前言：公子龙想说的话

嘿，亲爱的读者，跨越千山万水，我们相遇了^_^

分享一些我写过的文章吧，希望有一篇能够俘获你的芳心：

从自动化跨考计算机后，我来到阿里工作

两个月 4052 人，内推我们是认真的

北漂五年，创业、字节、和阿里，现在回家

计算机领域有哪些常见的比赛

[在 NLP 领域创业，真的很难]

[如何科学的打开 Leetcode]

[2020 互联网应届硕士的薪资情况]

[最好的互联网公司]

[在 NLP 领域创业，真的很难]

[25岁做一件5年后能够受益的事情]

[2019 年被裁的同事们]

[关于工作，我的一些小算盘]

[读研三年，值还是不值]

[谈谈计算机行业的秋招和春招]

[第一份工作是选择安逸，还是勇敢尝试？]

[从一年一度的宿舍聚会说起]

[研究生毕业后，再重新读个硕士]

[奔五十的他，选择和年轻人一决高下]

[2020 年的算法，降温之后会更好]

[关于计算机读研的小建议]

[爱情之旅，去北大读研]

[读研，竞赛，与实习]

[谈谈实习这件小事]

[谈谈我熟悉的第一位 CTO]

我今年 25 岁，6 月份研究生毕业，即将入职，如果你对我感兴趣，还想经常看到我的文章，欢迎关注我的公众号：公子龙，那里有添加我微信好友的方式哦，坑位不多，速来呀~ 与你不见不散~

概率题

富翁和乞丐的财富之旅

> 富翁和乞丐财富差距为z，每天富翁得到1的概率为p，乞丐得到1的概率为q，求将来某一天持平的概率

硬币质量问题

> 一枚硬币扔10次，8次正面朝上，求这枚硬币有问题的概率?

依旧设硬币有问题概率为p，有问题硬币正面朝上概率为q，一枚硬币扔10次，8次正面朝上为事件A，则

P(A|coin=Nomal)= $C_{10}^{8}(\frac{1}{2})^{10}$

P(A|coin=Abnomal)= $C_{10}^{8}q^{8}(1-q)^{2}$

P(A,coin=Abnomal)=P(A|coin=Abnomal)*P(coin=Abnomal)= $C_{10}^{8}q^{8}(1-q)^{2}p$

P(A)=P(A|coin=Nomal)*P(coin=Nomal)+P(A|coin=Abnomal)*P(coin=Abnomal)=
$C_{10}^{8}(\frac{1}{2})^{10}(1-p)+C_{10}^{8}q^{8}(1-q)^{2}p$

P(coin=Abnomal|A)=P(A,coin=Abnomal)/P(A)= $\frac{C_{10}^{8}q^{8}(1-q)^{2}p}{C_{10}^{8}(\frac{1}{2})^{10}(1-p)+C_{10}^{8}q^{8}(1-q)^{2}p}$ =

$$\frac{q^{8}(1-q)^{2}p}{(\frac{1}{2})^{10}(1-p)+q^{8}(1-q)^{2}p}$$

这个要先统计有问题硬币占总硬币数目的比例，以及这种问题硬币投掷产生的偏差才能计算，其中q=0.5时，也就是问题硬币表现得和普通硬币一致时，答案就是p，q<0.5时，答案小于p，q=0.8时， $q^{8}(1-q)^{2}$ 最大，在p不变时，答案最大

蓄水池采样

蓄水池抽样：从N个元素中随机的等概率的抽取k个元素，其中N无法确定

先给出代码：

```
Init : a reservoir with the size: k
for i= k + 1 to N
  M = random(1, i);
  if(M < k)
    SWAP the Mth value and ith value
end for
```

上述伪代码的意思是：先选中第1到k个元素，作为被选中的元素。然后依次对第k+1至第N个元素做如下操作：

每个元素都有k/x的概率被选中，然后等概率的（1/k）替换掉被选中的元素。其中x是元素的序号。

算法的成立是用数学归纳法证明的：

```
每次都是以 k/i 的概率来选择
例：k=1000的话， 从1001开始作选择，1001被选中的概率是1000/1001，1002被选中的概率是1000/1002，与我们直觉是相符的。
```

接下来证明：
假设当前是i+1，按照我们的规定，i+1这个元素被选中的概率是k/i+1，也即第 i+1 这个元素在蓄水池中出现的概率是k/i+1
此时考虑前i个元素，如果前i个元素出现在蓄水池中的概率都是k/i+1的话，说明我们的算法是没有问题的。

