```
Top candidate
vector<string> topCandidate(vector<pair<string, int>>& votes, int T) {
        unordered_map<string, int> m;
        vector<string> maxCandidate;
        int maxCount = 0;
        for(auto& p: votes) {
        if(p.second > T) continue;
        m[p.first]++;
        if(m[p.first] >= maxCount) {
                if(m[p.first] > maxCount) maxCandidate.clear();
                maxCount = m[p.first];
                maxCandidate.push_back(p.first);
       }
       return maxCandidate;
 }
// follow1:K, 给出小于时间 T 得票前 K 的 candidate
struct compare {
bool operator()(pair<string, int> p1, pair<string, int> p2) {
        return p1.second > p2.second;
    }
};
vector<string> bestCandi (vector<pair<string, int>> votes, int time, int k) {
       unordered_map<string, int> voteBeforeT;
       for(auto vote : votes) {
           if(vote.second <= time) {
                voteBeforeT[vote.first] += 1;
           }
       }
       priority_queue<pair<string, int>, vector<pair<string, int>>, compare> pq;
       for(auto candi : voteBeforeT) {
           if(pq.size() < k) pq.push(candi);
           else {
              if(candi.second > pq.top().second) {
                  pq.pop();
                  pq.push(candi);
              }
        vector<string> res;
        while(!pq.empty()) {
                res.push_back(pq.top().first);
                pq.pop();
       }
       return res;
}

//follow2: 各一个 candidate list，求出最小的 T，使得使用 follow1 function 可以得到该 list
//traverse T, call follow up2, compare

//given K and list, find the T
```

给选票的 array 和候选人 id 的 array，问有没有一个时间，得票最多的候选人和给定的候选人的 array 是一致的，人头对上就行，不考虑先后顺序。

先看看给定的候选人有多少个，然后直接调用 method2

每加一票，最多就影响一个候选人的排序，就跟面试官说，记录一个票数最少的人，超过他了就更新

LFU

维护一个 candidate 出现最大次数的 list，每次最大次数变更的时候，相对应维护这个 list

可以先按照投票 time, sort 一下选票的 array,然后 scan array，遇到一个新的时间更新下 LFU，同时 check 一下当前时间的排序状况。

坐距离最远的 idx

```
//存连续空位长度，每次取最大的出来，截成大概一半放回去，都是 log n 操作
// Follow up：如果持续不停地有人来，要如何设计算法和数据结构。
//先预处理一遍所有的两个相邻人的坐标(x, y) 放到 PQ 里，自定义 comparator 比区间长度(maxheap)，来一个人 poll out 最长的区间人坐进去。同时更新一下新产生的区间。
class interval{
	int start;
	int end;
	interval(int a, int b):start(a), end(b){}
};
struct cmp {
    bool operator () (interval a, interval b) {
	 int len1=a.end-a.start;
	int len2=b.end-b.start;
	if(len1==len2){
		return a.start-b.start;
	}
	else{
		return len2-len1;
	}
    }
};

priority_queue<interval, vector<interval>, cmp> pq;
interval res=pq.top();
return res.start+(res.end-res.start)/2;


expire hashmap
计时 map 的 set 和 get 功能
set(key, val, time) val time for key
get(key,time) return val if not found return
private:
	map<key, pair<val, time>> dict;
public:
	void put(key, value, expireTime){
		dict[key]=make_pair(value, expireTime);
	}
	val get(key, currentTime){
		if(dict.find(key)==dict.end()) return null;
		else if(dict.find(key).second>curent){
			return dict[key].first;
		}
		else
			return null;
	}
	void cleanup(time){
		auto it=dict.lower_bound(time);
		if(it==dict.begin())return;
		if(it->second.second==time) it--;
		dict.erase(dict.begin(), it);
	}
followup: 用少于 linear 的时间实现 cleanup
```

机器人

http://www.1point3acres.com/bbs/thread-289514-1-1.html

```
class Solutions {
private:
    int direction;
    bool move(Robot &bot, int x) {
        int turnRightCount = (x - direction) % 4;
        for(int i = 0; i < turnRightCount; i++) {
            bot.turnRight();
        }
        return bot.move();
    }
    void dfs(Robot &bot, int x, int y, unordered_set<pair<int, int>> &visited) {
        bot.clean();
        visited.insert(make_pair(x, y));
        pair<int, int> up = make_pair(x, y + 1);
        pair<int, int> down = make_pair(x, y - 1);
        pair<int, int> left = make_pair(x - 1, y);
        pair<int, int> right = make_pair(x + 1, y);
        if(visited.find(right) == visited.end() && move(bot, 1)) {
            dfs(bot, x + 1, y, visited);
             move(bot, 3);
        }
        if(visited.find(down) == visited.end() && move(bot, 2)) {
            dfs(bot, x, y - 1, visited);
            move(bot, 4);
        }
        if(visited.find(left) == visited.end() && move(bot, 3)) {
            dfs(bot, x - 1, y, visited);
            move(bor, 1);
        }
        if(visited.find(up) == visited.end() && move(bot, 4)) {
            dfs(bot, x, y + 1, visited);
            move(bor, 2);
} }
public:
    void cleanRoom(Robot &bot) {
        unordered_set<pair<int, int>> visited;
        dfs(bot, 0, 0, visited);
} }
```

机器人

http://www.1point3acres.com/bbs/thread-289514-4-1.html
http://www.1point3acres.com/bbs/thread-345555-3-1.html

```
vector<vector<int>> dirs{{-1,0}, {0,-1}, {1,0}, {0,1}};
set<pair<int, int>> visited;
void dfs(int x, int y, int d, set<pair<int, int>>&visited){
       visited.push(make_pair(x, y));
       this.clean();
       for(int i=0; i<4; i++){
              this.TureRight(i>0?1:0);//first go ahead, then each time one turn
              int nd=(d+i)%4;
              int i=x+dirs[nd][0];
              int j=y+dirs[nd][1];
              if(!visited.find(make_pair(x, y))){
                     if(this.Move()){
                            dfs(I, j, nd, visited);
                            this.Move();
                            //reset to direction before
                            this.TurnRight(2);
                     }
                     else
                            visited.push(make_pair(I, j));
              }
       }
       //go opposite direction of d
       this.TureRight(3)
}
```

```
void dfs(robot &myRobot, int x, int y, set<pair<int, int>>&visited, int d){
       if(visited.find(make_pair(x, y))) return;
       myRobot.clean();
       visited.push(make_pair(x,y));
       vector<vector<int>> dirs{{-1,0}, {0,-1}, {1,0}, {0,1}};
       //During each for loop iteration, the robot will turn left 90 degree
      //so after the for loop finish, the robot will face toward its original direction
       for(int i=0; i<4; i++){
              d=(d+1)%4;
              myRobot.turnleft(1);
              if(move())
                     dfs(myRobot, x+dirs[d][0], y+dirs[d][1], visited, d);
              else
                     visited.push(make_pair(x+dirs[d][0], y+dirs[d][1]));
              moveback();
       }
}
void moveback(){
       tureleft(2);
       move();
       turnleft(2);
}
```

1.You are given the following helper function: doubleRandBetween(double min, double max);
randBetween(1.3, 5.0) = 4.3
randBetween(1.3, 5.0) = 3.1
randBetween API class Rectangle {
public:
	Rectangle(int x1, int y1, int x2, int y2): x1(x1), y1(y1),x2(x2), y2(y2) {}
	double getArea() {
		return sqrt(pow((x2 - x1), 2) + pow((y2 - y1), 2));
	}
	pair<double, double> getRandomPoint() {
		pair<double, double> res = {randBetween(x1, x2), randBetween(y1, y2)}; return res;
private:
int x1, y1, x2, y2; };

Rectangle is (x1, y1) left bottom (x2, y2) right top
x = doubleRandBetween(x1, x2);
y = doubleRandBetween(y1, y2);

2.You're given N rectangles. Generate a point that is uniformly random from the set of given rectangles.矩形大小不一样，选中概率也不一样
randomPoint([recA, rec B, rec C]) = {1, 1} point in rectangle C
randomPoint([recA, rec B, rec C]) = {1, 4} point in rectangle A
randomPoint([recA, rec B, rec C]) = {2, 4} point in rectangle A
randomPoint([recA, rec B, rec C]) = {3, 3} point in rectangle B

```
pair<double, double> randomPont(vector<Rectangle>& list) {
	double total = 0.0;
	int n = list.size();
	vector<double> randomRec(n, 0.0);
	for(int i = 0; i < n; ++i) {
		total += list[i].getArea();
		randomRec[i] = total;
	}
	double target = randBetween(0, total);
	auto it = upper_bound(randomRec.begin(), randomRec.end(), target);
	int rec = it - random.begin();
	return randomRec[rec].getRandomPoint();
}

class point{
	int x, y;
	point(int a, int b){
		x=a;
		y=b;
	}
};

class Rectangle{
	int x1, x2, y1, y2;
};

point randomSelect1(rectangle r){
	return point(r.x1+rand()%(r.x2-r.x1+1), r.y1+rand()%(r.y2-r.y1+1));
}

point randomSelect2(vector<rectangle> rs){
	int selected=-1;
	int total=0;
	for(int i=0; i<rs.size(); i++){
		int area=(r[i].x2-r[i].x1)*(r[i].y2-r[i].y1);
		if(rand()%(total+area) >= total)
			seleted=I;
		total+=area;
	}
	return randomSelect1(rs[selected]);
}
```

随机生成矩形内的点

```
class Point{
       int x, y;
       Point(int a, int b):x(a), y(b){}
};

class Rectangle{
       int x1, x2, y1, y2;//top left and bottom right
};

Point randomSelect(Rectangle r){
       Int a=r.x1+rand()%(r.x2-r.x1+1);
       Int b=r.y1+rand()%(r.y2-r.y1+1);
       Return new Point(a, b);
}

Point weightedSelect(vector<Rectangle> rs){
       Int selected=-1;
       Int total=0;
       For(int i=0; i<rs.size(); i++){
              Int area=(rs[i].x2-rs[i].x1)*(rs[i].y1-rs[i].y2);
              If(rand()%(total+area)>=total){
                     Selected=I;
              }
              total+=area;
       }
       return randomSelect(rs[selected]);
}
```

