

Algorítmica 2019/20

Práctica 4:

Programación dinámica

Índice:

1-. Problema del viajante de comercio

- 1.1-.Planteamiento del problema para PD.
- 1.2-.Algoritmo basado en programación dinámica

2.- Eficiencia

- 2.1-.Eficiencia teórica
- 2.2-.Eficiencia empírica
- 2.3-.Eficiencia híbrida
- 2.4-.Comparación de algoritmos

3-.Valoración sobre el trabajo de cada miembro

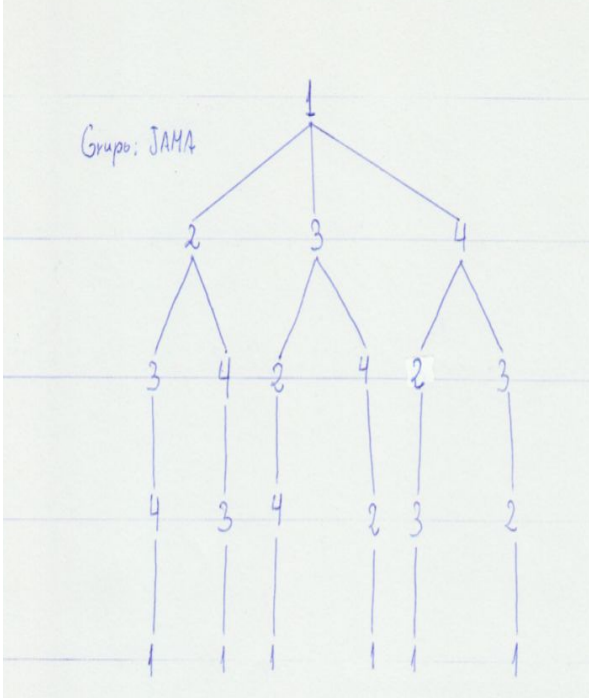
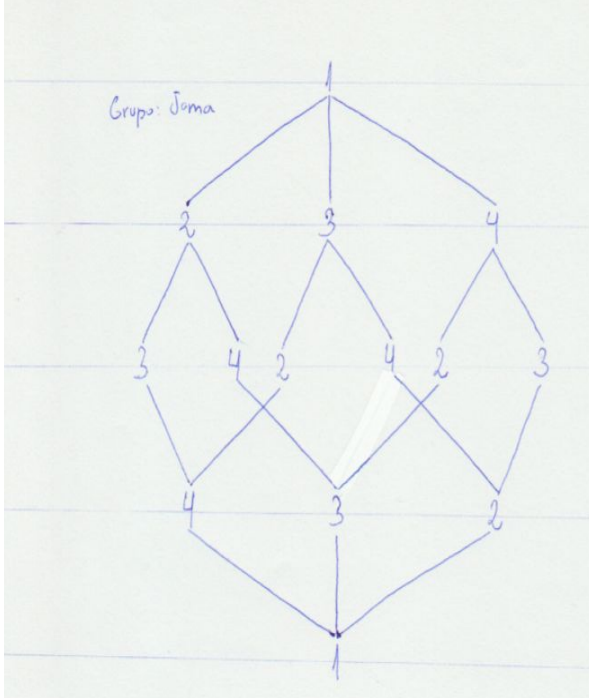
Realizado por:
Jorge Medina Romero
Alberto Robles Hernández
Ahmed Brek Prieto
Mohammed Lahssaini Nouijah

1-. Problema del viajante de comercio

1.1-.Planteamiento de la solución del problema para PD

Para entender el funcionamiento del algoritmo, primero es fundamental entender el principio de optimalidad de Bellman. Este principio dicta que “dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez, óptima”. Por lo que este algoritmo encontrará la ruta óptima, el ciclo hamiltoniano minimal de un conjunto de ciudades. Así, aplicamos recursividad sobre el conjunto de opciones inicial (creando subproblemas de tamaño $n - 1$, siendo así n -etápico) y seleccionando la única opción cuando sólo nos quede una alternativa, es decir, una ciudad por visitar. En esta situación estaremos considerando todos los posibles caminos. Ahora bien, vamos devolviendo las soluciones a los procesos padres, quienes irán recogiendo los subcaminos con mínimo coste de sus hijos, reuniendo así las subsoluciones óptimas para formar la solución óptima al problema inicial.

