Teoría de algoritmos 1 – 75.29

Trabajo Práctico Nº:

Integrantes:

Padrón	Nombre y Apellido	Email
92216	Alejandro Pernin	ale.pernin@gmail.com

Para uso de la cátedra
Primera entrega
Corrector
Observaciones
Segunda entrega
Corrector
Observaciones

Distancia de Edición

Alejandro Pernin

23 de noviembre de 2013

Índice

75.29 Teoría de Algoritmos I

Trabajo Práctico Nº 3

Problema del Empaquetamiento

Fecha de entrega: 27 de noviembre de 2013

Definición: Dado un conjunto de n objetos cuyo tamaño son {T1, T2, … Tn} , con $T_i \varepsilon (0, 1]$ se debe empaquetarlos usando la cantidad mínima de envases de capacidad 1.

Se pide
1) Demostrar que el problema del empaquetamiento es NP-Completo (para lo cual se debe utilizar alguno de los problemas NP-Completos vistos en clase)
2) Programar un algoritmo por fuerza bruta que busque la solución exacta del problema. Analizar el orden del mismo. Realizar mediciones empíricas tomando el tiempo que demora cada corrida, graficar en función de n y comparar con la curva teórica. Analizar los resultados de las mediciones.
3) Dado el siguiente algoritmo: Se abre el primer envase y se empaqueta el primer objeto, luego por cada uno de los objetos restantes se prueba si cabe en el envase actual que está abierto, si es así se lo empaqueta en el mismo envase y se continúa con el siguiente objeto, si no cabe se cierra el envase actual y se abre uno nuevo que pasa a ser el envase actual y se empaqueta el objeto y continúa con el próximo hasta lograr empaquetar todos los objetos. Este algoritmo sirve como una aproximación para resolver el problema del empaquetamiento.
Se pide implementar este algoritmo, analizar el orden y analizar que tan buena es la aproximación.
Para analizar que tan buena es la aproximación se usa la siguiente fórmula: Sea I una instancia cualquiera del problema del empaquetamiento, sea z(I) la solución óptima para esa instancia y sea A(I) la solución aproximada entonces se define $\frac{A(I)}{z(I)} \leq r(A)$ para todas las instancias I. Calcular r(A) para la aproximación dada (y demostrar que la cota está bien calculada). Realizar mediciones empíricas utilizando el algoritmo exacto del punto anterior y el algoritmo aproximado de este punto con el objeto de verificar que se cumple la relación.

Ejemplo:

T={0,4; 0,8; 0,5; 0,1; 0,7; 0,6; 0,1; 0,4; 0,2; 0,2}

Solución exacta:
E1={0,5; 0,4; 0,1}
E2={0,8; 0,2}
E3={0,7; 0,2; 0,1}
E4={0,6; 0,4}

Total 4 envases.

Solución aproximada:
E1={0,4}
E2={0,8}
E3={0,5; 0,1}
E4={0,7}
E5={0,6; 0,1}
E6={0,4; 0,2; 0,2}
Total 6 envases

Datos de entrada:
Los datos vendrán en archivos de textos con el siguiente formato:
<n>
<linea en blanco>
<T1>
<T2>
.
.
.
<Tn>
<EOF>

Invocación:
tdatp3 <E> | <A> <datos.txt>
donde el parámetro E indica calcular la solución exacta y el parámetro A calcular la solución aproximada.

Formato de salida:
La salida será por pantalla con el siguiente formato y con el encabezado establecido en las normas para la presentación de los TPs:

<Solución Exacta> | <Solucion Aproximada>: #Envases
<Tiempo de ejecución en mseg>.

Consideraciones para realizar las pruebas empíricas de medición de tiempo:
Se recomienda realizar varias corridas con distintos conjuntos de datos del mismo tamaño y promediar los tiempos medidos para obtener un punto a graficar. Repetir para valores de n crecientes hasta valores que sean manejables con el hardware donde se realiza la prueba

1. Resolucion

1.1. Demostración NP-Completo

Para la demostración de que este problema es NP-Completo, se utiliza otro problema cuya demostración de NP-Completo se considera ya conocida; reduciendo dicho conocido problema al problema de Bin Packing, demostramos que el problema también es NP.

Como problema de referencia se utilizará *Subset Sum*[1]. Primero definamos ambos problemas como problemas de decisión:

- Bin Packing: $\{< S, j > |S = \{s_1, s_2..., s_n\}; 0 < s_i < 1$, y todos los objetos $1, ..., n$ deben empaquetarse en envases de capacidad j.
- Subset Sum: $\{< C, k > |C = \{c_1, c_2..., c_n\}$ donde c_i es un entero positivo, y existe algun subconjunto $C' \subseteq C$ tal que la suma de sus elementos sume exactamente k.

Se considera demostrado y sabido que *Subset Sum* es NP-Completo, por medio de reducción demostraremos que *Bin Packing* también lo es. Primero consideramos que *Subset Sum* es NP-Completo aún en un caso restringido:

$k = \sum_{c_i \in C}^{n} c_i/2 = W/2$

Llamamos a esta instancia *Subset Half Sum*, instancia que se demostrará se soluciona facilmente con *Bin Packing*. Para ello mostramos que una instancia $< C, k >$ de *Subset Half Sum* $\leq p$ *Bin Packing* [2]. Primero se convierten los elementos en C a un problema de *Bin Packing* de 1 envase dividiendo los elementos en C por k. Eso es, para $C = \{c_1, c_2, ..., c_n\}$ crear $C' = \{s_i | s_i = c_i/k\}$. Creamos una instancia $< C', 2 >$ de *Bin Packing*. Si la respuesta a este *Bin Packing* es "si", también lo será para *Subset Half Sum*, de lo contrario "no".

