Week2

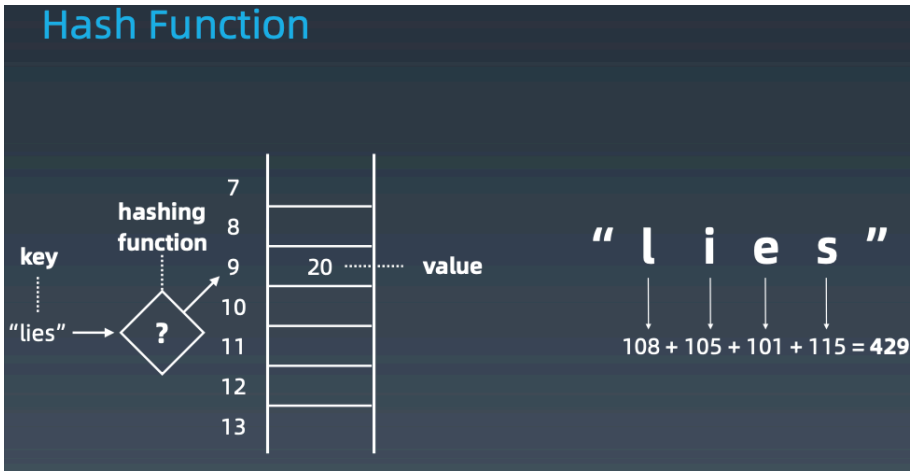
1) hash table

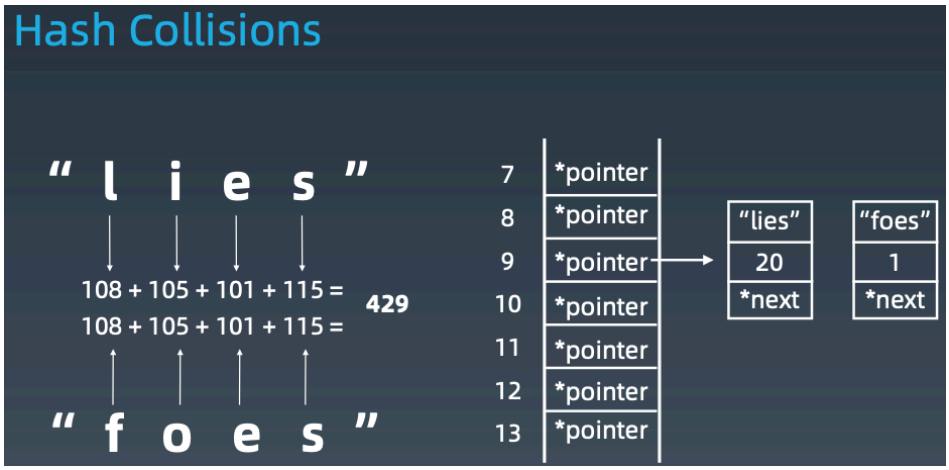
-根据关键码值（key value）而直接进行访问的数据结构

-通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度

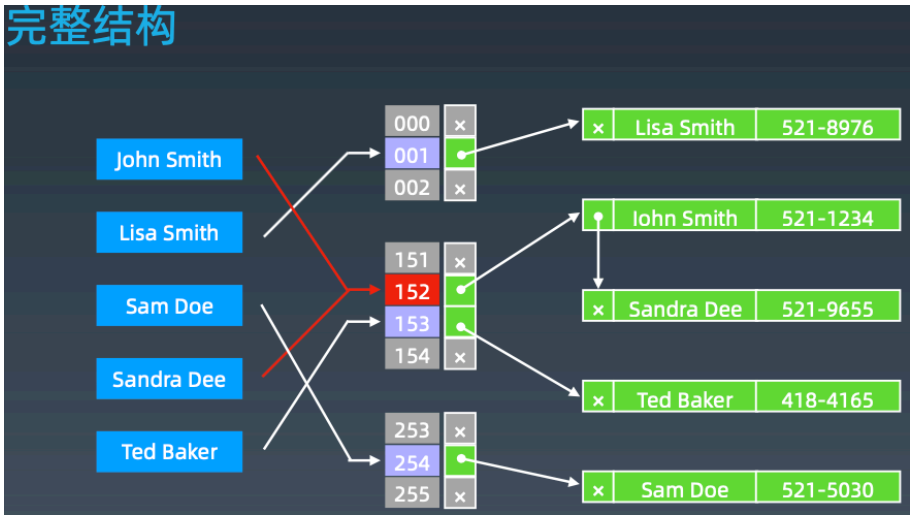
-映射函数叫 hash function 散列函数，存放记录的数组叫做哈希表

-类比：电话号码表，用户信息表





-一种 collision 解决方案：将键值相同的元素用链表连接

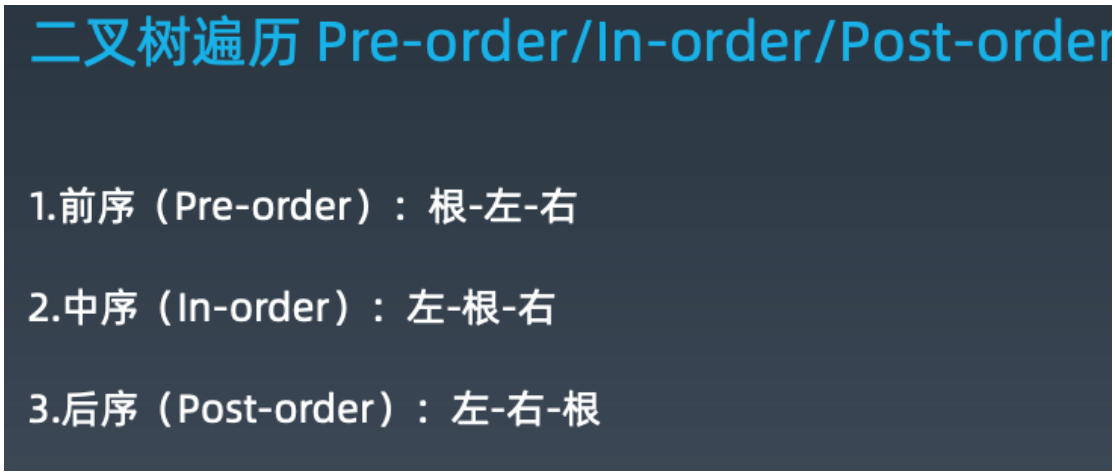


-作业：看懂 hashmap 的 get，put，写个总结

2）回顾做题四件套：clarification，possible solutions，coding，test cases

3）tree，binary tree，bst

-二叉树的遍历：



-二叉搜索树 **查询 插入**：logn 复杂度

-二叉搜索树 **删除：**找到第一个大于该节点的数来替换要删除的节点（右子树最小值）

-二叉搜索树 **查找** worst case：当树从根节点开始只有右子树，此时树变成了链表，查找为 o（n）

-思考：二叉树解法为何都是递归的？

4）堆 heap，二叉堆 binary heap

-time：

find-max: o(1)

delete-max: o(logn)

insert(create): o(logn) or o(1) (fibo heap)

-二叉堆只是堆的一种实现，比较简单，但 time 复杂度不是最好，还有很多其他更好的实现，比如 fibonacci 堆

Here are time complexities[5] of various heap data structures. Function names assume a min-heap.

Operation	find-min	delete-min	insert	decrease-key	meld
Binary[5]	$\Theta(1)$	$\Theta(\log n)$	$O(\log n)$	$O(\log n)$	$\Theta(n)$
Leftist	$\Theta(1)$	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$	$\Theta(\log n)$
Binomial[5][6]	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)$[a]	$\Theta(\log n)$	$O(\log n)$[b]
Fibonacci[5][7]	$\Theta(1)$	$O(\log n)$[a]	$\Theta(1)$	$\Theta(1)$[a]	$\Theta(1)$
Pairing[8]	$\Theta(1)$	$O(\log n)$[a]	$\Theta(1)$	$o(\log n)$[a][c]	$\Theta(1)$
Brodal[11][d]	$\Theta(1)$	$O(\log n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Rank-pairing[13]	$\Theta(1)$	$O(\log n)$[a]	$\Theta(1)$	$\Theta(1)$[a]	$\Theta(1)$
Strict Fibonacci[14]	$\Theta(1)$	$O(\log n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
2-3 heap[15]	$O(\log n)$	$O(\log n)$[a]	$O(\log n)$[a]	$\Theta(1)$	?