Given an integer r, find how many integer points are located in a circle with radius r

```
class Solution {
    int count (int r) {
        if (r < 1)
            return 1;
        int res = 1 + 4 * r;
        for (int i = 1; i < r; i++) {
            res += 4 * (int) Math.sqrt (r * r - i * i);
        }
        return res;
    }
}
```

```
class randomCountry {
private:
       int sum;
       vector<int> arr;
       randomCountry(vector<int>& populations) {
              arr = vector<int>(populations.size() + 1, 0);
              sum = 0;
              for (int i = 0; i < populations.size(); ++i) {
                     sum+=populations[i];
                     arr[i + 1] = arr[i] + populations[i];
               }
       }

       int getRandomCountry() {
              int num = rand() % sum;
              auto ptr = upper_bound(arr.begin(), arr.end(), num);
              return ptr - arr.begin();
       }
```

随机从'a'-'z'中取字符组成长度为 len 的密码。时间复杂度。

```
string password_gen(int len,vetor<string>& SetList){
        string resStr="";
        vector<char> charlist;  //stores all the char I select
        for(int i=0;i< SetList.size();i++){
                int pick=rand()% SetList[i].size();
                charlist.push_back(SetList[i][pick]);
        }
```

```
        int rescount=len-(int)charlist;
        for(int i=0;i<rescount;i++){
                int pick1=rand()%(int)SetList.size();
                int pick2=rand()%(int)SetList[pick1].size();
                charlist.push_back(SetList[pick1][pick2]);
        }
        //shuffle
        while(!charlist.empty()){
                int pick=rand()%(int)charlist.size();
                resStr+=charlist[pick];
                swap(charlist[pick],charlist[(int)charlist.size()-1]);
                charlist.pop_back();
        }
        return resStr;
}
```

玛丽医院

blank 的个数 = 16 x 8 - n；

对每一个格子生成一个长度为 4 的 list，0，1，2，3，对应 blank，颜色 1，颜色 2，颜色 3.

生成 0 - list.size 的随机数 index。

如果生成的是颜色，则判断是否符合规则。如果不符合规则，删除掉对应 index 的 list 里的节点，继续进行随机。如果符合规则，生成下一个。

如果生成的是 0，则判断当前 blank 数是否等于 blank 数。如果等于，则删掉 list 中 index 等于 0 的节点，继续随机。如果小于 blank 数，说明符合规则，继续生成下一个。

根据这个关键字把字典里可能的单词全部 print 出来。 类似于输入法里输入几个字母然后弹出推荐的可能你需要的单词。

```
Trie
struct TrieNode {
     bool isWord;
     TrieNode* children[26];
     TrieNode() {
         for(int i = 0; i < 26; i++) {
             children[i] = nullptr;
         }
         isWord = false;
     }
};

private:
    TrieNode *root;
public:
    void addWord(const string &word) {
        TrieNode *curr = root;
        for(auto c : word) {
            int i = c - 'a';
            if(curr->children[i] == nullptr) {
                curr->children[i] = new TrieNode();
         }
            curr = curr->children[i];
        }
        curr->isWord = true;
    }
    TrieNode* getPrefix(const string &prefix) {
        TrieNode *curr = root;
        for(auto c : prefix) {
            int i = c - 'a';
            if(curr->children[i] == nullptr) {
                return nullptr;
          }
            curr = curr->children[i];
        }
        return curr;
    }
    vector<string> solver(vector<string> &dictionary, string prefix) {
```

```
        root = new TrieNode();
        vector<string> res;
        for(auto word : dictionary) {
            addWord(word);
        }
        TrieNode *withPrefix = getPrefix(prefix);
        dfs(withPrefix, res, prefix);
        return res;
    }

    void dfs(TrieNode *root, vector<string> &res, string &path) {
        if(root == nullptr) return;
        if(root->isWord == true) {
            res.push_back(path);
            return;
        }
       for(int i = 0; i < 26; i++) {
           if(root->children[i] != nullptr) {
               path += char(i + 'a');
               dfs(root->children[i], res, path);
               path.pop_back();
           }
       }
    }
```

假设有一条高速公路，路面上有 n 辆车，每辆车有不同的整 数速度，但是都在 1-n 范围内。现在给你一个数组，代表每辆车的速度。车辆出发顺序即数组顺序，问最后可以形成几个集群， 每个集群的 size 是多少？ 可以理解为，虽然车辆速度不同，但是即使后面的车比前面的车速度快，因为不能超车，最后肯定只 能以前车的速度行驶，这就形成了一个集群。比如[2,4,1,3]，最后[2,4]是一个集群，[1,3]是一个集群

Follow up 是现在假设想再加入一辆车，这个车的速度比其他车都大，但是不确定这个车的出发顺序，让输出最后所有可能的每 个集群的大小(List of List)。要求是可以调整并调用之前的函数，但是只能调用一次。

cluster of trains

```
vector<int> getCluster(vector<int> &speeds) {
    vector<int> res;
    int curr;
    for(int i = 0; i < speeds.size(); i++) {
        if(i == 0 || speeds[i] > speeds[i - 1]) {
            curr++;
    } else {
            res.push_back(curr);
            curr = 1;
    }
}
    if(curr != 0) res.push_back(curr);
    return res;
}
```

```
vector<vector<int>> getCluster2(vector<int> &speeds) {
    vector<int> noHighSpeed = getCluster(speeds);
    vector<vector<int>> res;
    noHighSpeed.insert(noHighSpeed.begin(), 1);
    res.push_back(noHighSpeed);
    noHighSpeed.erase(noHighSpeed.begin());
    for(int i = 0; i < noHighSpeed.size(); i++) {
        noHighSpeed[i]++;
        res.push_back(noHighSpeed);
        noHighSpeed[i]--;
    }
       return res;
}
```

Find the longest substring with k unique characters in a given string
Longest Substring with At Most Two Distinct Characters 最多有两个不同字符的最长子串
Longest Substring with At Most K Distinct Characters 最多有 K 个不同字符的最长子串

```
int lengthOfLongestSubstringKDistinct(string s, int k) {
        int res = 0, left = 0;
        unordered_map<char, int> m;
        for (int i = 0; i < s.size(); ++i) {
            ++m[s[i]];
            while (m.size() > k) {
                if (--m[s[left]] == 0) m.erase(s[left]);
                ++left;
            }
            res = max(res, i - left + 1);
        }
        return res;
}
```

Longest Substring with At Least K Repeating Characters 至少有 K 个重复字符的最长子字符串

```
int longestSubstring(string s, int k) {
        if(s.size()==0 || k>s.size()) return 0;
        if(k==0) return s.size();

        map<char, int> freq;
        for(char c:s){
            freq[c]++;
        }
        int idx=0;
        while(idx<s.size() && freq[s[idx]]>=k) idx++;
        if(idx==s.size()) return s.size();

        int left=longestSubstring(s.substr(0, idx), k);
        int right=longestSubstring(s.substr(idx+1, s.size()-1), k);
        return max(left, right);
}
```

给你一个列表，每个元素都是指向某个结点的指针。每个结点都有左右两个相邻结点。所以有些结点是连着的。如果两个指针指向的结点之间的所有结点都被输入列表中的某个指针指向，那么我们就说两个指针在同一个连通区块中。求连通区块的个数。
https://www.careercup.com/question?id=18880663

```
int numBlocks(List<ListNode> list) {
        int count = 0;
        Set<ListNode> set0 = new HashSet<>(list);
        while (!set0.isEmpty()) {
            count++;
            ListNode currNode = set0.iterator().next();
            ListNode tempNode = currNode;
            while (set0.contains(tempNode.pre)) {
                set0.remove(tempNode.pre);
                tempNode = tempNode.pre;
            }
            tempNode = currNode;
            while (set0.contains(tempNode.after)) {
                set0.remove(tempNode.after);
                tempNode = tempNode.after;
            }
            set0.remove(currNode);
        }
        return count;
    }
```

```
int number_of_contiguous_blocks(unordered_set<node*> nodes, node* head)
{
    unordered_map<node*, bool> seen;
    int count = 0;
    for(node* node: nodes) {
        seen.insert(make_pair(node, false));
        if(seen.find(node->next) == seen.end()){
            count++;
            seen[node]=true;
        }
    }
    for(node* node:nodes){
       if( seen[node] && seen.find(node->next) != seen.end() )
          count--;
    }
    return count;
}
int connected(vector<ListNode*> ptr) {
        unordered_set<ListNode*> s;
        int cnt = 0;
        for(int i = 0; i < ptr.size(); ++i) {
            s.insert(ptr[i]);
        }
        for(auto it = s.begin(); it != s.end(); ++it) {
            if(s.count(it->prev) == 0) cnt++;
        }
}
Employee 关系
class Employee {
vector<int> id, manager, sz;
public:
       Employee(int N) {
           manager = vector<int>(N, -1);
            sz = vector<int>(N, 1);
            id = vector<int>(N, 0);
            for(int i = 0; i < N; ++i) id[i] = i;
        }
        void set_peer(int p, int q) {
            int pid = find(p);
            int qid = find(q);
            int mgr = max(manager[pid], manager[qid]);
            if(sz[pid] < sz[qid]) {
                id[pid] = qid, sz[qid] += sz[pid];
                manager[qid] = mgr;
            } else {
                id[qid] = pid, sz[pid] += sz[qid];
                manager[pid] = mgr;
        }

        void set_manager(int p, int q) {
            int qid = find(q);
            manager[qid] = p;
        }

        bool query_manager(int p, int q) {
            while(1) {
                int qid = find(q);
                if(manager[qid] == p) return true;
                if(manager[qid] == -1) return false;
                q=manager[qid];
            }
            return false;
        }
        int find(int p) {
            while(p != id[p]) {
                id[p] = id[id[p]];
                p = id[p];
            }
            return p;
        }}
```

