

FACULTAD DE INGENIERÍA  
UNIVERSIDAD DE BUENOS AIRES

TEORÍA DE ALGORITMOS

---

## Trabajo Práctico N2

---

*Autores:*

Addin Kevin

Cabrera Jorge

Gatti Nicolas

Orlando Juan Manuel

*Padrón:*

94280

93310

93570

93152

18 de Noviembre, 2016



# Contents

<b>1</b>	<b>Programación dinámica</b>	<b>2</b>
1.1	El problema de la mochila . . . . .	2
1.1.1	Análisis de complejidad . . . . .	2
1.1.2	Análisis de tiempos . . . . .	2
1.1.3	Implementación del algoritmo . . . . .	3
1.2	El problema del viajante de comercio . . . . .	4
1.2.1	Análisis de complejidad . . . . .	4
1.2.2	Análisis de tiempos . . . . .	4
1.2.3	Implementación del algoritmo . . . . .	6
<b>2</b>	<b>Flujo de redes</b>	<b>7</b>
2.1	Modelado del problema . . . . .	7
2.2	Análisis de complejidad . . . . .	10
2.3	Análisis de tiempos . . . . .	11
2.4	Implementación del algoritmo . . . . .	13

# 1 Programación dinámica

## 1.1 El problema de la mochila

### 1.1.1 Análisis de complejidad

El cálculo de la complejidad de este algoritmo es sencillo ya que lo que se realiza es iterar una matriz bidimensional, realizando en cada iteración una cantidad finita de operaciones  $O(1)$ . Luego para el obtener el listado de items a incluir en la mochila debemos realizar una iteración adicional  $O(n)$ .

Por lo tanto, la complejidad del algoritmo es

$$O(nW)$$

siendo  $n$  la cantidad de elementos en la mochila y  $W$  el peso máximo soportado por la mochila.

### 1.1.2 Análisis de tiempos

### 1.1.3 Implementación del algoritmo

## 1.2 El problema del viajante de comercio

### 1.2.1 Análisis de complejidad

En este algoritmo tenemos que analizar los distintos conjuntos de vertices posibles. Para cada conjunto de vertices posible, luego tenemos que determinar el optimo de cada vertice para recorrer ese conjunto. Para determinar ese optimo, tenemos que realizar un minimo entre el resto de los vertices del conjunto, haciendo que la complejidad del algoritmo en tiempo sea:

$$O(n^2 2^n)$$

siendo  $n$  la cantidad de vértices.

En cuanto el espacio, hay que tener en cuenta que tenemos  $n$  vértices, y por cada vertice necesitamos guardar el resultado optimo para los distintos conjuntos de vertices posibles. Por lo tanto, la complejidad en espacio del algoritmo es:

$$O(n 2^n)$$

siendo  $n$  la cantidad de vértices.

### 1.2.2 Análisis de tiempos

Para la medición de tiempos, correremos el archivo de prueba “p01.tsp” (renombrado como tsp1) y 7 archivos adicionales que contienen desde 15 vertices hasta 21 vertices inclusive, extraidos del archivo “att48.d.txt”. La idea es poder ver como varía el tiempo con tan solo aumentar la entrada en una unidad.

prueba	clocks
tsp1	593626
n15	597700
n16	1480802
n17	3629355
n18	8816650
n19	21099357
n20	49919981
n21	117923147

