

ICPC TEMPLATE

AC mountain(ac 山岳险天下)

2019 年 11 月 6 日

目录

1	一切的开始	3
1.1	宏定义	3
1.2	快速读	4
1.3	对拍	5
2	数据结构	6
2.1	BIT	6
3	图论	7
3.1	最短路	7
3.1.1	floyd	7
3.1.2	Bellman-ford	8
3.1.3	SPFA	8
3.1.4	folyd	9
3.1.5	Dijkstra	9
3.1.6	差分约束	10
3.2	次短路	10
3.3	图的连通性问题	12
3.3.1	维护连通性	12
3.3.2	桥	12
3.4	欧拉回路问题	15
3.4.1	圈套圈算法	15
3.4.2	二进制	16
3.5	树	16
3.5.1	最小生成树	16
3.6	Prufer 序列	17
3.7	二分图	20
3.7.1	二分图匹配	20
3.8	三元环计数	21
3.9	网络流	21

4 数学	27
4.1 BSGS	27
4.2 扩展中国剩余定理	27
4.2.1 值在 <code>__int128</code> 以内	27
4.2.2 值在 <code>__int128</code> 以外	28
4.3 扩展卢卡斯	29
4.4 fft	30
4.5 fwt	32
4.6 polya	33
4.7 大素数	33
4.8 线性基	36
4.9 洲阁筛	38
4.10 计算素数 k 次幂前缀和	38
4.11 自然幂数和	39
4.12 组合数打表	41
4.13 杜教筛	41
4.14 二次剩余	42
4.14.1 解 $x^2 = a(\text{mod } p)$	42
4.15 高精度	44
4.16 反素数	47
4.16.1 求小于 n 并且因子个数最多的那个数	47
4.17 高斯消元	48
4.18 高斯消元解异或方程	49
4.19 矩阵类（快速幂）	50
4.20 类欧几里得	51
4.21 牛顿迭代	52
4.22 母函数	52
4.22.1 莫比乌斯反演	52
4.23 斐波那契广义循环节	53
4.24 无名小定理	56
4.25 自然常数	57
4.26 欧拉常数	57
4.27 错排公式	57
4.28 伯努利数	57
4.29 自然幂数和	57
4.30 Catalan	57
4.31 pick 定理	57
4.32 正多边形外接圆半径	57
4.33 Fibonacci	58
5 计算几何	58
5.1 处理平面内所有直线围成的所有多边形	58

1	一切的开始	3
6	字符串	64
6.1	kmp	64
6.2	SA	64
6.3	回文树 1	66
6.4	回文树 2	67
7	杂项	68
7.1	退火	68
7.2	博弈	69
7.2.1	Bash	69
7.2.2	Wythoff	69
7.2.3	Fibonacci's Game	69
7.2.4	staircase nim	70
7.2.5	anti-nim	70
7.2.6	约数博弈	70
7.2.7	约数和倍数博弈	70
7.2.8	Chomp 博弈	70
7.2.9	树上删边游戏	70
7.2.10	SG	71
7.3	indiewar 的私人题	73
7.3.1	半圆概率	73
7.3.2	百囚徒挑战	74
7.3.3	投针问题	74
7.3.4	打怪兽	74
7.3.5	互素	74
7.3.6	半球	74
8	DP	74
8.1	背包	74
8.1.1	01 背包	74
8.1.2	完全背包	76
8.1.3	多组背包	77
8.1.4	分组背包	78
8.1.5	树形依赖背包	78

1 一切的开始

1.1 宏定义

by 杜教

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define rep(i,a,n) for (int i=a;i<n;i++)//注意范围[a.n)

```

```

4  #define per(i,a,n) for (int i=n-1;i>=a;i--)//注意范围[a,n-1]
5  #define pb push_back
6  #define mp make_pair
7  #define all(x) (x).begin(),(x).end()
8  #define fi first
9  #define se second
10 #define SZ(x) ((int)(x).size())
11 typedef vector<int> VI;
12 typedef long long ll;
13 typedef pair<int,int> PII;
14 mt19937 rand(random_device{}());
15 const ll mod=1000000007;
16 int rnd(int x) { return rand() % x;}
17 ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0);
    for(;b>=1){if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
18 ll gcd(ll a,ll b) { return b?gcd(b,a%b):a;}
19 // -----

```

- CMakeLists.txt (for CLion)

```

1 set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O2 -Dzerol -Wall")

```

- HDU Assert Patch

```

1 #ifdef ONLINE_JUDGE
2 #define assert(condition) if (!(condition)) { int x = 1, y = 0; cout << x / y << endl; }
3 #endif

```

1.2 快速读

```

1 inline char nc() {
2     static char buf[100000], *p1 = buf, *p2 = buf;
3     return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2) ? EOF : *p1++;
4 }
5 template <typename T>
6 bool rn(T& v) {
7     static char ch;
8     while (ch != EOF && !isdigit(ch)) ch = nc();
9     if (ch == EOF) return false;
10    for (v = 0; isdigit(ch); ch = nc())
11        v = v * 10 + ch - '0';
12    return true;
13 }
14

```

```

15 template <typename T>
16 void o(T p) {
17     static int stk[70], tp;
18     if (p == 0) { putchar('0'); return; }
19     if (p < 0) { p = -p; putchar('-'); }
20     while (p) stk[++tp] = p % 10, p /= 10;
21     while (tp) putchar(stk[tp--] + '0');
22 }

```

- 需要初始化
 - 需要一次读入
 - 不支持负数
-

```

1  const int MAXS = 100 * 1024 * 1024;
2  char buf[MAXS];
3  template<typename T>
4  inline bool read(T& x) {
5      static char* p = buf;
6      x = 0;
7      while (*p && !isdigit(*p)) ++p;
8      if (!*p) return false;
9      while (isdigit(*p)) x = x * 10 + *p++ - 48;
10     return true;
11 }
12
13 fread(buf, 1, MAXS, stdin);

```

1.3 对拍

```

1  #!/usr/bin/env bash
2  g++ -o r main.cpp -O2 -std=c++11
3  g++ -o std std.cpp -O2 -std=c++11
4  while true; do
5      python gen.py > in
6      ./std < in > stdout
7      ./r < in > out
8      if test $? -ne 0; then
9          exit 0
10     fi
11     if diff stdout out; then
12         printf "AC\n"
13     else
14         printf "GG\n"
15         exit 0

```

```

16     fi
17 done

```

- 快速编译运行

```

1  #!/bin/bash
2  g++ $1.cpp -o $1 -O2 -std=c++14 -Wall -Dzerol -g
3  if $? -eq 0; then
4      ./$1
5  fi

```

2 数据结构

2.1 BIT

```

1  struct Bit
2  {
3      vector<int> a;
4      int sz;
5      void init(int n)
6      {
7          sz=n+5;
8          for(int i=1;i<=n+5;i++)
9              a.push_back(0);
10     }
11     int lowbit(int x)
12     {
13         return x&(-x);
14     }
15     int query(int x)
16     {
17         int ans = 0;
18         for(;x;x-=lowbit(x))ans+=a[x];
19         return ans;
20     }
21     void update(int x,int v)
22     {
23         for(;x<sz;x+=lowbit(x))
24             a[x]+=v;
25     }
26 }bit;

```

3 图论

3.1 最短路

3.1.1 floyd

```

1 for (int k = 1; k <= n; k++) {
2     for (int i = 1; i <= n; i++) {
3         for (int j = 1; j <= n; j++) {
4             f[i][j] = min(f[i][j], f[i][k] + f[k][j]);
5         }
6     }
7 }

```

- 找最小环

```

1 int val[maxn + 1][maxn + 1]; // 原图的邻接矩阵
2 int floyd(const int &n) {
3     static int dis[maxn + 1][maxn + 1]; // 最短路矩阵
4     for (int i = 1; i <= n; ++i)
5         for (int j = 1; j <= n; ++j) dis[i][j] = val[i][j]; // 初始化最短路矩阵
6     int ans = inf;
7     for (int k = 1; k <= n; ++k) {
8         for (int i = 1; i < k; ++i)
9             for (int j = 1; j < i; ++j)
10                ans = std::min(ans, dis[i][j] + val[i][k] + val[k][j]); // 更新答案
11        for (int i = 1; i <= n; ++i)
12            for (int j = 1; j <= n; ++j)
13                dis[i][j] = std::min(
14                    dis[i][j], dis[i][k] + dis[k][j]); // 正常的 floyd 更新最短路矩阵
15    }
16    return ans;
17 }

```

- 利用 floyd 的 dp 思路求解

```

1 int dp[maxn][maxn][maxn];
2 int w[maxn];
3 int s[maxn];
4 bool cmp(int a, int b)
5 {
6     return w[a] < w[b];
7 }
8 rep(i, 1, n+1)
9 {

```

```

10     rep(j,1,n+1)
11     {
12         scanf("%d",&dp[i][j][0]);
13         rep(k,1,n+1)
14         {
15             dp[i][j][k] = 1e9;
16         }
17     }
18     s[i] = i;
19 }
20 sort(s+1,s+n+1,cmp);
21 rep(k,1,n+1)
22 {
23     rep(i,1,n+1)
24     {
25         rep(j,1,n+1)
26         {
27             dp[i][j][k] = min(dp[i][j][k-1],dp[i][s[k]][k-1]+dp[s[k]][j][k-1]);
28         }
29     }
30 }

```

- 传递闭包已知一个有向图中任意两点之间是否有连边，要求判断任意两点是否连通。

```

1  for (int k = 1; k <= n; k++)
2      for (int i = 1; i <= n; i++)
3          if (f[i][k]) f[i] = f[i] & f[k];

```

3.1.2 Bellman-ford

每次拿每条边去更新最短路（负权边）

```

1  for(int k = 1 ; k <= n - 1 ; k ++ )
2  {
3      for(int i = 1 ; i < m ; i ++ )
4      {
5          if(dis[v[i]] > dis[u[i]] + w[i])
6              dis[v[i]] = dis[u[i]] + w[i] ;
7      }
8  }

```

3.1.3 SPFA

一条边 (u,v)，在 dis[u] 没变的情况下不应该重新更新用队列记录要进行扩展的点，每次 dis[x] 被更新后就把 x 扔进队列里去更新其他点的最短路

```

1 void spfa(int s){
2     for(int i=0; i<=n; i++)
3         dis[i]=99999999; //初始化每点i到s的距离
4     dis[s]=0; vis[s]=1; q[1]=s; 队列初始化,s为起点
5     int i, v, head=0, tail=1;
6     while (head<tail){ 队列非空
7         head++;
8         v=q[head]; 取队首元素
9         vis[v]=0; 释放队首结点,因为这节点可能下次用来松弛其它节点,重新入队
10        for(i=0; i<=n; i++) 对所有顶点
11            if (a[v][i]>0 && dis[i]>dis[v]+a[v][i]){
12                dis[i] = dis[v]+a[v][i]; 修改最短路
13                if (vis[i]==0){ 如果扩展结点i不在队列中,入队
14                    tail++;
15                    q[tail]=i;
16                    vis[i]=1;
17                }
18            }
19        }
20    }
21 }

```

-SPFA 判断负环一个有向图有负环等价于某个点的最短路长度 $>n$ 方法 1: 某个点入队次数 $>n$ 时就有负环
方法 2: 记录 $len[x]$, 表示 1 到 x 的最短路的点的个数, $len[x] \geq n$ 时存在负环时间复杂度: $O(nm)$

3.1.4 folyd

$F(K,X,Y)=\text{Min}(F(K-1,X,Y) , F(K-1,X,K)+F(K-1,K,Y))$ 时间复杂度: $O(n^3)$

```

1 for(int k=1;k<=n;k++)
2     for(int i=1;i<=n;i++)
3         for(int j=1;j<=n;j++)
4             d[i][j]=min(d[i][j],d[i][k]+d[k][j]);

```

3.1.5 Dijkstra

Dijkstra 算法适用于边权为正的无向和有向图, 不适用于有负边权的图根据初始点, 挨个的把离初始点最近的点一个一个找到并加入集合, 集合中所有的点的 $d[i]$ 都是该点到初始点最短路径长度, 由于后加入的点是根据集合 S 中的点为基础拓展的, 所以也能找到最短路径

```

1 void dijkstra(int s) {
2     priority_queue<DistNode> Q;
3     Q.push_back(DistNode{0, s});
4     for(int i = 1; i <= n; i++) d[i] = INF; //把d[i]都设置为最大值
5     d[s] = 0;
6     memset(done, 0, sizeof(done));

```

```

7   while(!Q.empty()) {
8       DistNode x = Q.top(); Q.pop();
9       int u = x.u;
10      if(done[u]) continue; //如果这个点已经被提出过了, 直接抛弃 (适用于那种松弛之后重复放入队列的点)
11      done[u] = true;
12      for(int i = 0; i < Q[u].size(); i++)
13          Edge& e = edges(Q[u][i]);
14          if(e.dist + d[u] < d[e.to] && d[u] < INF) {
15              d[e.to] = e.dist + d[u];
16              p[e.to] = G[u][i];
17              Q.push_back(Edge(d[e.to], e.to)); //把松弛过后点的d值重新放入队列
18          }
19      }
20  }
21  }

```

3.1.6 差分约束

假设对于一张图来说, 1 号点到 x 的最短路长度为 $d[x]$ 则有: $d[y] \leq d[x] + w[x][y]$ 最短路问题可以给出这一类不等式的最大解, 取个反可以得到最小解 - 例题给定 n, m 和 m 个三元组 (l, r, k) , 求一个 01 串 s , 使得 $s[l...r]$ 中 1 的个数至少有 k 个, 求 1 最少的满足条件的 01 串 $n, m \leq 10000$ 设 $f[i]$ 为 $s[1...i]$ 中 1 的个数 $f[i] \leq f[i-1] + 1$; $f[i-1] \leq f[i]$; $f[l-1] \leq f[r] - k$ 等价于求该不等式组中 $f[n]$ 的最小解

3.2 次短路

给定一张有向带正权拓扑图, 求有几条 1 到 N 的路径的长度 \leq 1 到 N 的最短路 $+ K$ $N, M \leq 10^5$, $K \leq 100$, 边权 $\leq 10^9$

定义一条路径 $(X...Y)$ 的冗余度为它的长度减去 $X...Y$ 的最短路长度则题目要求的是 $1...N$ 冗余度不超过 K 的路径长度考虑路径 $(1, a, b, N)$ 冗余度为 $w(1, a) + w(a, b) + w(b, N) - d(1, N) = [w(1, a) + d(a, N) - d(1, N)] + w(a, b) + w(b, N) - d(a, N)$ 设 $p(x, y) = w(x, y) + d(y, N) - d(x, N)$, 则冗余度变成了 $a...N$ 的冗余度加上 $p(1, a)$ 设 $F(X, L)$ 表示 1 到 X 有几条路径满足 p 的和为 L $F(X, L) = \sum(F(Y, L - P(Y, X)))$ 时间复杂度: $O(m \log n + mk + n)$

```

1   #include <cstdio>
2   #include <queue>
3   #include <vector>
4   #include <cstring>
5   using namespace std;
6   int t, d[100001], f[100001][51], n, m, k, p;
7   bool working[100001][51];
8   struct ac{
9       int to, len;
10  };
11  vector<ac> head[100001];
12  vector<ac> h[100001];
13  int dp(int root, int l){
14      int ans = 0;
15      if (l < 0 || l > k) return 0;

```

