**MEMORIA DE LA IMPLEMENTACIÓN DEL MONTÍCULO BINOMIAL**

**POR JAVIER GÓMEZ MORALEDA**

## INTRODUCCIÓN

Para esta práctica, he decidido utilizar en lenguaje C++, ya que me ha resultado más cómodo al haberlo utilizado en la asignatura Estructura de Datos de segundo en todas las implementaciones. Para realizar el código me he apoyado en dichas implementaciones para saber por donde empezar, y de los apuntes que nos ha proporcionado el profesor para desarrollar las distintas funciones y los casos de prueba. En este documento explicaré la implementación que he utilizado junto con una breve descripción de las funciones. Además, proporciono varios casos de prueba que pueden ser ejecutados por el profesor, y unas graficas resultantes de ejecuciones con casos de prueba mucho más grandes. Los ficheros entregados son:

- ***binomial.h***: Fichero de cabecera que contiene la implementación de la estructura y sus funciones.
- ***dcola.h***: Fichero con la implementación de una cola utilizada en el fichero anterior (proporcionada por el profesor de la asignatura Estructura de Datos).
- ***binomial.cpp***: Fichero que contiene los casos de prueba.
- ***graficas.cpp***: Fichero con el código utilizado para la generación de tiempos.

## REPRESENTACIÓN

Como sabemos, un montículo binomial es una colección de árboles binomiales de distinto grado, y en el que cada árbol es un montículo de mínimos. Además, cada árbol Bk se puede descomponer en 2 árboles de grado k-1, y los hijos de un árbol Bk son también árboles binomiales de grado k-1 a 0.

Partiendo de esta base, he definido una estructura nodo, que contendrá toda la información para poder representar un árbol binomial. Por un lado, tenemos su valor de tipo entero (para simplificar, pero podría ser de cualquier tipo), su grado, un puntero al nodo hijo más a la izquierda, un puntero al nodo padre y un puntero al hermano de la derecha más cercano. De esta manera podemos ver que un nodo no tendrá hijos si su puntero hijo es nulo, será el hijo más a la derecha si su puntero hermano es nulo y pertenecerá a la raíz principal si su puntero padre es nulo. Además, está claro que el padre sólo puede visitar directamente al primero de sus hijos. Disponemos de dos constructores, uno al que se le pasa únicamente un elemento y otro al que le podemos pasar el resto de parámetros.

Siguiendo con la representación, tenemos el montículo binomial en sí, que únicamente contendrá un puntero al inicio que apuntará a la raíz del árbol de menor grado, otro puntero al nodo que contiene al elemento mínimo y un puntero al nodo anterior a este, para realizar operaciones de borrado y consulta del mínimo mucho más sencillas. Al igual que en el nodo, disponemos de dos constructores: uno para inicializar el montículo con un único elemento que formará un árbol de grado 0, y otro con un nodo inicio que puede contener un montículo en sí.

## OPERACIONES

- ***consultaMinimoMonticulo():*** Devuelve el puntero al nodo que contiene el mínimo.
- ***unionMonticulo(MonticuloBinomial m2):*** Realiza la unión entre dos montículos. Para ello modificará el montículo con el que realizamos la llamada a la función, incluyendo todos los árboles de m2. Primero realizamos una unión de todos los árboles de forma creciente en uno sólo y luego hacemos un recorrido de los mismos uniendo todos los árboles que tienen el mismo grado. Al final, necesito actualizar los punteros *_minimo* y *_anteriorMinimo*.
- ***insercionMonticulo(int elem):*** Inserta un elemento nuevo en el montículo. Para ello crea un nuevo montículo binomial con ese elemento y llama a la operación de unión.
- ***eliminarMinimoMonticulo():*** Elimina el mínimo del montículo. Para ello, desengancha la raíz de la lista principal del montículo y recorro los hijos desenganchándolos de su padre e invirtiendo el orden. Una vez he invertido el orden, el hermano más a la derecha será el inicio del nuevo montículo, el cual uniré al principal.
- ***decrecerClaveMonticulo(Nodo* p, int k):*** Decrece una clave del montículo suponiendo que tengo un puntero a dicha clave. Si la nueva k es menor que la clave anterior, voy comparando con su padre a que, o bien llega a la raíz, o bien el padre es menor. Será necesario actualizar el mínimo si k es menor.
- ***imprimeMonticulo():*** Imprime cada árbol del montículo por niveles. Como cada padre sólo puede acceder a un único hijo, resulta imposible recorrer los hijos y luego los nietos seguidos, ya que para acceder a los hijos del hijo más a la derecha es necesario pasar por el de más a la izquierda. Para solucionar este problema, utilizo el TAD DCola que nos proporcionó el profesor de Estructura de Datos. Ahí voy guardando todos los hijos del árbol y cuando muestro uno por pantalla, añado sus hijos y así sucesivamente hasta vaciar la cola.
- ***borraMonticulo(Nodo* n):*** Libera toda la memoria eliminando todos los nodos de un montículo. Utilizada para la creación de gráficas para reducir el uso de la memoria con montículos muy grandes.

