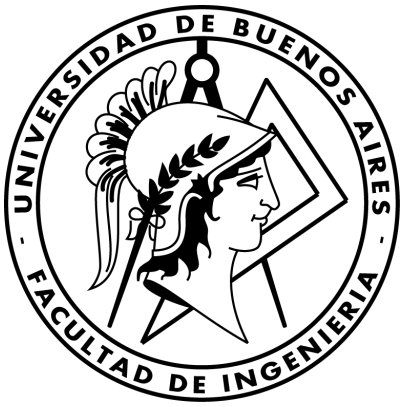

Trabajo Práctico N°3

75.29 - Teoría de Algoritmos I

Facultad de Ingeniería de la Universidad de Buenos Aires

1er. Cuatrimestre 2017

Federico Brasburg, *Padrón Nro. 96.653*
federico.brasburg.@gmail.com

Pablo Rodrigo Ciruzzi, *Padrón Nro. 95.748*
p.ciruzzi@hotmail.com

Andrés Otero, *Padrón Nro. 96.604*
oteroandres95@gmail.com

23 de junio de 2017

Índice

1. Programación Dinámica **3**
1.1. Cómo correrlo 3

2. Algoritmos Randomizados **3**
2.1. Cómo correrlo 3

3. Algoritmos Aproximados **3**
3.1. Cómo correrlo 3

4. Código **4**

1. Programación Dinámica

1.1. Cómo correrlo

2. Algoritmos Randomizados

2.1. Cómo correrlo

3. Algoritmos Aproximados

3.1. Cómo correrlo

4. Código

creador_grafos.py

```
import random


def crearDigrafoCompleto(n, nombre):
    cantVertices = n * (n - 1)
    arch = open(nombre, 'w')
    arch.write(str(n) + "\n")
    arch.write(str(cantVertices) + "\n")
    for i in range(n):
        for j in range(n):
            if i != j:
                arch.write(str(i) + " " + str(j) + " " + str(random.random() * 2) + " " + "\n")
                    # Crea una arista del vertice i al vertice j con un valor al azar entre 0 y 2
    arch.close()
```

grafo.py

```
PRIMERO = 0
SEGUNDO = 1
TERCERO = 2


class Arista(object):
    def __init__(self, id1, id2, peso):
        self.id1 = id1
        self.id2 = id2
        self.peso = peso

    def peso(self):
        return self.peso

    def __str__(self):
        return str(self.id1) + " a " + str(self.id2) + ", peso " + str(self.peso)


class Grafo(object):
    def __init__(self):
        """Crea un Grafo dirigido (o no) con aristas pesadas (o no)"""
        self.aristas = {}
        self.vertices = []
        self.aristas_list = []

    def devolver_aristas(self):
        """Devuelve las aristas del grafo"""
        return self.aristas

    def devolver_aristas_list(self):
        return self.aristas_list

    def devolver_vertices(self):
        return self.vertices
```

```
def devolver_cant_vertices(self):
    """Devuelve los nodos del grafo"""
    return len(self.vertices)

def agregar_vertice(self, id):
    """Agrega un vertice que se identifica con un nombre y un ID"""
    self.vertices.append(id)
    self.aristas[id] = {}

def agregar_arista_no_dirigida(self, id1, id2, peso=0):
    """Agrego una arista no dirigida entre los nodos con id1 y id2"""
    self.agregar_arista_dirigida(id1, id2, peso)
    self.agregar_arista_dirigida(id2, id1, peso)

def agregar_arista_dirigida(self, id1, id2, peso=0):
    """Agrego una arista dirigida entre los nodos con id1 y id2"""
    arista = Arista(id1, id2, peso)
    self.aristas[id1][id2] = arista
    self.aristas_list.append(arista)

def son_vecinos(self, id1, id2):
    """Devuelve si id1 y id2 son vecinos"""
    try:
        if self.aristas[id1][id2]:
            return True
        return False
    except:
        return False

def peso_arista(self, id1, id2):
    """Devuelve el peso de la arista entre id1 e id2"""
    if self.son_vecinos(id1, id2):
        return self.aristas[id1][id2].peso
    raise ValueError

def adyacentes(self, id):
    """Pide un id de un nodo existe y devuelve una lista de los id de sus adyacentes"""
    adyacentes = []
    for arista in self.aristas[id]:
        adyacentes.append(arista)
    return adyacentes

def _leer(self, nombre, dirigido=False):
    """Lee un grafo (dirigido o no) de un archivo con nombre"""
    try:
        mi_arch = open(nombre)
        cant_nodos = int(mi_arch.readline())
        for i in range(0, cant_nodos):
            self.agregar_vertice(i)
        cant_aristas = int(mi_arch.readline())
        for i in range(0, cant_aristas):
            linea = mi_arch.readline()
            numeros = linea.split(" ")
            peso = 0
            if len(numeros) > 2:
                peso = numeros[TERCERO].rstrip('\n')
```

```
                else:
                    numeros[SEGUNDO] = numeros[SEGUNDO].rstrip('\n')
                if dirigido:
                    self.agregar_arista_dirigida(int(numeros[PRIMERO]), int(numeros[SEGUNDO
                        ]), peso)
                else:
                    self.agregar_arista_no_dirigida(int(numeros[PRIMERO]), int(numeros[
                        SEGUNDO]), peso)
            mi_arch.close()
            return True
        except:
            return False

    def leer_dirigido(self, nombre):
        self._leer(nombre, True)

    def leer_no_dirigido(self, nombre):
        self._leer(nombre, False)
```

parser.py

```
from grafo import Grafo

CERO = 0
UNO = 1


class Parser(object):
    def leer_grafo_no_dirigido(self, nombre):
        """Lee un archivo de un grafo no dirigido sin peso"""
        try:
            grafo = Grafo()
            grafo.leer_no_dirigido(nombre)
            return grafo
        except:
            print "Ocurrio un error leyendo el archivo de grafo no dirigido " + nombre
            return False

    def leer_grafo_dirigido(self, nombre):
        """Lee un archivo de un grafo dirigido sin peso"""
        try:
            grafo = Grafo()
            grafo.leer_dirigido(nombre)
            return grafo
        except:
            print "Ocurrio un error leyendo el archivo de grafo dirigido " + nombre
            return False
```