Trabajo Práctico – Algoritmos y Estructuras de Datos – FCEN – UBA

El programa implementa un sistema para gestionar el conteo de votos de una elección nacional, tanto para presidente como para diputados (no para los cargos parlamentarios del mercosur ni para los senadores), con soporte para múltiples distritos. Además, asigna las bancas para los diputados como lo hacemos en la realidad, según el método D'Hondt.

Se decidió particionar el código en 3 clases distintas: 'SistemaCNE.java', 'Distrito.java' y 'Heap.java'. Esto fue con la intención de lograr un mayor encapsulamiento y así poder testear más fácilmente. De hecho, se empezó programando desde la más abstracta ('Heap.java'), hasta la que se usa en los 'tests' de la cátedra ('SistemaCNE.java'). Por lo tanto, resulta conveniente dar una breve explicación de cada clase, en ese orden.

'Heap.java'. Es una implementación de un clásico 'max-heap' de Fibonacci sobre un arreglo. Se usa para gestionar la asignación de bancas. Sus principales métodos son:

- 'bajar()'. Dado un índice del arreglo, lo 'baja', es decir, se fija si es menor que alguno de sus 'hijos', y, en caso de serlo, los cambia de lugar, y sigue de forma recursiva.
- 'modificarMaximo()'. Se toma el máximo elemento del arreglo (es decir, el primero, ya que se trata de un 'max-heap'), se cambia su valor, y se lo 'baja' (para que vuelva a ser un 'max-heap' y no se rompa el invariante de representación).
- 'Heap()'. Es el método constructor de la clase. Se necesita un arreglo de tipo 'Votos' (esta clase guarda la cantidad de votos de un partido, junto con su respectivo 'id'). Sobre el mismo arreglo (que se toma por referencia para eficientizar el uso de memoria y el tiempo), se fija desde el último sub-árbol y reordena de acuerdo a la lógica 'max-heap' (el nodo padre debe ser mayor o igual a sus hijos).

Antes de explicar la clase 'Distrito.java', es necesario un entendimiento básico sobre cómo funciona el método D'Hont para la asignación de escaños a los diputados.

	/1	/2	/3	/4	/5	/6	/7	/8	/9	/10	Escaños asignados	Escaños proporcionales
Partido A	[1] 391 000	[3] 195 500	[6] 130 333	[8] 97 750	[10] 78 200	[13] 65 166	[16] 55 857	[18] 48 875	[21] 43 444	39 100	**9**	8,21
Partido B	[2] 311 000	[5] 155 500	[7] 103 666	[11] 77 750	[14] 62 200	[17] 51 833	[20] 44 428	38 875	34 555	31 100	**7**	6,53
Partido C	[4] 184 000	[9] 92 000	[15] 61 333	[19] 46 000	36 800	30 666	26 285	23 000	20 444	18 400	**4**	3,86
Partido D	[12] 73 000	36 500	24 333	18 250	14 600	12 166	10 428	9125	8111	7300	**1**	1,53
Partido E	27 000	13 500	9000	6750	5400	4500	3857	3375	3000	2700	**0**	0,57
Partido F	12 000	6000	4000	3000	2400	2000	1714	1500	1333	1200	**0**	0,25
Partido G	2000	1000	666	500	400	333	285	250	222	200	**0**	0,04

Figura I. Asignación de escaños según el método D'Hont.

Tal y como se puede observar en la Figura I, para aplicar el método D'Hont se necesita conocer la cantidad de escaños que debe asignar el distrito en cuestión (en el caso de la Figura I, se trata de 21 escaños a ser asginados). Luego, se toman los votos de cada partido y se configura una tabla de forma que cada fila representa a un partido, y cada columna representa al cociente entre la cantidad de votos totales del partido y el número de columna. Luego, si la cantidad de escaños a asginar en ese distrito fuera 'x', se toman los 'x' elementos más grandes de la tabla (en el caso del ejemplo presentado, se toman los 21 mayores). Por último, a cada partido se le asigna la cantidad de cocientes que tenga en esa lista de los 'x' mayores (en el ejemplo, al partido A se le asignan 9 escaños).

Aquí es donde radica la mayor complejidad del trabajo práctico. Para cumplir con los límites de tiempo impuestos por la cátedra, una posibilidad era implementar este método usando un 'max-heap' (configurar la tabla completa y tomar los mayores es demasiado lento e innecesario). A grandes rasgos, el 'heap' funciona de la siguiente manera:

1. Se toman los datos sobre los votos de cada partido y se los transforma para que cumplan el invariante de representación de un 'max-heap' de Fibonacci.
2. Mientras haya escaños por asignar, se toma el máximo del arreglo (es decir, el primer elemento). Se le asigna un escaño al partido en cuestión y se divide la cantidad de votos por la cantidad de bancas ya asignadas a ese partido más uno (ya que se estaría sumando una nueva banca). Por último, se llama al método 'modificarMaximo()' para que se siga tratando de un 'heap'.

'Distrito.java'. Implementa lo que vendría a considerarse como un distrito propiamente dicho en el ámbito electoral (no guarda los votos para presidente). Contiene los valores como la cantidad de bancas que dispone en la cámara de diputados, la cantidad de votos en blanco, la cantidad de votos para cada partido, etc. Sus métodos principales son:

1. 'registrarMesa_D()'. Toma los votos para diputados de cada partido y modifica los atributos privados pertinentes. Se desecha el anterior 'heap' (si lo hubiese) y se construye uno nuevo con los valores actualizados. Se 'setea' el atributo '_yaSeCalculoBancas' en 'false'. Se hace esto y no se calculan directamente la asignación de bancas por pedidos de cumplir con distintas complejidades temporales para los métodos.
2. 'calcularBancas()'. Usando los conceptos explicados anteriormente, se calcula la cantidad de bancas para cada partido y se modifica el arreglo '_bancasPartidos'.

'SistemaCNE.java'. No incluye mayor complejidad que lo explicado anteriormente. Lo más interesante resulta ser la forma en la que se identifican las mesas que corresponden a cada distrito. Como resulta en la realidad, cada mesa de votación tiene un número que la identifica y que no es compartido con ninguna otra. Además, cada distrito tiene un rango de mesas, por ejemplo, una escuela en Belgrano puede tener desde la mesa 5440 hasta la 5450. Por ende, para tener noción de qué distrito le corresponde a cada mesa, solo se necesita un arreglo que guarde el número de la última mesa que tiene el distrito. Este arreglo puede estar ordenado y así obtener el distrito de una mesa con búsqueda binaria.