

Informe Práctica 2 Análisis y Diseño de Algoritmos

Ichiro Hitomi

Diego Martín García

Descripción de la realización de la práctica

Hemos desarrollado el main y los tres algoritmos (explicados posteriormente), dentro de lo que cabe, en paralelo, en base a la documentación de la práctica y la entrada y salida supuesta que se ha descrito en ella.

Para el análisis de eficiencia hemos usado Excel, como en la práctica 1.

Algoritmos

Hemos seleccionado tres algoritmos que trataban con enfoques diferentes: *Algoritmo voraz* que se centra en dar la solución de manera óptima pero aproximada, *backtracking* para garantizar una solución mediante fuerza bruta y *dinámica* para obtener una solución exacta mientras que acortamos el tiempo de ejecución.

Cada algoritmo tiene como parámetros el número de contenedores, atributos por cada contenedor y la lista de contenedores que va a probar.

Algoritmo Voraz

Se ha creado el algoritmo voraz con tres métodos principales:

El método *algoritmo* es el principal que ejecuta el método principal del algoritmo voraz llamado por el main, dentro de éste, se ordenan todos los contenedores (necesarios para el algoritmo voraz), para cada contenedor buscamos un contenedor más grande que quepa hacia delante llamando a un método auxiliar *esCompatible*, si es válido lo añadimos a nuestra lista de contenedores válidos para ese intento y seguimos con ese intento usando el contenedor más grande como el pequeño. Al final de cada intento, si vemos que el intento actual tiene más contenedores que el intento con más contenedores previo, sustituimos.

El método privado *esCompatible* sirve para, dado un contenedor “pequeño” y otro “grande”, vemos si todos los atributos caben dentro de este.

Por último, el método *imprimirSalidaEstandar* sirve para imprimir la lista de contenedores de la solución a la pantalla.

Algoritmo de Backtracking

Se ha creado el algoritmo de backtracking está compuesto por 4 métodos principales.

El método empieza(), es el que es llamado desde el main y que inicia el proceso del algoritmo. Pero el algoritmo no empieza una sola vez, intenta iniciar una cadena distinta empezando desde cada contenedor posible para ver cuál genera la ruta más larga. Dentro de este, se crea una lista de booleanos que representa los contenedores visitados, y un arraylist con el camino actual tomado. Y por último se llama a buscarCaminos().

El método buscarCaminos() hace una búsqueda recursiva del mejor camino desde el contenedor actual en adelante, recibiendo como parámetros contenedorActual, caminoActual y lista de visitados. Lo primero que hace este método es guardar los datos para la recursividad actualizando variables y guardando la mejor solución hasta el momento. Tras esto, se comprueban candidatos a ser el siguiente contenedor, llamando al método esCompatible(), y cuando se encuentra uno, se vuelve a llamar a buscarCaminos(). Cuando ya no se encuentra un contenedor compatible se vuelve hacia atrás en la cadena para comprobar todas las demás combinaciones para ver si había un camino mejor que hemos omitido.

El método esCompatible() comprueba si un contenedor “cabe” dentro de otro recibiendo como parametros los 2 contenedores a comprobar y devolviendo un booleano que depende del resultado. Lo primero que se hace es comprobar que no sea él mismo. Después se ordenan los atributos (para comparar las dimensiones de menor a mayor independientemente de su posición) de una copia de los contenedores, para no cambiar el orden de los atributos del original. Y por último se compra que sean compatibles, es decir, que los atributos de uno sean menores que los del otro.

Y, por último, el método imprimirSolucion(), que literalmente imprime la mejor de las soluciones obtenidas.

Algoritmo Dinámico

Se ha creado el algoritmo de Programación Dinámica que está compuesto por 4 métodos principales.

El método empezar(), es el punto de entrada y el que gestiona el inicio de los cálculos. Este método recorre todos los posibles contenedores iniciales (desde 0 a k) llamando a calcularLongitud(). A medida que obtiene resultados, compara con el mejor hasta el momento y actualiza las variables globales mejorLongitudGlobal e inicioMejorCamino para recordar cuál fue el mejor punto de partida encontrado.

