Estructura de datos treap

Gustavo Gutiérres-Sabogal

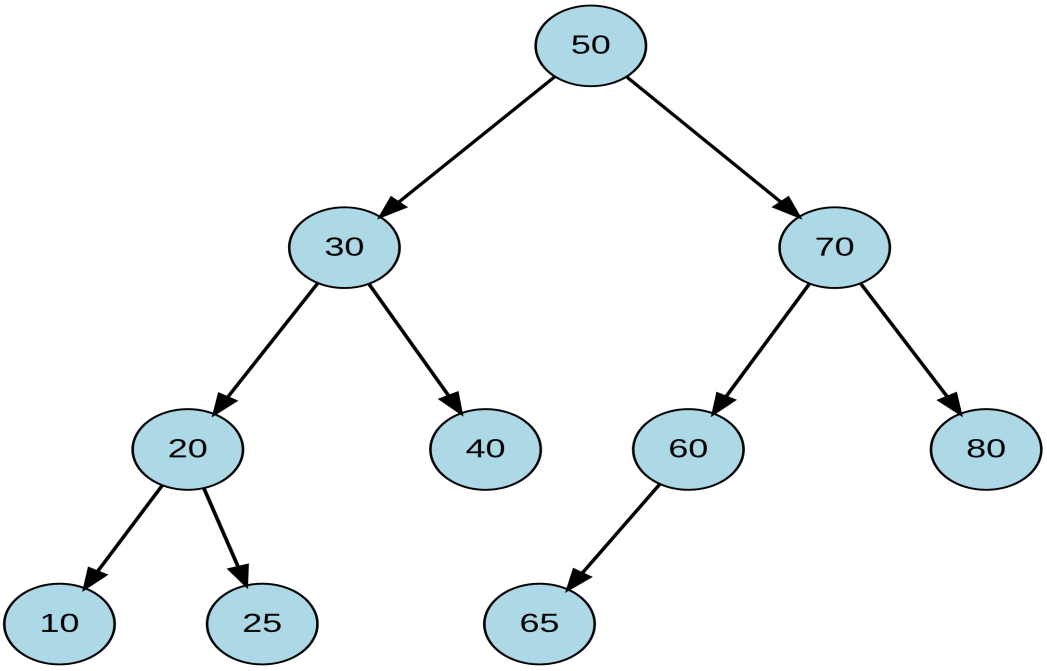
Introducción

Un treap [1] es una estructura de datos que combina las propiedades de un árbol binario de búsqueda (BST) y un heap. El nombre "treap" proviene de la fusión de las palabras "tree" (árbol) y "heap" (montículo).

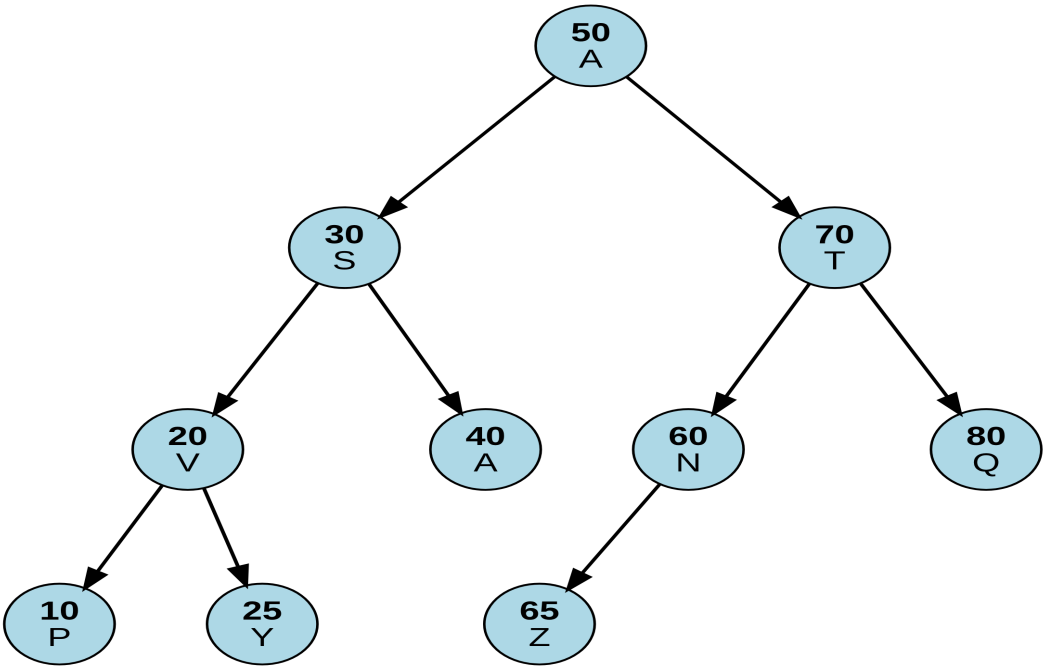
El treap resuelve un problema importante de los árboles binarios de búsqueda tradicionales: la degeneración. En un BST normal, si insertamos elementos en orden, el árbol se convierte en una lista enlazada, degradando las operaciones de $O(\log n)$ a $O(n)$.

