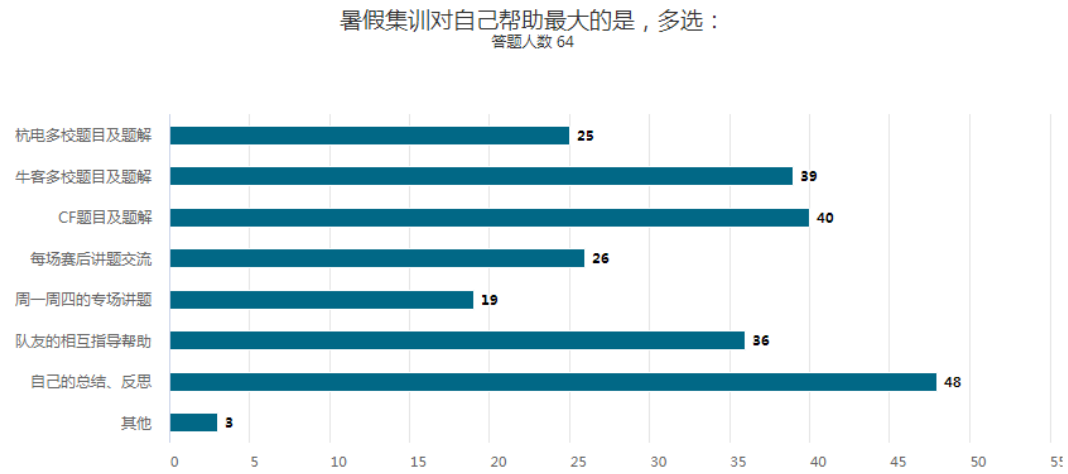
# 哈尔滨理工大学 ACM-ICPC 集训队暑假集训题解汇编

# (2019 年)

**目　录**

# 哈尔滨理工大学 ACM-ICPC 集训队 2019 暑假集训问卷调查结果

唐远新

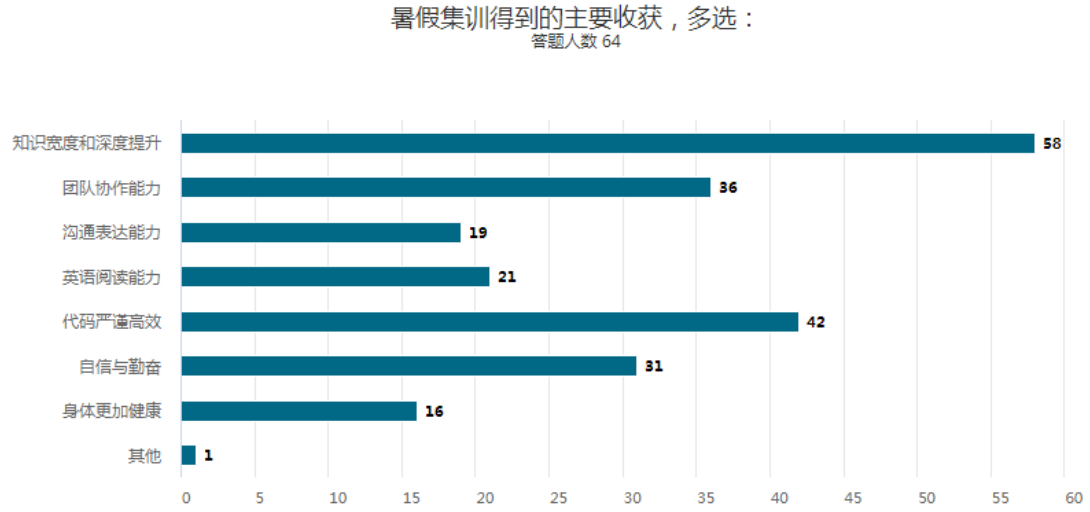



**A 暑假集训对自己帮助最大的是（选项外）：**

1、洛谷、vj 练习

2、通过牛客看到别人写的更好的代码，丰富了自己的模板，同时也通过看别人的代码学到了解决一道题的不同思路。

3、oi 集训队论文




**B 暑假集训得到的主要收获（选项外）：**

1、获得了更多的模板

**C 建议与计划**

**1、对讲题的建议**

不要直接讲答案，过程怎么来的讲清楚。

多锻炼大一讲题

多锻炼大一同学

多让几个人分享自己的方法

多一些互动

放慢节奏，精而不多

更精细

很好

加一些细节的讲解

减少时间提高效率

建议 cf 训练公开代码供大家学习

建议小屏讲题 这样可以私发消息

建议在讲题前告知需要的前置知识点,便于提前预习相应知识,这样听题时收获会更多

讲的都很好，自己有很多不足的地方

讲的很好

讲解细一点

讲题的部分题比较拐，认为一些经典题哪怕是多做，多讲一次也是有收获的。

讲题都很全面，没有建议

讲题更细致一些

讲题还挺满意的

讲题后有 word 文件

讲题时出现的知识点希望能附上一些比较详细的博客地址

讲题同学应该要留给大家短暂的思考，不能照着 ppt 念一遍就完事了。讲题同学必须要认真准备，不能去抄个博客抄个 ac 代码就算准备好了。

讲题应该多讲讲如何思考，而不是直接给出做法

可以多偏向于思路解析

可以具体的讲一下思路到底是怎么来的以及做题时遇到的坑点

可以适当的讲的慢一点

每次训练赛以后讲题挺好的

如果需要在讲题的时候写或者画一些东西，用鼠标在屏幕上写字不太方便，也不方便辨认

赛后补题用电脑讲题

随讲随问听没听懂

提前发题读题目

提前告诉听课同学讲什么方面的内容，以便基础差的同学提前预习，而不至于完全听不懂。

提前预告背景知识的准备

挺好的

希望讲题中间有休息时间，要不太枯燥。而且也需要点时间理解一下刚刚讲的，然后才能接着听。

希望能将每次讲题都形成题解，便于后期理解学习

希望能有更多人参与

像杭电这些网上找不到题解的比赛，赛后大家可以分享一下自己 AC 的代码，这样也可以学互相习下如何简洁高效的写代码

学长讲的很好

要都像陈鑫顾佳昊学长那样透彻就好了

已经很好了

有时候讲题人的题目讲的不够细致，但是让一些同学在训练时还要准备课件讲题，还是非常感谢的。

自己能力有限，很多时候都听不懂

最好有代码的支持

**2、对选题的建议**

3+5
cf 难度正好
不希望一个专题一个专题的选，希望混着选。
不要选太偏的知识点
从简单到难
大一打牛客杭电容易自闭
更能体现知识点
更注重算法的理解进行选题
还是有点 cfc 题类型的吧
杭电很难不容易做牛客难度适中
好
很好
金牌题对大部分同学难度太大，可以稍微降低一点难度。
尽量选铜牌和银牌题，不要选过难的题
可以多选包含新做法或者算法的题目
可以选一些中等题
没有
每次都可以针对不同类型不同专题出题
少选点金牌题
少选偏题怪题
数论的题很少
思维与算法同时兼顾
题的难度有阶梯性，很合适
无,选题质量一直很好
希望能系统的选择专题
选的很好
选题层次明显 分布均匀
选题都很好
选题很不错，思维题占了主要
选题难度合适
选题难易都有，很适合大一
选题希望难度有阶梯化
选题要挑有用的，不能一味地挑难的。无效题是浪费大家的时间。
选题应该更具有普遍性，难度分层，适当放上思维签到题，而不是放上一些模板题。
选一下基础题 在讲难题 最好一个类型两道
选择历年区域赛的题目
银牌题
应该难易结合
有易有难
这次个人训练采取在 cf 上选题我个人觉得非常好。
综合题型
最好是往年的真题，涵盖面广

**3、对训练时间和组织的建议**

安排的很合理了

不知道特定专题的题目应该找谁问，既能解决自己的问题也能巩固别人对该专题的理解。希望有人能拍桌子说 dp 题问我，或者图论问我。

当下的安排挺合理的

都好

对现在是时间安排感觉很合理

对于补题、刷题、写博客等可以队员间相互监督，通过队内布置任务等形式相互促进。

合适

很好

机房太热了

集训队互测

集训队互测可以激发学习兴趣

加个午休吧

建议安排午休

建议更加紧张一些

觉得很正常

可以选在下午

课余

密度有点高来不及补题

时间安排的挺好

时间紧凑而衔接

时间上我个人觉得不错。

挺好的

希望能留有短暂的假期

希望训练时间不要排的太紧，这样能有更多的时间补题和学习新知识。

下午稍微晚一点，中午睡一觉下午会更有精力

训练时间无建议，机房通风不是很好，在里面呆时间久了头会比较晕，影响学习进度。

要给足够补题和对新内容理解学习的时间

一天打半天的训练赛，剩下半天和晚上补题，学习新算法的时间不太充足

支持

中午加个午休吧

中午加午休

周六周日

最好不要与上课时间冲突

**4、对队友的建议**

别老是让我自己写题

除了板子题还应该提高思维能力，主要是动态规划类问题需要加强

都一起加油吧

锻炼身体，增强身体素质

队友常常不来

队友带带我

队友都很给力

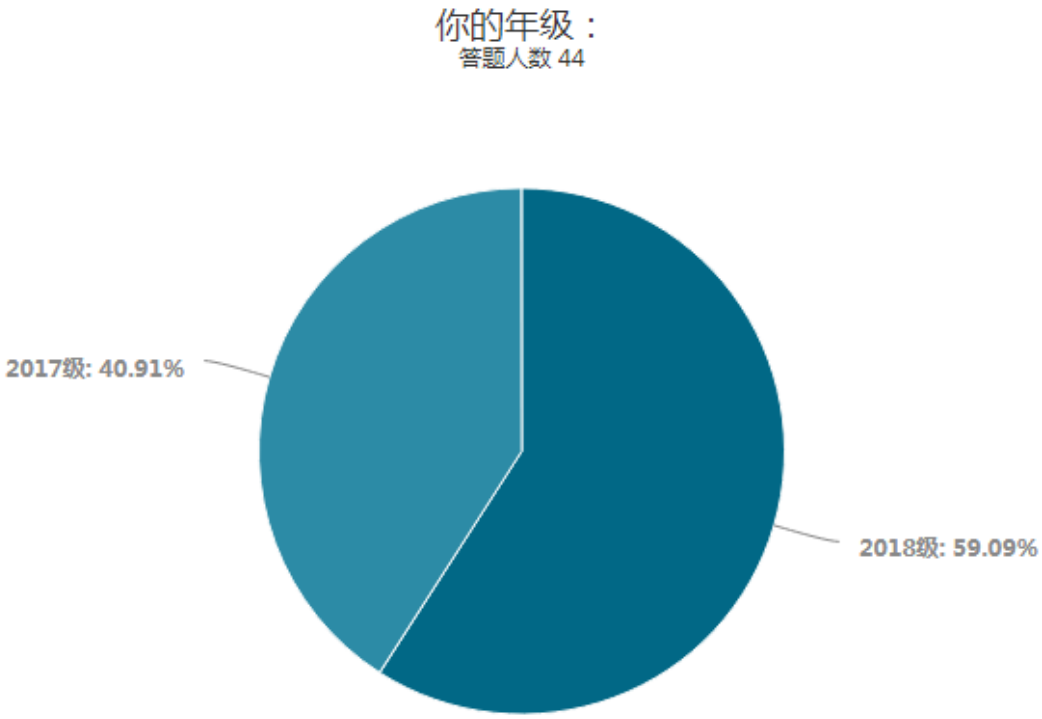
队友都很给力，一起进步

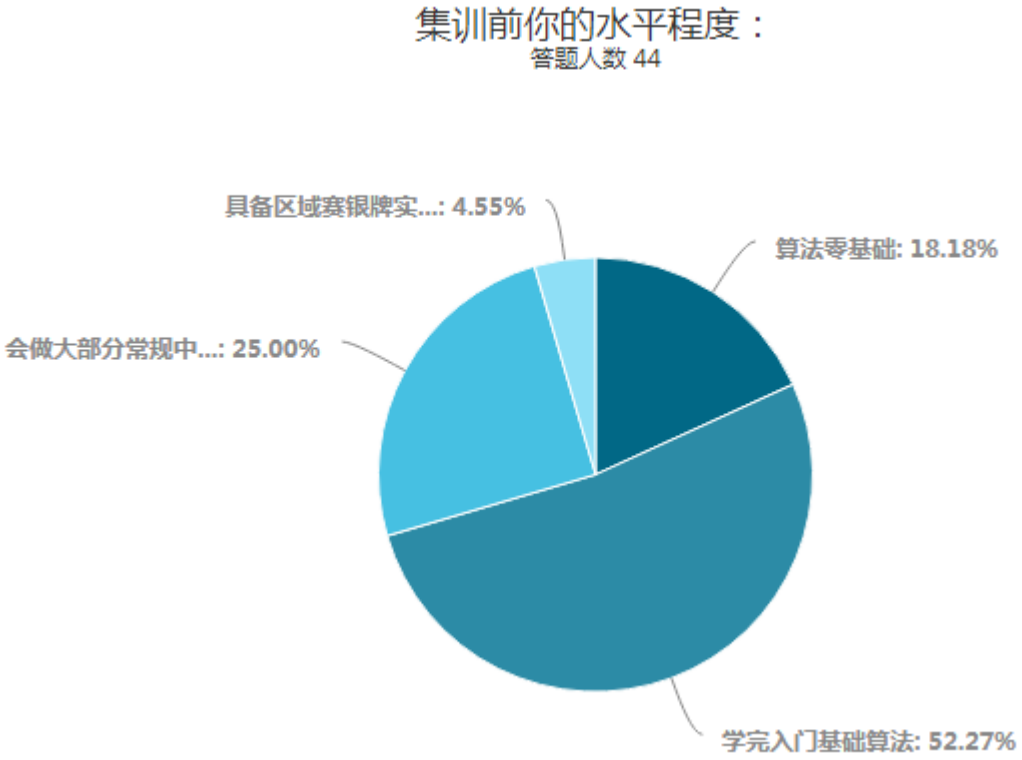
队友很好
队友很好，希望大家都坚持下去
队友很优秀
队友积极性不高
对每个算法尽量刷 10 个题以上
多读题，别卡单个题
多耐心
多做题多交流
分工更明确一些
感觉都挺好的，一起努力吧
沟通做的很到位
好好学习
好好学英语
很好
积极点
模板应该每人有一份自己的，每队也有一份自己的。
能够共同进步，一点一点克服难点
少打辣鸡游戏
提高沟通协作效率
挺好的
希望队友比赛期间能少唠嗑多专注于题目
希望队友可以多开动思维，不能死扣一道题而放弃看其他题或者说不愿看其他题，题意确定非常重要，读题的过程要尽可能的仔细，对于数据范围也应该尽可能的考虑进去，数据范围对于算法的选择有着直接影响。
希望队友能够更勤奋学习算法。
希望更默契
希望可以更专注一点
学习进去
寻找自己到适合自己，配合默契的队友，相互促进学习，相互扩展思维方式，代码风格不断融得入，相互分析每次训练赛时出现的问题，不断改正，感受到自己和队友不断的进步，团队的不断进步是非常开心的。努力把握好自己在团队中的位置，了解队友，在队友出现一些问题时要相互纠正，探讨。提高实力的同时，更是收获了非常好的友谊。
要充分利用队友的优势
一起加油
一起进步
在团队赛中，不要再讨论 a 过的题，赛后再讨论
增加交流
至少要 2 人一起想同一道题，写题，检查

**5、未来打算**

acm 第一人
挨个刷铜牌银牌题
不一定要多牛逼 但是能比别人学到更多东西就好
参加区域赛，保铜冲银
出国

打好基础，坚持 acm
大数据程序员
多锻炼
好好工作
好好加油
好好学习
好好学习天天向上
继续加油
继续努力
继续努力学习 争取可以拿到比赛机会
继续提升自己的能力
继续学习算法
继续学知识点,补完多校的题,做 cf
继续训练
继续做题
坚持下去，快速提高
坚持学习
进一个好的公司
近两年好好打 ACM，尽量拿到较好的名次（区域赛金）
就业
考研
考研，竞赛加油
没想好
努力提升自己，找到好的工作。
培养工程思维和能力。
确定自己的队友
实习就业
提高自己知识面广度与深度
提高自身能力
提升自我，尽早实习
通过队伍的不断努力，争取下学期的区域赛能够拿到银牌。
拓宽知识面，要弄懂算法原理
未来希望能稳定发挥，拿出自己的真正实 i 力。
希望可以跟住训练
希望能多进比赛
想奔着区域赛拿奖去
循序渐进，学习算法
找个好工作
整理模板，按照模板知识点补技能树。补多校题，扩充知识点，备战区域赛。
只想做好当下，补题去了~~
准备再次深入学习一些算法


你的年级：
答题人数 44
2017级: 40.91%
2018级: 59.09%


集训前你的水平程度：
答题人数 44
具备区域赛银牌实...: 4.55%
算法零基础: 18.18%
会做大部分常规中...: 25.00%
学完入门基础算法: 52.27%

# GYM 102012H 贪心+扫描线 (计 17-5 陈鑫)

## 一、题号与题目名称

GYM 102012H Rikka with A Long Colour Palette

## 二、题意简述

有 n 条线段[l,r]，k 种颜色，为每条线段染一种颜色，问能被 k 种颜色覆盖的区间的最大总长，并输出一种染色方案。

∑n<=2e6，1<=k<=2e5，0<=l<r<= 1e9。

## 三、主要知识点及其运用

本题涉及贪心和扫描线的思想。

## 四、完整解题思路描述

把 n 条线段的 2*n 个端点按位置排序，相同位置右端点更靠前，把 k 种颜色加入未染色队列 rCol 里。

从前到后扫描每个端点：

若 rCol 为空，说明当前端点到上个端点之间的区间已覆盖 k 种颜色，累加答案；

如果该端点是左端点，且 rCol 不空，则取队头为该线段立即染色，若 rCol 为空，随便染色会有后效性，不妨将该线段扔进延迟染色队列 rSeg 中；

如果该端点是右端点，且已被染色，则将该颜色归还到 rCol 中，若未被染色，则说明该线段既没有在左端点立即染色，又没有在右端点前延迟染色，不会再有贡献，随便染色即可。

处理完该端点后，若 rCol 不空，则从 rSeg 中取出延迟染色的线段一一为其分配颜色，直到两者其中一个为空。

总时间复杂度为排序 O(nlogn)+扫描位置 O(n)+延迟染色 O(n)。

## 五、技巧与坑点

排序时相同位置右端点更靠前，是因为右端点归色要先于左端点染色；

已经扫描完右端点的线段，若其出现在 rSeg 中，要及时剔除。

## 六、参考文献与学习资料

https://acm.ecnu.edu.cn/wiki/index.php?title=2018_ACM-ICPC_Xuzhou_Regional_Onsite

## 七、完整代码

```
#include<bits/stdc++.h>
#define sz(x) ((int)x.size())
#define mp(x, y) make_pair(x, y)
using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
const int MAXN = 400010;

int T, n, k, col[MAXN];
```

```
struct Node{
    int st, id;
    Node(){}
    Node(int _st, int _id){
        st = _st;  id = _id;
    }
    bool operator < (const Node &nd) const{
        if(st == nd.st) return id < nd.id;
        return st < nd.st;
    }
}seg[MAXN];

queue<int> rCol, rSeg;

int main(){
    scanf("%d", &T);
    while(T --){
        scanf("%d%d", &n, &k);
        while(sz(rCol)) rCol.pop();
        while(sz(rSeg)) rSeg.pop();
        for(int i = 1; i <= n; i ++){
            int l, r;
            scanf("%d%d", &l, &r);
            col[i] = 0;
            seg[i] = Node(l, i);
            seg[i + n] = Node(r, -i);
        }
        if(n < k){
            printf("0\n1");
            for(int i = 2; i <= n; i ++)
                printf(" 1");
            printf("\n");
            continue;
        }
        sort(seg + 1, seg + 2*n + 1);
        for(int i = 1; i <= k; i ++)
            rCol.push(i);
        ll ans = 0;
        for(int i = 1; i <= 2*n; i ++){
            if(sz(rCol) == 0)
                ans += seg[i].st - seg[i - 1].st;
            if(seg[i].id > 0){
                if(sz(rCol)){
                    int tc = rCol.front();
                    col[seg[i].id] = tc;
```

```
                rCol.pop();
            }
            else
                rSeg.push(seg[i].id);
        }
        else{
            if(col[- seg[i].id])
                rCol.push(col[- seg[i].id]);
            else
                col[- seg[i].id] = 1;
        }
        while(sz(rCol) && sz(rSeg)){
            int tc = rCol.front();
            int ts = rSeg.front();
            rSeg.pop();
            if(col[ts]) continue;
            col[ts] = tc;
            rCol.pop();
        }
    }
    printf("%I64d\n", ans);
    for(int i = 1; i <= n; i ++)
        printf("%d%c", col[i], i == n ? '\n' : ' ');
}
return 0;
}
```

# GYM 101981M exkmp+manacher （计 17-5 陈鑫）

## 一、题号与题目名称

GYM 101981M Mediocre String Problem

## 二、题意简述

给一个串 s 和串 t，求 s 的子串 s[i]~s[j]与 t 的前缀 t[1]~t[k]能拼成回文串的(i，j，k)三元组数，其中 j－i+1>k。

1<=|t|<|s|<=1e6。

## 三、主要知识点及其运用

本题涉及 exkmp 和 manacher 算法的应用。

## 四、完整解题思路描述

s[i]~s[j]与 t[1]~t[k]能拼成回文串，显然要满足：

1、s[i+k-1]~s[i]（注意顺序）与 t[1]~t[k]对应相等；

2、s[i+k]~s[j]是回文串。

我们将 s 串翻转，子问题 1 即求 s 的每个后缀与 t 的最长公共前缀，可用 exkmp 解决。

设翻转后 s 的后缀 s[i]~s[n]与 t 的最长公共前缀为 extend[i]，接下来只需求出以 s[i-1]为结尾的回文串数 c[i-1]，答案即为∑extend[i]*c[i-1]。

用 manacher 跑出每个中心 i 的最长回文半径 r[i]，差分后（c[i]++，c[i+r[i]]--）做前缀和即为所求 c[i]。

总时间复杂度为 O(|s|+|t|)。

## 五、技巧与坑点

翻转会使“以某一位开始向前形成的子串”转化为“以某一位开始的后缀”以简化某些字符串问题；

在对最长回文半径 r[i]做差分时，注意要对奇偶回文中心分别讨论。

## 六、参考文献与学习资料

https://blog.csdn.net/dyx404514/article/details/41831947

https://www.cnblogs.com/love-yh/p/7072161.html

## 七、完整代码

```
#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
const int MAXN = 1000010;

int n, m, c[MAXN];
int nxt[MAXN], extend[MAXN];
char s[MAXN], t[MAXN];
char Ma[MAXN << 1];
int Mp[MAXN << 1];

void pre_EKMP(char x[], int m, int nxt[]){
```

```
    nxt[0] = m;
    int j = 0;
    while(j + 1 < m && x[j] == x[j + 1]) j ++;
    nxt[1] = j;
    int k = 1;
    for(int i = 2; i < m; i ++){
        int p = nxt[k] + k - 1;
        int L = nxt[i - k];
        if(i + L < p + 1) nxt[i] = L;
        else{
            j = max(0, p - i + 1);
            while(i + j < m && x[i + j] == x[j]) j ++;
            nxt[i] = j;
            k = i;
        }
    }
}

void EKMP(char x[], int m, char y[], int n, int nxt[], int extend[]){
    pre_EKMP(x, m, nxt);
    int j = 0;
    while(j < n && j < m && x[j] == y[j]) j ++;
    extend[0] = j;
    int k = 0;
    for(int i = 1; i < n; i ++){
        int p = extend[k] + k - 1;
        int L = nxt[i - k];
        if(i + L < p + 1) extend[i] = L;
        else{
            j = max(0, p - i + 1);
            while(i + j < n && j < m && y[i + j] == x[j]) j ++;
            extend[i] = j;
            k = i;
        }
    }
}

void Manacher(char s[], int len){
    int l = 0;
    Ma[l ++] = '$';
    Ma[l ++] = '#';
    for(int i = 0; i < len; i ++){
        Ma[l ++] = s[i];
        Ma[l ++] = '#';
    }
```

```
    Ma[l] = 0;
    int mx = 0, id = 0;
    for(int i = 0; i < l; i ++){
        Mp[i] = mx > i ? min(Mp[2*id - i], mx - i) : 1;
        while(Ma[i + Mp[i]] == Ma[i - Mp[i]]) Mp[i] ++;
        if(i + Mp[i] > mx){
            mx = i + Mp[i];
            id = i;
        }
    }
}

int main(){
    scanf("%s", s);
    n = strlen(s);
    reverse(s, s + n);
    scanf("%s", t);
    m = strlen(t);
    EKMP(t, m, s, n, nxt, extend);
    Manacher(s, n);
    for(int i = 2; i < 2*n + 2; i ++){
        int r = Mp[i] / 2;
        if(i & 1){
            c[i/2 + 1] ++;
            c[i/2 + r + 1] --;
        }
        else{
            c[i/2] ++;
            c[i/2 + r] --;
        }
    }
    for(int i = 1; i <= n; i ++)
        c[i] += c[i - 1];
    ll ans = 0;
    for(int i = 1; i < n; i ++)
        ans += 1LL*extend[i]*c[i];
    cout << ans;
    return 0;
}
```

# 计蒜客 39272 树链剖分+线段树 （计 17-5 陈鑫）

## 一、题号与题目名称

计蒜客 39272 Tree

## 二、题意简述

有一棵 n 个节点的树，点有点权，有以下三种操作：

1 s t：将从节点 1 到 s 的路径上的点的点权|=t；

2 s t：将从节点 1 到 s 的路径上的点的点权&=t；

3 s t：询问节点 1 到 s 的路径上的点权异或和是否等于 t。

1<=n<=1e5。

## 三、主要知识点及其运用

本题考察树链剖分的基础及线段树的构造。

## 四、完整解题思路描述

直接树剖后用线段树维护区间异或和，发现异或和有如下性质：

若该位为 1，说明区间内有奇数个该位为 1 的点；

若该位为 0，说明区间内有偶数个该位为 1 的点。

or 一个数之后，会将原本为 0 的位变为 1：

若区间长度为奇数：

若异或和该位为 1，则有偶数个 0 变为 1，该位仍为 1；

若异或和该位为 0，则有奇数个 0 变为 1，该位变为 1；

等同于直接 or 该数。

若区间长度为偶数：

若异或和该位为 1，则有奇数个 0 变为 1，该位变为 0；

若异或和该位为 0，则有偶数个 0 变为 1，该位仍为 0。

等同于 and 该数的反。

and 一个数之后，会将原本为 1 的位变为 0，如上分析后发现，无论区间长度为奇数或偶数，都等同于直接 and 该数。

综上，设区间 or 的数为 u，and 的数为 v，异或和为 sum，

则当区间长度为奇数时，sum=sum&v|u；

长度为偶数时，sum=sum&v&(~u)。

而先 and 再 or 和先 or 再 and 是有区别的，所以当其中一个标记修改时，另一个标记都要继承其修改，以确保两者是等价的。

总时间复杂度 O(nlognlogn)。

## 五、技巧与坑点

因为位运算不满足交换律，所以当其中一个标记修改时，另一个标记要继承其修改。

## 六、参考文献与学习资料

https://www.cnblogs.com/ivanovcraft/p/9019090.html

https://www.cnblogs.com/xyq0220/p/10945971.html

## 七、完整代码

```
#include<bits/stdc++.h>
```

```
#define sz(x) ((int)x.size())
#define mp(x, y) make_pair(x, y)
using namespace std;

typedef long long ll;
const int MAXN = 100010;
const int inf = (1 << 30) - 1;

int n, m;
int cnt, tid[MAXN], rnk[MAXN];
int faz[MAXN], dep[MAXN], val[MAXN];
int siz[MAXN], son[MAXN], top[MAXN];
int tot, head[MAXN];
int sum[MAXN << 2], lazAd[MAXN << 2], lazOr[MAXN << 2];
#define lson (p << 1)
#define rson (p << 1 | 1)
#define mid ((l + r) >> 1)

struct Edge{
    int to, next;
}edges[MAXN << 1];

void addEdge(int x, int y){
    edges[++ tot].to = y;
    edges[tot].next = head[x];
    head[x] = tot;
}

void dfs1(int x, int fa){
    faz[x] = fa;
    dep[x] = dep[fa] + 1;
    siz[x] = 1;
    for(int i = head[x]; i; i = edges[i].next){
        int y = edges[i].to;
        if(y == fa) continue;
        dfs1(y, x);
        siz[x] += siz[y];
        if(siz[y] > siz[son[x]])
            son[x] = y;
    }
}

void dfs2(int x, int tp){
    top[x] = tp;
    tid[x] = ++ cnt;
```

```
    rnk[cnt] = x;
    if(son[x]) dfs2(son[x], tp);
    for(int i = head[x]; i; i = edges[i].next){
        int y = edges[i].to;
        if(y == faz[x] || y == son[x])
            continue;
        dfs2(y, y);
    }
}

void build(int p, int l, int r){
    lazAd[p] = inf;
    lazOr[p] = 0;
    if(l == r){
        sum[p] = val[rnk[l]];
        return;
    }
    build(lson, l, mid);
    build(rson, mid + 1, r);
    sum[p] = sum[lson] ^ sum[rson];
}

void updLaz(int p, int f, int v){
    if(f) lazAd[p] &= v, lazOr[p] &= v;
    else lazOr[p] |= v, lazAd[p] |= v;
}

void updSum(int p, int l, int r){
    if((r - l + 1) & 1) sum[p] = ((sum[p] & lazAd[p]) | lazOr[p]);
    else sum[p] = (sum[p] & lazAd[p] & (inf ^ lazOr[p]));
}

void push_down(int p, int l, int r){
    if(lazAd[p] != inf){
        updLaz(lson, 1, lazAd[p]);
        updLaz(rson, 1, lazAd[p]);
        lazAd[p] = inf;
    }
    if(lazOr[p]){
        updLaz(lson, 0, lazOr[p]);
        updLaz(rson, 0, lazOr[p]);
        lazOr[p] = 0;
    }
    updSum(lson, l, mid);
    updSum(rson, mid + 1, r);
```

```
}

void upd(int p, int L, int R, int f, int v, int l, int r){
    if(L <= l && r <= R){
        updLaz(p, f, v);
        updSum(p, l, r);
        return;
    }
    push_down(p, l, r);
    if(L <= mid) upd(lson, L, R, f, v, l, mid);
    if( R > mid) upd(rson, L, R, f, v, mid + 1, r);
    sum[p] = sum[lson] ^ sum[rson];
}

int ask(int p, int L, int R, int l, int r){
    if(L <= l && r <= R)
        return sum[p];
    push_down(p, l, r);
    int ans = 0;
    if(L <= mid) ans ^= ask(lson, L, R, l, mid);
    if( R > mid) ans ^= ask(rson, L, R, mid + 1, r);
    return ans;
}

void upd_path(int x, int y, int f, int v){
    while(top[x] != top[y]){
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        upd(1, tid[top[x]], tid[x], f, v, 1, n);
        x = faz[top[x]];
    }
    if(dep[x] > dep[y]) swap(x, y);
    upd(1, tid[x], tid[y], f, v, 1, n);
}

int ask_path(int x, int y){
    int ans = 0;
    while(top[x] != top[y]){
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        ans ^= ask(1, tid[top[x]], tid[x], 1, n);
        x = faz[top[x]];
    }
    if(dep[x] > dep[y]) swap(x, y);
    ans ^= ask(1, tid[x], tid[y], 1, n);
    return ans;
}
```

```
int main(){
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)
        scanf("%d", val + i);
    for(int i = 1; i < n; i ++){
        int u, v;
        scanf("%d%d", &u, &v);
        addEdge(u, v);
        addEdge(v, u);
    }
    dfs1(1, 0);
    dfs2(1, 1);
    build(1, 1, n);
    while(m --){
        int opt, s, t;
        scanf("%d%d%d", &opt, &s, &t);
        if(opt == 1)
            upd_path(1, s, 0, t);
        else if(opt == 2)
            upd_path(1, s, 1, t);
        else
            puts(ask_path(1, s) == t ? "NO" : "YES");
    }
    return 0;
}
```

