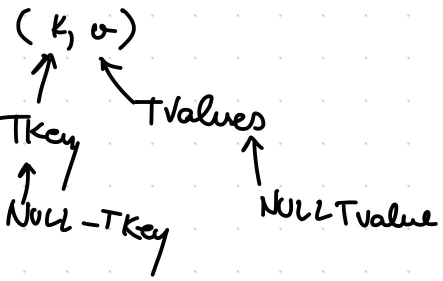
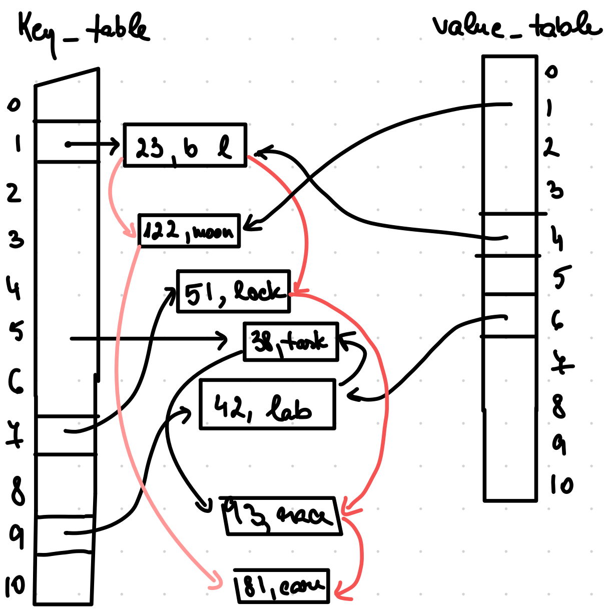
Implement a bidirectional Map (over a hash)

init
search (k) ⇒ v
reverseSearch(v) → $\Theta(1)$ on avarage
insert (k, v) → $\Theta(1)$ on avarage and amortized(?)
remove (k) ⇒ v → $\Theta(1)$ on avarage





BDMap

m: Integer
key_table: (↑BDMNode)[]
value_table: (↑BDMNode)[]
h_key : TFunction
h_value : TFunction

BDMNode
k : TKey
v : TValue
next_key: ↑BDMNode
prev_key: ↑BDMNode
next_value : ↑BDMNode

```
Function findKey (bdm, k)
    pos ← bdm.h_key(k)
    pNode ← bdm.key_table[pos]
    while (pNode ≠ NIL) AND ([pNode].k ≠ k) execute
            pNode ← [pNode].next_key
    endwhile
    findkey ← pNode
End-function
```

```
            returns a value
Function remove (bdm, k)
    node ← find Key( bdm, k)
    if (node = NIL) execute
        val ← NULTValue
    else
        val ← [node]. v
        remove_node( bdm, node)
    end-if
    remove ← val
end_function
```

```
Subalg    remove_ node (bdm, pNode)
    pos ← bdm . h_ key ([pNode]. k)
    if ([pNode].prev !=NIL) AND ([pNode].next_key ≠ NiL)   then
            pprev ← [pNode]. prev-key
            pnext ← [pNode]. next - key

            [pprev]. next_key ← [pNode]. next_ key
            [pnext] .prev_key ← [pNode]. prev_ key
    else if ([pNode].prev_key = NiL) then
            if ([pNode]. next_ key = NiL) then
                    bdm . keytable [pos] = NUL key
            else
                bdm. key_ table [pos] ← [pNode]. next_ key
            endif
    else
            [[pNode].prev _key]. next_ key=NiL
```

* same for h_table

Sorted Map

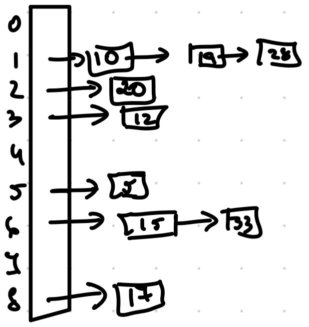
    m : integer
    T : (↑Node) []
    k : TFunction
    rel : Relation

Node

* merge Lists

    min Heap

k	h(k)
5	5
28	1
19	1
15	6
20	2
33	6
12	3
17	8
10	1



```
Node
        k : TKey
        next : ↑Node

        next_sorted : ↑Node
        prev_sorted : ↑Node
```