int steps(int n)， 返回 n 到 1 需要多少步，如果 n 是奇数 n = 3n + 1，偶数是 n = n/2，我先用 iterative 的方法写的。然后再让写一个 int maxSteps(int N)，输出 1<=n<=N 这个区间内最大的 steps(n)，很容易看出 redundancy，我就说用 Map 存一下，这个 map 是 n -> steps(n)，结果 implement 的时候卡住了，经姐姐提醒，把 steps(n)的方法改成了 recursion，这个 map 是要 maxSteps 和 steps 共用的。因为想了一下 map 放在哪里还有迭代改递归，然后一被 challenge 就紧张，所以写的慢，感觉面的不太好。.

```
int maxSteps(int N) {
        if(N<=1) return 0;
        int max = 1;
        HashMap<Integer, Integer> ht = new HashMap<>();
        ht.put(1, 0);
        for(int i=2;i<=N;++i) {
            if(ht.containsKey(i)) continue;
            max = Math.max(max, maxSteps(i, ht));
        }
        return max;
    }

int maxSteps(int val, HashMap<Integer, Integer> ht) {
        if(ht.containsKey(val)) return ht.get(val);
        int steps = 1 + maxSteps(val%2==0 ? val/2 : val*3+1, ht);
        ht.put(val, steps);
        return steps;
}
```

isSimilar Tree
两个树判断是否相同，或者在 swapping 意义下相同。类似利口妖零零，不过交换左右子树之后相同也满足条件

```
bool isSameTree(TreeNode* p, TreeNode* q) {
        if(p==NULL || q==NULL) {
            if(p==NULL && q==NULL){
                return true;
            }
            return false;
        }
        return (p->val == q->val && isSameTree(p->right,q->right) &&
                    isSameTree(p->left,q->left));
```

半个药片 probability

```
public double getStateProbability(int totalPills, int targetWhole, int targetHalf) {
    HashMap<List<Integer>, Double> states = new HashMap<>();
    Double prob = dfs(totalPills, 0, targetWhole, targetHalf, states);
    return prob;
}

public double dfs(int wholePills, int halfPills, int targetWhole, int targetHalf,
HashMap<List<Integer>, Double> states) {
    List<Integer> state = new ArrayList<>();
    state.add(wholePills);
    state.add(halfPills);
    if (states.containsKey(state)){
        return states.get(state);
    }
    double prob = 0;

    // some optimizations
    if (wholePills < targetWhole || wholePills + halfPills < targetHalf) {
        return 0;
    }
    if (wholePills == targetWhole && halfPills == targetHalf) {
        prob = 1;
    } else if (wholePills == 0) {
        prob = dfs(wholePills, halfPills - 1, targetWhole, targetHalf, states);
    } else if (halfPills == 0) {
        prob = dfs(wholePills - 1, halfPills + 1, targetWhole, targetHalf, states);
    } else {
```

```
        double wholeProb = (wholePills + 0.0) / (wholePills + halfPills);
        prob = wholeProb * dfs(wholePills - 1, halfPills + 1, targetWhole, targetHalf, states) + (1
- wholeProb) * dfs(wholePills, halfPills - 1, targetWhole, targetHalf, states);
    }
    states.put(state, prob);
    return prob;
}


struct node{
    int w, h;
    double p;
};

// runtime O(w^2)?
double prob(int w, int h, int wt, int ht){//wt:whole target
    queue<node> q;
    q.push({w, h, 1.0});
    double ans = 0.0;
    while(!q.empty()) {
        w = q.front().w;
        h = q.front().h;
        double p = q.front().p;
        q.pop();
        if(w == wt && h == ht) {
            ans += p;
        }
        else {
            if(w > wt) q.push({w-1, h+1, p*(double)w/(w+h)});
            if(h > 0) q.push({w, h-1, p*(double)h/(w+h)});
        }
    }
    return ans;
}
```

房间找宝藏

```
struct Room {
    int id;
    int keyToOpen;  // -1 for no need key
    vector<Room*> neighbors;
    bool hasTreasure;
    int hasKey;
};

class Solution {
    unordered_set<int> visited;
    unordered_set<int> keyHave;
    queue<Room*> needKey;
    bool foundTreasure;

    bool canFindTreasure(Room *room) {
        dfs(room);
        while(!needKey.empty()) {
             if(keyHave.find(needKey.front()->keyToOpen) != keyHave.end()) {
                 dfs(needKey.front());
                 needKey.pop();
             }
         }
       return foundTreasure;
     }

    void dfs(Room *room) {
           if(visited.find(room->id) != visited.end()) {
                return;
           }
           if(room->keyToOpen != -1 && keyHave.find(room->keyToOpen) == keyHave.end()) {
                needKey.push(room);
                return;
```

```
            }
            if(room->hasTreasure == true) {
                   foundTreasure = true;
                   return;
            }
            if(room->hasKey != -1) {
                   keyHave.insert(room->hasKey);
            }
            for(auto neighbor : room->neighbors) {
                   dfs(neighbor);
            }
        }
```

candy crush

```
    vector<vector<int>> candyCrush(vector<vector<int>>& board) {
        int m = board.size(), n = board[0].size();
        while (true) {
            vector<pair<int, int>> del;
            for (int i = 0; i < m; ++i) {
                for (int j = 0; j < n; ++j) {
                    if (board[i][j] == 0) continue;
                    int x0 = i, x1 = i, y0 = j, y1 = j;
                    while (x0 >= 0 && x0 > i - 3 && board[x0][j] == board[i][j]) --x0;
                    while (x1 < m && x1 < i + 3 && board[x1][j] == board[i][j]) ++x1;
                    while (y0 >= 0 && y0 > j - 3 && board[i][y0] == board[i][j]) --y0;
                    while (y1 < n && y1 < j + 3 && board[i][y1] == board[i][j]) ++y1;
                    if (x1 - x0 > 3 || y1 - y0 > 3) del.push_back({i, j});
                }
            }
            if (del.empty()) break;
            for (auto a : del) board[a.first][a.second] = 0;
            for (int j = 0; j < n; ++j) {
                int t = m - 1;
                for (int i = m - 1; i >= 0; --i) {
                    if (board[i][j]) swap(board[t--][j], board[i][j]);
                }
            }
        }
        return board;
    }
```

贪吃蛇

```
class SnakeGame {
private:
    int height, width, score;
    vector<pair<int, int>> pos, food;
public:
Void GenerateFood(int n){
       for (int i=0; i<n; )
      {
         int x = random() % width;
         int y = random() % height;
         food.push_back(make_pair(x, y));
         i++;
       }
}

SnakeGame(int width, int height, vector<pair<int, int>> food) {
        this->height=height;
        this->width=width;
        score=0;
        this->food=food;
        pos.push_back({0, 0});
    }

//移动的结果就是，蛇头变到新位置，去掉蛇尾的位置即可。需要注意的是去掉蛇尾的位置是在检测和蛇身的碰撞之前
       pair<int, int> head=pos.front();
```

```
        pair<int, int> tail=pos.back();
        pos.pop_back();
        if(direction=="U"){
            head.first--;
        }
        else if(direction=="D"){
            head.first++;
        }
        else if(direction=="L"){
            head.second--;
        }
        else if(direction=="R"){
            head.second++;
        }
        if(head.first<0 || head.first>=height || head.second<0 || head.second>=width
||count(pos.begin(), pos.end(), head)){
            return -1;
        }
        //更新 head pos, 之前的 head 不需处理，转变成身体
        pos.insert(pos.begin(), head);
        if(!food.empty() && head==food.front()){
            food.erase(food.begin());//erase(position)
            pos.push_back(tail);
            score++;
        }
        return score;
    }
};

void generateFood() {
    int x = 0;
    int y = 0;
    do {
        // Generate random x and y values within the map
        x = rand() % (mapwidth - 2) + 1;
        y = rand() % (mapheight - 2) + 1;

        // If location is not free try again
    } while (map[x + y * mapwidth] != 0);

    // Place new food
    map[x + y * mapwidth] = -2;
}

[1,2,3,4] => [1,3,2,4].
void wiggleSort(vector<int>& nums) {
        for(int i=0; i<(int)nums.size()-1; i++){
            if(i%2==0 && nums[i]>nums[i+1]){
                swap(nums[i], nums[i+1]);
            }
            if(i%2==1 && nums[i]<nums[i+1]){
                swap(nums[i], nums[i+1]);
            }
        }
    }
```