# POJ 1401 线段相交 （计 17-8 江绪桢）

## 一、题号与题目名称

POJ 1401 Intersection

## 二、题意简述

给定一个矩形的左上角和又下角坐标，再给定一条线段的两个端点的坐标。判断线段与矩形有没有公共点。

## 三、主要知识点及其运用

向量叉积，判断线段相交（快速排斥和跨立实验）

## 四、完整解题思路描述

枚举矩形的每条边，使用线段相交模板可以判断是否与线段有交点。然后再通过判断线段的端点是否在矩形内或矩形的边界上来判断线段是否在矩形内。

## 四、技巧与坑点

本题没有说明给定左上角和右下角点时的顺序。所以需要根据坐标先行判断。还有当线段完全在矩形内部时线段与矩形的边框自然没有交点，但这也是线段与矩形有公共点的一种合法情况。

## 六、参考文献与学习资料

http://poj.org/problem?id=1410

## 七、完整代码

```
#include <iostream>
#include <cstdio>
using namespace std;

struct point
{
    int x;
    int y;
    point(point a,point b)
    {
        this->x = b.x - a.x;
        this->y = b.y - a.y;
    }
    point(){}
};
int cross(point a,point b)
{
    return a.x*b.y - b.x*a.y;
}
bool touch(point abegin,point aend,point bbegin,point bend)
{
    if( min(abegin.x,aend.x)<=max(bbegin.x,bend.x)&&
```

```
            min(bbegin.x,bend.x)<=max(abegin.x,aend.x)&&
            min(abegin.y,aend.y)<=max(bbegin.y,bend.y)&&
            min(bbegin.y,bend.y)<=max(abegin.y,aend.y)
           )
        {
            point a(bbegin,abegin);
            point b(bbegin,bend);
            point c(bbegin,aend);
            int m = cross(a,b)*cross(c,b);///<=0
            a = point(aend,bbegin);
            b = point(aend,abegin);
            c = point(aend,bend);
            int mm = cross(a,b)*cross(c,b);///<=0
            if(m<=0&&mm<=0)
            {
                return 1;
            }
        }
        return 0;
}
bool inrectangle(point p,point a,point b)
{
        if(a.x<=p.x&&p.x<=b.x&&b.y<=p.y&&p.y<=a.y)return 1;
        return 0;
}
int main()
{
        int n;
        scanf("%d",&n);
        for(int i = 1;i<=n;i++)
        {
            point testbegin,testend;
            scanf("%d%d%d%d",&testbegin.x,&testbegin.y,&testend.x,&testend.y);
            point a[4];
            scanf("%d%d%d%d",&a[0].x,&a[0].y,&a[2].x,&a[2].y);
            if(a[0].x>a[2].x) swap(a[0].x,a[2].x);
            if(a[0].y<a[2].y) swap(a[0].y,a[2].y);
            a[1].x = a[0].x;a[1].y = a[2].y;
            a[3].x = a[2].x;a[3].y = a[0].y;
            bool ok = 0;
            for(int i = 0;i<4;i++)
            {
                if(touch(a[i],a[(i+1)%4],testbegin,testend))
                {
                    ok = 1;
```

```
                }
            }
            if(inrectangle(testbegin,a[0],a[2])||inrectangle(testend,a[0],a[2]))
            {
                ok = 1;
            }
            if(ok)
            {
                printf("T\n");
            }
            else
            {
                printf("F\n");
            }
        }
        return 0;
}
```

# POJ 1156 基础线段几何+最短路（计算机 17-8 江绪桢）

## 一、题号与题目名称

POJ 1156 The Doors

## 二、题意简述

在一个 10*10 大小的空间内有不超过 18 个墙，每个墙上有两个门。问从起点（0，5）到终点（10，5）的最短路径是多少。

## 三、主要知识点及其运用

判断线段是否绝对相交（必须交叉到两边出头成为“X”形才认为绝对相交），根据题意建图，最短路

## 五、完整解题思路描述

枚举每一个门的上下两个端点。将它们连线。每个门的两个端点分别抽象为图中的一个节点。建图时判断位于两个门之间的墙会不会挡住门。如果会挡住，那么认为这两个点之间的权值无穷大。如果不会挡住，那么用这两点之间的距离作为图中这条边的权值。

## 五、技巧与坑点

枚举端点时，同一个墙上的两个门之间即使没有被挡住也不能相互联通，因为这不符合问题的定义。可以使用平面几何的知识大幅简化本题中关于线段相交的判断。

## 六、参考文献与学习资料

http://poj.org/problem?id=1556

## 七、完整代码

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>
#include<iostream>
using namespace std;
const double INF = 1000000;
const double eps = 1e-8;
const int maxn = 1e3+3;
struct point
{
    double x,y;
    point() {}
    point(point a,point b)
    {
        this->x = b.x - a.x;
        this->y = b.y - a.y;
    }
    point(double a,double b)
    {
        x = a;
```

```
            y = b;
        }
} p[maxn];
double cross(point a,point b)
{
    return(a.x * b.y - a.y * b.x);
}
double dist(point a,point b)
{
    return sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
}
bool judge(point a,point b,point m,point n)
{
    if( min(a.x,b.x)<=max(m.x,n.x)&&
            min(m.x,n.x)<=max(a.x,b.x)&&
            min(a.y,b.y)<=max(m.y,n.y)&&
            min(m.y,n.y)<=max(a.y,b.y)
       )
    {
        int cro1 = cross(point(a,m),point(a,b))*cross(point(a,n),point(a,b));
        int cro2 = cross(point(m,a),point(m,n))*cross(point(m,b),point(m,n));
        if(cro1<-eps&&cro2<-eps)
        {
            return 1;
        }
    }
    return 0;
}

double d[maxn][maxn];

int main()
{
    int n;
    while(scanf("%d",&n)&&n!=-1)
    {
        for(int i = 0; i<n; i++)
        {
            double x;
            scanf("%lf",&x);
            for(int j = 1; j<=4; j++)
            {
                p[i*4+j].x = x;
                scanf("%lf",&p[i*4+j].y);
            }
```

```
}
p[0] = point(0,5);
p[4*n+1] = point(10,5);
memset(d,0,sizeof(d));
for (int i = 0 ; i <= 4 * n + 1 ; i++)
{
    for (int j = i + 1 ; j <= 4 * n + 1 ; j++)
    {
        int l = (i - 1) / 4, r = (j - 1) / 4;
        if (i == 0)
            l = -1;
        int flag = 1;
        if(l == r)
        {
            d[i][j] = d[j][i] = INF;
        }
        else
            for (int k = l + 1 ; k < r ; k++)
            {
                if (judge(p[4 * k + 1], point(p[4 * k + 1].x, 0), p[i], p[j]))
                {
                    flag = 0;
                    break;
                }
                if (judge(p[4 * k + 2], p[4 * k + 3], p[i], p[j]))
                {
                    flag = 0;
                    break;
                }
                if (judge(p[4 * k + 4], point(p[4 * k + 4].x, 10), p[i], p[j]))
                {
                    flag = 0;
                    break;
                }
            }
        d[i][j] = d[j][i] = (flag ?  dist(p[i], p[j]) : INF) ;
    }
}
for(int i = 0; i<=4*n+1; i++)
{
    for(int j = 0; j<=4*n+1; j++)
    {
        for(int k = 0; k<=4*n+1; k++)
        {
            d[j][k] = min(d[j][k],d[j][i]+d[i][k]);
```

```
                    }
                }
            }
            printf("%.2f\n",d[0][4*n+1]);
        }
        return 0;
}
```

# 大视野在线测评 1069 点集中求四个点框出的最大面积(计 17-8 江绪桢)

## 一、题号与题目名称

Lydsy 1069 [SCOI2007]最大土地面积

## 二、题意简述

在某块平面土地上有 N 个点，你可以选择其中的任意四个点，将这片土地围起来，当然，你希望这四个点围成的多边形面积最大。(N<=2000)

## 三、主要知识点及其运用

凸包、旋转卡壳算法

## 四、完整解题思路描述

在点集中选择四个点使得它们围成的面积最大。通过直观感受和凸包的定义可以知道这四个点一定要在点集的凸包上选取。首先对点集求凸包，然后在凸包上枚举两个点作为待寻找的四边形的对角线，然后使用旋转卡壳求面积的思想寻找使得对角线两侧的两个三角形面积最大的三角形顶点。维护总面积的最大值即可。

## 五、技巧与坑点

使用旋转卡壳算法时枚举的对角线的两个端点应当至少间隔一个点以容纳三角形的顶点。三角形顶点在旋转时不应越过三角形的底边。由于本题中间过程全部是 double 类型，所以要注意精度保证。需要使用 eps 来判断任意数与 0 的关系。

## 六、参考文献与学习资料

https://www.lydsy.com/JudgeOnline/problem.php?id=1069

## 七、完整代码

```
#include<iostream>
#include<cstdlib>
#include<cstdio>
#include<cmath>
#include<cstring>
#include<algorithm>

#define maxn 2005
#define eps 1e-4
using namespace std;
int n,top;
double ans = 0;
struct point
{
    double x,y;
    point() {}
    point(double a,double b)
    {
        x = a;
```

```
        y = b;
    }
    point(point a,point b)
    {
        x = b.x - a.x;
        y = b.y - a.y;
    }
} p[maxn],s[maxn];
double sign(double a)
{
    if (fabs(a)<=eps)
        return 0;
    else
        return a>0? 1 : -1;
}
double cross(point a,point b)
{
    double s = (a.x*b.y - a.y*b.x);
    return s;
}
double dis(point a,point b)
{
    return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
}
point operator -(point a,point b)
{
    return (point)
    {
        a.x-b.x,a.y-b.y
    };
}
double operator *(point a,point b)
{
    return a.x*b.y-a.y*b.x;
}

bool cmp(point a,point b)
{
    double x = cross(point(p[1],a),point(p[1],b));
    if(x>0)
        return 1;
    if(x == 0)
        return dis(p[1],a)<dis(p[1],b);
}
int main()
```

```
{
    scanf("%d",&n);
    for(int i = 1; i<=n; i++)
    {
        double aa,bb;
        scanf("%lf%lf",&aa,&bb);
        p[i] = point(aa,bb);
    }
    int t=1;
    for(int i = 2; i<=n; i++)
    {
        if (p[i].y<p[t].y||(p[i].y==p[t].y&&p[i].x<p[t].x))
        {
            t=i;
        }
    }
    swap(p[t],p[1]);
    sort(p+2,p+n+1,cmp);
    s[1]=p[1];
    s[2]=p[2];
    top = 2;
    for(int i = 3; i<=n; i++)
    {
        while (top>1&&cross(point(s[top-1],p[i]),point(s[top-1],s[top]))>=0)
            top--;
        s[++top]=p[i];
    }

    int i,j;
    s[top+1]=s[1];
    for(int x = 1; x<=top-2; x++)
    {
        i=(x+1)%top;
        j=(x+3)%top;
        for(int y = x+2; y<=top; y++)
        {
            while (i%top+1!=y&&sign((s[i+1]-s[x])*(s[y]-s[x])-(s[i]-s[x])*(s[y]-s[x])) == 1)
            //while
(i%top+1!=y&&sign(cross(point(s[i+1],s[x]),point(s[y],s[x]))-cross(point(s[i],s[x]),point(s[y],s[x]))) == 1)
                i=i%top+1;
            //while
(j%top+1!=x&&sign(cross(point(s[y],s[x]),point(s[j+1],s[x]))-cross(point(s[y],s[x]),point(s[j]-s[x]))) == 1)
            while (j%top+1!=x&&sign((s[y]-s[x])*(s[j+1]-s[x])-(s[y]-s[x])*(s[j]-s[x])) == 1)
                j=j%top+1;
            ans=max(ans,(s[i]-s[x])*(s[y]-s[x])+(s[y]-s[x])*(s[j]-s[x]));
```

```
            }
        }
        printf("%.3f\n",ans/2.0);
        return 0;
}
```

# HDU 4465 期望+精度控制 (网络 17-3 顾佳昊)

## 一、题号与题目名称

HDU 4465 Candy

## 二、题意简述

有两个盒子，一开始每个里面都有 n 个糖果。每天选择一个盒子。选第一个盒子的概率为 p,选第二个的概率为（1-p）。对于所选择的盒子，如果还有糖果，他会吃其中一个。

有一天，当打开一个盒子时，他发现没有糖果了。问另一个盒子里剩下的预期糖果数量。答案精确到小数点后四位。

1≤n≤2e5

0≤p≤1，小数点后 6 位

## 三、主要知识点及其运用

概率的计算方法以及精度控制方法。

## 四、完整解题思路描述

首先很自然就能得出概率 dp 的做法。设 dp[i][j]为第一个盒子中剩余糖果数为 i，第二个盒子中剩余糖果数为 j 的概率，转移方程 dp[i-1][j]+=dp[i][j]*p,dp[i][j-1]+=dp[i][j]*(1-p)，初始状态 dp[n][n]=1，目标为$\sum_{i=1}^{n} i * dp[0][i]$ + $\sum_{i=1}^{n} i * dp[i][0]$。可是这道题的范围是 2e5，时间和空间都不够，寻求公式做法。

令第一个盒子取光时第二个盒子的糖果的期望为 solve(p)，

solve(p)=Σ(n-i)*C(n+i,n)*(p^(n+1))*((1-p)^i) 注意他打开盒子时发现没了，所以是 p 的(n+1)次方

第二个盒子取光时第一个盒子的糖果的期望为 solve(1-p),ans=solve(p)+solve(1-p)当 p=0 或 p=1 时需要特判，答案为 n。

然而这样做虽然复杂度能通过，但是会 WA。原因是题中的精度要求很高，小数的高次方丢失精度太严重了。需要取对数进行计算。

solve(p)=Σ(n-i)*exp(log(C(n+i,n)*(p^(n+1))*((1-p)^i)))=Σ(n-i)*exp(log(C(n+i,n))+(n+1)*log(p)+i*log(1-p))

## 五、技巧与坑点

概率的计算公式容易想到，主要难点是如何减少精度损失。取对数进行计算是一种不错的方法，以后遇到类似的问题时可以借鉴。

## 六、参考文献与学习资料

http://acm.hdu.edu.cn/showproblem.php?pid=4465

## 七、完整代码

```
#include <bits/stdc++.h>
using namespace std;
#define LB long double
LB f[400005];
int n;
```

```
void init(){
    f[1]=0;
    for (int i=2;i<400005;i++) f[i]=f[i-1]+log(i);
}

double solve(LB p){
    double sum=0;
    for (int i=0;i<n;i++){
        sum+=(n-i)*exp((n+1)*log(p)+i*log(1-p)+(f[n+i]-f[n]-f[i]));
    }
    return sum;
}

int main(){
    init();
    LB p;
    int _=1;
    while (cin>>n>>p){
        double ans;
        if (p==0 || p==1) ans=n;
        else ans=solve(p)+solve(1-p);
        printf("Case %d: %.6lf\n",_++,ans);
    }
}
```

# HDU 4474 bfs (网络 17-3 顾佳昊)

## 一、题号与题目名称

HDU 4474 Yet Another Multiple Problem

## 二、题意简述

给一个数字 n 和 m 个 0-9 的数字，问 n 的最小多少倍中没有 m 中任意一个数字。

n ≤ 1e4

## 三、主要知识点及其运用

## 四、完整解题思路描述

从小到大枚举由合法的数字构成的整数，直到出现能被 n 整除或者队列为空，所以 bfs 就行了

如果有两个整数%n==x,那么显然后面那个可以 contine 了，因为增加一位 y 的话余数都等于(x*10+y)%n,前面的那个数可以覆盖后面的数的所有后续状态，并且更小。

## 五、技巧与坑点

从题目入手后有两个方向，即找符合要求的数字判断是否为 n 的整数倍，或者找 n 的整数倍判断是否符合要求。但是显然后一种没有办法进行剪枝，所以要使用第一种。

## 六、参考文献与学习资料

http://acm.hdu.edu.cn/showproblem.php?pid=4474

## 七、完整代码

```
#include <bits/stdc++.h>
using namespace std;
#define LL long long
bool cantuse[20];
vector<char> a;
queue<string> que;
bool vis[100005];
LL n,m;
void init(){
	memset(cantuse,0,sizeof(cantuse));
	a.clear();
	memset(vis,0,sizeof(vis));
	while (que.size()) que.pop();
}
int mod(string s,int md){
	int ret=0;
	for (int i=0;i<s.length();i++){
		ret=(ret*10+s[i]-'0')%md;
	}
	return ret;
}
string bfs(){
```

```
    while (que.size()){
        string k=que.front();
        que.pop();
        int md=mod(k,n);
        if (vis[md]) continue;
        vis[md]=1;
        if (md==0) return k;
        for (LL i=0;i<a.size();i++){
            string tmp=k+a[i];
            que.push(tmp);
        }
    }
    return "-1";
}

int main(){
    int _=1;
    while (cin>>n>>m){
        init();
        for (LL i=0;i<m;i++){
            LL x;
            cin>>x;
            cantuse[x]=1;
        }
        for (LL i=0;i<10;i++){
            if (!cantuse[i]) {
                if (i!=0){
                    string tmp="";
                    tmp+=char(i+'0');
                    que.push(tmp);
                }
                a.push_back(char(i+'0'));
            }
        }
        printf("Case %d: ",_++);
        cout<<bfs()<<endl;
    }
}
```

# ZOJ 4114 dp （网络 17-3 顾佳昊）

## 一、题号与题目名称

ZOJ 4114 Flipping Game

## 二、题意简述

给两个长度为 n 的 01 串，每次挑选第一个串其中的 m 位翻转（0 变成 1,1 变成 0,m 位不用连续）问经过 k 次的后得到第二个串的方案数。

n,m,k ≤ 100

## 三、主要知识点及其运用

DP

## 四、完整解题思路描述

若两个串的某一位相同，则该位经过了偶数次翻转，否则经过了奇数次的翻转。

dp[i][j]表示经过了 i 轮翻转后奇数位数等于 j 的方案数，偶数的位数等于 n-j。从 0-m 枚举 k，表示 j 个奇数位中有 k 个翻转成了偶数，(n-j)个偶数位中有(m-k)位翻转成了奇数位，所以 dp[i+1][j-k+(m-k)]+=dp[i][j]*C[j][k]*C[n-j][m-k] ;

## 五、技巧与坑点

看上去这题似乎有两种正确的初始值设置方法：dp[0][0]=1,求 dp[k][num_odd] OR dp[0][num_odd]=1,求 dp[k][0]。可是经过试验发现两种的答案不一样。仔细想想，我们的 dp 过程中只记录了奇数位的个数，并没有具体统计哪位是奇数，因此我们最后得到的 dp[k][num_odd]是所有的奇数位数等于答案的方案数，可是显然他们无法和要求的第二个串完全匹配。因此把初始值 dp[0][num_odd]设成 1 求 dp[k][0]才是正确的。

## 六、参考文献与学习资料

http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=4114

## 七、完整代码

```
#include <bits/stdc++.h>
using namespace std;
#define LL long long
LL dp[105][105],C[105][105];
const int mod= 998244353;
void init(){
    for (int i=0;i<=100;i++) C[i][0]=1;
    for (int i=1;i<=100;i++){
        for (int j=1;j<=i;j++){
            C[i][j]=(C[i-1][j-1]+C[i-1][j])%mod;
        }
    }
}
int main(){
```

```
    init();
    int T;
    cin>>T;
    while (T--){
        int n,q,m;
        cin>>n>>q>>m;
        string s1,s2;
        cin>>s1>>s2;
        int odd=0,even=0;
        for (int i=0;i<n;i++){
            if (s1[i]==s2[i]) even++;
            else odd++;
        }
        memset(dp,0,sizeof(dp));
        dp[0][odd]=1;
        for (int i=0;i<=q;i++){
            for (int j=0;j<=n;j++){
                for (int k=0;k<=m;k++){
                    if (j>=k && (n-j)>=(m-k)){
                        dp[i+1][j-k+(m-k)]+=dp[i][j]*C[j][k] %mod *C[n-j][m-k] %mod;
                        dp[i+1][j-k+(m-k)]%=mod;
                    }
                }
            }
        }
        cout<<dp[q][0]<<endl;
    }
}
```

# Codeforces 505D 构造+拓扑+bfs （网络 17-3 顾佳昊）

## 一、题号与题目名称

CodeForces 505D Mr. Kitayuta’s Technology

## 二、题意简述

给 n 个点，然后给出 m 个条件 a、b，表示 a 可到达 b，求最少需要建多少条有向边才能满足所有条件。

## 三、主要知识点及其运用

构造、拓扑、bfs

## 四、完整解题思路描述

首先对于每个单独的连通块（把边都看成双向边），我们可以把它拉成一条链或一个环。链的情况如样例一：1->2->3->4。这样前面的点都可以到达相对位置后面的点，且总边数最小，为点数-1。但是若该联通块中存在环，则无法拉成一根链，如样例二，但是我们可以通过加一条链尾到链首的边解决所有的环的冲突。在样例二中为 1->2->3->4->1。这样总边数最小，为点数。
所以先把边看成双向边，对每个连通块染色，然后拓扑判断是否存在环，若存在，答案+=连通块中的点数；否则+= 连通块中的点数-1。

## 五、技巧与坑点

需要想到链和环分别应该如何构造。

## 六、参考文献与学习资料

http://codeforces.com/contest/505/problem/D

## 七、完整代码

```
#include <bits//stdc++.h>
using namespace std;
struct EDGE{
    int to,next;
}edge[500005],dedge[500005];
int head[100005],dhead[100005],cnt,dcnt;
void addedge(int fr,int to,int k){
    if (!k){
        edge[cnt]={to,head[fr]};
        head[fr]=cnt++;
    }
    else{
        dedge[dcnt]={to,dhead[fr]};
        dhead[fr]=dcnt++;
        dedge[dcnt]={fr,dhead[to]};
        dhead[to]=dcnt++;
    }
}
int ans,color[100005],n,m,in[100005];
void init(){
```

```
    memset(color,-1,sizeof(color));
    memset(head,-1,sizeof(head));
    memset(dhead,-1,sizeof(dhead));
    cnt=dcnt=0;
}
queue<int> q2;
vector<int> pp;
void bfs(int q){
    while (q2.size()) q2.pop();
    pp.clear();
    queue<int> que;
    que.push(q);
    while (que.size()){
        int k=que.front();
        que.pop();
        pp.push_back(k);
        if (!in[k]) q2.push(k);
        for (int i=dhead[k];i!=-1;i=dedge[i].next){
            int v=dedge[i].to;
            if (color[v]!=-1) continue;
            color[v]=color[k];
            que.push(v);
        }
    }
}
void tuopu(int q){
    if (pp.size()<=1) return;
    while (q2.size()){
        int k=q2.front();
        q2.pop();
        for (int i=head[k];i!=-1;i=edge[i].next){
            int v=edge[i].to;
            in[v]--;
            if (!in[v]) q2.push(v);
        }
    }
    bool key=0;
    for (int i=0;i<pp.size();i++){
        if (in[pp[i]]){
            key=1;
            break;
        }
    }
    if (key) ans+=pp.size();
    else ans+=pp.size()-1;
```

```
}

int main(){
	init();
	cin>>n>>m;
	for (int i=0;i<m;i++){
		int u,v;
		cin>>u>>v;
		addedge(u,v,0);
		addedge(u,v,1);
		in[v]++;
	}
	int num_color=0;
	for (int i=1;i<=n;i++){
		if (color[i]==-1){
			color[i]=num_color++;
			bfs(i);
			tuopu(color[i]);
		}
	}
	cout<<ans<<endl;
}
```

# 牛客训练赛 49-E 筱玛爱游戏 线性基(计 17-8 于博文)

## 一、题号与题目名称

牛客训练赛 49-E 筱玛爱游戏

## 二、题意简述

筱玛是一个热爱游戏的好筱玛。最近，筱玛和马爷在玩这样一种游戏：
首先，桌面上一共有 n 个数。
两个人轮流从桌面上取走一个数，并把这个数放入集合中。
如果在某次操作结束后，集合中存在一个异或和为 0 的**非空**子集，那么进行这次操作的人输。
如果全部取完，则最后操作的人赢。
筱玛和马爷都聪明绝顶，他们都会按照最优策略进行游戏。
马爷作为筱玛的爷爷，决定让筱玛选择先手还是后手。
筱玛为了稳操胜券，想提前知道对于当前的游戏，是先手必胜还是后手必胜。
筱玛想考考你，让你帮他解决这个问题。

## 三、主要知识点及其运用

线性基

## 四、完整解题思路描述

线性基的特点就是可以异或出且仅能异或出原集合张成中的所有元素，所以只要判断最后他的线性基中有多少个元素即可

## 五、参考文献及学习资料

线性基：https://oi.men.ci/linear-basis-notes/
题目：https://ac.nowcoder.com/acm/contest/946/E

## 六、完整代码

```
#include <bits/stdc++.h>
using namespace std;

long long int zu[65],ji[65];
int q,h;
int main(){
   memset(zu,0,sizeof(zu));
   memset(ji,0,sizeof(ji));
   long long int t,n,m,zz,shu;ji[0]=1;
   for(int i=1;i<=62;i++){ji[i]=ji[i-1]*2;}
   scanf("%lld",&n);m=0;
   for(int i=1;i<=n;i++){
      scanf("%lld",&shu);zz=-1;
      for(int j=62;j>=0;j--){
         if(shu==0)break;
         if((shu&ji[j])!=0){
            if(zu[j]!=0)shu=(shu^zu[j]);
            else{zz=j;break;}
```