Table 1: Tabla de tiempos

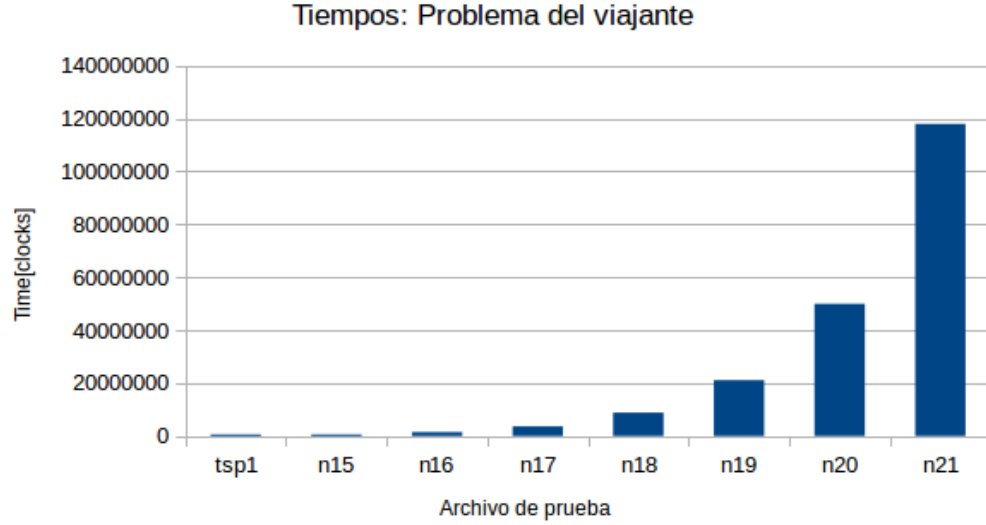


Figure 1: Gráfico de tiempos

Con el gráfico podemos observar que la ejecución del archivo de prueba “p01.tsp”(renombrado como tsp1) es casi instantanea.

También podemos apreciar que a medida que aumenta el número de nodos en tan solo una unidad, aumenta enormemente el tiempo insumido a más del doble.

En cuanto al archivo de prueba “fri26.tsp” de 26 nodos, no pudimos realizar la medición de tiempo debido a que el espacio requerido para la ejecución del algoritmo excedía nuestra memoria RAM disponible.

Si hacemos un poco de cuentas, mencionamos que la complejidad en espacio del algoritmo es  $O(n2^n)$ . Si  $n = 26$ , tenemos que  $26 * 2^{26} = 1744830464$ . Supongamos que requerimos 8B para almacenar el costo asociado a dicho vértice y conjunto, tendríamos:  $26 * 2^{26} * 8B \approx 14GB$ .

Nuestra implementación trato de optimizar el espacio lo más posible, utilizando para ello un entero para representar al conjunto de vértices, pero no pudo ser posible ejecutarlo.

### 1.2.3 Implementación del algoritmo

## 2 Flujo de redes

### 2.1 Modelado del problema

Para el modelado del problema, supongamos que tenemos 4 proyectos, con ganancias de 10, 5, 4 y 3, y supongamos que tenemos 3 areas con costos de 15, 8 y 4. Para realizar un proyecto dado requerimos contratar ciertas areas, por ejemplo:

Proyecto	Areas requeridas
1	1, 2
2	2,3
3	3
4	3

Table 2: Tabla de requisitos

Resolveremos este problema utilizando flujo de redes, basandonos en la versión del libro “Algorithm Design” [1].

Para ello lo que haremos será crear un grafo que contengan como vertices los proyectos, las areas y dos vertices adicionales,  $s$  y  $t$ .

Conectaremos el vertice  $s$  con cada proyecto y conectaremos cada area con el vertice  $t$ . Luego, conectaremos cada proyecto con las areas requeridas para su realización.

En cuanto a las capacidad de las aristas, las aristas que conecten el vertice  $s$  con los proyectos tendrán como capacidad la ganancia de realizar dicho proyecto. Para las aristas que conecten las areas con el vertice  $t$  tendrán como capacidad el costo de contratar dicha area. Por ultimo, las aristas que conectan los proyectos con las areas tendrán capacidad “Infinita”, de modo que no constituyan una limitación para el algoritmo. Esta capacidad infinita basta que sea igual a  $C + 1$ , siendo  $C = \sum g_i$ , donde  $g_i$  es la ganancia del proyecto  $i$ .



