aaa

Índice

1. Introducción

1.1. Objetivos

Mediante la realización de este trabajo práctico se pretende realizar un acercamiento al análisis e implementación de técnicas algorítmicas avanzadas para resolución de problemas, como así también a las estructuras que permiten su implementación.

En esta ocasión se hace énfasis en las técnicas que involucran el uso de *grafos*, con los distintos algoritmos que permiten recorrerlos, y los denominados *algoritmos dinámicos*.

1.2. Pautas de trabajo

Se brindan tres problemas, escritos en términos coloquiales, en donde para cada uno de ellos se requiere **encontrar un algoritmo** que brinde una **solución particular**, acotado por una determinada **complejidad temporal**. El algoritmo debe ser **implementado** en un lenguaje de programación a elección. Los datos son proporcionados y deben ser devueltos bajo formatos específicos de *input* y de *output*.

Posteriormente se deben realizar análisis teóricos y empíricos tanto de la de **correctitud** como de la **complejidad temporal** para cada una de las soluciones propuestas.

1.3. Metodología utilizada

Para cada ejercicio, se brinda primeramente una **descripción** del problema planteado, a partir de la cual se realiza una **abstracción** hacia un **modelo formal**, que permite tener un **entendimiento preciso** de las pautas requeridas.

Se expone, cuando las hay, una **enumeración de las características** elementales del problema; estas son aquellas que permiten **encuadrarlo** dentro de una **familia de problemas** típicos.

Se desarrolla posteriormente un análisis del conjunto **universo de posibles soluciones** (o factibles), caracterizando matemáticamente el concepto de **solución correcta** y, en los casos en que se solicita **optimización**[1], se definen las condiciones que dan forma ya sea a todo el subconjunto de **soluciones óptimas** que se encuadran dentro de las pretenciones del problema, o a una **solución particular** dentro del mismo (la cual denominamos *mejor solución*).

Luego de caracterizar para todo conjunto posible de entradas «*cómo se compone el conjunto solución*» correspondiente, se desarrolla un **pseudocódigo** en el que se expone «*cómo llegar a ese conjunto*»[2].

Habiendo planteado la **hipótesis de resolución** se demuestra, de manera informal o mediante inferencias matemáticas según sea necesario, que el **algoritmo propuesto** realmente permite obtener la **solución correcta**[3].

Después de demostrar la **correctitud de la solución**, y su **optimalidad** en caso de existir varias soluciones correctas, se realiza un análisis teórico de la **complejidad temporal** en donde se estima el comportamiento del algoritmo en términos de tiempo. Este análisis en particular se realiza con el objetivo de obtener una *cota superior asintótica*.

[1] Es decir, que la solución pertenezca al *subconjunto de soluciones que* ***maximicen*** *o* ***minimicen*** *una determinada función*

[2] Una explicación coloquial, obviando detalles puramente implementativos: arquitectura, lenguaje, etc.

[3] En caso de existir más de una solución correcta, se demuestra que el algoritmo obtiene al menos una de ellas o, dicho de otro modo, para problemas de optimización, se demuestra que ninguna del resto de las soluciones correctas es mejor que la solución propuesta por nuestro algoritmo

Luego de calcular la **cota de complejidad temporal**, se realiza una **verificación** empírica, junto con una **exposición gráfica** de los resultados obtenidos, mediante la combinación de técnicas básicas de medición y análisis de datos.

2. Instrucciones de uso

2.1. Herramientas utilizadas

Para la realización de este trabajo se utilizaron un conjunto de herramientas, las cuales se enumeran a continuación:

- `C++` como lenguaje de programación
  - `gcc` como compilador de C++
- `python` y `bash` para la realización de scripts
  - `python` para generar casos de prueba
  - `bash` para automatizar las mediciones
  - `python/matplotlib` para plotear los gráficos
- LaTeX para la redacción de este documento
- Se testeó bajo los siguientes Sistemas Operativos
  - `Debian GNU/Linux`
  - `Ubuntu`
  - `FreeBSD`, compilando a través de `gmake`
  - `Windows`, a través de `cygwin`

3. Desarrollo del TP

3.1. Problema 1: Robanúmeros

3.1.1. Descripción

En este problema se tiene que crear un algoritmo que juegue al *Robanúmeros* de forma tal que el jugador1 logre el mejor juego posible, y el jugador2 juegue de manera óptina durante cadu turno que le toque. El algoritmo tiene que tener una complejidad temporal de peor caso de $\mathcal{O}(n^3)$, con n la cantidad de cartas iniciales.

Reglas del *Robanúmeros*:

- Comienzo del juego:
  Se tiene una cantidad n ($n \in \mathbb{N}$) de cartas con valores enteros alineadas horizontalmente (c1, c2, ..., cn) sobre la mesa. Las cartas tienen que estar boca arriba.
- Turnos:
  Participan 2 jugadores, cada uno va alternando un turno.(Total de turnos t: 1,..,n).
- Elección de cartas:
  En cada turno el jugador tiene que elegir un extremo, el izquierdo (izq) o el derecho (der), de la secuencia de cartas desde el que irá tomando de 1 a n de las cartas adyacentes que están en la mesa. La cantidad de cartas elegidas variará según le sea conveniente al jugador, pero por lo menos tiene que tomar una carta en su turno.
- Fin del juego:
  El juego finaliza cuando no hay más cartas sobre la mesa. Se suman las cartas de cada jugador (p1: Ptos. Jug1, p2: Ptos. Jug2). Gana el que obtiene el mayor puntaje.