```
            }
        }
        if(zz==-1)continue;
        for(int j=zz-1;j>=0;j--)if((shu&ji[j])!=0)shu=(shu^zu[j]);
        for(int j=zz+1;j<=62;j++){
            if((zu[j]&ji[zz])!=0)zu[j]=(zu[j]^shu);
        }m++;zu[zz]=shu;
        ///for(int j=0;j<=10;j++)printf("%d ",zu[j]);puts("");
    }
    if(m%2==0)printf("Second\n");
    else printf("First\n");
}
```

# 计蒜客 A1998 分块+前缀和+二分 (计 17-8 于博文)

## 一、题号与题目名称

计蒜客 A1998 分块+前缀和+二分(计 17-8 于博文

## 二、题意简述

给一个树，root 为 1，n 个节点（不一定是二叉树）。一开始所有点权为 0，然后给出两种操作：

1.给深度为 L 的所有节点点权加 x

2.查询某个节点 x 作为根的子树所有点权和

n 个节点 q 个操作，都是 1e5

## 三、主要知识点及其运用

分块，前缀和，二分

## 四、完整解题思路描述

对于操作 1，很容易想到我们最好使用 lazy 标志，即入果一层上节点数很多的时候用一个标记记录下变化量而不是真正的修改所有的点。这就有了分块思想，如果这一层节点数很少，小于 sqrt(n)，则暴力修改，否则记录到 lazy 标记不真正修改。这样修改复杂度变为 q*2*sqrt(n)

对于操作 2，我们要知道节点 x 在每层上有多少个点。我们首先确定 x 所有点是什么。

我们对树遍历一遍，遍历时维护一个时间戳（计数），然后维护两个数组 head 和 tail，head 记录第一次到达当前点的时间，tail 记录最后一次到达（离开）当前点的时间。这样 head 和 tail 之间的时间经过的点就是它的子树节点。这就相当于把子树求和变成了线性区间求和，将树状的节点号变为了线性的时间戳。然后用树状数组维护区间和。

使用 dp[i]数组维护第 i 层所有点的时间戳。对于询问 x，如果第 i 层节点数大于 sqrt(n),我们只要在每一层 dp[i]中二分两个时间戳：head 和 tail，用一个减法就能计算出在第 i 层中有多少个节点在 x 的子树中。然后节点数乘以当前层的 lazy 标记即可，而对于节点数小于 sqrt(n)，则刚刚已经暴力更新到树里去了，只要一次树状数组求和即可，然后两个加起来就是最终答案。

## 五、技巧与坑点

首先是树拉成数组，其次是对不同深度点的拆解二分查找

## 六、参考文献与学习资料

题解：https://blog.csdn.net/GreyBtfly/article/details/82895864

题目：https://nanti.jisuanke.com/t/A1998

## 七、完整代码

```
#include<bits/stdc++.h>
#define sz(x) ((int)x.size())
using namespace std;
int n,m,a,b,c,md=0,ru=1;
```

```
int biao[100005],shen[100005],w[100005];
vector<int>bian[100005];
long long int he[100005],ji[100005];
vector<int>dp[100005];
void cha(int a,long long int b){
	while(a<=n){
		he[a]+=b;a+=(a&-a);
	}
}
long long int fan(int a){
	long long int f=0;
	while(a>0){
		f+=he[a];a-=(a&-a);
	}return f;
}
long long int qiu(int l,int r){
	return fan(r)-fan(l-1);
}

void bfs(int rt,int de){
	biao[rt]=ru;shen[rt]=de;
	md=max(md,de);
	dp[de].push_back(ru);
	ru++;
	int s=bian[rt].size();
	for(int i=0;i<s;i++){
		bfs(bian[rt][i],de+1);
	}w[rt]=ru-1;
}


int main(){
	scanf("%d%d",&n,&m);
	for(int i=1;i<n;i++){
		scanf("%d%d",&a,&b);
		bian[a].push_back(b);
	}bfs(1,0);int pan=sqrt(n);
	for(int i=0;i<=md;i++){
		if(dp[i].size()>=pan)dp[n+1].push_back(i);
	}
	while(m--){
		scanf("%d",&c);
		if(c==1){
			scanf("%d%d",&a,&b);
			if(dp[a].size()>=pan)ji[a]+=b;
```

```
                else{
                    for(int i=0;i<dp[a].size();i++){
                        cha(dp[a][i],b);
                    }
                }
            }
            else{
                scanf("%d",&a);
                long long int jie=qiu(biao[a],w[a]);
                int zhao=lower_bound(dp[n+1].begin(),dp[n+1].end(),shen[a])-dp[n+1].begin();
                while(zhao<dp[n+1].size()){
                    int ss=dp[n+1][zhao];
                    long long int q=upper_bound(dp[ss].begin(),dp[ss].end(),w[a])-lower_bound(dp[ss].begin(),dp[ss].end(),biao[a]);
                    jie+=q*ji[ss];zhao++;
                }printf("%lld\n",jie);
            }
        }
        return 0;
}
```

## ZOJ-4028 LIS(2018 浙江省赛-E 题)

### 一、题号与题目名称

ZOJ-4028 LIS(2018 浙江省赛-E 题)

### 二、题意简述

第一个数代表输入的组数，每组第一个数表示一个序列的数字个数，下面是序列中，以第 ai 个数结尾所能构成的最长上升子序列，其后的数字是每个数的上限以及下限

### 三、主要知识点及其运用

思维

### 四、完整解题思路描述

第一个数代表输入的组数，每组第一个数表示一个序列的数字个数，下面是序列中，以第 ai 个数结尾所能构成的最长上升子序列，其后的数字是每个数的上限以及下限

### 五、技巧与坑点

莽就完了

### 六、参考文献与学习资料

题目：https://vjudge.net/problem/ZOJ-4028

题解：https://blog.csdn.net/qq_40792618

### 七、完整代码

```
#include<bits/stdc++.h>
#define LL long long
#define lowbit(x) x&-x
using namespace std;
const int MAXN = 100010;
const int INF = 0x3f3f3f3f;
const double eps = 1e-9;
vector<int>di[100005];
vector<int>ma[100005];
int qian[100005];int hou[100005];
int you[100005];int chu[100005];
int main()
{
    int zu,shu;int jin;
    while(~scanf("%d",&zu)){
    while(zu--){
        scanf("%d",&shu);
        int da=0;
        for(int i=0;i<=shu;i++){di[i].clear();ma[i].clear();}
        for(int i=0;i<shu;i++){
            scanf("%d",&jin);
```

```
            di[jin].push_back(i);
            if(jin>da)da=jin;
            you[i]=di[jin+1].size();
        }
        for(int i=0;i<shu;i++){scanf("%d%d",&qian[i],&hou[i]);}
        for(int i=0;i<di[da].size();i++){
            int wei=di[da][i];
            if(i==0){chu[wei]=hou[wei];ma[da].push_back(chu[wei]);continue;}
            int zhi=chu[di[da][i-1]];if(zhi>hou[wei])zhi=hou[wei];
            chu[wei]=zhi;ma[da].push_back(zhi);
        }
        for(int i=da-1;i>=1;i--){
            for(int j=0;j<di[i].size();j++){
                int wei=di[i][j];int zhi;
                if(ma[i+1].size()>=you[wei]+1&&(j==di[i].size()-1||you[wei]!=you[di[i][j+1]])){
                    if(j==di[i].size()-1)zhi=ma[i+1][ma[i+1].size()-1]-1;
                    else{
                        int jian;
                        //if(i==3)printf("%d %d\n",you[di[i][j+1]],you[wei]);
                        jian=you[di[i][j+1]]-you[wei];
                        zhi=ma[i+1][you[wei]+jian-1]-1;
                    }
                }
                else zhi=hou[wei];
                if(zhi>hou[wei])zhi=hou[wei];
                if(j==0){chu[wei]=zhi;ma[i].push_back(zhi);continue;}
                if(zhi>chu[di[i][j-1]])zhi=chu[di[i][j-1]];
                chu[wei]=zhi;ma[i].push_back(zhi);
            }
        }
        for(int i=0;i<shu;i++){
            if(i==0)printf("%d",chu[i]);
            else printf(" %d",chu[i]);
        }
        puts("");
    }
    }
    return 0;
}
```

# ZOJ-4027 Sequence Swapping(2018 浙江省赛-D 题)

## 一、题号与题目名称

【ZOJ-4027】 Sequence Swapping(2018 浙江省赛-D 题）

## 二、题意简述

当且仅当 sk='('并且 s(k+1)=')'时可以交换二者，并且得到两者价值乘积的价值，问最多可以得到多少价值。

## 三、主要知识点及其运用

构造、拓扑、bfs

## 四、完整解题思路描述

首先，这道题要求只能左括号和右括号换，这就决定了每个左括号能到达的最右位置，如果这个括号可以被移动到第 j 个位置，那么他后面的左括号一定都在比他右的位置，
也就是 j+1 位之后，设 dp[i][j]是把第 i 个左括号移动到 j 右面（包括 j）位中能得到的最大价值
转移方程为：

dp[i][j]=max(dp[i+1][j+1]+移动到这里能得到的值，dp[i][j+1])(可以直接开始，因为 dp[i+1][j+1]一开始是 0 所以无影响)
而移动到这里所得到的值是（移动的这个左括号的值）*（他交换的右括号的值的和），用前缀和表示前多少个右括号的和，在输入的时候记录它前面有几个右括号，
他所在位减去他右边的左括号就是他右边的右括号的值数量。由此可知：右括号的值的和=前缀和（括号总数量-右边括号）-前缀和（左边的右括号）。

## 五、参考文献与学习资料

题解：https://blog.csdn.net/qq_40792618/article/details/80145467
题目：https://vjudge.net/problem/ZOJ-4027

## 六、完整代码

```
#include<bits/stdc++.h>
#define LL long long
#define lowbit(x) x&-x
using namespace std;
const LL MAXN = 100010;
const LL INF = 0x3f3f3f3f;
const double eps = 1e-9;

LL he[1005];
LL dp[1005][1005];
LL kuo[1005];
char chuan[1005];
LL val[1005];
LL wei[1005];
LL zhi[1005];
LL qi[1005];
```

```
int main()
{
	LL a,k;
	while(~scanf("%lld",&a))
	{
		while(a--)
		{
			scanf("%lld",&k);
			scanf("%s",chuan);
			LL ru=1;
			kuo[0]=0;
			LL ru1=1;
			for(LL i=0; i<k; i++)
			{
				scanf("%lld",&val[i]);
				if(chuan[i]==')')
				{
					kuo[ru]=kuo[ru-1]+val[i];
					ru++;
				}
				else
				{
					wei[ru1]=i;
					zhi[ru1]=val[i];
					qi[ru1]=ru-1;
					ru1++;
				}
			}
			ru1--;
			ru--;
			memset(dp,0,sizeof(dp));
			LL jie=0;
			for(LL i=ru1; i>=1; i--)
			{
				LL zuo=ru1-i;
				for(LL j=k-1-zuo; j>=wei[i]; j--)
				{
					LL hou=k-1-j-zuo;
					LL de=kuo[ru-hou]-kuo[qi[i]];
					if(j==k-1-zuo)dp[i][j]=zhi[i]*de+dp[i+1][j+1];
					else dp[i][j]=max(zhi[i]*de+dp[i+1][j+1],dp[i][j+1]);
					if(i==1)
					{
						if(dp[i][j]>jie)jie=dp[i][j];
					}
```

```
                }
                for(LL j=wei[i]-1; j>=wei[i-1]; j--)
                {
                    dp[i][j]=dp[i][j+1];
                    if(i==1)
                    {
                        if(dp[i][j]>jie)jie=dp[i][j];
                    }
                }
            }
            printf("%lld\n",jie);
        }
    }
    return 0;
}
/*
5
4
)))(
1 1 1 1
6
)())()
1 3 5 -1 3 2
6
)())()
1 3 5 -100 3 2
3
())
1 -1 -1
3
)))
-1 -1 -1
*/
```

# HDU 5534 完全背包 (计 17-4 潘子怡)

## 一、题号与题目名称

HDU 5534 Partial Tree

## 二、题意简述

T 组测试，给出点数 n，要求连(n-1)条边使之构成一棵生成树，并给出一个函数 f(d)表示度数为 d 的点的价值，问构造的树所有点价值总和的最大值。

1≤T≤2015

2≤n≤2015

0≤f(i)≤10000

最多 10 组数据 n>100

## 三、主要知识点及其运用

本题涉及树的特性和完全背包的应用。

## 四、完整解题思路描述

根据树的特性，树中每个点的度数至少为 1，对于一颗树边数为 n-1，结点总度数和 2*n-2。

考虑和为 n-2 的度数分配给 n 个点，度数 d 价值为 f(d)。可以理解成总容积为 n-2，有体积分别为 1~n-2 的 n-2 种物品，物品体积为 d，价值为 f(d+1)-f(1)的完全背包问题。

dp[i][j]：装有 1~i 物品，总体积为 j 的背包价值的最大值。状态转移方程为

j < i: dp[i][j] = dp[i - 1][j]

j >= i: dp[i][j] = max(dp[i - 1][j],dp[i][j - i] + f[i + 1] - f[1])

可以降成一维滚动数组：dp[j] = max(dp[j],dp[j - i] + f[i + 1] - f[1])

需要注意 dp 数组初始化为-inf，dp[0](dp[0][0])=0。

## 五、技巧与坑点

由于要保证度数总和=2*n-2，即保证背包装满，初始化 dp 数组时注意一下就好。

## 六、参考文献与学习资料

题目：http://acm.hdu.edu.cn/showproblem.php?pid=5534

## 七、完整代码

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 2020;
const int inf = 1e9 + 7;
int f[maxn],dp[maxn];
int main()
{
    int t;
    scanf("%d",&t);
    while(t --)
    {
        int n;
        scanf("%d",&n);
```

```
        for(int i = 1;i < n;i ++)
            scanf("%d",&f[i]);
        for(int i = 0;i <= n - 2;i ++)
            dp[i] = -inf;
        dp[0] = 0;
        for(int i = 1;i <= n - 2;i ++)
            for(int j = i;j <= n - 2;j ++)
                dp[j] = max(dp[j],dp[j - i] + f[i + 1] - f[1]);
        printf("%d\n",dp[n - 2] + n * f[1]);
    }
}
```

# LibreOJ 6062 线段树+二分图匹配 Hall 定理(计 17-4 潘子怡)

## 一、题号与题目名称

LibreOJ 6062 「2017 山东一轮集训 Day2」Pair

## 二、题意简述

给出一个长度为 n 的数列 a 和一个长度为 m 的数列 b，求 a 有多少个长度为 m 的子串与 b 匹配，即至少存在一种方案使两个数列中的数两两配对的和不小于 h。

1≤m≤n≤150000

1≤ai,bi,h≤10^9。

## 三、主要知识点及其运用

本题涉及线段树和二分图匹配 Hall 定理的应用。

## 四、完整解题思路描述

对 b 数组从小到大排好序，要完成两两匹配，必须满足对所有 i：1 ~ m，a 的对应区间内至少有 i 个数 + b[i] >= h，即满足 num[i] - i >= 0（num[i]：b[i] + a[j] >= h 的 j 的数量），即 min(num[i] - i) >= 0。

通过权值线段树维护 min(num[i] - i)：

1．建树，将代表[ x,x ]区间的叶子节点，权值初始化为 -x

2．对 j:1 ~ n 的 a[j]二分查找满足 b[i] + a[j] >= h 的 i 的最小值 id[j]，

3．区间更新，对[ id[j],m ]区间 + 1，并对[ id[j-m+1],m ]区间 - 1，此时根节点的权值为 min(num[i] - i)

遍历区间端点，更新线段树的同时，记录 min(num[i] - i) >= 0 区间的个数。

## 五、技巧与坑点

查询的区间均为[ 1,m ]，可以直接通过查询根节点权值，不用调用查询函数。

## 六、参考文献与学习资料

Hall 定理：https://blog.csdn.net/feynman1999/article/details/76037603

题目：https://loj.ac/problem/6062

类似题目：https://nanti.jisuanke.com/t/A1617

## 七、完整代码

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1.5e5 + 5;
struct Tree
{
    int lazy,val;
}tree[maxn << 2];
int n,m,h,a[maxn],b[maxn];

void push_up(int rt)
{
    tree[rt].val = min(tree[rt <<1].val,tree[rt <<1|1].val);
```

```
}

void push_down(int rt)
{
    if(!tree[rt].lazy)
        return ;
    int lazy = tree[rt].lazy;
    tree[rt <<1].lazy += lazy;
    tree[rt <<1|1].lazy += lazy;
    tree[rt <<1].val += lazy;
    tree[rt <<1|1].val += lazy;
    tree[rt].lazy = 0;
}

void build(int rt,int l,int r)
{
    tree[rt].lazy = 0;
    if(l == r)
    {
        tree[rt].val = -l;
        return ;
    }
    int mid = l + r >> 1;
    build(rt <<1,l,mid);
    build(rt <<1|1,mid + 1,r);
    push_up(rt);
}

void update(int rt,int l,int r,int L,int R,int x)
{
    if(L > R)return ;
    if(L <= l && r <= R)
    {
        tree[rt].val += x;
        tree[rt].lazy += x;
        return ;
    }
    push_down(rt);
    int mid = l + r >> 1;
    if(L <= mid)
        update(rt <<1,l,mid,L,R,x);
    if(R > mid)
        update(rt <<1|1,mid + 1,r,L,R,x);
    push_up(rt);
}
```

```
int main()
{
    while(~scanf("%d%d%d",&n,&m,&h))
    {
        for(int i = 1;i <= m;i ++)
            scanf("%d",&b[i]);
        sort(b + 1,b + m+1);
        build(1,1,m);
        int ans = 0;
        for(int i = 1;i <= n;i ++)
        {
            scanf("%d",&a[i]);
            a[i] = lower_bound(&b[1],&b[m + 1],h - a[i]) - &b[0];
            update(1,1,m,a[i],m,1);
            if(i >= m)
            {
                if(tree[1].val >= 0)
                    ans++;
                update(1,1,m,a[i - m + 1],m,-1);
            }
        }
        printf("%d\n",ans);
    }
}
```

# 计蒜客 A1617 线段树+二分图匹配 Hall 定理+尺取 (计 17-4 潘子怡)

## 一、题号与题目名称

计蒜客 A1617 LOVER II

## 二、题意简述

T 组测试，有 n 个女生 a[i]（i：1 ~ n）和 m 个男生 b[j] （j：1 ~ m），如果 a[i] + b[j] >= k，他们就可以在一起。q 次询问，每次问能否使所有女生跟 [ l,r ]区间内的男生配对。

1≤T≤10

1≤n,m≤2*10^5

0≤k≤10^9

0≤a[i]≤10^9

0≤b[i]≤10^9

1≤q≤10^5

1≤l≤r≤m

## 三、主要知识点及其运用

本题涉及线段树和二分图匹配 Hall 定理的应用。

## 四、完整解题思路描述

判断区间内 b[j]元素是否可以匹配全部 a[i]的方法同上题。枚举右端点 r，通过尺取法，得到满足匹配要求的区间最大左端点 maxl[r]。每次询问，判断左端点 l 和 maxl[r]的关系:l <= maxl[r]时可以配对。

## 五、技巧与坑点

对∀r ∈ [1,m]，maxl[r] >= maxl[r - 1]，可以通过尺取法得到所有 r：1 ~ m 的 maxl[r]。

## 六、参考文献与学习资料

题目：https://nanti.jisuanke.com/t/A1617

## 七、完整代码

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 2e5 + 5;
struct Tree
{
    int lazy,val;
}tree[maxn << 2];
int n,m,k,b[maxn],a[maxn],maxl[maxn];

void push_up(int rt)
{
    tree[rt].val = min(tree[rt <<1].val,tree[rt <<1|1].val);
}

void push_down(int rt)
```

```
{
    if(!tree[rt].lazy)
        return ;
    int lazy = tree[rt].lazy;
    tree[rt <<1].lazy += lazy;
    tree[rt <<1|1].lazy += lazy;
    tree[rt <<1].val += lazy;
    tree[rt <<1|1].val += lazy;
    tree[rt].lazy = 0;
}

void build(int rt,int l,int r)
{
    tree[rt].lazy = 0;
    if(l == r)
    {
        tree[rt].val = -l;
        return ;
    }
    int mid = l + r >> 1;
    build(rt <<1,l,mid);
    build(rt <<1|1,mid + 1,r);
    push_up(rt);
}

void update(int rt,int l,int r,int L,int R,int x)
{
    if(L > R)return ;
    if(L <= l && r <= R)
    {
        tree[rt].val += x;
        tree[rt].lazy += x;
        return ;
    }
    push_down(rt);
    int mid = l + r >> 1;
    if(L <= mid)
        update(rt <<1,l,mid,L,R,x);
    if(R > mid)
        update(rt <<1|1,mid + 1,r,L,R,x);
    push_up(rt);
}

int main()
{
```

```
    int t;
    scanf("%d",&t);
    while(t --)
    {
        scanf("%d%d%d",&n,&m,&k);
        memset(maxl,0,sizeof(maxl));
        for(int i = 1;i <= n;i ++)
            scanf("%d",&a[i]);
        sort(a + 1,a + n+1);
        build(1,1,n);
        int l = 0,r = 1;
        b[0] = n +1;
        for(;r <= m;r ++)
        {
            scanf("%d",&b[r]);
            b[r] = lower_bound(&a[1],&a[n + 1],k - b[r]) - &a[0];
            update(1,1,n,b[r],n,1);
            while(l <= m)
            {
                update(1,1,n,b[l],n,-1);
                if(tree[1].val < 0)
                {
                    update(1,1,n,b[l],n,1);
                    maxl[r] = l;
                    break;
                }
                l ++;
            }
        }
        int q;
        scanf("%d",&q);
        while(q --)
        {
            scanf("%d%d",&l,&r);
            if(l > maxl[r])
                puts("0");
            else
                puts("1");
        }
    }
}
```

# 计蒜客 A1607 前缀线性基 (计 17-4 潘子怡)

## 一、题号与题目名称

计蒜客 A1607 XOR

## 二、题意简述

T 组测试，给一个数组 A 包含 n 个元素、k， q 次询问，每次问 k | val 的最大值（val 为 i | [ l,r ]的 A[i]中任选出若干个数异或起来的值）。

1≤T≤10

1≤n≤10000, 1≤q≤100000, 0≤k≤100000

0≤A[i]≤10^8

1≤l≤r≤n

## 三、主要知识点及其运用

本题涉及前缀线性基的应用，也可以用线段树区间合并线性基的方法。

## 四、完整解题思路描述

对所有 i ∈ [ l,r ]，A[i] &= ~k。

维护：按从 1 ~ n 的顺序枚举右界 r，将后出现的数尽可能地放在高位，贪心维护序列的前缀线性基。维护过程记录前缀线性基、插入当前数的时间点 r，保证当前线性基存的是对应插入时间 r 最大的。

查询：求[ l,r ]区间的最大值时，从高位向低位遍历，将插入时间 >= l，且异或后答案变大的数，异或到 val 里。

最终答案 = k | val。

## 五、技巧与坑点

求 k | val 时，考虑两种方法的正确性：

1. 对所有 i ∈ [ l,r ]，A[i] &= ~k，之后维护和查询 A[i]序列的前缀线性基，最终答案 = k | val。

2. 维护和查询 A[i]序列的前缀线性基，计算 val 时所有 k & (1 << i)为 1 的 base[i]不参与异或运算，最终答案 = k | val。

方法 1 正确，方法 2 错误。

例如对于 k = 1001，base[3] = 1100，base[2] = base[1] = base[0] = 0000（均为二进制表示）

方法 1：val = base[3] = base[3] & ~k = 0100，k | val = 1101；

方法 2：val = 0000，k | val = 1001。方法 2 会将非最高位起主要贡献的数忽略。

## 六、参考文献与学习资料

线性基：https://blog.csdn.net/a_forever_dream/article/details/83654397
https://blog.csdn.net/qq_41552508/article/details/96561526

题目：https://nanti.jisuanke.com/t/A1607

类似题目：http://acm.hdu.edu.cn/showproblem.php?pid=6579
https://codeforces.com/problemset/problem/1100/F

## 七、完整代码

```
#include <bits/stdc++.h>
using namespace std;
```

```
const int maxn = 1e4 + 5;
int a[maxn],base[maxn][32],pos[maxn][32];
int main()
{
    int t;
    scanf("%d",&t);
    while(t --)
    {
        int n,q,k;
        scanf("%d%d%d",&n,&q,&k);
        memset(base,0,sizeof(base));
        memset(pos,0,sizeof(pos));
        for(int i = 1;i <= n;i ++)
        {
            scanf("%d",&a[i]);
            a[i]&=~k;
            for(int j = 30;j >= 0;j --)
            {
                base[i][j] = base[i - 1][j];
                pos[i][j] = pos[i - 1][j];
            }
            int id = i;
            for(int j = 30;j >= 0;j --)
            {
                if(!(a[i] & (1<<j)))
                    continue;
                if(base[i][j])
                {
                    if(id > pos[i][j])
                        swap(a[i],base[i][j]),swap(id,pos[i][j]);
                    a[i] ^= base[i][j];
                }
                else
                {
                    base[i][j] = a[i];
                    pos[i][j] = id;
                    break;
                }

            }
        }
        while(q --)
        {
            int l,r;
            scanf("%d%d",&l,&r);
```

```
            int ans = 0;
            for(int i = 30;i >= 0;i --)
            {
                if(pos[r][i] >= l)
                    ans  = max(ans ^ base[r][i],ans);
            }
            ans |= k;
            printf("%d\n",ans);
        }
    }
}
```

# Gym 100837 F 竞赛树+状态压缩 DP （计 17-4 陈仁苗）

## 一、题号与题目名称

Gym - 100837 F Controlled Tournament

## 二、题意简述

一共 N 个人，给出任意两个人之间的胜负关系，你的编号是 M。现在需要安排一棵竞赛树使得 M 能够胜出，问使竞赛树高度最小且 M 获胜的安排方案一共有多少个。(1≤N≤16)

## 三、主要知识点及其运用

本题涉及状态压缩 DP 及二叉树的深度理解。

## 四、完整解题思路描述

dp 建三维，dp[i][j][k]第一维是谁赢，第二维当前是第几层，第三维是当前参与比赛的人的集合。那么 dp[u][h][s]就可以通过枚举 s 的各种情况获得，枚举得到 i，然后要求 u 在 i 中，然后用 s^i 就能得到另一边的状态，观察另一边的状态中是否有人(k)能被打败，如果有的话，继续以 k 能被打败，h-1 层，s^i 的状态继续递归前进，最后得到最终答案。对于剪枝，一个是记忆化搜索 ，如果已经有结果了直接 return，其次是如果当前状态只有一个人了，那么他一定是冠军，返回 1，还有一个剪枝是 1<<h 表示当前层数最多的人数，如果状态表示的 1 的数量比 1<<h 要多，那么肯定不行直接 return 0，这是一个很强的剪枝，也比较难想到。

## 五、技巧与坑点

一个坑点在于正确理解层数少的含义，层数少就是说每一轮都没有人空闲都在比赛，那么很容易得出竞赛树的高度就等于 ceil(log2(n))，参照二叉树的性质。技巧在于状态的枚举以及每种状态中 1 个数的预处理。

## 六、参考文献与学习资料

https://blog.csdn.net/toohandsomeleaseId/article/details/96997618
https://blog.csdn.net/qq_41552508/article/details/90082361

## 七、完整代码

```
#include<bits/stdc++.h>

using namespace std;
#define ll long long
#define rep(i,a,b) for(int i=a;i<=b;i++)
#define per(i,a,b) for(int i=a;i>=b;i--)
#define ll long long
const int MAXN = 20+5;
int a[MAXN][MAXN],n,m,h,dp[MAXN][MAXN][1<<16],bits[1<<16];
int dfs(int u,int h,int s){
    if(dp[u][h][s]!=-1)return dp[u][h][s];
    if((1<<h)<bits[s])return 0;
    if(bits[s]==1)return 1;
    dp[u][h][s]=0;
    for(int i=s&(s-1);i;i=s&(i-1)){
```

```
        if(i&(1<<u)){
            int j=s^i;
            rep(k,0,n-1){
                if(a[u][k]&&((1<<k)&j)){
                    dp[u][h][s]+=dfs(u,h-1,i)*dfs(k,h-1,j);
                }
            }
        }
    }
    return dp[u][h][s];
}
int main()
{
    freopen("f.in","r",stdin);
    freopen("f.out","w",stdout);
    memset(bits,0,sizeof(bits));
    rep(i,1,(1<<16)-1){
        if(i&1)bits[i]=bits[i>>1]+1;
        else bits[i]=bits[i>>1];
    }
    scanf("%d%d",&n,&m);
    h=ceil(log2(n));
    rep(i,0,n-1){
        rep(j,0,n-1){
            scanf("%d",&a[i][j]);
        }
    }
    memset(dp,-1,sizeof(dp));
    int ans=dfs(m-1,h,(1<<n)-1);
    printf("%d\n",ans);
    return 0;
}
```