扫雷

```
vector<vector<char>> updateBoard(vector<vector<char>>& board, vector<int>& click) {
        if(board[click[0]][click[1]] == 'M'){
            board[click[0]][click[1]] = 'X';
            return board;
        }
        dfs(board, click[0], click[1]);
        return board;
    }

    bool inbound(vector<vector<char>> &board, int x, int y){
        return (x>=0 && x<board.size() && y>=0 && y<board[0].size());
    }

    void dfs(vector<vector<char>> &board, int x, int y){
        if(!inbound(board, x, y)) return;
        if(board[x][y] == 'E'){
            int count=0;
            if(inbound(board, x-1, y-1) && board[x-1][y-1]=='M') count++;
            if(inbound(board, x-1, y  ) && board[x-1][y  ]=='M')  count++;
            if(inbound(board, x-1, y+1) && board[x-1][y+1]=='M') count++;
            if(inbound(board, x, y-1  ) && board[x  ][y-1]=='M') count++;
            if(inbound(board, x, y    ) && board[x  ][y  ]=='M') count++;
            if(inbound(board, x, y+1  ) && board[x  ][y+1]=='M') count++;
            if(inbound(board, x+1, y-1) && board[x+1][y-1]=='M') count++;
            if(inbound(board, x+1, y  ) && board[x+1][y  ]=='M') count++;
            if(inbound(board, x+1, y+1) && board[x+1][y+1]=='M') count++;

            if(count>0){
                board[x][y]='0'+count;
            }
            else{
                board[x][y]='B';
                dfs(board, x-1, y-1);
                dfs(board, x-1, y);
                dfs(board, x-1, y+1);
                dfs(board, x, y-1);
                dfs(board, x, y);
                dfs(board, x, y+1);
                dfs(board, x+1, y-1);
                dfs(board, x+1, y);
                dfs(board, x+1, y+1);
            }
        }
    }

// A Function to place the mines randomly on the board
void placeMines(int mines[][2], char realBoard[][MAXSIDE])
{
    bool mark[MAXSIDE*MAXSIDE];

    memset (mark, false, sizeof (mark));

    // Continue until all random mines have been created.
    for (int i=0; i<MINES; )
     {
        int random = rand() % (SIDE*SIDE);
        int x = random / SIDE;
        int y = random % SIDE;

        if (mark[random] == false)
        {
            // Row Index of the Mine
            mines[i][0]= x;
            // Column Index of the Mine
            mines[i][1] = y;

            // Place the mine
            realBoard[mines[i][0]][mines[i][1]] = '*';
            mark[random] = true;
```

```
            i++;
        }
}

vector<int> shuffle() {
        vector<int> res = v;
        for (int i = 0; i < res.size(); ++i) {
            //rand()% res.size()
            int t = i + rand() % (res.size() - i);
            swap(res[i], res[t]);
        }
        return res;
    }
```

Bomb enermy

```
int maxKilledEnemies(vector<vector<char>>& grid) {

        int res=0;
        int row=grid.size();
        int col= row? grid[0].size():0;
        if(row==0) return 0;

        for(int i=0; i<row; i++){
            for(int j=0; j<col; j++){
                if(grid[i][j]=='0'){
                    int sum=0;
                    int left=j;
                    while(left>0&&grid[i][--left]!='W'){
                        if(grid[i][left]=='E')
                            sum++;
                    }

                    int right=j;
                    while(right<col-1&&grid[i][++right]!='W'){
                        if(grid[i][right]=='E')
                            sum++;
                    }

                    int top=i;
                    while(top>0&&grid[--top][j]!='W'){
                        if(grid[top][j]=='E')
                            sum++;
                    }

                    int bottom=i;
                    while(bottom<row-1&&grid[++bottom][j]!='W'){
                        if(grid[bottom][j]=='E')
                            sum++;
                    }
                    res=max(res, sum);
                }
            }
        }
        return res;
    }
```

```
class ZigzagIterator {
private:
    queue<pair<vector<int>::iterator, vector<int>::iterator>> q;

public:
    ZigzagIterator(vector<int>& v1, vector<int>& v2) {
        if(v1.size()) q.push(make_pair(v1.begin(), v1.end()));
        if(v2.size()) q.push(make_pair(v2.begin(), v2.end()));
    }

    int next() {
        auto p=q.front();
        q.pop();
        int val=*p.first;
        p.first++;
        if(p.first!=p.second) q.push(p);
        return val;
    }

    bool hasNext() {
        return q.size();
    }
};

BST iterator—next smallest number
private:
    TreeNode *current = NULL;
    stack<TreeNode*> s;
public:
    BSTIterator(TreeNode *root) {
       current = root;
    }
    bool hasNext() {
        while(current){
            s.push(current);
            current = current->left;
        }
        if(s.empty()){
            return false;
        }
        return true;
    }
    int next() {
        TreeNode* node = s.top();
        s.pop();
        current = node->right;
        return node->val;
}

//优化 morris
class BSTIterator {
public:
       TreeNode* curr;
BSTIterator(TreeNode *root) {
       curr = root; }
       bool hasNext() {
           if(curr != nullptr) return true;
           return false;
       }
int next() {
    while(curr != nullptr) {
        if(curr->left == nullptr) {
            int res = curr->val;
            curr = curr->right;
            return res;
        }
        TreeNode *prev = curr->left;
        while(prev->right != nullptr && prev->right != curr) {
                prev = prev->right;
```

```
            }
            if(prev->right == nullptr) {
                prev->right = curr;
                curr = curr->left;
            }
            else {
                int res;
                prev->right = nullptr;
                res = curr->val;
                curr = curr->right;
                return res;
             }
         }
         return -1; }
};


BST iterator – next largest element
Treenode * minValue(Treenode * node) {
    Treenode * current = node;
    while (current->left != NULL) {
      current = current->left;
    }
    return current;
}

Treenode * inOrderSuccessor(Treenode *root, Treenode *n)
{
    if( n->right != NULL )
        return minValue(n->right);

    Treenode *succ = NULL;
    while (root != NULL)
    {
        if (n->data < root->data)
        {
            succ = root;
            root = root->left;
        }
        else if (n->data > root->data)
            root = root->right;
        else
           break;
    }
    return succ;
}

TreeNode* inorderSuccessor(TreeNode* root, TreeNode* p) {
        stack<TreeNode*> s;
        bool b = false;
        TreeNode *t = root;
        while (t || !s.empty()) {
            while (t) {
                s.push(t);
                t = t->left;
            }
            t = s.top(); s.pop();
            if (b) return t;
            if (t == p) b = true;
            t = t->right;
        }
        return NULL;
}
```

2D vector flatten

```
private:
	vector<vector<int>>::iterator x, e;
	vector<int>::iterator y;
public:
	vector2D(vector<vector<int>>&vec2d){
		x=vec2d.begin();
		e=vec2d.end();
		if(x!=e){
			y=x->begin();
		}
	}
	int next(){
		return *y++;
	}
	bool hasNext(){
		while(x!=e && y==x->end()){
			++x;
			y=x->begin();
		}
		return x!=e;
	}
```

reverse list in K-group

```
ListNode* reverseKGroup(ListNode* head, int k) {
        if(head==NULL ||k==1) return head;
        ListNode *dummy=new ListNode(0);
        dummy->next=head;
        ListNode *pre=dummy;
        ListNode *cur=dummy;
        int num=0;
        while(cur->next){
            cur=cur->next;
            num++;
        }
        while(num>=k){
            cur=pre->next;
            for(int i=1; i<k; i++){
                ListNode *tmp=cur->next;
                cur->next=tmp->next;
                tmp->next=pre->next;
                pre->next=tmp;
            }
            pre=cur;
            num-=k;
        }
        return dummy->next;
    }
```

Delete duplicate

```
ListNode* deleteDuplicates(ListNode* head) {
        if(head==NULL) return NULL;
        ListNode *p=new ListNode(0);
        p->next=head;
        ListNode *pre=p;
        ListNode *cur=head;
        while(cur!=NULL){
            while(cur->next!=NULL && cur->val==cur->next->val){
                cur=cur->next;
            }
            if(pre->next==cur){
                pre=pre->next;
            }
            else{
                pre->next=cur->next;
            }
            cur=cur->next;
        }
        return p->next;
    }
```

```
Remove Nth node from end of the list
ListNode* removeNthFromEnd(ListNode* head, int n) {
        if(head==NULL) return NULL;
        ListNode* p;
        p->next = head;
        ListNode *slow=p;
        ListNode *fast=p;
        for(int i=0;i<n;i++){
            fast=fast->next;
        }
       //maintain the n gap
        while(fast->next){
            slow=slow->next;
            fast=fast->next;
        }
        ListNode *d=slow->next;
        slow->next = slow->next->next;
        delete d;

        return p->next;
}
```

BST longest 连续数列

```
//父到子 only
int longestConsecutive(TreeNode* root) {
        if (!root) return 0;
        int res = 0;
        dfs(root, 1, res);
        return res;
    }

void dfs(TreeNode *root, int len, int &res) {
        res = max(res, len);
        if (root->left) {
            if (root->left->val == root->val + 1)
                 dfs(root->left, len + 1, res);
            else dfs(root->left, 1, res);
        }
        if (root->right) {
            if (root->right->val == root->val + 1)
                 dfs(root->right, len + 1, res);
            else dfs(root->right, 1, res);
        }
}

//子到父
int longestConsecutive(TreeNode* root) {
        if (!root) return 0;
        int res = helper(root, 1) + helper(root, -1) + 1;
        return max(res, max(longestConsecutive(root->left), longestConsecutive(root->right)));
}

 int helper(TreeNode* node, int diff) {
        if (!node) return 0;
        int left = 0, right = 0;
        if (node->left && node->val - node->left->val == diff) {
            left = 1 + helper(node->left, diff);
        }
        if (node->right && node->val - node->right->val == diff) {
            right = 1 + helper(node->right, diff);
        }
        return max(left, right);
}
```

BST closest value

```
int closestValue(TreeNode* root, double target) {
        int res = root->val;
        while(root){
            if((double)root->val == target){
                return root->val;
            }
            if(abs(root->val - target) < abs(res - target)){
                res = root->val;
            }
            if(root->val > target){
                root = root->left;
            }
            else{
                root = root->right;
            }

        }
        return res;
    }
```

Buy and sell Stock

//最多一次交易

```
int maxProfit(vector<int>& prices) {
        int minprice = INT_MAX;
        int maxprofit = 0;
        //int res;
        for(int i=0; i<prices.size(); i++){
            minprice = min(minprice, prices[i]);
            maxprofit = max(maxprofit, prices[i]-minprice);
        }
        return maxprofit;
}
```

