1. 编程题，LeetCode的SummaryRange

给一个排序的Integer的array， [1,2,3,5,6,8,9,12]

输出一个string: “1->3,5->6,8->9,12″

Followup，如果输入有deplicate numbers，怎么办?

[1,2,2,3] ==>”1->3″

[1,2,2,5] ==>”1-> 2 ,5″

```
class Solution {
public:
    vector<string> summaryRanges(vector<int>& nums) {
        vector<string> ans;
        if (nums.size() == 0) return ans;
        int begin = -1, end = -1;
        for (int i = 0; i < nums.size(); ++i) {
            if (begin == -1) begin = i, end = i;
            if (i < nums.size() - 1 && (nums[i] == nums[i + 1]) || nums[i] + 1 == nums[i + 1]) {
                end = i + 1;
                continue;
            }
            if (nums[begin] == nums[end]) ans.push_back(to_string(nums[begin]));
            else {
                string tmp = to_string(nums[begin]) + "->" + to_string(nums[end]);
                ans.push_back(tmp);
            }

            begin = -1;
        }
        return ans;
    }
};
```

没有排序呢?

用set可以做到每次插入是o(logn) 获取interval是o(m)

还有一种unordered_map + unordered_set的做法，

插入是O(1) 获取是o(mlogm)

Set做法:

```
set<pair<int, int> > s;
    void addNum(int val) {
        auto it = s.lower_bound(make_pair(val, val));
        if (it != s.begin()) {
```

```
            --it;
            if (it->second + 1 < val) it++;
        }
        int start = val, end = val;
        while (it != s.end() && val + 1 >= it->first) {
            start = min(start, it->first);
            end = max(end, it->second);
            it = s.erase(it);
        }
        s.insert(make_pair(start, end));
    }

    vector<Interval> getIntervals() {
        vector<Interval> ans;
        for (auto &v: s) {
            ans.push_back(Interval(v.first, v.second));
        }
        return ans;
    }
```

unordered_map+unordered_set:
用set存所有出现过的数字，出现过直接过滤，这个很重要
然后对于没出现过的值x,去看左右，即x-1和x+1是否为某个区间端点
区间存在map中，对于一个区间[left, right], map 中存了 map[left] = right, map[right] = left
这样就可以保证插入数字是可以在任意端点扩展区间

```
unordered_set<int> s;
    unordered_map<int, int> mp;
    void addNum(int val) {
        if (s.find(val) != s.end()) return;

        int left = val, right = val;
        if (s.find(val - 1) != s.end()) {
            left = mp[val - 1];
            mp.erase(val - 1);
        }
        if (s.find(val + 1) != s.end()) {
            right = mp[val + 1];
            mp.erase(val + 1);
        }
        s.insert(val);
        mp[left] = right;
```

```
        mp[right] = left;
    }
    static bool cmp (Interval & x, Interval & y) {
        return x.start < y.start;
    }
    vector<Interval> getIntervals() {
        vector<Interval> ans;
        for (auto &v: mp) {
            if (v.first <= v.second)
                ans.push_back(Interval(v.first, v.second));
        }
        sort(ans.begin(), ans.end(), cmp);
        return ans;
    }
```

2. code .

excel Sheet Column Number

```
A -> 1
B -> 2
C -> 3
...
Z -> 26
AA -> 27
AB -> 28
...
```

观察，注意对于一个n位 excel列

假设从高位到低位分别为X1,X2...Xn

则转换为数字为

(X1 - 'A') * 26^(n-1) + (X2-'A') * 26^(n-2) ....+ (Xn-1 -'A') * 26 + (Xn-'A' + 1) +
26^(n-1) + 26^(n-2) + ... + 26

即为

(X1-'A'+1) * 26^(n-1) + (x2 -'A' + 1) * 26 ^(n-2) .. + (Xn-1 -'A' + 1) * 26 + (Xn-'A' + 1)

故等价于以下代码

```
class Solution {
public:
    int titleToNumber(string s) {
        int ans = 0;
        for (int i = 0; i < s.size(); ++i) {
            ans = ans * 26 + (s[i] - 'A' + 1);
        }
        return ans;
    }
```

```
};
```

excel sheet column title（从0开始 0->A)

```
1 -> A
2 -> B
3 -> C
...
26 -> Z
27 -> AA
28 -> AB
```

根据上述推导

n = (X1-'A'+1) * 26^(n-1) + (x2 -'A' + 1) * 26 ^(n-2) .. + (Xn-1 -'A' + 1) * 26 + (Xn-'A' + 1)

逆向推的话，首先算Xn

由于Xn-'A' +1可能为26，故先 n-- 然后mod ，即可得到Xn

然后n除以26，继续n--，然后mod

依此类推

```
class Solution {
public:
    string convertToTitle(int n) {
        string ans = "";
        while (n) {
            n--;
            ans = (char)(n % 26 + 'A') + ans;
            n /= 26;
        }
        return ans;
    }
};
```

3. reverseString和reverseHTML 空间复杂度要对

reverseString比较简单，

直接做reverseString with Html entity

reverse两遍即可

```
void rev(string &s, int begin, int end) {
    int l = end - begin + 1;
    for (int i = 0; i < l / 2; ++i) {
        swap(s[i + begin], s[l - 1 - i + begin]);
    }
```

```
}
void reverse_html(string &s) {
    rev(s, 0, s.size() - 1);
    int begin = -1;
    for (int i = 0; i < s.size(); ++i) {
        if (s[i] == ';') begin = i;
        if (s[i] == '&' && begin != -1) {
            rev(s, begin, i);
            begin = -1;
        }
    }
}
```

有点麻烦的有

1. https://leetcode.com/problems/reverse-words-in-a-string/description/

这里O(n)内存的还是比较好做的

```
void reverseWords(string &s) {
        vector<string> vec;
        string tmp = "";
        for (int i = 0; i < s.size(); ++i) {
            if (s[i] != ' ') {
                tmp += s[i];
            }
            if (s[i] == ' ' || i == s.size() - 1) {
                if (tmp != "") {
                    vec.push_back(tmp);
                    tmp = "";
                }
            }
        }
        reverse(vec.begin(), vec.end());
        s = "";
        for (int i = 0; i < vec.size(); ++i) {
            if (i) s = s + " ";
            s += vec[i];
        }
    }
```

O(1)内存怎么做呢

```
void reverseWords(string &s) {
        if (s.size() <= 0) return;
        int begin = -1, end = -1;
        int j = 0;
```

```
        for (int i = 0; i < s.size(); ++i) {
            if (s[i] != ' ') {
                if (begin == -1) begin = i;
                end = i;
            }
            if (s[i] == ' ' || i == s.size() - 1) {
                if (begin != -1) {
                    int now = j;
                    for (int k = begin; k <= end; ++k) s[j++] = s[k];
                    reverse(s.begin() + now, s.begin() + j);
                    if (j < s.size()) s[j++] = ' ';
                    begin = -1;
                }
            }
        }
        if (j && s[j - 1] == ' ') j--;
        s.resize(j);
        reverse(s.begin(), s.end());
    }
```

