ACM 模版整理

JLU_ 黑眼圈

2020 年 11 月 7 日

第一章　入门

1.1　比赛策略

1.1.1　热身赛准备

- 测试硬件（键盘、鼠标、显示器）
- 若使用 CodeBlocks，则 Settings → General Settings → Terminal to launch console programs 将方框里默认的终端改为 gnome-terminal -t $TITLE -x，在 Compiler Settings 里勾选上 -std=c++14
- 测试评测机的 bits/stdc++.h，输出格式的%lld %lf，匿名结构体，___gcd，assert
- 测试数组下标越界、除 0、递归深度、运行语句数
- 尝试各种错误类型，注意有无 MLE、PE、OLE，测试 CE 罚时，测试 WA 和 TLE 的先后顺序
- 测试行末空格是否被检查，如果可以则询问裁判
- 调戏裁判，尝试系统中的其他软件

```
int main1(int a, int b, int c, int d, int e, int f, int g) {
   static int tot;
   if (tot < 100000) return tot++, main1(a, b, c, d, e, f, g);
} // 测试栈深，可能和内存一样大，否则要决定是否要修改为手工栈等方式

void main2() {
   static int *list[MB/4]; // 例如720MB为180
   for(int i = 0; i < MB/4; i++) {
      list[i] = new int[1048576];
      for (int j = 0; j < 1048576; j++)
         list[i][j] = j;
   }
} // 测试内存超限，看是返回MLE还是RE（比较重要）

void main3() {
   const int MAX_RUN = 1e8;
   static int buf[1000][1000];
   for (int i = 0; i < MAX_RUN/1000/1000; i++)
      for (int j = 0; j < 1000; j++) for (int k = 0; k < 1000; k++)
         buf[j][k] = buf[rand()%1000][rand()%1000] * rand() * rand();
   printf("%d", f[999][999]);
} // 测试时间限制，自带大常数
```

1.1.2 战术研究

- 读新题的优先级高于一切，构造题不可开场做
- 读完题、交题前看一遍 Clarification
- 即使有 SPJ 的题目也要尽量和样例输出完全一致
- WA 时检查 INF 是否设小
- 每道题需要至少有两个人确认题意，上机前做法需要得到队友确认
- 带有猜想性质的算法应放在后面写
- 发现不会写但是过了一片应冲一发暴力
- 交完题目立马打印并让出机器
- 写题按 solution size 排序，写题超过半小时应考虑是否弃题
- 细节、公式等在上机前应在草稿纸上准备好，防止上机后越写越乱
- 检查所有东西是否清空和初始化
- 中后期题应考虑一人写题，另一人在一旁辅助，及时发现手误
- 对于取模的题，在输出前要再取模一次进行保险，检查负数可能
- 检查混淆变量的情况，如 $solve(n, m)$ 和 $solve(m, n)$
- 对于输出要舍入的，注意特判 $ans < eps$ 时设置 $ans = 0$ 以防 -0.00

1.1.3 打表尝试

- 直接尝试找规律
- 差分找规律，拉格朗日插值或 Berlekamp-Messay
- 找积性、相除
- 是否有循环节
- 凑量纲
- 猜想 $P(n)f(n) = Q(n)f(n-2) + R(n)f(n-1) + C$，其中二次多项式

1.1.4 对拍

```cpp
#include <bits/stdc++.h>
using namespace std;
mt19937 Rand(chrono::system_clock::now().time_since_epoch().count());

int main() {
   while (true) {
      static int kase; printf("Case #%d: \n", ++kase);
      system("./data > data.in"); system("cat data.in");
      system("./my < data.in > my.out");
      system("./brtf < data.in > std.out");
      if (system("diff my.out std.out"))
```

```
12          return !printf("FAILED\n");
13       printf("OK\n");
14    }
15 }
```

1.2 卡常

1.2.1 带缓冲区读入优化

```
1  struct fastIO {
2     char s[100000]; int it, len;
3     fastIO() { it = len = 0; }
4
5     inline char get() {
6        if (it < len) return s[it++];
7        it = 0; len = fread(s, 1, 100000, stdin);
8        if (len == 0) return EOF; else return s[it++];
9     }
10
11    bool notend() {
12       char c = get(); while(c == ' ' || c == '\n') c = get();
13       if (it > 0) it--;
14       return c != EOF;
15    }
16
17    inline lld readInt() {
18       lld r = 0; bool ng = 0; char c = get();
19       while (c != '-' && (c < '0' || c > '9')) c = get();
20       if (c == '-') ng = 1, c = get();
21       while (c >= '0' && c <= '9') r = r * 10 + c - '0', c = get();
22       return ng ? -r : r;
23    }
24 };
```

1.2.2 无 int128 时大模数乘法

```
1  inline lld mul(lld x, lld y, lld z) {
2     lld ans = (x * y - lld((long double)(x) * y / z) * z) % z;
3     if (ans < 0) ans += z; return ans;
4  }
5
6  inline lld add(lld x, lld y, lld z) {
7     return x + y - (x + y >= z ? z : 0);
8  }
```

1.2.3 Montgomery 大模数乘法

```cpp
using i64 = long long;
using u64 = unsigned long long;
using u128 = __uint128_t;

struct Mod64 {
    static u64 mod, inv, r2; u64 n_;
    Mod64(u64 n) : n_(init(n)) {}
    static u64 modulus() { return mod; }
    static u64 init(u64 w) { return reduce(u128(w) * r2); }
    Mod64 operator+(Mod64 rhs) const { return Mod64(*this) += rhs; }
    Mod64 operator*(Mod64 rhs) const { return Mod64(*this) *= rhs; }
    u64 get() const { return reduce(n_); }

    static void set_mod(u64 m) {
        mod = m; assert(mod & 1); inv = m;
        for(int i=0;i<5;++i) inv *= 2 - inv * m;
        r2 = -u128(m) % m;
    }

    static u64 reduce(u128 x) {
        u64 y = u64(x >> 64) - u64((u128(u64(x) * inv) * mod) >> 64);
        return i64(y) < 0 ? y + mod : y;
    }

    Mod64& operator+=(Mod64 rhs) {
        n_ += rhs.n_ - mod;
        if (i64(n_) < 0) n_ += mod;
        return *this;
    }

    Mod64& operator*=(Mod64 rhs) {
        n_ = reduce(u128(n_) * rhs.n_);
        return *this;
    }
};
```