El grafo nos quedaría de la siguiente manera:

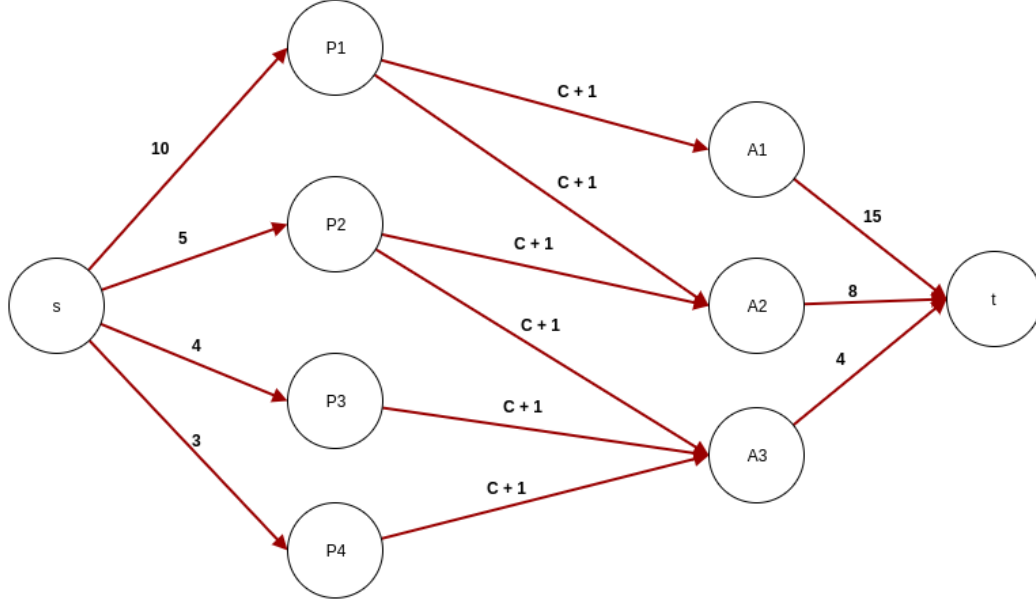


Figure 2: Grafo de ejemplo

Para obtener qué proyectos se realizarán y qué áreas se contratarán se debe calcular el corte mínimo de este grafo, esto es equivalente a calcular el flujo máximo.

Sea  $A$  un conjunto de vertices tal que si existe un proyecto dentro del mismo, también están las áreas requeridas para dicho proyecto. Definimos  $A' = A \cup \{s\}$ ,  $B' = (V - A) \cup \{t\}$ , donde  $V$  es el conjunto de vertices del grafo. Sea  $c(A', B')$  la capacidad del corte  $(A', B')$ , se puede demostrar que

$$c(A', B') = C - \left( \sum g_i - \sum c_i \right)$$

donde  $g_i$  es la ganancia del proyecto  $i$  y  $c_i$  es el costo para contratar el area  $i$ . Tenemos tres tipos de aristas: Las aristas que conectan  $s$  con los proyectos, las aristas que conectan las áreas con  $t$  y las aristas que conectan los proyectos con las áreas. Estas ultimas no contribuyen a dicho corte debido al supuesto inicial sobre  $A$ , esto es, si está un proyecto también están las áreas requeridas para el mismo.

Las aristas que conectan  $s$  con los proyectos que no están en  $A$  aportan al corte

$$\sum_{project \ i \notin A} g_i$$

Las aristas que conectan las areas que están en  $A$  con  $t$  aportan al corte

$$\sum_{area \ i \in A} c_i$$

Usando la definición de  $C$

$$\sum_{project \ i \notin A} g_i = C - \sum_{project \ i \in A} g_i$$

Por lo tanto, la capacidad del corte  $c(A', B')$  es

$$c(A', B') = \left( C - \sum_{project \ i \in A} g_i \right) + \sum_{area \ i \in A} c_i$$

$$c(A', B') = C - \left( \sum_{project \ i \in A} g_i - \sum_{area \ i \in A} c_i \right)$$

Entonces, si  $(A', B')$  es un corte con capacidad a lo sumo  $C$ , entonces el conjunto  $A = A' - \{s\}$  satisface el problema planteado, ya que no existirá ninguna arista del estilo  $(proy, area)$  que contribuya al corte.

Al maximizar el flujo, minimizamos el corte mínimo. Esto quiere decir que maximizamos  $\sum_{project \ i \in A} g_i - \sum_{area \ i \in A} c_i$ , que es lo que buscamos.