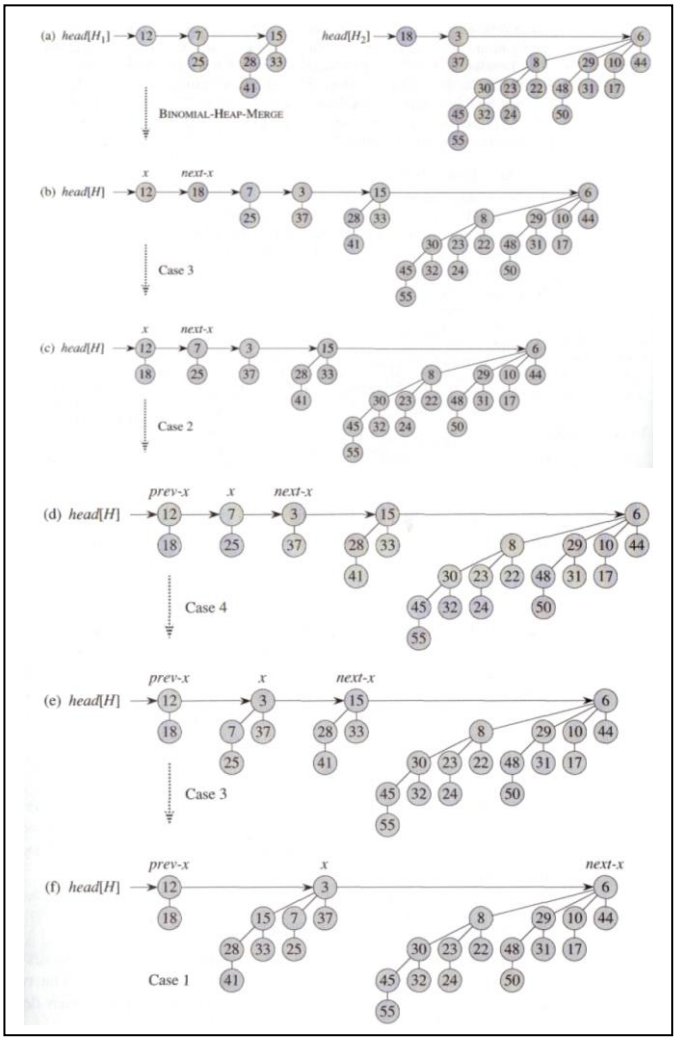
Además, existen otras funciones adicionales que funcionan de manera complementaria con estas operaciones. Más información en el código fuente.

## CASOS DE PRUEBA

Estos casos pueden ser ejecutados por el profesor ejecutando el fichero binomial.cpp, en el que dispone de un pequeño menú con las opciones. Las operaciones de inserción y mínimo no tienen su propio ejemplo ya que la primera es utilizada para construir el montículo y la segunda la utilizo en los tres ejemplos siguientes.

