Estructuras de dator y álgoritmos II
Tarea 1

Rubén Pérez Palacios Lic. Computación Matemática
Profesor: Dr. Carlos Segura González

31 de enero de 2022

Para implementar los métodos requeridos, se implemento un *templateclass* para poder hacer uso de polimorfismo y se acepten cualquier tipo de clase para la clave, que implemente un orden lexicográfico con el operador $<$; y cualquier clase para la prioridad que implemente un orden con los operadores $<$ e $=$. La forma de implementar la clase del heap avanzado además de la implementación usual del heap haremos uso de:

- Un map para poder almacenar la posición en que se encuentra cada clave y así poder actualizar estas.
- Función heapify down: Asegura que al cambiar el valor de la prioridad del nodo *act* el subárbol del nodo *act* sea un heap. Para ello verifica si la prioridad de *act* es mayor a las de sus hijos, de ser así entonces el subárbol de *act* puesto que por construcción ese subárbol ya es un heap, en caso contrario intercambiamos *act* por el hijo con mayor prioridad, ya que nos asegura que la raíz de este subárbol es mayor a la de sus dos hijos y por construcción al resto del subárbol, hacemos de nuevo lo mismo recursivamente para el hijo mayor. Esto eventualmente va decidir que el subárbol de *act* ya fue un heap o es un hoja pero por definición una hoja es un treap.
- Función heapify up: Asegura que al cambiar el valor de la prioridad del nodo *act* el conjunto de nodos del camino de *act* hacía la raíz cumplan la propiedad del heap. Checa si la prioridad del padre de *act* es mayor a la de este, en caso de ser así por construcción se cumple que el conjunto de nodos del camino de *act* hacía la raíz cumplan la propiedad del heap; de no ser así entonces intercambíamos el padre de *act* por el, por construcción el padre de *act* cumple con la propiedad del heap puesto que $p(act) > p(padre(act))$ entonces el padre de *act* con el valor de *act* también lo cumpliria, hacemos esto recursivamente con el padre de *act*. Esto eventualmente va decidir que el subárbol del padre de *act* ya fue un heap o es la raíz pero para poder ser así entonces por construcción *act* es el de mayor prioridad y por definición es un heap.

Cuando nosotros actulizamos la prioridad del nodo *act* entonces los únicos nodos comprometidos a no cumplir la prioridad del heap son todos los nodos del subárbol de *act* y el conjunto de nodos en el camino de *act* a la raíz, puesto que $heapifyup$ y $heapifydown$ nos aseguran que todos estos nodos se reordenan de manera en que todos cumplan con la propiedad del heap, concluimos que todos nuestro árbol después de actualizar *act* es un heap.

El resto de las funciones son implementadas como en un heap normal.