PRÁCTICA 4: PROGRAMACIÓN DINÁMICA

Ahmed El Moukhtari Koubaa
Damián Marín Fernández
Jesús Martín Zorrilla
Eduardo Segura Richart

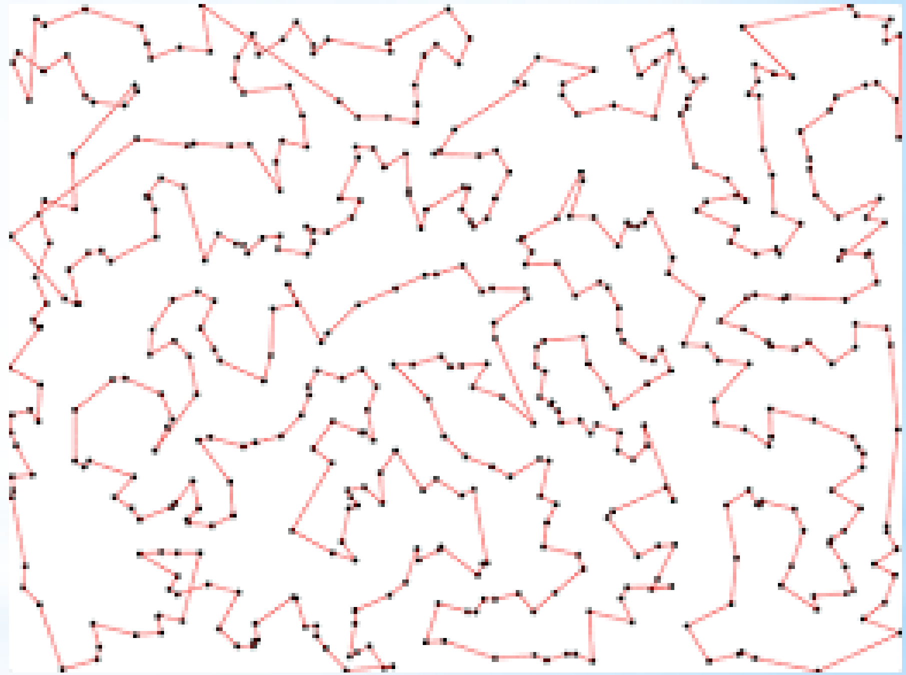
ÍNDICE

1.DESCRIPCIÓN DEL PROBLEMA

- Problema 1: Longest Common Subsecuence (LCS, Subsecuencia de caracteres más larga)

- Problema 2: Viajante de Comercio (TSP)

String A	a	c	b	a	e	d
String B	a	b	c	a	d	f



Descripción:

Este problema trata de en encontrar la ***subsecuencia*** común en ambas cadenas de izquierda a derecha y no necesariamente contigua.

Análisis:

- Es un problema ***multietápico***.
- Se cumple el ***POB*** ya que la búsqueda de una ***subsecuencia*** óptima incluye la búsqueda de una ***subsecuencia*** de la anterior también óptima.

Este problema también nos garantiza que cuando encontremos la solución será la óptima.

2.ESTUDIO DEL EJERCICIO GUIADO LCS

Análisis:

Se crea una matriz de enteros que se corresponde con una especie de producto de las cadenas a procesar. En esta matriz cada elemento nos indica si hay coincidencia o no entre el elemento de una cadena y la de la otra. Se recorren ambas cadenas (bucle anidado) y se van comparando, si estas coinciden se suma uno a los elementos que se encuentren en la esquina superior izquierda, es decir, en una diagonal. Finalmente, el último elemento de la matriz contiene el número de caracteres coincidentes.

		A	B	C	D	A
	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	2	2	2
B	0	1	2	2	2	2
D	0	1	2	2	3	3
E	0	1	2	2	3	3
A	0	1	2	2	3	4

LCS - "ACDA"

2.ESTUDIO DEL EJERCICIO GUIADO LCS

Análisis:

El algoritmo que hemos implementado es muy simple, recorre una cadena comparando todos sus elementos con la otra y si coinciden se añaden a una cadena resultado. Este algoritmo es de orden O(n^2) y para unas cadenas como son las del ejemplo de la solución (“ACBDEA” y “ABCDA”) nos da exactamente el mismo resultado (“ACDA”).
Atendiendo a los tiempos de ejecución y a los gastos de memoria son:

-Nuestro algoritmo 0.00001 s , 1524 KB
-La versión PD 0.08 s , 28756 KB

Como vemos el PD tiene peor eficiencia. Sin embargo, para un caso de ejecución mucho mayor que este vemos que los resultados son diferentes, lo cual nos indica que nuestro algoritmo no es correcto y que la solución óptima es la del algoritmo PD.

2.ESTUDIO DEL EJERCICIO GUIADO LCS

Conclusión:

Las técnicas de programación dinámica no son aplicables en el 100% de los casos. Cuando se aplican dichas técnicas se consiguen resultados óptimos. Vemos que en algunos casos y para algunos problemas en concreto mejoran bastante la eficiencia teórica y los tiempos de ejecución, sin embargo, nada nos garantiza que esto sea así para todos los problemas a resolver ni que la eficiencia obtenida sea "buena" (una cuadrática será mejor que una cúbica, pero peor que una lineal, hay que juzgar dependiendo del caso).

