7529. Teoría de Algoritmos I
Trabajo Práctico 3

Ferreyra, Oscar, *Padrón Nro. 89563*
fferreyra38@gmail.com

Martin, Débora, *Padrón Nro. 90934*
demartin@fi.uba.ar

Eisner, Ariel, *Padrón Nro. 90697*
aeeisnerg@gmail.com

2do. Cuatrimestre de 2016

Índice

1. Clases de complejidad

1.1. Ciclos con peso negativo en un grafo Hamiltoniano

Para la resolución de este problema se propone la utilización del Algoritmo de Floyd Warshall para encontrar el camino mínimo entre todos los vértices del grafo. Sobre la matriz resultante se analizan los números de la diagonal de la matriz de caminos. En caso de ser negativos, sabemos que pertenece a un ciclo negativo. Este algoritmo corre en tiempo $O(V^3)$, siendo V la cantidad de vértices del grafo.

```
FloydWarshall() {
  Armar la matriz de adyacencia F
  for( k from 1 to n )
    for( i from 1 to n )
      for( j from 1 to n )
        F[i,j]=min(F[i,j], F[i,k] + F[k,j])
      end for j
    end for i
  end for k

  for( i from 1 to n )
    if( F[i,i] ¡0 then )
      print “Ciclo negativo”
      break
    end if
  end for i
}
```

1.2. Ciclos de peso nulo en un grafo Hamiltoniano

Este problema es NP-Completo.

Primero, dado un ciclo simple en el grafo G, podemos determinar si la suma de los pesos de sus vértices en tiempo polinomial, luego el problema de encontrar un ciclo de peso 0 es NP.

Luego, reducimos el problema de la suma de subconjuntos a este problema. Este problema es: dado un conjunto de enteros, ¿existe algún subconjunto cuya suma sea exactamente cero? A continuación describimos el algoritmo de resolución de dicho problema:

Consideramos un conjunto de enteros $S = \{a_1, ..., a_n\}$. Construimos un digrafo ponderado G con $2n$ vértices, para el cual cada elemento a_i corresponde con dos vértices v_i y u_i

for each v_i
agregar una arista de v_i en u_i con peso a_i y agregar aristas de cada vértice v_j en él con peso 0.
for each u_i
agregar aristas de éste vértice u_i a cada uno de los v_j con peso 0.
if (encontramos un ciclo de peso 0 en G) **then**
todos los pesos desde v_j hasta u_j a lo largo del ciclo deben ser cero.
if (obtenemos un subconjunto $S0 \subseteq S$ cuya sumatoria da 0)
construimos un ciclo tomando todas las aristas (v_j, u_j) correspondientes al elemento en S0 y conectamos sus aristas a través de las aristas de ppeso cero, y finalmente obtenemos un ciclo de peso 0.

Por lo tanto, el problema es al menos tan difícil como el de suma de subconjuntos. Dado que éste es un problema NP-Completo, el problema de los ciclos de peso 0 también lo es.

1.3. Tareas con ganancia igual a un k dado

Utilizamos otra vez el problema de la suma de subconjuntos. Podemos reducir este problema a un subconjunto del mismo, pero en este caso la suma es el k dado.

Por lo tanto, podemos decir que el problmea es NP-Completo.

Supongamos que

$$sumaDeSubconjutos \leq_p programacionConDeadlinesYGanancias$$

Partiendo de esto, vemos que dada una instancia del problema de suma de subconjutos $X = (x[1..n], t)$; donde, $x[i]$ es el conjunto de números y t es el resultado de la suma del subconjunto, es posible construir en tiempo polinomial una instancia de nuestro problema de programación con deadlines y ganancias de la siguiente manera:

$$Y = (x[1..n], d[1..n], x[1..n], t), \forall i \in d[i] = t$$

Otra vez, dado que el problema de suma de subconjuntos es NP-Completo, y el problema de programación con deadlines y ganancias mayores a un k dado es al menos tan difícil como este, queda demostrado que es un problema NP-Completa.