//最多两次交易

```
int maxProfit(vector<int>& prices) {
//定义 local[i][j]为在到达第 i 天时最多可进行 j 次交易并且最后一次交易在最后一天卖出的最大利润，此为局部最优。
//定义 global[i][j]为在到达第 i 天时最多可进行 j 次交易的最大利润，此为全局最优。
//local[i][j] = max(global[i - 1][j - 1] + max(diff, 0), local[i - 1][j] + diff)
//global[i][j] = max(local[i][j], global[i - 1][j])
        int n=prices.size();
        if(n==0) return 0;
        vector<vector<int>> l(n, vector<int>(3, 0)), g(n, vector<int>(3,0));
        for(int i=1; i<n; i++){
            int diff=prices[i]-prices[i-1];
            for(int j=1; j<=2; j++){
                l[i][j]=max(g[i-1][j-1]+max(diff, 0), l[i-1][j]+diff);
                g[i][j]=max(l[i][j], g[i-1][j]);
            }
        }
        return g[n-1][2];
}
```

//最多 K 次交易

```
int maxProfit(int k, vector<int>& prices) {
        int n=prices.size();
        if(n==0) return 0;
        if(k>=n) return MP2(prices);
        vector<vector<int>> g(n+1, vector<int>(k+1, 0)), l(n+1, vector<int>(k+1, 0));
        for(int i=1; i<=n; i++){
            int diff=prices[i]-prices[i-1];
            for(int j=k; j>=1; j--){
                l[i][j]=max(g[i-1][j-1]+max(diff, 0), l[i-1][j]+diff);
                g[i][j]=max(l[i][j], g[i-1][j]);
            }
        }
        return g[n-1][k];
}
int MP2(vector<int> &prices){
        int res=0;
        for(int i=0; i<prices.size()-1; i++){
```

```
            if(prices[i+1]>prices[i]){
                res+=prices[i+1]-prices[i];
            }
        }
        return res;
    }
```

//无限交易

```
int maxProfit(vector<int>& prices) {
        int maxprofit=0;
        int n=prices.size();
        for(int i=0; i<n-1;i++){
            if(prices[i]<prices[i+1]){
                maxprofit += prices[i+1] - prices[i];
            }
        }
        return maxprofit;
}
//with cool down
//s0[i] = max(s0[i - 1], s2[i - 1]); // Stay at s0, or rest from s2
//s1[i] = max(s1[i - 1], s0[i - 1] - prices[i]); // Stay at s1, or buy from s0
//s2[i] = s1[i - 1] + prices[i]; // Only one way from s1
int maxProfit(vector<int>& prices) {
        int n=prices.size();
        if(n<=1) return 0;

        vector<int> s0(n,0);
        vector<int> s1(n,0);
        vector<int> s2(n,0);
         s0[0]=0;
        s1[0]=-prices[0];
        s2[0]=INT_MIN;
        for(int i=1; i<n;i++){
            s0[i]=max(s0[i-1], s2[i-1]);
            s1[i]=max(s1[i-1], s0[i-1]-prices[i]);
            s2[i]=s1[i-1]+prices[i];
        }
        return max(s0[n-1], s2[n-1]);
    }

Longest Subarray having sum of elements at most k
int atMostSum(int arr[], int n, int k)
{
    int sum = 0;
    int cnt = 0, maxcnt = 0;
    for (int i = 0; i < n; i++) {
        if ((sum + arr[i]) <= k) {
            sum += arr[i];
            cnt++;
        }
        // Else, remove first element of current
        // window and add the current element
        else if(sum!=0)
        {
            sum = sum - arr[i - cnt] + arr[i];
        }
         maxcnt = max(cnt, maxcnt);
    }
    return maxcnt;
}
```

问我 C++里面 move 和 copy 的差异。我愣了一下，没懂是问我什么，再三确认他到底想问什么。他给了 3 个构造函数分别为 `A(){};` `A(const A&){}; A(A&&){}`

我回答：一个构造函数是调用成员的默认构造函数去初始化，第二个构造函数是拷贝构造函数，但是 C++程序员通常会 delete it，防止有指针拷贝下的浅拷贝发生，第三个右值引用我说我不只是很了解。

move 语义转移的 move 构造函数中指定的堆空间使用权，而不是对象，也就是说对于简单对象，比如 int，即使对它使用 move 语义也是没用的。

问题中提到的顺序容器 vector，使用 move 用处不大，因为最终它还是要使用拷贝语义来保证容器的内存顺序

```
bool removeAPI()
vector<TreeNode*> removeTreeNode(TreeNode *root){
       TreeNode *newRoot=new TreeNode(-1);
       newRoot->left=root;
       dfs(newRoot);
       if(newRoot->left!=NULL){
              res.push_back(newRoot->left);
       }
       return res;
}
bool dfs(TreeNode *root, vector<TreeNode*> &res){
       if(root==NULL) return false;
       bool removeLeft=dfs(root->left, res);
       bool removeRight=dfs(root->right, res);
       if(removeLeft)root->left=NULL;
       if(removeRight) root->right=NULL;
       if(removeAPI(root)){
              if(root->left!=NULL) res.push_back(root->left);
              if(root->right!=NULL) res.push_back(root->right);
              return true;
       }
       return false;
}

vector<Node*> result;
vector<Node*> deleteSomeNodes(Node *root){
    helper(NULL, root);
    return result;
}
void helper(Node* prarent, Node * cur){
    if(!cur) return;
    Node * l = cur->left, * r = cur->right;
    if(shouldErase(cur)){ // if delete cur, cut off its left and right children
       cur->left = NULL, cur->right = NULL;
    }
    else if(shouldErase(parent)){//if cur is reserved and its parent is deleted, cur will be a new root
       result.push_back(cur);
  }
  helper(cur, l);
  helper(cur, r);
}

class Solution {
public:
    vector<TreeNode*> deleteNode(TreeNode *root, unordered_set<TreeNode*> deleted) {
        queue<TreeNode*> s;
        s.push(root);
        vector<TreeNode*> res;
        while(!s.empty()) {
            TreeNode *top = s.front();
            s.pop();
            if(top == nullptr) continue;
            helper(top, deleted, nullptr, s, true);
            res.push_back(top);
         }
         return res;
}
void helper(TreeNode* root, unordered_set<TreeNode*> &deleted, TreeNode *father, queue<TreeNode*>
&s, bool left) {
        if(root == nullptr) return;
        if(deleted.find(root) != deleted.end()) {
            if(father != nullptr) {
                if(left) father->left = nullptr;
                else father->right = nullptr;
            }
            s.push(root->left);
            s.push(root->right);
        }
        helper(root->left, deleted, root, s, true);
```

```
        helper(root->right, deleted, root, s, false);
    }
};
```

找金矿
http://www.1point3acres.com/bbs/thread-376373-2-1.html

```
shuffle array(Fisher-Yates shuffle Algorithm O(n))
for i from n - 1 downto 1 do
       j = random integer with 0 <= j <= i
       exchange a[j] and a[i]

void randomize(vector<int> A){
       for(int i=A.size()-1; i>0; i--){
              int j=rand()%(i+1);
              swap(A[i], A[j]);
       }
}


First Unique char in a stream
//use queue, double linked list
https://www.geeksforgeeks.org/find-first-non-repeating-character-stream-characters/
void firstnonrepeating(char str[])
{
    queue<char> q;
    int charCount[CHAR_MAX] = { 0 };
    for (int i = 0; str[i]; i++) {
        q.push(str[i]);
        charCount[str[i]-'a']++;
        while (!q.empty()){
            if (charCount[q.front()-'a'] > 1)
                q.pop();
            else{
                cout << q.front() << " ";
                break;
            }
        }
        if (q.empty())
            cout << -1 << " ";
    }
}
```

多叉树

```
struct TreeNode {
    vector<TreeNode*> children;
    int val;
};
int deep(TreeNode *root) {
    if(root == nullptr) return 0;
    queue<TreeNode*> q;
    q.push(root);
    int deep = 0;
    while(!q.empty()) {
        deep++;
        int size = q.size();
        for(int i = 0; i < size; i++) {
           TreeNode *front = q.front();
           q.pop();
           for(auto child : front->children) {
              q.push(child);
           }
        }
     }
     return deep;
}
```

```
class Solution {
    unordered_map<TreeNode*, int> happy_index;
    unordered_map<TreeNode*, int> relation(TreeNode *root) {
        dfs(root);
        return happy_index;
    }
    pair<int, int> dfs(TreeNode *root) {
        if(root == nullptr) {
            return make_pair(0, 0);
        }
        if(root->children.size() == 0) {
            return make_pair(1, root->val);
        }
        int num = 0;
        int sum = 0;
        for(auto child : root->children) {
            pair<int,int> happy = dfs(child);
            num += happy.first;
            sum += happy.second;
        }
        happy_index[root] = num / sum;
        return make_pair(num, sum);
    }
};
```

```
find closest number in sorted array
int binarySearch(vector<int> nums, int l, int r, int target){
       if(target<=nums[0]) return nums[0];
       if(target>=nums.back()) return nums.back();
       while(l<r){
              int mid=(l+r)/2;
              if(target==nums[mid])
                     return nums[mid];
              if(target<nums[mid])
                     hi=mid;
              else
                     l=mid+1;
       }
       return nums[mid];
}


//follow up: infinite array
int findPos(vector<int> nums, int target){
       int l=0, h=l;
       int val=nums[0];
       while(val<target){
              l=h;
              h=2*h;
              val=nums[h];
       }
       return binarySearch(nums, l, h, target);
}
```

```
double linked list deletion
bool delete(target){
       ListNode *to_remove=head;
       While(to_remove && to_remove->val!=target){
              To_remove=to_remove->next;
       }

       if(to_remove){
              if(to_remove==head)
                     head=head->next;
              if(to_remove==tail)
                     tail=tail->prev;
              if(to_remove->next)
                     to_remove->next->prev = to_remove->prev;
              if(to_remove->prev)
                     to_remove->prev->next=to_remove->next;
              delete to_remove;
              count--;
              return true;
       }
       return false;
}

intertion
template <typename T> void dll<T>::insert(T value) {
    Node *node = new Node;
    node->data = value;
// Case 1: There are no nodes yet
    if (head == nullptr) {
        head = node;
        tail = head;
        return;
    }
// case 2 - inserting at the head of the list
    if (node->data < head->data)
    {
        node->next = head;
        head = node;
        return;
    }
// case 3 - inserting at the end of the list
    if (node->data >= tail->data)
    {
        node->prev = tail;
        tail->next = node;
        tail = node;
        return;
    }
// general case - inserting into the middle
    Node* probe = head;
    while (probe && (node->data >= probe->data))
    {
        probe = probe->next;
    }
    if (probe)
    {
        node->next = probe;
        node->prev = probe->prev;
        probe->prev->next = node;
        probe->prev = node;
        return;
    }
return;
}
```

snake ladder
https://blog.csdn.net/u013578420/article/details/77429337

21 点
```
def cal21(num):
    dp=[1.0]*100
    dp[:22] = [0.0]*22
    for i in range(16, num-1, -1):
      for j in range(10,0,-1):
        dp[i]+=dp[i+j]*0.1
    return dp[num]
print(cal21(14))
```

initial: 0 - 21: 0.0 22 - 100: 1.0
16: dp[16]= (26-22)*0.1=0.5
15: dp[15]= (25-22)*0.1+(16)*0.1=0.4+0.5*0.1=0.45
14: dp[14]= (24-22)*0.1+(16)*0.1+(15)*0.1=0.3+0.05+0.045=0.395

