



# Teoría de Algoritmos I

Primer Cuatrimestre 2017

## Trabajo Práctico 3

Integrante	Padrón	Correo electrónico
Rodrigo De Rosa	97799	rodrigoderosa@outlook.com
Marcos Schapira	97934	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>3</b>
2.1. Algoritmo . . . . .	3
2.1.1. Funcionamiento . . . . .	3
2.1.2. Categoría de randomización . . . . .	3
<b>3. Algoritmos Aproximados</b>	<b>4</b>
3.0.1. Funcionamiento . . . . .	4
3.0.2. Análisis del Algoritmo . . . . .	4
<b>4. Ejecución de programas</b>	<b>5</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 esta basado en el algoritmo de kadane. Este busca la maxima suma de elementos contiguos dentro de un arreglo.

### 1.1.1. Funcionamiento

Este algoritmo funciona de la siguiente forma:

- Inicializa un **día de compra**, un **día de venta**, un **día de compra auxiliar**, todos como el primer día. Tambien se inicializa una **ganancia máxima** y una **ganancia temporal**, ambas como 0 ya que, hasta el momento, el día de compra es igual al día de venta.
- Luego itera sobre todos los días (valores diferentes de acciones) verificando si en el día actual( día  $i$  ) es más o menos favorable comprar acciones que en el día en el que se pretendía hacerlo hasta el momento( día  $k$  ), determinando el **día de compra auxiliar**. Esto asegura la obtencion de la mayor ganancia hasta el día  $i-1$ .
- A partir del día que determinó, calcula la **ganancia temporal** como la ganancia que se obtendría si las acciones fueran compradas en el **día de compra** y vendidas el **día actual**. Luego se verifica si la **ganancia temporal** es mayor a la **ganancia máxima**.
- 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.
- Dado que el algoritmo propuesto recorre una sola vez el arreglo, funciona en  $O(n)$ .

### 1.1.2. Ecuación de recurrencia

Para la ecuación de recurrencia se plantea lo siguiente:

- Para empezar, se debe tener en cuenta las siguientes consideraciones:
- Para la resolución del algoritmo se utilizan las siguientes variables:  $G_M$  = Ganancia Máxima,  $G_T$  = Ganancia Temporal,  $D_i$  = Día actual,  $D_V$  = Día de Venta,  $D_C$  = Día de Compra y  $D_{CAux}$  = Día de Compra Auxiliar.
- Para el paso  $i$  de la iteración, el  $D_{CAux}$  se actualiza si  $D_i < D_C$ . Esto se puede hacer porque en el paso  $i$  ya se tiene la ganancia máxima calculada y guardada en  $G_M$ . Como ahora se tiene un día de compra mejor al que se tenía hasta el paso  $i - 1$ , la mejor ganancia que se pudo obtener hasta el paso  $i - 1$ , es la que ya se obtuvo hasta el momento. Entonces se llevan las nuevas ganancias en  $G_T$  para poder observar si con el nuevo día de compra se puede obtener una mejor ganancia.
- La ganancia temporal se calcula como:  $G_T = D_i - D_{CAux}$  y la ganancia máxima se calcula como:  $G_M = D_V - D_C$ . Entonces, se actualiza la ganancia máxima cuando  $G_M < G_T$ . Notar que en dicho caso tambien se actualizan los días de compra y venta.
- Entonces, se define  $V_i$  como el valor de la acción en el día  $i$ .
- Con las consideraciones anteriormente mencionadas, se puede notar que la ganancia en el día  $i$  es: la ganancia del paso anterior, en el caso donde el valor de la acción es mayor en el día anterior que en el día actual, o el máximo entre la ganancia del día anterior y la ganancia del día actual.
- Consideremos el siguiente caso de valores de acciones:  $[2,3,4,1,4,5]$ . Cuando  $i = 0$  el día de compra y el día de venta son iguales entonces la ganancia es 0. Cuando  $i = 1$ , ahora el día de venta es 1 y el día de compra es 0, por lo tanto la ganancia es 1. Inductivamente sabemos que para  $i = 2$ , la ganancia es 2. Pero en el caso en que  $i = 3$  el día de compra pasa a ser 3, ya que la acción es mas barata, y la ganancia pasa a ser 0. Por eso, se dice que la ganancia del paso  $i$  es el maximo entre la ganancia del día anterior y la del día actual. En este caso se puede observar que la ganancia del día anterior es 2 y la del día actual es 0, entonces corresponde la ganancia del día anterior.

