



# Teoría de Algoritmos I

Primer Cuatrimestre 2017

## Trabajo Práctico 3

Integrante	Padrón	Correo electrónico
Rodrigo De Rosa	97799	rodrigoderosa@outlook.com
Marcos Schapira	—	schapiramarcos@gmail.com
Facundo Guerrero	97981	facundoiguerrero@gmail.com

# Índice

<b>1. Programación Dinámica</b>	<b>1</b>
1.1. Algoritmo . . . . .	1
1.1.1. Funcionamiento . . . . .	1
1.1.2. Ecuación de recurrencia . . . . .	1
<b>2. Algoritmos Randomizados</b>	<b>2</b>
2.1. Algoritmo . . . . .	2
2.1.1. Funcionamiento . . . . .	2
2.1.2. Categoría de randomización . . . . .	2
<b>3. Algoritmos Aproximados</b>	<b>3</b>
3.1. Algoritmo . . . . .	3
3.1.1. Funcionamiento . . . . .	3
<b>4. Ejecución de programas</b>	<b>4</b>

# 1. Programación Dinámica

En esta sección se analiza una solución al problema de la predicción de acciones a través de la programación dinámica.

## 1.1. Algoritmo

El algoritmo utilizado para resolver el problema planteado fue

### 1.1.1. Funcionamiento

Este algoritmo funciona de la siguiente forma:

- Determina un día de compra, un día de venta, un día de compra auxiliar, una ganancia máxima y una ganancia temporal.
- Itera sobre todos los días (valores diferentes de acciones) verificando si en el día actual es más o menos favorable comprar acciones que en el día en el que se pretendía hacerlo hasta el momento, determinando el día de compra auxiliar.
- A partir del día que determinó, calcula la ganancia temporal como la que se obtendría si las acciones fueran vendidas el día actual y verifica si es mayor a la ganancia máxima hasta el momento.
- En tal caso, determina el día de venta como el actual, el día de compra como el que previamente era el día de compra auxiliar y la ganancia máxima como la que era la ganancia temporal.
- Al finalizar la iteración, queda determinado el día de compra más conveniente, el día de venta más conveniente y la ganancia máxima obtenible.

### 1.1.2. Ecuación de recurrencia

Para la ecuación de recurrencia se plantea lo siguiente:

- Se tiene una variable  $C_i$  = Día en el que se compran las acciones hasta el paso  $i$ , con  $i = 1, \dots, n$ .
- Además, se tiene otra variable  $V_k$  = Día en el que se venden las acciones hasta el paso  $k$ , con  $k = i, \dots, n$ .
- Observar que  $k$  está relacionada con  $i$ , ya que el día de venta debe ser mayor o igual al día de compra. En el caso de ser igual, la ganancia sería 0.
- Se puede obtener la Ganancia Temporal del paso  $i, k$ , como  $GT_{i,k} = V_k - C_i$ .
- Entonces, la Ganancia Máxima es la máxima  $GT_{i,k}$  obtenida al final del algoritmo.
- Se puede definir la ganancia máxima para el paso  $i, k$  como:  $GM_{i,k} = \begin{cases} V_k - C_i & \text{si } GT_{i,k} > GM_{i,k} \\ GT_{i,k} - 1 & \text{si } GT_{i,k} \leq GM_{i,k} \end{cases}$

## 2. Algoritmos Randomizados

En esta sección se analiza una solución al problema de hallar el corte global mínimo en un grafo no dirigido a través de un algoritmo randomizado.

### 2.1. Algoritmo

Para resolver este problema se utilizó el algoritmo de Karger descrito en la bibliografía proporcionada por la cátedra.

#### 2.1.1. Funcionamiento

Sea el grafo  $G = (E, V)$ , el procedimiento del algoritmo es el siguiente:

- Mientras  $|V| > 2$ :
  - Se elige  $e(u, v) \in E$  aleatoriamente.
  - Se crea un  $w \in V$ , el cual reemplaza tanto a  $u$  como a  $v$  en todas las aristas en las que se encuentran. Es decir,  $w$  puede tener más de una arista que vaya a un mismo vértice  $q \in V$ .
  - Se elimina  $e(u, v)$  de  $E$ .
  - Si existe alguna  $e(v, v) \in E$  (arista de un vértice consigo mismo), se elimina.
- Se devuelven las aristas que unen a esos dos vértices como el corte mínimo.

#### 2.1.2. Categoría de randomización

Es un algoritmo *Monte-Carlo* porque para algún orden de selección aleatoria de aristas, el corte obtenido *no* es el mínimo. Es decir, es rápido siempre pero no siempre da resultados correctos.

La probabilidad de que este algoritmo devuelva un corte que sea mínimo es  $p \geq \left(\frac{n}{2}\right)^{-1}$  con  $n = |V|$ . Un dato adicional es que si el algoritmo se corre  $T = \left(\frac{n}{2}\right) \ln n$  veces, la probabilidad de no encontrar un corte mínimo es  $[1 - p]^T \leq \frac{1}{n}$  en un tiempo  $O(Tm) = O(n^2 m \log n)$  con  $m = |E|$ .

### **3. Algoritmos Aproximados**

En esta sección se analiza una solución al problema de la suma de subconjuntos a través de un algoritmo aproximado.

#### **3.1. Algoritmo**

Para resolver este problema se utilizó la estrategia polinómica descrita en la bibliografía proporcionada por la cátedra.

##### **3.1.1. Funcionamiento**

Este algoritmo

## 4. Ejecución de programas

Para correr cada algoritmo, se debe ejecutar el archivo principal de cada uno. Esto se hace de la siguiente forma:

En la carpeta `Programación Dinámica` abrir la consola y ejecutar `python main.py`

En la carpeta `Algoritmos Randomizados` abrir la consola y ejecutra `python main.py`

En la carpeta `Algoritmos Aproximados` abrir la consola y ejecutra `python main.py`