如果加入html entity的话

将以上的reverse全部用之前的reverse html的函数替代即可

4. 二叉树 转成 数组 节约空间,

参考：考虑类似heap的构造方法

如果是dense的BT tree

用一个数组 假设父亲节点index为i, 则左儿子 i * 2 + 1 右儿子 i * 2 + 2

这个思路一定要先讲

如果是sparse的Bt tree， 则类似Leetcode的297,转成字符串

```
string serialize(TreeNode* root) {
        string ans = "";
        queue<TreeNode*> q;
        if (root) q.push(root);
        while (!q.empty()) {
            TreeNode* node = q.front();
            q.pop();
            if (node) {
                ans += to_string(node->val) + " ";
                q.push(node->left);
                q.push(node->right);
```

```
        } else {
            ans += "# ";
        }
    }
    return ans;
}

// Decodes your encoded data to tree.
TreeNode *getnext(const string & data, int & cur) {
    if (cur >= data.size()) return NULL;
    string tmp = "";
    while (cur < data.size() && data[cur] != ' ') {
        tmp += data[cur];
        cur++;
    }
    cur++;
    if (tmp == "#") return NULL;
    TreeNode *node = new TreeNode(atoi(tmp.c_str()));
    return node;
}
TreeNode* deserialize(string data) {
    if(data.empty()) return NULL;
    int cur = 0;
    TreeNode *root = getnext(data, cur);
    TreeNode *ans = root;
    queue<TreeNode*> q;
    if (root) q.push(root);
    while (!q.empty()) {
        TreeNode * node = q.front();
        q.pop();

        if (node) {
            node->left = getnext(data, cur);
            node->right = getnext(data, cur);
            q.push(node->left);
            q.push(node->right);
        }
    }
    return ans;
}
```

然后优化就是压缩了，根据值域的分布去搞，假设都在int32内，

如果大的数字比较多, 则用1个bit存是否为null节点，接下来32个bit存节点value
如果小的数字比较多，则用1个bit存是否为null节点，接下来5个字节表示value占用几个bit, 然后再接下来若干个bit存value
简单实现第一种。

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Codec {
public:

    // Encodes a tree to a single string.
    void setnextbit(string & s, int & cur, int val) {

        int b = cur / 8;
        if (s.size() <= b) s += (char)0;
        if (val) s[b] |= (1 << (cur % 8));
        cur++;
    }
    void getnextbit(const string & s, int & cur, int &val) {
        int b = cur / 8;
        if (s[b] & (1 << (cur % 8))) val = 1;
        else val = 0;
        cur ++;
    }
    string comp(vector<string> &vec) {
        string ans = "";
        int cur = 0;
        for (int i = 0; i < vec.size(); ++i) {
            if (vec[i] == "#") {
                setnextbit(ans, cur, 1);
            } else {
                setnextbit(ans, cur, 0);
                int val = atoi(vec[i].c_str());
                for (int j = 0; j < 32; ++j) {
                    if (val & (1 << j)) setnextbit(ans, cur, 1);
                    else setnextbit(ans, cur, 0);
```

```
            }
        }
    }
    return ans;
}
string serialize(TreeNode* root) {
    string ans = "";
    queue<TreeNode*> q;
    if (root) q.push(root);
    vector<string> vec;
    while (!q.empty()) {
        TreeNode* node = q.front();
        q.pop();
        if (node) {
            vec.push_back(to_string(node->val) );
            q.push(node->left);
            q.push(node->right);
        } else {
            vec.push_back("#");
        }
    }
    return comp(vec);
}

// Decodes your encoded data to tree.
TreeNode *getnext(const string & data, int & cur) {
    if (cur / 8 >= data.size()) return NULL;
    int val;
    getnextbit(data, cur, val);
    if (val == 1) return NULL;

    int node_val = 0;
    for (int j = 0; j < 32; ++j) {
        getnextbit(data, cur, val);
        if (val ) node_val |= (1 << j);
    }
    TreeNode *node = new TreeNode(node_val );
    return node;
}
TreeNode* deserialize(string data) {
    if(data.empty()) return NULL;
    int cur = 0;
    TreeNode *root = getnext(data, cur);
```

```
        TreeNode *ans = root;
        queue<TreeNode*> q;
        if (root) q.push(root);
        while (!q.empty()) {
            TreeNode * node = q.front();
            q.pop();

            if (node) {
                node->left = getnext(data, cur);
                node->right = getnext(data, cur);
                q.push(node->left);
                q.push(node->right);
            }
        }
        return ans;
    }
};

// Your Codec object will be instantiated and called as such:
// Codec codec;
// codec.deserialize(codec.serialize(root));
```

更加进一步的做法：

在上一步的基础上做一次gzip压缩。霍夫曼编码，不过不一定能压多少

https://www.cnblogs.com/kuang17/p/7193124.html

5. 一个链表 每个node有一个array， 让你实现 查找 插入

参考: 这个没啥太复杂的吧，主要就是list的添加/删除节点啥的

就是unrolled linklist

Given a LinkedList, every node contains a array. Every element of the array is char
implement two functions
1. get(int index) find the char at the index
2. insert(char ch, int index) insert the char to the index

```
#define TOT_ARRAY_SIZE 5
struct Node{
    char chars[TOT_ARRAY_SIZE]; //定长5,反正总要有定长。
    int len; //表示数组里面实际有几个字母
    Node *next;
    Node(){ len = 0; next = NULL;}
};
struct LinkedList{
```

```
Node *head;
int totalLen;
LinkedList() { head = NULL; totalLen = 0;}
LinkedList(Node *node, int total) {
    head = node;
    totalLen = total;
}

char get(int index) { //index begin from 0
    if (index < 0 || index >= totalLen) return ' ';
    Node *node = head;
    while (node) {
        if (index >= node->len) {
            index -= node->len;
        }
        else
            return node->chars[index];
        node = node->next;
    }
    return ' ';
}

void insert(char c, int index) {
    if (index > totalLen) return;
    Node *node = head;
    while (node) {
        if (index >= node->len) {
            index -= node->len;
            if (index == 0 && node->len < TOT_ARRAY_SIZE) {//insert at the end
                index = node->len;
                break;
            }
        }
        else {
            break;
        }
        node = node->next;
    }

    if (!node) {
        Node *nxt = new Node();
        if (!head) {  //empty linkedlist
            head = nxt;
```