```

16     if (working[root][l]) {
17         working[root][l]=false;
18         return -1;
19     }
20     if(f[root][l]!=-1)
21         return f[root][l];
22     working[root][l]=true;
23     for (int i=0;i<h[root].size();i++) {
24         ac e= h[root][i];
25         int val=dp(e.to,d[root]+1-d[e.to]-e.len);
26         if (val!=-1) {
27             working[root][l]=false;
28             return -1;
29         }
30         ans=(ans+val)%p;
31     }
32     working[root][l] = false;
33     if (root==1&&l==0) ans++;
34     f[root][l]=ans;
35     return ans;
36 }
37 int main(){
38     scanf("%d",&t);
39     while(t--){
40         scanf("%d%d%d%d",&n,&m,&k,&p);
41         for(int i=1;i<=n;i++){
42             head[i].clear();
43             h[i].clear();
44         }
45         int a,b,c;
46         for(int i=1;i<=m;i++){
47             scanf("%d%d%d",&a,&b,&c);
48             ac e;
49             e.to=b;
50             e.len=c;
51             head[a].push_back(e);
52             e.to=a;
53             h[b].push_back(e);
54         }
55         memset(d,0x3f,sizeof(d));
56         memset(f,-1,sizeof(f));
57         queue<int>q;
58         q.push(1);
59         d[1]=0;
60         while(!q.empty()){
61             int x=q.front();

```

```

62         q.pop();
63         for(int i=0;i<head[x].size();i++){
64             if(d[head[x][i].to]>d[x]+head[x][i].len){
65                 d[head[x][i].to]=d[x]+head[x][i].len;
66                 q.push(head[x][i].to);
67             }
68         }
69     }
70     int ans=0,flag=1;
71     for(int i=0;i<=k;i++){
72         int val=dp(n,i);
73         if(val!=-1){
74             flag=0;
75             break;
76         }
77         ans=(ans+val)%p;
78     }
79     if(flag)
80         printf("%d\n",ans);
81     else
82         printf("-1\n");
83 }
84 return 0;
85 }

```

3.3 图的连通性问题

3.3.1 维护连通性

对于每个连通块维护一棵有根树， $F(x)$ 表示点 x 的父亲则假设我们要添加一条边 U,V ，首先求出 U,V 所在连通块的有根树树根 X,Y ，然后令 $F(X)=Y$ $\text{GetRoot}(X)$ ：若 $F(X)=0$ 则 $\text{return } X$ ，否则 $\text{return } F(X)=\text{GetRoot}(F(X))$ //并查集思想

3.3.2 桥

对于一个连通无向图，定义一条边 (u,v) 是桥，当且仅当断开这条边后图变得不连通图的大致形状： $O-O$ ，则中间的边就是桥强连通分量：没有桥的连通块

3.3.2.1 Tarjan 算法

对整个图进行 dfs，设 $\text{dfn}[x]$ 表示点 x 是第几个被搜到的 $\text{low}[x]$ 表示： x 通过非返祖边，且至多通过一条非树边能到达的最小 dfn

```

1 void tarjan(int now){
2     dfn[now]=low[now]=++cnt;
3     stack[++t]=now;
4     v[now]=1;

```

```

5   for(int i=f[now];i!=-1;i=e[i].next)
6       if(!dfn[e[i].v]){
7           tarjan(e[i].v);
8           low[now]=min(low[now],low[e[i].v]);
9       }
10      else if(v[e[i].v])
11          low[now]=min(low[now],dfn[e[i].v]);
12  if(dfn[now]==low[now]){
13      int cur;
14      do{
15          cur=stack[t--];
16          v[cur]=false;
17      }while(now!=cur);
18  }
19  }

```

-Tarjan 算法求桥非树边一定不是桥对于树边 (U,V)，它是桥等价于 $low[V] > dfn[U]$ 去掉所有的桥后，剩下的每个连通分量都是强连通分量 -有向图的强连通分量在做 Tarjan 算法时，如果 $tarjan(x)$ 后发现 $dfn[x]=low[x]$ ，则 x 的子树里的剩下的所有点构成一个强连通分量

有 N 个人，给你 M 对整数 (a,b)，表示第 a 个人认为 b 很厉害而这种关系具备传递性，也就是如果 a 认为 b 厉害，且 b 认为 c 厉害，则 a 认为 c 厉害求有多少人被所有人都觉得很厉害 $N,M \leq 10^5$ 等价于求有几个点，使得他们能到达所有点可以发现每个强连通分量中的人都是互相认为对方很厉害的，所以可以把它们看成一个人求出强连通分量并缩点，之后得到一个无环图就能做了

给定一个有向图，N 个点，求：1) 至少要选几个顶点，才能做到从这些顶点出发，可以到达全部顶点 2) 至少要加多少条边，才能使得从任何一个顶点出发，都能到达全部顶点按照套路，Tarjan 算法求 SCC，并缩点建图。那么对于问题 1，新的图中入度为 0 的点数即是答案。对于问题 2，答案为 $\max(\text{入度为 0 的点的点数}, \text{出度为 0 的点的点数})$ ，因为对于每个入度或出度为 0 的点，需要连一条边来解决，那么将出度为 0 的点连向入度为 0 的点是最佳的。

```

1   #include<stdio.h>
2   #include<iostream>
3   #include<string.h>
4   #include<vector>
5   using namespace std;
6   #define maxn 105
7   struct edge
8   {
9       int to,nxt;
10  }e[maxn*maxn];
11  int head[maxn],low[maxn],dfn[maxn],sta[maxn],belong[maxn],vis[maxn],num[maxn],rd[maxn],cd[maxn];
12  int en,top,scc,n,m,index;
13  void addedge(int u,int v)
14  {
15      e[en].to=v;
16      e[en].nxt=head[u];
17      head[u]=en++;

```

```
18 }
19 void tarjan(int x)
20 {
21     int v;
22     dfn[x]=low[x]=++index;
23     sta[top++]=x;
24     vis[x]=1;
25     for(int i=head[x];i!=-1;i=e[i].nxt)
26     {
27         v=e[i].to;
28         if(!dfn[v])
29         {
30             tarjan(v);
31             low[x]=min(low[x],low[v]);
32         }
33         else
34             if(vis[v])
35             {
36                 low[x]=min(low[x],dfn[v]);
37             }
38     }
39     if(low[x]==dfn[x])
40     {
41         scc++;
42         do
43         {
44             v=sta[--top];
45             vis[v]=0;
46             belong[v]=scc;
47             num[scc]++;
48         }while(x!=v);
49     }
50 }
51 int main()
52 {
53     std::ios::sync_with_stdio(false);
54     cin>>n;
55     memset(head,-1,sizeof(head));
56     memset(low,0,sizeof(low));
57     memset(dfn,0,sizeof(dfn));
58     memset(vis,0,sizeof(vis));
59     memset(rd,0,sizeof(rd));
60     memset(cd,0,sizeof(cd));
61     memset(num,0,sizeof(num));
62     top=0;
63     index=0;
```

```

64     scc=0;
65     for(int i=1;i<=n;i++)
66     {
67         while(1)
68         {
69             int v;
70             cin>>v;
71             if(v==0) break;
72             addedge(i,v);
73         }
74     }
75     for(int i=1;i<=n;i++)
76         if(!dfn[i]) tarjan(i);
77     for(int u=1;u<=n;u++)
78         for(int i=head[u];i!=-1;i=e[i].nxt)
79         {
80             int v=e[i].to;
81             if(belong[u]!=belong[v])
82             {
83                 cd[belong[u]]++;
84                 rd[belong[v]]++;
85             }
86         }
87     }
88     int ans1=0,ans2=0;
89     for(int i=1;i<=scc;i++)
90         if(rd[i]==0) ans1++;
91     for(int i=1;i<=scc;i++)
92         if(cd[i]==0) ans2++;
93     if(scc==1) cout<<1<<endl<<0<<endl;
94     else cout<<ans1<<endl<<max(ans1,ans2);
95     return 0;
96 }

```

3.4 欧拉回路问题

给定一张有向图，如何求出一条经过每条边恰好一次的回路必要条件：1. 这张图是个强连通分量 2. 每个点的出度等于入度可以发现这两个条件同时也是充分的

3.4.1 圈套圈算法

任选一个起点，从起点开始 dfs，每条边只能被走一遍，当没有边可以走的时候把 x 压入答案的队列中，最后的答案是反着的欧拉回路

3.4.2 二进制

有 n 个灯泡，灯泡有两种状态：开和关，每次可以操作某个灯泡，使得它状态取反定义一个局面为每个灯泡的状态以及最后一个被操作的灯泡的编号，可以将一个局面看成一个二元组 $(0 \dots 2^N - 1, 0 \dots N - 1)$ 现在你可以选定任何初始局面，求最少几步遍历所有局面 $N \leq 20$ 可以建立 2^N 个点，将每个局面看成边，求欧拉回路

3.5 树

树: N 个点 $N-1$ 条边的连通无向图, 分为有根树和无根树
树的叶子: 度数为 1 的点

3.5.1 最小生成树

3.5.1.1 Kruskal 算法

1) 把图中的所有边按代价从小到大排序 2) 把图中的 n 个顶点看成独立的 n 棵树组成的森林 3) 按权值从小到大选择边, 所选的边连接的两个顶点 u_i, v_i , 应属于两颗不同的树, 则成为最小生成树的一条边, 并将这两棵树合并作为一颗树 4) 重复 3), 直到所有顶点都在一颗树内或者有 $n-1$ 条边为止

3.5.1.2 Prim 算法

选定一个点做为一个集合 a，剩下的点为另一个集合 b 将横跨两个集合且权重在其中最小的边加入最小生成树将刚刚加入最小生成树的边中不在集合 a 中的点加入集合 a，直到所有的点加入集合 a

```

1  int prim(int e[][maxx],int n)
2  {
3      int ans=0;
4      memset(book,false,sizeof(book));
5      book[0]=true;
6      for(int i=1; i<n; i++)
7          dis[i]=e[0][i];
8      for(int i=1; i<n; i++)
9      {
10         int minn=inf,u=-1;
11         for(int j=0; j<n; j++)
12         {
13             if(!book[j]&&dis[j]<minn)
14             {
15                 minn=dis[j];
16                 u=j;
17             }
18         }
19         if(ans==inf)
20             return -1;
21         ans+=minn;
22         book[u]=true;
23         for(int v=0; v<n; v++)
24             if(!book[v])
25                 dis[v]=min(dis[v],e[u][v]);
26     }

```



```

27     return ans;
28 }

```

3.6 Prufer 序列

将一棵树变成一个序列：每次选择树上标号最小的叶子，删掉它，将与它相连的那个点的标号加到序列里，直到只剩下 2 个点可以证明：任意一个长度为 $n-2$ 的 $1 \dots n$ 的序列都是某棵树的 Prufer 序列所以可以推出： n 个点的无根树个数为 n^{n-2} - 无根树转化为 prufer 序列 1. 找到编号最小的度数为 1 的点 2. 删除该节点并在序列中添加与该节点相连的节点的编号 3. 重复 1,2 操作，直到整棵树只剩下两个节点 - prufer 序列转化为无根树 1. 每次取出 prufer 序列中最前面的元素 u 2. 在点集中找到编号最小的没有在 prufer 序列中出现的元素 v 3. 给 u, v 连边然后分别删除 4. 最后在点集中剩下两个节点，给它们连边

给出标号为 1 到 N 的点，以及某些点最终的度数，允许在任意两点间连线，可产生多少棵度数满足要求的树？一个含有 n 个节点的 prufer 序列有 $n-2$ 个数字。Prufer Sequence 中的每个数是 $1 \sim n$ 中的一个数。而每一个 prufer 序列和一棵树是唯一对应的。

对于 cnt 个度数限制为 d_i 的点，那么它会在 prufer 序列中出现 $d_i - 1$ 次

故 Prufer Sequence 的总长度 $sum = \sum_{i=1}^{cnt} (d_i - 1)$

所以 Prufer Sequence 不同的排列总数为

$$C_{n-2}^{sum} \times \frac{sum!}{\prod_{i=1}^{cnt} (d_i - 1)!}$$

而剩下 $(n-2-sum)$ 个位置可以随意排列 $(n-cnt)$ 个点

所以

$$ans = C_{n-2}^{sum} \times \frac{sum!}{\prod_{i=1}^{cnt} (d_i - 1)!} \times (n - cnt)^{(n-2-sum)}$$

化简得

$$ans = \frac{(n-2)!}{(n-2-sum)! \times \prod_{i=1}^{cnt} (d_i - 1)!} \times (n - cnt)^{(n-2-sum)}$$

```

1  #include <iostream>
2  #include <cstdio>
3  #include <algorithm>
4  #include <cmath>
5  #include <cstring>
6  #include <cctype>
7  using namespace std;
8  int read()
9  {
10     int x=0,f=1;char ch=getchar();
11     while (!isdigit(ch)){if (ch=='-') f=-1;ch=getchar();}
12     while (isdigit(ch)){x=x*10+ch-48;ch=getchar();}
13     return x*f;
14 }
15 int n,cnt,d[1050],sum,tot;

```

```

16 struct BigInt
17 {
18     int num[10000], len;
19
20     BigInt()
21     {
22         memset(num, 0, sizeof(num));
23         len = 1;
24     }
25
26     BigInt operator* (const int &rhs) const
27     {
28         BigInt ans;
29         ans.len = len + 6;
30         for (int i = 1; i <= len; i++)
31             ans.num[i] += num[i] * rhs;
32         for (int i = 1; i < ans.len; i++)
33             if (ans.num[i] > 9)
34             {
35                 ans.num[i + 1] += ans.num[i] / 10;
36                 ans.num[i] %= 10;
37             }
38         while (!ans.num[--ans.len])
39             return ans;
40     }
41
42     BigInt operator/ (const int &rhs) const
43     {
44         BigInt ans = *this;
45         ans.len++;
46         for (int i = ans.len; i; i--)
47         {
48             ans.num[i - 1] += ans.num[i] % rhs * 10;
49             ans.num[i] /= rhs;
50         }
51         while (!ans.num[--ans.len]);
52         return ans;
53     }
54 }ans;
55 int main()
56 {
57     n = read();
58     for (int i = 1; i <= n; i++)
59     {
60         d[i] = read();
61         if (!d[i])

```

```

62     {
63         puts("0");
64         return 0;
65     }
66     if (d[i] != -1)
67     {
68         sum += d[i] - 1;
69         cnt++;
70     }
71 }
72 if (sum > 2 * n - 2)
73 {
74     puts("0");
75     return 0;
76 }
77 ans.num[1] = 1;
78 for (int i = n - 1 - sum; i < n - 1; i++)
79     ans = ans * i;
80 for (int i = 1; i <= n - 2 - sum; i++)
81     ans = ans * (n - cnt);
82 for (int i = 1; i <= n; i++)
83 {
84     for (int j = 2; j <= d[i] - 1; j++)
85         ans = ans / j;
86 }
87 for (int i = ans.len; i; i--)
88     printf("%d", ans.num[i]);
89 return 0;
90 }

```

```

1  from math import factorial
2
3
4  def C(n, m):
5      if m > n:
6          return 0
7      return factorial(n) // factorial(m) // factorial(n - m)
8
9
10 def pow(x, k):
11     res = 1
12     while k > 0:
13         if k & 1:
14             res *= x
15         x *= x
16         k >>= 1

```