2.ESTUDIO DEL EJERCICIO GUIADO LCS

Objetivos:

- Recorrido de todas las ciudades una única vez regresando a la ciudad inicial

- Minimización de la distancia total recorrida

¿Cálculo de la distancia? → **EUCLÍDEA**

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

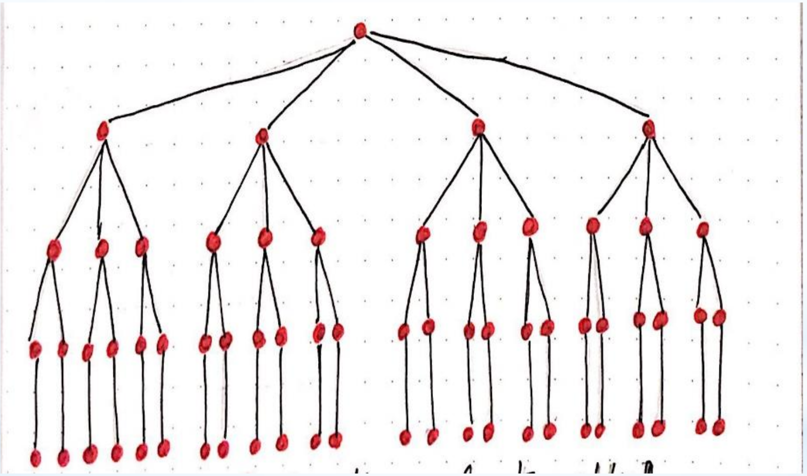
3.DESCRIPCIÓN DEL PROBLEMA: VIAJANTE DE COMERCIO

Posible solución → FUERZA BRUTA

- Eficiencia retórica factorial: **O(n!)**
- Para 20 ciudades, el computador más potente del mundo tardaría:

¡¡¡ 4 veces el tiempo de vida del universo !!!

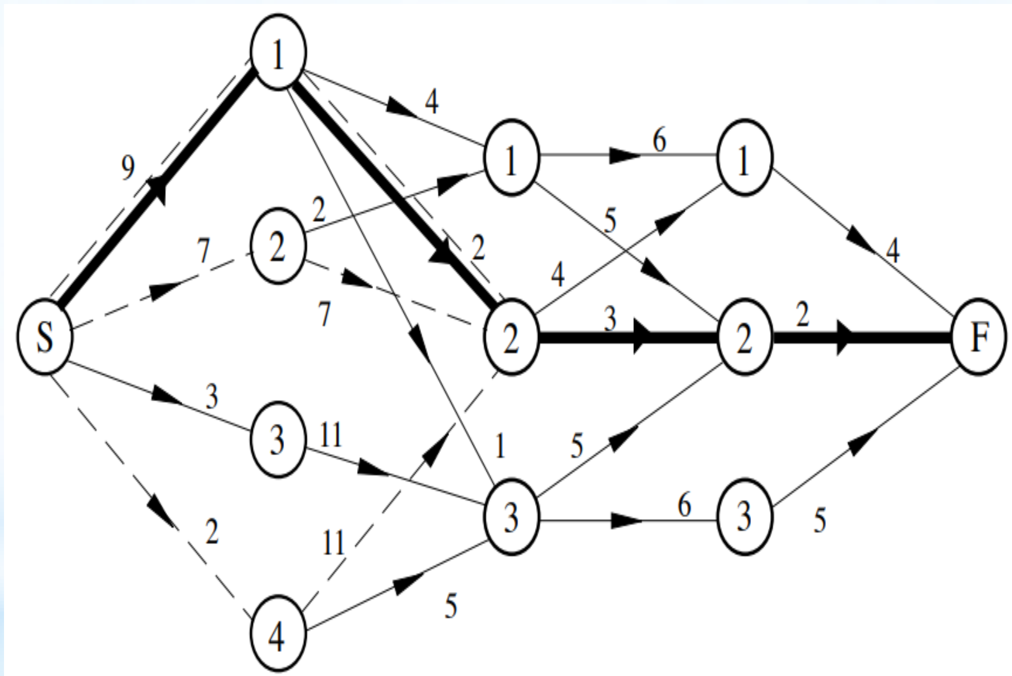
INVIABLE



3.DESCRIPCIÓN DETALLADA DEL ALGORITMO: VIAJANTE DE COMERCIO

<u>Análisis</u>:

- Es un problema *multietápico*.
- Cumple el *POB*.
- Para resolver un problema hay resolver sus *subproblemas*.



