Algoritmos y Estructuras de Datos III

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico 1

Segundo Cuatrimestre 2014

Integrante	LU	Correo electrónico
Ricardo Colombo	156/08	a@a.com
Federico Suarez	610/11	a@a.com
Juan Carlos Giudici	827/06	elchudi@gmail.com
Franco Negri	000/00	a@a.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Contenidos

Capítulo 1

Puentes sobre lava caliente

1.0.1. Introduccion

Cada participante de una competencia debe cruzar un puente dando saltos de tablon en tablon, teniendo en cuenta que pueden saltar una cantidad maxima de tablones de una sola vez. Sin embargo algunos de estos tablones estan rotos y se sabe de antemano cuales son los mismos. Se desea calcular la cantidad minima de saltos requerida para cruzar el puente.

1.0.2. Ejemplos y Soluciones

1.0.3. Desarrollo

Para la solucion de este problema recurrimos a la tecnica denominada programacion dinamica. Por el enunciado sabemos que tenemos n tablones, el participante puede saltar C tablones de una sola vez y cuales son las posiciones de los tablones rotos.Nos armamos dos arreglos donde cada posicion representa un tablon, en el primero guardamos 0 y 1 para indicar su estado (sano o roto respectivamente) llenando el mismo segun la entrada, llamemoslo puente, en el otro iremos guardando la cantidad minima de saltos para llegar a cada tablon,llamemoslo Distancias. Nuestro algoritmo ira recorriendo un tablon por vez, viendo si el mismo esta roto o no. En el caso de que el tablon este roto (segun indica nuestro arreglo puente) continuo al siguiente tablon. Para los primeros C tablones que esten sanos pondremos 1 en el arreglo Distancias, cuando el algoritmo se encuentre en el tablon i , con $C \leq i \leq n-1$, calculamos su cantidad minima de saltos de la siguiente manera:
- Buscamos el minimo entre i - C y i-1, en el arreglo de Distancias.
- Al minimo encontrado le sumamos uno y lo colocamos en la posicion i del arreglo Distancias.

Una vez completado el arreglo distancias debemos armar el recorrido, para el cual en primer paso buscamos dentro de las C ultimas posiciones del arreglo Distancias el minimo, llamemos j a su posicion. Luego agregamos este j a la solucion como el ultimo tablon. A partir de ahi en cada paso buscamos el minimo entre j-C y j-1, y lo agregamos adelante de nuestra solucion y reemplazamos el j con la posicion del nuevo minimo.

1.0.4. Demostracion

Demostraremos por induccion que en cada paso de este algoritmo obtenemos la minima cantidad de saltos posible para llegar a ese tablon:

P(i) = ."El valor guardado en la posicion i-1 del arreglo Distancias es la cantidad minima de saltos hasta el tablon i"

Caso base, P(i) con $0 \leq i < C$: El caso base son los primeros C tablones, donde C es la cantidad maxima de tablones que un participante puede saltar de una sola vez. En este caso la cantidad minima de saltos para cada tablon es trivialmente 1, ya que es el primer salto desde el punto de partida.

Paso inductivo, $P(n) \Rightarrow P(n+1)$ con $n \geq C$: Por hipotesis inductiva sabemos que tenemos la cantidad de saltos minima hasta el tablon n en el arreglo Distancias (entre las posiciones 0 y n-1). Si el tablon esta roto entonces la cantidad de saltos minima sera infinito. En el caso contrario, para calcular la minima cantidad de saltos para llegar al tablon n+1 revisamos los ultimos C tablones previos, y nos quedamos con el que requiera la menor cantidad de saltos para llegar hasta el, y llamaremos a esta cantidad Min. Entonces la

cantidad minima de saltos para llegar al tablon n+1 seria Min+1, y ahora veremos que efectivamente lo es. Sin Min+1 no fuera la cantidad minima de saltos para llegar al tablon n+1 entonces existe un tablon entre n-C y n-1 cuya cantidad minima de saltos para llegar hasta el es menor a Min, lo cual es absurdo porque seleccione al minimo. Puede pasar que exista un tablon entre n-C y n-1, distinto al seleccionado, cuya cantidad minima de saltos para llegar hasta el coincida con Min. En ese caso esta solucion es tan buena como la mia. Por lo tanto la cantidad de saltos minima para llegar al tablon n+1 es efectivamente Min+1.