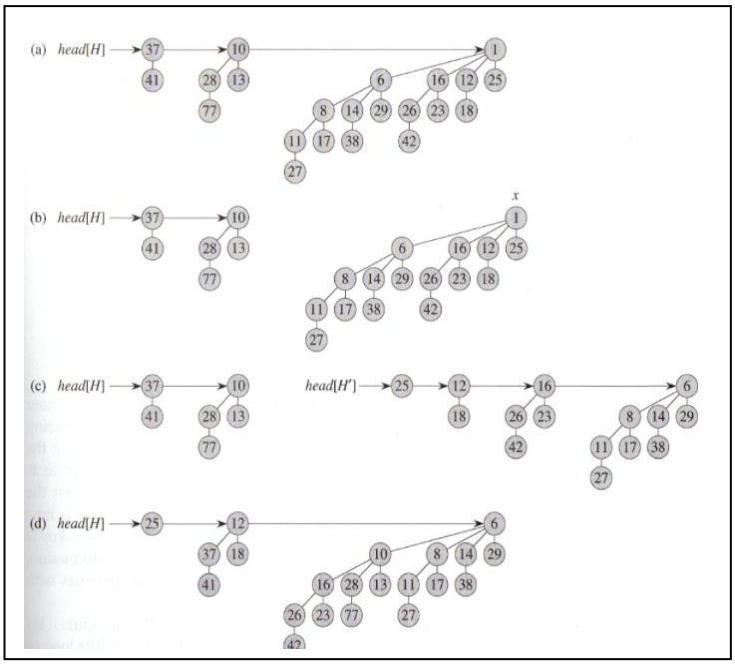
Para construir los montículos, empiezo insertando los nodos hojas, y voy avanzando insertando los hijos antes que los padres. Me ha parecido más cómodo que modificar punteros de los nodos, ya que los ejemplos tenían unos cuantos elementos.

## 1. Unión de dos montículos



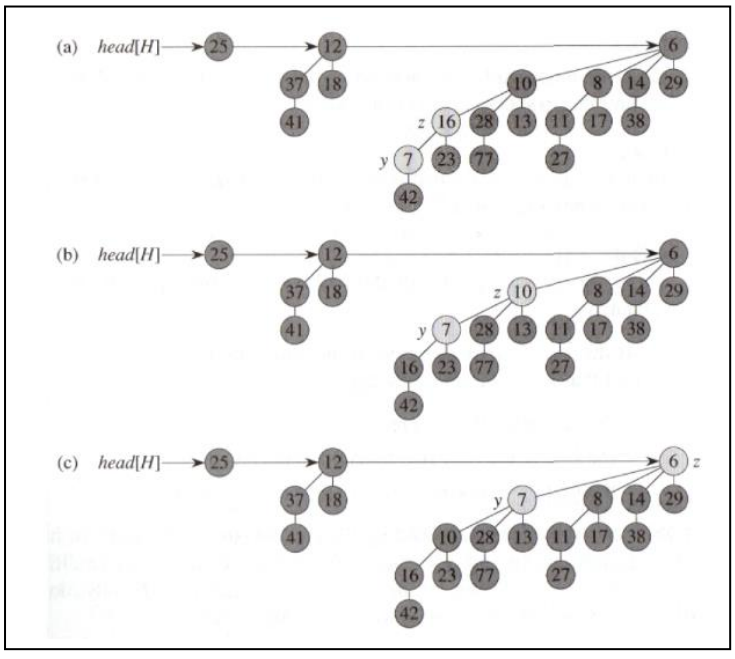

```
C:\Users\Javier\source\repos\MonticuloBinomial\Debug\MonticuloBinomial.exe
MONTICULO BINOMIAL
REALIZADO POR JAVIER GOMEZ MORALEDA (2021).

Ejemplos de ejecucion:
1. Ejemplo de la union de dos monticulos.
2. Ejemplo del borrado del minimo de un monticulo.
3. Ejemplo de decrementar un elemento del monticulo.

0. Salir.

Introduce una opcion: 1

MONTICULO 1

ARBOL DE GRADO 0:
12

ARBOL DE GRADO 1:
7 25

ARBOL DE GRADO 2:
15 28 33 41

MINIMO DEL MONTICULO 1: 7

MONTICULO 2

ARBOL DE GRADO 0:
18

ARBOL DE GRADO 1:
3 37

ARBOL DE GRADO 4:
6 8 29 10 44 30 23 22 48 31 17 45 32 24 50 55

MINIMO DEL MONTICULO 2: 3

UNION MONTICULO 1 Y 2

ARBOL DE GRADO 1:
12 18

ARBOL DE GRADO 3:
3 15 7 37 28 33 25 41

ARBOL DE GRADO 4:
6 8 29 10 44 30 23 22 48 31 17 45 32 24 50 55

MINIMO DE LA UNION: 3
```