```

17     return res
18
19
20 n = int(input())
21 k, Sum = 0, 0
22 a = []
23
24 for i in range(n):
25     x = int(input())
26     if x == -1:
27         k += 1
28     else:
29         x -= 1
30         Sum += x
31         a.append(x)
32
33 ans = C(n - 2, Sum) * factorial(Sum)
34 for x in a:
35     ans = ans // factorial(x)
36 ans *= pow(k, n - 2 - Sum)
37
38 print(ans)

```

3.7 二分图

可以分成两部分，使得这两部分内部没有边的图一个图是二分图等价于该图没有奇环

3.7.1 二分图匹配

给定一张二分图，求它的最大匹配匈牙利算法：每次找一条增广路

```

1 bool find(int x){
2     int i,j;
3     for (j=1;j<=m;j++){
4         if (line[x][j]==true && used[j]==false)
5             {
6                 used[j]=1;
7                 if (girl[j]==0 || find(girl[j])) {
8                     girl[j]=x;
9                     return true;
10                }
11            }
12    }
13    return false;
14 }

```

最小顶点覆盖：选最少的点覆盖所有边 | 二分图最小顶点覆盖 $=$ | 二分图最大匹配 | 最大独立集：选最多的点使得它们两两没边相连 | 二分图最大独立集 $=$ 总点数 $-$ | 二分图最小顶点覆盖 |

3.8 三元环计数

求无向图的三元环个数将点按照度数为第一关键字，标号为第二关键字从小到大排序，定义排序后每个点的序为 $\text{pos}[x]$ 对于每条无向边，变成有向边： pos 较小的点连向 pos 较大的点这样连完后每个点的出度都最多只有 $N^{0.5}$ 枚举三元环中 pos 最小的点 x ，然后枚举 x 的出边 y ，再枚举 y 的出边 z 如果 z 也在 x 的出边中的话就得到一个三元环时间复杂度： $O(N^{1.5})$

3.9 网络流

- dinic

```

1  const int maxn = 4e3+100;
2  const int maxm = 1e5+100;
3  const int inf = 0x7f7f7f7f;
4
5  typedef struct Dinic
6  {
7      typedef struct Edge
8      {
9          int u,v,w,nxt;
10     } Edge;
11     int head[maxn],hcnt;
12     int dep[maxn];
13     int cur[maxn];
14     Edge e[maxm];
15     int S,T,N;
16     void init()
17     {
18         memset(head,-1,sizeof head);
19         hcnt = 0;
20         S = T = N = 0;
21     }
22     void adde(int u,int v,int w)
23     {
24         e[hcnt].u = u,e[hcnt].v = v,e[hcnt].w = w;
25         e[hcnt].nxt = head[u];head[u] = hcnt++;
26         e[hcnt].u = v,e[hcnt].v = u,e[hcnt].w = 0;
27         e[hcnt].nxt = head[v];head[v] = hcnt++;
28     }
29     int bfs()
30     {
31         rep(i,0,N)
32         {
33             dep[i] = inf;

```

```

34     }
35     queue<int> q;
36     q.push(S); dep[S] = 0;
37     while(!q.empty())
38     {
39         int u = q.front();q.pop();
40         for(int i = head[u];~i;i = e[i].nxt)
41         {
42             int v = e[i].v,w = e[i].w;
43             if(w > 0 && dep[u] + 1 < dep[v])
44             {
45                 dep[v] = dep[u] + 1;
46                 if(v == T)
47                 {
48                     return 1;
49                 }
50                 q.push(v);
51             }
52         }
53     }
54     return dep[T] != inf;
55 }
56 int dfs(int s,int mw)
57 {
58     if(s == T) return mw;
59     for(int i = cur[s];~i;i=e[i].nxt)
60     {
61         cur[s] = i;
62         int v = e[i].v,w=e[i].w;
63         if(w <= 0 || dep[v] != dep[s] + 1)
64         {
65             continue;
66         }
67         int cw = dfs(v,min(w,mw));
68         if(cw <= 0)
69             continue;
70         e[i].w -= cw;
71         e[i^1].w += cw;
72         return cw;
73     }
74     return 0;
75 }
76 ll dinic()
77 {
78     ll res = 0;
79     while(bfs())

```

```

80     {
81         rep(i,0,N)
82         {
83             cur[i] = head[i];
84         }
85         while(int d = dfs(S,inf))
86         {
87             res += 1ll * d;
88         }
89     }
90     return res;
91 }
92 } Dinic;

```

- MCMF1

```

1 namespace mincostflow {
2     const int INF=0x3f3f3f3f;
3     struct node {
4         int to; int cap,cost; int rev;
5         node(int t=0,int c=0,int _c=0,int n=0):
6             to(t),cap(c),cost(_c),rev(n) {};
7     }; vector<node> edge[maxn];
8     void addedge(int from,int to,int cap,int cost) {
9         edge[from].push_back(node(to,cap,cost,edge[to].size()));
10        edge[to].push_back(node(from,0,-cost,edge[from].size()-1));
11    }
12    int dis[maxn];
13    bool mark[maxn];
14    void spfa(int s,int t,int n) {
15        memset(dis+1,0x3f,n*sizeof(int));
16        memset(mark+1,0,n*sizeof(bool));
17        static int Q[maxn],ST,ED;
18        dis[s]=0; ST=ED=0; Q[ED++]=s;
19        while (ST!=ED) {
20            int v=Q[ST]; mark[v]=0;
21            if ((++ST)==maxn) ST=0;
22            for (node &e:edge[v]) {
23                if (e.cap>0&&dis[e.to]>dis[v]+e.cost) {
24                    dis[e.to]=dis[v]+e.cost;
25                    if (!mark[e.to]) {
26                        if (ST==ED||dis[Q[ST]]>dis[e.to]) {
27                            Q[ED]=e.to,mark[e.to]=1;
28                            if ((++ED)==maxn) ED=0;
29                        } else {
30                            if (--ST<0) ST+=maxn;

```

```

31         Q[ST]=e.to,mark[e.to]=1;
32     }
33 }
34 }
35 }
36 }
37 } int cur[maxn];
38 int dfs(int x,int t,int flow) {
39     if (x==t||!flow) return flow;
40     int ret=0; mark[x]=1;
41     for (int &i=cur[x];i<(int)edge[x].size();i++) {
42         node &e=edge[x][i];
43         if (!mark[e.to]&&e.cap) {
44             if (dis[x]+e.cost==dis[e.to]) {
45                 int f=dfs(e.to,t,min(flow,e.cap));
46                 e.cap-=f; edge[e.to][e.rev].cap+=f;
47                 ret+=f; flow-=f;
48                 if (flow==0) break;
49             }
50         }
51     } mark[x]=0;
52     return ret;
53 }
54 pair<int,int> min_costflow(int s,int t,int n) {
55     int ret=0,ans=0;
56     int flow = INF;
57     while (flow) {
58         spfa(s,t,n); if (dis[t]==INF) break;
59         memset(cur+1,0,n*sizeof(int));
60         int len=dis[t],f;
61         while ((f=dfs(s,t,flow))>0)
62             ret+=f,ans+=len*f,flow-=f;
63     } return make_pair(ret,ans); //最大流, 最小费用
64 }
65 void init(int n) {
66     int i; for (int i = 1; i <= n; i++) edge[i].clear();
67 }
68 }

```

• MCMF2

```

1  const int maxn = 2e4+10;
2  namespace MCMF {
3      const int inf=0x3f3f3f3f;
4      struct Edge {
5          int to; int cap,cost; int rev;

```



```

6      Edge(int t=0,int c=0,int _c=0,int n=0):
7          to(t),cap(c),cost(_c),rev(n) {};
8  };
9  vector<Edge> edge[maxn];
10 void adde(int from,int to,int cap,int cost)
11 {
12     edge[from].push_back(Edge(to,cap,cost,edge[to].size()));
13     edge[to].push_back(Edge(from,0,-cost,edge[from].size()-1));
14 }
15
16 int dis[maxn];
17 bool mark[maxn];
18
19 void spfa(int s,int t,int n)
20 {
21     memset(dis,0x3f,sizeof dis);
22     memset(mark,0,sizeof mark);
23     static int Q[maxn],ST,ED;
24     dis[s]=0; ST=ED=0; Q[ED++]=s;
25     while (ST!=ED)
26     {
27         int v=Q[ST]; mark[v]=0;
28         if ((++ST)==maxn) ST=0;
29         for (Edge &e:edge[v])
30         {
31             if (e.cap>0&&dis[e.to]>dis[v]+e.cost)
32             {
33                 dis[e.to]=dis[v]+e.cost;
34                 if (!mark[e.to])
35                 {
36                     if (ST==ED||dis[Q[ST]]>dis[e.to])
37                     {
38                         Q[ED]=e.to,mark[e.to]=1;
39                         if ((++ED)==maxn) ED=0;
40                     }
41                     else
42                     {
43                         if ((--ST)<0) ST+=maxn;
44                         Q[ST]=e.to,mark[e.to]=1;
45                     }
46                 }
47             }
48         }
49     }
50 }
51 int cur[maxn];

```

```

52 int dfs(int x,int t,int flow)
53 {
54     if (x==t||!flow) return flow;
55     int ret=0; mark[x]=1;
56     for (int &i=cur[x];i<(int)edge[x].size();i++)
57     {
58         Edge &e=edge[x][i];
59         if (!mark[e.to]&&e.cap)
60         {
61             if (dis[x]+e.cost==dis[e.to])
62             {
63                 int f=dfs(e.to,t,min(flow,e.cap));
64                 e.cap-=f; edge[e.to][e.rev].cap+=f;
65                 ret+=f; flow-=f;
66                 if (flow==0) break;
67             }
68         }
69     }
70     mark[x]=0;
71     return ret;
72 }
73 pair<int,ll> mc(int s,int t,int n)
74 {
75     int ret=0;
76     ll ans=0;
77     int flow = inf;
78     while(flow)
79     {
80         spfa(s,t,n); if (dis[t]==inf) break;
81         memset(cur,0,sizeof cur);
82         int len=dis[t],f;
83         while ((f=dfs(s,t,flow))>0)
84             ret+=f,ans+=(ll)len*(ll)f,flow-=f;
85     }
86     return make_pair(ret,ans); //最大流, 最小费用
87 }
88 void init(int n)
89 {
90     for(int i = 1; i <= n; i++) edge[i].clear();
91 }
92 }

```

4 数学

4.1 BSGS

- 用于计算方程 $a^x = b \pmod{p}$

```

1  int BSGS(int a,int b,int p)
2  {
3      map<int,int > hash;
4      b %= p;
5      int t = (int)sqrt(p) + 1;
6      for(int j = 0; j < t; j++){
7          int val = 1ll * b * fpow(a,j,p) % p;
8          hash[val] = j;
9      }
10     a = fpow(a,t,p);
11     if(!a) return b == 0 ? 1 : -1;
12     for(int i = 0; i <= t;++i){
13         int val = fpow(a,i,p);
14         int j = hash.find(val) == hash.end() ? -1 : hash[val];
15         if(j >= 0 && i * t - j > 0) return i * t - j;
16     }
17     return -1;
18 }

```

4.2 扩展中国剩余定理

4.2.1 值在 `__int128` 以内

```

1  typedef __int128 ll;
2  void exgcd(ll a,ll b,ll &x,ll &y)
3  {
4      if(!b){
5          x = 1;y = 0;
6          return ;
7      }
8      exgcd(b,a % b,y,x);
9      y -= a / b * x;
10 }
11 ll inv(ll a,ll p)
12 {
13     ll x,y;
14     exgcd(a,p,x,y);
15     if(x < 0) x += p;
16     return x;
17 }

```

```

18 ll ex_crt()
19 {
20     bool flag = true;
21     ll m1,m2,c1,c2,tmp;
22     for(int i = 2;i <= n;++i){//m[i]为模,c[i]为余数
23         m1 = m[i - 1];m2 = m[i];
24         c1 = c[i - 1];c2 = c[i];
25         tmp = gcd(m1,m2);
26         if((c2 - c1) % tmp != 0) {
27             flag = false;
28             return -1;
29         }
30         m[i] = m1 / tmp * m2;
31         c[i] = inv(m1 / tmp,m2 / tmp) * (c2 - c1) / tmp % (m2 / tmp) * m1 + c1;
32         c[i] = (c[i] % m[i] + m[i]) % m[i];
33     }
34     return c[n];
35 }

```

4.2.2 值在 __int128 以外

```

1 import sys
2 def exgcd(a, b):
3     if b == 0:
4         return (a, 1, 0)
5     q = a // b
6     g, y, x = exgcd(b, a - q * b)
7     y -= q * x
8     return (g, x, y)
9
10 def gcd(a, b):
11     if b == 0:
12         return a
13     else:
14         return gcd(b, a % b)
15
16 N, lim = map(int, input().split())
17 M, ans = map(int, input().split())
18 f = 0
19 for i in range(0, N - 1):
20     m, r = map(int, input().split())
21     A = M
22     B = m
23     C = ((r - ans) % m)
24     if (ans - r) % gcd(M, m) != 0:

```

```
25     f = 1
26     g, x, y = exgcd(A, B)
27     x = (x * C // g) % (B // g)
28     ans += x * M
29     M *= B // g
30     ans %= M
31
32     if ans < 0:
33         ans += M
34     if f == 1:
35         ans = -1
36         print(ans)
37     else:
38         print(ans)
39     # print ans
```

4.3 扩展卢卡斯

```
1  ll n, m, p;
2  ll exgcd(ll a, ll b, ll &x, ll &y){
3      if(!b){
4          x = 1, y = 0;
5          return a;
6      }
7      ll res = exgcd(b, a % b, x, y);
8      ll t = x;
9      x = y;
10     y = t - a / b * y;
11     return res;
12 }
13
14 ll fpow(ll a, ll b, ll mod){
15     ll res = 1;
16     while(b){
17         if(b & 1) res = (res * a) % mod;
18         a = (a * a) % mod;
19         b >>= 1;
20     }
21     return res;
22 }
23
24 ll inv(ll a, ll p){
25     ll x, y;
26     exgcd(a, p, x, y);
27     if(x + p > p) return x;
```

```

28     return x + p;
29 }
30
31 inline ll crt(ll n, ll mod)
32 {
33     return n * (p / mod) % p * inv(p / mod, mod) % p;
34 }
35
36 ll fac(ll n, ll p, ll k){ //k = p^x
37     if(!n) return 1;
38     ll ans = 1;
39     for(int i = 2; i <= k; ++i)
40         if(i % p) ans = ans * i % k;
41     ans = fpow(ans, n / k, k);
42     for(int i = 2; i <= n % k; ++i)
43         if(i % p) ans = ans * i % k;
44     return ans * fac(n / p, p, k) % k;
45 }
46
47 ll C(ll n, ll m, ll p, ll k){ //k = p^x
48     if(n < m) return 0;
49     ll a = fac(n, p, k), b = fac(m, p, k), c = fac(n - m, p, k);
50     ll cnt = 0;
51     for(ll i = p; i <= n; i *= p) cnt += n / i;
52     for(ll i = p; i <= m; i *= p) cnt -= m / i;
53     for(ll i = p; i <= n - m; i *= p) cnt -= (n - m) / i;
54     return a * inv(b, k) % k * inv(c, k) % k * fpow(p, cnt, k) % k;
55 }
56
57 ll ex_Lucas()
58 {
59     ll t = p, ans = i;
60     t /= i;
61 }
62     ans = (ans + crt(C(n, m, i, tmp), tmp)) % p;
63 }
64     if(t > 1) ans = (ans + crt(C(n, m, t, t), t)) % p;
65     return ans % p;
66 }

```

4.4 fft

- 多项式快速计算