Para armar el recorrido, realizamos lo ya descripto en la seccion de desarrollo y como siempre vamos tomando el minimo en cada paso recorriendo el arreglo Distancias hacia atras llegamos a una solucion optima, que como vimos previamente, puede haber mas de una.

1.0.5. Complejidad

1.0.6. Experimentacion

Capítulo 2

Horizontes lejanos

2.0.7. Introduccion

Se esta diseñando un software de arquitectura, para el cual es necesario que dado un conjunto de edificios representados como rectangulos apoyados sobre una base en comun, se devuelva el perfil definido en el horizonte.
Estos edificios vienen representados por tuplas de tres elementos que representan donde comienza el edificio, su altura y donde termina, de las cuales tenemos que ir tomando en cada momento donde comienza un edificio la altura maxima alcanzada en ese punto.

2.0.8. Ejemplos y Soluciones

Consideremos el siguiente ejemplo del problema:
$[< 3, 2, 5 >; < 1, 4, 2 >; < 4, 1, 6 >; < 6, 8, 10 >]$
Cada una de estas tuplas de tres elementos se indica donde comienza el edificio en la primera coordenada, su altura en la segunda y donde termina en la tercera coordenada.

Lo primero que hacemos es ordenar estas tuplas en orden creciente por lo que representa donde comienza el edificio (llamemosla pared izquierda), quedandonos de la siguiente manera:
$[< 1, 4, 2 >; < 3, 2, 5 >; < 4, 1, 6 >; < 6, 8, 10 >]$

Por otro lado ordenamos los edificios por la coordenada donde terminan (llamemosla pared derecha).
$[< 1, 4, 2 >; < 3, 2, 5 >; < 4, 1, 6 >; < 6, 8, 10 >]$

2.0.9. Desarrollo

Como mencionamos anteriormente, tenemos como datos de entrada la posición donde comienza y termina cada edificio y además su altura, con lo cual representaremos a los edificios con tuplas de 3 elementos (posición de inicio o pared izquierda, altura, y posición donde termina o pared derecha). Primero organizaremos a los edificios en dos arreglos, donde cada arreglo contendrá el total de edificios, uno con un orden ascendente según pared izquierda y el otro también con un orden ascendente pero según pared derecha. Además tendremos un conjunto en el que iremos agregando los edificios que çomiencenz quitando los edificios que "terminen", es decir, aquí estarán los edificios .ªctivos"(que .ªmpezaronz no "terminaron") en cada momento. La idea del algoritmo es ir recorriendo las posiciones (del eje x) en las que haya una o más paredes. En cada punto lo que haremos es agregar a mi conjunto de activos los edificios que en ese punto tengan su pared izquierda, o sea que estén çomenzando", y quitar a los edificios que allí tengan su pared derecha, o sea que estén "terminando". Una vez completada esta labor, buscaremos el edificio activo que tenga la altura máxima y nos lo guardaremos, llamémoslo Max. Se puede ver claramente que el borde superior de la silueta en un punto dado va a estar determinado por el edificio más alto que haya en ese punto, es decir, el edificio activo más alto. Como en cada paso podemos conseguir el edificio activo más alto, proseguiremos así hasta el último punto y habremos armádonos la silueta.

2.0.10. Demostracion

Demostraremos por inducción que en cada paso de este algoritmo obtenemos la altura del edificio más alto en ese punto.