Maximum Vacation Days
```
int maxVacationDays(vector<vector<int>>& flights, vector<vector<int>>& days) {
        //j 是天数， i p 是城市, 从后往前
        int n=flights.size();
        int k=days[0].size();
        vector<vector<int>> dp(n, vector<int>(k,0));
        int res=0;
        for(int j=k-1; j>=0; j--){
            for(int i=0; i<n; i++){
                dp[i][j]=days[i][j];
                for(int p=0; p<n; p++){
                    //从倒数第二周开始算
                    if((i==p || flights[i][p]) && j<k-1){
                        dp[i][j]=max(dp[i][j], dp[p][j+1]+days[i][j]);
                    }
                    //最后要做的就是想上面所说在 dp[i][0]中找最大值
                    if(j==0 && (i==0||flights[0][i])){
                        res=max(res, dp[i][0]);
                    }
                }
            }
        }
        return res;
    }
```

给一个数组，n 的长度（n >=1），含有所有 1 - n 的数字，要求得到一个最小的 index，满足条件: 如果把数组小于等于 index 的部分排序好，并且把数组大于 index 的部分排序好，那么整个数组就排序好了。
```
int find(vector<int> nums) {
        if (nums.empty()) return 0;
        int nsum = 0, isum = 0;
        for (int i = 0; i < nums.size(); i++) {
                int iPlusOne = i + 1;
                isum += iPlusOne;
                nsum += nums[i];
                if (isum == nsum) return i;
        }
        return nums.size() - 1;
}
```

Find the length of the longest path in a binarytree(return number of edges)

```
int diameterOfBinaryTree(TreeNode* root) {
        if(root == NULL) return 0;
        int res = depth(root->left) + depth(root->right);
        return max(res, max(diameterOfBinaryTree(root->left), diameterOfBinaryTree(root->right)));

    }

int depth(TreeNode *root){
        if(root == NULL) return 0;
        return 1+max(depth(root->left), depth(root->right));
    }
```

sample k elements with equal probability

```
void selectKItems(int stream[], int n, int k)
{
    int i;  // index for elements in stream[]

    int reservoir[k];
    for (i = 0; i < k; i++)
        reservoir[i] = stream[i];

    // Use a different seed value so that we don't get
    // same result each time we run this program
    srand(time(NULL));

    // Iterate from the (k+1)th element to nth element
    for (; i < n; i++)
    {
        // Pick a random index from 0 to i.
        int j = rand() % (i+1);

        // If the randomly  picked index is smaller than k, then replace
        // the element present at the index with new element from stream
        if (j < k)
          reservoir[j] = stream[i];
    }
}
```

sum of coins

```
unordered_set<int> coinSum(int i, vector<int>& coins) {
        if(i == coins.size()) return unordered_set<int>{0};
        auto next = coinSum(i + 1, coins);
        unordered_set<int> res;
        for(int n: next) {
            int cur = n + coins[i];
            if(res.count(cur) == 0) res.insert(cur);
            cur = n;
            if(res.count(cur) == 0) res.insert(cur);
        }
       return res;
}

unordered_set<int> possible_sums(int i, vector<int>& coins, vector<int>& quantity) {
         if(i == int(coins.size())) return unordered_set<int>{0};
             auto remaining = possible_sums(i + 1, coins, quantity);
             unordered_set<int> ret_set;
             for(int q = 0; q <= quantity[i]; q++) {
                      for(int next_sum : remaining) {
                        int curr = next_sum + q * coins[i];
                        if(ret_set.find(curr) == ret_set.end()) ret_set.insert(curr);
                      }
              }
          return ret_set;
}

//backspace string
bool backspace2(string str, string pattern) {
    string res = "";
    for(char c: str) {
        if( c == '<'&& !str.empty()) res.pop_back();
        else res.push_back(c);
        }
       return res == pattern;
}
//O(1)
```

```
bool backspace3(string str, string pattern) {
        int i = str.size() - 1, j = pattern.size() - 1;
        int n = 0;
        for(; i >= 0; --i) {
            if(str[i] != '<' && n == 0){
                if(j >= 0){
                    if(str[i] == pattern[j]) --j;
                } else {
                    return false;
} }
            if(str[i] == '<'){
                ++n;
            } else if(str[i] != '<' && n != 0) {
                --n;
} }
        return j < 0;
}

//find single element
int a=0;
for(int n:nums){
        a^=n;
}
return a;

int findSingle(vector<int> nums) {
    int left = 0, right = nums.size() - 1;
    while(left + 1 < right) {
        int mid = (left + right) / 2;
        if(mid % 2 == 0) {
            if(nums[mid] == nums[mid - 1])  {//check
                right = mid;
             } else {
                left = mid; }
        } else {
            if(nums[mid] == nums[mid - 1]) {
                left = mid;
            } else {
                right = mid; }
    if(nums[left] != nums[left - 1] && nums[left] != nums[left + 1]) return nums[left];
    else return nums[right];
}

//find peek element
int findMax(vector<int> nums) {
        int left = 0, right = nums.size() - 1;

        while(left + 1 < right) {
            int mid = (left + right) / 2;
            if(nums[mid - 1] <= nums[mid] && nums[mid + 1] <= nums[mid]) {
                return nums[mid];
            }
            else if(nums[mid - 1] <= nums[mid] && nums[mid] <= nums[mid + 1]) {
                left = mid;
            } else {
                right = mid; }
        }
        if(nums[left] > nums[right]) return nums[left];
        else return nums[right];
 }


//list permutation
vector<string> permutationOfList(vector<string>& list) {
       vector<string> res;
       int n = list.size();
       if (n == 0 ) return res;
       vector<bool> visited(n, false);
       dfs("", 0, res, visited, list, n);
       return res;
}
void dfs(string temp, int len, vector<string>& res, vector<bool>& visited, vector<string>& list, int n) {
       if (len == n) {
               res.push_back(temp);
               return;
       } else {
               for (int i = 0; i < n ; ++i) {
```

```
                        if (!visited[i]) {
                                visited[i] = true;
                                dfs(temp + list[i], len + 1, res, visited, list, n);
                                visited[i] = false;
                        }
                }
        }
}

//find duplicate substree
vector<TreeNode*> findDuplicateSubtrees(TreeNode* root) {
        vector<TreeNode*> res;
        map<string, vector<TreeNode*>> m;
        serilize(root, m);
        for(auto it=m.begin(); it!=m.end(); it++){
            if(it->second.size()>1){
                res.push_back(it->second[0]);
            }
        }
        return res;
}

string serilize(TreeNode *root, map<string, vector<TreeNode*>> &m){
        if(root==NULL) return "";
        string s="("+ serilize(root->left, m) + to_string(root->val) + serilize(root->right, m) + ")";
        m[s].push_back(root);
        return s;
}

//path in a matrix via 1
int findPath(vector<vector<int>> matrix) {
    if(matrix.size() == 0 || matrix[0].size() == 0) {
       return true;
    }
    unordered_set<pair<int, int>> visited;
    queue<pair<pair<int, int>, int>> s;
    for(int i = 0; i < matrix[0].size(); i++) {
        if(matrix[0][i] == 1) s.push(make_pair(make_pair(0, i), 1));
    }
    while(!s.empty()) {
        pair<pair<int, int>, int> top_pair = s.front();
        s.pop();
        pair<int, int> top = top_pair.first;
        int count = top_pair.second;
        if(visited.find(top) != visited.end()) {
            continue;
        }
        visited.insert(top);
        int row = top.first, col = top.second;
        if(row == matrix.size() - 1) {
            return count;
        }
        if(col - 1 >= 0 && matrix[row][col - 1] == 1) {
            s.push(make_pair(make_pair(row, col - 1), count + 1));
        }
        if(col + 1 < matrix[0].size() &&
           matrix[row][col + 1] == 1) {
            s.push(make_pair(make_pair(row, col + 1), count + 1));
        }
        if(matrix[row + 1][col] == 1) {
            s.push(make_pair(make_pair(row + 1, col), count + 1));
        }
     }
     return -1;
}

//longest increasing sequence
dp i:1->n, j:0->i
dp[i]=max(dp[i], dp[j]+1)
OR
It=Lower_bound(res.begin(), res.end(),nums[i])
If(it==end) res.push_back(nums[i]);
*it=nums[i]
return res.size()l

//longest increasing path in matrix
//print the path
class Solution {
public:
```