Prueba: Si el mínimo de envases es 2 y cada envase tiene capacidad $\frac{W}{2k}$, entonces cada uno contiene un subconjunto de peso esactamente $\frac{W}{2k}$, y cada uno es una solución a *Subset Half Sum*. Si el mínimo de envases es mayor a 2(no puede ser 1 ya que $1 < W/k$), no existe ningún subconjunto de peso $\frac{W}{2k}$. De otro modo, tendriamos que usar ese subconjunto en el primer envase y los $\frac{W}{2}$ items en el segundo envase obteniendo una solución de 2 envases.

[1] `http://en.wikipedia.org/wiki/Subset_sum_problem`

[2] Notación ver: Algorithm Desing - Chapter 8, page 452

1.2. Fuerza Bruta

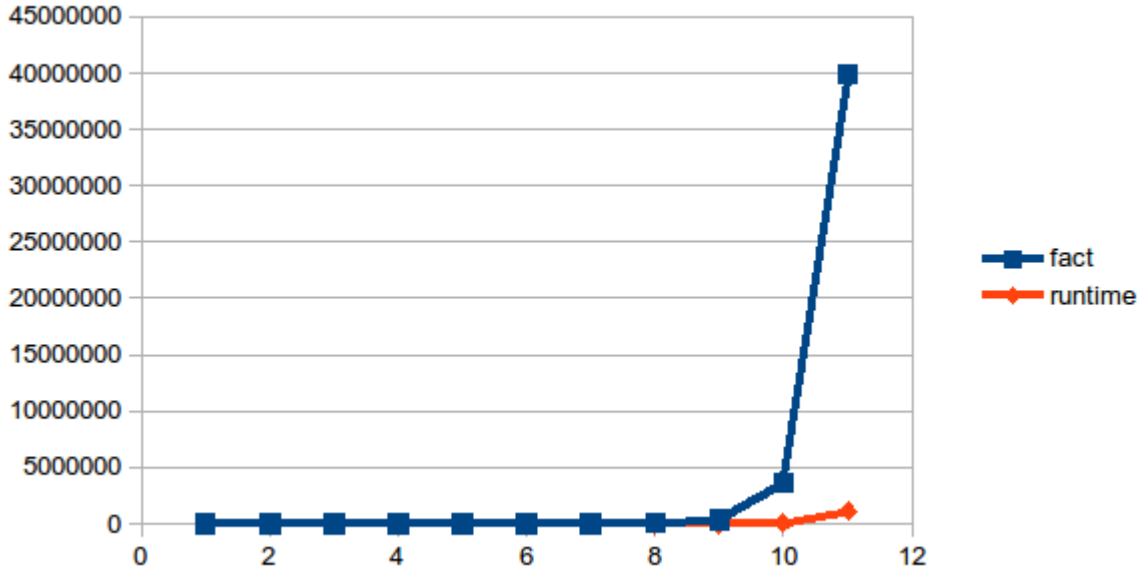
La resolución por fuerza bruta, implica probar todas las combinaciones posibles y de ellas obtener la que implique el menor uso de envases. El enfoque que se utiliza en este TP, es por cada combinación posible con los elementos del cojunto, correr el algoritmo de aproximación, bajo la hipótesis de que existe al menos una permutación del conjunto para el cuál el algoritmo de aproximación da el resultado óptimo.

Para hacer un análisis del mismo es preciso primero analizar cuantas permutaciones posibles hay a partir de un conjunto, para ello se analiza desde un punto de vista recurrente. Sea el conjunto $S = \{s_1, s_2, ..., s_n\}$ y la función $P(n)$ que dá la cantidad de permutaciones para un conjunto de n elementos, $P(n) = n * P(n-1)$. El caso base sería $P(1) = 1$ lo cuál es trivial ya que no hay permutaciones posibles para un elemento más que él mismo. Armando el arbol de recurrencias se llega a que $P(n) = n!$

Con motivo de analizar los tiempos que demanda la ejecución del programa, se toma el tiempo en el cuál se inició y concluyó, siendo la resta de los mismos el tiempo que tardó. El mismo se expresa en milisegundos y los valores cercanos a 0 son propensos a mayores errores relativos.

Para las pruebas se efectuaron datos al azar y ejecutaron varias veces, siendo el tiempo que se mostrará el promedio de los tiempos obtenidos. Para las pruebas de conjuntos de tamaños considerables (10 y 11) solo se efectuo una ejecución por el tiempo que llevaría hacer varias ejecuciones.

n	$fact(n)$	tiempo ms
1	1	0.068
2	2	0.078
3	6	0.11
4	24	0.5
5	120	2.57
6	720	14.16
7	5040	92.3
8	40320	716.91
9	362880	7238.83
10	3628800	78387
11	39916800	1138704



Aunque en el gráfico no se aprecie con exactitud, en la tabla es apreciable como los tiempos crecen factorialmente

2. Analisis de Orden

3. Codigo Fuente