P(i) = "Nuestro edificio Max es el edificio más alto en el punto i"

Caso base, P(1): En este caso nos encontramos con nuestro primer conjunto de paredes, que puede tener una o más paredes. Este sólo puede tener paredes izquierda ya que es donde çomienzan"nuestros primeros edificios y ningún edificio ha empezado antes como para que aparezca una pared derecha indicando su "finalización". Por lo tanto tenemos un conjunto de edificios que çomienzan.[en] este punto y nos basta recorrer ese conjunto para encontrar el edificio de mayor altura, al cual llamaremos Max. Entonces, sin duda, Max es el edificio más alto en este punto.

Paso inductivo, P(n) $\rightarrow$ P(n+1): Nuestra hipótesis inductiva nos dice que el edificio más alto en el punto n es Max. Ahora veamos qué ocurre en n+1. En este punto podemos encontrar un conjunto que contiene tanto paredes izquierda como derecha, con lo cual separaremos a este conjunto de paredes en dos subconjuntos. Por un lado el conjunto de paredes izquierda, y por otro el de paredes derecha. Recorreremos primero el conjunto de paredes izquierda agregando cada edificio de este conjunto al conjunto de edificios activos. Cada vez que alguno supere a nuestro Max actual, ese será el nuevo Max. Si después terminar el recorrido nuestro Max anterior cambió, entonces ese nuevo Max será el edificio más alto en este punto (y no es posible que este nuevo Max tenga su pared derecha en este punto ya que eso implicaría que ese edificio está .[em]pezandoz "terminando.[en] el mismo punto, lo cual no es válido). Y caso contrario seguiremos manteniendo nuestro Max de antes y deberemos fijarnos las paredes derecha para ver si este Max "termina". Ahora recorreremos el conjunto de paredes derecha quitando cada edificio que aparezca en este conjunto de nuestro conjunto de activos y fijándonos si está "terminando.[el] edificio Max en cada paso. Entonces cada vez que aparezca una pared derecha del edificio Max actual, lo reemplazaré por el edificio activo más alto. Por lo tanto, después de terminar el recorrido, nuestro edificio Max es el más alto de los edificios en ese punto.

Teniendo las alturas máximas para cada punto, es trivial armarnos la silueta ya que con las alturas máximas ya tenemos su borde superior en cada punto.

2.0.11. Complejidad

2.0.12. Experimentacion

Capítulo 3

Biohazard

3.1. Introducción

En este problema, se nos pide que ideemos un algoritmo que dados n productos quimicos que deben transportarse en camiones de un lugar a otro, tales que si un elemento i va en el mismo cambion que otro elemento j, esto conlleva una "peligrosidad.ªsociada h_{ij}. El objetivo aquí es utilizar la menor cantidad de camiónes posibles, pero que cada camión tenga una peligrosidad menor a una cota m.
La entrada del problema consiste en:

- Un entero **n** $\rightarrow$ Representarán el número de productos quimicos a transportar.
- Un entero **m** $\rightarrow$ Representará la cota de peligrosidad que ningun camión puede superar.
- **n-1** filas donde, para cada fila i consta de $n - i$ enteros:
  - h_{ii+1}, h_{ii+2} ... h_{in} $\rightarrow$ Representarán la peligrosidad asociada del elemento i con los elementos $i + 1$, $i + 2$ ... n.

La salida, por su parte, constará de una fila con:

- Un entero **C** $\rightarrow$ Representará la cantidad indispensable de camiónes que es necesaria para transportar los productos bajo las condiciones del problema.
- n enteros $\rightarrow$ Representarán en que camión viaja cada producto.

3.1.1. Ejemplo de entrada valida

Hagamos un pequeño ejemplo para que pueda ilustrarse bien el problema.
Supongamos que tenemos 3 productos quimicos, el producto 1 es muy inestable, por lo que si es transportado con el producto 2 la peligrosidad asciende a 40, y si se transporta con el producto 3 la peligrosidad es de 35. El producto 2 en cambio es de naturaleza mas estable, por lo que si es transportado con el producto 3 solo produce una peligrosidad de 3.
Por otro lado queremos que la peligrosidad por camión no supere el valor de 39.
Entonces la entrada para este problema será:

3 39
40 35
3

Para una entrada de estas dimenciones es posible buscar la mejor solucioón a mano.
Las posibles combinaciones son que los tres productos viajen juntos, que los tres viajen separados, que 1 y 2 viajen juntos, que 1 y 3 viajen juntos y que 2 y 3 viajen juntos y el producto sobrante viaje en otro camión.
La primera dá una peligrosidad de $40 + 35 + 3$ por lo que es inviable, la segunda es valida, pero se necesitan

3 camiones. Que 1 y 2 viajen juntos, tampoco es valida (peligrosidad muy alta), y finalmente las ultimas dos son validas (peligrosidad 35 y 3, respectivamente) y solo son necesarios dos camiones.
Luego las dos salidas que podrá devolver el algoritmo son:

- 2 1 2 1

o

- 2 1 2 2

3.2. Idea General de Resolución

Luego la idea del algoritmo es simple, probar todas las combinaciones posibles de camiones y de entre todas determinar cual es la que cumple con la cota de peligrosidad pedida y usa la menor cantidad de camiones posible. Ademas, se irá podando algunas ramas de una manera mas o menos inteligente para intentar lograr una mejor performance.
Antes de presentar el pseudocodigo vale aclarar un punto importante y es que este algoritmo encontraá siempre una solución. Esto se debe a que siempre es posible poner todos los productos quimicos en camiones separados, lo que nos dá una peligrosidad 0. Luego es posible usar esta como una cota contra la cual parar de chequear, si tenemos n productos quimicos, es simple ver que a lo sumo usarémos n camiones. Denominaremos a esta como la "peor soluciónza que es claro que es una solución valida, pero que usa la maxima cantidad de camiones.
En cuanto a las podas, utilizamos dos, una que en cada paso del backtrack chequea si la solución final que encontramos hasta el momento usa una cantidad menor de camiones que la solucion parcial que se esta construyendo. Es claro que de ser así, estamos la solucion parcial nunca podrá ser mejor, por lo tanto se poda.
La segunda chequea que la solución parcial que estamos construyendo no exceda el limite de peligrosidad pedido por el ejercicio. En caso de que esto suceda, tambien se poda.
Finalmente el pseudocodigo para resolver este problema queda así:

Algorithm 1 void FuncionPrincipal()

1: Generar una matriz con las peligrosidades entre los distintos productos
2: Se inicializa la solucion final, como la peor de las soluciones
3: Backtrack(tabla$_d e_p eligrosidad, solucion_p arcial, solucion_f inal$)
4: Mostar la solución final

Algorithm 2 Bool Backtrack(tabla$_d e_p eligrosidad, solucion_p arcial, solucion_f inal$)

1: Llamo a la funcion check(tabla$_d e_p eligrosidad, solucion_p arcial, solucion_f inal$)
2: Si check devuelve 2, la solucion parcial es mejor que la final
3: Pongo la solucion parcial como final
4: Corto la recursión y busco por otra rama
5: Si check devuelve 3, la solucion anterior usa menos camiones
6: esta rama no me sirve, podo
7: Si check devuelve 0, la cota de peligrosidad fué sobrepasada
8: esta rama no me sirve, podo
9: Si check devuelve 1, la solucion es valida, pero no está completa
10: continúo agregando camiones
11: Para cada valor i de 1 hasta n prueba meter el siguiente producto de la lista en el camion i y se llama a la funcion Backtrack()

Algorithm 3 int check(tabla$_d e_p eligrosidad, solucion_p arcial, solucion_f inal$)

```
1: Checkeo si la solucion final usa menos camiones                O(n)
2:     Si es verdad
3:         Devuelvo 3
4: Checkeo si la cota de peligrosidad fue sobrepasada             O(n^2)
5:     Si es verdad
6:         Devuelvo 0
7: Checkeo si la cada producto tiene un camion asignado           O(1)
8:     Si es verdad
9:         Devuelvo 2
10: Devuelvo 1
```