1.2.4 神秘指令与循环展开

```cpp
#define _FORTIFY_SOURCE 0
#pragma GCC optimize("Ofast,no-stack-protector,-funroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")

#include <bits/stdc++.h>
using namespace std;
const int N = 2e5 + 5, mod = 998244353;
using i64 = long long;
```

```

int T, n, b[N];
i64 s[N], ans;

int main() {
   ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
   for (cin >> T; T; --T) {
      cin >> n;
      for (int i = 1; i <= n; ++i)
         cin >> b[i], --b[i];
      s[0] = 1;
      for (int i = 1; i <= n; ++i) {
         for (int j = 0; j + 16 <= b[i]; j += 16) {
            for (int k = 0; k < 16; ++k)
               s[j + k + 1] += s[j + k];
            s[j + 16] %= mod;
         }
         for (int j = b[i] - (b[i] & 15); j < b[i]; ++j)
            s[j + 1] += s[j];
         if (i % 16 == 0)
            for (int j = 0; j <= b[i]; ++j)
               s[j] %= mod;
      }
      for (int i = 0; i < n; ++i)
         ans = (ans + s[i]) % mod;
      if (ans < 0)
         ans += mod;
      cout << ans << endl;
      ans = 0;
      memset(s + 1, 0, sizeof(i64) * n);
   }
   cerr << clock() << endl;
}
```

1.3 位运算

1.3.1 GCC 内置函数

```
int __builtin_popcount(unsigned int x); // 求1的个数
int __builtin_clz(unsigned int x); // 求前缀0的个数
int __builtin_ctz(unsigned int x); // 求后缀0的个数
```

1.3.2 四毛子算法

在一些状态简单的时候（例如树上 DFS 序的差分是 -1 和 1），将 $O(\log n)$ 个值的内部状态暴力维护，边界状态单独处理，剩下 $O(\frac{n}{\log n})$ 块使用 RMQ 算法维护，就可以将 RMQ 的复杂度降为 $O(n) - O(1)$。所谓的非常规大小分块。

1.3.3 枚举子码

设 x 的二进制表示一个集合，则 x 的子集可以通过如下代码枚举。

```
for (int y = x; y; y = x & (y-1)) {
   B[x] += A[y];
}
```

其他相关资料参考子集反演。

1.4 GNU C++ 库其他数据结构

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/hash_policy.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/trie_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>
typedef pair<int,int> pii;
using namespace __gnu_pbds;

cc_hash_table<int,bool> h1; // 开链表
gp_hash_table<int,bool> h2; // 线性探查

tree<pii,null_type,less<pii>,rb_tree_tag,tree_order_statistics_node_update>
    rbt; // 红黑树
tree<int,null_type,less<int>,splay_tree_tag,tree_order_statistics_node_update>
     splay; // Splay

trie<string,null_type,trie_string_access_traits<>,pat_trie_tag,
    trie_prefix_search_node_update> tri; // 字典树

priority_queue<int,greater<int>,pairing_heap_tag> Q1; // 小根配对堆
priority_queue<int,less<int>,binomial_heap_tag> Q2; // 大根二项堆
```

第二章　常用算法

2.1　莫队算法

2.1.1　区间莫队

```
int ans[MAXM], c[MAXN], n, m, l, r, tot;
struct QR { int l, r, id; } q[MAXM];
inline void add(int x) { } // 将x处的答案加进统计结果tot
inline void del(int x) { } // 将x处的答案从tot中消除

void solve() {
   scanf("%d %d", &n, &m);
   int FK = sqrt(n);
   for (int i = 1; i <= n; i++)
      scanf("%d", &c[i]);
   for (int i = 1; i <= m; i++)
      scanf("%d %d", &q[i].l, &q[i].r), q[i].id = i;

   sort(q+1, q+1+m, [FK] (QR a, QR b) -> bool {
      return (a.r/FK)==(b.r/FK) ? a.l < b.l : a.r < b.r;
   });

   for (int i = 1; i <= m; i++) {
      while (l < q[i].l) del(l++);
      while (l > q[i].l) add(--l);
      while (r < q[i].r) add(++r);
      while (r > q[i].r) del(r--);
      ans[q[i].id] = tot;
   }

   for (int i = 1; i <= m; i++)
      printf("%d\n", ans[i]);
}
```

普通莫队　需要注意块与块之间是 $O(1)$ 转移的。按 $n^{1/2}$ 分块，时间复杂度 $O(n^{3/2})$。

科学莫队　需要注意块与块之间是 $O(1)$ 转移的。求曼哈顿距离最小生成树。

带修莫队　将修改时间加入到询问中，在排序时第三关键字排序时间，按 $n^{2/3}$ 分块，时间复杂度 $O(n^{5/3})$。

2.1.2 树上莫队

树上莫队 按括号序分块，在括号序上跑莫队。DFS 一棵树时，到 x 点在 DFS 序数组里存 x，离开时存 $-x$，然后对应相应的 *add* 和 *del* 即可。

2.2 DLX

```
const int MAXN = 20010, MAXR = 70;
int L[MAXN], R[MAXN], U[MAXN], D[MAXN], H[MAXN], mm;
int s[MAXN], sz, col[MAXN], row[MAXN], cnt, ans[MAXR], tot;
#define DLF(i,A,s) for (int i = A[s]; i != s; i = A[i])

void init(int n, int m) {
   for (int i = 0; i <= m; i++)
      L[i] = i-1, R[i] = i+1, U[i] = D[i] = i, s[i] = 0;
   for (int i = 0; i <= n; i++) H[i] = -1;
   L[0] = m, R[m] = 0, sz = m+1, cnt = tot = 0, mm = m;
}

void link(int r, int c) {
   U[sz] = c, D[sz] = D[c], U[D[c]] = sz, D[c] = sz;
   if (H[r] < 0) H[r] = L[sz] = R[sz] = sz;
   else L[sz] = H[r], R[sz] = R[H[r]], L[R[H[r]]] = sz, R[H[r]] = sz;
   s[c]++, col[sz] = c, row[sz] = r, sz++;
} // if (A[r][c]) link(r, c); 注意下标 1<=r<=n, 1<=c<=m
```

2.2.1 精确覆盖

给出一个 n 行 m 列的矩阵，每个位置为 0/1，求出一组行，使得其中每个列有且仅有一个 1。可以应用在求解数独（给 1-9 每个数字都加限制），求解八皇后问题的解等等。