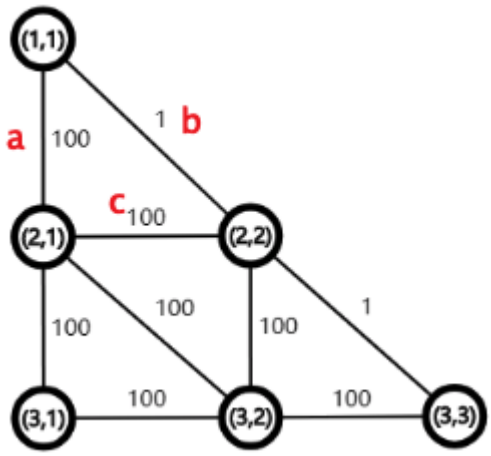
# ZOJ 4122 Triangle City Dijkstra+删边+欧拉路(计 17-4 陈仁苗)

## 一、题号与题目名称

ZOJ 4122 Triangle City （山东 2019 省赛 J 题）

## 二、题意简述

给出一个三角形城市，结构如下图所示。给出每个三角形的 a、b、c 三个边权，求从(1,1)到(n,n)的最长路，要求输出长度和路径，每条边在最长路中最多出现一次。(2≤n≤300)




## 三、主要知识点及其运用

本题主要涉及最短路边的记录和删除以及欧拉路的路径求解。

## 四、完整解题思路描述

每条边最多出现一次，马上想到欧拉路 or 欧拉回路，又由于(1,1)到(n,n)，确定是欧拉路，由欧拉路性质，有两个点是奇数度其他都偶数。由题意肯定原图中每个点都是偶数度，所以就需要删一些边使得满足欧拉路性质得到满足，可以想到是删掉(1,1)到(n,n)的一条路就行，且删掉的边边权和要尽可能小，因为我们要删掉的是(1,1)到(n,n)的最短路。那么这个题目就已经接近结束了，实现的话用一个 pre 记录上一个点是谁，pre2 记录边的标号，回溯的时候把相应边的 vis 变 0 就行，用邻接表的特性，同时将正反边的 vis 变 0(直接^1 就能将正边编号变反边，反之亦然)，然后跑一下 Hierholzers 把欧拉路弄出来就结束了，最长路长度就是所有边的和减去最短路

## 五、技巧与坑点

技巧一个是由于是双向边，所以在删掉一条边的时候要把对应的反边一起删掉，所以就可以利用数字的性质，让边从偶数开始存，比如 2^1=3,3^1=2，用一个 vis 表示边是否存在，记录最短路需要用一个两个数组分别记录点和边的编号。

## 六、参考文献与学习资料

https://blog.csdn.net/toohandsomeIeaseId/article/details/97054270

欧拉路入门题：https://www.luogu.org/problem/P2731
入门题题解：https://blog.csdn.net/toohandsomeIeaseId/article/details/97051916

# 七、完整代码

```
#include<bits/stdc++.h>

using namespace std;
#define ll long long
#define rep(i,a,b) for(ll i=a;i<=b;i++)
#define per(i,a,b) for(ll i=a;i>=b;i--)
#define ll long long
const ll MAXN = 300+5;
const ll MAXM = 300*300*6+5;
ll a[MAXN][MAXN],b[MAXN][MAXN],c[MAXN][MAXN],mp[MAXN][MAXN];
ll
n,head[MAXM],ver[MAXM],edge[MAXM],Next[MAXM],d[MAXM],tot,vis[MAXM],pre[MAXM]
,pre2[MAXM],s[MAXM],pos;
bool v[MAXM];
pair<ll,ll>p[MAXM];
priority_queue<pair<ll,ll> >q;
void addedge(ll x,ll y,ll z){
    ver[tot]=y,edge[tot]=z,vis[tot]=1,Next[tot]=head[x],head[x]=tot++;
}
void dij(){
    memset(d,0x3f,sizeof(d));
    memset(v,0,sizeof(v));
    d[1]=0;
    q.push(make_pair(0,1));
    while(q.size()){
        ll x=q.top().second;q.pop();
        if(v[x])continue;
        v[x]=1;
        for(ll i=head[x];i;i=Next[i]){
            ll y=ver[i],z=edge[i];
            if(d[y]>d[x]+z){
                pre[y]=x;
                pre2[y]=i;
                d[y]=d[x]+z;
                q.push(make_pair(-d[y],y));
            }
        }
    }
}
void dfs(ll u){
    for(ll i=head[u];i;i=Next[i]){
        if(vis[i]){
            vis[i]=0;
            vis[i^1]=0;
```

```
            dfs(ver[i]);
        }
    }
    s[++pos]=u;
}
int main()
{
    ll t;
    scanf("%lld",&t);
    while(t--){
        ll num=0,ans=0;
        memset(head,0,sizeof(head));
        tot=2;pos=0;
        scanf("%lld",&n);
        rep(i,1,n-1){
            rep(j,1,i){
                scanf("%lld",&a[i][j]);
                ans+=a[i][j];
            }
        }
        rep(i,1,n-1){
            rep(j,1,i){
                scanf("%lld",&b[i][j]);
                ans+=b[i][j];
            }
        }
        rep(i,1,n-1){
            rep(j,1,i){
                scanf("%lld",&c[i][j]);
                ans+=c[i][j];
            }
        }
        rep(i,1,n){
            rep(j,1,i){
                mp[i][j]=++num;
                p[num].first=i;
                p[num].second=j;
            }
        }
        rep(i,1,n-1){
            rep(j,1,i){
                addedge(mp[i][j],mp[i+1][j],a[i][j]);
                addedge(mp[i+1][j],mp[i][j],a[i][j]);
                addedge(mp[i][j],mp[i+1][j+1],b[i][j]);
                addedge(mp[i+1][j+1],mp[i][j],b[i][j]);
```

```
                addedge(mp[i+1][j],mp[i+1][j+1],c[i][j]);
                addedge(mp[i+1][j+1],mp[i+1][j],c[i][j]);
            }
        }
        memset(pre,0,sizeof(pre));
        dij();
        ans-=d[num];
        pre[1]=0;
        ll u=pre[num],v=pre2[num];
        while(1){
            vis[v]=0;
            vis[v^1]=0;
            v=pre2[u];
            u=pre[u];;
            if(u==0)break;
        }
        dfs(1);
        printf("%lld\n%lld\n",ans,pos);
        per(i,pos,1){
            printf("%lld   %lld%c",p[s[i]].first,p[s[i]].second,i==1?'\n':'
');
        }
    }
    return 0;
}
```

# Gym 101161 G Binary Strings 矩阵快速幂优化 DP （计 17-4 陈仁苗）

## 一、题号与题目名称

Gym-101161 G Binary Strings（2016-2017 ACM-ICPC Asia-Bangkok Regional Contest）

## 二、题意简述

## 三、主要知识点及其运用

动态规划以及矩阵快速幂对递推式的优化。

## 四、完整解题思路描述

首先一个显而易见(并不)的 DP，设 dp[i][0]表示长度为 i 的，以 0 为结尾的 01 串数量，dp[i][1]表示长度为 i 的，以 1 为结尾的 01 串数量，以 0 结尾的话，很显然前面可以跟 0 和 1，但如果以 1 结尾，那么由于不能出现连续 1，所以只能通过结尾为 0 的转换而来，所以得到 DP 方程如下：

dp[i][0]=dp[i-1][0]+dp[i-1][1],dp[i][1]=dp[i-1][0]

然后打个表就可以发现，dp[i][0]+dp[i][1]居然是一个斐波那契数列！

当然，也可以通过证明得到：

dp[i][0]+dp[i][1]=dp[i-1][0]+dp[i-1][1]+dp[i-1][0]=dp[i-1][0]+dp[i-1][1]+dp[i-2][0]]+dp[i-2][1]

把这个结论用 d 数组表示，那么 d[1]=2,d[2]=3,d[3]=5....

我们要得到的是 d[k]+d[2k]+d[3k]+....

给这个前缀和定义一个数组叫 sum[i],其意义是小于等于 i 的 k 的倍数的 d 数组的和

所以题目就是 sum[r/k]-sum[(l-1)/k]

由于在递推过程中，只有到 k 的倍数才更新 sum，所以在到达 k 的倍数之前，直接就让 sum 等于原来的值，这样的递推进行 k-1 次，然后最后一次再进行更新，这样的过程持续 n 次，我们就能得到小于等于 nk 长度的所有 k 的倍数的 d 数组的和！

所以就是

$$(dp[k-1] \quad dp[k] \quad sum[k])*\begin{pmatrix}0&1&0\\1&1&0\\0&0&1\end{pmatrix}^{k-1}*\begin{pmatrix}0&1&1\\1&1&1\\0&0&1\end{pmatrix}^{r/k}=(dp[2*k-1] \quad dp[2*k] \quad sum[2*k])$$

## 五、技巧与坑点

技巧主要在于 DP 方程的推导以及矩阵的构造，其中一个难点在于只计算 k 的倍数，可以采用前 k-1 个 sum 只是保留，直到 k 的倍数再进行更新。

## 六、参考文献与学习资料

https://blog.csdn.net/toohandsomeIeaseId/article/details/97172244

矩阵快速幂优化递推入门题：http://poj.org/problem?id=3070

## 七、完整代码

```
#include<bits/stdc++.h>

using namespace std;
#define ll long long
```

```
#define rep(i,a,b) for(ll i=a;i<=b;i++)
#define per(i,a,b) for(ll i=a;i>=b;i--)
#define ll long long
const ll MOD = 1e9+7;
ll n;
void mul(ll f[3],ll a[3][3]){
    ll c[3];
    memset(c,0,sizeof(c));
    rep(j,0,2){
        rep(k,0,2){
            c[j]=(c[j]+(ll)f[k]*a[k][j])%MOD;
        }
    }
    memcpy(f,c,sizeof(c));
}
void mul2(ll a[3][3],ll b[3][3]){
    ll c[3][3];
    memset(c,0,sizeof(c));
    rep(i,0,2){
        rep(j,0,2){
            rep(k,0,2){
                c[i][j]=(c[i][j]+(ll)a[i][k]*b[k][j])%MOD;
            }
        }
    }
    memcpy(a,c,sizeof(c));
}
void mulself(ll a[3][3]){
    ll c[3][3];
    memset(c,0,sizeof(c));
    rep(i,0,2){
        rep(j,0,2){
            rep(k,0,2){
                c[i][j]=(c[i][j]+(ll)a[i][k]*a[k][j])%MOD;
            }
        }
    }
    memcpy(a,c,sizeof(c));
}
int main()
{
    ll t;
    scanf("%I64d",&t);
    ll l,r,k;
    rep(_,1,t){
```

```
        scanf("%I64d%I64d%I64d",&l,&r,&k);
        ll p[3][3]={{1,0,0},{0,1,0},{0,0,1}};
        ll p2[3][3];
        ll a[3][3]={{0,1,0},{1,1,0},{0,0,1}};
        ll n=k-1;
        for(;n;n>>=1){
            if(n&1)mul2(p,a);
            mulself(a);
        }
        ll b[3][3]={{0,1,1},{1,1,1},{0,0,1}};
        mul2(p,b);
        ll f1[3]={1,1,0};
        ll f2[3]={1,1,0};
        n=r/k;
        memcpy(p2,p,sizeof(p));
        for(;n;n>>=1){
            if(n&1)mul(f1,p);
            mulself(p);
        }
        n=(l-1)/k;
        for(;n;n>>=1){
            if(n&1)mul(f2,p2);
            mulself(p2);
        }
        ll ans=(f1[2]-f2[2]+MOD)%MOD;
        printf("Case %I64d: %I64d\n",_,ans);
    }
    return 0;
}
```

# HDU 6562 Lovers 硬核线段树 (计 17-4 陈仁苗)

## 一、题号与题目名称

HDU 6562 Lovers（2018CCPC 吉林赛站 H 题）

## 二、题意简述

有 n 个空串，两种操作，第一种操作是让 l 到 r 区间内的串开头结尾加上 d，比如之前是 33，d 是 5，那就变成 5335，另一个操作是 l 到 r 区间内的所有串求和并输出。

## 三、主要知识点及其运用

线段树的维护和 lazy 标记更新。

## 四、完整解题思路描述

线段树需要维护五个变量，分别是表示的区间的和 sum，表示的区间的数字长度 len 的 $\sum_{i=l}^{r}10^{len_i}$ len，表示区间右边应该加上的数字的延迟标记为 lazr，左边应该加上的数字的延迟标记为 lazl，设这些数字的长度的长度为 $len_i$，那么 lazylen 就是 $\sum_{i=l}^{r}10^{len_i}$

查询是最好写的，直接按照普通的线段树加 sum 就完事了

更新的时候，首先更新 sum，因为原先的数字需要乘上 10 为新添的数字留个空位，所以第一项就是 sum*10，然后由于这一区间内所有数字都加上了 val，也就是第二项需要是(r-l+1)*val，第三项就是在最左边加上 val，由于之前维护的 len 是 $\sum_{i=l}^{r}10^{len_i}$，所以我们只需要在 len 的基础上*10 就是各个区间内的数字要放数字的位置。比如之前是 33 的话，val 是 5，之前的 len 是 10^2=100，乘上 10 就是千位的位置，正好与我们想要的位置符合。

然后更新 len，因为多出来俩数字，也就是多了俩 0，乘个 100 即可

接着更新 lazr，之前的 lazr*10 给新人空位置，然后+val 即可

lazl 的话是往原来数字的左边加，由于 lazlen 已经记录了当前应该往哪个位加，直接 lazl+val*lazlen 就可以

lazlen 是每次多一个数字，所以乘个 10 即可

接着讲最最最难的延迟标记下传，更新孩子的 sum 的时候，先让孩子的 sum 乘上父亲的 lazlen，使得孩子的 sum 右边留下空间放父亲的 lazr，然后加上孩子的(r-l+1)*父亲的 lazr，左边加就是父亲的 lazl*孩子的 len 再乘上父亲的 lazlen。

孩子的 len 就是本身乘两次父亲的 lazlen，因为 lazlen 是 lazr 和 lazl 各自的长度，所以总长是这俩共同贡献，需要乘两次

lazl 就是父亲的 lazl 跟到孩子的 lazl 后面，也就是父亲的 lazl*孩子的 lazlen+孩子之前的 lazl

lazr 就是孩子后面空出父亲的 lazlen 的距离，给父亲的 lazr 留位置，也就是孩子的 lazr*父亲的 lazlen+孩子之前的 lazr

lazlen 就是孩子的 lazlen 乘上父亲的 lazlen，因为它是单单一边的长度，所以只用一遍

## 五、技巧与坑点

技巧主要就是由于该题有大量需要维护的变量，导致代码实现难度很大，需要有较深厚的代码能力和对线段树原理的理解。每个变量之间的联系都需要时刻记住其作用和维护次序。

## 六、参考文献与学习资料

https://blog.csdn.net/toohandsomeIeaseId/article/details/97615426

## 七、完整代码

```
#include<iostream>


using namespace std;
#define ll long long
#define rep(i,a,b) for(ll i=a;i<=b;i++)
#define per(i,a,b) for(ll i=a;i>=b;i--)
#define ll long long
const ll MOD = 1e9+7;
const ll MAXN = 1e5+5;
char s[10];
struct node{
    ll l,r,sum,len,lazl,lazr,lazlen;
}a[MAXN<<2];
void build(ll rt,ll l,ll r){
    a[rt].l=l,a[rt].r=r;
    a[rt].sum=a[rt].lazl=a[rt].lazr=0;
    a[rt].lazlen=1;
    if(l==r){
        a[rt].len=1;
        return;
    }
    ll mid=(l+r)>>1;
    build(rt<<1,l,mid);
    build(rt<<1|1,mid+1,r);
    a[rt].len=(a[rt<<1].len+a[rt<<1|1].len)%MOD;
}
void spread(ll rt){
    if(a[rt].lazlen>1){

a[rt<<1].sum=((ll)a[rt].lazlen*a[rt<<1].sum%MOD+(ll)a[rt].lazr*(a[rt<<1].r-a
[rt<<1].l+1)%MOD+(ll)a[rt].lazl*a[rt<<1].len%MOD*a[rt].lazlen%MOD)%MOD;

a[rt<<1|1].sum=((ll)a[rt].lazlen*a[rt<<1|1].sum%MOD+(ll)a[rt].lazr*(a[rt<<1|
1].r-a[rt<<1|1].l+1)%MOD+(ll)a[rt].lazl*a[rt<<1|1].len%MOD*a[rt].lazlen%MOD)
%MOD;
```

```
            a[rt<<1].len=(ll)a[rt<<1].len*a[rt].lazlen%MOD*a[rt].lazlen%MOD;

a[rt<<1|1].len=(ll)a[rt<<1|1].len*a[rt].lazlen%MOD*a[rt].lazlen%MOD;


a[rt<<1].lazl=((ll)a[rt].lazl*a[rt<<1].lazlen%MOD+a[rt<<1].lazl)%MOD;

a[rt<<1|1].lazl=((ll)a[rt].lazl*a[rt<<1|1].lazlen%MOD+a[rt<<1|1].lazl)%MOD;

            a[rt<<1].lazr=((ll)a[rt<<1].lazr*a[rt].lazlen%MOD+a[rt].lazr)%MOD;

a[rt<<1|1].lazr=((ll)a[rt<<1|1].lazr*a[rt].lazlen%MOD+a[rt].lazr)%MOD;

            a[rt<<1].lazlen=((ll)a[rt<<1].lazlen*a[rt].lazlen)%MOD;
            a[rt<<1|1].lazlen=((ll)a[rt<<1|1].lazlen*a[rt].lazlen)%MOD;

            a[rt].lazl=a[rt].lazr=0;
            a[rt].lazlen=1;
        }
    }
    void update(ll rt,ll l,ll r,ll val){
        if(a[rt].l>=l&&a[rt].r<=r){

a[rt].sum=((a[rt].r-a[rt].l+1)*val%MOD+a[rt].sum*10ll%MOD+a[rt].len*val*10ll
%MOD)%MOD;
            a[rt].len=a[rt].len*100ll%MOD;
            a[rt].lazr=(a[rt].lazr*10ll+val)%MOD;
            a[rt].lazl=(a[rt].lazl+a[rt].lazlen*(ll)val)%MOD;
            a[rt].lazlen=a[rt].lazlen*10ll%MOD;
            return;
        }
        spread(rt);
        ll mid=(a[rt].l+a[rt].r)>>1;
        if(l<=mid)update(rt<<1,l,r,val);
        if(r>mid)update(rt<<1|1,l,r,val);
        a[rt].sum=(a[rt<<1].sum+a[rt<<1|1].sum)%MOD;
        a[rt].len=(a[rt<<1].len+a[rt<<1|1].len)%MOD;
    }
    ll query(ll rt,ll l,ll r){
        if(a[rt].l>=l&&a[rt].r<=r){
            return a[rt].sum;
        }
        spread(rt);
        ll mid=(a[rt].l+a[rt].r)>>1;
```

```
    ll ans=0;
    if(l<=mid)ans=(ans+query(rt<<1,l,r))%MOD;
    if(r>mid)ans=(ans+query(rt<<1|1,l,r))%MOD;
    return ans;
}
int main()
{
    ll t,n,m,l,r,d;
    scanf("%lld",&t);
    rep(_,1,t){
        printf("Case %lld:\n",_);
        scanf("%lld%lld",&n,&m);
        build(1,1,n);
        while(m--){
            scanf("%s",s);
            if(s[0]=='w'){
                scanf("%lld%lld%lld",&l,&r,&d);
                update(1,l,r,d);
            }
            else{
                scanf("%lld%lld",&l,&r);
                ll ans=query(1,l,r);
                printf("%lld\n",ans);
            }
        }
    }
    return 0;
}
```

# ZOJ 3937 More Health Points 树上 dfs+斜率优化 DP+可持久化维护凸壳 (计 17-4 陈仁苗)

## 一、题号与题目名称

ZOJ 3937 More Health Points（浙江 2016 省赛 B 题）

## 二、题意简述

给一棵有向树，任选一节点 i，从 j 节点出去，权值为 val[i]*1+val[i+1]*2+.....+val[i+j-1]*j，也可以选择不进入，求最大权值。

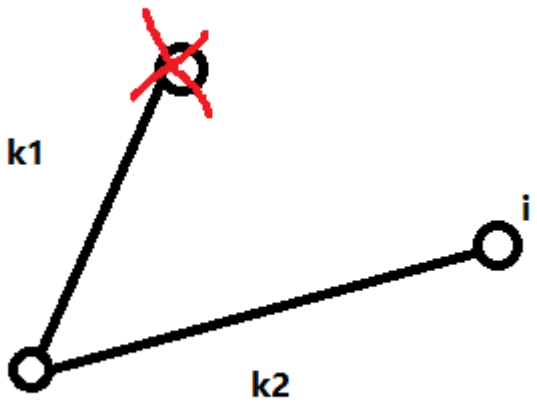
## 三、主要知识点及其运用

本题主要涉及树上 DFS，动态规划，斜率优化以及可持久化维护凸壳。

## 四、完整解题思路描述

首先先考虑如果不在树上，如何解决此题。对于一个序列，求 val[i]*1+val[i+1]*2+.....+val[i+j-1]*j，若定义 f[i]=val[1]*1+val[2]*2+.....+val[i]*i，sum[i]=val[1]+val[2]+.....+val[i]，假设对于 i 来说，[j+1,i]这一区间是最优解，定义 dp[i]为以 i 为终点的最优解，则 dp[i]=f[i]-f[j]-j*(sum[i]-sum[j])。将该式展开得到 dp[i]=f[i]-f[j]-j*sum[i]+j*sum[j]，移项可得 f[j]-j*sum[j]=-j*sum[i]+f[i]-dp[i]

设 j 为 x，y 为 f[j]-j*sum[j]，斜率为-sum[i]，然后维护一个下凸壳，由下凸壳性质可知，其斜率保持不递减，存在单调性，同时又由线性规划知识，f[i]-dp[i]为截距，当截距最小时 dp[i]最大，所以我们需要找到第一个大于-sum[i]的点，该点即满足要求的点。对于维护下凸壳，由于这是一个树上的斜率优化，序列中的数字在不断地改变，需要维护的点也都在改变，所以不能采用简单的单调队列进行维护，需要进行可持久化。首先可以发现，由于下凸壳中的点斜率保持递增，而每个点到 i 点的斜率又在递减，而当点的斜率大于到 i 点的斜率时，这个点就后面那个点就是下凸壳里面的点，如图所示：




然而他不能直接就改序列， 因为当 dfs 回溯时它需要回到原来的状态再继续新的征程，所以就需要满足可持久化需求，也就是需要将当前要插入的凸壳的位置和值都记下来，先改完然后再变回来。这里由于刚才说的性质也是具有单调性的，所以也可以二分实现。

## 五、技巧与坑点

技巧在于需要记录当前原来的位置，因为由于是在树上斜率优化，需要维护的点会变动，所以就需要进行可持久化，前面也不能像普通的斜率优化直接把前面踢掉用单调队列维护，需要二分查找相应位置。

## 六、参考文献与学习资料

https://blog.csdn.net/toohandsomeIeaseId/article/details/97613551
斜率优化入门题：https://blog.csdn.net/toohandsomeIeaseId/article/details/97487838
斜率优化 DP 题解：https://blog.csdn.net/toohandsomeIeaseId/article/details/97487838

## 七、完整代码

```
#include<bits/stdc++.h>

using namespace std;
#define ll long long
#define rep(i,a,b) for(int i=a;i<=b;i++)
#define per(i,a,b) for(int i=a;i>=b;i--)
const int MAXN = 1e5+5;
ll val[MAXN],ans,x[MAXN],y[MAXN];
int n,fa[MAXN],qh,qt,q[MAXN];
vector<int>v[MAXN];

void dfs(int u,int dep,ll sum,ll f){
    x[u]=dep,y[u]=f-dep*sum;
    int l=qh,r=qt-1,pos=qt;
    while(l<=r){
        int mid=(l+r)>>1;
        if((y[q[mid+1]]-y[q[mid]])>=-sum*(x[q[mid+1]]-x[q[mid]])){
            pos=mid;
            r=mid-1;
        }
        else{
            l=mid+1;
        }
    }
    ans=max(ans,-x[q[pos]]*sum-y[q[pos]]+f);
    l=qh,r=qt-1,pos=qt+1;
    while(l<=r){
        int mid=(l+r)>>1;

if((y[q[mid+1]]-y[q[mid]])*(x[u]-x[q[mid]])>=(x[q[mid+1]]-x[q[mid]])*(y[u]-y
[q[mid]])){
            pos=mid+1;
            r=mid-1;
        }
        else l=mid+1;
```

```
    }
    int oldval=q[pos],oldqt=qt;
    qt=pos,q[pos]=u;
    int len=v[u].size();
    rep(i,0,len-1){
        int y=v[u][i];
        dfs(y,dep+1,sum+val[y],f+(dep+1)*val[y]);
    }
    q[pos]=oldval,qt=oldqt;
}
int main()
{
    int t;
    scanf("%d",&t);
    while(t--){
        ans=0;
        scanf("%d",&n);
        rep(i,1,n)scanf("%lld",&val[i]),v[i].clear();
        rep(i,2,n){
            scanf("%d",&fa[i]);
            v[fa[i]].push_back(i);
        }
        qh=qt=1;q[1]=0;
        dfs(1,1,val[1],val[1]);
        printf("%lld\n",ans);
    }
    return 0;
}
```

## GYM 101987L 贪心(网 17-4 张学峰)

## 一、题号与题目名称

GYM 101987L Working Plan

## 二、题意简述

有 m 个工人和 n 天，每个工人要连续工作 w 天再休息 h 天，给出 n 天中每天有多少人工作，问是否存在可行解安排工人工作，并输出每天工作工人的编号。

1 <= m <=2000 ,1 <= n <=2000,1<=w,h<=n。

## 三、主要知识点及其运用

本题涉及贪心与双队列模拟。

## 四、完整解题思路描述

先要处理出来每天有多少个人需要在这天开始工作也就是代码中的 st 数组，然后每次安排人要优先安排当前需要工作天数最多的人，使用两个队列进行模拟，前一个队列要是工作队列，后一个是休息队列，队列元素是结构体类型，需要每个人的 id 号，每个人的当前去要工作天数 cnt，还有这个人下次开始的工作时间 next。

一个优先队列就是工作队列用来存当前可安排工作的需要天数最大的人，然后安排他这一天之后马上入下一个休息队列。

用一层循环遍历天数。然后立即查询休息队列中有没有可以回工作队列的也就是说结束休息可以再次开始工作。然后立即入工作队列。再查看当前工作天数是否有足够的人能安排工作，不能就安排不了。能够安排的话就取出工作队列的队首然后修改 next 和 cnt，如果 cnt！=0 入休息队列。最后判断是否两个队列全空。vector 存答案。

## 五、技巧与坑点

本题技巧就是工人的差别主要在于工作的天数，双队列模拟时候要处理好出入队顺序。

## 六、参考文献与学习资料

http://codeforces.com/gym/101987/attachments

https://blog.csdn.net/qq_41645482/article/details/89133063

## 七、完整代码

```
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<algorithm>
#include<queue>
#include<vector>
#include<stack>
#define ll long long
using namespace std;
const ll MAXN=2005;
vector<int>v[MAXN];
int day[MAXN],per[MAXN],st[MAXN];
```

```
struct Point
{
    int id,cnt,next;
    Point(int i,int c,int n){id=i;cnt=c;next=n;}
    bool operator < (const Point& r)const
    {
        return cnt<r.cnt;
    }
};
int main()
{
    int m,n,w,h;
    priority_queue<Point>q;
    queue<Point>q1;
    cin>>m>>n>>w>>h;
    for(int i=1;i<=m;i++){
        cin>>day[i];
    }
    st[0]=0;
    for(int i=1;i<=n;i++){
        cin>>per[i];
        st[i]=per[i];
    }
    for(int i=1;i<=n;i++){
        for(int j=i+1;j<=i+w-1;j++){
            st[j]-=st[i];
        }
    }
    bool f=1;
    //for(int i=n-w+2;i<=n;i++){
     //   if(st[i])f=0;
    //}
    for(int i=1;i<=m;i++){
        q.push(Point(i,day[i],1));
    }
    for(int i=1;i<=n;i++){
        while(q1.size()&&q1.front().next==i){
            Point now=q1.front();
            q1.pop();
            q.push(now);
        }
        while(st[i]--){
            if(q.size()==0){f=0;break;}
            Point now=q.top();
            q.pop();
```

```
            if(i+w-1>n){f=0;break;}
            v[now.id].push_back(i);
            now.next=i+w+h;
            now.cnt-=w;
            if(now.cnt!=0)q1.push(now);
        }
    }
    if(q.size()||q1.size())f=0;
    if(f)cout<<1<<endl;
    else {
        cout<<-1<<endl;return 0;
    }
    for(int i=1;i<=m;i++){
        int len=v[i].size();
        for(int j=0;j<len;j++){
            printf("%d%c",v[i][j],(j==len-1)?'\n':' ');
        }
    }
}
```

# GYM 102220E 贪心+最小生成树 (网 17-4 张学峰)

## 一、题号与题目名称

GYM 102220E Minimum Spanning Tree

## 二、题意简述

给一棵 n 个节点的带权树，把其中每条边视作一个点，如果另一个边有和这个边有相同的邻接点，那么视作的这两个点建一条边边权为原来的两条边权之和，求新建的图的最小生成树。