Ejemplo 3.1.1.1.

- Cartas iniciales:

2	-3	-2	5	5

- Turno1 (Jug1): Elige el extremo derecho y toma las 2 últimas cartas.

5	5

- Quedan sobre la mesa:

2	-3	-2

- Turno2 (Jug2): Elige el extremo izquierdo y toma 1 carta.

2

- Quedan sobre la mesa:

-3	-2

- Turno3 (Jug1): Elige el extremo derecho y toma 1 carta.

-2

- Quedan sobre la mesa:

-3

- Turno4 (Jug2): Sólo queda una carta, por lo que elige ésta. Es indistinto para este caso si el extremo elegido es el izquierdo o el derecho. | -3 |

- Finaliza el juego porque no hay más cartas. Se suman los puntajes de cada jugador.

Ptos. Jug1	Ptos. Jug2
5 + 5 + (-2) = 8	2 + (-3) = -1

- Formato de entrada y salida:

Input: 5 2 -3 -2 5 5

Output: 4 8 -1
der 2
izq 1
der 1
izq 1

Ejemplo 3.1.1.2.

2	-1	6

El Jug1 toma todas las cartas porque de esta manera obtiene el mayor puntaje.
Finaliza el juego en 1 turno porque no hay más cartas. Se suman los puntajes de cada jugador.
Este tipo de caso también se daría si todas las cartas tuvieran números positivos, sólo llegaría a jugar el Jug1.

Ptos. Jug1	Ptos. Jug2
2 + (-1) + 6 = 7	0

3.1.2. Planteamiento de resolución

Para una mayor claridad, vamos a reducir el problema a encontrar la mayor cantidad de puntos que se pueden sacar con el juego de cartas dado.
Sea f(i,j) = ”maxima cantidad de puntos que se pueden sacar en el juego que consiste en las cartas que estaban desde la posición i
hasta la j en el juego de cartas original”.
Sea suma(i,j) la suma de las cartas que estaban entre la posición i y j en el juego de cartas original.
Nuestro algoritmo se basa en que f(i,j) = — valor de la carta i si i=j
— suma(i, j) - min(f(i’, j’)) para todo i’j’que representen un juego que le dejo al oponente usando una movida válida sino

[habra que justificar esto?]

3.1.3. Justificación formal de correctitud

3.1.4. Cota de complejidad temporal

3.1.5. Verificación mediante casos de prueba

A continuación presentamos distintas instancias que sirven para verificar que el programa funciona correctamente.

Input			
n	c_1	...	c_n

Output		
t	p1	p2
e_1	c_1	
⋮	⋮	
e_t	c_t	

- n: #cartas iniciales.
- c_i con $1 \leq i \leq n$: c_i valor de la carta i.
- t: #turnos del juego.
- p1: puntaje total Jug1.
- p2: puntaje total Jug2.
- e_i con $1 \leq i \leq t$: e_i extremo elegido por el jugador en el turno i (izq o der).
- c_i con $1 \leq i \leq t$: c_i #cartas tomadas por el jugador en el turno i.

Según los valores de p1 y p2, podemos separar en 3 casos posibles:

1. Caso Empate entre Jug1 y Jug2:
   - Cartas con valor cero

Input			
3	0	0	0

Output			
1	0	0	
izq	3		

   - Cartas con valores negativos

Input			
3	-1	-2	-3

Output			
2	-3	-3	
izq	2		
izq	1		

2. Caso Perdedor Jug1:
   - Cartas con valores negativos

Input			
3	-2	-3	-1

Output			
2	-4	-2	
der	2		
izq	1		

3. Caso Ganador Jug1:
   - Cartas con valores positivos

Input			
3	1	2	3

Output			
1	6	0	
izq	3		

- Cartas con valores negativos

Input			
3	-5	-1	-3

Output			
	2	-4	-5
	der	2	
	izq	1	

- Cartas con valores positivos y negativos

Input				
4	2	-8	-8	3

Output			
	4	-5	-6
	der	1	
	izq	1	
	izq	1	
	izq	1	

Ejecutamos el programa con los distintos ejemplos y se llegó a la solución esperada. Por lo tanto, podemos concluir que el comportamiento del programa es correcto.

3.2. Problema 2: La centralita (de gas)

3.2.1. Descripción

3.2.2. Planteamiento de resolución

3.2.3. Justificación formal de correctitud

3.2.4. Cota de complejidad temporal

3.2.5. Verificación mediante casos de prueba

3.3. Problema 3: Saltos en La Matrix

3.3.1. Descripción

3.3.2. Planteamiento de resolución

3.3.3. Justificación formal de correctitud

3.3.4. Cota de complejidad temporal

3.3.5. Verificación mediante casos de prueba

4. Apéndices

4.1. Código Fuente (resumen)