3.DESCRIPCIÓN DETALLADA DEL ALGORITMO: VIAJANTE DE COMERCIO

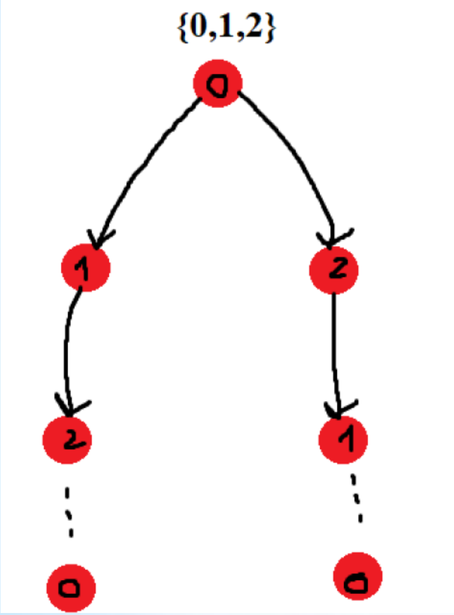
Estructuras de datos utilizadas:

MATRIZ 1 → Distancias entre ciudades

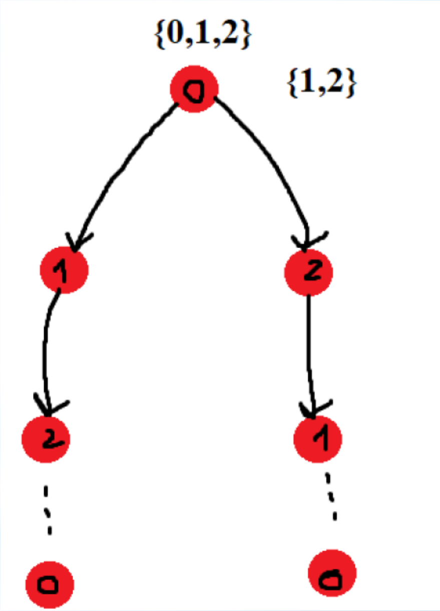
MATRIZ 2 $\left\{ \right.$ Distancias **mínimas** entre ciudades + Recorridos

3. DESCRIPCIÓN DETALLADA DEL ALGORITMO: VIAJANTE DE COMERCIO

Representación gráfica del problema



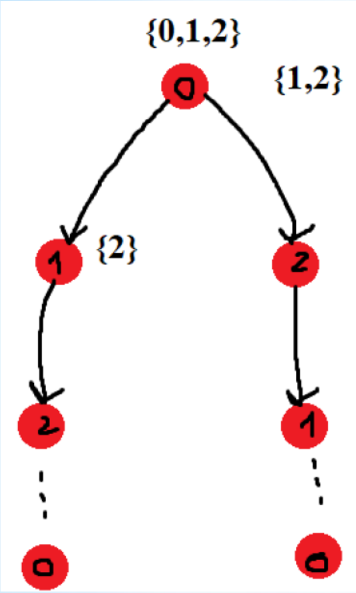
Partimos del conjunto de todas las ciudades y quitamos la que escojamos como ciudad de origen, en nuestro ejemplo 0.



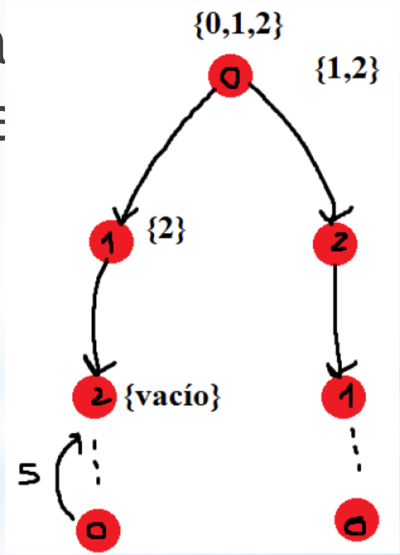
Nos quedan ahora dos ciudades que recorrer. Recorremos ese conjunto aplicando el algoritmo sobre cada una de las ciudades.

3.DESCRIPCIÓN DETALLADA DEL ALGORITMO: VIAJANTE DE COMERCIO

Representación gráfica del problema



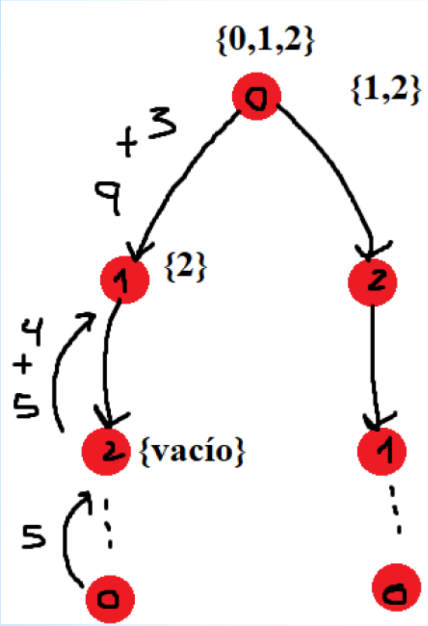
Seleccionando “1” como la siguiente ciudad y quitándola así del conjunto nos queda 2 como única ciudad a la que ir, volvemos a aplica el algoritmo nuevamente



Nos queda ahora un conjunto vacío y debemos volver a la ciudad de origen (“0”), haciendo esto devolvemos la iteración anterior el coste de ese camino (en nuestro ejemplo el coste es (2->0) = 5).