```
void del(int c) {
   L[R[c]] = L[c], R[L[c]] = R[c];
   DLF(i, D, c) DLF(j, R, i) U[D[j]] = U[j], D[U[j]] = D[j], --s[col[j]];
}

void add(int c) {
   R[L[c]] = L[R[c]] = c;
   DLF(i, U, c) DLF(j, L, i) U[D[j]] = D[U[j]] = j, ++s[col[j]];
}

bool dfs(int k) {
   if (!R[0]) return cnt = k, true;
   int c = R[0]; DLF(i, R, 0) if (s[c] > s[i]) c = i; del(c);
   DLF(i, D, c) {
      DLF(j, R, i) del(col[j]);
      ans[k] = row[i]; if (dfs(k+1)) return true;
      DLF(j, L, i) add(col[j]);
```

```
    }
    add(c); return 0;
}
```

2.2.2 可重复覆盖

给出一个 n 行 m 列的矩阵，每个位置为 0/1，求出一组行，使得其中每个列至少有一个 1。可以用在寻找最小的一组覆盖等等。

```
void del(int c) {
    DLF(i, D, c) L[R[i]] = L[i], R[L[i]] = R[i];
}

void add(int c) {
    DLF(i, U, c) L[R[i]] = R[L[i]] = i;
}

int branch() {
    int res = 0; static bool vis[MAXR];
    memset(vis, 0, sizeof(vis));
    DLF(i, R, 0) if (!vis[i]) {
        res++, vis[i] = true;
        DLF(j, D, i) DLF(k, R, j) vis[col[k]] = true;
    }
    return res;
} // 估价函数

void dfs(int now) {
    if (R[0] == 0) tot = min(tot, now); // 搜出一组
    else if (now + branch() < tot) {
        int temp = 2100000000, c;
        DLF(i, R, 0) if (temp > s[i]) temp = s[i], c = i;
        DLF(i, D, c) {
            del(i); DLF(j, R, i) del(j);
            dfs(now+1);
            DLF(j, L, i) add(j); add(i);
        }
    }
}
```

2.3 CDQ 分治

```
#include<cstdio>
#include<algorithm>
const int N=100100;
const int K=200100;
using namespace std;
```

```
6  int n,k,ans[N],cnt[N],sc[K],res[N],t;
7  struct node{
8     int x,y,z,v,id;
9     bool operator <(node b)const{
10       if (x==b.x && y==b.y) return z<b.z;
11       else if (x==b.x) return y<b.y;
12       else return x<b.x;
13    }
14 }q[N],tp[N];
15 void r(int &x){char ch=getchar(); x=0; while (ch<'0' || ch>'9')ch=getchar();
      while (ch>='0' && ch<='9') x=x*10+ch-'0',ch=getchar();}
16 void ins(int x,int v){for (;x<=k;x+=x&(-x)) sc[x]+=v;}
17 int get(int x){int cnt=0; for (;x>=1;x-=x&(-x)) cnt+=sc[x]; return cnt;}
18 void work(int l,int r){
19    if (l==r) return;
20    int mid=(l+r)>>1,L=l,R=mid+1;
21    for (int i=l; i<=r; i++)
22       if (tp[i].id<=mid) ins(tp[i].y,tp[i].v); else ans[tp[i].id]+=get(tp[i].y
          );
23    for (int i=l; i<=r; i++)
24       if (tp[i].id<=mid) ins(tp[i].y,-tp[i].v),q[L++]=tp[i]; else q[R++]=tp[i
          ];
25    for (int i=l; i<=r; i++) tp[i]=q[i];
26    work(l,mid); work(mid+1,r);
27 }
28 int main(){
29    r(n); r(k);
30    for (int i=1; i<=n; i++) r(q[i].x),r(q[i].y),r(q[i].z);
31    sort(q+1,q+1+n);
32    for (int i=1; i<=n; i++) if (q[i].x!=q[i-1].x || q[i].y!=q[i-1].y ||q[i].z
         !=q[i-1].z) tp[++t]=q[i],swap(tp[t].x,tp[t].z),tp[t].id=t,tp[t].v=1;
33    else tp[t].v++,res[t]++;
34    sort(tp+1,tp+1+t);
35    work(1,t);
36    for (int i=1; i<=t; i++) cnt[ans[i]+res[i]]+=res[i]+1;
37    for (int i=0; i<n; i++) printf("%d\n",cnt[i]);
38    return 0;
39 }
```

第三章　动态规划

3.1　插头 DP

```
#include<cstdio>
typedef long long LL;
const int rx
   [14]={1,3,9,27,81,243,729,2187,6561,19683,59049,177147,531441,1594323};
LL dp[2][1594324],Ans,S;
int lst[2][1594324],n,m,tx,ty,p,Z,L,R;
char str[15][15];
int f(int v,int L){return v%rx[L+1]/rx[L];}
int c(int v,int L,int x){return v-f(v,L)*rx[L]+x*rx[L];}
void INS(int id,int v,LL s){dp[id][v]?dp[id][v]+=s:(lst[id][0]++,lst[id][lst[
   id][0]]=v,dp[id][v]=s);}
int FLIP(int v,int L){int t=f(v,L),cnt=1,i=L;for (i+=t==1?1:-1;;i+=t==1?1:-1){
   if(f(v,i)!=0)cnt+=f(v,i)==t?1:-1;if(cnt==0)break;}return c(v,i,t);}
int main(){
   scanf("%d%d",&n,&m);
   for (int i=1; i<=n; i++){
      scanf("%s",str[i]);
      for (int j=0; j<m; j++) str[i][j]=='.'?(tx=i,ty=j):0;
   } lst[p][0]=1; lst[p][1]=0; dp[p][0]=1;
   for (int i=1; i<=tx; i++)
      for (int j=0; j<m; j++){
         lst[p^1][0]=0; p^=1;
         for (int k=1; k<=lst[p^1][0]; dp[p^1][lst[p^1][k]]=0,k++){
            Z=lst[p^1][k]; S=dp[p^1][lst[p^1][k]];
            if (j==0 && f(Z,m)) continue; else j==0?Z+=Z<<1:0; L=f(Z,j);R=f(Z,
               j+1);
            if (str[i][j]=='*'){if (!L&&!R) INS(p,Z,S);continue;}
            if (!L && !R) INS(p,c(c(Z,j,1),j+1,2),S);
            if (L==2 && R==1) INS(p,c(c(Z,j,0),j+1,0),S);
            if (L==1 && R==1) INS(p,c(c(FLIP(Z,j+1),j,0),j+1,0),S);
            if (L==2 && R==2) INS(p,c(c(FLIP(Z,j),j,0),j+1,0),S);
            if (L && !R) INS(p,Z,S),INS(p,c(c(Z,j,0),j+1,L),S);
            if (!L && R) INS(p,Z,S),INS(p,c(c(Z,j+1,0),j,R),S);
            if (i==tx && j==ty && Z==rx[j]+2*rx[j+1]) Ans+=S;
         }
      }
```