El método calcularLongitud() es el núcleo recursivo del algoritmo. Recibe como parámetro el contenedor actual “u”. Lo primero y más importante que hace es consultar la tabla memo, si el valor ya fue calculado previamente, entonces es distinto de -1, y es el valor de la longitud del camino, evitando así recálculos innecesarios en el futuro. Si no está calculado, explora todos los contenedores i, y si son compatibles, realiza la llamada recursiva. Durante este proceso, se guarda en la tabla siguiente cuál es el mejor contenedor siguiente para el contenedor actual. Finalmente, almacena el resultado

calculado en memo y lo devuelve. (En resumen, lo que hacen las tabla es almacenar datos de los cálculos en el momento que se hacen para no tener que hacerlos muchas veces después)

El método esCompatible() comprueba si un contenedor “cabe” dentro de otro recibiendo como parámetros los 2 contenedores a comprobar y devolviendo un booleano. La lógica es idéntica a la versión anterior del algoritmo backtracking: verifica que no sea el mismo contenedor, ordena los atributos de una copia de los datos (para comparar dimensiones de menor a mayor) y finalmente comprueba si todas las dimensiones del primero son menores que las del segundo.

Y, por último, el método imprimirSolucion() se encarga de mostrar la respuesta, reconstruyendo el camino. Utiliza la variable inicioMejorCamino para saber por dónde empezar y recorre la tabla siguiente (como si fuera una lista enlazada) para saltar de un contenedor a su sucesor óptimo, imprimiendo todos los índices hasta completar la cadena.

Main

Se ha creado un main simple para la ejecución de los algoritmos, donde pide por teclado el nombre del fichero y, después de obtener el número de contenedores y atributos por contenedor, hace una llamada a cada algoritmo, dando como parámetros los dos datos calculados previamente y la lista de contenedores, además de medir el tiempo que tarda cada uno de ellos.

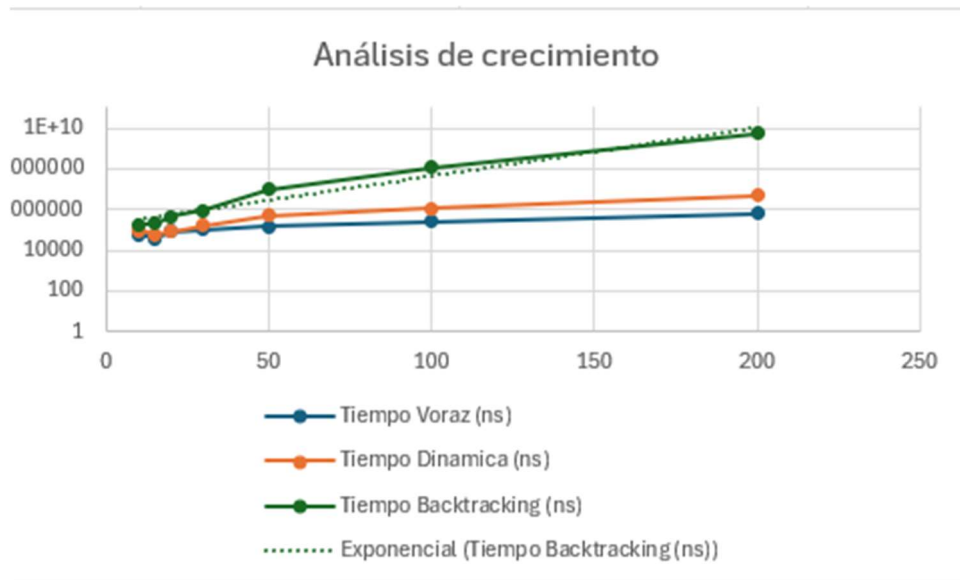
Análisis de Eficiencia

Como mencionamos previamente en el apartado de desarrollo de la práctica, hemos usado excel para hacer tablas y gráficas que nos permiten sacar las conclusiones finales.

