Teoría de algoritmos 1 – 75.29

Trabajo Práctico Nº: 3

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

Problema del Empaquetamiento

Alejandro Pernin

25 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]:

- Sea un conjunto S de números, hallar (de haber) un conjunto $S' \subseteq S$ tal que la suma de todos los elementos en S' sea exactamente t.

Sea T la suma de todos los números del conjunto S, consideramos el caso $t = \frac{T}{2}$, esto es $< S, \frac{T}{2} >$. Construyendo una instancia de *Bin Packing* dividiendo cada elemento de S por $T/2$, la instancia $< C, 2 >$ donde los elementos de C pueden almacenarse en dos envases de tamaño 1. Si existe un subconjunto $S' \subseteq S$ tal que la sumatoria de todos los elementos de S' sea exactamente $T/2$, Los elementos de C correspondientes a S' pueden almacenarse en un envase, y los restantes pertenecientes a $S - S'$ en el otro.

1.2. Analisis 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 $P(n) = n!$

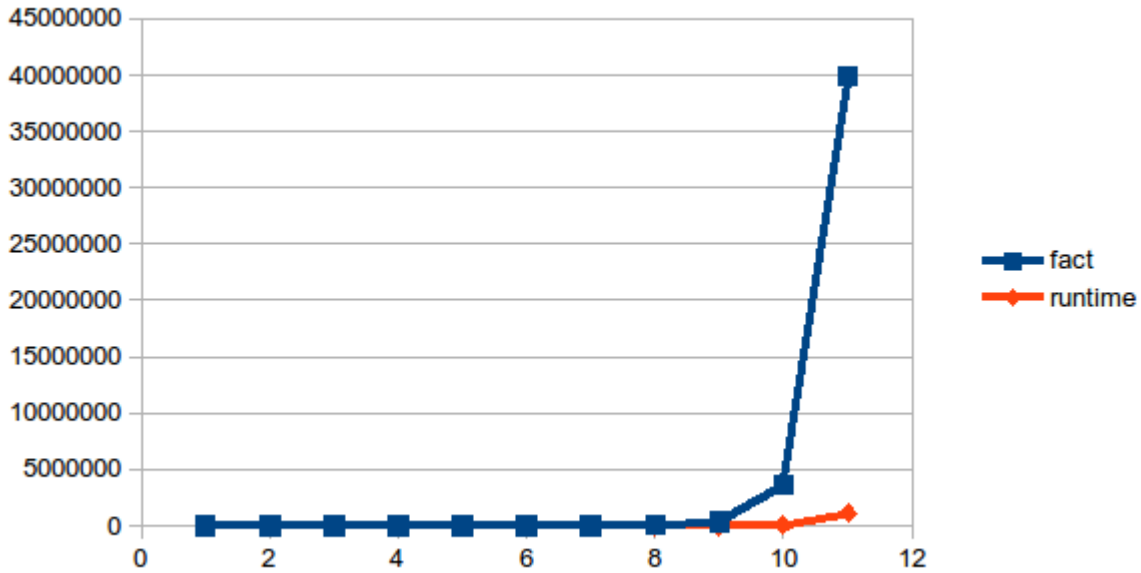
Con motivo de analizar los tiempos que demanda la ejecución del programa, se toma el tiempo en el cuál se inició y concluyó, siendo la resta de

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

los mismos el tiempo que tardó. El mismo se expresa en milisegundos (los valores cercanos a 0 son propensos a mayores errores relativos).

Para las pruebas se crearon 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 con una forma aproximada a $t(n) = n * t(n-1)$.

1.3. Analisis Aproximacion

Para analizar que tan buena es la aproximacion, se busca una cota $r(A)$ tal que $\frac{A(I)}{Z(I)} \leq r(A)\ \forall I$ instancia del problema.

Para hallar dicha cota, es preciso analizar las variaciones de $A(I)$ respecto de $z(I)$ para toda instancia I. A priori podemos decir que $z(I) \leq A(I)\ \forall I$ lo cuál tiene sentido ya que la aproximación a lo sumo puede ser tan buena como la óptima. De acá se deduce que $1 \leq r(A)$.

Realizando análisis de casos se pueden empezar a hacer ciertas conclusiones:

- Si $z(I) = 1 \rightarrow A(I) = 1 \wedge r(A) = 1$, lo cuál es trivial.

Ahora analizemos casos para $z(I) \geq 2$, para maximizar la cantidad de envases a utilizar, se debe maximizar la cantidad de casos de $s_i + s_{i+1} > 1$, de esta manera el algoritmo de aproximación utilizará más envases.

