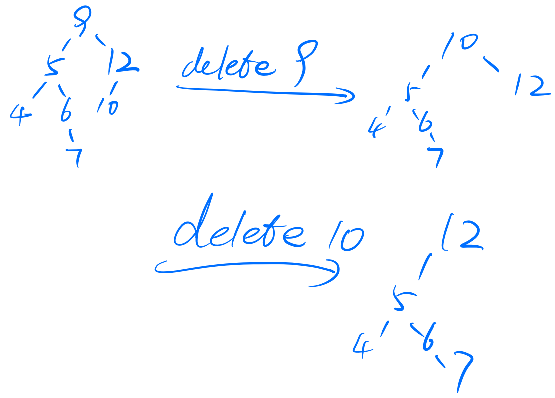
1.

eg.



Solution 1 is the same to AVL tree. After every deletion, check whether any subtrees are unbalance. If yes, just use rotation.

Solution 2. Choose successor or predecessor with equal probability to take the position of the deleted internal node (But it doesn't work sometimes)

Solution 3. Use Red-Black-Tree. The root and leaves (NIL) are black. If a node is red, then its children are black. Every node reaches leaves which passes the same number of black nodes. Any deletion needs to meet the rules above.

2.
```
percolateDown (hole, &heap): // heap[1,2,…n]
    tmp = heap[hole]
    while 2×hole ≤ currentSize : // currentSize = heap.currentSize
        child = 2×hole
        if child != currentSize and heap[child+1] < heap[child]:
            child ++
        if heap[child] < tmp
            heap[hole] = heap[child]
        else:
            break
        hole = child
    heap[hole] = tmp

ToMinHeap(& heap):
    for i = ⌊n/2⌋ to 1
        percolateDown(i, heap)
```

Time complexity is $O(n)$

Assume the depth of the heap is d

$d = \Theta(\lg n)$

For i th level element, percolateDown has at most $d-i$ times

ith level has at most 2^{i-1} elements.

$$T(n) = \Theta(\sum_{i=1}^{d}(d-i)\cdot 2^{i-1}) = \Theta(\sum_{i=1}^{d} d\cdot 2^{i-1} - \sum_{i=1}^{d} i\cdot 2^{i-1})$$

$$= \Theta[d(2^d-1) - (d\cdot 2^d - 2^d + 1)]$$

$$= \Theta[2^d - d - 1]$$

$$= \Theta(2^d)$$

$$\because d = \Theta(\lg n)$$

$$\therefore T(n) = \Theta(2^{\lg n}) = \Theta(n)$$

3. Yes, It can reduce conflicts and save memory

Assume there exists a fixed divisor d for x and p.

Let $x = n\cdot d$ $p = m\cdot d$ $m, n \in \mathbb{Z}$

If $\ell = x \bmod p$, namely $\ell = x - kp = n\cdot d - k\cdot m\cdot d = (n-km)d$ $k \in \mathbb{Z}$

then $d|\ell$.

only xd ($x\in\mathbb{Z}$) is possible to be occupied by value.

Other positions can't be used. It's a waste of memory. And increase the probability of conflicts.

So better to let p be prime.

The scenario: x is always multiple of fixed integer which divide p.

eg. In the exam whose content is all multiple choices, better to use prime.

4. linear probe

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
34	0	45			1000	6	23	7			28	12	29	11	30	33

Double hashing

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
29	0				34	6	7		1000	23	28	12	11	30	45	33

I prefer double hashing. Because the numbers will be distributed more uniformly. And there will be less collisions. while linear probe is easy to make numbers together which degrades query efficiency

5. Solution 1: we want to delete number x, need to find it and make the slot empty

To find it, need to use hash function. If no collision and x is in the table, just make slot h(x) empty.

If collisions exist (the key $\neq x$), then find "the next one" until find it or traverse every slot. (If the slot is empty, find "the next one".)

① In linear probe, id of "the next one" is (hash_id + 1) mod tableSize

② In double hashing, id of "the next one" is (hash_id + h2(key)) mod tableSize.

Solution 2: If we delete number X, just find it and mark it "deleted" which means the slot can be put new value but not empty for finding. So during finding, if the key $\neq X$, then finding "the next one" until find it or find empty slot. Finding empty slot means x isn't in the table.