Volviendo al grafo de ejemplo, al aplicar el algoritmo de Ford Fulkerson, obtenemos un corte  $c(S, T)$ , tal que

$$S = \{s, Proyecto3, Proyecto4, Area3\}$$

$$T = \{t, Proyecto1, Proyecto2, Area1, Area2\}$$

La conclusión que obtenemos de este resultado es que:

- Los proyectos a realizar son: Proyecto 3, Proyecto 4
- Las areas a contratar son: Area 3
- Los proyectos a descartar: Proyecto 1, Proyecto 2

- Las areas a no contratar son: Area1, Area 2

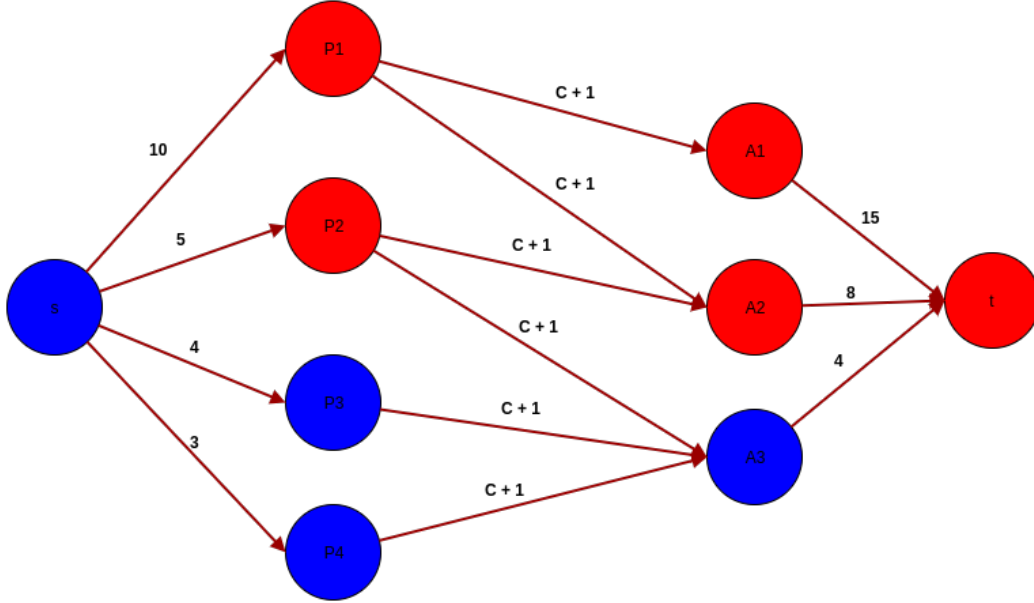


Figure 3: Resultado aplicado al grafo. En azul el conjunto S, en rojo el conjunto T.

Se puede observar que no hay ninguna arista de capacidad  $C + 1$  que contribuya a la capacidad del corte. Esto significa que no existe ningun proyecto  $i$  en el conjunto S tal que alguna de sus areas requeridas esté en el conjunto T, lo cual es correcto ya que en ese caso no se cumplirían las restricciones necesarias para la realización del proyecto.

## 2.2 Análisis de complejidad

Para realizar el análisis vamos a suponer lo siguiente. Sea  $G$  el grafo y sea  $m$  la cantidad de aristas y  $n$  la cantidad de vertices. Diremos que cada vértice tiene al menos una arista incidente, por lo tanto  $m \geq \frac{n}{2}$ , de modo tal que  $O(m + n) = O(m)$ , con el fin de simplificar las cuentas.

Lo primero que podemos observar es que necesitamos calcular un camino desde el nodo  $s$  hasta el nodo  $t$ . Utilizando BFS, logramos esto con una complejidad de  $O(m)$ .

Luego, aplicamos el método "augment", que tiene una complejidad de  $O(m)$  debido a que se calcula el cuello de botella del camino obtenido en el BFS, teniendo que iterar por todas las aristas, y luego se itera nuevamente cada arista para mejorar el flujo donde cada iteración realiza operaciones finitas en tiempo  $O(1)$ .