-二叉堆（大顶）性质：

1、通过完全二叉树来实现

2、树中任意节点的值总是大于等于其子节点的值

-二叉堆实现细节：

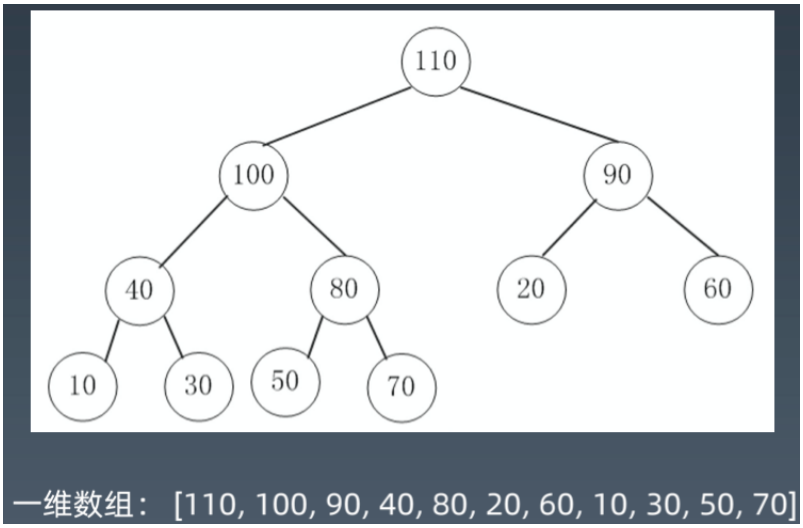
1、通过**数组**实现

2、数组中各索引 i 所对应的节点：

-索引 i 的左孩子：2*i + 1

-索引 i 的右孩子：2*i + 2

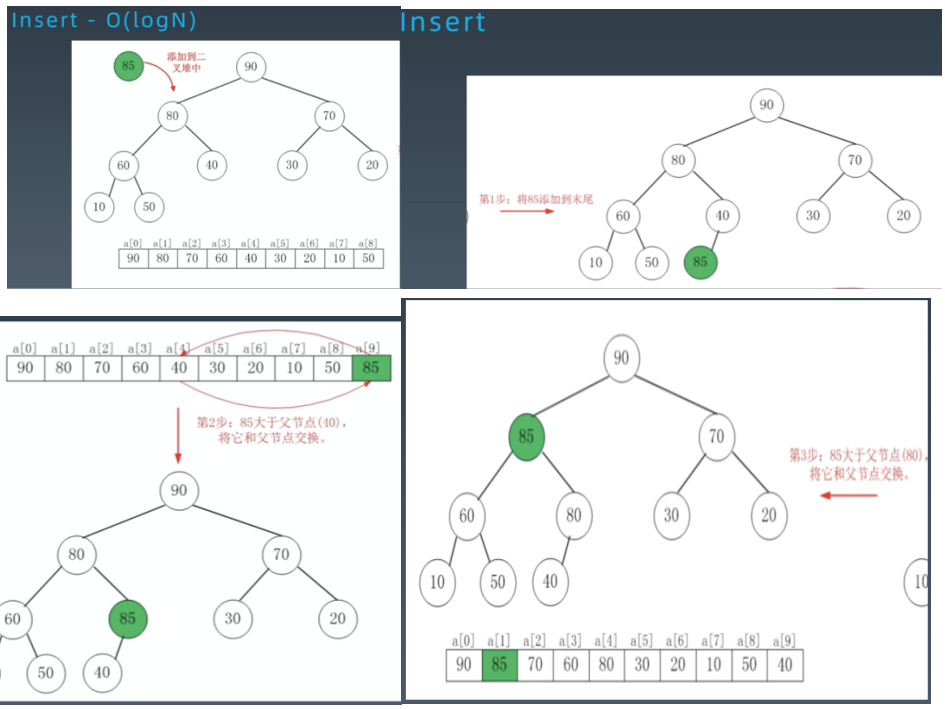
-索引 i 的父节点是 floor（(i-1）/2）



3、二叉堆 insert 插入操作

-新元素一律先插入到堆的底部

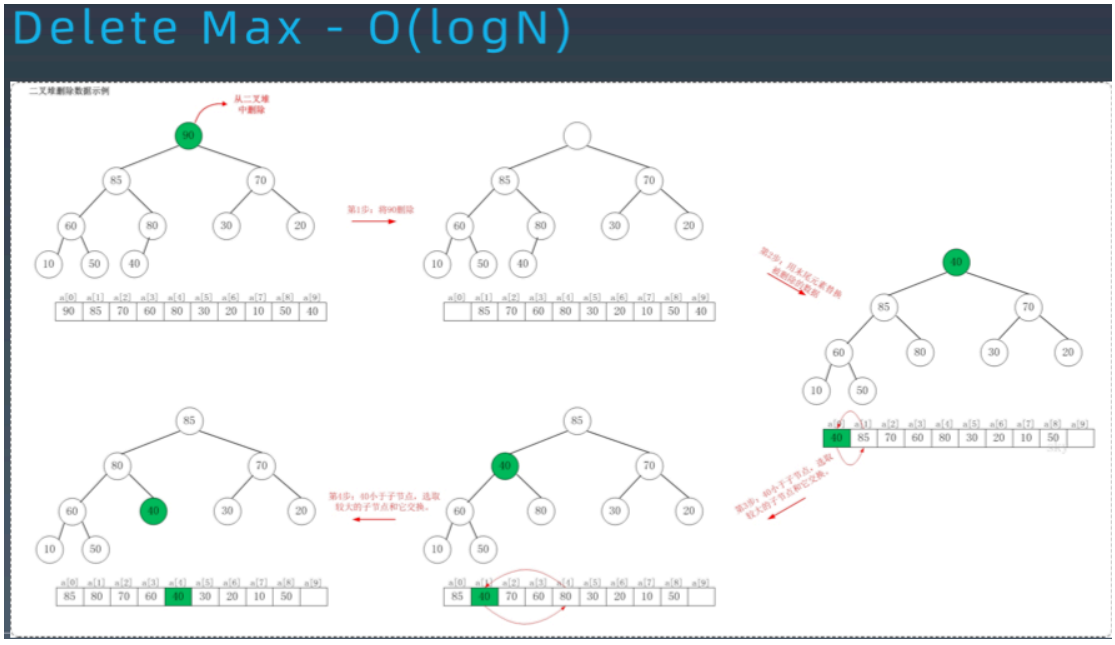
-依次向上调整整个堆的结构（一直到根即可）**heapify-up**



4、delete max 删除堆顶操作

-**为了保证堆一直保持一个*完全*二叉树的形态，**此时需要将**堆尾元素**替换到顶部（等价于堆顶元素被替代删除掉）

-**依次从顶部向下调整整个堆的结构（一直到堆尾即可）heapify-down**



5、堆的实现代码：

https://shimo.im/docs/Lw86vJzOGOMpWZz2/read

6、heap sort

https://www.geeksforgeeks.org/heap-sort/

5）图：

-表示：adjacency matrix；adjacency list；

-常用 bfs，dfs：

重点：使用 visited = set（）记录已经遍历过的点

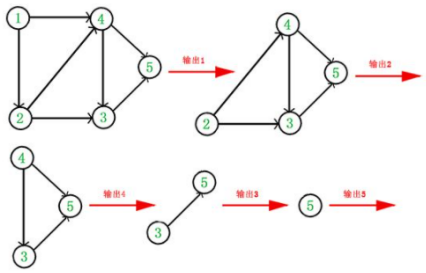
```
def BFS(graph, start, end):

    queue = []
    queue.append([start])

    visited = set() # 和数中的BFS的最大区别

    while queue:
        node = queue.pop()
        visited.add(node)

        process(node)
        nodes = generate_related_nodes(node)
        queue.push(nodes)
```

```
def dfs(node, visited):
    if node in visited: # terminator
        # already visited
        return

    visited.add(node)

    # process current node here.