Además, en una memoria vamos almacenando los cálculos realizados anteriormente (guardamos las distancias de los caminos calculados) para, si se repiten en otra etapa, poder reutilizar las cifras calculadas sin necesidad de repetir cálculos y perder tiempo volviendo a calcular las mismas operaciones varias veces.

Fuerza Bruta	Programación Dinámica
$O(n!)$ 4 Ciudades $\rightarrow [1,2,3,4]$ Partimos de la ciudad 1	$O(n^2 * 2^n)$ 4 Ciudades $\rightarrow [1,2,3,4]$ Partimos de la ciudad 1
	
<p>Basado en fuerza bruta el algoritmo recorre todo el árbol, comprobando todas las permutaciones posibles y obteniendo la óptima. Como se puede ver repite el cálculo de subproblemas.</p>	<p>Basado en programación dinámica el algoritmo utiliza una memoria para guardar las soluciones de los distintos subproblemas (ya que el algoritmo en n-etápico como hemos visto anteriormente), por lo que ahorrar tener que realizar el mismo sub-problema más de una vez, también obtiene el camino óptimo.</p>

1.2-.Algoritmo basado en programación dinámica

Pseudocódigo del algoritmo

$$G(i,S) = \min\{ C_{ik} + g(k,S-\{K\}) \}$$

```
FUNCIÓN TSP_PD(ciudadActual,S)
    IF(numero de elementos de S == 1)
        DEVOLVER(Costes[ciudadActual][S[1]] + Costes[S[1]][0] )
    IF(MEMORIA[ciudadActual][S] ya calculada)
        CosteFina l= MEMORIA[ciudadActual][S]
    ELSE
        FOR(i : in S)
            CosteActual = Costes[ciudadActual][S[i]] + TSP_PD(S[i], S - {S[i]})
            IF(CosteActual < CosteFinal)
                CosteFinal = CosteActual
            END
        END
    END
    DEVOLVER(CosteFinal)
END
```

2-.Eficiencia.

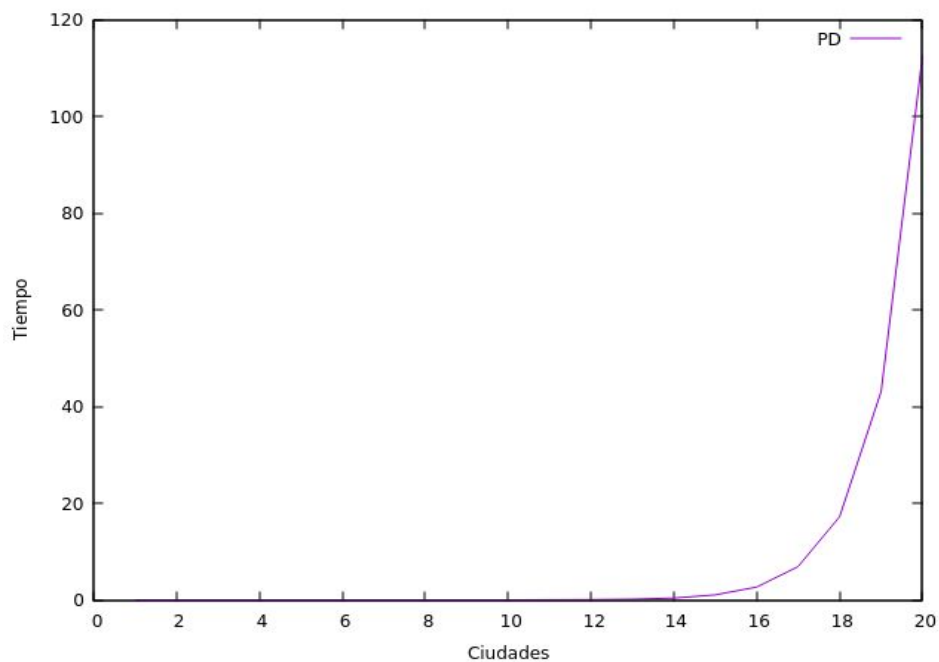
2.1-. Eficiencia teórica

La eficiencia de TSP mediante programación dinámica:

Hay como mucho, $O(n \cdot 2^n)$ subproblemas, cada uno de ellos tiene una eficiencia lineal $O(n)$, ya que cada vez que se llama recursivamente a la función está recorre el conjunto S . Por tanto, la eficiencia total de ejecución es $O(n^2 \cdot 2^n)$.

2.2-. Eficiencia empírica

Nº ciudades	Tiempo
1	4e-06
2	5e-06
3	1.5e-05
4	2.3e-05
5	5.1e-05
6	0.000251
7	0.000864
8	0.002209
9	0.002803
10	0.007335
11	0.0208
12	0.055581
13	0.145947
14	0.3817
15	1.02883
16	2.64001
17	6.86536
18	17.1848
19	43.0345
20	112.426

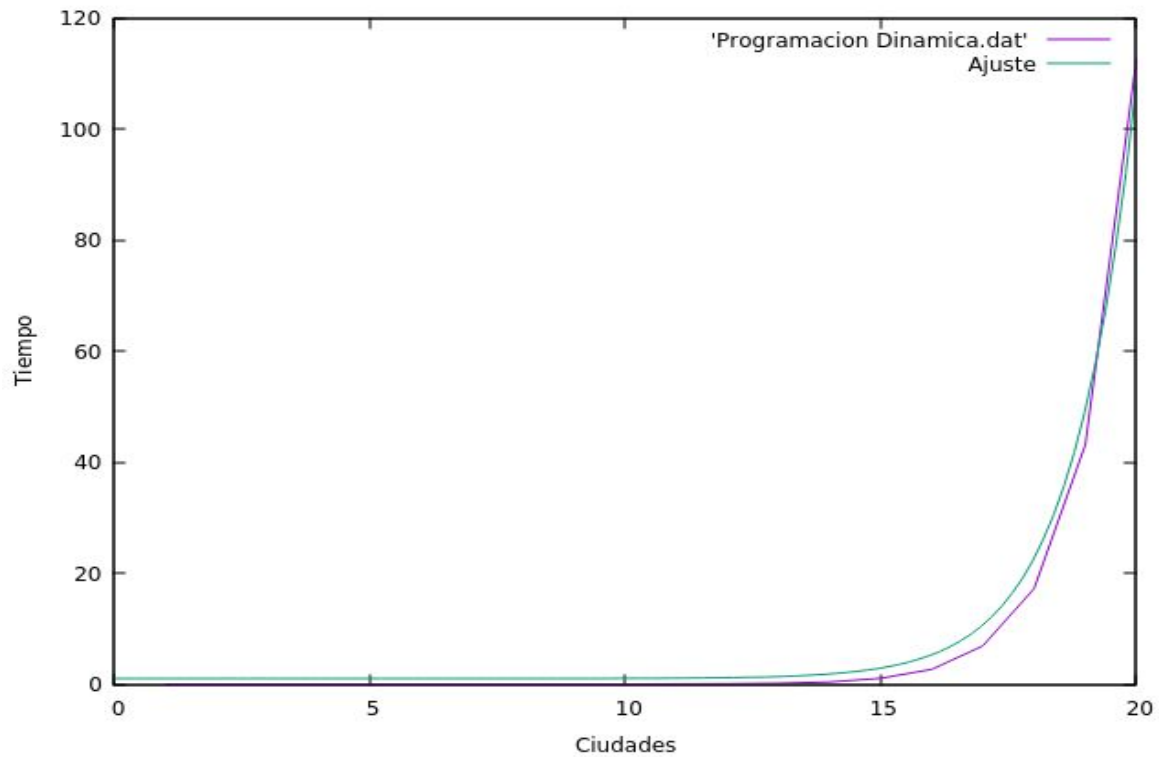


2.3.- Eficiencia híbrida

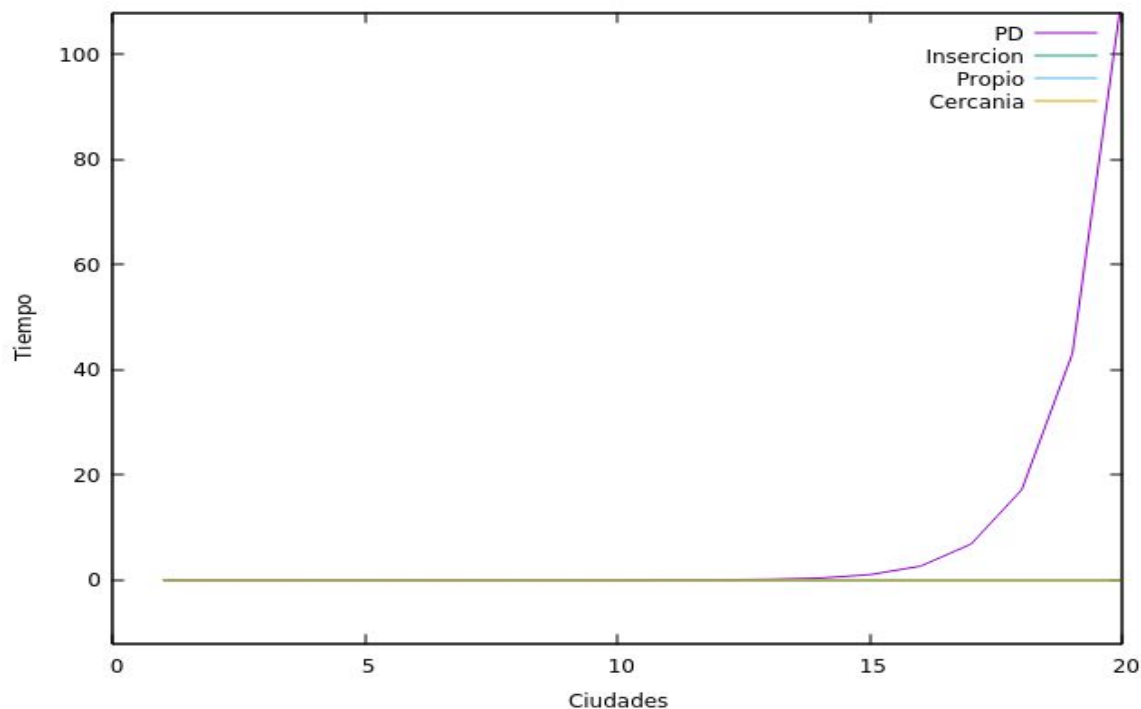
Ajustando los datos obtenidos en la eficiencia empírica a $n^2 * 2^n * a_0 + a_1$, ya que la eficiencia teórica de este algoritmo es de orden $O(n^2 * 2^n)$, obtenemos que la curva ajustada queda:

$$f(x) = x^2 * 2^x * 2.55102e - 07 + 1$$

$$R^2 = 0.9868457$$



2.4-.Comparación de algoritmos



Podemos ver cómo el algoritmo por programación dinámica es mucho más ineficiente que los demás, ya que crece de forma exponencial en comparación con la aproximación voraz tanto de inserción, cercanía o el algoritmo greedy propio que usamos en la práctica anterior.

Programación dinamica	$O(n^2 \cdot 2^n)$
Greedy cercania	$O(n^2)$
Greedy insercion	$O(n^3)$

Aunque el algoritmo con el enfoque de programación dinámica como hemos visto anteriormente es optimal, es decir, aunque puede llegar a ser muy ineficiente en tiempo y espacio, obtiene el ciclo hamiltoniano minimal.

Ocurre lo contrario que los algoritmos que siguen el enfoque Greedy, que son mucho más eficientes en tiempo y en espacio, pero no obtienen el camino óptimo como vimos en la práctica anterior.