对这个问题可以用归纳法来证明：k < i <=N
1.当i=k+1的时候，蓄水池的容量为k，第k+1个元素被选择的概率明显为k/(k+1)，此时前k个元素出现在蓄水池的概率为 k/(k+1)，很明显结论成立。
2.假设当 j=i 的时候结论成立，此时以 k/i 的概率来选择第i个元素，前i-1个元素出现在蓄水池的概率都为k/i。
证明当j=i+1的情况：
即需要证明当以 k/i+1 的概率来选择第i+1个元素的时候，此时任一前i个元素出现在蓄水池的概率都为k/(i+1).
前i个元素出现在蓄水池的概率有2部分组成，①在第i+1次选择前得出现在蓄水池中，②得保证第i+1次选择的时候不被替换掉
①.由2知道在第i+1次选择前，任一前i个元素出现在蓄水池的概率都为k/i
②.考虑被替换的概率：
首先要被替换得第 i+1 个元素被选中(不然不用替换了)概率为 k/i+1，其次是因为随机替换的池子中k个元素中任意一个，所以不幸被替换的概率是 1/k，故
前i个元素(池中元素)中任一被替换的概率 = k/(i+1) * 1/k = 1/i+1
则(池中元素中)没有被替换的概率为：1 - 1/(i+1) = i/i+1
综合① ②,通过乘法规则
得到前i个元素出现在蓄水池的概率为 k/i * i/(i+1) = k/i+1
故证明成立

吃月饼

月神特别喜欢吃月饼，中秋节时快手发了10个月饼，已知月神一天至少吃一个月饼；请问，月神在3天内将10个月饼全部吃完的概率为?

采用插板法，10个月饼排成一行，如果在2天内吃完，就在里面插入一个板子，所以是C91,所以就应该是(C90+C91+C92)/(C90+C91+C92+C93+C94+C95+.....+C99)=23/256

红球绿球

一个箱子中有15%的红球和85%的绿球，小明随机取出1个球，他不能看到球，但他根据手感判断该球为红色。已知小明根据手感判断颜色的正确概率为80%，那么他取到的球实际为红色的概率为?

小明取到红球，判断为红球的概率：0.15*0.8=0.12

小明取到绿球，判断为红球的概率：0.85*0.2=0.17

以上两种情况为题设所给出的前提条件，则在此基础上，小明取到的球实际为红球的概率为：

0.12/（0.12+0.17）≈ 0.41

几何分布

Beta星球非常重男轻女，一个家庭如果一胎生女儿的话，会继续生下一个孩子，直到生男孩为止。已知生男孩和女孩的概率都是50%，每个家庭至少会生一个孩子，那么Beta星球平均每个家庭的孩子数量为?

这是一个几何分布，几何分布的期望等于E(X)=1/p，设家庭有k个孩子的概率为p，则$p = (\frac{1}{2})^k$

$E(X) = \sum_{k=1}^{\infty} k \times (\frac{1}{2})^k = 2$

乘法原理问题

7 个同学围坐一圈，要选 2 个不相邻的作为代表，有多少种不同的选法

先选1个同学出来，7种选法；再选第2个同学出来，4种选法。又因为第一次选a第二次选b，和第二次选b第一次选a，情况一样，是重复的。所以总共有：7 * 4 / 2 = 14 种选法。

编程题

完整的c++程序

```
#include<vector>
#include<iostream>

using namespace std;

int getNums(int n) {
    vector<int> dp(n + 1, 0);
    dp[0] = 1;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= i; j <<= 1) {
            dp[i] += dp[i - j];
        }
    }
    return dp[n];
}

int main() {
    int m;
    cin >> m;
    for(int i = 0; i < m; i++) {
        int n;
        cin >> n;
        cout << getNums(n) << endl;
    }
    return 0;
}
```

二分查找

[你真的会写二分吗]

```
while(left < right) {
  int mid = left + (right - left) >> 1;
  if(vals[mid] < key) left = mid + 1;
  else right = mid;
}
return left;
```