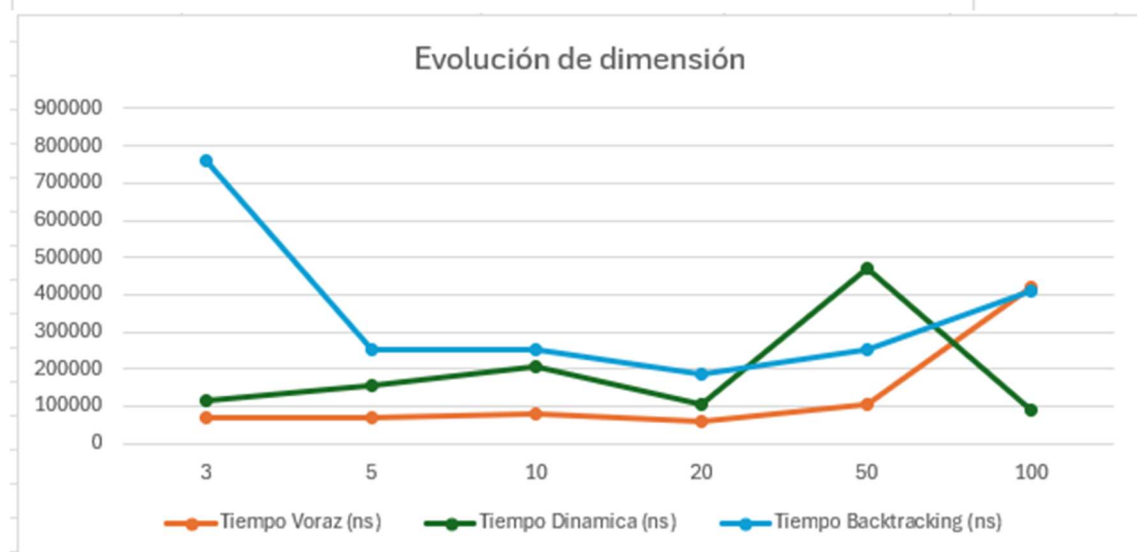
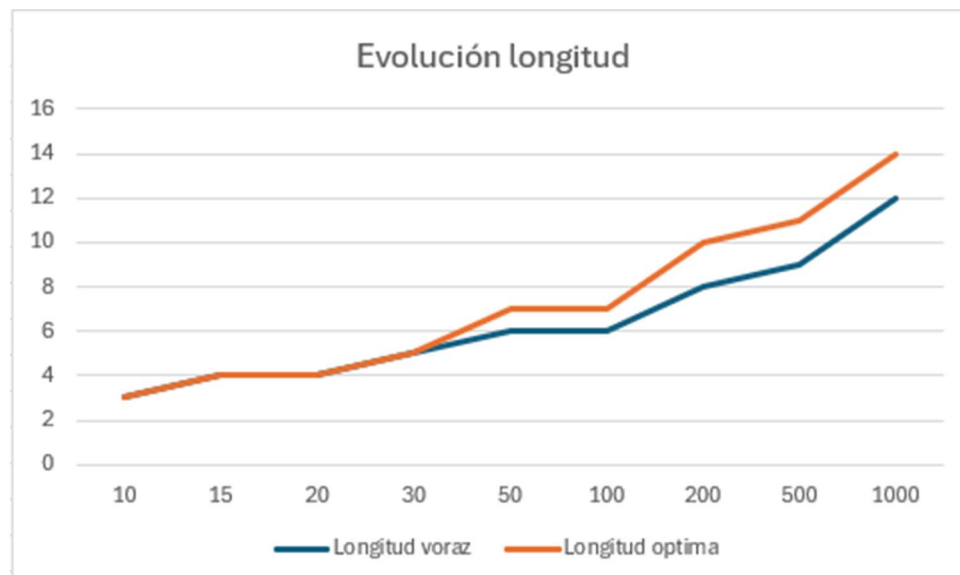
En concreto, hemos creado dos tablas, uno para sacar el crecimiento según el número de contenedores y un número de atributos fijos, otro para sacar el crecimiento según las dimensiones del contenedor, con el número de contenedores fijos.

Análisis crecimiento		N = 5				
K	Tiempo Voraz (ns)	Tiempo Dinamica (ns)	Tiempo Backtracking (ns)	Longitud voraz	Longitud optima	Porcentaje acierto voraz
10	58601	88043	185666	3	3	100%
15	39021	58708	210237	4	4	100%
20	78001	78080	459295	4	4	100%
30	107479	156207	937619	5	5	100%
50	146461	478597	9278481	6	7	86%
100	253958	1103595	116508313	6	7	86%
200	639080	5125654	5483385935	8	10	80%
500	2036266	30002638 -		9	11	82%
1000	8085053	108640808 -		12	14	86%

- Nota: Para K>500 no hemos analizado backtracking ya que el tiempo crecía demasiado



Nota: Hemos usado una escala logarítmica en el gráfico para que se observe mejor la evolución de los datos