```

1 const double PI = acos(-1.0);
2 struct Complex{

```

```

3     double x, y; // 实部和虚部 x+yi
4     Complex(double _x = 0.0, double _y = 0.0) {
5         x = _x;
6         y = _y;
7     }
8     Complex operator-(const Complex &b) const {
9         return Complex(x - b.x, y - b.y);
10    }
11    Complex operator+(const Complex &b) const {
12        return Complex(x + b.x, y + b.y);
13    }
14    Complex operator*(const Complex &b) const {
15        return Complex(x * b.x - y * b.y, x * b.y + y * b.x);
16    }
17 };
18 /*
19  * 进行 FFT 和 IFFT 前的反转变换。
20  * 位置 i 和 (i 二进制反转后位置) 互换
21  * len 必须去 2 的幂
22  */
23 void change(Complex y[], int len){
24     for (int i = 1, j = len / 2; i < len - 1; ++i){
25         if (i < j) swap(y[i], y[j]);
26         // 交换互为小标反转的元素, i < j 保证交换一次
27         // i 做正常的+1, j 左反转类型的+1, 始终保持 i 和 j 是反转的
28         k = len / 2;
29         while(j >= k){
30             j -= k;
31             k >>= 1;
32         }
33         if (j < k) j += k;
34     }
35 }
36 /*
37  * 做 FFT
38  * len 必须为 2^k 形式,
39  * on==1 时是 DFT, on==-1 时是 IDFT
40  */
41 void fft(Complex y[], int len, int on) {
42     change(y, len);
43     for (int h = 2; h <= len; h <= 1) {
44         Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
45         for (int j = 0; j < len; j += h) {
46             Complex w(1, 0);
47             for (int k = j; k < j + h / 2; k++) {
48                 Complex u = y[k];

```

```

49         Complex t = w * y[k + h / 2];
50         y[k] = u + t;
51         y[k + h / 2] = u - t;
52         w = w * wn;
53     }
54 }
55 }
56 if (on == -1)
57     for (int i = 0; i < len; i++) y[i].x /= len;
58 }

```

4.5 fwt

- 计算集合卷积

```

1 void fwt(ll *a, int n)
2 {
3     for(int d = 1; d < n; d <= 1){
4         for(int m = d < 1, i = 0; i < n; i += m){
5             for(int j = 0; j < d; ++j){
6                 ll x = a[i + j], y = a[i + j + d];
7                 a[i + j] = (x + y), a[i + j + d] = (x - y);
8                 //xor:a[i + j] = x + y, a[i + j + d] = (x - y + mod) % mod;
9                 //and:a[i + j] = x + y;
10                //or:a[i + j + d] = x + y;
11            }
12        }
13    void ufwt(ll *a, int n)
14    {
15        for(int d = 1; d < n; d <= 1){
16            for(int m = d < 1, i = 0; i < n; i += m){
17                for(int j = 0; j < d; ++j){
18                    ll x = a[i + j], y = a[i + j + d];
19                    a[i + j] = 1LL * (x + y) / 2, a[i + j + d] = (1LL * (x - y) / 2);
20                    //xor:a[i + j] = (x + y) / 2, a[i + j + d] = (x - y) / 2;
21                    //and:a[i + j] = x - y;
22                    //or:a[i + j + d] = y - x;
23                }
24            }
25        void work(ll *a, ll *b, int n)
26        {
27            fwt(a, n);
28            fwt(b, n);
29            for(int i = 0; i < n; ++i) a[i] *= b[i];
30            ufwt(a, n);
31        }

```


4.6 polya

- m 种颜色, n 个长度

```

1
2  const int N = 1e5 + 7;
3  int euler(int n){
4      int ans = n;
5      for(int i = 1; i <= cnt && prime[i] * prime[i] <= n; ++i)
6          if(n % prime[i] == 0){
7              ans -= ans / prime[i];
8              while(n % prime[i] == 0) n /= prime[i];
9          }
10     if(n > 1) ans -= ans / n;
11     return ans;
12 }
13 int n,m;
14 int main()
15 {
16     getprime();
17     while(scanf("%d%d",&m,&n) && (n || m)){
18         ll ans = 0;
19         for(int i = 1; i * i <= n; ++i){
20             if(n % i == 0)
21                 ans += i * i == n ? euler(i) * fpow(m,n / i) : euler(i) * fpow(m,n / i) + euler(n / i)
22                     * fpow(m,i);
23         }
24         ans += n & 1 ? n * fpow(m,n / 2 + 1) : (fpow(m,n >> 1) + fpow(m,n / 2 + 1)) * (n >> 1);
25         printf("%lld\n",ans / (n << 1LL));
26     }
27     return 0;

```

4.7 大素数

- 包含分解, 判断过程为 $O(n^{\frac{1}{4}})$

```

1  typedef long long ll;
2  int pri[] = {2,3,5,7,11,13,17,19,23,29,31};
3  ll minfacotr;
4  ll mult(ll a, ll b, ll mod) // 大数乘法
5  {
6      ll ans = 0;

```

```

7   while(b)
8   {
9       if (b&1)
10      {
11          ans+= a;
12          if (ans >= mod)
13              ans -= mod;
14      }
15      b >>= 1;
16      a <<= 1;
17      if (a >= mod)
18          a-= mod;
19  }
20  return ans;
21 }
22 ll qpow(ll x, ll n, ll mod)
23 {
24     ll ans = 1;
25     while(n)
26     {
27         if (n&1)
28             ans = mult(ans, x, mod);
29         x = mult(x, x, mod);
30         n >>= 1;
31     }
32     return ans;
33 }
34 bool wintness(ll n, ll a)
35 {
36     ll p = qpow(a, n-1, n);
37     if(p != 1)
38         return false;
39     ll s = n - 1;
40     while(!(s&1)&& p == 1)
41     {
42         s >>= 1;
43         p = qpow(a, s, n);
44     }
45     if (p == 1 || p == n - 1)
46         return true;typedef long long ll;
47 int pri[] = {2,3,5,7,11,13,17,19,23,29,31};
48 ll minfacotr;
49 ll mult(ll a, ll b, ll mod) // 大数乘法
50 {
51     ll ans = 0;
52     while(b)

```

```
53 {
54     if (b&1)
55     {
56         ans+= a;
57         if (ans >= mod)
58             ans -= mod;
59     }
60     b >>= 1;
61     a <<= 1;
62     if (a >= mod)
63         a-= mod;
64 }
65 return ans;
66 }
67 ll qpow(ll x, ll n, ll mod)
68 {
69     ll ans = 1;
70     while(n)
71     {
72         if (n&1)
73             ans = mult(ans, x, mod);
74         x = mult(x, x, mod);
75         n >>= 1;
76     }
77     return ans;
78 }
79 bool wintness(ll n, ll a)
80 {
81     ll p = qpow(a, n-1, n);
82     if(p != 1)
83         return false;
84     ll s = n - 1;
85     while(!(s&1)&& p =
86         return false;
87 }
88 bool miller_rabin(ll n) // 判断素数
89 {
90     if (n < 32)
91     {
92         for(int i = 0; i < 11; ++i)
93             if(n == pri[i])
94                 return true;
95         return false;
96     }
97     for(int i = 0 ; i < 10; ++i)
98         if(!wintness(n,pri[i]))
```

```

99         return false;
100     return true;
101 }
102 ll gcd(ll a, ll b)
103 {
104     return b ? gcd(b, a % b) : a;
105 }
106 ll pollard_rho(ll n, ll c) // Pollard_rho 大数分解
107 {
108     ll x= rand() % n, y = x, i = 1, k = 2, d;
109     while(1)
110     {
111         i++;
112         x =(mult(x, x, n) + c) % n;
113         d = gcd(y-x+n,n);
114         if(d > 1 && d < n)
115             return d;
116         if(y == x)
117             return n;
118         if (i == k)
119         {
120             y = x;
121             k <= 1;
122         }
123     }
124 }
125 ll fac[maxn];
126 int cnt = 0;
127 void findfactor(ll n)
128 {
129     if(miller_rabin(n))
130     {
131         minfacotr = min(minfacotr,n); // 求最小素数因子
132         fac[++cnt] = n; // 储存素数
133         return ;
134     }
135     ll p = n;
136     while(p >= n)
137         p = pollard_rho(n, rand() % (n - 1) + 1);
138     findfactor(p);
139     findfactor(n / p);
140 }

```

4.8 线性基

```

1  struct L_B{
2      ll d[64],p[64];
3      int cnt;
4      void init(){// 初始化
5          memset(d,0,sizeof d);
6          memset(p,0,sizeof p); // 单位基
7          cnt = 0; // 单位基中1的数量
8      }
9      bool insert(ll val){ // 插入 如果x的1数位不存在则插入val, 存在则赋值为0
10         for(int i = 63;i >= 0;--i)
11             if(val & (1LL << i)){
12                 if(!d[i]){
13                     d[i] = val;
14                     break;
15                 }
16                 val ^= d[i];
17             }
18         return val > 0;
19     }
20     ll query_max(){ // 查询最大异或和
21         ll ret = 0;
22         for(int i = 63;i >= 0;--i)
23             if((ret ^ d[i]) > ret)
24                 ret ^= d[i];
25         return ret;j >= 0;--j)
26             if(d[i] & (1LL << j)) d[i] ^= d[j];
27         for(int i = 0;i <= 63;++i)
28             if(d[i]) p[cnt++] = d[i];
29     }
30     ll kth_query(ll k){ // 查询第k小值
31         int ret = 0;
32         if(k >= (1LL << cnt))
33             return -1;
34         for(int i = 63;i >= 0;--i)
35             if(k & (1LL << i)) ret ^= p[i];
36         return ret;
37     }
38 }
39 L_B merge(const L_B &n1,const L_B &n2) // 暴力合并两个线性基
40 {
41     L_B ret = n1;
42     for(int i = 63;i >= 0;--i)
43         if(n2.d[i]) ret.insert(n1.d[i]);
44     return ret;
45 }

```

4.9 洲阁筛

4.10 计算素数 k 次幂前缀和

```

1  const int N = 1e6 + 7;
2  ll n,k,sqrtn;
3  int m;
4  ll a[N << 1],cnt;
5  int pos1[N],pos2[N];
6  ull g[N << 1]; // 构造函数
7  int pri[N],pcnt;
8  bool ispri[N];
9  void get_pri()
10 {
11     for(int i = 2;i < N;++i){
12         if(!ispri[i]) pri[++pcnt] = i;
13         for(int j = 1;j <= pcnt && i * pri[j] < N;++j){
14             ispri[i * pri[j]] = 1;
15             if(i % pri[j] == 0) break;
16         }
17     }
18 }
19
20 inline int get_pos(ll x)
21 {
22     return x <= sqrtn ? pos1[x] : pos2[n / x];
23 }
24 void Discretization(){ // 离散化处理
25     for(ll i = 1,j;i <= n;i = j + 1){
26         a[++cnt] = n / i;
27         j = n / (n / i);
28     }
29     reverse(a + 1,a + 1 + cnt);
30     for(int i = 1;i <= cnt;++i)
31         if(a[i] <= sqrtn) pos1[a[i]] = i;
32         else pos2[n / a[i]] = i;
33 }
34 void calc_g() // 预处理构造函数
35 {
36     for(int i = 1;i <= cnt;++i) g[i] = a[i] - 1;
37     for(int i = 1;i <= m;++i)
38         for(int j = cnt;j >= 1 && a[j] >= pri[i] * pri[i];--j)
39             g[j] -= g[get_pos(a[j] / pri[i])] - g[get_pos(pri[i] - 1)];
40 }
41 ull calc_h(ll i,ll j){ // 计算递归函数
42     if(i <= 1) return 0;

```

```

43     ull sum = 0;
44     int res;
45     for(res = j; res <= m && pri[res] * pri[res] <= i; ++res)
46         for(ll o = pri[res], e = 1; o <= i; o *= pri[res], ++e)
47             sum += (ull)(e * k + 1) * (calc_h(i / o, res + 1) + 1);
48     if(pri[res - 1] <= i)
49         sum += (ull)(k + 1) * (g[get_pos(i)] - g[get_pos(pri[res - 1])]);
50     return sum;
51 }
52 int t;
53 int main()
54 {
55     get_pri();
56     scanf("%lld%lld", &n, &k);
57     sqrtn = (ll)sqrt(n); cnt = 0;
58     m = upper_bound(pri + 1, pri + 1 + pcnt, sqrtn) - pri - 1;
59     Discretization();
60     calc_g();
61     ull ans = (ull)(k + 1) * (g[get_pos(n)] - m);
62     for(int i = 1; i <= m; ++i)
63         for(ll j = pri[i], e = 1; j <= n; j *= pri[i], ++e)
64             ans += (ull)(e * k + 1) * (calc_h(n / j, i + 1) + 1);
65     ans++;
66     printf("%ull\n", ans);
67     return 0;
68 }

```

4.11 自然幂数和

```

1 void init()
2 {
3     //预处理组合数
4     for(int i = 0; i < N; ++i){
5         C[i][0] = C[i][i] = 1;
6         if (i == 0)
7             continue;
8         for (int j = 1; j < i; ++j)
9             C[i][j] = (C[i - 1][j] % mod + C[i - 1][j - 1] % mod) % mod;
10    }
11    //预处理逆元
12    inv[1] = 1;
13    for (int i = 2; i < N; ++i)
14        inv[i] = (mod - mod / i) * inv[mod % i] % mod;
15    //预处理伯努利数
16    B[0] = 1;

```

```
17     for (int i = 1; i < N; ++i)
18     {
19         ll ans = 0;
20         if (i == N - 1)
21             break;
22         for (int j = 0; j < i; ++j)
23         {
24             ans += C[i + 1][j] * B[j];
25             ans %= mod;
26         }
27         ans *= -inv[i + 1];
28         ans = (ans % mod + mod) % mod;
29         B[i] = ans;
30     }
31 }
32 ll Work(int k)
33 {
34     ll ans = inv[k + 1];
35     ll sum = 0;
36     for(int i=1; i<=k+1; ++i)
37     {
38         sum += C[k + 1][i] * tmp[i] % mod * B[k + 1 - i] % mod;
39         sum %= mod;
40     }
41     ans *= sum;
42     ans %= mod;
43     return ans;
44 }
45 int main()
46 {
47     int t;
48     init();
49     scanf("%d", &t);
50     while(t--)
51     {
52         int k;
53         scanf("%lld %d", &n, &k);
54         n %= mod;
55         tmp[0] = 1;
56         for(int i = 1; i < N; ++i)
57             tmp[i] = tmp[i - 1] * (n + 1) % mod;
58         printf("%lld\n", Work(k));
59     }
60     return 0;
61 }
```

4.12 组合数打表

```

1 ll f[N],inv[N];
2 inline ll C(ll a,ll b){return a < 0 || b < 0 || a < b ? 0 : f[a] * inv[b] % mod * inv[a - b] % mod;}
3 void get_table(int index)
4 {
5     f[0] = inv[0] = 1;
6     for(int i = 1;i <= index;++i) f[i] = f[i - 1] * i % mod;
7     inv[index] = fpow(f[index],mod - 2);
8     for(;index;--index)
9         inv[index - 1] = inv[index] * index % mod;
10 }

```

4.13 杜教筛