排序算法

算法	最好时间	最坏时间	平均时间	额外空间	稳定性
选择	n^2	n^2	n^2	1	稳定
冒泡	n	n^2	n^2	1	不稳定
插入	n	n^2	n^2	1	稳定
希尔	n	n^2	$n^{1.3}$(不确定)	1	不稳定
归并	$nlog_2n$	$nlog_2n$	$nlog_2n$	n	稳定
快排	$nlog_2n$	n^2	$nlog_2n$	log_2n至n	不稳定
堆	$nlog_2n$	$nlog_2n$	$nlog_2n$	1	不稳定
基数	n*k	n*k	n*k	n+k	稳定

[十大排序算法]

[排序算法的复杂度、实现和稳定性]

快速排序

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        quickSort(nums, 0, nums.size() - 1);
    }

private:
    void quickSort(vector<int>& nums, int left, int right) {
        if(left >= right) return ;
        int mid = partition(nums, left, right);
        quickSort(nums, left, mid - 1);
        quickSort(nums, mid + 1, right);
    }
```

```
    int partition(vector<int>& nums, int left, int right) {
        int key = nums[left];
        while(left < right) {
            while(left < right && nums[right] >= key) right--;
            nums[left] = nums[right];
            while(left < right && nums[left] < key) left++;
            nums[right] = nums[left];
        }
        nums[left] = key;
        return left;
    }
};
```

单链表排序

```
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        ListNode* root = merge_sort(head);
        return root;
    }

private:
    ListNode* merge_sort(ListNode* head) {
        if(!head || !head->next) {
            return head;
        }
        ListNode *slow, *fast;
        slow = head, fast = head->next;
        while(fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        ListNode *heada = head, *headb = slow->next;
        slow->next = NULL;
        return merge_list(merge_sort(heada), merge_sort(headb));
    }

    ListNode* merge_list(ListNode* heada, ListNode *headb) {
        ListNode *res = new ListNode(0);
        ListNode *p = res;
        while(heada || headb) {
            if(heada && (!headb || heada->val < headb->val)) {
                res->next = new ListNode(heada->val);
                res = res->next;
                heada = heada->next;
            }
            if(headb && (!heada || heada->val >= headb->val)) {
```

```
                res->next = new ListNode(headb->val);
                res = res->next;
                headb = headb->next;
            }
        }
        return p->next;
    }
};
```

二叉树

二叉搜索树第k大

```
leetcode 230 二叉搜索树第k大
class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        stack<TreeNode*> st;
        while(root || !st.empty()) {
            while(root) {
                st.push(root);
                root = root->left;
            }
            root = st.top();
            if(--k == 0) {
                return root->val;
            }
            st.pop();
            root = root->right;
        }
        return -1;
    }
};
```

链表

移除倒数第N个节点

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode *fast = head, *slow = head;
        while(n--){
            fast = fast->next;
        }
        if(fast == NULL){
```

```
            head = head->next;
            return head;
        }
        while(fast->next != NULL){
            fast = fast->next;
            slow = slow->next;
        }
        slow->next = slow->next->next;
        return head;
    }
};
```

链表有环问题

[判断一个单链表是否存在环型链接并找出环的开始节点]

```
1. 判断链表是否有环
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode *slow = head, *fast = head;
        while(slow && fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if(slow == fast) {
                return true;
            }
        }
        return false;
    }
};

2. 判断链表是否有环，如果有环，求环的第一个节点
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
```

```
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        ListNode *slow = head, *fast = head;
        while(slow && fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if(slow == fast) {
                ListNode *p = head, *q = slow;
                while(p && q) {
                    if(p == q) {
                        return p;
                    }
                    p = p->next;
                    q = q->next;
                }
            }
        }
        return NULL;
    }
};
```

二叉树转链表

```
leetcode 114
class Solution {
public:
    void flatten(TreeNode* root) {
        if(!root) return ;
        flatten(root->left);
        flatten(root->right);
        TreeNode* tmp = root->right;
        root->right = root->left;
        root->left = NULL;
        while(root->right) {
            root = root->right;
        }
        root->right = tmp;
    }
};
```

DFS

[leetcode 79]

```
class Solution {
public:
    bool exist(vector<vector<char>>& board, string word) {
```

```
        for(int i = 0; i < board.size(); i++) {
            for(int j = 0; j < board[0].size(); j++) {
                if(dfs(board, word, i, j))
                    return true;
            }
        }
        return false;
    }

private:
    bool dfs(vector<vector<char>>& board, string word, int x, int y) {
        if(word.empty()) return true;
        int row = board.size(), col = board[0].size();
        if(x < 0 || y < 0 || x >= row || y >= col || board[x][y] != word[0])
return false;
        char ch = board[x][y];
        board[x][y] = '_';
        word = word.substr(1);
        bool ret = (dfs(board, word, x - 1, y)
                    || dfs(board, word, x + 1, y)
                    || dfs(board, word, x, y - 1)
                    || dfs(board, word, x, y + 1));
        board[x][y] = ch;
        return ret;
    }
};
```