Análisis dimension		K=20						
N	Tiempo Voraz (ns)	Tiempo Dinamica (ns)	Tiempo Backtracking (ns)	Longitud voraz	Longitud optima	Porcentaje acierto voraz		
3	68338	117320	762470	5	5	100%		
5	68411	153881	253905	3	3	100%		
10	78143	205169	254429	3	3	100%		
20	58625	107405	185474	2	2	100%		
50	107432	468299	254026	2	2	100%		
100	420274	87909	410426	1	1	100%		
Tiempo Dinamica (ns)	Tiempo Backtracking (ns)	nlogn	n*n	n^4	c=tVoraz/nlogn	c=tDinamica/n*n	c=tBack/n^4	
88043	185666	10	100	10000	5860.1	880.43	18.5666	
58708	210237	17.64136889	225	50625	2211.903184	260.924444	4.15282963	
78080	459295	26.02059991	400	160000	2997.6634	195.2	2.87059375	
156207	937619	44.31363764	900	810000	2425.41587	173.5633333	1.157554321	
478597	9278481	84.94850022	2500	6250000	1724.115195	191.4388	1.48455696	
1103595	116508313	200	10000	100000000	1269.79	110.3595	1.16508313	
5125654	5483385935	460.2059991	40000	1600000000	1388.682462	128.14135	3.427116209	
Media					2243.98626	214.7601844	2.847711779	

Con esta aproximación, ajustando las curvas, podemos decir que los algoritmos tienen como fórmulas:

Voraz: $2243.98 * \log(n)$

Dinámica: $214.76 * n * n$

Backtracking: $2.84 * n^4$

También hicimos pruebas con las variaciones del tiempo de ejecución con N con una K constante, sin embargo, como el impacto del tiempo es lineal y casi despreciable comparado a la evolución de K, podemos confirmar que el tiempo de ejecución varía sobre todo por el número de contenedores y no la comparación de atributos.

Uno de los problemas que observamos con el algoritmo voraz es que, a cambio de tener un rendimiento muy alto en comparación con los otros algoritmos, al ser un algoritmo que toma decisiones en base a la información que tiene en el momento sin tener en cuenta efectos futuros, vemos que no puede encontrar la lista de contenedores de mayor longitud posible cuando el número de contenedores aumenta, por lo que se puede considerar como una solución óptima hasta K=30.

Como explicamos previamente, el algoritmo de backtracking desde K=500 hace que aumente el tiempo de ejecución de forma muy considerable, que medirlo tardaría demasiado tiempo, por lo que se dejó de calcular.

El algoritmo dinámico es más lento que la voraz ya que tiene que rellenar una tabla entera, sin embargo, es mucho más rápida que el de backtracking ya que no le hace falta recalcular subproblemas que ha resuelto previamente, haciendo que sea un punto medio entre la solución óptima y tiempo de ejecución.

Conclusiones:

En base a nuestro análisis, podríamos decir que el algoritmo voraz es funcional para casos con pocos contenedores ya que los datos no se desvían, sin embargo, cuando aumenta el número de contenedores y los datos del algoritmo voraz empiezan a ser estimaciones, la mejor opción es el algoritmo dinámico, que es el punto medio entre optimización y una solución exacta.

Declaración de Uso de Inteligencia Artificial:

- Para la clase de programación dinámica, de la forma en que estaban planteados los códigos, no podía imprimir la solución del algoritmo desde el main (para que eso no contase en el tiempo de ejecución del algoritmo). Por lo que me ayude de "GEMINI" con el siguiente prompt: "dado este código, dame un método que imprima la salida sin ningún argumento en él:" junto con el código hasta ese momento de la clase "dinámica".

El resultado que me dio fue: el método imprimirSolucion() que está puesto en el código actual, además de la recomendación de cambiar las variables mejorLongitudGlobal e inicioMejorCamino a atributos de la clase en vez de variables de métodos concretos. Este último cambio junto con los necesarios para hacer el código funcionar de nuevo fueron hechos por mí.

- Al implementar el primer algoritmo voraz, se hizo una consulta a Gemini para saber la mejor forma de ordenar los datos iniciales, donde nos dio la respuesta de crear una clase Contenedor, donde su uso se aplicó para todos los algoritmos.

Estimación porcentual de reparto de trabajo:

Ichiro Hitomi: 50%

Diego Martín García: 50%

Bibliografía

- PDF Tema 2: Diseño de Algoritmos