```

1 const int N = 4e6 + 7;
2 const int mod = 1e9 + 7;
3 int prime[N],cnt;
4 ll g[N],phi[N];
5 bool isprime[N];
6 ll n,m;
7 inline ll gao(ll x)
8 {
9     x %= mod;
10    return (x + 1) % mod * x % mod * inv2 % mod;
11 }
12 void get_phi() // 预处理phi或者Mobius, $n^{2/3}$ 的前缀和
13 {
14     phi[1] = 1;
15     for(int i = 2;i < N;++i){
16         if(!isprime[i]){
17             prime[++cnt] = i;
18             phi[i] = i - 1;
19         }
20         for(int j = 1;j <= cnt && i * prime[j] < N;++j)
21         {
22             isprime[i * prime[j]] = 1;
23             if(i % prime[j] == 0){
24                 phi[i * prime[j]] = phi[i] * prime[j];
25                 break;
26             }
27             phi[i * prime[j]] = phi[i] * (prime[j] - 1);
28         }
29     }
30     for(int i = 1;i < N;++i) g[i] = (g[i - 1] + phi[i] * i % mod) % mod;

```

```

31 }
32
33 inline ll cal(ll x) //
34 {
35     x %= mod;
36     return x * (2 * x + 1) % mod * (x + 1) % mod * inv6 % mod;
37 }
38 map<ll ,ll>mp; // 存已到达情况
39 ll solve(ll pos) // 递归
40 {
41     if(pos < N) return g[pos];
42     if(mp[pos]) return mp[pos];
43     ll res = cal(pos),last;
44     for(ll i = 2;i <= pos;i = last + 1){
45         last = pos / (pos / i);
46         res = ((res - (gao(last) - gao(i - 1)) * solve(pos / i) % mod) % mod + mod) % mod;
47     }
48     mp[pos] = res;
49     return res;
50 }
51 ll work(ll pos)
52 {
53     ll last,ans = 0;
54     for(ll i = 1;i <= pos;i = last + 1){
55         last = pos / (pos / i);
56         ans = (ans + (last - i + 1) % mod * solve(pos / i) % mod) % mod + mod;
57         ans %= mod;
58     }
59     return (ans + pos) % mod * inv2 % mod;
60 }
61 int main()
62 {
63     get_phi();
64     scanf("%lld%lld",&n,&m);
65     printf("%lld\n",((work(m) - work(n - 1)) % mod + mod) % mod);
66     return 0;
67 }

```

4.14 二次剩余

4.14.1 解 $x^2 = a(mod p)$

```

1 struct T{
2     ll p, d;
3 };
4 ll w;

```

```

5 //二次域乘法
6 T multi_er(T a, T b, ll m)
7 {
8     T ans;
9     ans.p = (a.p * b.p % m + a.d * b.d % m * w % m) % m;
10    ans.d = (a.p * b.d % m + a.d * b.p % m) % m;
11    return ans;
12 }
13 //二次域上快速幂
14 T power(T a, ll b, ll m)
15 {
16     T ans;
17     ans.p = 1;
18     ans.d = 0;
19     while(b){
20         if(b & 1)
21             ans = multi_er(ans, a, m);
22         b >>= 1;
23         a = multi_er(a, a, m);
24     }
25     return ans;
26 }
27 //求勒让德符号
28 ll Legendre(ll a, ll p)
29 {
30     return qpow(a, (p-1)>>1, p);
31 }
32 ll mod(ll a, ll m)
33 {
34     a %= m;
35     if(a < 0) a += m;
36     return a;
37 }
38 ll Solve(ll n, ll p)
39 {
40     if(p == 2) return 1;
41     if (Legendre(n, p) + 1 == p)
42         return -1;
43     ll a = -1, t;
44     while(true) {
45         a = rand() % p;
46     }
47     int main()
48     {
49         int t;
50         scanf("%d", &t);

```

```

51 while(t--){
52     int n, p;
53     scanf("%d %d",&n,&p);
54     n %= p;
55     int a = Solve(n, p); //x * x = n % p
56     if(a == -1) {
57         puts("No root");
58         continue;
59     }
60     int b = p - a;
61     if(a > b) swap(a, b);
62     if(a == b)
63         printf("%d\n",a);
64     else printf("%d %d\n",a,b);
65 }
66 return 0;
67 }

```

4.15 高精度

```

1 struct BigInteger {
2     typedef unsigned long long ll;
3
4     static const int BASE = 100000000;
5     static const int WIDTH = 8;
6     vector<int> s;
7
8     BigInteger& clean(){while(!s.back() && s.size() > 1) s.pop_back(); return *this;}
9     BigInteger(ll num = 0) {*this = num;}
10    BigInteger(string s) {*this = s;}
11    BigInteger& operator = (long long num) {
12        s.clear();
13        do {
14            s.push_back(num % BASE);
15            num /= BASE;
16        } while (num > 0);
17        return *this;
18    }
19    BigInteger& operator = (const string& str) {
20        s.clear();
21        int x, len = (str.length() - 1) / WIDTH + 1;
22        for (int i = 0; i < len; i++) {
23            int end = str.length() - i*WIDTH;
24            int start = max(0, end - WIDTH);
25            sscanf(str.substr(start,end-start).c_str(), "%d", &x);

```

```

26         s.push_back(x);
27     }
28     return (*this).clean();
29 }
30
31 BigInteger operator + (const BigInteger& b) const {
32     BigInteger c; c.s.clear();
33     for (int i = 0, g = 0; ; i++) {
34         if (g == 0 && i >= s.size() && i >= b.s.size()) break;
35         int x = g;
36         if (i < s.size()) x += s[i];
37         if (i < b.s.size()) x += b.s[i];
38         c.s.push_back(x % BASE);
39         g = x / BASE;
40     }
41     return c;
42 }
43
44 BigInteger operator - (const BigInteger& b) const {
45     assert(b <= *this); // 减数不能大于被减数
46     BigInteger c; c.s.clear();
47     for (int i = 0, g = 0; ; i++) {
48         if (g == 0 && i >= s.size() && i >= b.s.size()) break;
49         int x = s[i] + g;
50         if (i < b.s.size()) x -= b.s[i];
51         if (x < 0) {g = -1; x += BASE;} else g = 0;
52         c.s.push_back(x);
53     }
54     return c.clean();
55 }
56
57 BigInteger operator * (const BigInteger& b) const {
58     int i, j; ll g;
59     vector<ll> v(s.size()+b.s.size(), 0);
60     BigInteger c; c.s.clear();
61     for(i=0;i<s.size();i++) for(j=0;j<b.s.size();j++) v[i+j]+=ll(s[i])*b.s[j];
62     for (i = 0, g = 0; ; i++) {
63         if (g == 0 && i >= v.size()) break;
64         ll x = v[i] + g;
65         c.s.push_back(x % BASE);
66         g = x / BASE;
67     }
68     return c.clean();
69 }
70
71 BigInteger operator / (const BigInteger& b) const {
72     assert(b > 0); // 除数必须大于0
73     BigInteger c = *this; // 商:主要是让c.s和(*this).s的vector一样大
74     BigInteger m; // 余数:初始化为0

```

```

72         for (int i = s.size()-1; i >= 0; i--) {
73             m = m*BASE + s[i];
74             c.s[i] = bsearch(b, m);
75             m -= b*c.s[i];
76         }
77         return c.clean();
78     }
79     BigInteger operator % (const BigInteger& b) const { //方法与除法相同
80         BigInteger c = *this;
81         BigInteger m;
82         for (int i = s.size()-1; i >= 0; i--) {
83             m = m*BASE + s[i];
84             c.s[i] = bsearch(b, m);
85             m -= b*c.s[i];
86         }
87         return m;
88     }
89
90     int bsearch(const BigInteger& b, const BigInteger& m) const{
91         int L = 0, R = BASE-1, x;
92         while (1) {
93             x = (L+R)>>1;
94             if (b*x<=m) {if (b*(x+1)>m) return x; else L = x;}
95             else R = x;
96         }
97     }
98     BigInteger& operator += (const BigInteger& b) {*this = *this + b; return *this;}
99     BigInteger& operator -= (const BigInteger& b) {*this = *this - b; return *this;}
100    BigInteger& operator *= (const BigInteger& b) {*this = *this * b; return *this;}
101    BigInteger& operator /= (const BigInteger& b) {*this = *this / b; return *this;}
102    BigInteger& operator %=(const BigInteger& b) {*this = *this % b; return *this;}
103
104    bool operator < (const BigInteger& b) const {
105        if (s.size() != b.s.size()) return s.size() < b.s.size();
106        for (int i = s.size()-1; i >= 0; i--)
107            if (s[i] != b.s[i]) return s[i] < b.s[i];
108        return false;
109    }
110    bool operator >(const BigInteger& b) const{return b < *this;}
111    bool operator<=(const BigInteger& b) const{return !(b < *this);}
112    bool operator>=(const BigInteger& b) const{return !(*this < b);}
113    bool operator!=(const BigInteger& b) const{return b < *this || *this < b;}
114    bool operator==(const BigInteger& b) const{return !(b < *this) && !(b > *this);}
115 };
116
117 ostream& operator << (ostream& out, const BigInteger& x) {

```

```

118     out << x.s.back();
119     for (int i = x.s.size()-2; i >= 0; i--) {
120         char buf[20];
121         sprintf(buf, "%08d", x.s[i]);
122         for (int j = 0; j < strlen(buf); j++) out << buf[j];
123     }
124     return out;
125 }
126
127 istream& operator >> (istream& in, BigInteger& x) {
128     string s;
129     if (!(in >> s)) return in;
130     x = s;
131     return in;
132 }

```

4.16 反素数

4.16.1 求小于 n 并且因子个数最多的那个数

```

1  int pri[] = {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61};
2  ll n, ans, tmp;
3  const ll INF = 0x3f3f3f3f;
4  void dfs(int step, ll sum, ll num) //num 为因子个数
5  {
6      if(step == 16) return;
7      if(num > tmp){
8          ans = sum;
9          tmp = num;
10     }
11     if(sum > n) return ;
12
13     if(num == tmp && sum < ans) //因子个数一样,选取较小的那个数
14         ans = sum;
15     for(int i = 1; i <= 63; ++i){
16         if(n / pri[step] < sum) break;
17         dfs(step + 1, sum * pri[step], num * (i + 1));
18     }
19 }
20
21 //求一个最小的正整数,使得它的因子个数为 n
22 int p[16] = {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53};
23 void dfs(int dept, ull tmp, int num)
24 {
25     if(num > n) return;
26     if(num == n && ans > tmp)

```

```

27     ans = tmp;
28     for(int i=1; i <= 63;++i){
29         if(ans / p[dept] < tmp)
30             break;
31         dfs(dept+1,tmp * p[dept],num * (i + 1));
32     }
33 }

```

4.17 高斯消元

```

1  const int N = 307;
2  int x[N],a[N][N]; // x[N]解集, a[N][N]系数
3  bool free_x[N];
4  int gcd(int a,int b){return b ? gcd(b,a % b) : a;}
5  int lcm(int a,int b){return a / gcd(b,a % b) * b;}
6  int Gauss(int equ,int var)//equ个方程, var个变元
7  {
8      int free_x_num,i,j,row,max_r,col;//row表示行,col表示列,max_r表示列最大的行,free_x_num变元数量
9      int free_index,LCM,ta,tb,temp;// free_index变元下标
10     for(i = 0;i <= var;++i){
11         x[i] = 0;
12         free_x[i] = true;//第i个元素是否是变元
13     }
14     for(row = 0,col = 0;row < equ && col < var;++row,++col){
15         max_r = row;
16         //找到col最大的行, 进行交换 (除法时减小误差)
17         for(i = row + 1;i < equ;++i) if(abs(a[i][col]) > abs(a[max_r][col])) max_r = i;
18         //与第row行交换
19         if(max_r != row) for(j = row;j < var + 1;++j) swap(a[row][j],a[max_r][j]);
20         if(a[row][col]==0){
21             //说明该col列第row行以下全是0了, 则处理当前行的下一列.
22             row--;
23             continue;
24         }
25         for(i = row + 1;i < equ;++i)//枚举被删行
26             if(a[i][col]){
27                 LCM = lcm(abs(a[i][col]),abs(a[row][col]));
28                 ta = LCM / abs(a[i][col]);
29                 tb = LCM / abs(a[row][col]);
30                 if(a[i][col] * a[row][col] < 0)tb = -tb;//异号的情况是相加
31                 for(j = col;j < var + 1;++j)
32                     a[i][j] = a[i][j] * ta - a[row][j] * tb;
33             }
34         /*求解小数解,防止溢出
35         for(int i = row + 1; i < equ; ++i)

```



```

36         if(fabs(a[i][col]) > eps){
37             double t1 = a[i][col]/a[row][col];
38             for(int j = col; j <= var;++j) a[i][j] -= a[row][j] * t1;
39         }*/
40     }
41     for (i = row; i < equ; ++i) if(a[i][col]) return -1; // 无解
42     if (row < var){// 多解
43         for(i = row - 1; i >= 0; --i){
44             free_x_num = 0;
45             for (j = 0; j < var; ++j)
46                 if(a[i][j] && free_x[j]) free_x_num++, free_index = j;
47             if (free_x_num > 1) continue; // 无法求解出确定的变元.
48             temp = a[i][var];
49             for (j = 0; j < var; ++j) if (a[i][j] && j != free_index) temp -= a[i][j] * x[j];
50             x[free_index] = temp / a[i][free_index]; // 求出该变元.
51             free_x[free_index] = 0; // 该变元是确定的.
52         }
53         return var - row; // 自由变元有 var - row 个.
54     }
55     for (i = var - 1; i >= 0; --i){// 唯一解
56         temp = a[i][var];
57         for (j = i + 1; j < var; ++j)
58             if (a[i][j]) temp -= a[i][j] * x[j];
59         if (temp % a[i][i]) return -2; // 说明有浮点数解, 但无整数解.
60         x[i] = temp / a[i][i];
61     }
62     return 0;
63 }

```

4.18 高斯消元解异或方程

```

1  int Guass_xor(int equ, int var)
2  {
3      int row, col;
4      for(row = 0, col = 0; row < equ && col < var; ++row, ++col){
5          int maxr = row;
6          for(int i = row; i < equ; ++i)
7              if(a[i][col] > a[maxr][col]) maxr = i;
8              if(maxr != row) for(int j = col; j <= var; ++j) swap(a[maxr][j], a[row][j]);
9              if(!a[row][col]){
10                 row--;
11                 continue;
12             }
13             for(int i = row + 1; i < equ; ++i)
14                 if(a[i][col])

```

```

15         for(int j = col; j <= var; ++j) a[i][j] ^= a[row][j];
16     }
17     for(int i = row; i < equ; ++i) if(a[i][col]) return -1; //无可行解
18     if(row < var) // 存在多解
19         for(int i = row - 1; i >= 0; --i){
20             int num = 0, index;
21             for(int j = 0; j < var; ++j)
22                 if(a[i][j] && vis[j]) num++, index = j;
23             if(num > 1) continue;
24             for(int j = 0; j < var; ++j)
25                 if(j != index) x[index] ^= (a[i][j] && x[j]);
26             vis[index] = false;
27         }
28     for(int i = var - 1; i >= 0; --i){ //唯一解
29         x[i] = a[i][var];
30         for(int j = i + 1; j < var; ++j) x[i] ^= (a[i][j] && x[j]);
31     }
32     return 0;
33 }

```

4.19 矩阵类（快速幂）