En el caso de $z(I) = 2$ expresamos una posible solución óptima como:
$\{0{,}5 + h; 0{,}5 - h\}\{0{,}5; 0{,}5\}$ con h el menor incremento posible.
Reorganizando los elementos de la siguente manera:
$\{0{,}5 + h; 0{,}5; 0{,}5; 0{,}5 - h\}$
El algoritmo de aproximación nos dará:
$\{0{,}5 + h\}\{0{,}5; 0{,}5\}\{0{,}5 - h\}$
Siendo en este caso $r(A) = 3/2$

En el caso de $z(I) = 3$ expresemos primero una posible solución óptima:
$\{0{,}5 + h; 0{,}5 - h\}\{0{,}5; 0{,}5\}\{0{,}5 + h; 0{,}5 - h\}$
Reorganizando los elementos de la siguente manera:
$\{0{,}5 + h; 0{,}5; 0{,}5 + h; 0{,}5; 0{,}5 - h; 0{,}5 - h\}$
El algoritmo de aproximación nos dará:
$\{0{,}5 + h\}\{0{,}5\}\{0{,}5 + h\}\{0{,}5; 0{,}5 - h\}\{0{,}5 - h\}$
Siendo en este caso $r(A) = 5/3$

Extendiendo esto a todo $z(I)$ posible, sea $z(I) = n$ entonces:

- n par $\rightarrow A(I) = n + n/2$.
- n impar $\rightarrow A(I) = n + n/2 + 1$ ($/$ división entera).

Con estos dos casos posibles la expresión $r(A)$ nos queda:

- n par $\rightarrow \dfrac{n + n/2}{n} = \dfrac{3}{2}$
- n impar $\rightarrow \dfrac{n + n/2 + 1}{n}$ Como es división entera de un número impar, no es posible realizar la simplificación que se hizo arriba. El término independiente $+1$ se hace menos respresentativo a medida que aumenta n, analizando para $n = 3$ queda $r(A) = 5/3$ que resulta ser mayor que $3/2$

Por ende definimos que la cota superior $r(A) = \frac{5}{3}$.

Ejemplo de una de las ejecuciones de prueba[2]:

```
Solucion exacta
{0.6;0.4}
{0.5;0.5}
{0.6;0.4}
Son 3 envases
time elapsed 9.7439289093 ms

Solucion aproximada
{0.6}
{0.5}
{0.6}
{0.5;0.4}
{0.4}
Son 5 envases
time elapsed 0.0810623168945 ms
```

[2]Ver sección: 3

2. Analisis de Orden

En esta sección se analizarán los órdenes de los módulos principales:

- Aprox: Itera cada elemento del conjunto e intenta meterlo en el envase abierto, de no poderse se abre otro envase. $O(n)$
- Brutus: Por cada permutación del conjunto ($n!$), ejecuta el algoritmo de aproximación. $O(n * n!)$

3. Ejecución y Pruebas

Siendo que el lenguaje utilizado es Python, no es necesaria ninguna compilación. El programa se puede ejecutar directamente mediante el siguiente formato:

```
python tdatp3.py <A|E> <data_file>
```

o bien:

```
./tdatp3.py <A|E> <data_file>
```

También con motivo de facilitar las pruebas y del armado de un conjunto de datos para las mismas, se facilitan ciertos scripts:

- gen.py: Genera un conjunto aleatorio de datos de tamaño n. Se invoca con un argumento de la siguiente manera:

  `python gen.py <n>` ó `./gen.py <n>`

- testaprox.sh: Genera un conjunto aleatorio de datos utilizando gen.py y ejecuta el programa para obtener una aproximación. También require como argumento el tamaño deseado del conjunto:

  `./testaprox.sh <n>`

- testexact.sh: Idem testaprox pero para la solución exacta.

  `./testexact.sh <n>`

- testcomp.sh: Combinación anteriores, genera un conjunto de datos y ejecuta tanto la aproximación como la óptima, con motivo de ser comparados ambos resultados.

  `./testcomp.sh <n>`

4. Conclusiones

Se diseñó un algoritmo que implemente dos enfoques de resolución a un problema de Bin Packing, con la consecuente comparación de ambas. Se comparó un algoritmo de aproximación (lineal) con uno exacto mediante fuerza bruta, tanto en diferencias en la solución como en el tiempo de obtención de la misma.

Se apreció como para conjuntos chicos (menores a 10 elementos), el algoritmo de fuerza bruta ejecuta en un tiempo razonable, y que con conjuntos de elementos mayores, el tiempo requerido aumentaba factorialmente.

Asimismo se analizó el algoritmo de aproximación, obteniendo que sus resultados se obtienen en tiempo lineal y se calculó una cota del error.