3.DESCRIPCIÓN DETALLADA DEL ALGORITMO: VIAJANTE DE COMERCIO

Representación gráfica del problema



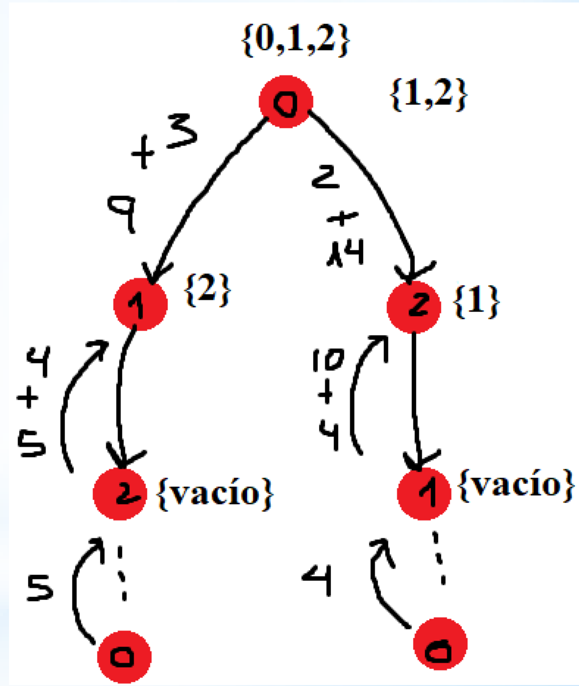
Vamos devolviendo las distancias mínimas a las ciudades de las que procedemos y sumamos para obtener el coste del camino.

Repitiendo lo mismo en la otra “rama” nos quedarían dos recorridos con dos distancias.

{0,1,2} = 12

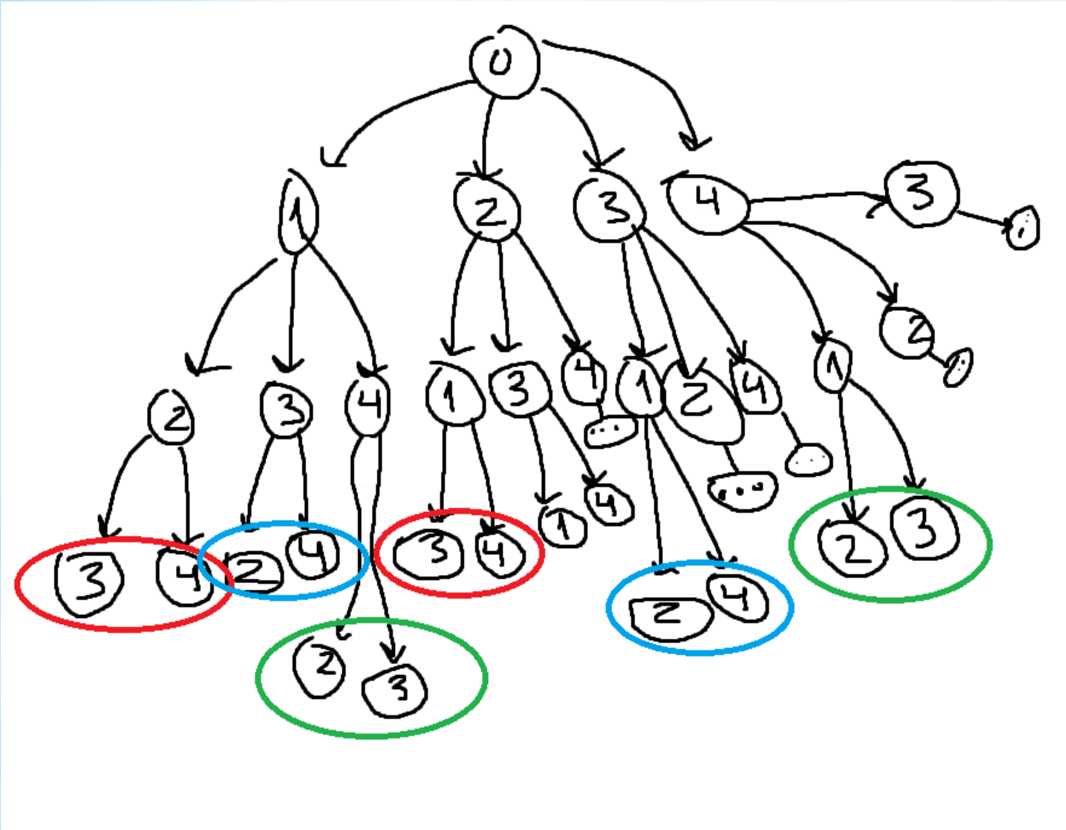
{0,2,1} = 16

Nos quedamos, una vez más, con el mínimo de estos dos recorridos.



3.DESCRIPCIÓN DETALLADA DEL ALGORITMO: VIAJANTE DE COMERCIO

Diferencia principal con la solución por fuerza bruta (O(n!))



Como vemos en la imagen, hay varios conjuntos ya resueltos que se repiten de los cuales ya sabemos cuál es la mejor opción.

Con la solución por fuerza bruta se repiten todos estos cómputos, pero con programación dinámica los realizamos una vez y los guardamos para reutilizarlos más tarde.

3.DESCRIPCIÓN DETALLADA DEL ALGORITMO: VIAJANTE DE COMERCIO

- El cálculo de las distancias (j, vacío) requiere n-1 consultas de la matriz.
- El cálculo de todas las distancias (j, conjunto de ciudades) de manera que 1 ≤ |ciudades| = k ≤ n -2
- El cálculo de las distancias (1, ciudades\{j}): n-1 sumas.

$$\Theta\left(2(n-1)+\sum_{k=1}^{n-2}(n-1)k\binom{n-2}{k}\right)=\Theta\left(n^2 2^n\right)$$

3.EFICIENCIA TEÓRICA: VIAJANTE DE COMERCIO

Un conjunto no se puede comparar de manera directa con otro ni hacer la búsqueda, inserción de elementos O(1).