$1 \le T \le 1000, 2 \le n \le 100000, \sum n \le 1e6.$

## 三、主要知识点及其运用

本题涉及贪心思想。

## 四、完整解题思路描述

首先思考暴力的想法，在这棵树中，对于一个点，如果我们建图找最小生成树的话，如果这个点连了 x 条边，那么每条边化作一个点，也就会建立 C(x,2)条边，这样建出的边会极其多，边存不下，同时最小生成树也会超时。

那么对于一树中一个点，他会建出一个完全图，对于每一个树中的点都会有一个完全图，在最后的那颗最小生成树中，我们只要贪心的使得边权和最小即可。

由于这是一棵树，所以他相邻的点建出的完全图最多只有一个公共点，如果有两个以上公共点那么一定出现了环，这与题目给的是树是矛盾的。

那么既然所有的完全图是联通的而相邻的完全图且有且一个公共点，所以对于每个完全图内部求最小生成树，保证了完全图内部的权值和最小也保证了联通性，当全部的点都这样考虑当然度数只有 1 的点不考虑那么每小部分的权值和最小，联通性也保障了，那么全部的权值和一定就最小。

一个完全图内部的最小生成树，为（x-2）*Min，x 为一个点邻接的点的个数，Min 为边权最小的边

## 五、技巧与坑点

本题所给的是一棵树，这是整个题的突破点，每两个相邻节点的完全图有且仅有一个公共点，本题的技巧在于贪心最小生成树的边的定位，坑点在于联通性的证明。

## 六、参考文献与学习资料

http://codeforces.com/gym/102220

## 七、完整代码

```
#include<bits/stdc++.h>
#define ll long long
const ll MAXN = 1e6+5;
const ll inf = 0x7fffffff;
using namespace std;
struct node
{
    ll v,val;
    node(){}
    node(ll a,ll b){v=a;val=b;}
};
```

```
ll ans=0;
vector<node>vv[MAXN];
int main()
{
	ll t;
	scanf("%I64d",&t);
	while(t--)
	{
		ll n;
		ans=0;
		scanf("%I64d",&n);
		for(int i=0;i<=n;i++)vv[i].clear();
		for(ll i=1;i<=n-1;i++){
			ll u,v,val;
			scanf("%I64d%I64d%I64d",&u,&v,&val);
			vv[u].push_back(node(v,val));
			vv[v].push_back(node(u,val));
		}

		//cout<<Min<<endl;
		for(ll i=1;i<=n;i++){
			ll sz=vv[i].size();
			ll Min=inf;
			if(sz==1)continue;
			for(ll j=0;j<sz;j++){
				ll val=vv[i][j].val;
				ans+=val;
				Min=min(Min,val);
			}
			ans+=(sz-2)*Min;
		}
		printf("%I64d\n",ans);
	}
}
```

# HDU 6252 差分约束(网 17-4 张学峰)

## 一、题号与题目名称

HDU 6252 Subway Chasing

## 二、题意简述

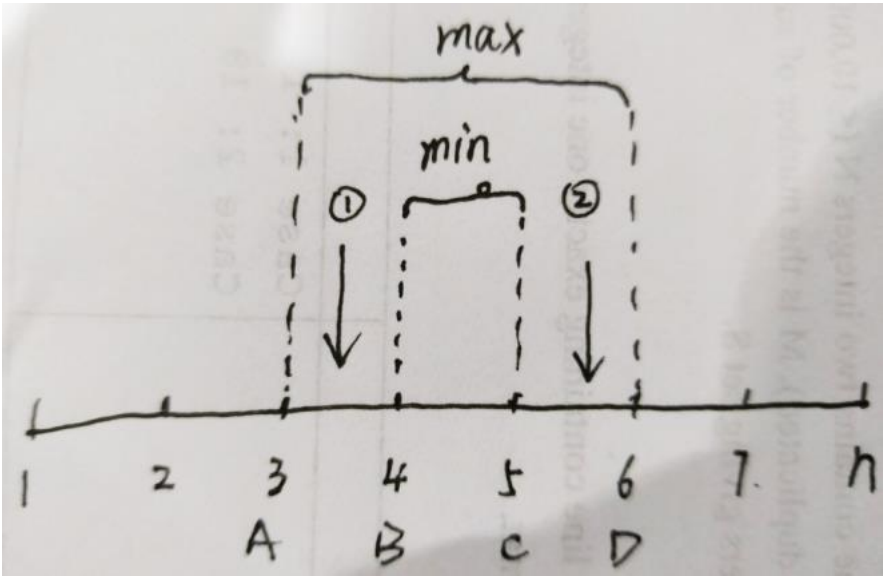
有两个人在 1 号点要到 n 号点去，小 a 出发比小 b 早 X 分钟，然后会有 m 个时间点，每个时间点都会给你小 a 所在区间（A,B）A，B 最大相差 1，以及小 b 所在区间（C,D）C,D 最大相差 1。输出一组可行的一到 n 之间每段所花费时间满足条件的解。

1≤T≤30， 1≤N,M≤2000， 1≤X≤10，1≤A,B,C,D≤N， A≤B≤A+1， C≤D≤C+1

## 三、主要知识点及其运用

本题考察差分约束。

## 四、完整解题思路描述




本题使用差分约束，主要难点在不等式的构造，对于一般情况，如图，一号在（3，4）之间，二号在（5，6）之间，那么两者最小的距离差就是趋近于 B 到 C 但是不等于 B 到 C，而最大就是趋近于 A 到 D 但是不等于 A 到 D,所以这一个条件可以构造出两个不等式 Sc - Sb < x 且 Sd – Sa > x ，对于在单点的情况同上处理一样，这里构建的不等式都要改成等号。

特别地，当两个人都在单点上时，那么就是一个确定的距离之差比方说 Sd-Sa=x，不等式要写成 Sd – Sa >=x 且 Sd-Sa <= x 。

## 五、技巧与坑点

本题重点在建边，坑点在于如何处理特殊情况。

## 六、参考文献与学习资料

https://blog.csdn.net/qq_41645482/article/details/89407877

http://acm.hdu.edu.cn/showproblem.php?pid=6252

## 七、完整代码

```
#include<iostream>
```

```
#include<string.h>
#include<algorithm>
#include<vector>
#include<stdio.h>
#include<queue>
using namespace std;
const int MAXN=5e4+5;
const int INF=0x7fffffff;
struct Point
{
    int v,c;
    Point(){};
    Point(int a,int b){v=a,c=b;}
};
vector<Point>v[MAXN];
int dis[MAXN],cnt[MAXN];
bool vis[MAXN];
int n,m,x;
void spfa()
{
    memset(vis,0,sizeof(vis));
    for(int i=0;i<n;i++){
        dis[i]=INF;
    }
    vis[n]=1;
    dis[n]=0;
    queue<int>q;
    while(!q.empty())q.pop();
    q.push(n);
    bool f=1;
    memset(cnt,0,sizeof(cnt));
    cnt[n]=1;
    while(!q.empty())
    {
        int u=q.front();
        q.pop();
        vis[u]=0;
        for(int i=0;i<v[u].size();i++){
            int vv=v[u][i].v;
            if(dis[vv]>dis[u]+v[u][i].c){
                dis[vv]=dis[u]+v[u][i].c;
                if(!vis[vv]){
                    vis[vv]=1;
                    q.push(vv);
                    if(++cnt[vv]>n){f=0;break;}
```

```
                }
            }
        }
        if(f==0)break;
    }
    if(f==0)cout<<" IMPOSSIBLE"<<endl;
    else{
        for(int i=1;i<n;i++){
                //cout<<dis[i]<<endl;
            cout<<" "<<dis[i]-dis[i-1];
        }
        cout<<endl;
    }
}
int main()
{
    int t,cnt=0;
    scanf("%d",&t);
    while(t--)
    {
        scanf("%d%d%d",&n,&m,&x);
        for(int i=0;i<=n;i++){
            v[i].clear();
        }
        for(int i=1;i<n;i++){
            v[i].push_back(Point(i-1,-1));
            v[n].push_back(Point(i,0));
        }
        for(int i=1;i<=m;i++){
            int a,b,c,d;
            scanf("%d%d%d%d",&a,&b,&c,&d);
            a--,b--,c--,d--;
            if(a==b&&c==d){
                v[c].push_back(Point(a,-x));
                v[a].push_back(Point(c,x));
            }
            else{
                v[d].push_back(Point(a,-(x+1)));
                v[b].push_back(Point(c,(x-1)));
            }
        }
        cout<<"Case #"<<++cnt<<":";
        spfa();
    }
}
```

# HDU 6230 (Manacher+树状数组\主席树) (网 17-4 张学峰)

## 一、题号与题目名称

HDU 6230 Palindrome

## 二、题意简述

给一个字符串，求存在多少“一个半回文串”。 一个半回文串：

(x 个字母)+ (回文中心) +(x-1 个字母)+(回文中心)+(x 个字母)

例如: a b c b a b c

1<=n<=500000

## 三、主要知识点及其运用

本题考察 Manacher 和树状数组的组合应用。

## 四、完整解题思路描述

分析什么时候会出现一个半回文串，只有两个回文串的回文半径互相相交时，就会出现一个半回文串。回文半径明显可以用 Manacher 算法处理，所以问题转化成 a[i]表示以 i 为中心的回文串的半径，如何求相交的对数。

此时容易想到对于每个 i 处理一个 b[i]表示回文串最右端，用主席树处理回文串左端到回文串中心这段有多少个右端点大于 i 的就好了，但是有更方便、更快捷的数据结构来实现也就是树状数组。先预处理一个 vector 数组，存入以 i 为左端点的回文串的中心坐标，然后遍历数组，取出 vector 中的每一项，用树状数组在对应位置+1，并且查询 [ i+1 , i 的回文右端点 ]的个数。

## 五、技巧与坑点

本题主要难在求交叉的区间对数，考察树状数组的灵活运用。

## 六、参考文献与学习资料

https://blog.csdn.net/qq_41645482/article/details/98373225

http://acm.hdu.edu.cn/showproblem.php?pid=6230

## 七、完整代码

### 树状数组做法：

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const ll MAXN = 5e5+5;
int b[MAXN],a[MAXN];
char s[MAXN],Ma[MAXN*2];
int Mp[MAXN*2],c[MAXN];
void add(int x,int k)
{
    for(;x<=MAXN;x+=x&-x)
        c[x]+=k;
}
```

```
ll ask(int x)
{
    ll ans=0;
    for(;x;x-=x&-x)
        ans+=c[x];
    return ans;
}
void Manacher(char s[],int len)  ///  Manacher
{
    int l=0;
    Ma[l++]='$';
    Ma[l++]='#';
    for(int i=0;i<len;i++){
        Ma[l++]=s[i];
        Ma[l++]='#';
    }
    Ma[l]=0;
    int mx=0,id=0;
    for(int i=0;i<l;i++){
        Mp[i]=mx>i?min(Mp[2*id-i],mx-i):1;
        while(Ma[i+Mp[i]]==Ma[i-Mp[i]])Mp[i]++;
        if(i+Mp[i]>mx){
            mx=i+Mp[i];
            id=i;
        }
    }
}
ll sum[MAXN*30];
int L[MAXN*30],R[MAXN*30],T[MAXN];
int cnt=0;
vector<int>v[MAXN];
int main()
{
    ll ans;
    int t;
    scanf("%d",&t);
    while(t--)
    {
        cnt=0;
        scanf("%s",s);
        memset(c,0,sizeof(c));
        int len=strlen(s);

        Manacher(s,len);
        ans=0;
```

```
            int n=len;
            for(int i=1;i<=n;i++){
                v[i].clear();
            }
            for(int i=1;i<2*len+2;i++){
                if(Ma[i]!='#'){
                    int pos=i/2,now=(Mp[i]-1)/2;
                    b[pos]=min(pos+now,n);
                    a[pos]=max(1,pos-now);
                    v[a[pos]].push_back(pos);
                }
            }
            for(int i=1;i<=n;i++){
                int len=v[i].size();
                for(int j=0;j<len;j++){
                    int now=v[i][j];
                    add(now,1);
                }
                ans+=ask(b[i])-ask(i);
            }
            printf("%lld\n",ans);
        }
    }
```

## 主席树做法：

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const ll MAXN = 5e5+5;
int b[MAXN],a[MAXN];
char s[MAXN],Ma[MAXN*2];
int Mp[MAXN*2];
void Manacher(char s[],int len)    ///    Manacher
{
    int l=0;
    Ma[l++]='$';
    Ma[l++]='#';
    for(int i=0;i<len;i++){
        Ma[l++]=s[i];
        Ma[l++]='#';
    }
    Ma[l]=0;
    int mx=0,id=0;
```

```
        for(int i=0;i<l;i++){
            Mp[i]=mx>i?min(Mp[2*id-i],mx-i):1;
            while(Ma[i+Mp[i]]==Ma[i-Mp[i]])Mp[i]++;
            if(i+Mp[i]>mx){
                mx=i+Mp[i];
                id=i;
            }
        }
}
ll sum[MAXN*30];
int L[MAXN*30],R[MAXN*30],T[MAXN];
int cnt=0;

int update(int pre,int l,int r,int x)     ///建主席树
{
        int rt=++cnt;
        L[rt]=L[pre];R[rt]=R[pre];sum[rt]=sum[pre]+1;
        int mid=(l+r)/2;
        if(l<r){
            if(x<=mid)L[rt]=update(L[pre],l,mid,x);
            else R[rt]=update(R[pre],mid+1,r,x);
        }
        return rt;
}
ll query(int u,int v,int l,int r,int k)    ///查询区间大于 k 的有多少个
{
        ll ret=0;
        int mid=(l+r)/2;
        if(k<=l)return sum[v]-sum[u];
        if(k<=mid)ret += query(L[u],L[v],l,mid,k);
        ret+=query(R[u],R[v],mid+1,r,k);
        return ret;
}
int main()
{
        ll ans;
        int t;
        scanf("%d",&t);
        while(t--)
        {
            cnt=0;
            scanf("%s",s);
            int len=strlen(s);
            Manacher(s,len);
            ans=0;
```

```
            int n=len;
            for(int i=1;i<2*len+2;i++){
                if(Ma[i]!='#'){
                    int pos=i/2,now=(Mp[i]-1)/2;
                    b[pos]=min(pos+now,n);a[pos]=max(1,pos-now);
                }
            }
            T[0]=0;
            for(int i=1;i<=n;i++){
                T[i]=update(T[i-1],1,n,b[i]);
            }
            for(int i=1;i<=n;i++){
                    ans+=query(T[a[i]-1],T[i-1],1,n,i);
            }
            printf("%lld\n",ans);
        }
    }
```

# GYM 102056L （思维+暴力)(计 17-2 吕昌昊)

## 一、题号与题目名称

GYM 102056L Eventual ... Journey

## 二、题意简述

在一个不一定连通的图上有 n 个点 m 条边，每个点的点权是 1 或者 0，在图中可以进行两种操作，跳跃和移动。跳跃操作可以让你从一个点跳到与其点权相同的点，移动操作可以让你从一个点沿着边走到下一个点。每种操作的价值都是 1。对于每个点，求出从该点到所有点的最短距离∑(j=1...n)(d(i,j))，并分别输出。

- n，m(均为 1e5)。
- 一行 n 个数字代表每个点的权值。
- m 行每行两个数字代表一条边。
- 保证任意两点可达，无重边。

## 三、主要知识点及其运用

本题涉及简单的分类讨论与模拟技巧。

## 四、完整解题思路描述

首先我们列出所有的操作可能：

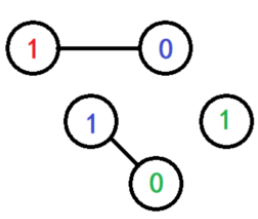
0——移动——1
0——跳跃——0
0——移动——1——跳跃——1
0——跳跃——0——移动——1
0——跳跃——0——移动——1——跳跃——1
1——移动——0
1——跳跃——1
1——移动——0——跳跃——0
1——跳跃——1——移动——0
1——跳跃——1——移动——0——跳跃——0

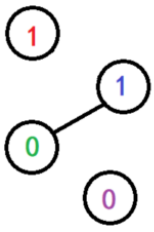
可见不存在任何一条路径长度超过 3，换句话说，从长度为三的路径再走一步一定不是最短路。

将上述所有从 1 开始的路径都拿出来模拟一下：

经过模拟发现一个点权为(x)的点一共可以分成几种情况讨论：

直接走到所有的(x)点，共花费 sum(x)-1。
直接走到所有直接相连的(!x)点，共花费 cnt(x)。
如果没有直接相连的(!x)，则花费 3*t(!x)。
然后再花费 2*(sum(x)-t(!x))走完全部点。




其中 sum(x)代表权值为(x)的点的个数。
cnt(x)为这个权值为(x)的点直接连接的(!x)点的数量。
t(x)代表权值为(x)并且没有(!x)直接相连的点的数量（被孤立）。

## 五、技巧与坑点

本题主要难在找到并计算长度为 3 的路径贡献。

## 六、参考文献与学习资料

https://www.bilibili.com/video/av38542305?from=search&seid=12042768026819448470

## 七、完整代码

```
#include <bits/stdc++.h>
using namespace std;

int a[100005];
vector<int> V0[100005];
vector<int> V1[100005];
int cnt0[100005];
int cnt1[100005];
int main(){
    memset(cnt0,0,sizeof cnt0);
    memset(cnt1,0,sizeof cnt1);
    int n,m;
    int sum0 = 0,sum1 = 0;///0 的数量，1 的数量
    cin >> n >> m;
    for(int i = 1;i <= n;i ++){
        cin >> a[i];
        sum1 += a[i];
    }
    sum0 = n - sum1;
    for(int i = 0;i < m;i ++){
        int fr,to;
        cin >> fr >> to;
        if(a[fr] != a[to]){///起点终点权值相同等于无边
            if(a[fr]){
                cnt0[fr] ++;///起点为 1，连接的 0 的数量
                cnt1[to] ++;///终点为 0，连接的 1 的数量
            }
            else{
                cnt1[fr] ++;///起点为 0，连接的 1 的数量
                cnt0[to] ++;///终点为 1，连接的 0 的数量
            }
        }
    }
    int t0 = 0,t1 = 0;///孤立的 0，孤立的 1
    for(int i = 1;i <= n;i ++){
        if(!a[i] && !cnt1[i]){t0 ++;}///第 i 个点是 0 并且这个点没连接任何 1
        if(a[i] && !cnt0[i]){t1 ++;}///第 i 个点是 1 并且这个点没连接任何 0
    }
```

```
    for(int i = 1;i <= n;i ++){
        if(i != 1){cout << " ";}
        int ans = 0;
        if(a[i] == 1){///第 i 个点是 1
            if(cnt0[i]){///第 i 个点连接了若干个 0
                ans = sum1 - 1;///权值为 1 的点能跳跃到所有 1
                ans += cnt0[i];///权值为 1 的点能移动到所有相邻的 0
                ans += 2 * (sum0 - cnt0[i]);
                ///剩下的权值为 0 的点要先走相邻的 0 再跳过去
            }
            else{///第 i 个点没有连接任何 0，是个被孤立的 1
                ans = sum1 - 1;///权值为 1 的点能跳跃到所有 1
                ans += 3 * t0;///想到达所有的孤立的 0 都要先跳到一个有 0 相邻的 1
                              ///然后从 1 移动到 0，再从 0 跳到这个孤立的 0
                ans += 2 * (sum0 - t0);
                ///剩下的 0 就可以先跳这个 0 相连的 1 再移动到 0
            }
        }
        else{///第 i 个点是 0
            if(cnt1[i]){///第 i 个点连接了若干个 1
                ans = sum0 - 1;///权值为 0 的点能跳跃到所有 0
                ans += cnt1[i];///权值为 0 的点能移动到所有相邻的 1
                ans += 2 * (sum1 - cnt1[i]);
                ///剩下的权值为 1 的点要先走相邻的 1 再跳过去
            }
            else{///第 i 个点没有连接任何 1，是个被孤立的 0
                ans = sum0 - 1;///权值为 0 的点能跳跃到所有 0
                ans += 3 * t1;///想到达所有的孤立的 1 都要先跳到一个有 1 相邻的 0
                              ///然后从 0 移动到 1，再从 1 跳到这个孤立的 1
                ans += 2 * (sum1 - t1);
                ///剩下的 1 就可以先跳这个 1 相连的 0 再移动到 1
            }
        }
        cout << ans;
    }
    cout << endl;

    return 0;
}
```

# GYM 102056F （计算几何）(计 17-2 吕昌昊)

## 一、题号与题目名称

GYM 102056F Interstellar ... Fantasy

## 二、题意简述

给定空间内点 O(Ox,Oy,Oz)，作以 R 为半径的球的球心，又给定空间内两点 A(Ax,Ay,Az)、B(Bx,By,Bz)，求解 A 到 B 的最短距离。路径不可穿过球体，但可以在球面行走。保留精度至 1e-6。

## 三、主要知识点及其运用

本题涉及基本的计算几何能力，对于三角函数，余弦定理等计算。

## 四、完整解题思路描述

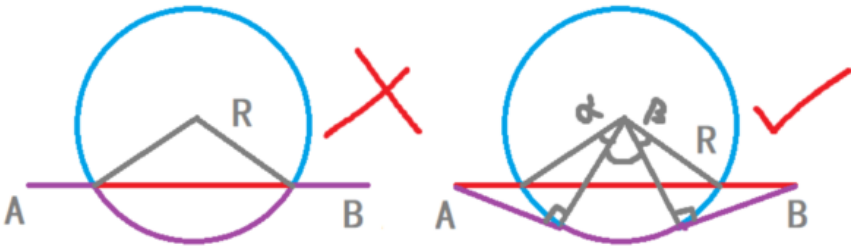
一道相对简单的计算几何，最短路径一定在一个平面内，所以将三维化为二维，此题会变得十分直观。

明显 A、B 两点连接的线段只有以下几个情况：

1) A、B 同点，距离直接输出 0
2) A、B 都在圆心一侧
3) A、B 所连线段与圆心距离大于等于 R
4) A、B 所连线段与圆心距离小于 R

情况 2 判断：我们发现当∠OAB 或∠OBA 有一个大于等于 90° 了，那么两个点 A、B 就一定不会与圆相交，也就是说答案就是 dist(A,B)。但是如果两个角都是锐角，那我们还不能确定这条直线是否与圆有交点。所以这种情况要进行下面的判断。

情况 3、4 判断：在情况 2 判断过后，我们剩下的计算就是圆心到线段距离与 R 的大小关系。如果 R> 距离 D，那答案就是 dist(A,B)，否则就是两个线段长与一段弧长的和。




## 五、技巧与坑点

本体要点在于能否判断出上图的情况，对于 R > D 的情况能否找到正确的路线。

## 六、参考文献与学习资料

https://www.bilibili.com/video/av38542305?from=search&seid=12042768026819448470

## 七、完整代码

```
#include <bits/stdc++.h>
using namespace std;
const double pi = acos(-1);
double O_r;
struct node{
    double x,y,z;
```

```
    }O,s,t;
    double p2(double x){
        return x * x;
    }

    double dist(node a,node b){
        return p2(a.x - b.x) + p2(a.y - b.y) + p2(a.z - b.z);
    }

    double cosin(double a,double b,double c){
        return ((a + b - c)/(2 * sqrt(a) * sqrt(b)));
    }

    int main(){
        int T;scanf("%d",&T);
        while(T --){
            scanf("%lf%lf%lf%lf",&O.x,&O.y,&O.z,&O_r);
            scanf("%lf%lf%lf%lf%lf%lf",&s.x,&s.y,&s.z,&t.x,&t.y,&t.z);
            if(s.x == t.x && s.y == t.y && s.z ==
t.z){printf("0.00000000\n");continue;}
            double dis_os = dist(O,s);
            double dis_ot = dist(O,t);
            double dis_st = dist(s,t);
            double cos_ost = cosin(dis_os,dis_st,dis_ot);
            double cos_ots = cosin(dis_ot,dis_st,dis_os);
            double cos_sot = cosin(dis_os,dis_ot,dis_st);
            if(acos(cos_ost) >= pi / 2 || acos(cos_ots) >= pi / 2){
               printf("%.8f\n",sqrt(dis_st));continue;
            }
            double sin_ost = sqrt(1 - p2(cos_ost));
            double h = sqrt(dis_os) * sin_ost;
            if(h >= O_r){printf("%.8f\n",sqrt(dis_st));continue;}
            double Alpha = acos(O_r / sqrt(dis_os));
            double Beta = acos(O_r / sqrt(dis_ot));
            double Delta = acos(cos_sot) - Alpha - Beta;
            double ans1 = sqrt(dis_os) * sin(Alpha);
            double ans2 = sqrt(dis_ot) * sin(Beta);
            double ans3 = O_r * Delta;
            printf("%.8f\n",ans1 + ans2 + ans3);
        }

        return 0;
    }
```

# GYM 102056I （三维逆向 DP+滚动数组）(计 17-2 吕昌昊)

## 一、题号与题目名称

GYM 102056I Misunderstood ... Missing

## 二、题意简述

英雄有两种属性：攻击力 A 与攻击增量 D(初始均为 0)。每回合开始 A 都会增加 D。现有 n 个回合，每回合可以选择三种操作之一：

1、攻击敌人，造成 A+ai 点伤害。

2、使属性 D 增加 bi 点。

3、使属性 A 增加 ci 点。

求 n 回合之后，英雄能打出的伤害上限。攻击后英雄的 A，D 都会保持，不会归零。

回合数 n(1-100)。数组 a、b、c(均为 1e9)。

## 三、主要知识点及其运用

本题考察对 DP 中状态的推导判断以及对滚动数组的使用技巧。

## 四、完整解题思路描述

本题使用 DP 求解应该是比较明显的思路，因为我们无法通过证明每回合使用何种操作使伤害总值最大的贪心方式。但是我们无法使用伤害值(ans>1e9)进行 DP。只能从本题所给数据最小的 n(回合数)来考虑。

现在我们已经确定了第一维状态(回合数)，那么还有什么在空间允许范围内的状态呢。

既然 n 已经很小了，那么 n 回合中攻击的次数也不过 n 次罢了。所以不妨从攻击次数下手，但是就算我们知道了攻击次数，也无法在 100 以内枚举若干次攻击的回合，这个攻击回合的集合过于庞大，所以下一步应该尝试将这个集合转化成我们能处理的信息。

我们知道每回合是一个三选一：

如果选一(攻击)，获得的收益是立即的(A+ai)，并且是对所有状态都没有影响的。

如果选二(D+=bi)，因为 D 是一个增量，每个回合都会产生影响。此时我们好像必须知道哪个回合进行了攻击，才能确定收益的和。

设集合 x 为第 i 回合选了二之后的回合中选择攻击的回合，一共有 j 次，那么我们此次选择获得的收益为：

[(x1-i) + (x2-i) + … + (xj-i)] * bi = [∑(x – i)] * bi = [∑x – (i*j)] * bi

我们发现此次的收益是与在 i 回合后所有攻击回合的下标和相关。

如果选三(A+=ci)，此时我们就不用知道具体的攻击回合了，在 i 回合后攻击了多少次，就产生了 ci*j 的贡献。

经过刚才的一顿推理，我们又得到了两个需要知道的东西：j(在 i 回合后攻击的次数)和∑x(这 j 次攻击的回合下标和)。因为只有 100 回合，所以求和之后的∑x 不过 5050，此时的空间复杂度为 100*100*5050。所以我们还要使用滚动数组对空间进行优化。

DP 的维度解决了，下面解决怎么 DP。我们知道了第二维代表在 i 回合后的攻击次数，既然是在…之后，我们很明显的不能正向推，因为处理 j 时，求不出 j 到底是多少。不能正向，那就反向，反向处理明显比正向处理要方便得多， j 已经在 j+1 处得到了，而且∑x 这个维度也可以反向得到。最有价值的是，第 n 回合我们一定选择攻击，所以 DP 的初始状态直接就得到了(dp[n][1][n] = a[n])。所以此题的 DP 应该是逆向的。

那么随着另外两个 DP 的维度的推出，并且知道了应该逆向处理以及如何优化。那么此题至此终于可以进行求解了。

下面列出转移状态以及方程。

状态 i：当前回合

状态 j：当前回合之后的所有回合中的选一的回合数

状态 k：当前回合之后的所有选一回合的下标和。

方程 1：dp[i][j+1][k+i] = max(dp[i][j+1][k+i] , dp[i+1][j][k] + ai)

这个方程代表第 i 回合是否选择操作一。

方程 2：dp[i][j][k] = max(dp[i][j][k] , dp[i+1][j][k] + max ( (j * ci) , (k- i*j) * bi) ) )

