FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Algoritmos D&C y Programacion Dinamica

[75.29/95.06] Teoria de Algoritmos I
Primer cuatrimestre de 2022

Alumno	Padron	Email
BENITO, Agustin	108100	abenito@fi.uba.ar
BLÁZQUEZ, Sebastián	99673	sblazquez@fi.uba.ar
DEALBERA, Pablo Andres	106585	pdealbera@fi.uba.ar
DUARTE, Luciano	105604	lduarte@fi.uba.ar
PICCO, Martín	99289	mpicco@fi.uba.ar

Entrega:	Primera
Fecha:	Miercoles 27 de Abril del 2022

Índice

1. Parte 1: Un evento exclusivo

Todos los años la asociación de un importante deporte individual profesional realiza una preclasificación de los n jugadores que terminaron en las mejores posiciones del ranking para un evento exclusivo. En la tarjeta de invitación que enviarán suelen adjuntar el número de posición en la que está actualmente y a cuantos rivales superó en el ranking (únicamente entre los invitados). Contamos con un listado que tiene el nombre del jugador y la posición del ranking del año pasado. Ese listado está ordenado por el ranking actual.

Ejemplo:

```
A,3 | B,4 | C,2 | D,8 | E,6 | F,5 |

A → Ranking actual 1 → superó a 1 entre los preclasificados (C)
B → Ranking actual 2 → superó a 1 entre los preclasificados (C)
C → Ranking actual 3 → superó a 0 entre los preclasificados (-)
D → Ranking actual 4 → superó a 2 entre los preclasificados (E y F)
E → Ranking actual 5 → superó a 1 entre los preclasificados (F)
F → Ranking actual 6 → superó a 0 entre los preclasificados (-)
```

En este caso el problema debería retornar:

```
A → 1 (1)
B → 2 (1)
C → 3 (0)
D → 4 (2)
E → 5 (1)
F → 6 (0)
```

Se pide:

- Explicar cómo se puede resolver este problema por fuerza bruta. Analizar complejidad espacial y temporal de esta solución
- Proponer una solución utilizando la metodología de división y conquista que sea más eficiente que la propuesta anterior. (incluya pseudocódigo y explicación)
- Realizar el análisis de complejidad temporal mediante el uso del teorema maestro.
- Realizar el análisis de complejidad temporal desenrollando la recurrencia
- Analizar la complejidad espacial basándose en el pseudocódigo.
- Dar un ejemplo completo del funcionamiento de su solución

2. Parte 2: Ciclos negativos

La detección de ciclos negativos tiene una variedad de aplicaciones en varios campos. Por ejemplo en el diseño de circuitos electrónicos VLSI, se requiere aislar los bucles de retroalimentación negativa. Estos corresponden a ciclos de costo negativo en el grafo de ganancia del amplificador del circuito. Tomando como entrada de nuestro problema un grafo ponderado con valores enteros (positivos y/o negativos) dirigido donde un nodo corresponde al punto de partida, queremos conocer si existe al menos un ciclo negativo y en caso afirmativo mostrarlo en pantalla.

Se pide:

- Proponer una solución al problema que utiliza programación dinámica. Incluya relación de recurrencia, pseudocódigo, estructuras de datos utilizadas y explicación en prosa.

- Analice la complejidad temporal y espacial de su propuesta.
- Programe la solución
- Determine si su programa tiene la misma complejidad que su propuesta teórica.

Formato de los archivos:
El programa debe recibir por parámetro el path del archivo donde se encuentra el grafo. El archivo con el grafo es un archivo de texto donde la primera línea corresponde al nodo inicial. Luego continúa con una línea por cada eje direccionado del grafo con el formato: ORIGEN,DESTINO,PESO.
Ejemplo: "grafo.txt"

```
B
D,A,-2
B,A,3
D,C,2
C,D,-1
B,E,2
E,D,-2
A,E,3
...
```

Debe resolver el problema y retornar por pantalla la solución.
En caso de no existir ciclos negativos: "No existen ciclos negativos en el grafo"
En caso de existir ciclos negativos: "Existe al menos un ciclo negativo en el grafo. A,E,D → costo: -1"

2.1. Detalles de implementación

El algoritmo fue implementado en Python y no tiene dependencias aparte de tener instalado cualquier versión de `python3`.

2.1.1. Ejecución del programa

El programa contiene un `shebang` para ser ejecutado en una terminal de la siguiente forma:

```
./src/parte_2.py <filename>
```

El comprimido entregado incluye un archivo ejemplo en `assets/grafo.txt` con grafos ejemplos, por ejemplo:

```
./src/parte_2.py ./assets/grafo.txt
```

```
Graph({'D': {'A': '-2', 'C': '2'}, 'B': {'A': '3', 'E': '2'}, 'C': {'D': '-1'}, 'E': {'D': '-2'},
```

3. Parte 3: Un poco de teoría

1. Hasta el momento hemos visto 3 formas distintas de resolver problemas. Greedy, división y conquista y programación dinámica.

   a) Describa brevemente en qué consiste cada una de ellas

   b) ¿Cuál es la mejor de las 3? ¿Podría elegir una técnica sobre las otras?

2. Un determinado problema puede ser resuelto tanto por un algoritmo Greedy, como por un algoritmo de División y Conquista. El algoritmo greedy realiza N^3 operaciones sobre una matriz, mientras que el algoritmo de Programación Dinámica realiza N^2 operaciones en total, pero divide el problema en N^2 subproblemas a su vez, los cuales debe ir almacenando mientras se ejecuta el algoritmo. Teniendo en cuenta los recursos computacionales involucrados (CPU, memoria, disco) ¿Qué algoritmo elegiría para resolver el problema y por qué?

Pista: probablemente no haya una respuesta correcta para este problema, solo justificaciones correctas