```
            }
            else {
                Node *tail = head;
                while (tail->next) tail = tail->next;
                tail->next = nxt;
            }
            node = nxt;
        }

        if (node->len == TOT_ARRAY_SIZE) {//如果本node已满,需要把最后一个char放到下一个node
            if (node->next == NULL || node->next->len == TOT_ARRAY_SIZE) //如果下一个node也满了或者是空就新建一个
            {
                Node *nxt = new Node();
                nxt->next = node->next;
                node->next = nxt;
            }
            Node *nxt = node->next;
            for (int i = nxt->len - 1; i >= 1; i--) {
                nxt->chars[i] = nxt->chars[i - 1];
            }
            nxt->len += 1;
            nxt->chars[0] = node->chars[node->len - 1];
            node->len -= 1;
        }

        node->len += 1;
        for (int i = node->len - 1; i > index; i--) {
            node->chars[i] =  node->chars[i - 1];
        }
        node->chars[index] = c;
        totalLen += 1;
    }

    void del(int index) {
        if (index < 0 || index >= totalLen) return;
        Node *node = head;
        Node *pre = NULL;
        while (node) {
            if (index >= node->len) {
                index -= node->len;
```

```
            }
            else {
                break;
            }
            pre = node;
            node = node->next;
        }

        if (pre != NULL && pre->len + node->len - 1 <= TOT_ARRAY_SIZE)//尝试和前驱合并
        {
            for (int i = 0; i < node->len; ++i) {
                if (i == index) continue;
                pre->chars[pre->len++] = node->chars[i];
            }
            node->len = 1;
        }

        node->len -= 1;
        totalLen -= 1;

        if (node->len == 0) {
            if (pre) {
                pre->next = node->next;
            }
            else head = node->next; //无前驱代表是头节点
            delete node;
        }
        else {
            for (int i = index; i < node->len; ++i)
            {
                node->chars[i] = node->chars[i + 1];
            }
        }
    }
};
```

接着问: 有没有可能更快呢?

当然有可能了

用rope数据结构

其实就是BST的变种

用splay tree可做， STL中的rope实际上是红黑树，可是谁能现场写TMD红黑树，splay树能讲出来就不错了，注意碰到这个题能讲出用splay优化是极大的加分项，实在不行就用嘴BB几句说用平衡二叉搜索树优化

6.
Given a stream of input, and a API int getNow() to get the current time stamp
1). void record(int val) to save the record.
2). double getAvg() to calculate the averaged value of all the records in 5 minutes
follow up
3). 5分钟内数据太多了怎么办。
4). 如果算中位数呢？分两种, 一个是调用不多(quick sort) 另外一个 调的频繁(两个heap维护一下)

基础版:
注意先问清楚单位，假设是秒

```
#define EXPIRED_TIME 300
struct Node {
    int val;
    int create_time;
    Node() {
        val = create_time = 0;
    }

    Node(int _val, int _create_time) {
        val = _val;
        create_time = _create_time;
    }
};
struct MovingAverage{
    queue<Node> q;
    long long sum;
    MovingAverage() {
        sum = 0;
    }

    bool isExpired(int now, int create_time) {
        return now - create_time >= EXPIRED_TIME;
    }
    void removeExpired() {
        while (!q.empty() && isExpired(time(0), q.front().create_time)) {
            sum -= q.front().val;
```

```
            q.pop();
        }
    }

    void record(int x) {
        Node node(x, time(0));
        q.push(node);
        sum += x;
        removeExpired();
    }

    double getAverage() {
        removeExpired();

        if (q.empty()) return 0;
        return sum * 1.0 / q.size();
    }
};
```

注意这里有个问题，如果对record和getaverage的性能要求特别高，不想出现突刺情况，就是某个请求要等比较久

可以把removeExpired函数拿出来专门用一个线程去跑，每秒调用一次

```
#define EXPIRED_TIME 300
struct Node {
    int val;
    int create_time;
    Node() {
        val = create_time = 0;
    }

    Node(int _val, int _create_time) {
        val = _val;
        create_time = _create_time;
    }
};
struct MovingAverage{
    queue<Node> q;
    long long sum;
    bool lock;
    MovingAverage() {
        sum = 0;
        lock = false;
    }
```

```
bool isExpired(int now, int create_time) {
    return now - create_time >= EXPIRED_TIME;
}
void removeExpired() {
    getLock();
    while (!q.empty() && isExpired(time(0), q.front().create_time)) {
        sum -= q.front().val;
        q.pop();
    }
    freeLock();
}

void record(int x) {
    Node node(x, time(0));
    getLock();
    sum += x;
    q.push(node);
    freeLock();
    //removeExpired();
}

double getAverage() {
    //removeExpired();
    getLock();
    double ans = 0;
    if (!q.empty()) ans = sum * 1.0 / q.size();
    freeLock();
    return ans;
}

void getLock() {
    while (true) {
        for (int i = 0; i < 5; ++i) {
            if (__sync_bool_compare_and_swap(&lock, false, true) == true) return;
        }
        usleep(1000);
    }
}

void freeLock() {
    __sync_bool_compare_and_swap(&lock, true, false);
}
```

```
};
void cleanExpired(MovingAverage * moving_average) {

    while (true) {
        usleep(1000000);
        cout << 123 << endl;
        moving_average->removeExpired();
    }
}
int main() {
    MovingAverage moving_average;
    thread t(cleanExpired, &moving_average);
    t.join();
    return 0;
}
```

注意freeLock()里也需要cas函数操作而不是直接给 lock赋值回来

因为程序在运行时内存实际的访问顺序和程序代码编写的访问顺序不一定一致

例如下面两个线程

```
 // thread 1
while (!ok);
do(x);

// thread 2
x = 42;
ok = 1;
```

此段代码中，ok 初始化为 0，线程 1 等待 ok 被设置为 1 后执行 do 函数。假如说，线程 2 对内存的写操作乱序执行，也就是 x 赋值后于 ok 赋值完成，那么 do 函数接受的实参就很可能出乎程序员的意料，不为 42。

如果想让他严格按照顺序执行，需要memory_barrier， 而cas实际上就有这么一个效果，

你如果code中写了要释放锁，那么这条命令一定会被当即执行，而不是说等一会儿

有兴趣可以看下https://stackoverflow.com/questions/2972389/why-is-compareandswap-instruction-considered-expensive

接着 内存优化版，这里基于基础版本优化:

注意到expire是根据秒来的,那么我们就把每一秒相同的数据合并即可

```
#define EXPIRED_TIME 300
struct Node {
    int val;
    int create_time;
    int num;
```

```
    Node() {
        val = create_time = num = 0;
    }

    Node(int _val, int _create_time) {
        val = _val;
        create_time = _create_time;
        num = 1;
    }
};
struct MovingAverage{
    deque<Node> q;
    long long sum;
    int cnt;
    MovingAverage() {
        sum = 0;
        cnt = 0;
    }

    bool isExpired(int now, int create_time) {
        return now - create_time >= EXPIRED_TIME;
    }
    void removeExpired() {
        while (!q.empty() && isExpired(time(0), q.front().create_time)) {
            sum -= q.front().val;
            cnt -= q.front().num;
            q.pop_front();
        }
    }

    void record(int x) {
        int now = time(0);
        sum += x;
        cnt += 1;
        if (now == q.back().create_time) {
            q.back().val += x;
            q.back().num ++;
            return;
        }
        Node node(x, now);
        q.push_back(node);
        removeExpired();
    }
```