## 2. Eliminar el mínimo




```
C:\Users\Javier\source\repos\MonticuloBinomial\Debug\MonticuloBinomial.exe
MONTICULO BINOMIAL
REALIZADO POR JAVIER GOMEZ MORALEDA (2021).

Ejemplos de ejecucion:
1. Ejemplo de la union de dos monticulos.
2. Ejemplo del borrado del minimo de un monticulo.
3. Ejemplo de decrementar un elemento del monticulo.

0. Salir.

Introduce una opcion: 2

MONTICULO

ARBOL DE GRADO 1:
37 41

ARBOL DE GRADO 2:
10 28 13 77

ARBOL DE GRADO 4:
1 6 16 12 25 8 14 29 26 23 18 11 17 38 42 27

MINIMO: 1

MONTICULO TRAS EL BORRADO

ARBOL DE GRADO 0:
25

ARBOL DE GRADO 2:
12 37 18 41

ARBOL DE GRADO 4:
6 10 8 14 29 16 28 13 11 17 38 26 23 77 27 42

MINIMO TRAS EL BORRADO: 6
```

## 3. Decrementar un elemento

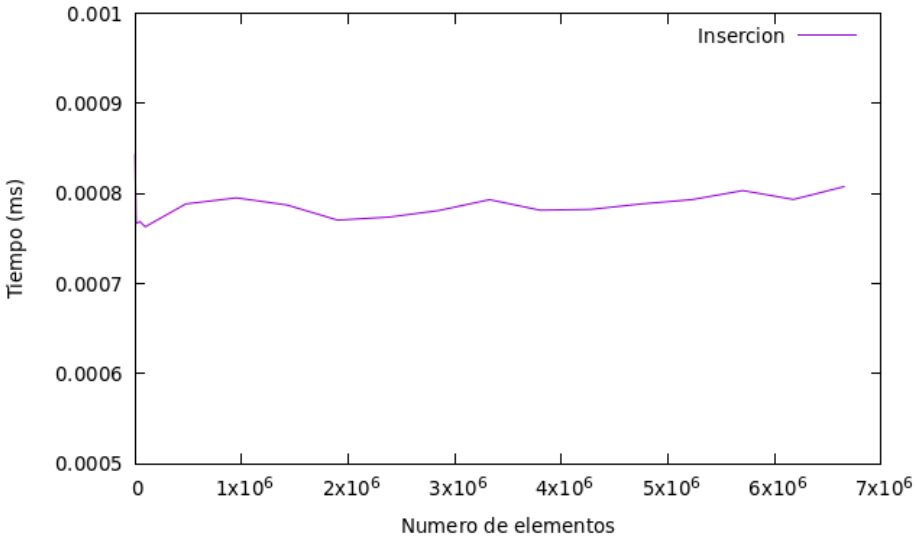


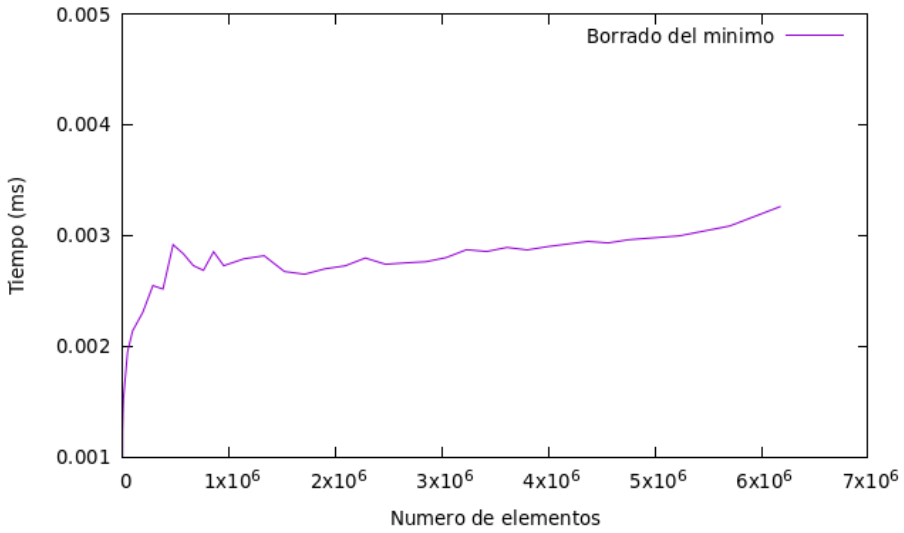

```
C:\Users\Javier\source\repos\MonticuloBinomial\Debug\MonticuloBinomial.exe
MONTICULO BINOMIAL
REALIZADO POR JAVIER GOMEZ MORALEDA (2021).

Ejemplos de ejecucion:
1. Ejemplo de la union de dos monticulos.
2. Ejemplo del borrado del minimo de un monticulo.
3. Ejemplo de decrementar un elemento del monticulo.

0. Salir.

Introduce una opcion: 3

MONTICULO

ARBOL DE GRADO 0:
25

ARBOL DE GRADO 2:
12 37 18 41

ARBOL DE GRADO 4:
6 10 8 14 29 16 28 13 11 17 38 26 23 77 27 42

MINIMO: 6

MONTICULO TRAS DECREMENTAR LA CLAVE 26

ARBOL DE GRADO 0:
25

ARBOL DE GRADO 2:
12 37 18 41

ARBOL DE GRADO 4:
6 7 8 14 29 10 28 13 11 17 38 16 23 77 27 42

MINIMO TRAS DECREMENTAR: 6
```

