# 八、Hash table

insert、search、remove：平均 O(1)，最差 O(n)

h(key) = c(t(key))，t：哈希函数，c转为小于buckets大小的数

h：buckets中位置（不考虑冲突）

冲突处理：

① separate chaining（非 open chaining，其它都是open...）

每个bucket是一个linked list，

平均复杂度都是 O(N/M)（M bucket数量）

C++默认实现方式

② linear probing

每次+1直到找到，即 h，h+1，h+2，...

cluster 连续都被占用的

> Tiancheng Jiao
> EECS 281
> tcjiao

③ quadratic probing

h(k)，h(k)+1，h(k)+4，h(k)+9，...

要 mod M

④ double hashing

new index = (t(key) + j · t'(key)) mod M，j = 1, 2, 3, ...

t'(key) = q - (t(key) mod q)（q prime，q < M）

注意 t' 是个固定值，一开始就计算好，每次都增加这个固定值

Dynamic hashing：当α达到一定比例时增加尺寸

代码：

```
unordered_map<int,int> a;
a[1]=2; //插入 ， if(a[2]== ...) //访问，若不存在会插入默认值
```

```
auto it = a.find(6);
if(it != a.end()) //找到
it->first; // key , it->second; // value
```
first, second 后面没有括号！

a.erase(3); // 删除

```
unordered_set<int> b;
b.insert(4); //插入   b.erase(3); //删除
if ( b.find(5) == b.end()) //没找到
```

## 2、Tree, Graph

Tree: no cycles, all notes connected in one part

Simple tree: undirected ; Rooted tree: one root

Root, parent, child

Sibling: with same parent , ancestor: parent of parent, descendent: child of child

internal node: has children , external (leaf node): no children

height: 描述整个tree的, depth: 描述1个node(距离 root 距离) (depth(root) =0)

Ordered tree: linear ordering for the children of each node (depth (null ptr) = 0)

Complete binary tree : array表示，前面都有，后面全空

array index : root 1, left 2*i, right 2*i+1

| | insert(best) | insert(worst) | remove(worst) | parent | child | best space | worst space |
|---|---|---|---|---|---|---|---|
| array | O(1) | O(n) | O(n) | O(1) | O(1) | O(n) | O(2^n) |
| pointer | O(1) | O(n) | O(n) | O(n) | O(1) | O(n) | O(n) |

Preorder : 1、node, 2、left, 3、right

Inorder: 1、left, 2、node, 3、right

Postorder: 1、left, 2、right, 3、node

Binary search tree: left < root ≤ right

avg O(log n), worst O(n)

level order: 类似于广度优先

visit nodes in order of increasing depth in tree.

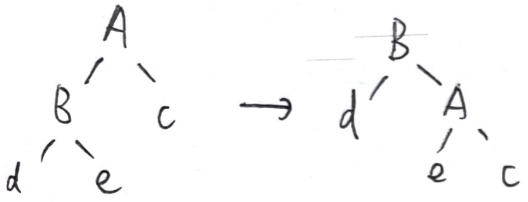
In order successor: inorder 访问的下一个

AVL: |height(left) - height(right)| ≤ 1 (for any node)

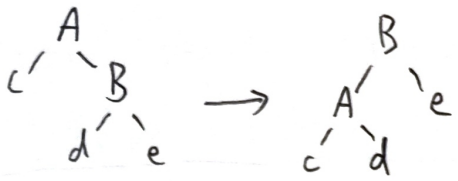
worst case search/insert/remove: O(log n) sort: n log n

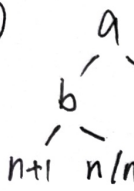
bal(n) = hight(left) - height(right)

Rotate right:

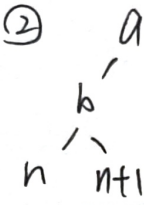



Rotate left:

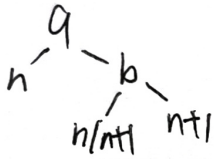



① 

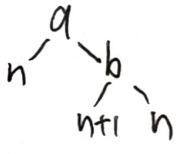
, right

② 

b left
then total right

③ 

, left

④ 

b right
then total left

---

Simple graph: no parallel edges, no self-loop

directed: edges one-way (numeric_limits<double>::infinity())

Complete: $|E| = \frac{|V| \cdot |V-1|}{2}$, dense: 快满的, sparse: E较少的

Adjacency list: 存每个点跟所有相邻的点 / Distance matrix

find edge: O(E/V) distance: O(1) (worst, best, avg)

Space: O(V+E)

distance: worst O(V), best O(1), avg O(1+E/V)

MST: Prim's: 先随便找1个点放进去, 然后每次找离在里面的点最近的点放进去

Table: $O(V^2)$

Heap: 用pq, 先次有更新的话放进pq, O(E log E), 对sparse好

Kruskal: 一直选最短的边, 只要不形成cycle就行, O(E log E)

Dijkstra (最短路径) $O(V^2)$ (遍历表), Heap (用pq存点) O(E log E)

每次松弛新点后, 选更新点里面距离起始位置最短的, 不会走回头路

---

enum: enum Test{A,B,C}, Test x=A; if(x==B)

enum class Test2{A,B,C}, Test2 x=Test2::A, if(x==Test2::B)

priority_queue<int, vector<int>, greater<int>>;

3_Algorithm

① Brute-force: 枚举所有(不判断), $2^n$或n!, 在每种情况里面再判断

② Greedy: 每次拿当前情况的最优解, 不回退, Dijkstra算greedy

③ Divide and conquer: 把问题分成多部分, quick sort (merge sort是 combine and conquer) (bottom up)

④ Dynamic programming: 储存前面的结果

⑤ Backtracing: Brute-force加约束条件, 不符合条件的不添加进去

promising() 是否符合约束条件 solution()是否到达终点

一般作 in-place sway

```
void check_node (node v)
    if (solution (v)) {write solution}
    else for each node u adjacent to v
        if (promising(u)) check_node (u)
```

⑥ Branch and Bound: 记录当前最优解, cost超过就放弃

lowerbound(): 当前的cost + 未来的最小量

upperbound(): 当前最优解

promising(): lowerbound < upperbound

```
void check_node (Node v, Best currBest)
    if(promising(v, currBest))
        if(solution(v))
            update (currBest)
        else for each child u of v
            checknode (u, currBest)
    return currBest
```