```
    double getAverage() {
        removeExpired();
        double ans = 0;
        if (!q.empty()) ans = sum * 1.0 / cnt;
        return ans;
    }

};
```

接下来求中位数，注意求中位数 压缩内存就不能那么压缩了,所以都在基础版本上改

两种做法

1). GetMedian调用的不多用quickselect

```
int findKth(vector<Node>& v, int k, int start, int end) {
        Node tmp = v[start]; //pivot
        int l = start, r = end;
        while (l < r) {
            while (l < r && v[r].val > tmp.val) r--;
            while (l < r && v[l].val <= tmp.val) l++;
            swap(v[l], v[r]);
        }
        swap(v[start], v[r]);
        if (k == r) return v[r].val;
        if (k < r) return findKth(v, k, start, r - 1);
        return findKth(v, k - r - 1, r + 1, end);
    }
    double getMedian() {
        if (q.empty()) return 0;
        vector<Node> tmp;
        while (!q.empty()) {
            tmp.push_back(q.front());
            q.pop();
        }

        vector<Node> v = tmp;
        double ans = 0;
        if (v.size() % 2 == 0)
            ans = 0.5 * (findKth(v, v.size() / 2, 0, v.size() - 1) +
                    findKth(v, v.size() / 2 - 1, 0, v.size() - 1));
        else
            ans = findKth(v, v.size() / 2, 0, v.size() - 1);
```

```
        for (int i = 0; i < tmp.size(); ++i) {
            q.push(tmp[i]);
        }

        return ans;
    }
```

2).调用的多用 multiset 或者 2个heap（其实也是两个mutltiset，然后保持大小一样或者相差1即可)

插入 logn 查询 O(1)

```
multiset<int> s;
multiset<int> :: iterator it1, it2;
void addNum(int x) {
    int sz = s.size();
    s.insert(x);
    if (!sz) it1 = it2 = s.begin();
    else if (sz % 2 == 1) {
        if (x >= *it1) it2++;
        else it1--;
    } else {
        if (x > *it1 && x < *it2) it1++, it2--;
        else if (x >= *it2) it1++;
        else it1 = --it2;
    }
}

void delNum(int x) {
    int sz = s.size();
    auto it = s.find(x);
    if (sz == 0 || it == s.end()) return;
    if (sz == 1) {
        s.erase(it);
        it1 = it2 = s.end();
        return;
    }
    if (sz % 2 == 1) {
        if (*it1 == x) {
            auto tmp = it1;
            --it1; ++it2;
            s.erase(tmp);
        }
        else if (*it1 > x) {
            s.erase(it);
```

```
            ++it2;
        }
        else {
            s.erase(it);
            --it1;
        }
    }
    else {
        if (*it1 == x) {
            auto tmp = it1;
            ++it1;
            s.erase(tmp);
        }
        else if (*it2 == x) {
            auto tmp = it2;
            it2--;
            s.erase(tmp);
        }
        else if (*it1 > x) {
            it1++;
            s.erase(it);
        }
        else {
            it2--;
            s.erase(it);
        }
    }
}

double getMedian() {
    if (it1 == s.end()) return 0;
    return (*it1 + *it2) * 0.5;
}
```

两个heap版:

```
multiset<int> low, high;
void addNum(int x) {
    low.insert(x);
    int y = *low.rbegin();
    high.insert(y);
    low.erase(low.find(y));

    if (low.size() < high.size()) {
```

```
        low.insert(*high.begin());
        high.erase(high.begin());
    }
}
void delNum(int x) {
    if (x > *low.rbegin()) high.erase(high.find(x));
    else low.erase(low.find(x));

    if (low.size() < high.size()) {
        low.insert(*high.begin());
        high.erase(high.begin());
    }
    else if (low.size() > high.size() + 1) {
        int y = *low.rbegin();
        high.insert(y);
        low.erase(low.find(y));
    }
}

double getMedian() {
    if (low.size() == 0) return 0;
    if (low.size() > high.size()) return *low.rbegin();
    else return (*low.rbegin() + *high.begin()) * 0.5;
}
```

3). 如果内存放不下，求中位数怎么办。

能做到他问这个follow up 说明这个做题速度可以

如果是单机压内存就用树状数组，二分做

如果说单机存不下就多机，也是树状数组

反正我只想到了这个思路

7. git version。找到全部commits，让实现bfs。然后让找两个commit最早的公共commit，先bfs搜索其中一个commit的所有ancestor，用hashmap存一下，然后bfs搜索第二个commit的祖先。这里有两个地方可以提前结束搜索，提出来应该很好。

```
unordered_set<int> vis;
struct Node {
    int id;
    vector<Node*> parents;
    Node() {}
    Node(int _id) { id = _id;}
```

```
};
vector<int> gao(Node * root) {
    vector<int> ans;
    if (root == NULL) return ans;
    queue<Node*> q;
    q.push(root);
    vis.insert(root->id);
    while (!q.empty()) {
        Node *node = q.front();
        q.pop();
        ans.push_back(node->id);
        for (int i = 0; i < node->parents.size(); ++i) {
            if (vis.find(node->parents[i]->id) != vis.end()) continue;
            vis.insert(node->parents[i]->id);
            q.push(node->parents[i]);
        }
    }

    return ans;
}
```

第二问:

bfs一遍， 然后用map存下到每个点的距离

另外一个再bfs的时候更新下。

优化就是双向bfs

两个队列分别从两个节点出发

每次都分别走一层

然后谁先碰到另外一方走过的 就返回那个节点即可

直接写最优解:

```
struct Node {
    int id;
    vector<Node*> parents;
    Node() {}
    Node(int _id) { id = _id;}
};
int gao(Node * node1, Node * node2) {
    if (!node1 || !node2) return -1;
    vector<Node*> roots;
    roots.push_back(node1);
    roots.push_back(node2);
    unordered_set<int> vis[2];
```

```
    queue<Node*> q[2];
    for (int i = 0; i < 2; ++i) {
        q[i].push(roots[i]);
        vis[i].insert(roots[i]->id);
    }

    while (!q[0].empty() && !q[1].empty()) {
        for (int i = 0; i < 2; ++i) {
            int sz = q[i].size();
            while (sz--) {
                Node *tmp = q[i].front();
                q[i].pop();
                for (int j = 0; j < tmp->parents.size(); ++j) {
                    if (vis[i ^ 1].find(tmp->parents[j]->id) != vis[i^1].end()) return tmp->parents[j]->id;
                    if (vis[i].find(tmp->parents[j]->id) != vis[i].end()) continue;
                    vis[i].insert(tmp->parents[j]->id);
                    q[i].push(tmp->parents[j]);
                }
            }
        }
    }

    return -1;
}
```

8. 两个vector的distinct union

sort之后 2指针移动

```
void addNum(vector<int>& v, int x) {
    if (v.size() > 0 && v[v.size() - 1] == x) return;
    v.push_back(x);
}
vector<int> getUnion(vector<int> &v1, vector<int> &v2) {
    int i = 0, j = 0;
    vector<int> ans;
    while (i < v1.size() && j < v2.size()) {
        if (v1[i] < v2[j]) addNum(ans, v1[i++]);
        else if (v1[i] > v2[j]) addNum(ans, v2[j++]);
        else {
            addNum(ans, v1[i++]);
            j++;
        }
```