Al asignar prioridades aleatorias a cada nodo, el treap se mantiene balanceado con **alta probabilidad**, garantizando operaciones de búsqueda, inserción y eliminación en tiempo esperado $O(\log n)$.

Los treaps son una extensión probabilística de los árboles binarios de búsqueda. Como tal, implementan los mismos conceptos matemáticos: conjunto, función. Cuando un BST implementa el concepto de conjunto, en sus nodos solo se almacena un dato: el elemento del conjunto.



Si en lugar del concepto de conjunto se quisiera representar una relación de asociación el árbol tendría dos datos por cada nivel. Uno de esos datos se denomina la llave y el otro es al valor asociado a esa llave. En la siguiente figura la llave es elemento que se presenta en la parte superior de cada nodo y el valor asociado se presenta debajo. Note que pueden haber valores repetidos: nodo con llave 50 y nodo con llave 40 ambos tienen un valor asociado de A. Sin embargo las llaves deben ser únicas en el árbol.



Lo anterior tiene como consecuencia que puedan existir diferentes estructuras de datos que internamente usan árboles binarios de búsqueda. Por ejemplo, en el caso de la libraría estándar de C++ los tipos `sety map` usan internamente variantes de BSTs.

```
#include <map>
#include <set>

using namespace std;

int main() {
  // Declares an association from strings to strings.
  map<string, string> a;
  // Declares a set of strings
  set<string> b;
}
```

Treap

El objetivo de la estructura de datos treap es solucionar el problema clásico que tienen los árboles BST: cuando se realiza la inserción de elementos ordenados el árbol pierde su balance. La consecuencia de lo anterior es que las operaciones de búsqueda, inserción y borrado de elementos incrementan su tiempo de ejecución.

Existen diferentes estructuras de datos que tratan de solucionar el mismo problema. Por ejemplo, los árboles rojo-negro y los árboles AVL. Sin embargo la solución que cada una de estas estructuras le da al problema es difícil de implementar.

La solución de los treaps es elegante y simple: unir dos estructuras de datos, un BST y un heap. Formalmente, un treap T es un árbol binario donde cada nodo v contiene un par ordenado (k_v, p_v) tal que:

1. Propiedad del BST: para todo nodo v:
   - Para cada nodo u en los nodos del sub-árbol *izquierdo* de v se cumple: $u_k < v_k$.
   - Para cada nodo u en los nodos del sub-árbol *derecho* de v se cumple: $v_k < u_k$.
2. Propiedad del MaxHeap: para todo nodo v con padre u: $p_v \leq p_u$

La definición anterior es para treaps que implementan la noción de conjunto. Cada k_v es uno de los elementos del conjunto. Para el caso de la noción de asociación cada nodo v del treap tiene la forma (k_v, v_v, p_v). El elemento adicional v_v es el valor asociado a la llave k_v. Las propiedades siguen siendo las mismas y este valor no interviene en ellas.

Ejercicios

1. Realice una revisión bibliográfica de la estructura de datos treap. Para ello puede comenzar (pero no limitarse) con [1] y [2]. Es importante que realice una lectura crítica de los documentos. Es libre (como siempre) de utilizar herramientas de inteligencia artificial para su investigación. Tenga eso si en cuenta que la forma de evaluación será una sustentación donde dichas herramientas no estarán presentes.
2. Implemente la estructura de datos `TreapSet` que implementa el concepto de conjunto. La clase debe ser genérica en el tipo de los elementos del conjunto y debe proveer por lo menos las siguientes operaciones.
   - Constructor de un conjunto vacío.
   - Destructor
   - `size`: retorna la cardinalidad del conjunto.
   - `insert`: adiciona un elemento al conjunto.
   - `remove`: retira un elemento del conjunto.
   - `member`: recibe un elemento y retorna si éste hace parte del conjunto.
3. Implemente la estructura de datos `TreapMap` que implementa el concepto de asociación. La clase debe ser genérica en los tipos de datos `Key` y `Value`. El primero es el tipo de dato de la llave y el segundo el de los valores asociados a la llave. La estructura debe contar al menos con las siguientes operaciones:
   - Constructor de un mapa vacío.
   - Destructor

- `insert`: recibe una llave y su respectivo dato y los inserta dentro de la estructura.
- `remove`: recibe una llave y elimina de la estructura de datos la llave y su elemento asociado.
- `find`: recibe una llave y retorna el valor asociado a ella o un valor por defecto si no se encuentra.

Bibliografía

[1] C. R. Aragon and R. Seidel, "Randomized search trees," *Algorithmica*, vol. 16, no. 4–5, pp. 464–497, 1996, doi: 10.1007/BF01940876.

[2] R. Seidel and C. R. Aragon, "Randomized search trees," University of California, Berkeley, UCB/CSD-96-915, 1996.