Se han llevado a cabo dos implementaciones:

1) Se usan estructuras de datos complejos **(versión ineficiente)**
2) Se usa una codificación del conjunto para hacer las operaciones descritas anteriormente O(1) **(versión eficiente)**

$$\Theta\left(2(n-1)+\sum_{k=1}^{n-2}(n-1)k\binom{n-2}{k}\right)=\Theta\left(n^2 2^n\right)$$

3.EFICIENCIA TEÓRICA: VIAJANTE DE COMERCIO

Versión ineficiente: **O(n^3*2^n)**

```
double TSP(const vector<vector<double>> & L, int ciu_actual, int vistas, vector<vector<double>> & min_path){
  double minimo;
  if (vistas != ((1 << L.size()) - 1)){
    if (min_path[ciu_actual][vistas] == INT_MAX){        O(1)
      double distancia = minimo = INT_MAX;
      const int size = L.size();
                                                                   O(n)
      for (int i = 0; i < size; ++i){
        if (i == ciu_actual || (1<<i) & vistas) continue;
        distancia = L[ciu_actual][i] + TSP(L, i, vistas|(1<<i), min_path);   O(2^n)
        if (distancia < minimo) minimo = distancia;
      }

      min_path[ciu_actual][vistas] = minimo;      O(1)
    }
    else {                                         O(1)
      minimo = min_path[ciu_actual][vistas];
    }
  }
  else {
    minimo = L[ciu_actual][0];     // Ya fueron visitadas todas las ciudades enteras      O(1)
  }

  return minimo;
}
```

3.EFICIENCIA TEÓRICA: VIAJANTE DE COMERCIO

Versión eficiente: **O(n^2*2^n)**

```
double TSP(const vector<vector<double>> & L, int ciu_actual, int vistas, vector<vector<double>> & min_path){
  double minimo;
  if (vistas != ((1 << L.size()) - 1)){
    if (min_path[ciu_actual][vistas] == INT_MAX){
      double distancia = minimo = INT_MAX;
      const int size = L.size();

      for (int i = 0; i < size; ++i){
        if (i == ciu_actual || (1<<i) & vistas) continue;
        distancia = L[ciu_actual][i] + TSP(L, i, vistas|(1<<i), min_path);
        if (distancia < minimo) minimo = distancia;
      }

      min_path[ciu_actual][vistas] = minimo;
    }
    else {
      minimo = min_path[ciu_actual][vistas];
    }
  }
  else {
    minimo = L[ciu_actual][0];      // Ya fueron visitadas todas las ciudades enteras
  }

  return minimo;
}
```

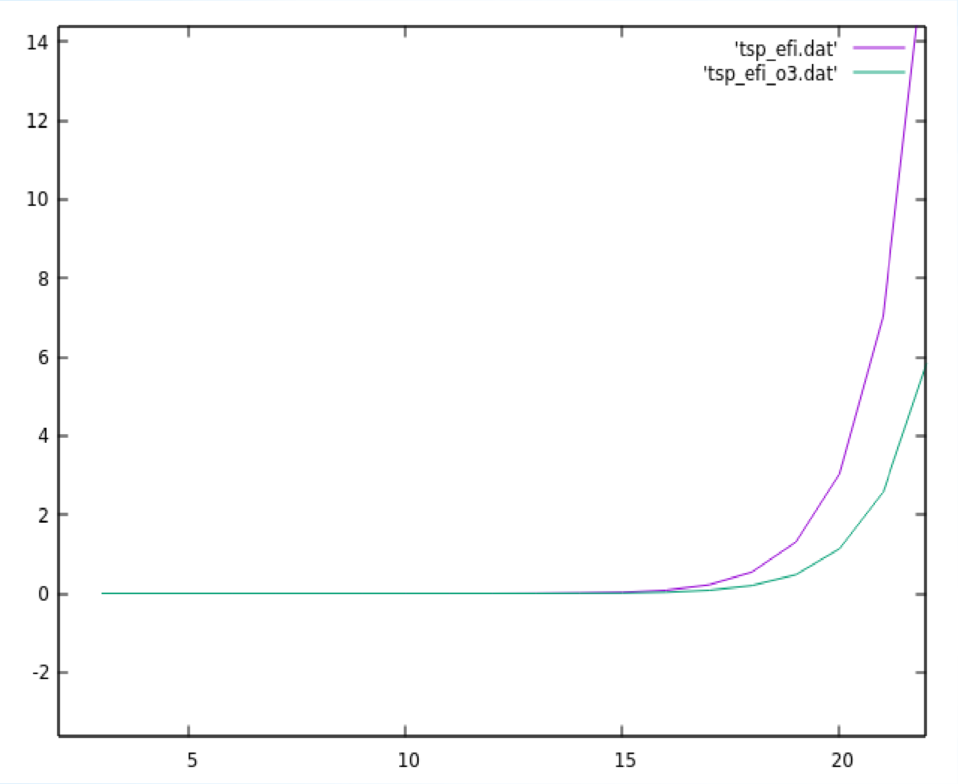
$O(1)$ $O(2^n)$ $O(n)$ $O(1)$ $O(1)$ $O(1)$