```
    }
    if (i < v1.size()) {
        while (i < v1.size()) addNum(ans, v1[i++]);
    }
    if (j < v2.size()) {
        while (j < v2.size()) addNum(ans, v2[j++]);
    }

    return ans;
}
merge k类似，只不过用优先队列而已
void addNum(vector<int>& v, int x) {
    if (v.size() > 0 && v[v.size() - 1] == x) return;
    v.push_back(x);
}
struct Node {
    int val;
    int i, j;
    Node() {}
    Node(int _val, int _i, int _j) {
        val = _val; i = _i; j = _j;
    }
    bool operator < (const Node & cmp) const {
        if (val == cmp.val) return i < cmp.i;
        return val > cmp.val;
    }
};
vector<int> getUnion(vector<vector<int> > vecs) {
    vector<int> ans;
    priority_queue<Node> q;
    for (int i = 0; i < vecs.size(); ++i) {
        if (vecs[i].size() == 0) continue;
        q.push(Node(vecs[i][0], i, 0));
    }
    while (!q.empty()) {
        Node tmp = q.top();
        q.pop();
        addNum(ans, tmp.val);
        if (vecs[tmp.i].size() > tmp.j + 1) {
            q.push(Node(vecs[tmp.i][tmp.j + 1], tmp.i, tmp.j + 1));
        }
    }
    return ans;
```

}
实际的题应该是这个：
Given n sorted stream, and a constant number k. The stream type is like iterator and it has two functions, move() and getValue(), find a list of numbers that each of them appears at least k times in these streams. Duplicate numbers in a stream should be counted as once.
这个题在上面的基础上有3点要注意:
1) 从某个队列里拿数据的时候，要判断是否有重复的，有重复的就continue
2) 存两个全局变量，一个是上一个数是多少，一个是统计个数
3) heap的大小比k小的时候可以break，这里要问清楚这个流是不是无穷尽的，一般他的follow up就是说有某个流突然特别长

9 .给一个list， 如何把里面的字符分配到尽量少的子list里，并且每个子list没有重复元素
['a','b','c','a','a','b']， 可以分成['a', 'b', 'c'], ['a', 'b'], ['a']
['a', 'a', 'a', 'b', 'b', 'b']，可以分成['a', 'b'], ['a', 'b'], ['a', 'b']

先给出了O(n*2)的解法，后来发现可以先数一遍字符个数，找到出现最多的，比如a出现3次，就建3个子list，然后把每种字符round robin那样放进各个list就行了， 这样是O(n)

```
vector<vector<int> > gao(vector<int> & v) {
    unordered_map<int, int> mp;
    for (int i = 0; i < v.size(); ++i) {
        mp[v[i]]++;
    }
    int mx = 0;
    for (auto it : mp) {
        mx = max(mx, it.second);
    }
    vector<vector<int> > ans;
    for (int i = 0; i < mx; ++i) {
        vector<int> tmp;
        ans.push_back(tmp);
    }
    int cur = 0;
    for (auto it :mp) {

        for (int i = 0; i < it.second; ++i) {
            ans[cur].push_back(it.first);
            cur = (cur + 1) % mx;
        }
    }
```

```
    return ans;
}
```

10.Validate Python code

大概就是4个rule

1. 刚开始不能缩进
2. control block的下一行要缩进
3. 缩进要一样

The input is String[], each line is a string
if it's valid, return -1, otherwise return the line number.
follow up1: what is the last line is control block?
follow up2: what if there is comment line (" #")

```
struct Node {
    int level;
    bool iscontrol;
    Node() { level = 0; iscontrol = 0;}
    Node(int _level, int _iscontrol) {
        level = _level;
        iscontrol = _iscontrol;
    }
};
stack<Node> st;
int base ;
int getIndent(string & s) {
    int sum = 0;
    for (int i = 0; i < s.size(); ++i) {
        if (s[i] == ' ') sum++;
        else break;
    }
    if (sum > 0 && base == 0) {
        base = sum;
    }
    if (base)
        return sum / base;
    return 0;
}
bool isControl(string &s) {
    int sz = s.size();
    for (int i = sz - 1; i >= 0; --i) {
        if (s[i] == ' ') continue;
        if (s[i] == ':') return true;
        return false;
```

```
    }
    return false;
}

string removeComment(string &s) {
    string tmp = "";
    int valid = 0;
    for (int i = 0; i < s.size(); ++i) {
        if (s[i] == '#') break;
        tmp += s[i];
        if (s[i] != ' ') valid ++;
    }

    if (valid == 0) tmp = "";
    return tmp;
}

int isvalid(vector<string> &v) {
    int lastline = 0;
    for (int i = 0; i < v.size(); ++i) {
        string tmp = removeComment(v[i]);
        if (tmp.empty()) continue;
        int level = getIndent(tmp);
        bool iscontrol = isControl(tmp);
        if (st.empty()) {
            if (level) return i;
        }
        else if (st.top().iscontrol) {
            if (level != st.top().level + 1) return i;
        }
        else {
            while(!st.empty() && st.top().level >= level) {
                st.pop();
            }
            if (!st.empty() && !st.top().iscontrol) return i;
        }
        st.push(Node(level, iscontrol));
        lastline = i;
    }
    if (!st.empty() && st.top().iscontrol) return lastline;

    return -1;
}
```

11.给一个数组，例如[8,5,3,6,1,4,7], 返回任意一个local minimum，也就是任意一个谷值，比如在这个例子里，返回3和1都是正确的。数组没有重复的数，所以不会出现[1,1,1,1,1]这种没有结果的情况情况。
O(n)遍历不写了

直接logn

```
int gao(vector<int> &v) {
    int l = 0, r = v.size() - 1;

    while (l < r) {
        int mid = (l + r) / 2;
        if (v[mid] < v[mid + 1]) {
            r = mid;
        }
        else {
            l = mid + 1;
        }
    }
    return v[r];
}
```

12.quintiles的个数，比如2就是求中位数，3就是把数组分成3等分的那两个数的值。
数值是用Pair给出的，(value count)值和个数的Pair, 比如(1, 3)表示数组中有3个1.
1)Pair的顺序不是sorted.
还有就是Pair的个数。
题目的示例
3 三等分
3 三个Pair
7 2 数组中有2个7
6 2 数组中有2个6
5 2 数组中有2个5
输出应该是
5
6
k-th quintiles的index是N*k/Q.
(N是数组长度，在第一步时对所有Pair中count求和就是了），Q是quintiles的个数
预处理求和 然后二分即可