    ...
    for next_node in node.children():
        if not next_node in visited:
            dfs(next_node, visited)
```

-拓扑排序：

https://zhuanlan.zhihu.com/p/34871092

> 它是一个 DAG 图，那么如何写出它的拓扑排序呢？这里说一种比较常用的方法：
> 1、从 DAG 图中选择一个 没有前驱（即入度为0）的顶点并输出。
> 2、从图中删除该顶点和所有以它为起点的有向边。
> 3、重复 1 和 2 直到当前的 DAG 图为空或**当前图中不存在无前驱的顶点为止**。后一种情况说明有向图中必然存在环。



> 于是，得到拓扑排序后的结果是 { 1, 2, 4, 3, 5 }。
> 通常，一个有向无环图可以有一个或多个拓扑排序序列。

参考链接：拓扑排序（Topological Sorting）

伪代码：

```
TOPOLOGICAL-SORTING-GREEDY(g)
  let inDegree be every verties inDegree Array
  let stack be new Stack
  let result be new Array
  for v equal to every vertex in g
    if inDegree[v] == 0
      stack.push(v)
  end
  while stack.empty() == false
    vertex v = stack.top()
    stack.pop()
    result.append(v)
    for i equal to every vertex adjacent to v
      inDegree[i] = inDegree[i] - 1
      if inDegree[i] == 0
        stack.push(i)
    end
  end
  return result.reverse()
```

时间复杂度:O(V+E)，V表示顶点的个数，E表示边的个数。

- Dijkstra:
https://www.bilibili.com/video/av25829980?from=search&seid=13391343514095937158

-minimum spanning tree:
https://www.bilibili.com/video/av84820276?from=search&seid=17476598104352152051