- Entonces, se define la ganancia temporal para el paso  $i$  como:

$$G_{Mi} = \begin{cases} G_T[i-1] & \text{si } V_i \leq V_{i-1} \\ V_i - D_{CAux} & \text{si } V_i > V_{i-1} \end{cases}$$

- Finalmente, vale notar que la ganancia máxima al finalizar la iteración será la ganancia temporal calculada en el paso  $n - \text{esimo}$ .
- Se ve claramente que el orden de dicho algoritmo es  $O(n)$  ya que solamente se recorre el arreglo una vez para obtener la máxima ganancia.

## 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-Stein descrito en la bibliografía proporcionada por la cátedra.

#### 2.1.1. Funcionamiento

Primero presentaremos el algoritmo de Karger, dado que el algoritmo de Karger-Stein es una optimización sobre este mismo:

**Karger:**

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

- Mientras  $|V| > t$ :
  - 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 devuelve el grafo resultante con  $t$  vertices.

**Karger-Stein:**

Al algoritmo arriba descrito lo llamaremos **MinCut** y procederemos a explicar como funciona la optimización de Karger-Stein sobre el algoritmo original. Esta optimización utiliza un algoritmo recursivo, que llamaremos **FastMinCut** y que recibe como parámetro a un grafo  $G = (V, E)$ . El funcionamiento es el siguiente:

- Si  $|V| > 8$ :
  - $t < -\frac{|V|}{\sqrt{2}} + 1$
  - Se obtienen dos grafos,  $G'_1$  y  $G'_2$  a través de llamados a **FastMinCut**( **MinCut**( $G$ ,  $t$ ) ).
  - Se verifica cuál de los dos grafos recibidos tiene el menor tamaño y se devuelve.
- Sino:
  - Se hace un llamado a **MinCut**( $G$ , 2). Es decir, se utiliza Karger tradicional.

#### 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 el algoritmo original devuelva un corte que sea mínimo es  $p \geq \binom{n}{2}^{-1}$  con  $n = |V|$ . Un dato adicional es que si el algoritmo se corre  $T = \binom{n}{2} \ln n$  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  $m = |E|$ . En cuanto a la optimización de Karger-Stein, la probabilidad es  $P(n) = \frac{1}{\log n}$ .

### 3. Algoritmos Aproximados

En esta sección se analiza una solución al problema de la suma de subconjuntos a través de un algoritmo aproximado. Para resolver este problema se utilizó la estrategia polinómica descrita en la bibliografía proporcionada por la cátedra.

#### 3.0.1. Funcionamiento

El problema de la suma de subconjuntos (subset sum) consiste en, a partir de un conjunto  $S$  de enteros positivos y un target  $t$  también entero positivo, saber si existe algún subconjunto de  $S$  cuya suma sea exactamente  $t$ . Este problema es NP-Completo.

A partir de él se puede derivar a una aproximación completamente polinómica mediante el “recorte” o “trimming” de cada subconjunto que se va generando en el algoritmo exacto. Este mecanismo se sostiene de la idea de que si dos números pertenecen a  $S$  y tienen valores similares entonces no tiene mucho sentido mantener a ambos explícitamente (en referencia al algoritmo aproximado). Así es como mediante un parámetro de aproximación  $\sigma$  tal que:  $0 < \sigma < 1$

Se eliminan tantos elementos de  $S$  como sea posible ya que por cada elemento eliminado va a haber otro que pertenezca a  $S$  y lo represente. Así es como el algoritmo logra dado un conjunto  $S$  y un parámetro  $t$  devolver la mayor suma de elementos menor o igual a  $t$ . A la vez el algoritmo obtiene por parámetro a  $\sigma$ , con lo cual la suma que devuelve está a un factor de  $(1 + \sigma)$  del valor real.

#### 3.0.2. Análisis del Algoritmo

La tabla a continuación muestra resultados del algoritmo con  $\sigma$  variables. A la vez los elementos en cada instancia y el valor de  $t$  fueron generados aleatoriamente.  $Z$  es el valor devuelto por el algoritmo.

Cuadro 1: Resultados para $N = 350$			
T	$\sigma$	Zreal	Z real porcentual
2213	0,94	2211	%99,9
2246	0,46	2245	%99,9
2182	0,65	2182	%100
2620	0,74	2618	%99,9
173	0,62	172	%99,4

Como se puede apreciar, los porcentajes son extremadamente altos y caen dentro del factor esperado.

## 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`