这个方程代表第 i 回合不选操作一，但之后的回合攻击力会上升，上升的收益是选二或选三的较大者。

## 五、技巧与坑点

本题重点在于如何在给定的条件内找到合适的转移状态以及能否想到逆向处理 DP 的思路，坑点在于能不能计算枚举上下界，如果上下界计算错误就会造成结果的错误。

## 六、参考文献与学习资料

https://www.bilibili.com/video/av38542305?from=search&seid=12042768026819448470

https://blog.csdn.net/xianpingping/article/details/86378057

## 七、完整代码

```
#include<bits/stdc++.h>
using namespace std;

long long dp[2][105][5055];
long long a[105],b[105],c[105];
int main(){
    int T,n;
    scanf("%d",&T);
    while(T --){
        memset(dp,0,sizeof dp);
        scanf("%d",&n);
        for(int i = 1;i <= n;i ++){
            scanf("%lld%lld%lld",&a[i],&b[i],&c[i]);
        }
        dp[n % 2][1][n] = a[n];
        for(int i = n - 1;i >= 1;i --){
            for(int j = 1;j <= n - i;j ++){
                int low = (i + i + j + 1) * j / 2;
                int high = (n - j + 1 + n) * j / 2;
                for(int k = low;k <= high;k ++){
                    dp[i % 2][j + 1][k + i] = max(dp[i % 2][j + 1][k + i],
                                                 dp[(i + 1) % 2][j][k] + a[i]);
                    dp[i % 2][j][k] = max(dp[i % 2][j][k],dp[(i + 1) % 2][j][k]
                                          + max(c[i] * j,b[i] * (k - i * j)));
                }
            }
```

```
            memset(dp[(i + 1)% 2],0,sizeof dp[(i + 1) % 2]);
        }
        long long ans = -1;
        for(int i = 1;i <= 100;i ++){
            for(int j = 1;j <= 5050;j ++){
                ans = max(ans,dp[1][i][j]);
            }
        }
        printf("%lld\n",ans);
    }

    return 0;
}
```

# GYM 102056C （中国剩余定理+优化枚举)(计 17-2 吕昌昊)

## 一、题号与题目名称

GYM 102056C Heretical ... Möbius

## 三、题意简述

给给出一个长为 200 的 01 字符串（在题目中被分成 10 个 20 长度的字符串依次输入），代表 μ(x)至 μ(x+199)的绝对值，μ(i)代表 i 的莫比乌斯函数值。要求在 1-1e9 的范围内是否有一个可行解 x 使得 200 个值一一对应。

## 三、主要知识点及其运用

本题需要了解莫比乌斯函数的定义以及对中国剩余定理以及扩展欧几里德的考察，还有对于常数级时间复杂度的优化，对于枚举的优化。

## 四、完整解题思路描述

莫比乌斯函数 μ(x)的定义：

1、对于 x=1， μ(x)=1

2、如果 x 能拆分成 k 个不同素数的积且 k 是奇数， μ(x)=-1

3、如果 x 能拆分成 k 个不同素数的积且 k 是偶数， μ(x)=1

4、剩余的情况， μ(x)=0

其实对于情况 4，x 一定存在一个平方数因子。

11101110011011101010
11100100111011101110
11100110001010101110
11001110111011001110
01101110101011101000
11101110111011100110
01100010111011001110
11101100101001101110
10101110010011001110
11101110011011101010

莫比乌斯函数的前 200 项的绝对值（样例 1）如上所示。我们可以发现 4 的倍数，9 的倍数，25 的倍数，49 的倍数等等素数平方数的位置都是 0。也就是说对于一个大小为 200 的区间，我们能使用 4,9,25,49,121,169 这 6 个数字，以求出起点位置对于每个数的模数可能是多少，然后枚举所有集合，使用中国剩余定理求出可能解并判断是否为正解。

枚举 4 的时候，j 循环从 0 跑到 3，字符串下标从 0 开始。

| 下标 | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 样例 | 1 | 1 | 1 | 0 | 1 | 1 |
| 下标 | 6 | 7 | 8 | 9 | 10 | 11 |
| 样例 | 1 | 0 | 0 | 1 | 1 | 0 |

此时 j=0，就相当于我认为第 0 位的答案下标是 4 的倍数，也就是 0，但很明显不是，也就是说(4-0)不可行。一直判断到 j=3，我们发现 j+k*4 的所有字符串下标都是 0（k 是常数），那么就相当于 4 的一个可能的解是到下标 3 是 0，那么其模数就是 1。

使用一个 vector 数组 V[]存放对于每个枚举的质平方数的可能解（不是可行解）。比如对于 4 找到模数

2，那么在 4 对应的 vector 里插入一个 2。但是这种枚举方式存在缺陷，如果所输入的字符串是一个纯 0 串，那么对于所有的 i、j 都是可能解，在这种情况下，解的数量就变成了 4*9*25*49*121*169。这个数量级过大，一定会超时。此时进行优化一，判断所给字符串中 0 的数量大于一个值是一定无解的。经过程序跑出的答案是 93，CF 数据不够极限，87 就可以过。小于 93 时，极限解数量还不过 2e5，完全可以暴力。有一篇题解是用判断 6 个集合乘积大小的优化方式，∏V[i].size()>10000 就是无解情况，此处没有证明出来。

由于 4，9，25，49，121，169 都是互质的，因此中国剩余定理可以对这六个同余方程进行求解 x，求解过程中还需要使用 EXGCD 求出 ti。

令$M=\prod_{i=1}^{k} m_i$，即M是所有$m_i$的最小公倍数

$t_i$为同余方程$\frac{M}{m_i}t_i \equiv 1(\mod m_i)$的最小非负整数解

则有一个解为$x=\sum_{i=1}^{k} a_i \frac{M}{m_i} t_i$

通解为$x+i*M(i\in Z)$

特别的，最小非负整数解为$(x\%M+M)\%M$

在中国剩余定理求解之后，暴力验证我们得到的 x 是不是正解，令循环 i 从 x 到 x+199，如果 str[i]是 0，其实就是 i 存在平方因子，否则不存在。那么此时使用优化二，由于判断平方因子一般都是 sqrt(n) 的，n 还大，判断次数还多，所以我们应该预处理比较大的平方数。

我选择的方法是将 1000 以内的质数筛出来，然后对于大于 1000 的数字，我把所有的小于 1e9 的 k*i*i 都放在 set 里维护，对于一个待判断的值，先跑质数表，再使用 S.count()就能够很快得到所判断的值是否存在平方因子。

## 五、技巧与坑点

本题主要难在如何使用中国剩余定理求解，坑点就是如何优化常数，是一个典型的卡常题。

## 六、参考文献与学习资料

https://blog.csdn.net/wxh010910/article/details/86136119

https://www.bilibili.com/video/av38542305?from=search&seid=12042768026819448470

## 七、完整代码

```
#include<bits/stdc++.h>
using namespace std;
const int inf = 0x3f3f3f3f;

char in[205];
int P2[10] = {4,9,25,49,121,169};
int ans,X[10]; ///存放一组解
vector<int> V[10];
bool P[1005];///素数表
set<int> S;

void init(){
   memset(P,false,sizeof P);
```

```
    for(int i = 2;i <= 1001;i ++){///1000以内素数
        int f = 1;
        for(int j = 2;j * j <= i;j ++){
            if(i % j == 0){f = 0;break;}
        }
        if(f){P[i] = true;}
    }
    for(int i = 997;i * i <= 1e9;i ++){///set存放所有拥有1000以上平方因子的数
        for(int j = i * i;j <= 1e9;j += i * i){
            S.insert(j);
        }
    }
    ans = inf;
}

int input(){
    for(int i = 0;i < 10;i ++){
        scanf("%s",in + (20 * i));
    }
    int cnt_Z = 0;
    for(int i = 0;i < 200;i ++){
        if(in[i] == '0'){cnt_Z ++;}
    }
    return cnt_Z;
}

bool check(int a,int d){
    for(int i = a;i < 200;i += d){
        if(in[i] == '1'){return false;}
    }
    return true;
}

long long gcd(long long a,long long b,long long &G,long long &x,long long &y){
    if(!b){G = a,x = 1,y = 0;}
    else{gcd(b,a % b,G,y,x),y -= (x * (a / b));}
}

bool hasSqrtFactor(int x){///判断是否有平方因子
    for(int i = 2;i <= 997;i ++){
        if(!P[i]){continue;}
        if(x % (i * i) == 0){return true;}
    }
    return S.count(x);
}
```

```
bool checkRes(int Res){
    for(int i = 0;i < 200;i ++){
        int t = hasSqrtFactor(Res + i);
        if(in[i] == '1' && t){return false;}
        if(in[i] == '0' && !t){return false;}
    }
    return true;
}

int CRT(){ ///CRT 求解
    long long M = 901800900;///4*9*25*49*121*169
    long long G,y,x = 0;
    for(int i = 0;i < 6;i ++){
        long long m = M / P2[i];
        gcd(P2[i],m,G,G,y);
        x = (x + y * m * X[i]) % M;
    }
    int res = (x + M) % M;
    if(checkRes(res)){return res;}
    if(res + M <= 1000000000 - 199 && checkRes(res + M)){
        return res + M;
    }
    return inf;
}

void dfs(int x){ ///枚举所有的解集
    if(x == 6){
        ans = min(ans,CRT());
        return ;
    }
    for(int i = 0;i < V[x].size();i ++){
        X[x] = V[x][i];
        dfs(x + 1);
    }
}

int main(){
    init();
    int Zero = input();
    if(Zero > 93){puts("-1");return 0;}
    for(int i = 0;i < 6;i ++){
        for(int j = 0;j < P2[i];j ++){
            if(check(j,P2[i])){
                V[i].push_back(P2[i] - j);
```

```
                ///枚举可能模数
            }
        }
    }
    ///int Sum = V[0].size()*V[1].size()*V[2].size()*V[3].size()*V[4].size()*V[5].size();
    ///if(Sum >= 10000){puts("-1");return 0;}
    dfs(0);
    if(ans == inf){puts("-1");}
    else{printf("%d\n",ans);}
    return 0;
}
```

# GYM 102056G （复杂模拟+二分查找）(计 17-2 吕昌昊)

## 一、题号与题目名称

GYM 102056G Omnipotent ... Garland

## 二、题意简述

在对于给定的只含字母 BC 的串，串长为 n，问能否将其分割成 m 个子序列，每个子序列的长度都是 k 的倍数，并且满足子序列中有两个相邻的字母 B。（字符串视为一个环）一条边。

数据组数 T(1e5)，n,m,k(均为 1e6)，长度为 n 的字符串 s，sum_n < 1e6。

## 三、主要知识点及其运用

本题涉及复杂的模拟与二分查找的简单操作。

## 四、完整解题思路描述

首先考虑无解情况：

| | |
|---|---|
| n % k != 0 | 此时无法分割为若干个长度为 k 倍数的串 |
| m * k > n | 串太长 |
| cnt_B > m * 2 | 每个子序列至少有两个 B，数量不够 |

可以证明剩下的情况都一定有解。

如果 m == 1：

明显此时只需要判断是否存在两个 B 相邻。

如果 m == 2：

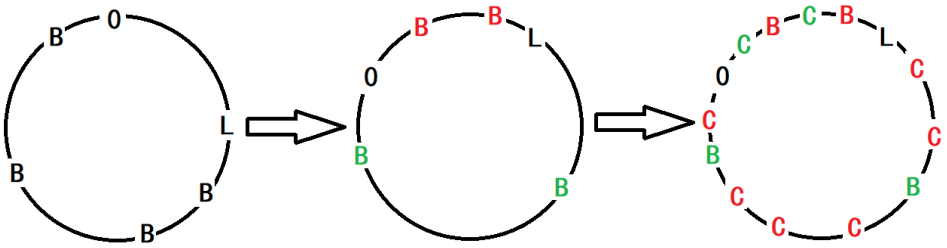
假设我们找到 4 个 B，其位置分别为 k0，k1，k2，k3。那么对于任意两个 B(k0 和 k1)进行枚举，把这两个 B 中间的所有 C 放到下一个序列。这样会剩下很多 C，再把这些 C 进行分配。Check 一下是否可以成功分配。如果不行就找下一对 B(k1 和 k2，k2 和 k3)。

如果 m > 2：

和 m == 2 的情况类似，可以先用同种策略将 B 取到仅剩下 4 个，然后再按照 m == 2 的情况跑一遍。判断的时候更加复杂，这也是此题最难的地方，需要十足的耐心和细心，还需要较强的数据处理能力，考察基本功。

这道题可以有很多种模拟方法，不一定需要使用二分查找。

在代码中，可以把多余的 B 都换成 C，只留下 2 * m 个 B，可以证明这样处理并不会造成无解。然后对于 2 * m 个 B 进行一个比较复杂的模拟即可。处理过程中可以每次找到两个相邻的 B，在集合 C 中进行二分查找，这样就能找到一段 C，这段 C 的长度是 k - 2，对于不同情况将字母从串中剔除直到剩下 4 个 B，此时整个串一定能被分割成两个序列且满足要求，但是长度却不一定是 2 * k 了，并且使用上面的方法不一定会得到最优解，这时我们处理出某一个序列的可能长度 L，然后转动这个环，直到前 L 个字母里出现了两个 B，这样就能像下图一样直接暴力出两个合法序列了。



## 五、技巧与坑点

本题需要十足的耐心与细心，虽然不涉及任何高级数据结构和算法知识，但是对于基本功的考验却是实实在在的，这也奠定了此题的难度，现场榜单中的定位是金牌题（虽然 JLS 定位的是银牌）。

## 六、参考文献与学习资料

https://www.bilibili.com/video/av38542305?from=search&seid=12042768026819448470

## 七、完整代码

```
#include <bits/stdc++.h>
using namespace std;

int n,m,k;
string in;

void solve_1(){
    bool f = false;
    for(int i = 0;i < n;i ++){
        if(in[i] == 'B' && in[(i + 1) % n] == 'B'){
            f = true;
            break;
        }///找到一对相邻的 B 就可以
    }
    if(!f){cout << "No" << endl;return ;}
    cout << "Yes" << endl;
    cout << n;
    for(int i = 0;i < n;i ++){
        cout << " " << i;
    }cout << endl;
    return ;
}

void solve_2(){
    vector<int> B,C;
    for(int i = 0;i < n;i ++){
        if(in[i] == 'B' && B.size() >= 2 * m){in[i] = 'C';}
        if(in[i] == 'B'){B.push_back(i);}
        else{C.push_back(i);}
    }
    printf("Yes\n");
    int s = max(k - 2,0);///子序列中 C 的数量
    while(B.size() > 4){
        int p1 = lower_bound(C.begin(),C.end(),B[B.size()-2]) - C.begin();
        ///找到第一个 B 的位置
        int p2 = lower_bound(C.begin(),C.end(),B.back()) - C.begin();
        ///找到第二个 B 的位置
```

```
        cout << s + 2;///子序列长度
        if(p1 >= s){////{x,x,x,c1,c2,c3,c4...cs,p1}
            for(int i = p1 - s;i < p1;i ++){
                cout << " " << C[i];
            }
            cout << " " << B[B.size() - 2] << " " << B.back() << endl;
            C.erase(C.begin() + p1 - s,C.begin() + p1);
        }
        else if(C.size() - p2 >= s){////{x,x,x,p1,x,x,x,p2,c1,c2,c3...cs}
            cout << " " << B[B.size() - 2] << " " << B.back();
            for(int i = C.size() - s;i < C.size();i ++){
                cout << " " << C[i];
            }cout << endl;
            C.erase(C.end() - s,C.end());
        }
        else{////{x,x,x,p1,c1,c2...cs,p2}
            cout << " " << B[B.size() - 2];
            for(int i = p2 - s;i < p2;i ++){
                cout << " " << C[i];
            }cout << " " << B.back() << endl;
            C.erase(C.begin() + p2 - s,C.begin() + p2);
        }
        B.pop_back();B.pop_back();
    }
    B.insert(B.end(),C.begin(),C.end());///把 B 集合和 C 集合合并
    sort(B.begin(),B.end());///排序，此时仅剩下两个序列未处理
    int L = (B.size() / 2 / k) * k;///计算两个序列其中一个的长度
    ///cout << L << endl;
    int cnt = 0;
    for(int i = 0;i < L;i ++){
        if(in[B[i]] == 'B'){cnt ++;}
    }///看看第一个序列有没有两个 B
    vector<int> ans[2];
    for(int i = 0; ;i ++){///转动这个圆环
        if(cnt == 2){
            for(int j = 0;j < B.size();j ++){
                if(j >= i && j < i + L){///落在 L 里的 B 是序列 0，C 是序列 1
                    if(in[B[j]] == 'B'){ans[0].push_back(B[j]);}
                    else{ans[1].push_back(B[j]);}
                }
                else{///落在另外一段里的 B 是序列 1，C 是序列 0
                    if(in[B[j]] == 'B'){ans[1].push_back(B[j]);}
                    else{ans[0].push_back(B[j]);}
                }
            }
```

```
            break;
        }
        if(in[B[i + L]] == 'B'){cnt ++;}
        if(in[B[i]] == 'B'){cnt --;}
    }
    for(int i = 0;i < 2;i ++){
        cout << ans[i].size();
        for(int j = 0;j < ans[i].size();j ++){
            cout << " " << ans[i][j];
        }cout << endl;
    }
}

int main(){
    int T;cin >> T;
    while(T --){
        cin >> n >> m >> k;
        cin >> in;
        int B_cnt = 0;
        for(int i = 0;i < n;i ++){
            if(in[i] == 'B'){B_cnt ++;}
        }
        if((n % k) || ((long long)m * k > n) || (B_cnt < m + m)){
            cout << "No" << endl;continue;
        }
        if(m == 1){solve_1();}
        else {solve_2();}
    }

    return 0;
}
```

# GYM 102220D 树链剖分套珂朵莉树 (计 17-4 段学松)

## 一、题号与题目名称

2019 东北赛 D. Master of Data Structure

## 二、题意简述

* 给以一颗有 n 个节点的树，每个节点初始值都是 0. 给你 m 次操作,n<5e5,m<2000
* 操作有：
  * 操作 1:对一条链上每一个数都加 x
  * 操作 2:对一条链上每一个数都异或 x
  * 操作 3:对一条链上每一个数都减 x(如果该数小于 x，则不减)
  * 操作 4:询问一条链上的数的和
  * 操作 5:询问一条链上的数的异或和
  * 操作 6:询问一条链上的数的最大值减最小值
  * 操作 7:询问一条链上与 k 最接近的数.

## 三、主要知识点及其运用

本题涉及树链剖分以及珂朵莉树等数据结构。

## 四、完整解题思路描述

读完题之后，看到树上路径操作，应该马上想到树链剖分，树链剖分之后就把树上的问题转化成序列的问题了。注意到操作次数只有 2000 次，我们可以利用珂朵莉树时间复杂度只与操作次数有关的性质，用珂朵莉树维护区间即可。时间复杂度 O（m^2logm）,然而题目给了 6s 是没问题的。

珂朵莉树的所有操作几乎都是暴力，所以我们只需要按照题目要求，直接按照每一项暴力模拟即可。珂朵莉树的具体实现，请参考完整代码部分。

## 五、技巧与坑点

本题技巧就是利用珂朵莉树的时间复杂度只与操作次数有关，而本题的操作只有 2000 次，这样可以直接树链剖分套珂朵莉树暴力解决问题。另外,调用底层库 pbds 可以减少 stl::set 的常数，加上读入优化可以让代码跑的更快。

注意珂朵莉树 split 操作时要先分右区间，然后再分左区间，否则会 RE。

## 六、参考文献与学习资料

题目链接：http://codeforces.com/gym/102220/problem/D
珂朵莉树教程：https://www.luogu.org/problemnew/solution/CF896C

## 七、完整代码

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>///调用底层库
#include <ext/pb_ds/tree_policy.hpp>///调用底层库
#define en putchar('\n')
using namespace std ;
using namespace __gnu_pbds ;
template <typename T>///底层库红黑树的声明.
class pbTree: public tree<T,null_type,less<T>,rb_tree_tag,tree_order_statistics_node_update>{};
///底层库实现一个 set,操作一次的时间复杂度为 logn,但是常数比 stl 小很多.
```

```
int64_t mins,maxs ;
class Chtholly_Tree
{
public:
    struct Node
    {
        int l,r;mutable int64_t val ;
        Node(int _l,int _r=-1,int64_t _val=0):l(_l),r(_r),val(_val){}
        bool operator < (const Node& b) const{return l<b.l ;}
    };
    Chtholly_Tree (int n=0){if(n) clear(n);}
    pbTree<Node> s ;///声明一个 set
    inline void clear(int n) {s.clear() ;s.insert(Node(n+1,n+1,0));/*!!别忘了插入!!!!!*/}
    inline void insert(int l,int r,int64_t val=0)///插入一段区间[l,r] -> val
        {s.insert(Node(l,r,val)) ;}
    inline auto spilt(int mid) -> decltype (s.begin())
    { ///找出左端点为 mid 的区间.如果不存在,
      ///则把包含 mid 的区间拆分成[l,mid-1]和[mid,r].O(logn)
        auto it = s.lower_bound(Node(mid)) ;
        if(it!=s.end()&&it->l == mid) return it ;
        -- it ;
        int l(it->l),r(it->r);
        int64_t val(it->val);
        s.erase(it) ;s.insert(Node(l,mid-1,val)) ;
        return s.insert(Node(mid,r,val)).first ;
    }

    inline int64_t Operate(int l,int r,int id,int64_t val=0)
    {///对区间的操作[包括询问,修改等等,直接暴力] O(klogn)
        int64_t ans(0) ;
        auto itR(spilt(r+1)),itL(spilt(l)) ;
        for(;itL!=itR&&itL!=s.end();++itL) {///直接暴力每一个值相等的区间
            switch (id)
            {
                case 1:
                    itL->val += val;break;
                case 2:
                    itL->val ^= val;break ;
                case 3:
                    if(itL->val<val) break ;
                    itL->val -= val ;
                    break ;
                case 4:
                    ans += itL->val*(itL->r-itL->l+1) ; break ;
                case 5:
```

```
                    ans ^= ((itL->r-itL->l+1)&1)?itL->val:0 ; break ;
                case 6:
                    maxs = max(maxs,itL->val);
                    mins = min(mins,itL->val);
                    break ;
                case 7:
                    mins = min(abs(itL->val-val),mins) ;
                    break ;
            }
        }
        return    ans;
    }

};
Chtholly_Tree myTree ;///声明一个珂朵莉树

const int MAXN = 500005;
const int MAXM = 500005;
struct Edge {int to,Next;}edge[MAXM<<1];
int head[MAXN],tot;

void add_edge(int u,int v)
{
    edge[tot].to = v ;
    edge[tot].Next = head[u];
    head[u] = tot ++ ;
}

void addedge(int u,int v)
{
    add_edge(u,v) ;add_edge(v,u) ;
}

int Size[MAXN],top[MAXN],fa[MAXN],deep[MAXN],Son[MAXN],id[MAXN],cur ;///树链剖分用到的定义

void dfs1(int pos=1,int _deep = 1,int _fa=0)
{
    deep[pos] = _deep ;
    fa[pos] = _fa ;
    Son[pos] = 0 ;
    Size[pos] = 1 ;
    for(int i=head[pos];i!=-1;i = edge[i].Next) {
        int p = edge[i].to;
        if(p == _fa) continue ;
        dfs1(p,_deep+1,pos) ;
```

```
            Size[pos] += Size[p] ;
            if(Size[p] > Size[Son[pos]]) Son[pos] = p ;
        }
}

void dfs2(int pos=1,int _top=1)
{
        id[pos] = ++cur ;
        top[pos] = _top ;
        if(Son[pos]) dfs2(Son[pos],_top) ;
        for(int i = head[pos];i!=-1;i = edge[i].Next) {
            int to = edge[i].to;
            if(to == fa[pos]||to == Son[pos]) continue ;
            dfs2(to,to) ;
        }
}

void Init(int n)
{
        tot = cur = 0 ;
        for(int i = 0 ; i <= n; ++i) {
            head[i] = -1 ;
        }
}

int64_t Q(int u,int v,int _id,int64_t val=0)
{
        int64_t ans (0) ;
        mins = 0x3f3f3f3f;maxs = 0 ;
        while(top[u]!=top[v]) {
            if(deep[top[u]]<deep[top[v]]) swap(u,v) ;
            switch (_id)
            {
            case 1:case 2:case 3:case 6:case 7:
                myTree.Operate(id[top[u]],id[u],_id,val) ;
                break ;
            case 4:
                ans += myTree.Operate(id[top[u]],id[u],_id,val) ;
                break ;
            case 5:
                ans ^= myTree.Operate(id[top[u]],id[u],_id,val) ;
                break ;
            }
            u = fa[top[u]];
        }
```

```
    if(deep[u]>deep[v]) swap(u,v);
    switch (_id)
    {
    case 1:case 2:case 3:case 6:case 7:
        myTree.Operate(id[u],id[v],_id,val) ;
        break ;
    case 4:
        ans += myTree.Operate(id[u],id[v],_id,val) ;
        break ;
    case 5:
        ans ^= myTree.Operate(id[u],id[v],_id,val) ;
        break ;
    }
    if(_id == 6) return maxs - mins ;
    if(_id == 7) return mins ;
    return ans ;
}

const int SIZ =100000 +3;
char buf1[SIZ];
char *p1=buf1,*p2=buf1;
inline char getcharEx(){///究极 getchar
    if(p1==p2)p1=buf1,p2=buf1+fread(buf1,1,SIZ,stdin);
    return p1==p2?EOF:*p1++;
}

template <class T>
inline void read(T &x) {///究极快读
    static char ch;static bool f;
    for(ch=f=0;ch<'0' || '9'<ch;f|=ch=='-',ch=getcharEx());
    for(x=0;'0'<=ch && ch<='9';x=((x+(x<<2))<<1)+ch-'0',ch=getcharEx());
    x=f?-x:x;
}

template <class T>
inline void write(T x)
{
    if (x < 0) putchar('-'),x = -x;
    if (x >= 10) write (x / 10);
    putchar(x % 10 + '0');
}

int main()
{
    int T,n,m ;
```

```
    read(T) ;
    while(T--)
    {
        read(n);read(m) ;
        myTree.clear(n) ;
        myTree.insert(0,n,0LL) ;
        Init(n) ;
        int u,v ;
        for(int i(1);i<n;++i) {
            read(u);read(v) ;
            addedge(u,v) ;
        }
        dfs1();dfs2() ;///树链剖分两遍 dfs.
        int op,l,r;
        int64_t val ;
        while(m--)
        {
            read(op);read(l);read(r) ;
            if(op<=3||op == 7) {
                read(val) ;
                int64_t ans(Q(l,r,op,val)) ;
                if(op == 7) {write(ans);en;}
            } else {
                write(Q(l,r,op)) ;en ;
            }
        }
    }
    return 0;
}
```

# 计蒜客 CONTEST 1555I 回文自动机+预处理(计 17-4 段学松)

## 一、题号与题目名称

JISUANKE CONTEST 1555 I.Skr

## 二、题意简述

给出一个由数字 '0' – '9' 组成的字符串,求这个串所有本质不同的回文串作为数字加在一起的和并对 1e9+7 取余数。

## 三、主要知识点及其运用

本题主要用到预处理的思想，以及回文树等数据结构。

## 四、完整解题思路描述

首先思考暴力的想法，我们直接用 Manacher 算法跑出所有的回文字串，然后用字符串 hash 判重。这样子做显然会超时。

我们分析一下这样做超时的原因,无非是由于 Manacher 算法不能很好的利用不同回文串之间的关系，这时候我们需要一种新的数据结构来解决回文串问题。

我们定义回文自动机并构造一个回文自动机。回文自动机由两颗树构成，两棵树分别维护长度为奇数的回文串和长度为偶数的回文串。在最开始的时候，回文树只有两个节点，分别为 0 号节点和 1 号节点。0 号节点用来表示长度为偶数的回文串的树根，1 号节点用来表示长度为奇数的回文串的树根，类似字典树，树上的每个节点都对应一种类型的回文串。每增加一个节点，都会产生新的种类的回文串。我们可以称树上的每一个节点为一个状态。类似 AC 自动机，回文自动机的每一个节点 p 也都有 fail 指针，指向 p 对应的回文串的最长回文后缀。我们利用不断判断 fail 指针对应的状态来配合完成插入字符的操作。我们构造完回文自动机以后，并不是用回文自动机完成匹配，而是用回文自动机构造完的各个状态来获取信息。当插入完所有的字符以后，节点个数 tot-2 就是本质不同的回文串的个数。

学会了定义回文自动机的话，这道题就变成了回文自动机的模板题。我们只创建一颗回文自动机，然后遍历每一个回文状态即可。

如果用 len[i]表示 i 号状态回文串的长度，用 str[]数组表示原字符串，no[i]表示状态 i 对应的回文串的右端点在原 str 数组中的位置，那么该状态在 str 数组中的位置就是[i-len[no[i]]+1,i]。我们只需要把中间的字串转换成数字进行累加，然后输出答案就行了。

## 五、技巧与坑点

直接转换成数字会 TLE 到飞的，我们得先预处理一下，注意到 4567 = 1234567 - 1230000，所以预处理一下 10 的 n 次方记做 powten[]，再对原串做一次前缀和，记作 val[]，然后需要哪一段的话，直接做差就行了，具体细节请参考完整代码部分。

## 六、参考文献与学习资料

题目链接：https://nanti.jisuanke.com/t/A1955
回文自动机教程：https://www.luogu.org/problemnew/solution/P3649

## 七、完整代码

```
#include<bits/stdc++.h>
#define mod(x) (((x)%Mod+Mod)%Mod)
const int64_t Mod (1000000007) ;
```

```
using namespace std ;
/**
类似 AC 自动机的一种回文串匹配自动机,也就是一棵字符树.准确的说,是两颗字符树,0 号表示回文串长度为偶数的树,1 号表示回文串长度为奇数的树.每一个节点都代表一个字符串,并且类似 AC 自动机那样,有字符基个儿子,它的第 i 个儿子就表示将字符基的第 i 个字符接到它表示的字符串两边形成的字符串.
同样类似 AC 自动机的是,每一个节点都有一个 fail 指针,fail 指针指向的点表示当前串后缀中的最长回文串.
特殊地,0 号点的 fail 指针指向 1,非 0,1 号点并且后缀中不存在回文串的节点不指向它本身,而是指向 0.

