□哈希表：空间换时间.

哈希函数的设计："键"的分布越均匀越好

(一)整型

小范围整数直接使用

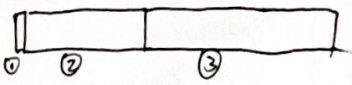
小范围负整数进行偏移 ~100~100 → 0~200

大整数：取模，ex：身份证号：54722198020796612

计 mod 10^6 → 096612 → 分布不均匀

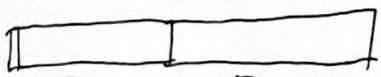
⇒解决办法：模一个素数.

浮点数：32bit (float)



① sign
② exponent (8-bit)
③ fraction (23-bit)

64bit (double)



① sign
② exponent (11-bit)
③ fraction (52-bit)

直接当整型处理

String

$code = c*26^3 + o*26^2 + d*26^1 + e*26^0$

$= c\cdot B^3 + o\cdot B^2 + d\cdot B^1 + e\cdot B^0$

$hash(code) = (c\cdot B^3 + o\cdot B^2 + d\cdot B^1 + e\cdot B)\ \%\ M$

$= [((c\cdot B + o)\cdot B + d)\cdot B + e]\ \%\ M$

$= [(((c\%M)\cdot B + o)\%M\cdot B + d)\%M\cdot B + e]\%M$

⇒ int hash = 0

```
for(int i=0; i<s.length(); i++)
    hash = hash * B + s.charAt(i)) % M
```

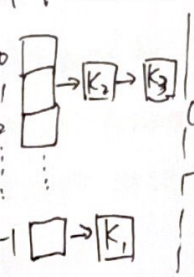
复合类型 Date: year, month, day

hash(date) = ((date.year % M)·B + date.month) % M + date.dy) % M.

```
int a=42; (Integer)a).hashCode() → 42
int b=-42; ((Integer)a).hashCode() → -42
```

double ⇒ (Double).hashCode()

□哈希冲突的处理方法：Seperate Chaining



(hashCode(K1) & (0x7fffffff)) % M
(使 hashCode 为正).

HashMap 就是一个 TreeMap 数组

HashSet 就是一个 TreeSet 数组

当哈希冲突达到一定程度，每一个位置从链表 → 红黑树 (TreeMap)

链表：O(N/M) 红黑树：O(log(N/M))

平均每个地址承载的元素多过一定程度，扩容.

N/M >= upperTol

平均每个地址承载的元素少过一定程度，缩容

~~N/M <= lowerTol~~

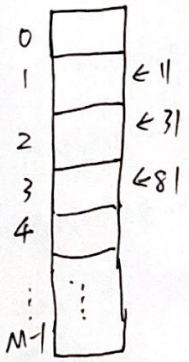
N/M < lowerTol

M → 2*M 不是素数在table中取素数

resize平均复杂度为 O(1)

哈希冲突更多的处理方法：

开放地址： 计：hash(x) = x % 10.



1) 线性探测，遇到冲突找下一个空的索引.

2) 平方探测，+1, +4, +9, +16.

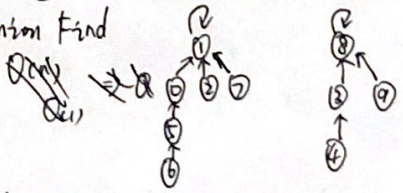
3) 二次哈希，用第二个 hash2(key).

4) rehashing，用其他的.

并查集 Union Find

union(p,q)

isConnected(p,q)



都是O(h)复杂度.

Union是把P的root指向q的root.

isConnected(p,q)判断P,q的root是否相等.

基于rank的优化：union的时候：选择深度小的树指向深度高的.

基于size的优化：node少的指向node多的.

路径压缩：Path-compression：parent[p] = parent[parent[p]] (在find过程中)