```

1  int sz;
2  struct Matrix{
3      ll a[10][10];
4      Matrix(){
5          for(int i = 0; i < sz; ++i)
6              for(int j = 0; j < sz; ++j)
7                  if(i == j) a[i][j] = 1;
8                  else a[i][j] = 0;
9      }
10     void init(){
11         memset(a, 0, sizeof a);
12     }
13     void show(){
14         for(int i = 0; i < sz; ++i)
15             for(int j = 0; j < sz; ++j)
16                 dd(i), dd(j), de(a[i][j]);
17     }
18     void Relation_matrix(ll *s){
19         init();
20         for(int i = 0; i < sz; ++i){
21             if(i + 1 < sz) a[i][i + 1] = 1;
22             a[sz - 1][i] = s[i];
23         }

```

```

24     }
25     Matrix operator *(const Matrix& tmp) const{
26         Matrix res;
27         res.init();
28         for(int i = 0;i < sz;++i)
29             for(int j = 0;j < sz;++j)
30                 for(int k = 0;k < sz;++k)
31                     res.a[i][j] = (res.a[i][j] + a[i][k] * tmp.a[k][j] % mod) % mod;
32         return res;
33     }
34     Matrix operator +(Matrix tmp){
35         for(int i = 0;i < sz;++i)
36             for(int j = 0;j < sz;++j)
37                 a[i][j] += tmp.a[i][j];
38     }
39     friend Matrix fpow(Matrix a,ll b){
40         Matrix res;
41         while(b){
42             if(b & 1) res = res * a;
43             a = a * a;
44             b >>= 1;
45         }
46         return res;
47     }
48     ll get_fib(){
49         Matrix tmp;
50         tmp.init();
51         for(int i = 0;i < sz;++i)
52             tmp.a[i][0] = 1;
53         Matrix ans = *this;
54         ans = ans * tmp;
55         return ans.a[0][0];
56     }
57 };

```

4.20 类欧几里得

- 求解 $f(a,b,c,n) = \sum_{i=0}^n \lfloor \frac{a*i+b}{c} \rfloor$

```

1 ll f(ll a,ll b,ll c,ll n)
2 {
3     if(a == 0)
4         return (b / c) * (n + 1) % mod;
5     if(a >= c)
6         return (n * (n + 1) / 2 % mod * (a / c) % mod + f(a % c,b,c,n)) % mod;
7     if(b >= c)

```

```

8     return ((b / c) * (n + 1) % mod + f(a, b % c, c, n)) % mod;
9     //ll m = (a * n + b) / c % mod;
10    ll m = (n / c * a + (n % c * a + b) / c);
11    return ((m % mod) * (n % mod) % mod - f(c, c - b - 1, a, m - 1) + mod) % mod;
12 }

```

4.21 牛顿迭代

```

1  #define f(x) () //原函数
2  #define f2(x) () //导函数
3  double Newton(double x){
4      for(int i = 0; i < 100; ++i){
5          x = x - f(x) / f2(x); //牛顿迭代法
6          if(fabs(f(x)) < eps) return x;
7      }
8      return -1;
9  }

```

4.22 母函数

- n 个人分成任意组, 每组人数必须为素数, 有多少种分法, $n < 150$;

```

1  void solve()
2  {
3      memset(a, 0, sizeof a);
4      for(int i = 0; i <= 160; i += 2)
5          a[i] = 1;
6      memset(b, 0, sizeof b);
7      for(int i = 1; pri[i] <= 150; ++i){
8          for(int j = 0; j <= 150; ++j)
9              for(int k = 0; k + j <= 150; k += pri[i])
10                 b[k + j] += a[j];
11          for(int j = 0; j <= 150; ++j){
12              a[j] = b[j];
13              b[j] = 0;
14          }
15      }
16  }

```

4.22.1 莫比乌斯反演

- 求有多少个数对 (x, y) , 满足 $a \leq x \leq b, c \leq y \leq d$, 且 $\gcd(x, y) = 1$

```

1 void Mobius()
2 {
3     cnt = 0;
4     mu[1] = 1; sum[1] = 1;
5     for(int i = 2; i < maxn; ++i){
6         if(!vis[i]){
7             prime[++cnt] = i;
8             mu[i] = -1;
9         }
10        for(int j = 1; j <= cnt; ++j){
11            if(i * prime[j] >= maxn) break;
12            vis[i * prime[j]] = true;
13            if(i % prime[j] == 0){
14                mu[i * prime[j]] = 0;
15                break;
16            }
17            mu[i * prime[j]] = -mu[i];
18        }
19        sum[i] = sum[i-1] + mu[i];
20    }
21 }
22 ll solve(int b, int d)
23 {
24     if(b > d) swap(b, d);
25     ll ans = 0, pos;
26     for(int i = 1; i <= b; i = pos+1){
27         pos = min(b / (b / i), d / (d / i));
28         ans += 1LL * (b / i) * (d / i) * (sum[pos] - sum[i - 1]);
29     }
30     return ans;
31 }

```

4.23 斐波那契广义循环节

- $f(n) = af(n-1) + bf(n-2) \quad f(1) = c \quad f(2) = d$

```

1 const int N = 2;
2 const ll mod = 1000000007;
3
4 ll fac[2][505];
5 int cnt, ct;
6
7 ll pri[6] = {2, 3, 7, 109, 167, 500000003};
8 ll num[6] = {4, 2, 1, 2, 1, 1};
9

```

```
10 struct Matrix
11 {
12     ll m[N][N];
13 } ;
14
15 Matrix A;
16 Matrix I = {1, 0, 0, 1};
17
18 Matrix multi(Matrix a, Matrix b)
19 {
20     Matrix c;
21     for(int i=0; i<N; i++)
22     {
23         for(int j=0; j<N; j++)
24         {
25             c.m[i][j] =0;
26             for(int k=0; k<N; k++)
27             {
28                 c.m[i][j] += a.m[i][k] * b.m[k][j];
29                 c.m[i][j] %= mod;
30             }
31         }
32     }
33     return c;
34 }
35
36 Matrix power(Matrix A, ll n)
37 {
38     Matrix ans = I, p = A;
39     while(n)
40     {
41         if(n & 1)
42         {
43             ans = multi(ans, p);
44             n--;
45         }
46         n >>= 1;
47         p = multi(p, p);
48     }
49     return ans;
50 }
51
52 ll quick_mod(ll a, ll b)
53 {
54     ll ans = 1;
55     a %= mod;
```

```
56 while(b)
57 {
58     if(b & 1)
59     {
60         ans = ans * a % mod;
61         b--;
62     }
63     b >>= 1;
64     a = a * a % mod;
65 }
66 return ans;
67 }
68
69 ll Legendre(ll a,ll p)
70 {
71     ll t = quick_mod(a,(p-1)>>1);
72     if(t == 1) return 1;
73     return -1;
74 }
75
76 void dfs(int dept,ll product = 1)
77 {
78     if(dept == cnt)
79     {
80         fac[1][ct++] = product;
81         return;
82     }
83     for(int i=0; i<=num[dept]; i++)
84     {
85         dfs(dept+1,product);
86         product *= pri[dept];
87     }
88 }
89
90 bool OK(Matrix A,ll n)
91 {
92     Matrix ans = power(A,n);
93     return ans.m[0][0] == 1 && ans.m[0][1] == 0 &&
94         ans.m[1][0] == 0 && ans.m[1][1] == 1;
95 }
96
97 int main()
98 {
99     fac[0][0] = 1;
100    fac[0][1] = 2;
101    fac[0][2] = 500000003;
```

```

102 fac[0][3] = 1000000006;
103 ll a,b,c,d;
104 while(cin>>a>>b>>c>>d)
105 {
106     ll t = a * a + 4 * b;
107     A.m[0][0] = a;
108     A.m[0][1] = b;
109     A.m[1][0] = 1;
110     A.m[1][1] = 0;
111     if(Legendre(t,mod) == 1)
112     {
113         for(int i=0; i<4; i++)
114         {
115             if(OK(A,fac[0][i]))
116             {
117                 cout<<fac[0][i]<<endl;
118                 break;
119             }
120         }
121     }
122     else
123     {
124         ct = 0;
125         cnt = 6;
126         dfs(0,1);
127         sort(fac[1],fac[1]+ct);
128         for(int i=0;i<ct;i++)
129         {
130             if(OK(A,fac[1][i]))
131             {
132                 cout<<fac[1][i]<<endl;
133                 break;
134             }
135         }
136     }
137 }
138 return 0;
139 }

```

4.24 无名小定理

- 1. 设 $x = \gcd(\sum_{i=1}^{n-1} C(n, i))$
- 当 n 是素数是 $x = n$ (根据 1.)
- 当 n 有多个素因子 $x = 1$ (根据 1.)
- n 只有一个素因子, 答案为该素因子 (根据 1.)

- 当 $a > b$, $\gcd(a, b) = 1$, 那么 $\gcd(a^m - b^m, a^n - b^n) = a^{\gcd(m, n)} - b^{\gcd(m, n)}$
- $(n+1) * lcm(\sum_{i=0}^n C(n, i)) = lcm(\sum_{i=1}^{n+1} i)$
- 判断组合数 $C(n, m)$ 的奇偶性, 当 $n \& m == m$ 为奇数, 反之就是偶数.
- 三角形求圆半径: 边长为 a, b, c 的三角形面积为 S , 则外接圆半径为 $a * b * c / (4S)$, 内切圆半径为 $2S / (a + b + c)$

4.25 自然常数

$$e = 2.7182818284590452353602874713526624$$

4.26 欧拉常数

$$C = 0.57721566490153286060651209$$

4.27 错排公式

$$D_1 = 0$$

$$D_2 = 1$$

$$D_n = (n-1)(D_{n-1} + D_{n-2})$$

4.28 伯努利数

$$\sum_{i=0}^n F_i = 1, F_n = \sum_{i=0}^{n-1} C(n, i) F_i$$

4.29 自然幂数和

$$\sum_{i=1}^n i^k = \frac{1}{k+1} \sum_{i=1}^{k+1} C(k+1, i) * F_{k+1-i} * (n+1)^i$$

4.30 Catalan

$$1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, \dots$$

$$h(n) = h(n-1) * (4n-2) / (n+1), h(0) = 1, h(n) = C(2n, n) / (n+1) (n=0, 1, 2, \dots)$$

$$h(n) = c(2n, n) - c(2n, n-1) (n=0, 1, 2, \dots)$$

- 将 $n+2$ 边形沿弦切割成 n 个三角形的不同切割数
- $n+1$ 个数相乘, 给每两个元素加上括号的不同方法数
- n 个节点的不同形状的二叉树数
- 从 $n * n$ 方格的左上角移动到右下角不升路径数

4.31 pick 定理

计算点阵中顶点在格点上的多边形面积公式, 该公式可以表示为: $2 * S = 2 * a + b - 2$, 其中 a 表示多边形内部的点数, b 表示多边形边界上的点数, S 表示多边形的面积.

4.32 正多边形外接圆半径

$$R = \frac{a}{2 \sin(\frac{\pi}{n})}$$

4.33 Fibonacci

- $-F(n+m) = F(n+1)F(m) + F(n)F(m-1)$
- $F(n)^2 = (-1)^{n+1} + F(n-1)F(n+1)$
- 前 n 项有 $\lceil \frac{2n}{3} \rceil$ 个奇数项
- 如果 $\text{fib}(k)$ 能被 x 整除, 则 $\text{fib}(k*i)$ 都可以被 x 整除。
- $f(0)+f(1)+f(2)+\dots+f(n)=f(n+2)-1$
- $f(1)+f(3)+f(5)+\dots+f(2n-1)=f(2n)$
- $f(2)+f(4)+f(6)+\dots+f(2n)=f(2n+1)-1$
- $[f(0)]^2+[f(1)]^2+\dots+[f(n)]^2=f(n)*f(n+1)$
- $f(0)-f(1)+f(2)-\dots+(-1)^n$
- $f(n)=(-1)^n*[f(n+1)-f(n)]+1$
- $f(n+m)=f(n+1)*f(m)+f(n)*f(m-1)$
- $[f(n)]^2=(-1)^{n-1}+f(n-1)*f(n+1)*f(2n-1)=[f(n)]^2-[f(n-2)]^2$
- $3f(n)=f(n+2)+f(n-2)$
- $f(2n-2m-2)[f(2n)+f(2n+2)]=f(2m+2)+f(4n-2m)$ [$n>m>=-1$, 且 $n\geq 1$]
- $F_n = \frac{1}{\sqrt{5}} * (\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}} (\frac{1-\sqrt{5}}{2})^n$

5 计算几何

5.1 处理平面内所有直线围成的所有多边形

```

1  const int MAXN=1e6+10;
2  const double eps=1e-8;
3  const double pi=acos(-1.0);
4  const ll INF=0x3f3f3f3f3f3f3f3f;
5
6  inline int dcmp(double x){
7      if(fabs(x)<eps) return 0;
8      return (x>0? 1: -1);
9  }
10
11 inline double sqr(double x){ return x*x; }
12
13 struct Point{
14     double x,y;
15     Point(){ x=0,y=0; }
16     Point(double _x,double _y):x(_x),y(_y){}
17     void input(){ scanf("%lf%lf",&x,&y); }
18     void output(){ printf("%.2f %.2f\n",x,y); }
19     friend istream &operator >>(istream &os,Point &b){
20         os>>b.x>>b.y;
21         return os;
22     }
23     friend ostream &operator <<(ostream &os,Point &b){

```

```

24         os<<b.x<< ' ' <<b.y;
25         return os;
26     }
27     bool operator ==(const Point &b)const{
28         return (dcmp(x-b.x)==0&&dcmp(y-b.y)==0);
29     }
30     bool operator !=(const Point &b)const{
31         return !((dcmp(x-b.x)==0&&dcmp(y-b.y)==0));
32     }
33     bool operator <(const Point &b)const{
34         return (dcmp(x-b.x)==0? dcmp(y-b.y)<0 : x<b.x);
35     }
36     double operator ^(const Point &b)const{ //叉积
37         return x*b.y-y*b.x;
38     }
39     double operator *(const Point &b)const{ //点积
40         return x*b.x+y*b.y;
41     }
42     Point operator +(const Point &b)const{
43         return Point(x+b.x,y+b.y);
44     }
45     Point operator -(const Point &b)const{
46         return Point(x-b.x,y-b.y);
47     }
48     Point operator *(double a){
49         return Point(x*a,y*a);
50     }
51     Point operator /(double a){
52         return Point(x/a,y/a);
53     }
54     double len2(){ //长度平方
55         return sqr(x)+sqr(y);
56     }
57     double len(){ //长度
58         return sqrt(len2());
59     }
60     double polar(){ //向量的极角
61         return atan2(y,x); //返回与x轴正向夹角(-pi~pi]
62     }
63     Point change_len(double r){ //转化为长度为r的向量
64         double l=len();
65         if(dcmp(l)==0) return *this; //零向量
66         return Point(x*r/l,y*r/l);
67     }
68     Point rotate_left(){ //逆时针旋转90度
69         return Point(-y,x);

```