follow up
edge cases

```
int longestIncreasingPath(vector<vector<int>>& matrix) {
    int m = matrix.size();
    if (m == 0) return 0;
    int n = matrix[0].size();
    //LIP len to [i, j]
    vector<vector<int>> mem(m, vector<int>(n, 0));
    vector<vector<vector<int>>> path(m, vector<vector<int>> (n, vector<int>()));
    pair<int, int> maxPoint{0,0};
    int maxlen = 0;
    for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                int len = dfs(i, j, m, n, mem, matrix, path);
                if (len > maxlen){
                    maxlen = len;
                   maxPoint.first = i;
                   maxPoint.second = j;
                }
           }
        }
        vector<int> longestPath = path[maxPoint.first][maxPoint.second];
        for (int i = longestPath.size() - 1; i >=0 ; --i) {
               printf("%d ", longestPath[i]);
        }
        return maxlen;
    }
int dfs(int x, int y, int m, int n, vector<vector<int>>& mem,vector<vector<int>>&
matrix,vector<vector<vector<int>>>& path) {
        if (mem[x][y] != 0) return mem[x][y];
        int maxlen = 1;
        vector<int> maxPath;
        maxPath.push_back(matrix[x][y]);
        vector<int> d{0,1,0,-1,0};
        for (int i = 0; i < 4; ++i) {
            int dx = x + d[i];
            int dy = y + d[i+1];
            if (dx < 0 || dx >= m || dy < 0 || dy >= n || matrix[dx][dy] <= matrix[x][y] ) continue;
            int len = 1 + dfs(dx, dy, m, n, mem, matrix, path);
            vector<int> tempPath(path[dx][dy].begin(), path[dx][dy].end());
            tempPath.push_back(matrix[x][y]);
            if (len > maxlen){
                maxlen = len;
                maxPath = vector<int>(tempPath.begin(), tempPath.end());
            }
        }
        mem[x][y] = maxlen;
        path[x][y] = maxPath;
        return maxlen;
    }
};
```

公交车换乘

```
struct Bus {
    int id;
    unordered_set<char> stops;
    bool operator==(const Bus &a) const {
        return a.id == this->id;
    }
    Bus(int id, string stops) {
        this->id = id;
        for(char c : stops) {
            this->stops.insert(c);
} }
};
bool hasIntersection(Bus a, Bus b) {
    for(auto stop : a.stops) {
        if(b.stops.find(stop) != b.stops.end()) {
            return true;
} }
    return false;
}

int leastSwitch(vector<Bus> buses, char start, char target) {
    queue<pair<Bus, int>> q;
    unordered_set<int> visited;
    for(int i = 0; i < buses.size(); i++) {
        if(buses[i].stops.find(start) != buses[i].stops.end())
```

```
        {
            q.push(make_pair(buses[i], 0));
        }
    }
    while(!q.empty()) {
        Bus curr = q.front().first;
        int switches = q.front().second;
        q.pop();
        if(visited.find(curr.id) != visited.end()) {
            continue;
        }
        visited.insert(curr.id);
        if(curr.stops.find(target) != curr.stops.end()) {
            return switches;
        }
        for(int i = 0; i < buses.size(); i++) {
            if(curr == buses[i]) continue;
            if(hasIntersection(curr, buses[i]) &&
               visited.find(buses[i].id) == visited.end()) {
                q.push(make_pair(buses[i], switches + 1));
        }
        return -1;
}

add/remove/change a char to find longest word
bool removeHelper(string &s, unordered_set<string> &dict){
        if(dict.find(s) == dict.end()) {
                        return false;
        }
        if(s.size() == 1 && dict.find(s) != dict.end()) {
                return true;
        }
        bool res = false
        for(int i = 0; i < s.size(); i++) {
               char tmp = s[i];
               s.erase(i);
               res = res || removeHelper (s, dict);
               s.insert(i, tmp);
        }
        return true;
}

bool dfsRemove(string word, unordered_set<string> &dict) {
    if(dict.find(word) != dict.end()) {
        return false;
    }
    if(word.size() == 1 && valid(word)) {
        return true;
    }
    for(int i = 0; i < word.size(); i++) {
        string substr = word.substr(0, i + 1) + word.substr(i + 1);
        if(dfs(substr, dict)) {
                return true;
        }
    }
    return false;
}

void addNextWord(string word, unordered_set<string> &wordDict, queue<string> &toVisit){
        wordDict.erase(word);
        for(int i=0; i<word.size(); i++){
            char letter=word[i];
            for(int j=0; j<26;j++){
                word[i] = 'a'+j;
                if(wordDict.find(word) != wordDict.end()){
                    toVisit.push(word);
                    wordDict.erase(word);
                }
            }
            word[i]=letter;
        }
    }

int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
        unordered_set<string> wordDict(wordList.begin(), wordList.end());
        //wordDict.insert(endWord);
        queue<string> toVisit;
        addNextWord(beginWord, wordDict, toVisit);
        int res=2;
```

```
        while(!toVisit.empty()){
            int wired=toVisit.size();
            for(int i=0; i<wired; i++){
                string word=toVisit.front();
                toVisit.pop();
                if(word == endWord){
                    return res;
                }
                else{
                    addNextWord(word, wordDict, toVisit);
                    //res++;
                }
            }
            res++;
        }
        return 0;
    }

//isSubsequence 优化
bool isSubsequence(string s, string t) {
        if(s.size() == 0) return true;
        unordered_map<char, vector<int>> m;
        int n = t.size();
        //Preprocess
        for(int i = 0 ; i < n ; ++i){
            m[t[i]].push_back(i);
        }
        if(m.count(s[0]) == 0)return false;
        int prev = m[s[0]][0];
        cout << prev << endl;
        int i = 0;
        for(i = 1 ; i < s.size(); ++i){
            char c = s[i];
            if(m.count(s[i]) == 0)return false;
            int l = -1;
             int r = m[c].size();
            while(l+1 < r){
                int mid = l + (r - l) / 2;
                if(m[c][mid] > prev){
                    r = mid;
                }else{
                    l = mid; }
             }
            if(r == m[c].size())return false;
            prev = m[c][r];
        }
        if(i == s.size()){
            return true;
}
        else{
            return false;
} }

move element at most K left, as much sorted as can
vector<int> asSorted (vector<int>& nums, int k) {
       priority_queue<int, vector<int>, greater<int>> pq;//min-heap
       vector<int> res;
       for(int i: nums) {
               pq.push(i);
               if(pq.size() > k) {
                      res.push_back(pq.top());
                       pq.pop();
               }
        }
       while(!pq.empty()) {
               res.push_back(pq.top());
               pq.pop();
        }
        return res;
}

找到错误枝 node

Longest path in DAG: topological sort -> dp
//bfs
int longestPath(vector<pair<string, string>> pairs) {
    unordered_map<string, vector<string>> graph;
    unordered_map<string, int> inDegree;
```

```
    queue<pair<string, int>> q;
    int res = 0;
    for(auto p : pairs) {
        graph[p.first].push_back(p.second);
        inDegree[p.second] += 1;
        inDegree[p.first] += 0;
}
    for(auto node : inDegree) {
        if(node.second == 0) {
            q.push(make_pair(node.first, 1));
        }
}
    while(!q.empty()) {
        string node = q.front().first;
        int dis = q.front().second;
        q.pop();
        res = max(res, dis);
        for(auto neighbor : graph[node]) {
               q.push(make_pair(neighbor, dis + 1));
        }
}
       return res;
}

//dfs
class Solution {
public:
    int max_dis;
    vector<string> max_path;
    void helper(unordered_map<string, vector<string>> &graph, vector<string> &path, string start, int dis) {
        if(graph[start].size() == 0) {
            if(dis > max_dis) {
                max_path = path;
            }
             return;
        }
        for(auto neighbor : graph[start]) {
            path.push_back(neighbor);
            helper(graph, path, neighbor, dis + 1);
            path.pop_back();
} }

vector<string> longestPath(vector<pair<string, string>> pairs) {
        unordered_map<string, vector<string>> graph;
        unordered_map<string, int> inDegree;
        vector<string> starts;
        vector<string> path;
        max_dis = 0;
        for(auto p : pairs) {
            graph[p.first].push_back(p.second);
            inDegree[p.second] += 1;
            inDegree[p.first] += 0;
}
        for(auto node : inDegree) {
            if(node.second == 0) {
                starts.push_back(node.first);
            }
}
        for(auto start : starts) {
            path = vector<string>({start});
            helper(graph, path, start, 1);
}
       return max_path;
}

//top n word count usr
struct Compare {
       bool operator()(pair<string, int>& a, pair<string, int>& b) {
       return a.second > b.second; }
};
vector<string> topNwordCount (vector<Message>& list) {
       unordered_map<string, int> m;
       for(Message msg: list) {
              m[msg.usr] += msg.text.size();
       }
       priority_queue<pair<string, int>, vector<pair<string, int>>, Compare> pq;
       for(auto p: m) {
              if(!pq.empty() && pq.size() == N && pq.top().second < p.second) pq.pop();
              pq.push(make_pair(p.first, p.second)); }
```

```
        vector<string> res;
        while(!pq.empty()) {
                res.push_back(pq.top().first);
                pq.pop(); }
        reverse(res.begin(), res.end());
        return res;
}

// given a target node in a directed graph, find the shortest cycle including this node, return the whole path.
struct Node {
        string label_;
        unordered_set<Node*> adjacents_;
        Node(const string& label) : label_(label) {}
        void link(Node* v) { adjacents_.insert(v); }
};

vector<Node*> shortest_cycle_path(Node* s)
{
        unordered_map<Node*, Node*> parents;
        deque<Node*> q{ 1, s };
        while (!q.empty()) {
                Node* u = q.back();
                q.pop_back();
                for (auto v : u->adjacents_) {
                        if (parents.count(v) > 0) continue;
                        parents[v] = u;
                        if (v == s) {
                                q.clear();
                                break;
                        } else {
                                q.push_front(v);
                         }
        }
}

// recover path
vector<Node*> path;
auto current = s;
do {
        q.push_front(v);
        current = parents[current];
} while (current != nullptr && current != s);
if (current != nullptr) path.push_back(s);
if (path.size() == 1) path.clear(); // no cycle
reverse(path.begin(), path.end());
return path;



//max rectangle with given point
int maxRectangle(vector<Point> points) {
    unordered_set<Point, hasher> dict;
    int res = 0;
    for(auto p : points) {
        dict.insert(p);
    }
    for(int i = 0; i < points.size(); i++) {
        for(int j = i + 1; j < points.size(); j++) {
            if(points[i].x != points[j].x && points[i].y != points[j].y) {
                Point p1(points[i].x, points[j].y);
                Point p2(points[i].y, points[j].x);
                if(dict.find(p1) != dict.end() && dict.find(p2) != dict.end()) {
                        res = max(res, abs(p1.x - p2.x) * abs(p1.y - p2.y));
     return res; }
}

struct Line {
    int c;
    pair<int, int> range;
};
bool inRange(int x, pair<int,int> range) {
    return (x - range.first)*(x - range.second) <= 0;
}
pair<int, int> overlapRange(Line a, Line b) {
    return make_pair(max(a.range.first,b.range.first), min(a.range.second,b.range.second));
}
int maxArea(vector<Line>& hlines, vector<Line>& vlines) {
    int maxA = 0;
```

```
    for (int i = 0; i < hlines.size() - 1; ++i) // O(N)
        for (int j = i+1; j < hlines.size(); ++j) { // (N)
            auto range = overlapRange(hlines[i], hlines[j]);
            if (range.first >= range.second) continue;
            int xmin = INT_MAX, xmax = INT_MIN;
            int yi = hlines[i].c, yj = hlines[j].c;
            for (auto& vline : vlines) { // (M)
                if (inRange(vline.c, range) && inRange(yi, vline.range)
                   && inRange(yj, vline.range)) {
                    xmin = min(xmin, vline.c);
                    xmax = max(xmax, vline.c);
                }
            }
            if (xmin < xmax) maxA = max(maxA, (xmax - xmin)*abs(yi - yj));
        }
        return maxA;
}

//找第一个前缀不符合的单词
int findfirstUnmatchPrefix(string prefix, vector<string> dict) {
       if(dict.empty()) return -1;
        int plen = prefix.size(), n = dict.size();
        if(dict[0].substr(plen) != prefix) return 0;
        int l = 0, r = n -1;
        while(l < r) {
               int mid = l + (r - l) / 2;
               if(dict[mid].substr(plen) == prefix) l = mid + 1;
               else r = mid;
        }
        if(dict[l].substr(plen) == prefix) return -1;
        return l;
}

//任意一棵树对称
bool isSymmetric(TreeNode* t1, TreeNode* t2) {
       if(!t1 && !t2) return true;
        if(!t1 || !t2) return false;
        int i = 0, j = t2->children.size() - 1;
        if(t1->val == t2->val){
               while(i < t1->children.size() && j >= 0) {
                       if(!isSymmetric(t1->children[i], t2->children[j])) break;
                       ++i, --j;
                if(i < t1->children.size() || j >= 0) return false;
                else return true;
        }
        else return false;
}

//BST 是否对称
bool isSymmetric(TreeNode* root) {
       if(!root) return true;
        return isMirror(root, root);
}
bool isMirror(TreeNode* t1, TreeNode* t2) {
       if(!t1 && !t2) return true;
       if(!t1 || !t2) return false;
       if(t1->val == t2.val){
               return isMirror(t1->left, t2->right) && isMirror(t1->right, t2->left);
       return false;
}

//pair<char, int> iterator
class Encode {
private:
        vector<int>::iterator it;
        char prev, cur;
        int prevCnt, curCnt;
public:
        Encode(ArrayIterator it): it(it){
               prev = '$';
               prevCnt = (it.hasNext())? 1 : -1;
        }
        bool hasNext(){
               return prevCnt > 0;
        }
        pair<char, int> next(){
```

```
                if(prev == '$') prev = it.next();
                while(it.hasNext()) {
                        char t = it.next();
                        if(t != prev) {
                                pair<char, int> res;
                                res.first = prev; res.second = prevCnt;
                                prev = t, prevCnt = 1;
                                return res;
                        } else {
                                prevCnt++;
                        } }
                pair<char, int> res;
                res.first = prev;
                res.second = prevCnt;
                prevCnt=-1;
                return res;
        }
}

//任意插 char， 组成回文，求最短回文
void LCS(vector<int> &idx1, vector<int> idx2, int &commonLen, string s1, string s2) {
    vector<vector<int>> dp(s1.size() + 1, vector<int>(s2.size() + 1, 0));
    for(int i = 1; i <= s1.size(); i++) {
        for(int j = 1; j <= s2.size(); j++) {
            if(s1[i] == s2[j]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            } else {
                dp[i][j] = max(dp[i][j - 1], dp[i - 1][j]);
            }
} }
        commonLen = dp[idx1.size()][idx2.size()];
        int i = s1.size(), j = s2.size();
        while(i >= 0 && j >= 0) {
        if(dp[i - 1][j] == dp[i][j - 1] && dp[i][j] == dp[i][j - 1]) {
                    i - -;
                    j - -;
                    idx1.push_back(i - 1);
                    idx2.push_back(j - 1);
                }
                else if(dp[i - 1][j] == dp[i][j]) {
                    i - -; }
                else {
                    j - -;
        }
}
string formPalidrome(vector<int> &idx1, vector<int> &idx2, const string &s1, const string &s2) {
    string res;
    int i = 0, j = 0;
    for(int k = 0; k < idx1.size();) {
        if(i < idx1[k]) {
            res += s1[i];
            i++;
        }
        else if(j < idx2[k]) {
            res += s2[j];
            j++;
        }
        else {
            res += s1[i];
            i++;
            j++;
            k++;
}

//A-B C-D 映射
unordered_map<int, vector<int>> dictB;
for(int i = 0; i < B.size(); i++) {
        dictB[B[i]] = i;
}
for(int i = 0; i < A.size(); i++) {
        dictA[A[i]] = dictB[B[i]];
}


公司里员工的上下层关系
struct TreeNode {
    vector<TreeNode*> children;
```

```
    int val;
};
int deep(TreeNode *root) {
    if(root == nullptr) return 0;
    queue<TreeNode*> q;
    q.push(root);
    int deep = 0;
    while(!q.empty()) {
        deep++;
        int size = q.size();
        for(int i = 0; i < size; i++) {
               TreeNode *front = q.front();
               q.pop();
               for(auto child : front->children) {
                   q.push(child);
        }
    }
    return deep;
}

class Solution {
    unordered_map<TreeNode*, int> happy_index;
    unordered_map<TreeNode*, int> relation(TreeNode *root) {
        dfs(root);
        return happy_index;
    }
    //num, sum
    pair<int, int> dfs(TreeNode *root) {
        if(root == nullptr) {
            return make_pair(0, 0);
        }

        if(root->children.size() == 0) {
                    return make_pair(1, root->val);
        }
        int num = 0;
        int sum = 0;
        for(auto child : root->children) {
            pair<int,int> happy = dfs(child);
            num += happy.first;
            sum += happy.second;
        }
        happy_index[root] = num / sum;
        return make_pair(num, sum);
};
```