Lo que faltaría determinar es la cantidad de iteraciones que realiza este algoritmo. Dentro del método "augment", el método "bottleneck" calcula el cuello de botella y en función de ese resultado se mejora el flujo del camino obtenido. En el peor de los casos, tendríamos que en cada iteración a realizar obtengamos un cuello de botella de 1, provocando que tengamos que iterar hasta  $C$  veces como máximo para maximizar el flujo, siendo  $C = \sum_{e \text{ out of } s} c_e$ , para que de esta forma no quede ningún camino de  $s$  a  $t$  en el grafo residual y pueda terminar el algoritmo.

En conclusión, la complejidad del algoritmo es:

$$O(C * m)$$

Sea  $p$  el número de proyectos,  $a$  el número de áreas y  $r$  la cantidad total de requisitos por parte de todos los proyectos. Estos números determinan la cantidad de aristas que tendrá nuestro grafo sobre el cual aplicaremos el algoritmo de Ford-Fulkerson.

Dicha relación la podemos calcular teniendo en cuenta que para crear el grafo agregamos  $p$  aristas que unen al vértice  $s$  con cada proyecto, agregamos  $a$  aristas que unen cada área con el vértice  $t$ , y luego tenemos una arista por cada requisito entre un proyecto y un área, agregándose  $r$  aristas más. Por lo tanto tenemos que:

$$m = p + a + r$$

Por lo tanto, la complejidad del algoritmo utilizando estos parámetros es:

$$O(C * (p + a + r))$$

## 2.3 Análisis de tiempos

Para poder analizar los tiempos de ejecución, supondremos como casos una entrada donde el número de proyectos y el número de áreas esté fijado a 1000. Las ganancias y los costos los dejaremos fijo a una unidad. Variaremos la cantidad de restricciones  $r$  para ver como se comporta el algoritmo.  $C = \sum g_i$  no variará, por lo que solo tendrá influencia la cantidad de restricciones.

r	clocks
1000	94334
2000	142673
4000	301215
8000	266351
16000	348620
32000	723408
64000	1507858
128000	3052011
256000	5779835
512000	11684001

Table 3: Tabla de tiempos

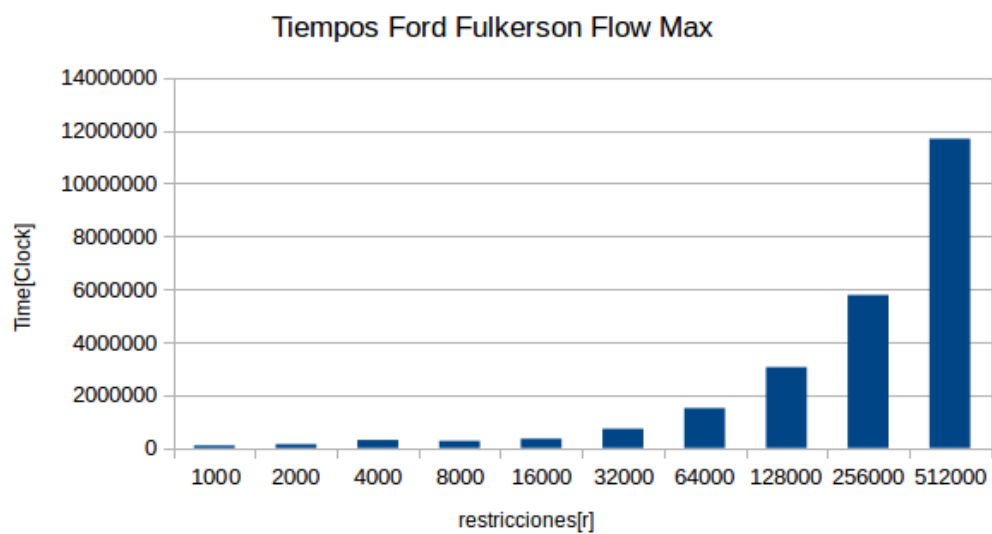


Figure 4: Gráfico de tiempos

Por medio del gráfico, se puede verificar el comportamiento lineal del algoritmo. Esto es, cuando duplicamos la cantidad de restricciones, se duplica el tiempo insumido para la ejecución del algoritmo.

## **2.4 Implementación del algoritmo**

### **Referencias**

- [1] Algorithm Design, Jon Kleinberg, Eva Tardos, 7.11 Project Selection