功能:
    1.求串 S 本质不同回文串的个数(tot-2)
    2.求串 S 内每一个本质不同回文串出现的次数(cnt[i])
    3.求串 S 内回文串的个数(其实就是 1 和 2 结合起来)
    4.求以下标 i 结尾的回文串的个数(num[i])
    5.求以 i 结尾的最长回文串的长度(len[i])
    6.本质不同回文串的位置
**/
//回文字动机每次新建点就相当于增加了一种回文串.

char str[2000005] ;
int64_t val[2000005],powten[2000005] ;
int getVal(int l,int r)
{
    return mod(val[r]-(val[l-1]*powten[r-l+1])) ;
}
class Palindromic_Tree {
public:
    #define DEBUG(x) cout<<#x<<" == "<<x<<","
    #define MAXN   2000005///字符串的长度
    #define N 10 ///字符集大小
    int Next[MAXN][N] ;//Next 指针,Next 指针和字典树类似,指向的串为当前串两端加上同一个字符构成
    int fail[MAXN] ;//fail 指针,表示最长回文后缀的状态编号,失配后跳转到 fail 指针指向的状态
    int cnt[MAXN] ;/*调用 count()之后它表示该状态出现的次数*/
    int num[MAXN]/*表示回文后缀的个数,也表示 fail 指针的深度*/ ;
    int len[MAXN]/*表示该状态的回文串的长度*/,S[MAXN] ;/*存放添加的字符*/
    int last ;/*指向上一个字符所在的状态的编号,方便下一次 insert*/
    int id[MAXN]/*i 字符在树中的状态编号*/,no[MAXN]/*i 状态在原串中的位置*/;
    int n /*添加的字符个数*/,tot/*添加的节点个数*/;
    Palindromic_Tree(){clear() ;}
    inline int newNode (int l) {///新建节点
        memset(Next[tot],0,sizeof(Next[tot])) ;
        num[tot] = cnt[tot] = 0 ;
        len[tot] = l ;
        return tot ++ ;
    }
    inline void clear () {///初始化
```

```
    tot = 0 ;
    newNode (0) ;/*0 表示存放偶数长度串的树的根*/
    newNode ( -1 )/*1 表示存放奇数长度串的树的根*/ ;
    last = n = 0 ;fail[0] = 1 ;///刚开始,0 号状态的 fail 指针要指向 1 号状态.
    S[n] = -1 ;//开头放一个字符集中没有的字符,减少特判
}
inline int get_fail (int x) {///和 KMP 一样,失配后找一个尽量最长的
    while (S[n-len[x]- 1]!= S[n]) x = fail[x] ;
    return x ;
}
inline void insert(const char* str,int n=0)
{
    n = n?n:strlen(str);
    val[0] = 0 ;powten[0] = 1 ;
    for(int i(1);i<=n;++i) {///预处理 10 的次方以及原字符串的前缀和
        val[i] = (val[i-1]*10+str[i-1]-'0')%Mod ;
        powten[i] = (powten[i-1] * 10) %Mod;
    }
    for(int i(0);i<n;++i) id[i] = insert(str[i]) ;
}
inline int insert (int c) {///插入的时候返回状态的编号.
    S[++n] = c -= '0'; ///!!!要是大写字母的话,注意这里!!!
    int cur(get_fail(last)) ;//通过上一个回文串找这个回文串的匹配位置
    if (!Next[cur][c]) {//如果这个回文串没有出现过,说明出现了一个新的本质不同的回文串
        int now (newNode(len[cur] + 2)) ;//新建状态节点
        fail[now] = Next[get_fail(fail[cur])][c] ;//和 AC 自动机一样建立 fail 指针,以便失配后跳转
        Next[cur][c] = now ;
        num[now] = num[fail[now]] + 1 ;///更新 fail 指针深度
    }
    ++ cnt[last = Next[cur][c]] ;///更新次数
    no[last]=n;///存储当前状态对应的原字符串的位置.
    return last ;
}
inline int getNum(int i) ///返回以 i 结尾的不同回文串个数
    {return num[id[i]] ; }
inline int getCnt(int i) ///返回调用 count()后 S 中等于以 i 结尾的最长回文串的字串个数
    {return cnt[id[i]];}
inline int getLongest(int i)///返回以 i 为结尾的最长回文串长度
    {return len[id[i]] ;}
inline int size() ///返回本质不同的回文字串数目
    {return tot-2;}
inline int64_t count ()
{///  统计 S 中所有节点 i 表示的本质不同的串的个数
    int64_t ans(0) ;
    for (int i(tot-1);i>=0;--i) {///逆序相加,每个点都会比它的父亲节点先算完,于是父亲节点能加到所有
```

```
子孙.
                cnt[fail[i]] += cnt[i] ;
                if(i<2) continue ;
                int r(no[i]),l(r-len[i]+1) ;
                ans = (ans + getVal(l,r))%Mod ;
                ///这里也可以算其他贡献
            }
            return ans ;
        }
        #undef N
        #undef MAXN
} ;
Palindromic_Tree tree ;

int main()
{
        scanf("%s",str);
        tree.clear() ;
        tree.insert(str) ;
        printf("%lld\n",tree.count());
        return 0 ;
}
/*
一组测试数据:
in: 999999999999999
out: 103333318
*/
```

# GYM 100548G 回文自动机+DFS （计 17-4 段学松）

## 一、题号与题目名称

GYM 100548 G. The Problem to Slow Down You

## 二、题意简述

给了两个字符串，问有多少个字符串对，这个字符串对要求，S1 出现在第一个字符串，S2 出现在第二个字符串且 S1=S2，并且是回文串。

## 三、主要知识点及其运用

本题考察回文自动机的应用以及 DFS。

## 四、完整解题思路描述

本题只需要建立两颗回文自动机，分别存下两个字符串，然后分别对它们的奇根并行 dfs 跑一遍和偶根 dfs 跑一遍，把所有存在在两个回文树里的并且相等的串都统计一下就好了。

也就是说在 dfs 的过程中 ans += cnt1[i]*cnt2[j] ;

## 五、技巧与坑点

本题重点就是考察建立回文自动机,并且在两颗树上并行 dfs，并行 dfs 这里不太好想，注意不要写错了。

## 六、参考文献与学习资料

题目链接: http://codeforces.com/gym/100548/

## 七、完整代码

```
#include <bits/stdc++.h>
using namespace std ;

class Palindromic_Tree {
public:
    #define DEBUG(x) cout<<#x<<" == "<<x<<","
    #define MAXN  200005///字符串的长度
    #define N 26 ///字符集大小
    int Next[MAXN][N] ;//Next 指针,Next 指针和字典树类似,指向的串为当前串两端加上同一个字符构成
    int fail[MAXN] ;//fail 指针,表示最长回文后缀的状态编号,失配后跳转到 fail 指针指向的状态
    int cnt[MAXN] ;/*调用 count()之后它表示该状态出现的次数*/
    int num[MAXN]/*表示回文后缀的个数,也表示 fail 指针的深度*/ ;
    int len[MAXN]/*表示该状态的回文串的长度*/,S[MAXN] ;/*存放添加的字符*/
    int last ;/*指向上一个字符所在的状态的编号,方便下一次 insert*/
    int id[MAXN]/*i 字符在树中的状态编号*/,no[MAXN]/*i 状态在原串中的位置*/;
    int n /*添加的字符个数*/,tot/*添加的节点个数*/;
    Palindromic_Tree(){clear() ;}
    inline int newNode (int l) {///新建节点
```

```
        memset(Next[tot],0,sizeof(Next[tot])) ;
        num[tot] = cnt[tot] = 0 ;
        len[tot] = l ;
        return tot ++ ;
    }
    inline void clear () {///初始化
        tot = 0 ;
        newNode (0) ;/*0 表示存放偶数长度串的树的根*/
        newNode ( -1 )/*1 表示存放奇数长度串的树的根*/ ;
        last = n = 0 ;fail[0] = 1 ;///刚开始,0 号状态的 fail 指针要指向 1 号状态.
        S[n] = -1 ;//开头放一个字符集中没有的字符,减少特判
    }
    inline int get_fail (int x) {///和 KMP 一样,失配后找一个尽量最长的
        while (S[n-len[x]- 1]!= S[n]) x = fail[x] ;
        return x ;
    }
    inline void insert(const char* str,int n=0)
    {
        n = n?n:strlen(str);
        for(int i(0);i<n;++i) id[i] = insert(str[i]) ;
    }
    inline int insert (int c) {///插入的时候返回状态的编号.
        S[++n] = c -= 'a'; ///!!!要是大写字母的话,注意这里!!!
        int cur(get_fail(last)) ;//通过上一个回文串找这个回文串的匹配位置
        if (!Next[cur][c]) {//如果这个回文串没有出现过,说明出现了一个新的本质不同的
回文串
            int now (newNode(len[cur] + 2)) ;//新建状态节点
            fail[now] = Next[get_fail(fail[cur])][c] ;//和 AC 自动机一样建立 fail
指针,以便失配后跳转
            Next[cur][c] = now ;
            num[now] = num[fail[now]] + 1 ;///更新 fail 指针深度
        }
        ++ cnt[last = Next[cur][c]] ;///更新次数
        no[last]=n;///存储当前状态对应的原字符串的位置.
        return last ;
    }
    inline void count ()
    {/// 统计 S 中所有节点 i 表示的本质不同的串的个数
        for (int i(tot-1);i>=0;--i) {///逆序相加,每个点都会比它的父亲节点先算完,
于是父亲节点能加到所有子孙.
            cnt[fail[i]] += cnt[i] ;
            ///这里也可以算其他贡献
        }
    }
    #undef N
```

```
    #undef MAXN
} ;
Palindromic_Tree tree1,tree2 ;

int64_t ans ;
void dfs(int pos1,int pos2)
{
    if(pos1!=0&&pos1!=1) ans += 1LL*tree1.cnt[pos1]*tree2.cnt[pos2] ;
    for(int i(0);i<26;++i) {
        int nxt1(tree1.Next[pos1][i]);
        int nxt2(tree2.Next[pos2][i]) ;
        if(nxt1&&nxt2) {
            dfs(nxt1,nxt2) ;
        }
    }
}

char str1[2000005],str2[2000005] ;
int main()
{
    int T,t(0) ;
    scanf("%d",&T) ;
    while(T--)
    {
        scanf("%s%s",str1,str2) ;
        tree1.clear() ;tree2.clear() ;
        tree1.insert(str1) ;tree2.insert(str2) ;
        tree1.count() ;tree2.count() ;
        ans = 0 ;
        dfs(1,1) ;
        dfs(0,0) ;
        printf("Case #%d: %lld\n",++t,ans) ;
    }
}
```

# HDU 5992 kdTree 求空间最临近(计 17-8 江绪桢)

## 二、题号与题目名称

2016ACM/ICPC 亚洲区青岛站 K

## 三、题意简述

在一个城市有若干个旅馆，每个旅馆有一个固定的价格。题中给出每个旅馆的坐标和价格。

有若干个客人需要住旅馆。每个人有一个可以接受的最大价格。每次询问给出一个客人的位置坐标和他可以接受的最大价格。要求输出距离客人最近的，并且客人付得起钱的旅馆。如果有多个旅馆距离客人一样远并且价格都在客人可接受的范围内，那么输出这些旅馆中编号最小的一个。

## 三、主要知识点及其运用

本题涉及 kdTree 的建立，空间划分和空间最邻近点的查询。

## 四、完整解题思路描述

每个旅馆有位置坐标和价格三个维度，要求找出满足约束条件的最近点。属于 kdTree 的基本应用。直接使用三个维度划分三维空间建立 kdTree，然后对点集进行搜索并通过回溯确定最近点位置。

## 五、技巧与坑点

旅馆价格作为第三个维度在实际意义上与空间直角坐标系的 z 轴不同，不能直接当作 z 轴理解。在搜索时应当做特殊处理。

## 六、参考文献与学习资料

题目链接：http://acm.hdu.edu.cn/showproblem.php?pid=5992
kdTree 教程：https://blog.csdn.net/silangquan/article/details/41483689

## 七、完整代码

```
#include <bits/stdc++.h>
#define MAXN 200010
using namespace std;
int T,n,q;
struct point {
    int data[3];///data[0]表示 x 轴坐标，data[1]表示 y 轴坐标，data[2]表示价格
    ///写成数组形式存储是为了后续方便比较
    int id;
    point() {}
    point(int a,int b,int c,int id) {
        data[0] = a;
        data[1] = b;
        data[2] = c;
        this->id = id;
    }
} poi[MAXN],beifeng[MAXN];///poi 会被重新排序，使用 beifeng 记录原始位置，使用 id 变量进行映射

struct node { ///kdtree 的一个节点
    point poi;
```

```
    node* leftchild;
    node* rightchild;
    int div;///div 表示这个节点是按照哪个维度分割点集的
}*root,*tail,mem[MAXN];///mem 是开辟了一片存储空间存储 kdTree，tail 指针指向这片存储空间中第一个还没有被使用的地址

void kdTreePrep() {
    tail = mem;
    root = mem;
}

int64_t sqr(int s) { ///求平方
    return (int64_t)s*(int64_t)s;
}

int64_t sqrdistance(point a,point b) { ///求两个点欧拉距离的平方
    return sqr(a.data[0] - b.data[0])+sqr(a.data[1] - b.data[1]);
}

struct cmp {
    int div;
    cmp(int x):div(x) {}
    bool operator() (point a,point b) { ///如果返回 1，表示括号中第一个点在 div 维度上的值小于第二个点
        ///（如果两个点在 div 维度的值相同，则比较(div+1)%3 维度）
        for(int i = 0; i<=2; i++) {
            if(a.data[(div+i)%3] != b.data[(div+i)%3]) {
                return a.data[(div+i)%3]<b.data[(div+i)%3];
            }
        }
        return 1;
    }
};

node * buildKdTree(point* p,int l,int r,int div) {
    ///将 a[l]到 a[r]的所有节点按照 div 维度的值划分成两部分，挂在一个根节点两侧
    ///根节点选为从 a[l]到 a[r]的中位节点(类比于中位数)
    if(l>=r)
        return NULL;
    node*pp = tail++;///一个新节点被建立后，tail 指针向后挪一格
    pp->div = div;
    int mid = (l+r)/2;
    nth_element(p+l,p+mid,p+r,cmp(div));///寻找中位数
    pp->poi = p[mid];
    pp->leftchild = buildKdTree(p,l,mid,(div+1)%3);
    pp->rightchild = buildKdTree(p,mid+1,r,(div+1)%3);
```

```
    return pp;
}

int64_t mindis = 1e18,id = -1;

void search(point lookfor,node* p) { ///p 表示当前搜索到的节点，lookfor 就是我当前想要搜索的节点
    if(!p)
        return;///搜到空节点，返回
    int div = p->div;
    if(p->poi.data[2]<=lookfor.data[2]) { ///维度从 0 开始算，第二维存储的是客人可以接受的最大价格，
        //cout<<p->poi.data[0]<<" "<<p->poi.data[1]<<" "<<p->poi.data[2]<<" "<<lookfor.data[2]<<endl;
        ///所以如果这个价格不能接受，那么不论这个旅馆距离远近，都不能更新答案
        ///只有旅馆价格达到要求，才会去考虑远近并更新答案
        int64_t dis = sqrdistance(p->poi,lookfor);
        if(dis<mindis || (dis == mindis&&p->poi.id < id)) {
            mindis = dis;
            //cout<<"mindis = "<<mindis<<endl;
            id = p->poi.id;
        }///如果当前搜到的这个点距离客人的距离比已知的最短距离小，那么更新最短距离和目的地的
编号
        ///当前搜到的点与客人距离与之前搜到的点相同，但是这个点的编号更小，那么这个点也更优（题
意规定的），所以也更新
    }///维护当前经过的节点中的可能的最优答案
    if(div == 2) { ///如果当前搜到的节点的子树是按照价格划分平面
        if(lookfor.data[2] < p->poi.data[2]) { ///如果客人能接受的价格比当前搜到的这个节点的价格低
            ///那么肯定要向价格较低的方向去继续搜索,左子树总是比右子树小
            search(lookfor,p->leftchild);
        } else { ///如果客人能接受的价格比当前搜到的这个节点的价格高，那么旅馆价格可以适当增减，
            ///为了寻找到距离最近的，左右子树都要去搜索
            search(lookfor,p->leftchild);
            search(lookfor,p->rightchild);
        }
    } else {
        int64_t d = lookfor.data[div] - p->poi.data[div];
        d = d*d;
        if(lookfor.data[div] < p->poi.data[div]) {
            search(lookfor,p->leftchild);
            if(mindis>d) { ///如果当前搜到的点的划分线另一侧也有可能存在距离客人位置更近的点
                search(lookfor,p->rightchild);
            }
        } else {
            search(lookfor,p->rightchild);
            if(mindis>d) {
                search(lookfor,p->leftchild);
            }
```

```
        }
    }
}



int main() {

    scanf("%d",&T);
    while(T--) {
        scanf("%d%d",&n,&q);
        for(int i = 0; i<n; i++) {///必须从 0 开始，因为后面会使用 nth_element，参考 stl 迭代器，不从 0 开始容易 re
            int a,b,c;
            scanf("%d%d%d",&a,&b,&c);
            poi[i] = point(a,b,c,i);
            beifeng[i] = point(a,b,c,i);
        }
        kdTreePrep();
        root = buildKdTree(poi,0,n,0);
        point lookfor;
        while(q--) {
            int a,b,c;
            scanf("%d%d%d",&a,&b,&c);
            lookfor = point(a,b,c,0);
            mindis = 1e18;
            id = -1;
            search(lookfor,root);
            printf("%d %d %d\n",beifeng[id].data[0],beifeng[id].data[1],beifeng[id].data[2]);
        }
    }

    /**nth_elememt 的功能演示
    cout<<33<<" "<<24<<" "<<17<<" "<<4<<" "<<53<<" "<<6<<endl;
    for(int i = 0; i<=5; i++) {
        int xx[] = {33,24,17,4,53,6};
        cout<<"i = "<<i<<endl;
        nth_element(xx+0,xx+i,xx+6);
        for(int i= 0; i<=5; i++) {
            cout<<xx[i]<<" ";//x[i]左边的元素都比 x[i]小，x[i]右边的元素都比 x[i]大，其余元素的顺序不保证。
        }
        cout<<endl;
    }*/
}
```

# HDU 6219 计算几何+dp 求最大空凸包(计 17-8 江绪桢)

## 一、题号与题目名称

2017ACM/ICPC 亚洲区沈阳站 C

## 二、题意简述

给定一个整数 n(3<=n&&n<=50)。然后给出 n 个点的坐标（-1000<=x&&x<=1000，-1000<=y&&y<=1000）。要求从这 n 个点当中选出任意个，按顺序连接后可以框出一个凸多边形。凸多边形需要满足的要求是这个凸多边形内部不能含有任何其他的点。

要求找到面积最大的满足上述条件的多边形，输出其面积。

## 三、主要知识点及其运用

本题主要用到计算几何叉积和极角排序的方法，配合动态规划求解

## 四、完整解题思路描述

由于给定点集中点的个数很少，所以可以使用 O(n^3)的解法。枚举点集中每个点作为空凸包的起始点，考虑在所有位于起始点上方或右方的点建立空凸包。先使用极角排序给所有点编号，然后从小到大枚举所有的节点和起始点连线作为空凸包中最后一条边。使用之前记录的最大空凸包加上最后一个的三角形的面积可以进行 dp 的转移。

## 五、技巧与坑点

当存在两个点和起始点处于同一条直线上时，之能更新最终答案，但是不能更新 dp 数组存储的值，否则会导致下次转移时将距离起始点较近的一个点被包含在凸包内部，不符合最大空凸包的定义。

## 六、参考文献与学习资料

题目链接：http://acm.hdu.edu.cn/showproblem.php?pid=6219

## 七、完整代码

```
#include <bits/stdc++.h>
const double eps = 1e-8;
using namespace std;
inline int sqr(int x) {
    return x*x;
}
struct point {
    int x,y;
    point(){};
    point(int a,int b):x(a),y(b) {}
    friend point operator -(const point &a,const point &b){
        return point (a.x - b.x,a.y - b.y);
    }
};
int det(const point &a,const point &b){
    return a.x*b.y - a.y*b.x;
}
```

```
int dot(const point &a,const point &b){
    return a.x*b.x + a.y*b.y;
}
vector<point> p;
bool cmpMeiJu(point& a,point& b){
    if(a.y<b.y)return 1;
    else if(a.y == b.y){
        return a.x<b.x;
    }
    return 0;
}
point cankao;
bool cmpJiJiao(point &a,point &b){
    int k = det(a-cankao,b-cankao);
    if(k == 0){
        return abs(dot(a,a))<abs(dot(b,b));
    }else{
        return k>0;
    }
}
double EmpthConvexPolygons(vector<point> &a){
    double ans = 0;
    double dp[a.size()+10][a.size()+10];
    memset(dp,0,sizeof(dp));
    cankao = a[0];///因为前面已经经过排序，a[0]一定是这个点集中最下且最左的点，就是极角排序的参考点。
    sort(a.begin()+1,a.end(),cmpJiJiao);
    for(int i = 2;i<(int)a.size();i++){
        int j = i-1;
        while(j>=0 && det(a[i] - cankao,a[j] - cankao) == 0){
            j--;
        }
        bool flag = (j == i-1);
        while(j>=0){
            int k = j-1;
            while(k>=0&&det(a[j] - a[k],a[i] - a[j])<0){///没有形成凸多边形
                k--;
            }
            double area = ((double)abs(det(a[i] - a[0],a[j] - a[0])))/(2.0);
            if(k>=0) area+=dp[j][k];
            if(flag){
                dp[i][j] = area;
            }
            ans = max(ans,area);
            j = k;
```

```
            }
            if(flag){
                for(int jj = 1;jj<i;jj++){
                    dp[i][jj] = max(dp[i][jj],dp[i][jj-1]);
                }
            }
        }
        return ans;
}
double largestEmptyConvexPolygons(vector<point> &a){
        sort(a.begin(),a.end(),cmpMeiJu);
        auto it1 = a.begin(),it2 = a.end();
        double ans = 0;
        for(int i = 0;i<=(int)a.size()-3;i++){
            vector<point>b(it1+i,it2);
            ans = max(ans,EmpthConvexPolygons(b));
        }
        return ans;
}
int T,n;
int main() {
        scanf("%d",&T);
        while(T--) {
            scanf("%d",&n);
            p.clear();
            int a,b;
            for(int i = 1; i<=n; i++) {
                scanf("%d%d",&a,&b);
                p.push_back(point(a,b));
            }
            printf("%.1f\n",largestEmptyConvexPolygons(p));
        }
        return 0;
}
```

# HDU 5985 概率+数学推导 (计 17-8 江绪桢)

## 一、题号与题目名称

2016ACM/ICPC 亚洲区青岛站 D

## 二、题意简述

给 N（N<=10）种硬币，每种硬币有 ni 个，每个硬币每次投掷以后正面朝上的概率相等为 pi （ni 的总和不超过 1e6）。

每次都把所有的硬币全部投掷出去，然后把所有反面朝上的都扔掉。重复这个操作，直到只剩下一种硬币。剩下的这一种硬币作为幸运硬币。要求计算每一种硬币成为幸运硬币的概率。

## 三、主要知识点及其运用

本题考查概率公式的数学推导。

## 四、完整解题思路描述

本题中给定的概率值 p 在 0.4~0.6 之间，当硬币投掷次数很多时，几乎不会对后面的概率产生影响。直接取 200 作为近似的无穷值，推导出概率表达式后进行计算即可。

## 五、技巧与坑点

无穷值大小的选定，概率公式的推导与化简。

## 六、参考文献与学习资料

题目链接: http://acm.hdu.edu.cn/showproblem.php?pid=5985

## 七、完整代码

```
#include<bits/stdc++.h>
using namespace std;
int T;
int n;
struct {
    int num;
    double p;
} coins[20];
double die[20][205];
double ksm(double a,int b) {
    if(b == 0)
        return 1.0;
    double tmp = a;
    double ret = 1.0;
    while(b>0) {
        if(b&1) {
            ret *= tmp;
        }
        tmp*=tmp;
        b>>=1;
```

```
    }
    return ret;
}
int main() {
    scanf("%d",&T);
    while(T--) {
        scanf("%d",&n);

        for(int i = 1; i<=n; i++) {
            scanf("%d%lf",&coins[i].num,&coins[i].p);
        }
        if(n == 1) {
            printf("%.6f\n",1.0);
            continue;
        }
        for(int i = 1; i<=n; i++) {
            double p = coins[i].p;
            for(int j = 1; j<=201; j++) {
                die[i][j] = ksm((1-p),coins[i].num);
                p*=coins[i].p;
            }
        }
        for(int i = 1; i<=n; i++) {
            double ans = 0.0;
            for(int s = 1; s<=200; s++) {
                double k = die[i][s+1] - die[i][s];
                double other = 1.0;
                for(int j = 1; j<=n; j++) {
                    if(j == i) {
                        continue;
                    } else {
                        other*=die[j][s];
                    }
                }
                ans+=k*other;
            }
            printf("%.6f",ans);
            if(i!= n)
                printf(" ");
        }
        printf("\n");
    }
}
```

## HDU 5934 强联通分量+缩点(计 17-10 郭留阳)

## 四、题号与题目名称

2016 年中国大学生程序设计竞赛（杭州）B

## 五、题意简述

给出了 n 个炸弹的坐标，以及每个炸弹的爆炸半径，以及点燃这个炸弹的花费。求引爆所有炸弹的最小花费（当一个炸弹被引爆，该炸弹的爆炸半径内的炸弹都会被引爆）。

## 三、主要知识点及其运用

本题涉及到了强联通分量以及缩点。

## 四、完整解题思路描述

对每一个炸弹向它能够引爆的炸弹连有向边，发现在同一个强联通分量内（任意两点相互可达）的所有炸弹我们点燃花费最小的即可。进一步考虑，将原图求强联通分量后缩点是一个有向无环图，所有入度为 0 的联通块是需要被引爆的，我们引爆该联通块内花费最小的即可。

## 五、技巧与坑点

Tarjan 算法求联通块以及缩点。

## 六、参考文献与学习资料

强联通分量：https://blog.csdn.net/qq_21125183/article/details/80605172

## 七、完整代码

```
#include<bits/stdc++.h>
using namespace std;
#define LL long long
#define sca(x) scanf("%d",&x)
#define pb(x) push_back(x)

const int N = 2005;
struct node
{
    LL x,y,r,c;
} a[N];
vector<int>V[N];
int in[N];

void read(int n) //读入炸弹的坐标 半径以及花费
{
    for(int i=1; i<=n; i++)
    scanf("%lld%lld%lld%lld",&a[i].x,&a[i].y,&a[i].r,&a[i].c);
}

bool judge(int i,int j)//判断一个炸弹时候在另一个炸弹的爆炸范围
{
```

```
    LL del=1LL*(a[j].x-a[i].x)*(a[j].x-a[i].x)+(a[j].y-a[i].y)*(a[j].y-a[i].y);
    LL tmp=1LL*a[i].r*a[i].r;
    if(del<=tmp) return true;
    return false;
}

int s;
int vis[N];
vector<int>G[N];
int bel[N];
int low[N],df[N];
int clo;
stack<int>S;

void tarjan(int u)   //tarjan 相关 （求强连通分量）
{
    low[u]=df[u]=++clo;
    S.push(u);
    for(int i=0;i<V[u].size();i++)
    {
        int v=V[u][i];
        if(!df[v])
        {
            tarjan(v);
            low[u]=min(low[u],low[v]);
        }
        else if(!bel[v])
        {
            low[u]=min(low[u],df[v]);
        }
    }
    if(low[u]==df[u])
    {
        int tmp;
        ++s;
        do
        {
            tmp=S.top();
            S.pop();
            bel[tmp]=s;    //表示该店属于哪个强连通分量
            G[s].pb(tmp); //存储该连通分量都有哪些点
        }while(tmp!=u);
    }
}
```

