Montículo de Fibonacci

Manuel Sánchez Pérez

Notas sobre la implementación

La siguiente práctica consiste en la implementación de un montículo de Fibonacci como una plantilla C++ `FibHeap<T,C,A>`, siendo `T` el tipo de los elementos del montículo, `C` el tipo comparador utilizado (`std::less` por defecto), y `A` el allocator utilizado por la estructura de datos (`std::allocator` por defecto).

Dicha estructura forma parte del proyecto `edalib`, una propuesta que estoy manteniendo junto con Manuel Freire para mejorar las estructuras de datos utilizadas en EDA y asignaturas similares.

Dependencias

La práctica hace uso de C++11, y ha sido probada en diferentes compiladores. Además hace uso de alguna biblioteca externa para test unitarios y timing. *Todas ellas incluidas en la entrega, ésta incluye un* `Makefile` *generado por CMake listo para compilar en linux con GCC. Si quiere probar la entrega en algún otro entorno, como Windows y Visual Studio, contacte conmigo y le pasaré el proyecto VS generado.*

El código utiliza biicode, un gestor de dependencias para C y C++. Esto no afecta para nada a la entrega, biicode utiliza CMake y como mencioné más arriba la entrega contiene el Makefile generado por CMake (Carpeta `build/` en la raíz de la entrega). De todos modos biicode es muy fácil de instalar, en Debian y derivados:

```
wget http://apt.biicode.com/install.sh && chmod +x install.sh && ./install.sh
```

El código de la práctica, `FibHeap.hpp` se encuentra en la carpeta `blocks/manu343726/edalib/`, al igual que el archivo con los tests unitarios `test.cpp`.

Estilo del código

He procurado que la implementación sea lo más cercana posible a la guía, haciendo uso de funciones auxiliares para aplicar acciones sobre conjuntos de nodos.
Así donde en la guía dice "*Por cada nodo de la cadena de x haz...*" he podido escribir código del estilo `do_forwards(x, [](node* n){ ... });`, donde `do_forwards()` es una función que ejecuta la función dada sobre todos los nodos de la cadena del nodo especificado, y `[](node* n){}` es una expresión lambda representando la acción a ejecutar en cada nodo. Existen funciones equivalentes para toda la jerarquía de un nodo (`do_foreach()`) y la línea de descendencia de un nodo (`do_downwards()`).

Aserciones y estado interno de la estructura de datos

También he intentado mantener unos chequeos exaustivos sobre el estado interno de la estructura de datos, verificando que esta cumple todas las precondiciones y postcondiciones requeridas en cada momento. La mayoría de dichas aserciones se encuentran en funciones auxiliares del estilo `_check_integrity_XXX()` y están pensadas para que no tengan ningún impacto en el rendimiento en compilaciones no debug.

Casos de prueba. Rendimiento.

Como parte de los test unitarios que incluye edalib (Vea `test.cpp`, el único `.cpp` del proyecto), he incluido una nueva batería de pruebas para `FibHeap`.

Dichas pruebas consisten en la inserción consecutiva de n elementos seguida de su posterior eliminación, comprobando que la estructura mantiene como mínimo el elemento correcto:

```
describe("Testing Fibheap", []()
{
    describe("Testing FibHeap<int,std::allocator>", []()
    {
        testFibHeap<int, 1000>();
    });
});
```

Puede cambiar el segundo parámetro de la plantilla de función `testFibHeap()` para especificar el tamaño de la prueba.

He probado tres casos de prueba, 1000,10000,10000000 elementos, con las optimizaciones del compilador disponibles al máximo.
Aunque el tiempo de ejecución de `FibHeap::extract_min()` es del orden de decenas de microsegundos, tan cortos que me ha sido imposible hacer una buena comparativa y gráfica llamada por llamada, el crecimiento que he observado se ajusta a la teoría (Salvo pequeños picos que achaco a algún posible bug interno muy escurridizo). Por lo que he observado, si el tiempo de ejecución promedio de `extract_min()` con `n = 1000` es de 1 microsegundo, con `n = 10000` es más o menos 10-20 microsegundos, y 80-120 microsegundos con `n = 1000000`.
Esto se refleja en el tiempo completo de ejecución, pasando de milisegundos, a segundos, y minutos en el último caso.

Es muy posible, volviendo al "bug escurridizo", que mi implementación contenga algún error. Agradecería feedback por su parte tras la corrección e incluso dedicar una tutoría a repasar mi implementación.