Análisis de Algoritmos

Bingo 2.0

GRUPO #5

1. Barzola Valeria
2. Marquez Joangie
3. Riofrío Adriana
4. Vélez Alex

Tabla de Contenidos

Definición Formal del Problema

Dados 3 archivos csv, donde cada uno corresponde a un color (amarillo, azul o rojo) y contiene una cantidad N de tablas. Cada línea del archivo representa una tabla y tiene una estructura id, num1, num2, ..., numM, siendo M la cantidad de números de la tabla (que depende del color de la tabla). Las tablas amarillas y azules tienen 14 números, mientras que las rojas tienen 11. El juego inicia con una ronda donde los participantes usan las tablas de color amarillo, seguida de una ronda para utilizar las tablas de color azul y termina con las tablas de color rojo. En cada ronda, el usuario irá ingresando cada bolita que se haya sacado del bombo hasta completar el número de bolitas correspondientes al color de la ronda. Al terminar la ronda se deberá presentar el id de la tabla ganadora; sin embargo, en el caso de que no tenga ninguna tabla ganadora y no haya ganado ningún otro participante, se procederá a ingresar otra bolita para finalmente presentar el id de la tabla ganadora si tuviese alguna o presentar que no tiene tablas ganadoras.

Opciones candidatas

Para este proyecto propusimos algunas opciones para desarrollar el algoritmo con el que se solucionaría el problema, pero al final se eligió la alternativa que ofrecía más ventajas en relación a la eficiencia tanto de tiempo como de espacio.

Los dos aspectos principales que consideramos fueron, por cada ronda, cómo almacenar la información de las tablas en memoria y cómo se marcarían las bolitas que salieran para al final determinar si el usuario tiene o no una tabla ganadora.

Una de las primeras ideas fue guardar, por cada tabla, los siguientes elementos: el id y una lista de enteros con todos los números de la tabla, de manera que al sacar una bolita se recorra toda la lista para verificar que el número de la bolita se encuentra en esta; si el número está, entonces se elimina. Al final, una tabla sería ganadora si su lista de números asociada está vacía. Sin embargo, esta opción era un poco ineficiente en términos del tiempo

de ejecución, que sería O(n), ya que tendríamos que explorar toda la lista de números para determinar si un número estaba o no en la tabla.

Para mitigar eso, pensamos en ordenar los números de esa lista, de modo que al ingresar una bolita se haga una búsqueda binaria y con esto mejorar la eficiencia del tiempo de ejecución al momento de ingresar una bolita a O(lg n). Aún así, una desventaja de esta segunda opción es que el tiempo para cargar las tablas al leer archivo aumentaría, puesto que tendríamos que ordenar los números de la lista.

Para evitar que el tiempo de carga sea mayor debido al ordenamiento de los números y para mejorar el tiempo de búsqueda al ingresar una bolita, al final se optó por tener un arreglo de booleanos del 1 al 20 en lugar de una lista con los números de la tabla; cada posible número del bombo está representado por un índice en el arreglo y el valor del arreglo en ese índice indica si el número correspondiente se encuentra o no en la tabla. En el arreglo se marcan todos los elementos como falso, a excepción de aquellos cuyos índices corresponden a los números de la tabla. Por lo tanto, al ingresar una bolita, solo se toma el valor del arreglo en el índice del número de la bolita para verificar si está o no en la tabla, lo que sería un tiempo O(1). Si es verdadero, entonces el número está en la tabla y se cambia a falso. Para ver si esa tabla es ganadora, solo se recorre el arreglo y se comprueba que el valor de todos los elementos es falso.

Con esta solución no sólo mejoramos el tiempo de búsqueda cuando sale una bolita, sin tener que sacrificar tiempo al construir la tabla, sino que también mejoramos la cantidad de espacio requerido para almacenar las tablas. Por ejemplo, en la ronda de las tablas amarillas se ocupa para cada tabla 14 enteros para la lista, lo que es un gasto de memoria de 14 x 4 bytes = 56 bytes, pero con la implementación de la lista de booleanos se ocupan 20 booleanos en la lista, lo que es un gasto de memoria de 20 x 1 bit = 2.5 bytes.

Finalmente, para evitar tener que recorrer toda la lista de booleanos para verificar si una tabla era ganadora o no, se decidió sacrificar un poco de memoria y mantener un contador por cada tabla que indica la cantidad de números que aún no están marcados, con lo que, para determinar si una tabla es ganadora o no, solo se comprueba si ese contador es igual a cero.