```
vector<int> sum;
vector<pair<int, int> >  ps;
void init(vector<pair<int, int> > & v) {
    if (v.size() == 0) return;
    sum.resize(v.size());
    sort(v.begin(), v.end());
    sum[0] = v[0].second;
    for (int i = 1; i < v.size(); ++i) sum[i] = sum[i - 1] + v[i].second;
}

double getQuantile(int q, int k) {
    if (k == 0) return ps[0].first;
    else if (k == q) return ps[ps.size() - 1].first;
    int total = sum[ps.size() - 1];
    double tmp = ceil(total * k * 1.0 / q);
    int idx = (int) tmp;
    int pos = lower_bound(sum.begin(), sum.end(), idx) - sum.begin();
    if (total * k % q != 0)
        return ps[pos].first;
    else
        return (ps[pos].first + ps[pos + 1].first) / 2.0;
}
```

13. Root to Leaf Min Cost

Given a tree,(binary tree possibily) every tree edge has a cost， find the least cost or find the leaf node that the cost of path that from root to leaf is the least.

基础版: dfs就行

假设边用个vector存

```
void dfs(int u, vector<int> & ans, vector<int> &pre, vector<vector<pair<int, int> > >& g) {
    if (g[u].size() == 0) {
        ans[u] = 0;
        return;
    }

    for (int i = 0; i < g[u].size(); ++i) {
        int v = g[u][i].first;
        dfs(v, ans, pre, g);
        if (ans[v] != INT_MAX && ans[v] + g[u][i].second < ans[u]) {
            ans[u] = ans[v] + g[u][i].second;
            pre[u] = v;
```

```
        }
    }
}

void solve(int n, vector<vector<pair<int, int> > >& g) {
    vector<int> ans(n, INT_MAX);
    vector<int> pre(n, -1);
    dfs(0, ans, pre, g);
    cout << ans[0] << endl;
    int x = 0;
    while (x != -1) {
        cout << x << endl;
        x = pre[x];
    }
}
```

扩展到dag

其实就是个dag的dp

做拓扑排序的时候dp就行了，我看网上的题解都是dijkstra，但我觉得拓扑排序可做而且复杂度还低

dag版本:

注意要算每个点的出度，然后存一份逆向边

时间复杂度O(n + m)

```
void solve(int n, vector<vector<pair<int, int> > >& g) {
    vector<vector<pair<int, int> > > rev;
    for (int i = 0; i < g.size(); ++i) {
        vector<pair<int, int> > tmp;
        rev.push_back(tmp);
    }
    vector<int> ans(n, INT_MAX);
    vector<int> pre(n, -1);
    vector<int> d(n, 0);
    queue<int> q;
    for (int i = 0; i < g.size(); ++i) {
        d[i] = g[i].size();
        if (d[i] == 0) {
            ans[i] = 0;
            q.push(i);
        }

        for (int j = 0; j < g[i].size(); ++j) {
```

```
            rev[g[i][j].first].push_back(make_pair(i, g[i][j].second));
        }
    }

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int i = 0; i < rev[u].size(); ++i) {
            int v = rev[u][i].first;
            int w = rev[u][i].second;
            if (ans[u] != INT_MAX && ans[u] + w < ans[v]) {
                ans[v] = ans[u] + w;
                pre[v] = u;
            }

            d[v]--;
            if (d[v] == 0) q.push(v);
        }
    }

    cout << ans[0] << endl;
    int x = 0;
    while (x != -1) {
        cout << x << endl;
        x = pre[x];
    }
}
```

14，新题很简单，就是给你一个List 里面有ads，然后写一个get（） function，来随机get一个list里面的ad，不能重复，而且get 完了后 return null。

每次random一个，然后跟List最后一个交换即可， 接着抛弃最后一个

```
int gao(vector<int> &v, int &tot) {
    if (tot == 0) return -1;
    int idx = rand() % tot;
    swap(v[idx], v[tot - 1]);
    tot--;
    return v[tot];
}
int solve(vector<int> & v) {
    int tot = v.size();
    return gao(v, tot);
}
```

15 . auto complete

Say I'm typing on a phone. Given a prefix String,and a dictionary.
Find all auto-complete word for the given prefix string

```
struct Node {
    bool has_word;
    Node *next[26];
    Node() {
        for (int i = 0; i < 26; ++i) next[i] = NULL;
        has_word = false;
    }
};
Node root;
void insert(const string & s) {
    if (s.empty()) return;
    Node * node = &root;
    for (int i = 0; i < s.size(); ++i) {
        int c = (int) (s[i] - 'a');
        if (node->next[c] == NULL) node->next[c] = new Node();
        node = node->next[c];
    }
    node->has_word = true;
}
void dfs(Node *node,const string & prefix, vector<string> & ans) {
    if (node->has_word) ans.push_back(prefix);
    for (int i = 0; i < 26; ++i) {
        if (node->next[i]) {
            string tmp = prefix + (char)(i + 'a');
            dfs(node->next[i], tmp, ans);
        }
    }
}
vector<string> getAutoStrings(const string &s) {
    vector<string> ans;
    if (s.empty()) return ans;
    Node *node = &root;
    for (int i = 0; i < s.size(); ++i) {
        int c = (int) (s[i] - 'a');
        if (node->next[c] == NULL) return ans;
        node = node->next[c];
    }
    dfs(node, s, ans);
```

```
    return ans;
}
```

二分搜索版本:

注意将所有字符串塞入vector后排个序

然后做法如下

```
vector<string> solve(const string & s) {
    vector<string> ans;
    if (s.empty()) return ans;
    int idx = lower_bound(v.begin(), v.end(), s) - v.begin();
    for (int i = idx; i < v.size(); ++i) {
        if (v[i].substr(0, s.size()) != s) break;
        ans.push_back(v[i]);
    }
    return ans;
}
```

hash版本:

其实用map就行了， map[string, vector]就可以

```
vector<string> v;
unordered_map<string, vector<int> > mp;
void add(const string &s) {
    int idx = v.size();
    v.push_back(s);
    string tmp = "";
    for (int i = 0; i < s.size(); ++i) {
        tmp += s[i];
        mp[tmp].push_back(idx);
    }
}
vector<string> solve(const string & s) {
    vector<string> ans;
    if (s.empty()) return ans;
    for (int i = 0; i < mp[s].size(); ++i) {
        ans.push_back(v[mp[s][i]]);
    }
    return ans;
}
```

还有一题是输入 DV 返回DataViewer这种

Dic里有 GraphView和GraphViewController

input GraVi 或者 GV等
output GraphView, GraphViewController

D获得一个list ,v 获得一个list 然后就是得到在这个k个list中都出现的
代码就比较长了