3.3. Complejidad

Por cada producto quimico, el algoritmo de backtrack intenta ponerlo en cualquiera de los n posibles camiones, y realiza $O(n^2)$ chequeos intentando podar.
Entonces la formula quedará:

$$T(i) = T(i-1)n + n^2$$

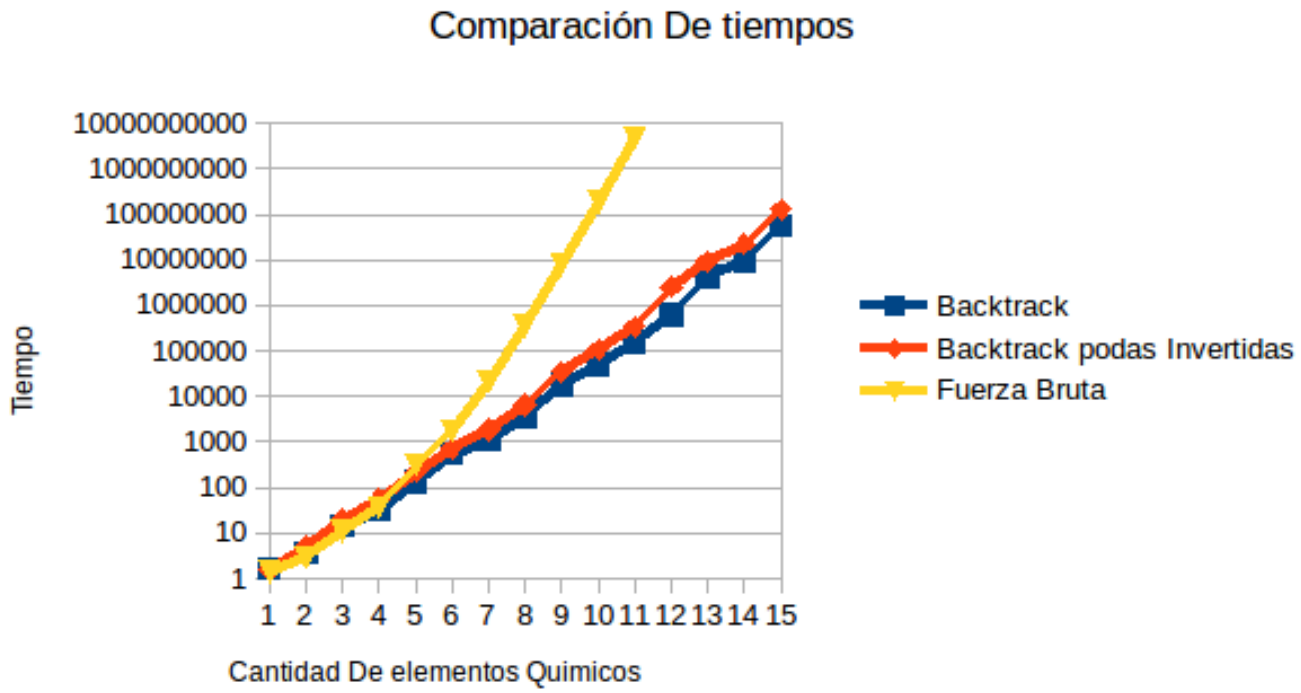
$$T(1) = n + n^2$$

Luego para el peor de los casos el algoritmo tendrá una complejidad de $O(n^n)$.