```

70     }
71     Point rotate_right(){ //顺时针旋转90度
72         return Point(y,-x);
73     }
74     Point rotate(Point p,double ang){ //绕点p逆时针旋转ang度
75         Point v=(*this)-p;
76         double c=cos(ang),s=sin(ang);
77         return Point(p.x+v.x*c-v.y*s,p.y+v.x*s+v.y*c);
78     }
79     Point normal(){ //单位化, 逆时针旋转90°
80         return Point(-y/len(),x/len());
81     }
82 };
83
84 inline double cross(Point a,Point b){ //叉积
85     return a.x*b.y-a.y*b.x;
86 }
87
88 inline double dot(Point a,Point b){ //点积
89     return a.x*b.x+a.y*b.y;
90 }
91
92
93 double rad(Point a,Point b){ //两个向量的夹角
94     return fabs(atan2(fabs(cross(a,b)),dot(a,b)));
95 }
96
97 bool is_parallel(Point a,Point b){ //判断向量是否平行
98     double p=rad(a,b);
99     return dcmp(p)==0||dcmp(p-pi)==0;
100 }
101
102 struct Line{
103     Point s,e;
104     Line(){ }
105     Line(Point _s,Point _e):s(_s),e(_e){} //两点确定直线
106     Line(Point p,double ang){ //一个点和斜率(弧度制)确定直线
107         s=p;
108         if(dcmp(ang-pi/2)==0){
109             e=s+Point(0,1);
110         }
111         else{
112             e=s+Point(1,tan(ang));
113         }
114     }
115     Line(double a,double b,double c){ //ax+by+c=0

```

```

116         if(dcmp(a)==0){
117             s=Point(0,-c/b);
118             e=Point(1,-c/b);
119         }
120         else if(dcmp(b)==0){
121             s=Point(-c/a,0);
122             e=Point(-c/a,1);
123         }
124         else{
125             s=Point(0,-c/b);
126             e=Point(1,(-c-a)/b);
127         }
128     }
129     void input(){
130         s.input();
131         e.input();
132     }
133     void adjust(){
134         if(e<s) swap(e,s);
135     }
136     double polar(){ //极角
137         return atan2(e.y-s.y,e.x-s.x); //返回与x轴正向夹角(-pi~pi]
138     }
139     double angle(){ //倾斜角
140         double k=atan2(e.y-s.y,e.x-s.x);
141         if(dcmp(k)<0) k+=pi;
142         if(dcmp(k-pi)==0) k-=pi;
143         return k;
144     }
145     Point operator &(const Line &b)const{ //求两直线交点
146         Point res=s;
147         double t=((s-b.s)^(b.s-b.e))/((s-e)^(b.s-b.e));
148         res.x+=(e.x-s.x)*t;
149         res.y+=(e.y-s.y)*t;
150         return res;
151     }
152 };
153
154 double polygon_area(vector<Point> p){ //多边形的有向面积，加上绝对值就是面积 正值表示输入点按照逆时针
    否则为顺时针
155     int n=p.size(); double area=0;
156     for(int i=1;i<n-1;i++) area+=cross(p[i]-p[0],p[i+1]-p[0]);
157     return fabs(area/2);
158 }
159
160 struct PSLG{ //平面直线图 处理平面内所有直线围成的所有多边形 传入直线交点之间的每条线段

```

```

161 struct Edge{
162     int from,to;
163     double ang;
164     Edge(){ ang=from=to=0; }
165     Edge(int s,int t,double a){ from=s,to=t,ang=a; }
166 };
167 int n,m,face_cnt; //平面个数 包括外面最大的多边形
168 double area[MAXN]; //每个多边形面积
169 Point point[MAXN]; //平面内所有的点
170 vector<Edge>edge;
171 vector<int>G[MAXN];
172 vector<vector<Point> >face;
173 int vis[2*MAXN],left[2*MAXN],pre[2*MAXN]; //left表示这条边的左侧属于哪个面
174 void Init(){
175     face.clear();
176     edge.clear();
177     for(int i=0;i<n;i++) G[i].clear();
178     n=m=0;
179 }
180 PSLG(){ Init(); }
181 void AddEdge(int from, int to){ //需要建立反向边帮助寻找下一条边
182     edge.pb(Edge(from,to,(point[to]-point[from]).polar()));
183     edge.pb(Edge(to,from,(point[from]-point[to]).polar()));
184     m=edge.size();
185     G[from].pb(m-2);
186     G[to].pb(m-1);
187 }
188 void Build(){
189     for(int u=0;u<n;u++){
190         int d=G[u].size();
191         for(int i=0;i<d;i++){
192             for(int j=i+1;j<d;j++){
193                 if(edge[G[u][i]].ang>edge[G[u][j]].ang)
194                     swap(G[u][i],G[u][j]);
195                 for(int i=0;i<d;i++) pre[G[u][(i+1)%d]]=G[u][i]; //从u出发的i条边顺时针旋转的第一条边是pre[i]
196             }
197         }
198         face_cnt=0; memset(vis,0,sizeof(vis));
199         for(int u=0;u<n;u++){
200             for(int i=0;i<G[u].size();i++){
201                 int e=G[u][i];
202                 if(!vis[e]){
203                     face_cnt++;
204                     vector<Point> polygon;
205                     while(1){
206                         vis[e]=1;
207                         left[e]=face_cnt;

```

```

207         int from=edge[e].from;
208         polygon.pb(point[from]);
209         e=pre[e^1]; //逆时针旋转最多的一条边即为顺时针转动的第一条边
210         if(e==G[u][i]) break;
211     }
212     face.pb(polygon);
213 }
214 }
215 }
216 for(int i=0;i<face_cnt;i++) area[i]=polygon_area(face[i]);
217 }
218 vector<pair<double,int> >tmp[MAXN];
219 void Insert(Line *line,int m){
220     for(int i=0;i<m;i++){
221         for(int j=i+1;j<m;j++){
222             if(!is_parallel(line[i].e-line[i].s,line[j].e-line[j].s)){
223                 Point inter=line[i]&line[j];
224                 point[n++]=inter;
225                 tmp[i].pb({dot(inter-line[i].s,line[i].e-line[i].s),n-1});
226                 tmp[j].pb({dot(inter-line[j].s,line[j].e-line[j].s),n-1});
227             }
228         }
229         sort(tmp[i].begin(),tmp[i].end());
230         for(int j=1;j<tmp[i].size();j++) AddEdge(tmp[i][j-1].se,tmp[i][j].se);
231     }
232     Build();
233 }
234 }pslg;
235
236 Line line[MAXN];
237
238 int main(void){
239     int n; scanf("%d",&n);
240     for(int i=0;i<n;i++) line[i].input();
241     pslg.Insert(line,n);
242     sort(pslg.area,pslg.area+pslg.face_cnt);
243     printf("%d %.6f %.6f\n",pslg.face_cnt-1,pslg.area[pslg.face_cnt-2],pslg.area[0]);
244     int q; scanf("%d",&q);
245     while(q--){
246         int p; scanf("%d",&p);
247         if(p>=pslg.face_cnt) puts("Invalid question");
248         else printf("%.6f\n",pslg.area[pslg.face_cnt-p-1]);
249     }
250     return 0;
251 }

```

6 字符串

6.1 kmp

- border

```

1 void get_fail(int f[],char s[])
2 {
3     int j = f[0] = 0;
4     int n = strlen(s);
5     rep(i,1,n)
6     {
7         while(j && s[i] != s[j]) j = f[j-1];
8         f[i] = j += s[i] == s[j];
9     }
10 }
```

- kmp

```

1 void kmp(int f[],char p[],char s[])
2 {
3     int n = strlen(s);
4     int m = strlen(p);
5     int j = 0;
6     rep(i,0,n)
7     {
8         while(j && s[i] != p[j]) j = f[j-1];
9         if(s[i] == p[j]) j++;
10        if(j == m)
11        {
12            cout << i - j + 2 << endl;
13            j = f[j-1];
14        }
15    }
16 }
```

6.2 SA

```

1 const int N=4e5+100;
2 const int maxn = 2e5+100;
3 const int inf=1e9+9;
4
5 namespace SA {
6     char s[N];
7     int sa[N],x[N],y[N],hep[N],height[N],n,m;
```



```

8 void init()
9 {
10     n = 0;
11 }
12 void add(char c)
13 {
14     // c -= 'a';
15     n++;
16     s[n]=c;
17 }
18 void Sort() {
19     for(int i=0;i<=m;++i) hep[i]=0;
20     for(int i=1;i<=n;++i) ++hep[x[i]];
21     for(int i=1;i<=m;++i) hep[i]+=hep[i-1];
22     for(int i=n;i>=1;--i) sa[hep[x[y[i]]]--]=y[i];
23 }
24 void Pre_sa() {
25     for(int i=1;i<=n;++i) x[i]=s[i],y[i]=i;
26     m=223;Sort();
27     for(int w=1,p=0;m=p,p<n;w<=1) {
28         p=0;
29         for(int i=1;i<=w;++i) y[++p]=n-w+i;
30         for(int i=1;i<=n;++i) if(sa[i]>w) y[++p]=sa[i]-w;
31         Sort(),swap(x,y),x[sa[1]]=p=1;
32         for(int i=2;i<=n;++i)
33             x[sa[i]]=(y[sa[i]]==y[sa[i-1]]&&y[sa[i]+w]==y[sa[i-1]+w])?p:++p;
34     }return;
35 }
36 ll Pre_height() {
37     for(int i=1;i<=n;++i) x[sa[i]]=i;
38     int k=0,res=0;
39     for(int i=1;i<=n;++i) {
40         k-=k>0;
41         int j=sa[x[i]-1];
42         while(i+k<=n&&j+k<=n&&s[i+k]==s[j+k]) ++k;
43         height[x[i]]=k,res+=k;
44     }return res; //直接返回height数组的和
45 }
46 ll solve()
47 {
48     /**
49     给你一个长为N的字符串，求不同的子串的个数？
50     对于一个后缀sa[i]，它产生了n-sa[i]个前缀，减去height[i]个相同的前缀(与前一个比较)，
51     则产生了n-sa[i]-height[i]个子串。累加后即结果。
52     */
53     ll ans = 0;

```

```
54     for (int i = 1; i <= n; i++)
55     {
56         ans += n + 1 - sa[i] - height[i];
57     }
58     return ans;
59 }
60 ll gao()
61 {
62     Pre_sa();
63     Pre_height();
64     return solve();
65 }
66 }
```

6.3 回文树 1

```
1 struct Pal
2 {
3     int ch[maxn][26], f[maxn], len[maxn], s[maxn];
4     int cnt[maxn];
5     int num[maxn];
6     int last, sz, n;
7
8     int newnode(int x)
9     {
10         memset(ch[sz], 0, sizeof(ch[sz]));
11         cnt[sz] = num[sz] = 0, len[sz] = x;
12         return sz++;
13     }
14     void init()
15     {
16         sz = 0;
17         newnode(0), newnode(-1);
18         last = n = 0, s[0] = -1, f[0] = 1;
19     }
20
21     int get_fail(int u)
22     {
23         while(s[n - len[u] - 1] != s[n])
24             u = f[u];
25         return u;
26     }
27
28     void add(int c)
29     {
```

```

30     c -= 'a';
31     s[++n] = c;
32     int u = get_fail(last);
33     if(!ch[u][c])
34     {
35         int np = newnode(len[u] + 2);
36         f[np] = ch[get_fail(f[u])][c];
37         num[np] = num[f[np]] + 1;
38         ch[u][c] = np;
39     }
40     last = ch[u][c];
41     cnt[last]++;
42 }
43
44 void count()
45 {
46     for(int i = sz - 1; ~i; i--)
47         cnt[f[i]] += cnt[i];
48 }
49 } pa;

```

6.4 回文树 2

```

1 struct Palindromic_Tree {
2     int son[N][26]; //转移边
3     int fail[N]; //fail 指针
4     int cnt[N]; //当前节点表示的回文串在原串中出现了多少次
5     int num[N]; //当前节点 fail 可以向前跳多少次
6     int len[N]; //当前节点表示的回文串的长度
7     int S[N]; //插入的字符串
8     int last; //最后一次访问到的节点，类似 SAM
9     int n; //插入的字符串长度
10    long long p; //自动机的总状态数
11
12    int newnode(int l) {
13        memset(son[p], 0, sizeof(son[p]));
14        cnt[p] = 0;
15        num[p] = 0;
16        len[p] = l;
17        return p++;
18    }
19
20    void init() {
21        p = 0;
22        newnode(0);

```

```

23     newnode(-1);
24     last = 0;
25     n = 0;
26     S[n] = -1;
27     fail[0] = 1;
28 }
29
30 int get_fail(int x) {
31     while (S[n - len[x] - 1] != S[n]) x = fail[x];
32     return x;
33 }
34
35 void add(int c) {
36     c -= 'a';
37     S[++n] = c;
38     int cur = get_fail(last); //通过上一次访问的位置去扩展
39     if (!son[cur][c]) { //如果没有对应的节点添加一个新节点
40         int now = newnode(len[cur] + 2);
41         fail[now] = son[get_fail(fail[cur])][c]; //通过当前节点的 fail 去扩展出新的 fail
42         son[cur][c] = now;
43         num[now] = num[fail[now]] + 1; //记录 fail 跳多少次
44     }
45     last = son[cur][c];
46     cnt[last]++; //表示当前节点访问了一次
47 }
48 void count() {
49     //如果某个节点出现一次，那么他的 fail 也一定会出现一次，并且在插入的时候没有计数
50     for (int i = p - 1; i >= 0; i--) cnt[fail[i]] += cnt[i];
51 }
52 } AUT;

```

7 杂项

7.1 退火

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const int maxn = 1e5 + 10;
6  const double eps = 1e-8;
7  const double delta = 0.98;
8  const double inf = 1e18;
9
10 struct Point { double x, y; } p[maxn];

```

```

11
12 double dis(Point A, Point B) { return sqrt((A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y)); };
13
14 double Simulate_Annea(int n)
15 {
16     Point S;
17     S.x = S.y = 0;
18     double t = 1000;
19     double res = inf;
20     while(t > eps)
21     {
22         int k = 0;
23         for(int i = 0; i < n; i++) if(dis(S, p[i]) > dis(S, p[k])) k = i;
24         double d = dis(S, p[k]);
25         res = min(res, d);
26         S.x += (p[k].x - S.x) / d * t;
27         S.y += (p[k].y - S.y) / d * t;
28         t *= delta;
29     }
30     return res;
31 }
32
33 int main()
34 {
35     int n;
36     scanf("%d", &n);
37     for(int i = 0; i < n; i++) scanf("%lf%lf", &p[i].x, &p[i].y);
38     printf("%.3f\n", Simulate_Annea(n));
39     return 0;
40 }

```

7.2 博弈

7.2.1 Bash

只有一堆 n 个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取 m 个。最后取光者得胜。考虑 $n = k*(1+m)+r$ 即可 $r = 0$ ，先手必败，无论先手怎么取（设取的是 a ），后手都能凑出 $1+m-a$

7.2.2 Wythoff

有两堆火柴棍，每次可以从某一堆取至少 1 根火柴棍（无上限），或者从两堆取相同的火柴棍数。最后取完的是胜利者。考虑状态 (a,b) $a[i] = i*(1+\sqrt{5})/2$ $b[i] = a[i]+i$ ；因为有浮点数参与运算，范围过大要考虑手写高精度

7.2.3 Fibonacci's Game

有一堆个数为 n 的石子，游戏双方轮流取石子，满足：1) 先手不能在第一次把所有的石子取完；2) 之后每次可以取的石子数介于 1 到对手刚取的石子数的 2 倍之间（包含 1 和对手刚取的石子数的 2 倍）。先手胜当且