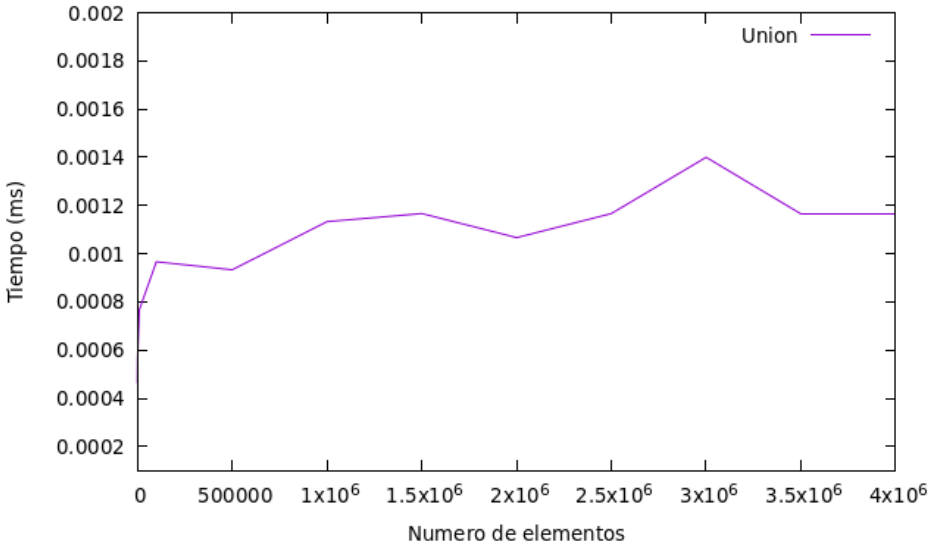
## GRÁFICAS DE TIEMPOS

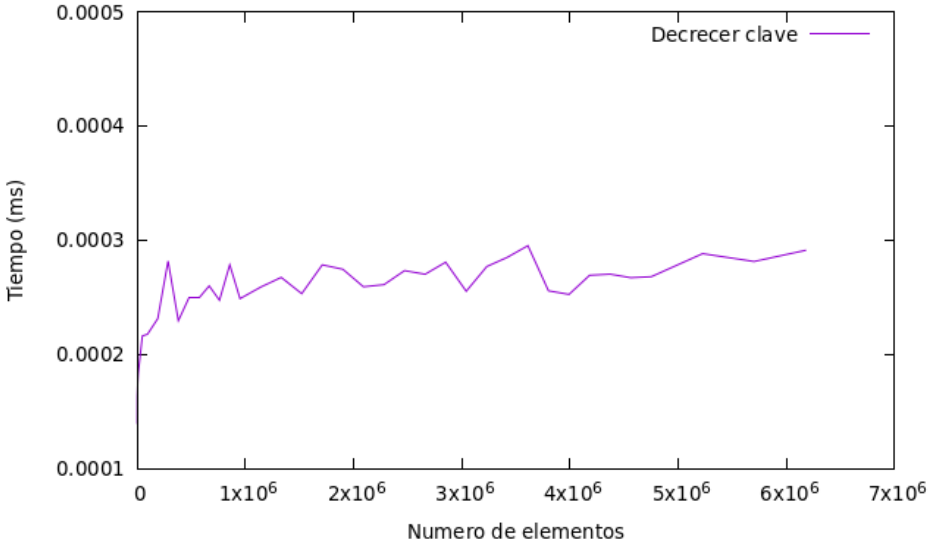
Para terminar esta memoria, se han generado unas gráficas para probar la eficiencia de la implementación de las operaciones de inserción, borrado del mínimo, eliminación del mínimo y decrecer una clave. La operación para consultar el mínimo se ha obviado puesto que consiste en devolver un puntero almacenado directamente en la estructura de forma inmediata. Para cumplir nuestro propósito, se han generado las gráficas con la librería GnuPlot partiendo de unos ficheros de texto realizados a mano a partir de las distintas mediciones. Esto es debido a que, al intentar automatizar el proceso, la memoria llegaba a llenarse y de esta otra forma he podido medir la eficiencia de estructuras mucho mayores.

En cuanto a la obtención de los tiempos, debido a que eran medidas muy pequeñas, para calcular el tiempo de una operación en una estructura de tamaño n, se han realizado un 10% de llamadas respecto al tamaño de la estructura para luego dividir el tiempo resultante entre el número de llamadas. Por ejemplo, si mi estructura tiene tamaño 1000, he llamado 100 veces a una operación, y el tiempo obtenido se divide entre 100. Además, esto se repite 3 veces y se calcula la media para asegurarnos de que es un valor correcto. Todo este proceso se realiza en todas las operaciones salvo en la unión, puesto que no se puede unir varias veces una estructura.






En todos los ficheros de tiempos salvo en la unión, la primera columna representa el tamaño de la estructura, la segunda el tiempo medido y la tercera el resultado real de una única llamada a la operación. Es cierto que, para algunos tamaños pequeños, el tiempo medido es menor a 20ms, pero me ha parecido conveniente añadirlo para ver como comienza la gráfica.

## CONCLUSIÓN FINAL

En las gráficas podemos observar que el tiempo de las operaciones aún teniendo estructuras de datos gigantescas, es realmente pequeño. Por ejemplo, en la inserción se mantiene prácticamente constante. En otras como el borrado del mínimo o decrecer una clave, se ve una tendencia logarítmica. La unión es la que consta con menos puntos debido a que no he podido realizar varias llamadas consecutivas. Aún así, habiendo medido varias veces, nunca he obtenido resultados muy dispares.

Toda esta memoria es fruto de horas y horas de trabajo en la creación del código, casos de prueba y gráficas. Considero que he aprendido bastante bien cómo funciona la estructura y a la realización de medidas de tiempo correctamente. Cualquier aportación o mejora de este trabajo será bien recibida.