```
    printf("%lld\n",Ans);
    return 0;
}
```

3.2 决策单调性

```
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;

const int Maxn=500000;

int n;
int num[Maxn+5];
double dp[2][Maxn+5];

inline double calc(int x,int now){
    return num[x]-num[now]+sqrt((double)fabs(x-now));
}

void divide(int left,int right,int l,int r,double ans[Maxn+5]){
    if (left>right) return;
    int mid=(left+right)/2,rec=l;
    ans[mid]=0;
    for (int i=l;i<=r && i<=mid;i++){
        if (ans[mid]<=calc(i,mid)){
            ans[mid]=calc(i,mid);
            rec=i;
        }
    }
    divide(left,mid-1,l,rec,ans);
    divide(mid+1,right,rec,r,ans);
}

int main(){
    //freopen("in.txt","r",stdin);

    scanf("%d",&n);
    for (int i=1;i<=n;i++) scanf("%d",&num[i]);

    divide(1,n,1,n,dp[0]);
    reverse(num+1,num+n+1);
    divide(1,n,1,n,dp[1]);
    reverse(dp[1]+1,dp[1]+n+1);

    for (int i=1;i<=n;i++) printf("%d\n",(int)ceil(max(dp[0][i],dp[1][i])));
```

```
42    return 0;
43 }
```

3.3 数位 DP

从高位向地位的数位 DP: 使用一个 0/1 状态保存是否 x 已经小于上界，其他状态来表示高位信息。

```
1  int get(char *x, bool flag){
2     memset(dp, 0, sizeof(dp));
3     dp[0][0][0][0] = 1;
4     int len = strlen(x);
5     reverse(x , x + len);
6     int fr = 0;
7     for (int i = len - 1; i >= 0; i--){
8        int to = fr ^ 1;
9        memset(dp[to], 0, sizeof(dp[to]));
10       for (int op = 0; op < 2; op++)
11          for (int j = 0; j < m; j++)
12             for (int k = 0; k < m; k++)
13                for (int l = 0; l <= (op?9:(x[i]-'0')); l++){
14                   int opt = op?1 : (l < (x[i] - '0'));
15                   (dp[to][opt][_s][_k] += dp[fr][op][j][k]) %= mod;
16                }
17       fr = to;
18    }
19    int cnt = 0;
20    for (int i = 0; i < m; i++){
21       (cnt += flag?dp[fr][0][i][0]:0) %= mod; //flag为1表示上界为闭
22       (cnt += dp[fr][1][i][0]) %= mod;
23    }
24    return cnt;
25 }
```

LightOJ1205 回文数 求区间内回文数数量

```
1  #include <iostream>
2  #include <cstring>
3  #include <string>
4  #include <vector>
5  #include <queue>
6  #include <cstdio>
7  #include <set>
8  #include <cmath>
9  #include <map>
10 #include <algorithm>
11 #define INF 0x3f3f3f3f
12 #define MAXN 10000005
13 #define Mod 10001
```

```cpp
using namespace std;
int dight[40],tmp[40];
long long dp[40][100][100];
long long dfs(int start,int pos,int s,bool limit)
{
   if(pos<0)
      return s;
   if(!limit&&dp[pos][s][start]!=-1)
      return dp[pos][s][start];
   int end;
   long long ret=0;
   if(limit)
      end=dight[pos];
   else
      end=9;
   for(int d=0; d<=end; ++d)
   {
      tmp[pos]=d;
      if(start==pos&&d==0)
         ret+=dfs(start-1,pos-1,s,limit&&d==end);
      else if(s&&pos<(start+1)/2)
         ret+=dfs(start,pos-1,tmp[start-pos]==d,limit&&d==end);
      else
         ret+=dfs(start,pos-1,s,limit&&d==end);
   }
   if(!limit)
      dp[pos][s][start]=ret;
   return ret;
}
long long solve(long long a)
{
   memset(dight,0,sizeof(dight));
   int cnt=0;
   while(a!=0)
   {
      dight[cnt++]=a%10;
      a/=10;
   }
   return dfs(cnt-1,cnt-1,1,1);
}
int main()
{
   memset(dp,-1,sizeof(dp));
   int t,cnt=1;
   scanf("%d",&t);
   while(t--)
   {
      long long x,y;
```

```
62      scanf("%lld%lld",&x,&y);
63      if(x>y)
64         swap(x,y);
65      printf("Case %d: %lld\n",cnt++,solve(y)-solve(x-1));
66   }
67   return 0;
68 }
```

3.4 状态压缩动态规划

Problem

BZOJ 4197 (互质集合方案数) 题意：给定数字 2,3,4…,N. 问从中选出若干个数字分成两个集合，使得两个集合间任意元素对互质的方案数。
题解：讨论互质只讨论质因数即可，当 N 不大时 ($\sqrt{N}$ 内质因数的数量很小)，即可对小于等于 $\sqrt{N}$ 的质因数状态压缩，大于 $\sqrt{N}$ 的质因数分类讨论即可（因为这类质因数每个数字最多含有一个）

3.5 斜率优化 DP

将转移方程化为 $y = kx + c$ 的形式，从 dp[j]->dp[i]，c=dp[i],x、y 为关于 j 的项，k 为斜率与 i 有关。

BZOJ 4518 转移方程：$dp[i] = min\{dp[j] + (sum[i] - sum[j])^2\}$