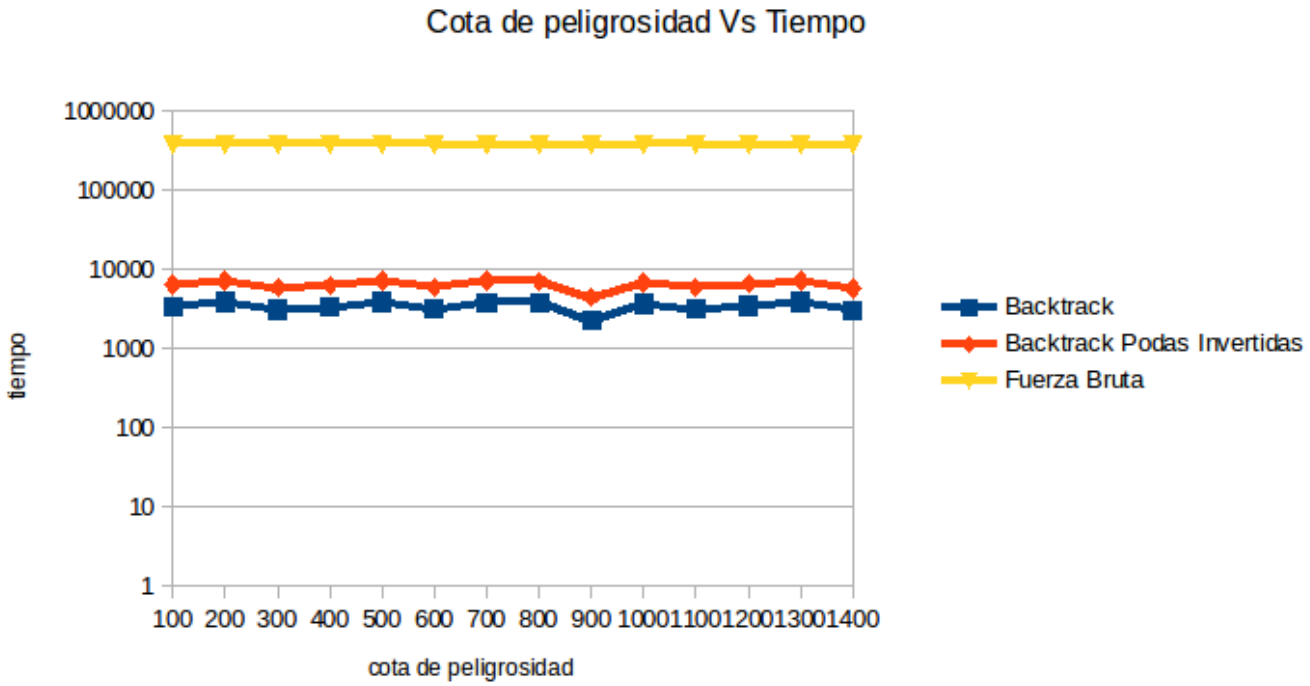
3.4. Testing

3.4.1. Caso Random

Para testear la performance de nuestro algoritmo se creó un generador de entradas que fabricará instancias random del problema. Luego se comparará el tiempo que tarda nuestro algoritmo contra uno que encuentre la solución utilizando fuerza bruta y tambien contra otro backtraking, pero con las podas invertidas, osea primero chequea si la cantidad de camiones de la solucion parcial es menor a la cantidad de camiones de la mejor solución hasta el momento y luego chequea que la cota de complejidad sea la correcta, para ver si esto varía de alguna manera la complejidad.
El testeo consistio en generar 40 instancias del problema para cada n diferente (con un m fijo), luego se tomó la media y graficaron los resultados:



En este grafico puede verse que nuestro algoritmo es notablemente superior a uno algoritmo de fuerza bruta y levemente mejor al backtraquing con las podas invertidas. Para los casos de 14 y 15 elementos el backtraking pudo arrojar una respuesta en un tiempo admisible, mientras que el de fuerza bruta ya tardaba tiempos completamente fuera de escala.
Se relaió lo mismo variando la cota de peligrosidad m para ver como se veían afectados los diversos algoritmos ($n = 9$):



La clara conclución es que ningun algoritmo se ve afectado por la cota de peligrosidad.

3.4.2. Peor caso

Otro test que podemos intentar realizar es ver si nuetro algormitmo presenta alguna mejora al de fuerza bruta en el peor de los casos, esto es, en el caso de que cada producto tenga que viajar forzosamente en un

camion diferente.
Para testear esto tomamos nuevamente 40 entradas y las corremos en los tres algoritmos.

3.5. Resultados

Capítulo 4

Apéndice