仅当 n 不是 Fibonacci 数。

7.2.4 staircase nim

每层有若干个石子，每次可以选择任意层的任意个石子将其移动到该层的下一层。最后不能操作的人输把所有奇数阶梯看成 N 堆石子做 nim。把石子从奇数堆移动到偶数堆可以理解为拿走石子，就相当于几个奇数堆的石子在做 Nim。

7.2.5 anti-nim

正常的 nim 游戏是取走最后一颗的人获胜，而反 nim 游戏是取走最后一颗的人输。一个状态为必胜态，当且仅当：1) 所有堆的石子个数为 1，且 NIM_sum (异或和) $\neq 0$ 2) 至少有一堆的石子个数大于 1，且 $NIM_sum = 0$

7.2.6 约数博弈

游戏初始状态包含 $1-n$ ，这 n 个正整数。甲乙两个人轮流玩这个游戏。每轮游戏中，游戏者任意选择一个还存在的数，然后删掉它和它所有的约数。第一个删掉所有数的人获胜。先手必胜反证法 - 考虑一个新的规则“不准写数字 1”。依然先手必胜证明：如果在新规则下后写者必胜，则原游戏中的先写者写下数字 1，然后他就变成了新规则下的后写者。

7.2.7 约数和倍数博弈

游戏初始状态包含 $2-n$ ，这 $n-1$ 个正整数。甲乙两个人轮流玩这个游戏，每轮游戏中，游戏者任意选择一个还存在的数，然后删掉它和它所有的约数、倍数。第一个删掉所有数的人获胜。先手必败

7.2.8 Chomp 博弈

有一个 $n * m$ 的棋盘，每次可以取走一个方格并拿掉它右边和上面的所有方格。拿掉左下角的格子 (1,1) 者输。- 除了 $1*1$ 的棋盘，先手必败 - 对于其他大小的棋盘，先手必胜。

7.2.9 树上删边游戏

给出一个有 N 个点的树，有一个点作为树的根节点。游戏者轮流从树中删去边，删去一条边后，不与根节点相连的部分将被移走。谁无法移动谁输。叶子节点的 SG 值为 0；中间节点的 SG 值为它的所有子节点的 SG 值加 1 后的异或和。HDU 3590 $N \leq 100$ 颗树，每棵树有 $m \leq 100$ 个点，双方每次可以选择一棵树的一条边删去，并且把不与根相连的边一并删去，不能操作者赢先手必胜当且仅当：1) 游戏的 SG 函数不为 0 且游戏中某个单一游戏的 SG 函数大于 1

2) 游戏的 SG 函数为 0 且游戏中没有单一游戏的 SG 函数大于 1

```

1 VI v[100005];
2 int getsg(int x,int pre)
3 {
4     int ans = 0;
5     rep(i,0,v[x].size())
6         if(v[x][i] != pre)
7             {
8                 ans ^= (1+getsg(v[x][i],x));
9             }

```

```

10     return ans;
11 }
12
13 int main()
14 {
15     int t,n;
16     while(scanf("%d",&t)!=EOF)
17     {
18         int sum = 0,ans = 0;
19         while(t--)
20         {
21             scanf("%d",&n);
22             rep(i,1,n+1)
23             {
24                 v[i].clear();
25             }
26             rep(i,1,n)
27             {
28                 int x,y;
29                 scanf("%d%d",&x,&y);
30                 v[x].push_back(y);
31                 v[y].push_back(x);
32             }
33             int s = getsg(1,-1);
34             if(s > 1)
35                 ans++;
36             sum ^= s;
37         }
38         if((sum == 0&&ans==0)|| (sum!=0&&ans>=1))
39         {
40             cout<<"PP"<<endl;
41         }
42         else
43         {
44             cout<<"QQ"<<endl;
45         }
46     }
47     return 0;
48 }

```

7.2.10 SG

```

1  define MAX 1005
2  /* 计算从1-n范围内的SG值。
3     Array(存储可以走的步数, Array[0]表示可以有多少种走法)

```

```

4   Array[] 需要从小到大排序 */
5   /*HDU1847博弈SG函数
6   1.可选步数为1-m的连续整数，直接取模即可，SG(x) = x % (m+1);
7   2.可选步数为任意步，SG(x) = x;
8   3.可选步数为一系列不连续的数，用GetSG(计算) */
9   int SG[MAX], hash[MAX];
10  void GetSG(int Array[], int n = MAX-1)
11  {
12      memset(SG, 0, sizeof(SG));
13      for(int i = 0; i <= n; ++i)
14      {
15          memset(hash, 0, sizeof(hash));
16          for(int j = 1; j <= Array[0]; ++j)
17          {
18              if(i < Array[j])
19                  break;
20              hash[SG[i - Array[j]]] = 1;
21          }
22          for(int j = 0; j <= n; ++j)
23              if(!hash[j])
24              {
25                  SG[i] = j;
26                  break;
27              }
28      }
29  }

```

S-Nim 游戏仅仅是限制了每一次从每一堆中选取的个数，依旧用 sg 函数计算即可。经典的 Nim 游戏中 $sg(x) = x$ ，所以结果就是每一堆的状态直接 xor 即可。S-Nim 游戏先计算每一堆的 sg 函数值，然后判断方法依旧是用 xor。

```

1   const int maxn = 10100;
2   const int N = 110;
3   int SG[maxn], f[N], s[maxn];
4
5   int k, n, m, t;
6
7   void getsg(int n, int nn)
8   {
9       SG[0] = 0;
10      int tt = 0;
11      for(int i = 1; i <= 10000; i++)
12      {
13          tt++;
14          //memset(s, 0, sizeof s);
15          for(int j = 0; f[j] <= i && j < nn; j++)

```



```

16     {
17         s[SG[i-f[j]]] = tt;
18     }
19     for(int j = 0;;j++)
20         if(s[j]!=tt)
21         {
22             SG[i] = j;
23             break;
24         }
25     }
26 }
27
28 int main()
29 {
30     while(scanf("%d",&k)!=EOF&&k)
31     {
32         for(int i =0;i<k;i++)
33             scanf("%d",&f[i]);
34         sort(f,f+k);
35         scanf("%d",&n);
36         gets(g(10000,k));
37         for(int i = 0;i<n;i++)
38         {
39             int ans = 0;
40             scanf("%d",&m);
41             for(int i = 0;i<m;i++)
42             {
43                 scanf("%d",&t);
44                 ans^=SG[t];
45             }
46             if(ans)
47                 cout<<"W";
48             else
49                 cout<<"L";
50         }
51
52         cout<<endl;
53     }
54 }

```

7.3 indiewar 的私人题

7.3.1 半圆概率

在一个圆内任取 n 个点，则这 n 个点出现在同一个半圆内的概率是多少转化为 $2n$ 条半径的古典概型问题：

$$P_n = 2n/(2^n) = n/2^{n-1}$$

7.3.2 百囚徒挑战

监狱决定给关押的 100 名囚徒一次特赦的机会，条件是囚徒通过一项挑战。

所有囚徒被编号为 1-100，对应他们编号的 100 个号码牌被打乱顺序放在了 100 个抽屉里。

每个囚徒需要从所有抽屉里打开至多半数 (50 个)，并从中找出对应自己编号的号码牌。

如果找到了则该名囚徒的任务成功。所有囚徒会依次单独进入挑战室完成任务，并且从第一个囚徒进入挑战室开始，直到所有囚徒结束挑战为止囚徒之间任何形式的交流都是禁止的。

当一名囚徒完成任务后，挑战室会被恢复为他进入之前的样子（号码牌当然也放回原来的抽屉里）。

在这 100 名囚徒中，任意一名囚徒的失败都会导致整个挑战失败，只有当所有囚徒全部成功完成任务时，他们才会统一得到特赦的机会。最后，在开始挑战之前，监狱给了所有囚徒一个月时间商量对策。那么，囚徒究竟有多大的几率得到释放？ $p(n) = 1 - \sum_{m=n+1}^{2n} \frac{1}{m}$

7.3.3 投针问题

假定有一块画着许多等距的平行线的地板，现往板上投放一根细针，求细针与平行线相交的概率。假设平行线间距为 a ，针长为 l 且满足 $l < a$ $p = \frac{2l}{a\pi}$

$$l \geq a \quad p = 1 - \frac{1}{\pi} \left(2 \arcsin \frac{a}{l} - \frac{2l}{a} + \frac{2}{a} \sqrt{l^2 - a^2} \right)$$

7.3.4 打怪兽

假设你打一下怪，怪掉总血量的 0% 到 100% 之间的一个随机数，(包括 [0,1] 区间内的所有实数诸如 0.5%) 那么，求你把怪打死的一共要打几次的数学期望。不妨设怪物血量为 1，你每次攻击输出 [公式] 间均匀的随机伤害。假设 $f(x)$ 为打死血量为 x 的怪物所用的期望次数。这个比较反常识，虽然我们是要求 $f(1)$ ，但是，我们却选择求看起来似乎更难的 $f(x)$ $f(x) = e^x$

7.3.5 互素

随机地取两个正整数，他们互素的概率为 $p = \frac{6}{\pi^2}$

7.3.6 半球

在一个球内任取 n 个点，则这 n 个点落在同一个半球内的概率 $p = \frac{n^2 - n + 2}{2^n}$

$$\text{圆 } p = \frac{2n}{2^n}$$

8 DP

8.1 背包

8.1.1 01 背包

- normal $f_i = \max(f_i, f_{i-1} + w)$ 需要按照 i 从大到小的顺序更新，确保每个物品只会选一次

```

1  memset(dp, 0xcf, sizeof dp);
2  dp[0] = 0;
3  rep(i, 0, n)
4  {
5      cin >> v[i] >> w[i];

```

```

6   per(j,v[i],m+1)
7   {
8       dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
9   }
10 }

```

- 计数不超过 m 的方案数 $f_{i+} = f_{i v}$
- 删除加入物品的顺序不影响结果，假设被删除的物品是最后一次加入的，那么倒过来还原即可。 $f_{i-} = f_{i v}$ 需要按照 i 从小到大的顺序更新。给定 a_i 和 b_i ，表示第 i 个商店有 a_i 个商品可以买，单价为 b_i 元，给出 m 个询问，问用 c 元在 $1 \sim r$ 商店买东西的方案数一种物品的背包可以看成 $\sum_{i=0}^a x^{ib} = \frac{1-x^{(a+1)b}}{1-x^b}$ ，所以可以先用 $(a+1)b$ 去做一个 01 背包 (系数为负)，再除以一个 x^b 的 (系数为负)01 背包。从生成函数来看， $\frac{1}{1-x^b} = \sum_{i=0}^{\infty} x^{bi}$ ，即做一遍完全背包就可以等效

然后对可逆背包的预处理，由于 $\frac{1-x^b}{1-x^{(a+1)b}} = (1-x^b) * \sum_{i=0}^{\infty} x^{i*(a+1)b}$ ，于是反过来对 x^b 做 01 背包，对 $(a+1)b$ 做完全背包就可以

```

1   int n,m;
2   int a[maxn*10],b[maxn*10],f[maxn*10][maxn],g[maxn*10][maxn];
3
4   void gao(int *dp,int w)
5   {
6       per(i,w,maxn)
7       {
8           dp[i] = (dp[i] - dp[i-w]+mod)%mod;
9       }
10  }
11
12  void gao2(int *dp,int w)
13  {
14      rep(i,w,maxn)
15      {
16          dp[i] = (dp[i] + dp[i-w]+mod)%mod;
17      }
18  }
19
20  int main(int argc, char const *argv[])
21  {
22      ios_base::sync_with_stdio(false), cin.tie(0);
23      // cout.tie(0);
24      int T,cas;
25      cin >> T;
26      cas = 1;
27      f[0][0] = 1;
28      rep(i,0,maxn)
29          g[0][i] = 1;
30      while(T--)

```

```

31 {
32     prr(cas++);
33     cin >> n >> m;
34     rep(i,1,n+1)
35     {
36         cin >> a[i] >> b[i];
37         a[i] = (a[i] + 1) * b[i];
38     }
39     rep(i,1,n+1)
40     {
41         memcpy(f[i],f[i-1],sizeof(f[i]));
42         memcpy(g[i],g[i-1],sizeof(g[i]));
43         gao(f[i],a[i]);gao2(f[i],b[i]);
44         gao(g[i],b[i]);gao2(g[i],a[i]);
45     }
46     int ans = 0;
47     rep(i,0,m)
48     {
49         int l,r,c;
50         cin >> l >> r >> c;
51         l = (l+ans)%n+1;
52         r = (r+ans)%n+1;
53         if(l > r)
54         {
55             swap(l,r);
56         }
57         ans = 0;
58         rep(i,0,c+1)
59         {
60             ans = (ans + 1ll*f[r][i]*g[l-1][c-i])%mod;
61         }
62         printf("%d\n",ans);
63     }
64 }
65 return 0;
66 }

```

8.1.2 完全背包

- normal 需要按照 i 从小到大的顺序更新，意为要么停止选，要么接着多选一个。

```

1 dp[0] = 0;
2 rep(i,0,n)
3 {
4     cin >> v >> w;
5     rep(j,v,m+1)

```

```

6     {
7         dp[j] = max(dp[j], dp[j-v]+w);
8     }
9 }
10 cout << dp[m] << endl;

```

- 计数 same
- 删除 same

8.1.3 多组背包

- 二进制拆分二进制拆分，将一个物品拆成 $O(\log k)$ 个 01 背包的物品。eg: $10 = 1 + 2 + 4 + 3$ ，可以表示 $1 - 10$ $O(nm \log(k))$
-

```

1  memset(dp,0,sizeof dp);
2      rep(i,0,n)
3      {
4          cin >> v[i] >> w[i] >> s[i];
5      }
6      int cnt = 0;
7      rep(i,0,n)
8      {
9          for(int j = 1; j < s[i]; j <= 1)
10             {
11                 ww[cnt] = j * w[i];
12                 vv[cnt] = j * v[i];
13                 s[i] -= j;
14                 cnt++;
15             }
16             if(s[i])
17             {
18                 ww[cnt] = s[i] * w[i];
19                 vv[cnt++] = s[i] * v[i];
20             }
21         }
22         rep(i,0,cnt)
23         {
24             per(j,vv[i],m+1)
25             {
26                 dp[j] = max(dp[j], dp[j-vv[i]] + ww[i]);
27             }
28         }
29         cout << dp[m] << endl;

```

- 单调队列按%v 的余数分组，每组滑窗求区间最大值 $O(nm)$, 但不见得比上面快

8.1.4 分组背包

n 个物品，每个物品只能选一个，体积为 v_i ，种类为 k_i 。求总体积恰好 m 的情况下能拿走物品种类数的最大值。将所有物品按 k 分组状态： $f_{i,j,k,s}$ 表示考虑前 i 组，这一组内考虑了前 j 个物品，总体积为 k ，第 i 组物品是否被选择的情况为 s 时，种类数的最大值。

8.1.5 树形依赖背包

以 1 为根的树上有 n 个节点，每个节点有一个物品，体积 v_i ，价值 w_i 。选了一个点就必须选它的父亲。求总体积不超过 m 的情况下能拿走物品总价值的最大值。

按照 DFS 的顺序进行 DP。往下搜的时候，强行将儿子选入背包中。往上回溯的时候，可以选择要这棵子树的 DP 值，或者不要。