1.4. Tareas con ganancia mayor a un k dado

Esta es una versión extendida del problema explayado en el punto anterior. La demostración es muy similar, pero debemos tener en cuenta que ahora podemos revisar una cantidad mayor de valores, es decir, todos los que cumplen la condición de ser mayores al k dado

2. Algoritmos de aproximación

2.1. El problema del viajante de comercio

Para implementar la solución aproximada a este problema se utilizó el algoritmo de Prim para calcular el árbol de tendido mínimo. La complejidad de dicho algoritmo depende de las operaciones de una cola de prioridad que contiene los vertices, de las iteraciones que se realizan sobre todas las aristas de cada vértice. Se tomará $|V|$ como la cantidad de vértices del grafo y $|A|$ como la cantidad de aristas. Las operaciones más importantes de la cola de prioridad son $O(\log |V|)$ cada una, excepto el contains que es $O(|V|)$. Por otro lado el ciclo principal es $O(|V|^2 * |A|)$ porque recorre para cada vertice, sus aristas, y verifica si su arista adyacente está en la cola de prioridad. El mencionado sería el peor caso, pero a medida que va avanzando la ejecución, la cola se va vaciando y el contains pasa a ser insignificante. Por último, se agregan todos los vertices y las aristas pertenecientes al árbol en un nuevo grafo. Esta operación es $O(|A| \log |A|)$, en el peor caso que sería que todas las aristas del grafo original debieran ser agregadas al nuevo grafo. Pero en la mayoría de los casos será mucho menor y dependerá de la cantidad de aristas del árbol.

Además del algoritmo de Prim, luego se busca el preorder del árbol de tendido mínimo. Este recorre todos los vértices del grafo por lo que es $O(|V|)$.

Los tiempos de ejecución fueron llamativamente inferiores a los obtenidos por programación dinámica, aunque las soluciones obtenidas no resultaron siempre iguales a las anteriores. A continuación se muestran dichos tiempos de ejecución. Es de notar que en esta ocasión sí fue posible realizar la corrida para grafos con mayor cantidad de vertices.

Cantidad de ciudades	Tiempo de ejecución (seg)
15	$4{,}86 * 10^{-3}$
17	$5{,}74 * 10^{-3}$
19	$8{,}82 * 10^{-3}$
21	$6{,}35 * 10^{-3}$
23	$6{,}15 * 10^{-3}$
48	$13{,}4 * 10^{-3}$

Cuadro 1: Tiempo en función de la cantidad de ciudades para la aproximación por el árbol de tendido mínimo

2.2. El problema de la mochila

Se refactorizó el algoritmo utilizado en la primera parte, agregando la ecuación de recurrencia indicada en el enunciado. El algoritmo tiene la lógica explicada en el enunciado. Las mejoras consisten en aplicar otra ecuación de recurrencia para evaluar los pesos, esto es

A continuación se muestra una tabla con los tiempos de ejecución del algoritmo con algunos de los archivos de prueba.

Archivo de prueba	Tiempo de ejecución (seg)
knapPI_11_50_1000	0,138287280
knapPI_12_50_1000	0,080562404
knapPI_13_50_1000	0,083951296
knapPI_14_50_1000	0,075953632
knapPI_15_50_1000	0,103165037
knapPI_16_50_1000	0,101494485
knapPI_11_200_1000	1,145706132
knapPI_12_200_1000	0,859632826
knapPI_13_200_1000	0,971113896
knapPI_14_200_1000	1,076139513
knapPI_15_200_1000	0,761086035
knapPI_16_200_1000	1,348929039
knapPI_11_500_1000	5,400308580
knapPI_12_500_1000	4,315799133
knapPI_13_500_1000	4,253145449
knapPI_14_500_1000	9,521136486
knapPI_15_500_1000	4,433933508

Cuadro 2: Tiempo en función de la cantidad de paquetes y el tamaño de la mochila

Para instancias más grandes no era posible llevar a cabo las pruebas con la máquina con la que disponíamos.