```
void solve(int n)
{
    for(int i=1;i<=n;i++)   //建立的图可能不连通，需要多次求解
    {
        if(!df[i])
        {
            tarjan(i);
        }
    }

    for(int i=1;i<=n;i++)    //缩点相关
    {
        for(int j=0;j<V[i].size();j++)//枚举所有的点，如果两个点处于不同的连通分量内则统计度数。
        {
            if(bel[i]!=bel[V[i][j]])//这里我们并没有加边，此题只要统计度数即可。
            {
                in[bel[V[i][j]]]++;
            }
        }
    }
    LL ans=0;
    for(int i=1;i<=s;i++)
    {
        if(in[i]==0)//对于入度为 0 的强连通分量我们取花费最小的即可。
        {
            LL mini=0x3f3f3f3f;
            for(int j=0;j<G[i].size();j++)mini=min(mini,a[G[i][j]].c);
            ans+=mini;
        }
    }
    cout<<ans<<endl;
}

void init(int n)//初始化相关
{
    s=0;clo=0;
    while(!S.empty())S.pop();
    memset(vis,0,sizeof(vis));
    memset(low,0,sizeof(low));
    memset(df,0,sizeof(df));
    memset(in,0,sizeof(in));
    memset(bel,0,sizeof(bel));
    for(int i=1;i<=n;i++)G[i].clear(),V[i].clear();
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
```

```
                if(i!=j&&judge(i,j)){
                    V[i].pb(j);
                }
}
int main()
{
    int t;
    sca(t);
    int cas=1;
    while(t--)
    {
        int n;
        sca(n);
        read(n);
        init(n);
        printf("Case #%d: ",cas++);
        solve(n);
    }
}
```

## HDU 5468 dfs+容斥原理(计 17-10 郭留阳)

## 二、题号与题目名称

2015 ACM/ICPC Asia Regional Shanghai Online A

## 二、题意简述

给了 n 个点（1e5），n-1 条边形成的树，1 为根节点，每一个点有一个数字，求对于每一个点在它的子树中有多少个数与它互质。

## 三、主要知识点及其运用

本题主要涉及素数分解以及容斥原理的应用

## 四、完整解题思路描述

假设我们掌握了如何求解一个数与一个集合中有多少个数是互质的方法，本题也完成了一大半了。我们在遍历以该点为根的子树之前记录答案，在遍历完该点为根的子树后做差即可。

## 五、技巧与坑点

技巧：假设我们要求 x 与一个集合 s 中有多少个数是互质的。我们需要将 s 中的数质因数分解，分解成 k 个不同的质因数,然后采用二进制枚举判断得到当前数的因子,用一个数组记录当前因子的出现次数。然后将 x 做同样的操作，二进制枚举，进行容斥即可。具体在下面代码注释。

坑点：如果当前点的值为 1，那么它与自己也是互质的，答案需要加 1。

## 六、参考文献与学习资料

https://blog.csdn.net/u014664226/article/details/48878581

## 七、完整代码

```
#include<bits/stdc++.h>
#define LL long long
using namespace std;
#define pb(x) push_back(x)
#define sca(x) scanf("%d",&x)
#define mp(x,y) make_pair(x,y)

const int N = 2e5+5;
const int maxn = 1e5+100;
vector<int>v[N];//存边
vector<pair<int,int> >_fac[N]; //first 存数值， second 存储的是取的个数用于二进制枚举（奇加偶减）。

typedef pair<int,int>pii;

int ans[N],num[N],a[N],sz[N];  //ans 记录答案 num 记录该因子出现的次数 a 存点权 sz 存的是以当前点为根子树的大小
int b[1005],c[1005];

void init( )
{
```

```
    for(int i=2;i<=maxn;i++) //筛出每个数的质因子
    {
        int now=i;
        int tot=0;
        for(int j=2;j*j<=now; )
        {
            if(now%j==0)
            {
                b[tot++]=j;
                while(now%j==0)now/=j;
            }
            if(j==2)j++;
            else j+=2;
        }
        if(now!=1)b[tot++]=now;

        for(int j=1;j<(1<<tot);j++) //枚举子集，即求当前数所有的因子，因为我们统计的是不互质的个数
        {                           //然后通过相减得到互质的个数。所以，每个数只对它的因子有贡
献。
            int fac=1;
            int sig=0;
            for(int k=0;k<tot;k++)
            {
                if((1<<k)&j)
                {
                    fac*=b[k];
                    sig++;
                }
            }
            _fac[i].pb(mp(fac,sig)); //sig 用于奇加偶减
        }
    }
}

int cal(int k,int val)
{
    int tot=_fac[k].size();
    int res=0;
    for(int i=0;i<tot;i++) //二进制枚举+容斥
    {
        pii tmp=_fac[k][i];
        if(tmp.second%2) res+=num[tmp.first];
        else res-=num[tmp.first];
        num[tmp.first]+=val;
    }
```

```
    return res;
}

void dfs(int u,int fa)
{
    int pre=cal(a[u],0); //我们在进入子树之前，分解 a[u]，记录有多少个数与 a[u]不互质
    sz[u]=1;
    for(int i=0;i<v[u].size();i++)
    {
        int to=v[u][i];
        if(to==fa)continue;
        dfs(to,u);
        sz[u]+=sz[to];
    }
    int now=cal(a[u],1); //当前子树已经遍历完了，我们重新统计有多少个数与 a[u]不互质，并且将 a[u]所能提供的因子插入到数组中。
    ans[u]=sz[u]-1-(now-pre);//子树的大小减去本身再减去不互质的个数即为互质的个数。
    if(a[u]==1)ans[u]++;//a[u]为 1 时与自己也互质。
}

int main()
{
    int cas=1;
    int n;init();
    while(~scanf("%d",&n))
    {
        memset(num,0,sizeof(num));
        memset(ans,0,sizeof(ans));
        for(int i=1;i<=n;i++)v[i].clear();
        for(int i=1;i<n;i++)
        {
            int x,y;
            scanf("%d%d",&x,&y);
            v[x].pb(y);
            v[y].pb(x);
        }
        for(int i=1;i<=n;i++)scanf("%d",&a[i]);
        printf("Case #%d: ",cas++);
        dfs(1,1);
        for(int i=1;i<=n;i++)printf("%d%c",ans[i],i==n?'\n':' ');
    }
}
```

# HDU 4742 cdq 分治 (计 17-10 郭留阳)

## 一、题号与题目名称

2013 ACM/ICPC Asia Regional Hangzhou Online Pinball Game 3D

## 二、题意简述

给出了 n 个点（x,y,z），n（1e5）个点之间没有顺序要求。求用这些点组成的最长三维递增序列是多长，并且求组成最长的方案数。

## 三、主要知识点及其运用

Cdq 分治以及树状数组维护区间最大值

## 四、完整解题思路描述

普通的 n^2 的 dp 肯定不行，考虑到如何优化。设 pair dp[N]，为以当前点为结尾长度为 first,方案数为 second。然后用 cdq 分治优化 dp，时间复杂度 nlognlogn。

## 五、技巧与坑点

树状数组维护区间最大值，以及 dp 的转移。

## 六、参考文献与学习资料

https://blog.csdn.net/wu_tongtong/article/details/78785836

## 七、完整代码

```
#include<bits/stdc++.h>
#define LL long long
#define sca(x) scanf("%d",&x)
#define fir first
#define sec second
#define lowb(i) (i&(-i))
#define mp(x,y) make_pair(x,y)
#define pb(x) push_back(x)

using namespace std;
const int N = 1e5+5;
const int mod    = 1e9+7;

typedef pair<int,int>pii;
struct node
{
    int x,y,z;
    friend bool operator <(node a,node b)
    {
        if(a.x!=b.x) return a.x<b.x;
        if(a.y!=b.y) return a.y<b.y;
        return a.z<b.z;
    }
```

```
    pii ans;

}a[N],b[N];
int n;

bool cmp(node a,node b)
{
    return a.y<=b.y;
}

bool cmp1(node a,node b)
{
    return a.x<=b.x;
}

void modify(pii a,pii &b)
{
    if(b.first<a.first)b=a;
    else if(b.first==a.first)b.second+=a.second;
}

struct BIT
{
    pii c[N];
    void init( )
    {
        for(int i=1;i<N;i++)c[i]=mp(0,0);
    }
    void upd(int x,pii ans)
    {
        for(int i=x;i<N;i+=lowb(i)) modify(ans,c[i]);
    }

    pii ask(int x)
    {
        pii ans;
        ans.first=0,ans.second=0;
        for(int i=x;i;i-=lowb(i))modify(c[i],ans);
        return ans;
    }
    void clr(int x)
    {
        while(x<N)
        {
            c[x]=mp(0,0);
```

```
            x+=lowb(x);
        }
    }
}bit;

void cdq(int l,int r)
{
    if(l>=r) return ;
    int m=(l+r)>>1;
    cdq(l,m);
    sort(a+l,a+m+1,cmp );//左边的 y 有序
    sort(a+1+m,a+r+1,cmp );//右边的 y 有序
    int i,j;
    for(i=l,j=m+1;j<=r;j++)//由于 x 已经小于，我们只需要在归并 y 的时候
    {                                    //在树状数组 z 的位置插入当前的 dp 值即可
        while(a[i].y<=a[j].y &&i<=m)
        {
            bit.upd(a[i].z,a[i].ans);
            i++;
        }
        pii tmp=bit.ask(a[j].z);//对于右区间的每一个查询能够保证小于它的点已经插入了
        tmp.fir++;
        modify(tmp,a[j].ans);
    }
    for(int i=l;i<=m;i++) bit.clr(a[i].z);//树状数组每次用完后要清
    sort(a+m+1,a+1+r);//由于刚才的重拍打乱了 x，现在要还原回去。
    cdq(m+1,r);
}
vector<int>v;
int main()
{
    int t;
    sca(t);
    while(t--)
    {
        sca(n);
        bit.init();
        v.clear();
        for(int i=1;i<=n;i++)
        {
            scanf("%d%d%d",&a[i].x,&a[i].y,&a[i].z);
            a[i].ans.first=1;
            a[i].ans.second=1;
            v.pb(a[i].z);
        }
```

```
        sort(v.begin(),v.end());
        v.erase(unique(v.begin(),v.end()),v.end());
        for(int i=1;i<=n;i++)a[i].z=lower_bound(v.begin(),v.end(),a[i].z)-v.begin()+1;
        sort(a+1,a+1+n);
        cdq(1,n);
        pii ans=mp(0,0);
        for(int i=1;i<=n;i++) modify(a[i].ans,ans);
        printf("%d %d\n",ans.first,ans.second%mod);
    }
}
```

# Codeforce 600 E 树上启发式合并 (计 17-10 郭留阳)

## 三、题号与题目名称

Codeforce 600 E Lomsat gelral

## 二、题意简述

给出一颗有根树，n（1e5）个节点，每一个节点有一个颜色，要求统计任意一个子树中出现颜色最多的颜色之和。

## 三、主要知识点及其运用

树上启发式合并。

## 四、完整解题思路描述

预处理出每个点的重儿子，对于每一个点，我们先处理其轻儿子，然后消除影响，然后处理其重儿子，不消除影响，然后再将父节点和轻儿子与重儿子的答案合并。

## 五、技巧与坑点

技巧：树上启发式合并。

坑点：注意同一个子树中可能有多个出现次数最多的颜色。

## 六、参考文献与学习资料

http://codeforces.com/blog/entry/44351

## 七、完整代码

```
#include<bits/stdc++.h>
using namespace std;
#define LL long long
#define sca(x) scanf("%d",&x)
#define pb(x) push_back(x)

const int N = 1e5+5;

vector<int>V[N];
int son[N],sz[N],Son;
int a[N];
void dfs_1(int u,int fa)//找出重儿子
{
    int mx=-1;
    sz[u]=1;
    for(auto v:V[u]){
        if(v!=fa){
            dfs_1(v,u);
            sz[u]+=sz[v];
            if(sz[v]>mx)mx=sz[v],son[u]=v;
        }
    }
```

```
}

int cnt[N];
LL sum,Max;
LL ou[N];

void cal(int u,int fa,int val)
{
	cnt[a[u]]+=val;
	if(cnt[a[u]]==Max)sum+=a[u];
	if(cnt[a[u]]>Max)
	{
		Max=cnt[a[u]];
		sum=a[u];
	}
	for(auto i:V[u]){
		if(i==fa||i==Son)continue;
		cal(i,u,val);
	}
}
int n;
void dfs(int u,int fa,int f)
{
	for(auto i:V[u]){   //f=1,代表当前是轻儿子要消除影响，否则不消除
		if(i==fa||i==son[u])continue;//先处理轻儿子最后处理重儿子
		dfs(i,u,1);                          //就可以不用清空重儿子的影响。
	}
	if(son[u])dfs(son[u],u,0),Son=son[u];//处理重儿子。
	cal(u,fa,1);
	ou[u]=sum;                                   //记录答案
	if(son[u])Son=0;
	if(f)cal(u,fa,-1),sum=0,Max=0;              //如果当前是轻儿子，消除影响。
}

int main()
{
	sca(n);
	for(int i=1;i<=n;i++)sca(a[i]);
	for(int i=0;i<n-1;i++)
	{
		int u,v;
		sca(u),sca(v);
		V[u].pb(v);
		V[v].pb(u);
	}
```

```
    dfs_1(1,0);
    dfs(1,0,0);
    for(int i=1;i<=n;i++)printf("%lld%c",ou[i],i==n?'\n':' ');
}
```

# ACM-ICPC 2018 沈阳网络赛 C Convex Hull (计 17-9 蒲巍)

## 六、题号与题目名称

ACM-ICPC 2018 沈阳网络赛 C

## 七、题意简述

- 题中定义了一个函数 gay

$$gay(i) = \begin{cases} 0, & if \quad i = k*x*x, x > 1, k \geq 1 \\ i*i, & else \end{cases}$$

，

- 要求计算

$$\sum_{num=1}^{n} (\sum_{i=1}^{num} gay(i)) \mod p$$

- 多组输入，对于每组输入给一个 n 和一个 p

$$1 \leq n \leq 10^{10}, 1 \leq p \leq 10^{11}.$$

## 三、主要知识点及其运用

本题涉及到了莫比乌斯函数的性质。

## 四、完整解题思路描述

- 显然当 n 存在平方及高次方因子时， ( μ (n))^2 =0
- 其他情况下( μ (n))^2 = 1
- 通过这个式子我们就能很好的解决筛选出非平方及高次方因子
- 接下来我们运用这个变一下型
- $$\sum_{num=1}^{n} \sum_{i=1}^{num} gay(i) = \sum_{num=1}^{n} \sum_{i=1}^{num} \mu^2(i) i^2$$
- 我们应该考虑优化这个 $\sum_{num=1}^{n} \sum_{i=1}^{num}$ 式子
-
- 我们可以很容易的发现大概是这样的
- 1
- 1 2
- 1 2 3
- 1 2 3 4
- 1 2 3 4 5
- 所以这个式子可以替换成

$$= \sum_{i=1}^{n} \mu^2(i) i^2 (n - i + 1)$$
$$= (n+1) \sum_{i=1}^{n} \mu^2(i) i^2 - \sum_{i=1}^{n} \mu^2(i) i^3$$

-

$$\sum_{i=1}^{n}\mu^2(i)i^2$$

- 对于 这个式子，我们考虑找出所有的含平方及以上因子的数，做一个莫比乌斯容斥
- 做一个简单的推演，我们不难发现其规律

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 4 | | | | 8 | | | | 12 | | | | 16 | | | | 20 | | | | 24 | | | | 28 | | | | 32 | | | | 36 | -1 |
| | | | | | | | | 9 | | | | | | | | | 18 | | | | | | | | | 27 | | | | | | | | 36 | -1 |
| | | | | | | | | | | | | | | | 16 | | | | | | | | | | | | | | | | 32 | | | | 0 |
| | | | | | | | | | | | | | | | | | | | | | | | | 25 | | | | | | | | | | | -1 |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 36 | 1 |

- 我们可以再次化简这个式子

$$\sum_{i=1}^{n}\mu^2(i)i^2$$
$$=\sum_{i=1}^{\sqrt{n}}\mu(i)\sum_{k=1}^{\lfloor\frac{n}{i}\rfloor}(ki^2)^2$$
$$=\sum_{i=1}^{\sqrt{n}}\mu(i)i^4\sum_{k=1}^{\lfloor\frac{n}{i}\rfloor}k^2$$

- 式子左侧的代表所有的 n 所有的能产生平方的数
- 中间的莫比乌斯函数作为容斥因子
- 右侧则表示 所有的 （小于 n 的 含平方的） 数 的平方
- 所以 最后化简成

$$(n+1)\sum_{i=1}^{\sqrt{n}}\mu(i)i^4\sum_{k=1}^{\lfloor\frac{n}{i^2}\rfloor}k^2-\sum_{i=1}^{\sqrt{n}}\mu(i)i^6\sum_{k=1}^{\lfloor\frac{n}{i^2}\rfloor}k^3$$

- 用前 n^2、n^3 的前 n 项和的公式做优化
- 然后 sqrt(n)直接求解

## 五、技巧与坑点

技巧：用前 n 项（n^2、n^3）和做优化。

坑点：然后还会爆 longlong，需要开 int128，用快速乘法会 t

## 六、参考文献与学习资料

https://blog.csdn.net/qkoqhh/article/details/82532516#commentsedit

## 七、完整代码

```
#include<bits/stdc++.h>
#define LL long long
using namespace std;

const LL N=1e5+10;
__int128 f[N];
__int128 s[N];
bool nt[N];
LL mod;
```

```
int cnt = 0;
int p[N];
LL mob[N];



void prime(){
	nt[0] = nt[1] = mob[1] = 1;
	for(int i = 2;i<N;i++){
		if(!nt[i]){
			p[cnt ++] = i;
			mob[i] = -1;
		}
		for(int j = 0;j<cnt ;j++){
			if(1LL * p[j] * i > N) break;
			nt[p[j] * i] = 1;
			mob[p[j] * i] = i % p[j] == 0? 0: -mob[i];
			if(i % p[j] == 0) break;
		}
	}
}

LL square(LL n){
	__int128 x=n,y=n+1,z=2*n+1;
	if(x%2==0) x/=2;
	else if(y%2==0) y/=2;
	else z/=2;

	if(x%3==0) x/=3;
	else if(y%3==0) y/=3;
	else z/=3;
	return (x%mod * y%mod * z%mod);
}

LL cube(LL n){
	__int128 x=n,y=n+1;
	if(x%2==0) x/=2;
	else y/=2;
	return x%mod * x%mod * y%mod * y%mod;
}

int main(){
	LL n;
	prime();
	while(~scanf("%lld%lld",&n,&mod)){
```

```
        for(LL i = 0;i*i<=n;i ++){
            f[i] = i%mod * i%mod * i%mod * i%mod;
            s[i] = i%mod * i%mod * i%mod * i%mod * i%mod * i%mod;
        }
        LL ans=0;
        for(LL i=1;i*i<=n;i++){
            ans = (ans + mod + (n+1)%mod * f[i]%mod * square(n/(i*i))%mod * mob[i] - s[i]%mod * cube(n/(i*i))%mod * mob[i])%mod;
        }
        printf("%lld\n",ans);
    }
}
```

# ACM-ICPC 2017 亚洲区（西安赛区）网络赛 F Trig Function (计 17-9 蒲巍)

## 四、题号与题目名称

ACM-ICPC 2017 亚洲区（西安赛区）网络赛 F Trig Function

## 二、题意简述

- 对于任意的 x，满足 f(cos(x)) = cos(n * x)
- 给出两个整数 n ， m
- 计算 x^m 的系数 并 mod 998244353
- 多组输入，其中 1<=n<=1e9 ,0<=m<=1e4。

## 三、主要知识点及其运用

切比雪夫多项式

## 四、完整解题思路描述

根据切比雪夫多项式第一形式可知。

**定理** 对任何正整数 $n$，必有

$$\cos n\theta = \sum_{k=0}^{\left[\frac{n}{2}\right]} (C_{n-k}^{k} + C_{n-1-k}^{k-1})(-1)^k 2^{n-1-2k} \cos^{n-2k}\theta ,$$

$$\sin n\theta = \sum_{k=0}^{\left[\frac{n-1}{2}\right]} C_{n-1-k}^{k} (-1)^k 2^{n-1-2k} \cos^{n-1-2k}\theta \sin\theta .$$

。

## 五、技巧与坑点

技巧：总结失误，选好切题方向，认真积累基础知识。

坑点：如果用算出前几项找规律求解的话，很难找到正解。

## 六、参考文献与学习资料

https://www.cnblogs.com/ECJTUACM-873284962/p/7562111.html

## 七、完整代码

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn=1e5+5;
const int mod=998244353;
ll fac[maxn]={1};
ll n,m;
void init(){
    for(int i=1;i<maxn;i++)
        fac[i] = fac[i-1]*i%mod;
}
ll qmod(ll x,int q){
    ll res=1;
    while(q){
```

```
        if(q%2) res = res*x%mod;
        x = x*x%mod;
        q /= 2;
    }
    return res;
}
int main(){
    init();
    while(~scanf("%lld%lld",&n,&m)){
        if(m>n) puts("0");
        else if(n%2 && m%2 == 0) puts("0");
        else if(n%2==0 && m%2) puts("0");
        else{
            ll fz = n % mod;
            if(m >= 1){
                for(int i = n-m+1;i <= n+m-1;i++){
                    if(i%2 == (n+m-2)%2){
                        fz=fz * i % mod;
                    }
                }
                ll tmp = fz*qmod(fac[m],mod-2)%mod;
                if((n-m)/2%2)
                    tmp = -tmp;
                printf("%lld\n",(tmp+mod)%mod);
            }
            else{
                ll t=1;
                for(int i=n+m-1;i <= n-m;i++){
                    if(i%2 == (n+m-2)%2)
                        t = t*i%mod;
                }
                ll tmp = fz*qmod(fac[m],mod-2)%mod*qmod(t, mod-2)%mod;
                if((n-m)/2%2) tmp = -tmp;
                printf("%lld\n",(tmp+mod)%mod);
            }
        }
    }
}
```

# ACM-ICPC 2018 南京赛区网络预赛 F An Easy Problem On The Trees (计 17-9 蒲巍)

## 一、题号与题目名称

ACM-ICPC 2018 南京赛区网络预赛 F An Easy Problem On The Trees

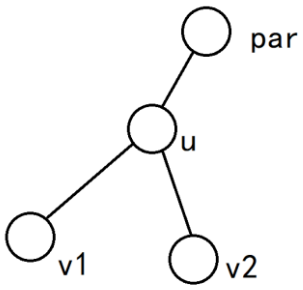
## 二、题意简述

- 给你 n 个节点、n-1 个边的一颗树，m 个操作，操作有三种：
  1. 连接 a 与 b。
  2. 以 x 为根节点，将 b 与其父节点连接的边删除。
  3. 询问从 x 开始最后走回 x 的期望步数（每条边等概率移动）

## 三、主要知识点及其运用

求解数学期望，动态树来维护信息。

## 八、 完整解题思路描述




- 考虑叶节点 u，则 E(u)=E(par)+1 。对于倒数第二层节点 u，则 E(u)=E(par)+2d(u)−1 （d(u)是 u 的度数）
- 对于叶子结点相信大家都能明白 只有一条路能走
- 对于倒数第二层节点我们来做一个证明
- E(u) = E(v1)/3 + E(v2)/3 + (E(par)+1)/3
- 并且
- E(v1) = E(v2) = E(u) + 1
- 化简一下
- E(u)=E(par)+2*3 − 1 （在本图中 d（u） = 3）
- 另外在画几个图就能推算出 E(u)=E(par)+2d(u)−1

于是我们大胆假设，对于任意 f(u) ，存在 g(u) 使 f(u)=f(par)+g(u) f(u)=f(par)+g(u) 。那么 g(u) 到底是什么呢？最后两层的规律告诉我们，g(u)=2*s(u)−1，其中 s(u)表示以 u 为根的子树大小。通过数学归纳法可以证明这个结论。于是我们考虑答案是什么，由于第一步会任意走到根节点的一个子节点当中且不可能跳转到其它兄弟节点上，于是

- $$ans = \left( \sum_{(root,v)\in E} f(v)/d(root) \right) + 1 = \frac{2s(root)-2}{d(root)}$$

数据结构使用动态树来维护。

## 五、技巧与坑点

坑点：关于期望公式的推导，值的维护。

# 六、参考文献与学习资料

https://www.cnblogs.com/flashhu/p/8324551.html

# 七、完整代码

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;

const int N = 1e5 + 5;
const int mod = 998244353;

int f[N];
int c[N][2];
int sz[N];
int szxv[N];
int deg[N];
bool flag[N];
ll qpow(ll a,ll b){
    ll ans = 1;
    while(b){
        if(b & 1) ans = (ans * a) % mod;
        a   = (a * a) % mod;
        b >>= 1;
    }
    return ans ;
}
bool isroot(int x){///它不是它父亲的偏爱儿子，所有它是根
     return c[f[x]][0] != x && c[f[x]][1] != x;
}

void push_up(int x){
    sz[x] = szxv[x] + sz[c[x][0]] + sz[c[x][1]] + 1;
}
void push_down(int x){
    if(!x) return ; ///如果 x 是 0  直接退出
    if(flag[x]){    ///如果有延时标记
        swap(c[x][1],c[x][0]);
        if(c[x][1]) flag[c[x][1]] ^= 1;
        if(c[x][0]) flag[c[x][0]] ^= 1;
        flag[x] = 0 ;
    }
}
void rotate(int x,int k){    ///k == 0  左孩子   k == 0  右孩子
    int y = f[x]; ///y 是 x 的父亲
    int z = f[y]; ///z 是 y 的父亲  x 的爷爷
```

```
    int tmp = c[x][k]; ///tmp  是 x 的偏爱孩子
    if(!isroot(y)) ///如果 y 不是根  y 是 z 的偏爱孩子
        c[z][c[z][1] == y] = x;   ///原来链接 x 父亲 y 的边   链接到 x 上
    f[x] = z;       ///记录 x 的父亲为 y
    c[x][k] = y; ///x 的偏爱孩子  换成 y
    f[y] = x;       ///更新 y 的父亲
    c[y][!k] = tmp;   ///和 x  的偏爱孩子  相反的那个孩子  接到  y 上
    if(tmp) f[tmp] = y;   ///如果它存在   跟新它的父亲

    push_up(y);     ///更新所维护的值
    push_up(x);     ///同
}
void splay(int x){
    push_down(x);
    while(!isroot(x)){     ///当 x 是偏爱孩子的时候
        int y = f[x];       ///y 是 x  的父亲
        int z = f[y];       ///z 的 y  的父亲  x  的爷爷
        push_down(z);
        push_down(y);
        push_down(x);
        int k    = (c[y][0] == x); ///k 为  y 给  x 的兄弟节点的编号

        if(isroot(y)){                  ///如果 y 是根  即 y 不是偏爱孩子
            rotate(x,k);                ///做一次上移操作  将 x 原来的父亲 y  放到 x 的兄弟的那侧  将 x 的兄弟
接到 y 上  方向与原来的相反
            break;
        }
        if(c[z][!k] == y) rotate(y,k),rotate(x,k); ///旋转
        else rotate(x,k),rotate(x,!k);     ///旋转
    }
}
void access(int x){
    int u = x;
    int v = 0;
    while(u){
        splay(u);
        szxv[u] += sz[c[u][1]] - sz[v];
        c[u][1] = v;
        push_up(u);
        v = u;
        u = f[u];
    }
}

int findroot(int x){
```

```
    access(x);
    splay(x);
    push_down(x);
    while(c[x][0]) {x = c[x][0]; push_down(x);}
    return x;
}

void makeroot(int x){
    access(x);
    splay(x);
    flag[x] ^= 1;
}
void link(int x,int y){
    makeroot(x);
    //splay(x);
    f[x] = y;
    szxv[y] += sz[x];
}
void cut(int x,int y){
    makeroot(x); ///x 成为根
    access(y);     ///访问 y
    splay(y);      ///y 变成根
    deg[y] --;     ///y 的度数减少一个
    int p = c[y][0]; ///p 是 y 的左儿子
    do{
        push_down(p); ///递推 lazy 标记
        if(c[p][1]) p = c[p][1]; ///如果 p 有右儿子就是他的右儿子
        else break;
    }while(1);
    deg[p] --;
    f[c[y][0]] = 0;
    c[y][0] = 0;
    push_up(y);
}
int main(){
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i = 1;i<=n;i++) sz[i] = 1;
    for(int i = 1;i<n;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        if(findroot(u) != findroot(v)){
            link(u,v);
            deg[u] ++;
            deg[v] ++;
```

```
        }
    }
    int t,x,y;
    while(m --){
        int op,x,y;
        scanf("%d",&op);
        if(op == 1){
            scanf("%d%d",&x,&y);
            if(findroot(x) != findroot(y) && x != y){
                link(x,y);
                deg[x] ++;
                deg[y] ++;
            }
            else puts("-1");
        }
        else if(op == 2){
            scanf("%d%d",&x,&y);
            if(findroot(x) == findroot(y) && x != y){

                cut(x,y);
            }
            else puts("-1");
        }
        else{
            scanf("%d",&x);
            makeroot(x);
            ll res = 2ll * (sz[x] -1) * qpow(deg[x],mod-2)%mod;
            cout<<sz[x]<<" "<<deg[x]<<endl
            printf("%lld\n",res);
        }

    }
}
```