```
1  #include <cstdio>
2  #include <cstring>
3  #include <algorithm>
4  #define LL long long
5  #define sqr(x) ((x)*(x))
6  using namespace std;
7
8  const int Maxn=3000;
9
10 int n,m;
11 int dis[Maxn+5];
12 LL sum[Maxn+5],avg;
13 LL sum2[Maxn+5];
14 LL f[Maxn+5],g[Maxn+5];
15
16 LL getX(int x){return 2LL*sum[x];}
17 LL getY(int x){return f[x]+sqr(sum[x]);}
18
19 struct Point{
20    LL x,y;
21    int idx;
22    Point(){}
23    Point(int i0,LL x0,LL y0){
24       idx=i0;
25       x=x0;
```

```
26      y=y0;
27   }
28   LL getDP(int i){return f[idx]+sqr(sum[i]-sum[idx]);}
29   Point operator -(const Point &p){return Point(0,x-p.x,y-p.y);}
30   LL operator *(const Point &p){return x*p.y-y*p.x;}
31 }poi[Maxn+5];
32 int lc,rc,ri[Maxn+5];
33 Point stk[Maxn+5];
34 int top;
35
36 inline bool cmpPoint(Point a,Point b){
37   if (a.x!=b.x) return a.x<b.x;
38   return a.y<b.y;
39 }
40
41 inline bool cmpK(int a,int b){
42   return sum[a]<sum[b];
43 }
44
45 inline void addPoint(Point &p){
46   while(top>=2 && (stk[top]-stk[top-1])*(p-stk[top])<=0LL) top--;
47   stk[++top]=p;
48 }
49
50 void CDQ(int l,int r){
51   if (l>=r) return; int mid=(l+r)/2;
52   CDQ(l,mid); CDQ(mid+1,r);
53
54   rc=lc=0;
55   for (int i=l;i<=mid;i++) if (f[i]>=0) poi[++lc]=Point(i,getX(i),getY(i));
56   for (int i=mid+1;i<=r;i++) ri[++rc]=i;
57   if (!lc) return;
58   sort(poi+1,poi+lc+1,cmpPoint);
59   sort(ri+1,ri+rc+1,cmpK);
60
61   stk[top=1]=poi[1];
62   if (lc>1) stk[top=2]=poi[2];
63   for (int i=3;i<=lc;i++) addPoint(poi[i]);
64
65   for (int i=1,j=1;i<=rc;i++){
66      int now=ri[i];
67      while(j<top && stk[j].getDP(now)>=stk[j+1].getDP(now)) j++;
68      if (g[now]==-1 || g[now]>stk[j].getDP(now)) g[now]=stk[j].getDP(now);
69   }
70 }
71
72 int main(){
73   //freopen("in.txt","r",stdin);
```

```
74      //freopen("out.txt","w",stdout);
75      scanf("%d%d",&n,&m);
76      for (int i=1;i<=n;i++) scanf("%d",&dis[i]);
77      sum[0]=0; for (int i=1;i<=n;i++) sum[i]=sum[i-1]+dis[i];
78      for (int i=0;i<=n;i++) g[i]=sqr(sum[i]);
79
80      for (int i=2;i<=m;i++){
81          memcpy(f,g,sizeof(g));
82          memset(g,-1,sizeof(g));
83          CDQ(1,n);
84      }
85
86      printf("%lld\n",g[n]*m-sqr(sum[n]));
87      return 0;
88  }
```

第四章　数论

4.1　欧几里得

4.1.1　拓展欧几里得算法

裴蜀定理　$ax + by = \gcd(a, b)$

```
lld extgcd(lld a, lld b, lld &x, lld &y) {
   if (!b) return x = 1, y = 0, a;
   lld d = extgcd(b, a % b, y, x);
   return y -= (a / b) * x, d;
}
```

4.1.2　两数间分母最小的数

在 $ay + 1 < bx$ 时求 $a/b < p/q < x/y$。

```
void euclid(lld a, lld b, lld x, lld y, lld &p, lld &q) {
   lld K = a / b; a -= K * b, x -= K * y;
   if (x > y) p = q = 1; else euclid(y, x, b, a, q, p);
   p += K * q;
}
```

4.1.3　类欧几里得算法

$$f(k_1, k_2, a, b, c, n) = \sum_{i=0}^{n} x^{k_1} \left\lfloor \frac{ax + b}{c} \right\rfloor^{k_2}$$

如果 $a \geq c$ 或 $b \geq c$，那么就有

$$f(k_1, k_2, a, b, c, n) = \sum_{i=0}^{n} x^{k_1} \left(\left\lfloor \frac{a}{c} \right\rfloor x + \left\lfloor \frac{b}{c} \right\rfloor + \left\lfloor \frac{(a\%c)x + (b\%c)}{c} \right\rfloor \right)^{k_2}$$

后面这部分进行二项式展开后是若干项类似于 $f(k_1', k_2', a\%c, b\%c, c, n)$ 的东西。考虑使用二项式展开、$x^t = \sum_{i=0}^{x-1} (i+1)^t - i^t$ 和插值。下面给出一个特例的推导过程。

$$f(a, b, c, n) = \sum_{i=0}^{n} \left\lfloor \frac{ai + b}{c} \right\rfloor$$

当 $a \geq c$ 或 $b \geq c$ 时，将整除部分拖出来。

$$f(a, b, c, n) = f(a\%c, b\%c, c, n) + \frac{n(n+1)}{2} \left\lfloor \frac{a}{c} \right\rfloor + (n+1) \left\lfloor \frac{b}{c} \right\rfloor$$

否则的话，令 $m = \left\lfloor \frac{an+b}{c} \right\rfloor$，则上式可以表示为

$$
\begin{aligned}
f(a,b,c,n) &= \sum_{i=0}^{n}\sum_{j=1}^{m}\left[\left\lfloor\frac{ai+b}{c}\right\rfloor \geq j\right] = \sum_{i=0}^{n}\sum_{j=0}^{m-1}\left[\left\lfloor\frac{ai+b}{c}\right\rfloor \geq j+1\right] \\
&= \sum_{i=0}^{n}\sum_{j=0}^{m-1}[ai \geq jc+c-b] = \sum_{i=0}^{n}\sum_{j=0}^{m-1}[ai > jc+c-b-1] \\
&= \sum_{i=0}^{n}\sum_{j=0}^{m-1}\left[i > \frac{jc+c-b-1}{a}\right] = \sum_{j=0}^{m-1}\left(n-\left\lfloor\frac{jc+c-b-1}{a}\right\rfloor\right) \\
&= nm - f(c,c-b-1,a,m-1)
\end{aligned}
$$

对于这三个函数的模版：

$$
f(a,b,c,n) = \sum_{i=0}^{n}\left\lfloor\frac{ai+b}{c}\right\rfloor,\quad g(a,b,c,n) = \sum_{i=0}^{n} i\left\lfloor\frac{ai+b}{c}\right\rfloor,\quad h(a,b,c,n) = \sum_{i=0}^{n}\left\lfloor\frac{ai+b}{c}\right\rfloor^2
$$