BFS

[leetcode 1091]

```
class Solution {
public:
    int shortestPathBinaryMatrix(vector<vector<int>>& grid) {
        int n = grid.size();
        if(grid[0][0] == 1|| grid[n - 1][n - 1] == 1) {
            return -1;
        }

        queue<vector<int>> q;
        q.push({0, 0, 1});
        while(!q.empty()) {
            vector<int> node = q.front();
            q.pop();
            for(int i = -1; i <= 1; i++) {
                for(int j = -1; j <= 1; j++) {
                    int x = node[0] + i, y = node[1] + j;
                    if(x < 0 || x >= n || y < 0 || y >= n
                       ||(i == 0 && j == 0) || grid[x][y] == 1) {
```

```
                    continue;
                }
                if(x == n - 1 && y == n - 1) {
                    return node[2] + 1;
                }
                q.push({x, y, node[2] + 1});
                grid[x][y] = 1;
            }
        }
    }
    return -1;
}
};
```

拓扑序

拓扑排序

求幂

```
1. leetcode 50 pow
class Solution {
public:
    double myPow(double x, int n) {
        if(n == 0.0)
            return 1.0;
        long t = n;
        if(n < 0) {
            x = 1 / x;
            t = -t;
        }
        double res = 1.0;
        while(t) {
            if(t & 1) {
                res *= x;
            }
            t >>= 1;
            x *= x;
        }
        return res;
    }
};

2. leetcode 372 super pow
class Solution {
public:
    int superPow(int a, vector<int>& b) {
        if(b.empty()) return 1;
```

```
        int t = b.back();
        b.pop_back();
        return pow(superPow(a, b), 10) * pow(a, t) % base;
    }

private:
    const int base = 1337;
    int pow(int a, int k) {
        a %= base;
        int res = 1;
        while(k) {
            if(k & 1) res = (res * a) % base;
            a = (a * a) % base;
            k >>= 1;
        }
        return res;
    }
};
```

两个有序数组第k大

```
leetcode 4
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        int n1 = nums1.size(), n2 = nums2.size();
        return (findKth(nums1, nums2, (n1 + n2 + 1) >> 1) +
                findKth(nums1, nums2, (n1 + n2 + 2) >> 1)) / 2.0;
    }

private:
    int findKth(vector<int> nums1, vector<int> nums2, int k) {
        int n1 = nums1.size(), n2 = nums2.size();
        if(n1 == 0) return nums2[k - 1];
        if(n2 == 0) return nums1[k - 1];
        if(k == 1) return min(nums1[0], nums2[0]);
        int mid1 = min(n1, k / 2), mid2 = min(n2, k / 2);
        if(nums1[mid1 - 1] <= nums2[mid2 - 1])
            return findKth(vector<int> (nums1.begin() + mid1, nums1.end()),
nums2, k - mid1);
        else
            return findKth(nums1, vector<int> (nums2.begin() + mid2,
nums2.end()), k - mid2);
    }
};
```

包含k个数组的至少一个元素

```
leetcode 632
class Solution {
public:
    vector<int> smallestRange(vector<vector<int>>& nums) {
        int n = nums.size();
        vector<int> res{0, INT_MAX};
        vector<pair<int, int>> total;
        for(int i = 0; i < n; i++) {
            for(auto v : nums[i]) {
                total.push_back(make_pair(v, i));
            }
        }
        sort(total.begin(), total.end());
        unordered_map<int, int> mm;
        int m = total.size();
        int count = 0;
        int left = 0, right = 0;
        for(; right < m; right++) {
            mm[total[right].second]++;
            if(mm[total[right].second] == 1) count++;
            while(count == n) {
                if(total[right].first - total[left].first < res[1] - res[0]) {
                    res = {total[left].first, total[right].first};
                }
                if(mm[total[left].second] == 1) count--;
                mm[total[left].second]--;
                left++;
            }
        }
        return res;
    }
};
```

