de los N – 1 árboles.
     - Para N = 4 y nodo inicial = 0,los arreglos son:{0, 0, 0} y {T, F, F, F}
  3. Para cada *k* de 0 a N – 1 tal que $k \neq$ *inicial*:
     a. Clonar el primer arreglo y guardar *k* en la primer posición.
     b. Clonar el segundo arreglo y guardar *true* en la *k*-ésima posición.
     c. Procesar la solución actual (invocar un método recursivo).
     - *No clonar los arreglos para el primer valor de k: modificar los valores, procesar la solución y regresar a los valores originales.*

Ejercicio

- Crear un método `processSolution`:
  1. Recibe: nodo inicial, grafo, nodo actual (*k*), y los dos arreglos.
  2. Si el camino actual ya está completo:
     a. Calcular el tiempo como la suma de los pesos de los arcos que unen a los nodos en el orden en que conforman la solución.
     b. Incluir el peso de los arcos que unen al nodo inicial con el camino.
     c. Si es necesario, actualizar la mejor solución encontrada y su tiempo.
  3. Si no, ejecutar un proceso semejante al paso 3 del método anterior, sobre los nodos no visitados.

Ejercicio

- Probar la precisión los métodos implementados con el grafo mostrado en diapositivas anteriores:

```
int graph[][] = {  {0, 4, 1, 2},
                   {3, 0, 5, 6},
                   {2, 3, 0, 2},
                   {3, 8, 2, 0}
                 };
```

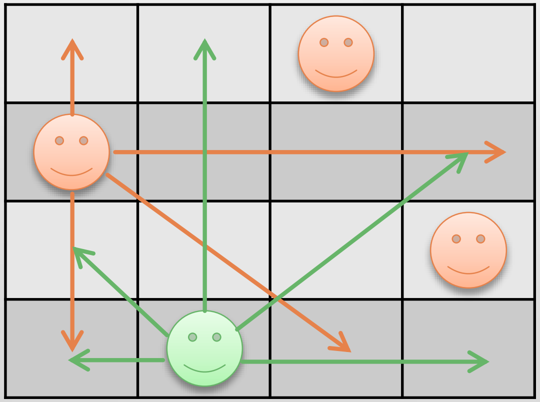
- Comprobar que la complejidad temporal del algoritmo de *búsqueda exhaustiva* es exponencial utilizando grafos aleatorios de $6 \leq N \leq 12$ nodos, contando el tiempo y el número de arreglos creados para cada N.

Backtracking

- ¿Y si el tiempo del camino actual (aún incompleto) es 15 y el mejor tiempo encontrado es 14?
- ¿Vale la pena seguir explorando ese camino?
- *Backtracking* consiste en ***cortar*** el espacio de búsqueda cuando un camino (que generará muchas alternativas) ya no es promisorio.
- Crea un método `fastProcessSolution` que primero calcule el tiempo que lleva el camino actual. Si ya superó al mínimo encontrado, finalizar la exploración por ahí.

El problema de las 8 reinas

- ¿Cuántas formas existen para acomodar ocho reinas en un tablero de ajedrez sin que se ataquen mutuamente?
- Las reinas atacan horizontal, vertical y diagonalmente.
- Se puede generalizar a N reinas en un tablero de N x N.
- Una solución para N =4:



Backtracking para N reinas

- Un algoritmo eficiente implica el uso de estructuras de datos que lo hagan eficiente.
- Este representación del problema de las N reinas haría ineficiente cualquier algoritmo.
- Y no es sólo por el desperdicio de espacio
  - $N^2 - N$ celdas no se usarían.

×	×	☺	×
☺	×	×	×
×	×	×	☺
×	☺	×	×

Backtracking para N reinas

- Backtracking consiste en terminar la exploración con soluciones parciales no promisorias.
- La estructura de datos matricial te permite tener dos reinas en la misma fila y/o en la misma columna.
  - Esta matriz promueve exploraciones no promisorias.

×	☺	☺	×
×	×	×	×
×	×	×	☺
×	×	×	☺

Backtracking para N reinas

- Eliminamos la posibilidad de tener dos reinas en la misma columna aplanando la estructura de datos.
  - Cada posición del arreglo guardará el número de fila $[0..N-1]$

×	×	☺	×
☺	×	×	☺
×	×	×	×
×	☺	×	×

→

1	3	0	1

Backtracking para N reinas

- Eliminamos la posibilidad de tener dos reinas en la misma fila garantizando que todos los números sean diferentes.
  - Esto se logra a través del algoritmo.

×	×	☺	×
☺	×	×	×
×	×	×	☺
×	☺	×	×

1	3	0	2

Backtracking para N reinas

- De esta manera obtenemos una estructura de solución semejante al problema del ~~agente~~ mandilón viajero.
- Utilizando esta estructura y garantizando que los índices sean diferentes, ¿toda solución parcial tiene futuro? si no, ¿cómo lo sabemos?

×	×	☺	×
☺	×	×	×
×	☺	×	×
×	×	×	☺

1	2	0	3

Backtracking para N reinas

- No se garantiza que tenga futuro.
  - Puede haber ataque en diagonal.
- En el ejemplo, las siguientes reinas está en la misma diagonal:
  - [0, 1], [1, 2]
  - [3, 2], [2, 3]

☺	×	×	×
×	☺	×	×
×	×	×	☺
×	×	☺	×

0	1	3	2

Backtracking para N reinas

- En la siguiente solución parcial (no promisoria), las siguientes reinas está en la misma diagonal:
    - [0, 0], [2, 2]
    - [3, 1], [2, 2]

☺	×	×	×
×	×	×	×
×	×	☺	×
×	☺	×	×

0	3	2	?

Backtracking para N reinas

- ¿Cuál es la fórmula para saber si están en diagonal?
  - La diferencia absoluta en las filas debe ser igual a la diferencia absoluta en las columnas de ambas reinas.
  - Recordar que el número de columna es el índice de la solución y el número de fila es el valor en tal índice.
- Implementar un método que reciba un valor de N > 1 y devuelva el número de formas diferentes que se pueden acomodar N reinas en un tablero de N x N de forma que no se ataquen entre sí.