```
struct data { int f, g, h; data() { f = g = h = 0; } };
data calc(int a, int b, int c, lld n) {
   if (!a && b<c) {
      return data();
   } else if (a >= c || b >= c) {
      data tmp = calc(a%c, b%c, c, n);
      tmp.h += n*(n+1)*(2*n+1)/6*(a/c)*(a/c) + (n+1)*(b/c)*(b/c)
            + 2*(b/c)*tmp.f + 2*(a/c)*tmp.g + n*(n+1)*(a/c)*(b/c);
      tmp.f += n*(n+1)/2*(a/c) + (n+1)*(b/c);
      tmp.g += n*(n+1)*(2*n+1)/6*(a/c) + n*(n+1)/2*(b/c);
      return tmp;
   } else {
      lld m = (a*n+b)/c;
      data nxt = calc(c, c-b-1, a, m-1), tmp;
      tmp.f = n*m - nxt.f;
      tmp.g = (n*(n+1)*m - nxt.f - nxt.h) / 2;
      tmp.h = m*(m+1)*n - 2*nxt.g - 2*nxt.f - tmp.f;
      return tmp;
   }
}
```

4.1.4 连分数逼近

拥有此形式的分数我们称为连分数。

$$
a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}}
$$

通常记为 $[a_0, a_1, a_2, \cdots, a_n]$。其渐进分数有如下性质：

$$
\begin{cases} p_k = a_k p_{k-1} + p_{k-2} \\ q_k = a_k q_{k-1} + q_{k-2} \end{cases}
\quad
\begin{cases} p_k q_{k-1} - q_k p_{k-1} = (-1)^{k-1} \\ p_k q_{k-2} - q_k p_{k-2} = (-1)^k a_k \end{cases}
\quad
\frac{p_{2k}}{q_{2k}} < \frac{p_{2k+2}}{q_{2k+2}},\ \frac{p_{2k-1}}{q_{2k_1}} > \frac{p_{2k+1}}{q_{2k+1}}
$$

4.2 唯一分解

4.2.1 约数个数的上界

$n \leq$	10^1	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9
$\max c(n)$	2	3	4	5	6	7	8	8	9
$\max d(n)$	4	12	32	64	128	240	448	768	1344
$n \leq$	10^{10}	10^{11}	10^{12}	10^{13}	10^{14}	10^{15}	10^{16}	10^{17}	10^{18}
$\max c(n)$	10	10	11	12	12	13	13	14	15
$\max d(n)$	2304	4032	6720	10752	17280	26880	41472	64512	103680

4.2.2 Miller-Rabin 素性测试

```
bool millerRabin(lld n, lld base) {
   lld n2 = n-1, s = __builtin_ctzll(n2); n2 >>= s;
   lld t = fpow(base, n2, n); if (t == 1 || t == n-1) return true;
   for (s--; s >= 0; s--) if ((t=mul(t,t,n))==n-1) return true;
   return false;
}

bool isPrime(lld n) {
   static lld bases[12] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 };
   if (n <= 2) return n == 2;
   for (int i = 0; i < 12 && bases[i] < n; i++)
      if (!millerRabin(n, bases[i])) return false;
   return true;
}
```

4.2.3 Pollard-rho 大数分解

```
void PollardRho(lld n) {
   if (isPrime(n)) {
      report_factor(n);
   } else if (n & 1) {
      auto f = [n] (lld x) -> lld { return add(1, mul(x, x, n), n); };
      for (int i = 1; n > 1; i++) {
         lld x = i, y = f(x), q = __gcd(y-x, n);
         while (q == 1) x = f(x), y = f(f(y)), q = __gcd((y-x+n)%n, n) % n;
         if (q != 0 && q != n) { PollardRho(q), PollardRho(n/q); return; }
      }
   } else {
      while (!(n & 1)) report_factor(2), n >>= 1;
      PollardRho(n);
   }
}
```

4.2.4 圆上整点

```
void gaussInteger(lld R, vector<pair<int,int>> &tot) {
   auto check = [&tot] (lld r, lld rr) -> void {
      if (r == 1 || r % 4 != 1) return;
      for (int n = 1; n*n*2 < r; n++) {
         int m = int(sqrt(r-n*n)+1e-5);
         if (m * m + n * n != r) continue;
         if (gcd(m,n) != 1 || m <= n) continue;
         tot.emplace_back(rr*(m*m-n*n), 2*n*m*rr);
      }
   };

   for (int i = 1; i * i <= R; i++) if (R % i == 0) {
      check(R/i, i); if (i * i != R) check(i, R/i);
   }
}
```

4.2.5 逆元

非素数 丢进拓展欧几里得算法。

素数求单个 $\operatorname{inv} m = m^{MOD-2} \mod MOD$

素数求连续区间 线性可以求出 $[1, m]$ 所有逆元。