点是否是关于 x = ? 的直线对称

```
class Solution {
public:
    bool isReflected(vector<pair<int, int>>& points) {
        if(points.size() == 0) return true;
        unordered_map<int, unordered_set<int>> dict;
        int max_p = INT_MIN;
        int min_p = INT_MAX;
        int first_y = points[0].second;
        for(auto p : points) {
            dict[p.second].insert(p.first);
            if(p.second == first_y) {
                max_p = max(p.first, max_p);
                min_p = min(p.first, min_p);
            }
        }
        double line = (min_p + max_p) / double(2);
        for(auto points_in_line : dict) {
            unordered_set<int> st = points_in_line.second;
            for(auto p : st) {
                if(st.find(2 * line - p) == st.end()) {
                    return false;
        return true;
};
```

给两个数 n 和 sum，要求输出一个数组，长度为 n，元素的和为 sum

```
vector<int> random(int n, int sum) {
    vector<int> buckets(n);
    for(int i = 0; i < sum; i++) {
        buckets[rand() % n]++;
    return buckets;
}
```

```
vector<int> random2(int n, int sum, int cap) {
    vector<int> buckets(n);
    unordered_set<int> blackList;
    for(int i = 0; i < sum; i++) {
        int rnum = rand() % (n - blackList.size());
        int idx = 0;
        for(int j = 0; j < n; j++) {
            if(idx == rnum && blackList.count(j) == 0) {
                buckets[j]++;
             }
            if(blackList.count(j) == 0) idx++;
        }
        if(buckets[rnum] == cap) {
            blackList.insert(rnum);
        }
      }
      return buckets;
}

Find largest word in dictionary by deleting some characters of given string
string findLongestWord(string s, vector<string>& d) {
        string res;
        for(string word:d){
            int j=0;
            int k=0;
            while(j<s.size() && k<word.size()){
                if(s[j]==word[k]){
                    k++;
                }
                j++;
            }
            if(k==word.size() && (res.size()<word.size()|| (res.size()==word.size()&&res>word))){
                res=word;
            }
         return res;
    }

//给一个数，问是不是两个完全平方数的和
bool isSquare(int n) {
    if(n < 0) return false;
    return pow(int(sqrt(n)), 2) == n;
}

bool twoSquare(int n) {
    for(int i = 1; i < ceil(sqrt(n)); i++) {
        if(isSquare(n - i * i)) return true;
    }
    return false;
}

candy crush P109
generate maze P94
判断两个点能否组成 line，使剩余点在一边 P81
```