3.EFICIENCIA TEÓRICA: VIAJANTE DE COMERCIO

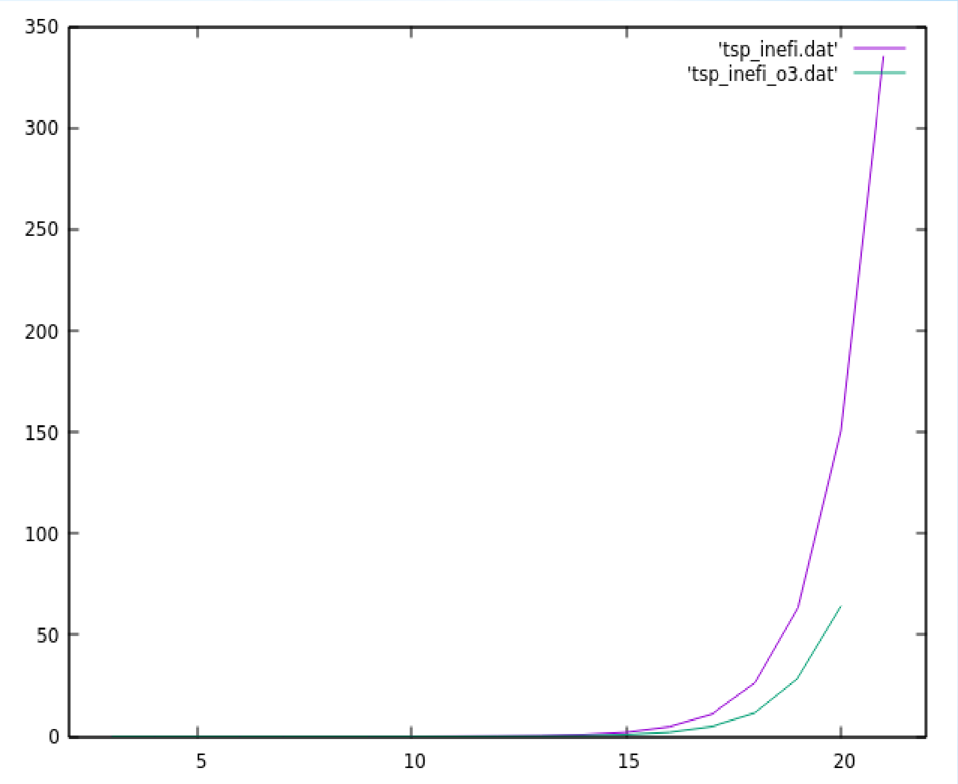
Tamaño	Tsp_efi (seg)	Tsp_efi_o3(seg)	Tsp_inefi(seg)	Tsp_inefi_o3(seg)
3	1.7e-06	1.6e-06	7.79e-05	7.21e-05
4	2.3e-06	1.9e-06	9.49e-05	4.62e-05
5	4.9e-06	3.2e-06	0.000144	6.44e-05
6	1.01e-05	5.5e-06	0.0003068	0.0001704
7	2.33e-05	1.13e-05	0.0008113	0.0003957
8	0.0001172	5.03e-05	0.0028611	0.0009518
9	0.0001429	5.99e-05	0.0067731	0.0029404
10	0.0004014	0.0001311	0.0179408	0.0070537
11	0.0008452	0.0003039	0.0475733	0.0184985
12	0.0020924	0.0009809	0.125775	0.0504506
13	0.0050979	0.0018118	0.337929	0.129913
14	0.0126424	0.0040761	0.785438	0.331708
15	0.0308974	0.0103001	1.93137	0.828955
16	0.0830521	0.0290099	4.70658	1.96987
17	0.218223	0.0833021	11.144	4.81945
18	0.551045	0.204286	26.407	11.6401
19	1.30288	0.476054	63.401	28.6161
20	3.03347	1.1356	150.93	63.9061
21	7.01205	2.56695	335.021	142.84
22	16.5317	5.81844	697.51	323.97

