Maps

Map is a general term for data structures that stores a collection of key values and associated data values, and actually implies nothing specific about the physical structure that is used, or about the interface.

However, some specific connotations have rubbed off from the implementations in the C++ Standard Template Library and the Java library.

Key values must be unique, but different key values may be associated with the same data value, so data values are not necessarily unique.

Most commonly, the underlying physical structure is chosen to provide $\Theta(\log N)$ average search cost, so some sort of balanced binary tree or a hash table is usual.

The Java Map Interface

The major elements of the Java Map interface:

```
boolean containsKey(key)
```

does the map contain this key?

```
boolean containsValue(value)
```

does the map contain this value?

```
Object get(key)
```

return value to which this key maps

```
Object put(key, value)
```

map this key to this value

```
Object remove(key)
```

remove the mapping from this key

Implementations of the Java Map Interface

`HashMap`:

- uses a hash table for the underlying physical structure (more on this later…)
- under ideal conditions, `get()` is Θ(1) on average

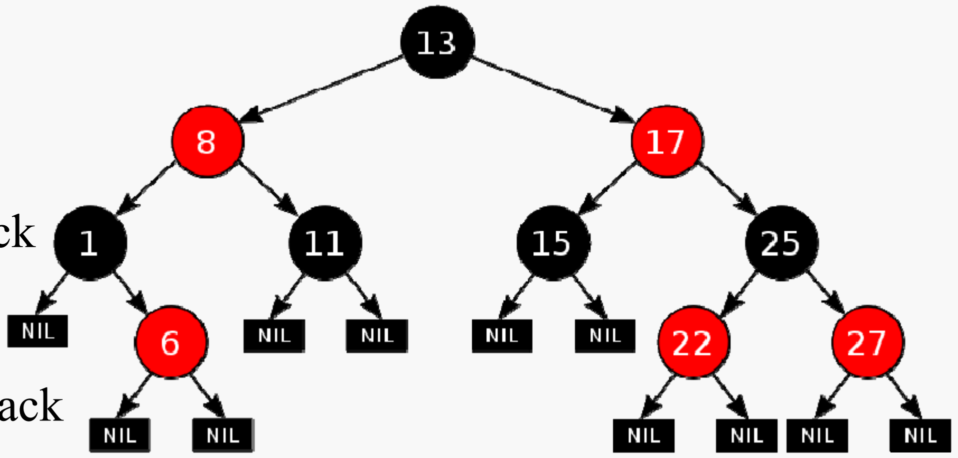
`TreeMap`:

- uses a red-black tree for the underlying physical structure
- under all conditions, `get()` is Θ(log N) in worst case

Red-Black Trees

Requirements:

- has the BST property
- every node is colored either red or black
- the root is black
- if a node is red, its children must be black
- every path from the root to a `null` reference must contain the same number of black nodes



Implication: the height of a red-black tree with N nodes is at most 2log(N+1).

Therefore, search cost is guaranteed to be no worse than $\Theta(\log N)$.

Implementation is similar to, but somewhat simpler than AVL.

Rebalancing cost is somewhat less than for AVL.

Height bound is somewhat worse than for AVL.