```
void init() {
   inv[1] = 1;
   for (int i = 2; i < MAXN; i++)
      inv[i] = 1ll * (MOD-MOD/i) * inv[MOD%i] % MOD;
}
```

特殊模数 当 m 被 $MOD+1$ 整除时可以直接写 $(MOD+1)/m$。

4.3 离散对数

4.3.1 指标与原根

若 $\operatorname{ord}_m a = \varphi(m)$ 时，称 a 是 m 的一个原根。当 p 为奇质数时，$m = 1, 2, 4, p, p^\alpha, 2p^\alpha$ 有原根。原根是简化剩余系的生成元，共有 $\varphi(\varphi(m))$ 个。枚举从 $g = 2$ 开始，依次检查 $g^{\varphi(m)/p}$ 即可。

```
lld getOrd(lld a) {
   lld res = mod-1;
   for (int i = 0; i < w; i++) {
      lld qwq = fpow(a, res/=zs[i], mod);
      while (qwq != 1) qwq = fpow(qwq, fp[i], mod), res *= fp[i];
   }
   return res;
} // O(c(n)log(n)), 需要 w=c(mod-1), fp[i]=p_i, zs[i]={p_i}^{e_i}
```

4.3.2 Baby Step Gaint Step

```
lld bsgs(lld a, lld b, lld p) {
   a %= p, b %= p;
   if (a == 0) return !b ? p!=1 : -1;
   unordered_map<lld, lld> mp;
   lld m = ceil(sqrt(p+0.5)), aa = fpow(a, m, p);
   for (lld i = 0, ans; i <= m; i++)
      mp[ans = i ? ans*a%p : b] = i;
   for (lld i = 1, ans = 1; i <= m; i++)
      if (mp.count(ans = ans * aa % p))
         return i * m - mp[ans];
   return -1;
}
```

4.3.3 拓展 BSGS

```
lld exBSGS(lld a, lld b, lld mod) {
   lld d, g = 0, k = 1;
   while ((d = gcd(a, b)) != 1) {
      if (b % d) return -1;
      g++, mod /= d, b /= d, k = k * a / d % mod;
      if (b == k) return g;
   }

   lld m = ceil(sqrt(mod+0.5)), aa = fpow(a, m, mod), ans;
   unordered_map<lld, lld> mp;
   for (int i = 0; i <= m; i++)
      mp[ans = i ? ans*a%mod : b] = i;
   for (int i = 1; i <= m; i++)
      if (mp.count(k = k * aa % mod))
         return i * m - mp[k] + g;
   return -1;
}
```

4.3.4 Pohlig-Hellman

使用群论语言描述。先考虑计算 p^e 阶循环群 $G=\langle g\rangle$ 的离散对数 $g^x=h$。

1. 初始化 $x_0:=0,\gamma:=g^{p^{e-1}}$。
2. 对于 $k\in\{0,\cdots,e-1\}$ 分别做
   (a) 计算 $h_k:=(g^{-x_k}h)^{p^{e-1-k}}$。
   (b) 使用 BSGS 算法计算 $d_k\in\{0,\cdots,p-1\}$，其中 $\gamma^{d_k}=h_k$。
   (c) 令 $x_{k+1}:=x_k+p^kd_k$。
3. 返回 x_e。

当 $e \ll p$ 时，这个算法优于 BSGS，复杂度为 $O(e\sqrt{p})$。再考虑一般合数阶循环群 $G = \langle g \rangle$，首先将阶数刻画为 $n = \prod_{i=1}^{r} p_i^{e_i}$，现在解方程 $g^x = h$。

1. 对于 $i \in \{1, \cdots, r\}$ 分别做
   (a) 计算 $g_i := g^{n/p_i^{e_i}}, h_i := h^{n/p_i^{e_i}}$。
   (b) 利用上面算法计算 x_i，使得 $g_i{}^{x_i} = h_i$。
2. 利用中国剩余定理解方程组 $x \equiv x_i \mod p_i^{e_i}$。

4.3.5 拓展欧拉定理

$$a^b \equiv \begin{cases} a^b, & b \le \varphi(m) \\ a^{b\%\varphi(m)+\varphi(m)}, & \text{otherwise} \end{cases} \mod m$$

适用于底数与模数不互质的情况。当无穷指数的时候，每一层都取模欧拉函数，递归下去。

4.3.6 威尔逊定理

$$\prod_{k=1}^{m} k^{[\gcd(k,m)=1]} \equiv \begin{cases} -1 (\mathrm{mod}\, m) & \text{if } m = 4, p^\alpha, 2p^\alpha \\ 1 (\mathrm{mod}\, m) & \text{otherwise} \end{cases}$$

$$(p-1)! + 1 \equiv 0 \ (\mathrm{mod}\, p)$$

4.4 数论方程

4.4.1 不定方程

求解 $Ax + By = C$，利用裴蜀定理，$Ax_0 + By_0 = \gcd(A, B)$，如果 $\gcd(A, B) \mid C$ 则有解。

$$\begin{cases} x = \frac{Cx_0 + Bt}{\gcd(A,B)} \\ y = \frac{Cy_0 - At}{\gcd(A,B)} \end{cases}$$

4.4.2 线性同余方程

求解 $ax \equiv b \ (\mathrm{mod}\, n)$，转化成 $ax - ny = b$，利用裴蜀定理转化为 $ax - ny = \gcd(a, n)$ 的解的问题。一个可行解为 $ans = x_0 \frac{b}{\gcd(a,n)}$，解间隔为 $s = \frac{n}{\gcd(a,b)}$，最小整数解调整进 s 范围即可。

4.4.3 同余方程组

```
pair<lld,lld> modeqs(lld b[], lld w[], int k) {
   lld bj = b[0], wj = w[0], x, y, d;
   for (int i = 1; i < k; i++) {
      b[i] %= w[i]; d = extgcd(wj, w[i], x, y);
      if ((bj - b[i]) % d != 0) return make_pair(0LL, -1LL);
      x = x * (b[i] - bj) / d % (w[i] / d), y = wj / d * w[i];
      bj = ((bj + x * wj) % y + y) % y, wj = y;
   }
   return make_pair(bj, wj);
} // a = b[i] (mod w[i])
```

4.4.4 高次同余方程

求解 $x^k \equiv a \pmod m$。由原根定义知 $g^t \equiv x, g^s \equiv a$，转化为求解 $tk \equiv s \pmod{m-1}$。

4.4.5 对数方程

参考离散对数。

4.4.6 二次剩余