Descripción de la Solución Propuesta

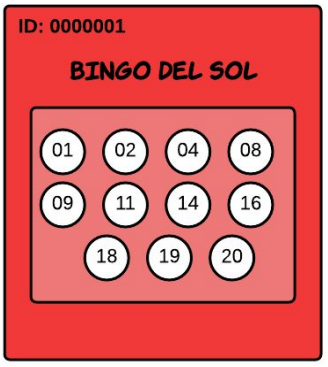
La solución propuesta para el proyecto está compuesta por los siguientes pasos:

1. Por cada ronda se lee el archivo de las tablas correspondientes al color de la ronda y se carga un arreglo de tablas similar al siguiente:

   [[0000001, [true, true, ..., false], 14],
   [0000002, [true, false, ..., false], 14],
   ...
   [0000200, [true, true, ..., true], 14]]

   Donde por cada tabla perteneciente al arreglo:
   - tabla[0] representa el id de la tabla
   - tabla [1] contiene el arreglo de booleanos correspondiente a los números de la tabla
   - tabla[2] representa la cantidad de números no marcados en la tabla

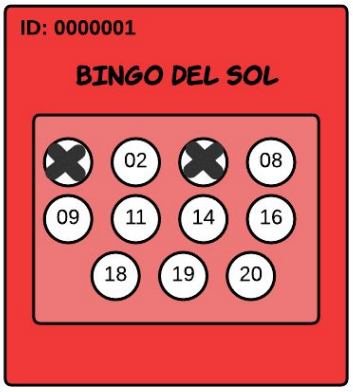
   Por ejemplo, el arreglo:
   [0000001,[true,true,false,true,false,false,false,true,true,false,true,false,false,true,false,true,false,true,true,true],11]
   Representa la siguiente tabla:



2. Por cada bolita que salga en una ronda, el programa le pide al usuario el número de la bolita que haya salido, luego se recorre todo el arreglo de tablas y se marca el número si se encuentra en la tabla.

Para marcarlo primero se verifica en el arreglo de booleanos si el índice dado por el número de la bolita es true, si lo es, cambia el valor de ese índice a false y disminuye el contador de bolitas no marcadas en la tabla.

Por ejemplo, luego de que hayan salido los números **1** y **3**, el arreglo se encontraría de la siguiente manera:
[0000001,[**false**,true,false,**false**,false,false,false,true,true,false,true,false,false,true,false,true,false,true,true,true],**9**]



3. Luego de jugar 14 bolitas, se verificará si alguna tabla del arreglo tiene cero bolitas no marcadas, si existe, presenta el id de la tabla ganadora, si no pregunta si alguien más ha ganado.

4. Si la respuesta del usuario es positiva, la ronda finaliza sin tener ninguna tabla ganadora, caso contrario el programa pregunta por una última bolita, la marca en las tablas que la contengan y verifica el contador nuevamente para concluir si alguna tabla ha ganado o no con este último número.

5. Finaliza la ronda y procede a jugarse la siguiente en el caso de que exista, si la ronda finalizada es la última ronda, entonces el juego finaliza.

Algoritmo

(Algoritmo seleccionado con su respectivo análisis)

1. Algoritmo para crear la matriz de tablas

```
1  def crear_tablas(color:str,n_bolas:int):
2    archivo = input("Ingrese el nombre del archivo de las tablas "+color+"\n")
3    with open(archivo) as tabla_file:
4      print("Creando tablas...")
5      super_tablas = []
6      for line in tabla_file:
7        booleanos = [False for i in range(20)]
8        line = line.strip().split(",")
9        tabla = [line[0],booleanos,n_bolas]
10       for i in range(1,n_bolas+1):
11         booleanos[int(line[i])-1] = True
12       super_tablas.append(tabla)
13   return super_tablas
```

El tiempo de ejecución de crear_tablas es $\Theta(n*m)$, donde n es el número de tablas que hay en un archivo y m el número de bolas de cada tabla, que depende del color de la tabla. Por cada tabla del archivo se crea un arreglo de booleanos en el que se marca cada número de la tabla, como se observa en la línea 11.

2. Algoritmo para verificar ganador

```
1  def verificar_ganador(tablas):
2    for tabla in tablas:
3      if(tabla[2]==0):
4        return tabla[0]
5    return -1
```

El tiempo de ejecución de verificar_ganador en el peor de los casos es O(n), donde n es el número de tablas, y se da en el caso en el que no exista ninguna tabla ganadora o que la tabla ganadora esté al final del arreglo, ya que la línea 3 se ejecutaría n veces.