```
struct Node {
    unordered_map<int, vector<int> > v;
    bool has_word;
    Node * next[26];
    Node() {
        for (int i = 0; i < 26; ++i) next[i] = NULL;
        has_word = false;
    }
};
Node root;
void insert(const string & s, int idx, int k) {
    Node * node = &root;

    for (int i = 0; i < s.size(); ++i) {
        int x = s[i] - 'a';
        if (!node->next[x]) node->next[x] = new Node();
        node = node->next[x];
    }
    node->v[k].push_back(idx);
    node->has_word = true;
}

void split(const string & s, vector<string> & words) {
    string tmp = "";
    for (int i = 0; i < s.size(); ++i) {
        if (s[i] >= 'A' && s[i] <= 'Z') {
            if (!tmp.empty()){
                words.push_back(tmp);
                tmp = "";
            }
        }
        tmp += tolower(s[i]);
    }

    if (!tmp.empty()) words.push_back(tmp);
}
void init_one(const string & s, int idx) {
    vector<string> words;
```

```
    split(s, words);
    for (int i = 0; i < words.size(); ++i) insert(words[i], idx, i);
}

void init(vector<string> & dic) {
    for (int i = 0; i < dic.size(); ++i) {
        init_one(dic[i], i);
    }
}

void dfs(Node * node, vector<int> &v, int k) {
    if (node->has_word) {
        auto it = node->v.find(k);
        if (it != node->v.end()) {
            for (int i = 0; i < it->second.size(); ++i) {
                v.push_back(it->second[i]);
            }
        }
    }

    for (int i = 0; i < 26; ++i) {
        if (node->next[i]) dfs(node->next[i], v, k);
    }
}

void search(const string & s, vector<int>& v, int k) {
    Node * node = &root;
    for (int i = 0; i < s.size(); ++i) {
        int x = s[i] - 'a';
        if (!node->next[x]) return;
        node = node->next[x];
    }
    dfs(node, v, k);
}

void gao(const string & s, vector<int> & ans) {
    vector<string> words;
    split(s, words);
    unordered_map<int, int> mp;
    for (int i = 0; i < words.size(); ++i) {
        vector<int> v;
        search(words[i], v, i);
        for (int j = 0; j < v.size(); j++) mp[v[j]] += 1;
```

```
    }

    for (auto it : mp) {
        if (it.second == words.size()) ans.push_back(it.first);
    }
}
int main()
{
    vector<string> dic = {"GraphView", "DetailedDataView",
    "DataGraphView", "DataController", "MouseClickHandler",
    "GraphViewController", "MathCalculationHandler", "DataScienceView"};
    init(dic);
    string s;
    while (cin >> s) {
        vector<int> ans;
        gao(s, ans);
        for (int i = 0; i < ans.size(); ++i) cout << dic[ans[i]] << " " ;
        cout << endl;
    }
    return 0;
}
```

16. expired map

这个题是个好题，为什么这么说呢，因为可聊的东西多啊，所以通过率就高

基础版本:

```
struct Node {
    string val;
    int expired_time;
    Node() {}
    Node(string _val, int _expired_time) {
        val = _val;
        expired_time = _expired_time;
    }
};

map<string, Node> mp;

int get(const string & key, string & val) {
    int now = time(0);
    auto it = mp.find(key);
    if (it == mp.end()) return -1;
    if (it->second.expired_time <= now) {
        mp.erase(key);
```

```
        return -1;
    }
    val = it->second.val;
    return 0;
}

void put(const string & key, const string & val, int duration) {
    int now = time(0);
    int expired_time = now + duration;
    Node node(val, expired_time);
    mp[key] = node;
}
```

优先队列优化删除过期元素：

```
struct Node {
    string val;
    int expired_time;
    Node() {}
    Node(string _val, int _expired_time) {
        val = _val;
        expired_time = _expired_time;
    }
    bool operator <(const Node &cmp) const {
        if (expired_time == cmp.expired_time)
            return val < cmp.val;

        return expired_time > cmp.expired_time;
    }
};

map<string, Node> mp;
priority_queue<Node> q;

void check_expired() {
    int now = time(0);
    while (!q.empty() && q.top().expired_time <= now)
    {
        Node tmp = q.top();
        q.pop();
        mp.erase(tmp.val);
    }
}
```

```
int get(const string & key, string & val) {
    check_expired();
    int now = time(0);
    auto it = mp.find(key);
    if (it == mp.end()) return -1;
    val = it->second.val;
    return 0;
}

void put(const string & key, const string & val, int duration) {
    check_expired();
    int now = time(0);
    int expired_time = now + duration;
    Node node(val, expired_time);
    mp[key] = node;
    q.push(Node(key, expired_time));
}
```

加多线程再优化:

```
bool lock = false;
void getlock() {
    while (true) {
        for (int i = 0; i < 5; ++i) {
            if (__sync_bool_compare_and_swap(&lock, false, true) == true) return;
        }
        usleep(1000);
    }
}

void unlock() {
    __sync_bool_compare_and_swap(&lock, true, false);
}

void check_expired() {
    int now = time(0);
    while (true)
    {
        getlock();
        if(!q.empty() && q.top().expired_time <= now)
        {
            Node tmp = q.top();
            q.pop();
            mp.erase(tmp.val);
           unlock();
```

```
        } else {
            unlock();
            sleep(1);
        }


    }
}

int get(const string & key, string & val) {
    getlock();
    int now = time(0);
    auto it = mp.find(key);
    int ret = 0;
    if (it == mp.end()) ret = -1;
    else if (it->second.expired_time <= now) ret = -1;
    else
        val = it->second.val;
    unlock();
    return ret;
}

void put(const string & key, const string & val, int duration) {
    getlock();
    int now = time(0);
    int expired_time = now + duration;
    Node node(val, expired_time);
    mp[key] = node;
    q.push(Node(key, expired_time));
    unlock();
}
```

接着还能优化吗?

当然了，直接开喷就行了:

如果不是因为面试时间不够，我不会用map，因为直接server一挂缓存数据全没了

我会选基于共享内存的hashtable,每个节点是个linked list

这样lock granularity会特别小，并且用cas可以实现lock-free

然后priority queue这个用 skip list 替换，也可以在多线程里效率很高

结了，不吹逼的讲，这应该是工程中的很优的解法了

能讲出这些东西面试官应该目瞪口呆了，心想：你TM懂的不少啊

但是注意要循序渐进，别TM上来自己就要写个hashtable，那是闲的蛋疼。

17. normalize title
Given a rawTitle, and a list(or array) of clean titles. For each clean title, the raw title can get a "match point". For example, if raw title is "senior software engineer" and clean titles are "software engineer" and "mechanical engineer", the "match point" will be 2 and 1. In this case we return "software engineer" because it has higher "match point”.

这题我的思路就是建倒排链