字符串

回文字符串

```
1. leetcode 5 最长连续回文串 DP
class Solution {
public:
    string longestPalindrome(string s) {
        if(s.empty()) return "";
        int n = s.size();
        vector<vector<int>> dp(n, vector<int> (n, 0));
```

```
        string longString = s.substr(0, 1);
        for(int i = 0; i < n; i++) dp[i][i] = 1;
        for(int i = n - 1; i >= 0; i--) {
            for(int j = i + 1; j < n; j++) {
                if(s[i] == s[j] && (dp[i + 1][j - 1] > 0 || j == i + 1)) {
                    dp[i][j] = dp[i + 1][j - 1] + 2;
                    if(j - i + 1 > longString.size()) {
                        longString = s.substr(i, j - i + 1);
                    }
                }
            }
        }
        return longString;
    }
};
```

2. leetcode 647 计算回文串数目

```
class Solution {
public:
    int countSubstrings(string s) {
        if(s.empty()) return 0;
        int n = s.size();
        vector<vector<bool>> dp(n, vector<bool> (n, false));
        for(int i = 0; i < n; i++)
            dp[i][i] = true;
        for(int i = n - 1; i >= 0; i--) {
            for(int j = i + 1; j < n; j++) {
                if(s[i] == s[j]) {
                    dp[i][j] = dp[i + 1][j - 1];
                    if(i + 1 == j)
                        dp[i][j] = true;
                }
            }
        }
        int res = 0;
        for(int i = 0; i < n; i++) {
            for(int j = i; j < n; j++) {
                if(dp[i][j])
                    res++;
            }
        }
        return res;
    }
};
```

3. leetcode 516 最长回文子串

```
class Solution {
public:
    int longestPalindromeSubseq(string s) {
```

```
        int n = s.size();
        vector<vector<int>> dp(n, vector<int> (n, 0));
        for(int i = 0; i < n; i++) {
            dp[i][i] = 1;
        }
        for(int i = n - 1; i >= 0; i--) {
            for(int j = i + 1; j < n; j++) {
                if(s[i] == s[j]) dp[i][j] = dp[i + 1][j - 1] + 2;
                else dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
            }
        }
        return dp[0][n - 1];
    }
};
```

句子和词典

```
leetcode 139 140 判断一个句子是否由一个词典构成，并给出所有的构成方法
class Solution {
public:
    vector<string> wordBreak(string s, vector<string>& wordDict) {
        int n = s.size();
        vector<bool> dp(n + 1, 0);
        dp[0] = true;
        for(int i = 0; i < n; i++) {
            for(auto w : wordDict) {
                if(i + w.size() <= n && s.substr(i, w.size()) == w) {
                    if(dp[i] == true)
                        dp[i + w.size()] = dp[i];
                }
            }
        }
        vector<string> res;
        dfs(res, s, "", n, wordDict, dp);
        return res;
    }

private:
    void dfs(vector<string>& res, string s, string cur, int pos,
             vector<string>& wordDict, vector<bool>& dp) {
        if(pos == 0) {
            res.push_back(cur);
            return ;
        }
        if(!dp[pos]) return ;
        for(auto w : wordDict) {
            int newPos = (int)pos - (int)w.size();
            if(newPos >= 0 && s.substr(newPos, w.size()) == w) {
                string newCur = (cur == "" ? w : w + " " + cur);
```

```
                dfs(res, s, newCur, newPos, wordDict, dp);
            }
        }
    }
};
```

位运算

位运算处理数组中的数

[解法]

```
leetcode 137
题目描述：给定一个包含n个整数的数组，除了一个数出现一次外所有的整数均出现三次，找出这个只出现一次的整数。
题目分析：对于除出现一次之外的所有的整数，其二进制表示中每一位1出现的次数是3的整数倍，将所有这些1清零，剩下的就是最终的数。用ones记录到当前计算的变量为止，二进制1出现“1次”（mod 3 之后的1）的数位。用twos记录到当前计算的变量为止，二进制1出现“2次”（mod 3 之后的 2）的数位。当ones和twos中的某一位同时为1时表示二进制1出现3次，此时需要清零。即用二进制模拟三进制计算。最终ones记录的是最终结果。

class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int ones = 0, twos = 0, threes = 0;
        int  n = nums.size();
        for(int i = 0; i < n; i++) {
            twos |= ones & nums[i];
            ones ^= nums[i];
            threes = ~(ones & twos);
            ones &= threes;
            twos &= threes;
        }
        return ones;
    }
};
```

动画展示leetcode解题思路

网址：https://github.com/MisterBooo/LeetCodeAnimation