5. Referencias

- Wikipedia: Big O Notation - `http://en.wikipedia.org/wiki/Big_O_notation`
- Python Wiki: Time Complexity - `https://wiki.python.org/moin/TimeComplexity`
- 'Introduction to Algorithms'- Cormen, Leiserson, Rivest, Stein -(ISBN:0-262-03293-7)
- 'Algorithm Desing' - Kleinberg, Tardos - (ISBN: 0-321-29535-8)
- Case Western Reserve University - Course EECS 454 Resources

6. Codigo Fuente

6.1. tdatp3.py

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 import sys
5 from types import *
6 import time
7 import itertools
8
9 class Objeto():
10   def __init__(self, tam):
11     assert type(tam) is FloatType
12     self.tam = tam
13
14   def __str__(self):
15     return str(self.tam)
16
17   def getTam(self):
18     return self.tam
19
20 class Envase():
21   def __init__(self):
22     self._list =[]
23     self.max=1
24     self.count=0
25
26   def __str__(self):
27     s = "{"
28     s += "%s" %self._list[0]
29     for i in self._list[1:]:
30       s+= ";%s" %i
31     s += "}"
32     return s
33
34   def envasar(self, obj):
35     if (self.count+obj.getTam())<=self.max:
36       self._list.append(obj)
37       self.count+=obj.getTam()
38       return True
39     else:
40       return False
41 """
```

```
42 Dada una lista de objetos, crea todas las
43 permutaciones posibles
44 O(n* n!)
45 """
46 def brutus(l):
47   n=len(l)
48   gen=itertools.permutations(l)
49   (e,c)=(None,None)
50   while True: #O(n!)
51     try:
52       aux=gen.next()
53     except StopIteration:
54       break
55     x=aprox(aux) #O(n)
56     if (x[1]<c)or(c is None):
57       c=x[1]
58       e=x[0]
59   return (e,c)
60
61 """
62 El algoritmo de aproximacion recibe como parametro
63 una lista que contiene los objetos a envasar
64 O(n)
65 """
66 def aprox(l):
67   r=[]
68   e=Envase()
69   count=1
70   for obj in l: #O(n)
71     if not e.envasar(obj):
72       r.append(e)
73       e=Envase() #Se abre otro envase
74       count+=1
75       e.envasar(obj)
76   r.append(e)
77   return (r,count)
78
79 def loadFile():
80   try:
81     _file = open(sys.argv[2])
82   except IOError:
83     print "El archivo no pudo ser abierto, verificar parametros"
84     usage()
85     return False
86   l=[]
```

```
87    n=_file.readline().replace('\n','')
88    n=int(n)
89     _file.readline() #Linea en blanco
90    for line in _file:
91      tam=line.replace('\n','').replace(',','.')
92      tam=float(tam)
93      obj=Objeto(tam)
94      l.append(obj)
95    return l
96
97  def usage():
98    print "Usage:"
99    print "python tdatp3.py A|E <archivo datos> \n or"
100   print "./tdatp3 A|E <archivo datos>"
101
102 def checkArgs():
103
104   if (len(sys.argv) is not 3)or(sys.argv[1] is not "A" and sys.argv[1] is not "E"):
105     usage()
106     return False
107   return True
108
109 def runBrutus(l):
110   start=time.time()
111   print "Solucion exacta"
112   (r,c)= brutus(l)
113   for e in r:
114     print e
115   print "Son %s envases" %c
116   elapsed=time.time()-start
117   print "time elapsed %s ms" %(elapsed*1000)
118
119 def runAprox(l):
120   start=time.time()
121   print "Solucion aproximada"
122   (r,c) = aprox(l)
123   for e in r:
124     print e
125   print "Son %s envases" %c
126   elapsed=time.time()-start
127   print "time elapsed %s ms" %(elapsed*1000)
128
129
130 def main():
131   print "Teoria de Algoritmos 1 - [75.29]"
```

```
132   print "TP3 - Problema del Empaquetamiento"
133   print "Autores: Alejandro Pernin (92216)\n"
134   l=None
135   if not checkArgs():
136     return
137   l=loadFile()
138   if not l:
139     return
140   caso = sys.argv[1]
141   if caso is 'A':
142     runAprox(l)
143   elif caso is 'E':
144     runBrutus(l)
145   else:
146     print "Algo fallo"
147
148 if __name__ == '__main__':
149   main()
```

6.2. Test Scripts

6.2.1. gen.py

```
#!/usr/bin/python

import sys
from random import randint as r

_file = open('data_aux.txt','w+')
n = int(sys.argv[1])
_file.write(sys.argv[1]+'\n')
_file.write("\n")
for i in range(n):
  num=r(1,10)
  if num<10:
    s='0,'+str(num)+'\n'
  else:
    s='1,0\n'
  _file.write(s)
```

6.2.2. testaprox.sh

```
#!/bin/bash
./gen.py $1
./tdatp3.py A data_aux.txt
```

6.2.3. testexact.sh

```
#!/bin/bash
./gen.py $1
./tdatp3.py E data_aux.txt
```

6.2.4. testcomp.sh

```
#!/bin/bash
clear
./gen.py $1
./tdatp3.py E data_aux.txt
echo -e "##################\n"
./tdatp3.py A data_aux.txt
```