3. Algoritmo para jugar una ronda

```
1  def ingresar_bolita(tablas, i):
2    numero_jugado = -1
3    while(numero_jugado==-1):
4      numero_jugado = input(f"Ingrese la bolita jugada {i}:  ")
5      if(numero_jugado.isnumeric()):
6        numero_jugado = int(numero_jugado)
7        if(numero_jugado<1 or numero_jugado>20):
8          print("Por favor ingrese un número entre uno y 20")
9          numero_jugado=-1
10     else:
11       print("Por favor ingrese un numero valido")
12       numero_jugado=-1
13
14   for tabla in tablas:
15     if(tabla[1][numero_jugado-1]):
16       tabla[1][numero_jugado-1] = False
17       tabla[2]-=1


18 def jugar_ronda(tablas, n_bolas):
19   for i in range(1,n_bolas+1):
20     ingresar_bolita(tablas,i)
21   idGanador = verificar_ganador(tablas)
22   if idGanador!=-1:
23     return idGanador
24
25   ganador = ""
26   while ganador.upper() !="S" and ganador.upper()!="N":
27     ganador = input("¿Ha habido un ganador? (S/N): ")
28   if ganador.upper()=="N":
29     ingresar_bolita(tablas,n_bolas+1)
30     idGanador = verificar_ganador(tablas)
31     if idGanador!=-1:
32       return idGanador
33   return -1
```

Para la función ingresar_bolita, las líneas de la 2 a la 12 son ignoradas en el análisis debido a que son validaciones. De esta manera, el tiempo de ejecución de ingresar_bolita es $\Theta(n)$, donde n es el número de tablas de la ronda, puesto que por cada tabla se verifica en la línea 15 si esa tabla tiene el número de la bolita ingresada por parámetro.

Por otro lado, en la función jugar_ronda, la línea 20 se ejecuta m veces, donde m es el número de bolitas por ese color de ronda; pero, dado que ingresar_bolita tiene un tiempo de ejecución $\Theta(n)$, el tiempo de ejecución de esa línea sería $\Theta(n*m)$. La línea 21 tiene un tiempo de ejecución en el peor de los casos de O(n), este se da cuando no se gana con las primeras m bolitas y se necesita sacar una bolita más, por lo que también se ejecutarían las líneas 29 y 30, cada una con un tiempo de ejecución de $\Theta(n)$ y O(n) respectivamente. Por lo que al final tendríamos:

$T(n,m) = \Theta(n*m) + O(n) + \Theta(n) + O(n) = \Theta(n*m)$

Por lo que el tiempo de ejecución de jugar_ronda es $\Theta(n*m)$

4. Main

```
17 mensaje = "Ronda de tablas"
18 info = (("amarillas",14),("azules",14),("rojas",11))
19 print("Bienvenidos al sistema de bingo!!\n")
20 print("No olvides crear un archivo por cada color para ingresar tus tablas!")
21
22 for inf in info:
23   color, n_bolas = inf
24   print("\n{mensaje:>30} {color}")
25   tablas = crear_tablas(color,n_bolas)
26   while tablas is None:
27     tablas = crear_tablas(color,n_bolas)
28
29   id_ganador = jugar_ronda(tablas,n_bolas)
30   if(id_ganador==-1):
31     print("No tienes ninguna tabla ganadora en esta ronda")
32   else:
33     print(f"La tabla ganadora es: {id_ganador}")
34     print("Fin del Bingo 2.0! Disfrute su día")
```

En la función principal se juegan 3 rondas, una para cada color. Para jugar cada ronda, en la línea 25 se crea la tabla para esta; esta operación tiene un tiempo de ejecución de $\Theta(n*m)$, donde n es el número de tablas de ese color y m es el número de bolitas para ese color de tabla. En la línea 29 se jugaría la ronda respectiva, cuyo tiempo de ejecución es $\Theta(n*m)$.

Para cada color tendríamos: $T(n,m) = 2\ \Theta(n*m) = \Theta(n*m)$.

Finalmente, el tiempo de ejecución total para todo el juego sería:

$$\Theta(n_1 * m_1) + \Theta(n_2 * m_2) + \Theta(n_3 * m_3)$$

Donde para 1<=i<=3:

n_i : número de tablas de la ronda i

m_i : número de bolitas que se juegan para la ronda i

Comentarios del grupo sobre el trabajo realizado aplicados al análisis de algoritmos

---

Debido al conocimiento adquirido en el curso, pudimos analizar los tiempos de ejecución de las posibles soluciones candidatas al problema, y en base a ello pudimos escoger con buen criterio el mejor algoritmo en términos de consumo de tiempo y espacio para resolver el problema.

Es importante considerar el impacto de la optimización de los algoritmos que resuelvan problemas de la vida real que puedan automatizar eficientemente tareas de los usuarios, en el caso de este proyecto, otro algoritmo menos eficiente podría ser la diferencia entre que nuestro usuario cante o no bingo antes que los demás.

Declaración de participación de los integrantes de grupo en la realización del proyecto

---

Definición del problema, V.B., J.M., A.R. y A.V.; creación del algoritmo, V.B., J.M., A.R. y A.V.; edición del documento, V.B., J.M., A.R. y A.V.; pruebas del algoritmo, V.B., J.M., A.R. y A.V.; análisis del algoritmo, V.B., J.M., A.R. y A.V.; diseño de las imágenes, V.B.

Referencias bibliográficas

No se han tenido referencias bibliográficas para este proyecto.