3.EFICIENCIA EMPÍRICA: VIAJANTE DE COMERCIO

Versión eficiente

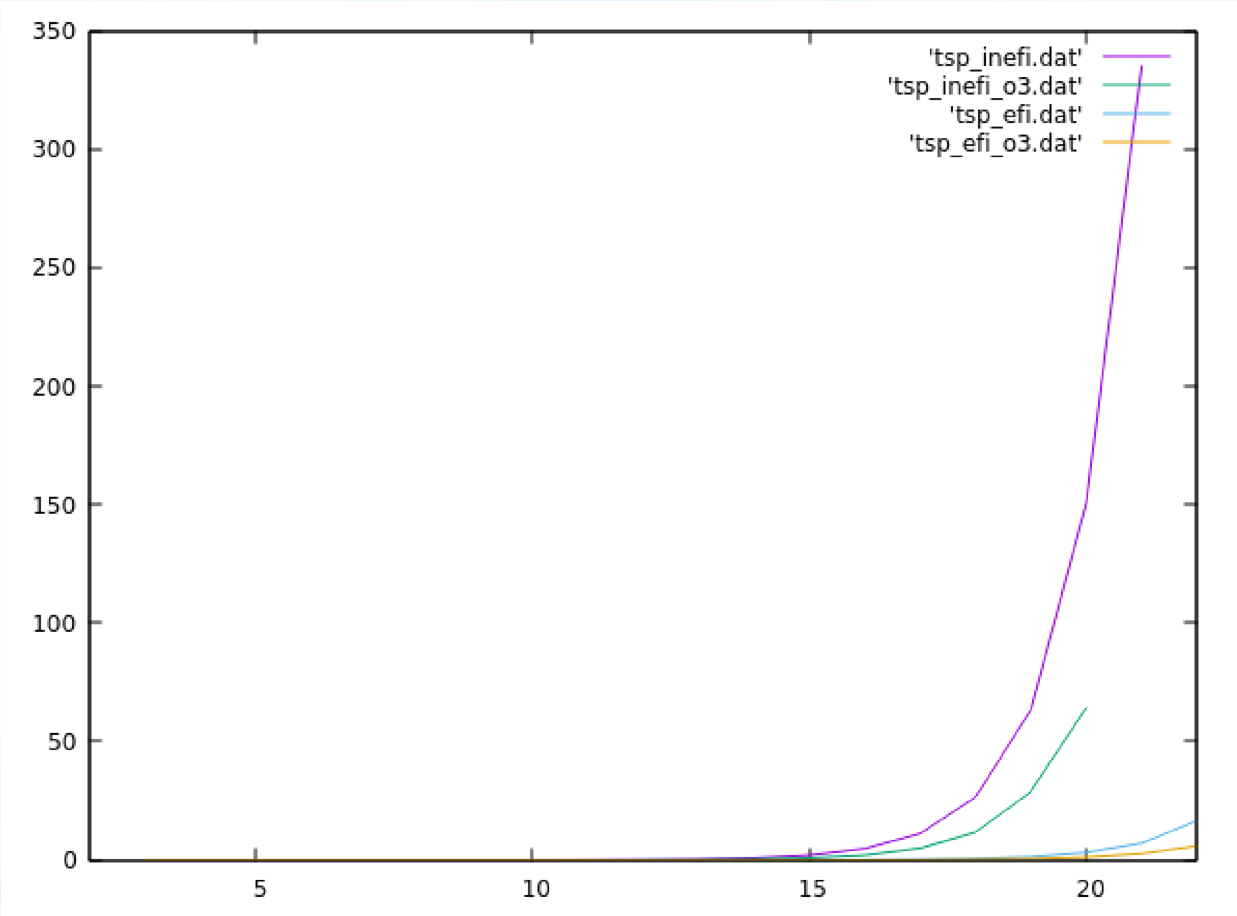


Versión ineficiente



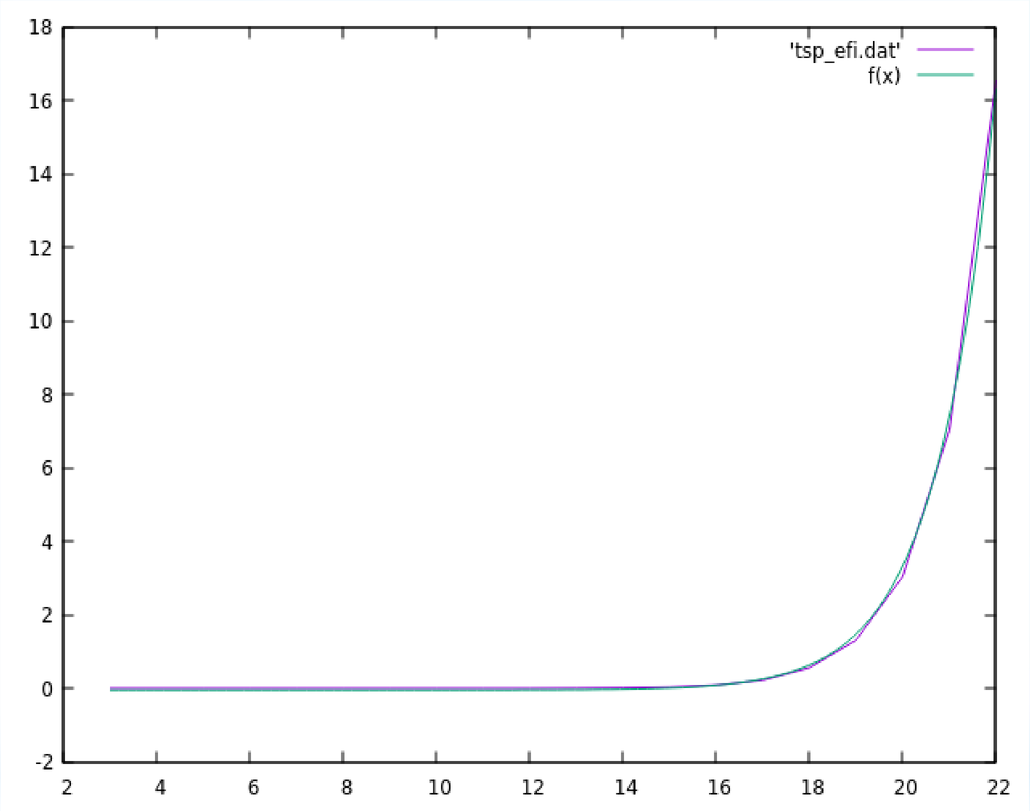
3.EFICIENCIA EMPÍRICA: VIAJANTE DE COMERCIO

Comparación global



3.EFICIENCIA EMPÍRICA: VIAJANTE DE COMERCIO

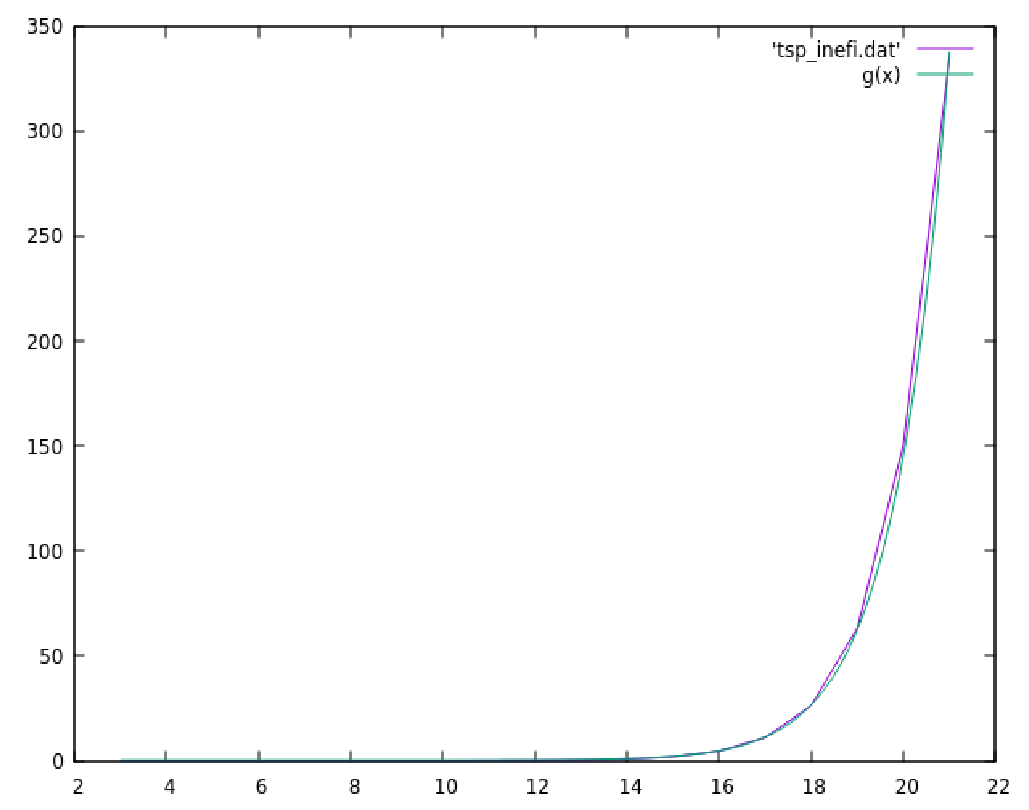
Versión eficiente



f(n) = 8.04861e-09 * (x^2) * (2^n) -0.0562579

3.EFICIENCIA HÍBRIDA: VIAJANTE DE COMERCIO

Versión ineficiente



f(n) = 1.73574e-08 * (x^3) * (2^n) + 0.218465

3.EFICIENCIA HÍBRIDA: VIAJANTE DE COMERCIO

Para 3 ciudades:

Recorrido mínimo: {0,1,2}

Costo = [(0,1) = 5.88 + (1,2) = 1.29 + (2,0) = 5.42] = 12.597

```
Distancias entre ciudades
0 5.88233 5.42148
5.88233 0 1.2919
5.42148 1.2919 0


coste minimo: 12.5957 0-->1-->2-->0
```

3.CASO DE EJECUCIÓN: VIAJANTE DE COMERCIO

Para 4 ciudades:

Recorrido mínimo: {0,1,2}

Costo = [(0,3) + (3,1) + (1,2) + (2,0)] = [3.348+4.487+1.29+5.42] = 14.549

```
Distancias entre ciudades
0 5.88233 5.42148 3.34819
5.88233 0 1.2919 4.48743
5.42148 1.2919 0 4.83011
3.34819 4.48743 4.83011 0


coste minimo: 14.549 0-->3-->1-->2
```

3.CASO DE EJECUCIÓN: VIAJANTE DE COMERCIO

Ulysses16.tsp:

TOUR_SECTION
1 14 13 12 7 6 15 5 11 9 10 16 3 2 4 8
-1

```
coste minimo: 73.9876 1-->14-->13-->12-->7-->6-->15-->5-->11-->9-->10-->16-->3-->2-->-->8
```

3.CASO DE EJECUCIÓN: VIAJANTE DE COMERCIO

PROGRAMACIÓN DINÁMICA

- Resultados óptimos.
- Reducción del tiempo de ejecución.
- Útil en problemas combinacionales con operaciones repetitivas.

- Alto consumo de memoria.
- Limitación del tamaño del problema a resolver.

4.CONCLUSIONES