```
map<string, vector<int> > mp;
void Init(vector<string> & title) {
    for (int i = 0; i < title.size(); ++i) {
        string tmp = "";
        for (int j = 0; j < title[i].size(); ++j) {
            if (title[i][j] != ' ') tmp += title[i][j];
            if (j == title[i].size() - 1 || title[i][j] == ' ') {
                if (tmp != "") {
                    mp[tmp].push_back(i);
                    tmp = "";
                }
            }
        }
    }
}
string solve(string & target, vector<string> & title) {
    unordered_map<int, int> cnt;
    int ans = 0, id = 0;
    string tmp = "";
    for (int i = 0; i < target.size(); ++i) {
        if (target[i] != ' ') tmp += target[i];
        if (i == target.size() - 1 || target[i] == ' ') {
            if (tmp != "") {
                for (int j = 0; j < mp[tmp].size(); ++j) {
                    int idx = mp[tmp][j];
                    if (cnt.find(idx) == cnt.end()) cnt[idx] = 1;
                    else cnt[idx] += 1;

                    if (cnt[idx] > ans) {
                        ans = cnt[idx];
                        id = idx;
                    }
                }
```

```
                tmp = "";
            }
        }
    }

    return title[id];
}
```

优化 simhash? 没了解

18
Interviewees said this question(and follow up) is the same as Leetcode 243
word distance 1,2,3 leetcode
1) 两个词不相同

```
int shortestDistance(vector<string>& words, string word1, string word2) {
        int pos1 = -1;
        int ans = words.size() + 1;
        for (int i = 0; i < words.size(); ++i) {
            if (words[i] == word1 || words[i] == word2) {
                if (pos1 != -1 && words[i] != words[pos1]) {
                    ans = min(ans, i - pos1);
                }
                pos1 = i;
            }
        }
        return ans;
    }
```

2) 是个类，查询量很大

```
unordered_map<string, vector<int> > mp;
    WordDistance(vector<string> words) {
        for (int i = 0; i < words.size(); ++i)
            mp[words[i]].push_back(i);
    }

    int shortest(string word1, string word2) {
        int i = 0, j = 0;
        int ans = INT_MAX;
        while (i < mp[word1].size() && j < mp[word2].size()) {
            if (mp[word1][i] < mp[word2][j]) {
                ans = min(ans, mp[word2][j] - mp[word1][i]);
                i ++;
            } else {
```

```
                ans = min(ans, mp[word1][i] - mp[word2][j]);
                j ++;
            }

        }
        return ans;
    }
```

3) 两个词可以一样

```
int shortestWordDistance(vector<string>& words, string word1, string word2) {
        int pos = -1;
        int ans = words.size() + 1;
        for (int i = 0; i < words.size(); ++i) {
            if (words[i] == word1 || words[i] == word2) {
                if (pos != -1 && (word1 == word2 || words[i] != words[pos])) {
                    ans = min(ans, i - pos);
                }
                pos = i;
            }

        }
        return ans;
    }
```

19.Now you have a dice, and throw it multiple times.
Find the possibility the sum of points is a target number

```
int dfs(int dice, int target) {
    if (vis[dice][target] != -1) return vis[dice][target];
    int ans = 0;
    if (dice == 0) {
        if (target == 0) ans = 1;
    }
    else if (dice * 6 >= target && target >= dice) {
        for (int i = 1; i <= 6; ++i) {
            if (target >= i)
                ans += dfs(dice - 1, target - i);
        }
    }
    return vis[dice][target] = ans;
}
```

刚开始可以不写记忆化，随后follow up 可以写

然后dp版本:

```
for (int i = 1; i <= 6; ++i) {
        dp[1][i] = 1;
    }
    for (int i = 2; i <= dice; ++i) {
        for (int j = i; j <= target; ++j) {
            for (int k = 1; k <= 6; ++k) {
                if (j - k < i - 1) break; //target小于骰子个数  明显不可能
                dp[i][j] += dp[i - 1][j - k];
            }
        }
    }
```

20 You are given a list of jobs, each job has an ID number(type is long).
Implement two functions,
1.expire(long jobid) to set a job as "expired"
2.isexpired(long jobid) to check if a job is "expired"

刚开始用map

然后map肯定不给用的

用bitmap 如果说单机存不下

问问值域看能不能压缩下， 再之后看能不能用多机存

接着就说bloom filter

bloom filter的主要作用是把bit比较多的hash后存到bit比较小的内存空间里

比如64位的存到16位里

证明错误率 https://www.douban.com/note/342448148/

21. How to find the minimum value in a stack with constant time complexity?

两个栈 优化点是1个栈只存差值

基础:

```
stack<int> s;
    stack<int> mi;
    void push(int x) {
        s.push(x);
        if(mi.empty()) mi.push(x);
        else {
            mi.push(min(x, mi.top()));
        }
    }
```

```
    void pop() {
        s.pop();
        mi.pop();
    }

    int top() {
        return s.top();
    }

    int getMin() {
        return mi.top();
    }
```

优化内存版:

```
stack<long long> df;
    long long mi;
    MinStack() {
        while (!df.empty()) df.pop();
        mi = 0;
    }

    void push(int x) {
        if (df.empty()) {
            mi = x;
            df.push(0);
        }
        else {
            long long d = mi - (long long)x;
            df.push(d);
            mi = min(mi, (long long)x);
        }
    }

    void pop() {
        long long x = df.top();
        if (x > 0) mi += x;
        df.pop();
    }

    int top() {
        if (df.top() > 0) return mi;
        return (int)(mi - df.top());
    }
```

```
    int getMin() {
        return mi;
    }
```

22.

two sorted array 找median leetcode上有

```
double gao(vector<int> &nums1, vector<int> &nums2, int target) {
        int st1 = 0, st2 = 0;
        int sz1 = nums1.size(), sz2 = nums2.size();
        while (true)
        {
            if (!sz1 || !sz2)
            {
                if (!sz2) return nums1[st1+target-1];
                else return  nums2[st2 + target-1];

            }
            if (target == 1)
            {
                return  min(nums1[st1], nums2[st2]);
            }
            int x, y;
            if (sz1 >= sz2)
            {
                y = min(target / 2, sz2);
                x = target - y;

            }
            else
            {
                x = min(target / 2, sz1);
                y = target - x;
            }
            if (nums1[st1 + x - 1] > nums2[st2 + y - 1])
            {
                target = target - y;
                st2 = st2 + y;
                sz2 = sz2 - y;
            }
            else if (nums1[st1 + x - 1] < nums2[st2 + y - 1])
            {
                target = target - x;
```

```
            st1 = st1 + x;
            sz1 = sz1 - x;
        }
        else
            return nums1[st1 + x  - 1];
    }
}
double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

    int target1 = (nums1.size() + nums2.size() + 1) / 2;
    int target2 = (nums1.size() + nums2.size() + 2) / 2;
    return (gao(nums1,nums2,target1) + gao(nums1,nums2,target2)) / 2.0;
}
```