```
struct ext {
   lld W, a, b; ext(lld aa, lld bb, lld w) { a = aa, b = bb, W = w; }
   ext operator*(const ext &s) const {
      return ext((a*s.a+b*s.b%MOD*W)%MOD, (a*s.b+b*s.a)%MOD, W); }
   ext operator^(lld n) const {
      ext s(1,0,W),a=*this; for(;n;n>>=1,a=a*a)if(n&1)s=s*a; return s; }
};

lld cipolla(lld k) {
   if (fpow(k, (MOD-1)/2) == MOD-1) return -1; else if (k == 0) return 0;
   lld a = 1; static mt19937 Rand(time(NULL));
   while (fpow((a*a%MOD-k+MOD)%MOD, (MOD-1)/2) != MOD-1) a = Rand() % MOD;
   ext x = ext(a, 1, (a*a%MOD-k+MOD)%MOD)^((MOD+1)/2);
   assert(x.b == 0); return min(x.a, MOD-x.a);
}
```

4.4.7 佩尔方程

对于 $x^2 - dy^2 = 1$，先暴力求出一组最小特解，然后利用递推关系。

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 & dy_1 \\ y_1 & x_1 \end{bmatrix} \begin{bmatrix} x_{n-1} \\ y_{n-1} \end{bmatrix}$$

注意 $\sqrt{d}$ 的连分数是周期性的，即可以表示成 $[a_0; < a_1, a_2, \cdots, a_{n-1}, 2a_0 >]$。记 $[a_0, a_1, \cdots, a_{n-1}]$ 的有理表示为 $\frac{p}{q}$，循环节长度为 s，那么在 s 为偶数时，最小特解为 $x_0 = p, y_0 = q$，在 s 为奇数时，最小特解为 $x_0 = 2p^2 + 1, y_0 = 2pq$。

4.4.8 勾股方程

$a^2 + b^2 = c^2$ 的本原解为 $a = m^2 - n^2, b = 2mn, c = m^2 + n^2$。

4.5 积性函数

设 $f : \mathbb{Z}^+ \to \mathbb{C}$，且对于 n, m 有 $\gcd(n, m) = 1$ 时，$f(nm) = f(n)f(m)$ 成立，则称它为积性函数。如果条件可以减弱为不用互质，则称为完全积性函数。

4.5.1 埃式筛

大区间素数筛选 求出 $[a, b]$ 中的所有素数，其中 $a \le 10^9, b - a \le 10^6$。注意到 $[a, b]$ 中的每个合数最大质因子不会超过 $\sqrt{b}$，再将区间平移到 $[0, b - a]$ 即可。$O(n \log n)$

4.5.2 线性筛

考虑让每个数只被它的最小素因子筛选到。$O(n)$

```
char miu[MAXN], c[MAXN], isnp[MAXN];
int phi[MAXN], d[MAXN], pri[MAXN], pcn;

void init_prime() {
   isnp[0] = isnp[1] = 1;
   miu[1] = phi[1] = d[1] = 1;

   for (int i = 2; i < MAXN; i++) {
      if (!isnp[i]) {
         pri[++pcn] = i, c[i] = 1, d[i] = 2;
         miu[i] = -1, phi[i] = i-1;
      }

      for (int j = 1; j <= pcn; j++) {
         if (i * pri[j] >= MAXN) break;
         isnp[i * pri[j]] = 1;
         if (i % pri[j] == 0) {
            c[i*pri[j]] = c[i]+1, d[i*pri[j]] = d[i] / (c[i]+1) * (c[i]+2);
            miu[i*pri[j]] = 0, phi[i*pri[j]] = phi[i] * pri[j];
            break;
         } else {
            c[i*pri[j]] = 1, d[i*pri[j]] = 2*d[i];
            miu[i*pri[j]] = -miu[i], phi[i*pri[j]] = phi[i] * (pri[j]-1);
         }
      }
   }
}
```

4.5.3 杜教筛

设 $f(n), g(n)$ 是数论函数，我们称

$$(f*g)(n)=\sum_{d|n} f(d)g\left(\frac{n}{d}\right)$$

是这两个函数的狄利克雷卷积。设 $G=\{f|f\text{ is multiplicative function}\}$，* 是按上述规则构成的运算，则 $(G,*)$ 构成一个阿贝尔群，我们称这个群为积性函数的狄利克雷卷积群，其单位元是 $e(n)$。注意可以利用类似埃式筛的方法暴力进行狄利克雷卷积。

该群中常见运算有 $\mu*1=e, \varphi*1=id, id*\mu=\varphi, 1*1=d, \mu*d=1, id*1=\sigma, id^k*1=\sigma_k$。可以发现，$\mu$ 和 1 互为逆元。

对于给定积性函数 $f(n)$，我们希望找到一个 $g(n)$，使得它们的狄利克雷卷积的前缀和尽可能方便的计算。通常 $g(n)$ 是在其狄利克雷卷积环上找一个逆元素或好算的。

$$\sum_{k=1}^{n}\sum_{ij=k} g(i)f(j)=\sum_{i=1}^{n}\sum_{j=1}^{\lfloor n/i\rfloor} g(i)f(j)=\sum_{i=1}^{n} g(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$$

所以有

$$g(1)S(n)=\sum_{i=1}^{n}(f*g)(i)-\sum_{i=2}^{n}g(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$$

后面的东西分块计算。假设我们对 $g(n)$ 和 $(f*g)(n)$ 求单点前缀和是 $O(1)$ 的，那么复杂度为 $O(n^{3/4})$，如果我们先线性筛预处理前 $O(n^{2/3})$ 项，那么时间复杂度降为 $O(n^{2/3})$。注意它可以同时求出根号分块值处的函数值。

当你需要求和的式子为 $f(n)=n^k\varphi(n)$ 时，配 $g(n)=n^k$ 即可。

$$M(n)=\sum_{d=1}^{n}\mu(d)=1-\sum_{d=2}^{n}M\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$$
$$\phi(n)=\sum_{d=1}^{n}\varphi(d)=\frac{n(n+1)}{2}-\sum_{d=2}^{n}\phi\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$$
$$B(n)=\sum_{d=1}^{n}d\varphi(d)=\frac{n(n+1)(2n+1)}{6}-\sum_{d=2}^{n}dB\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$$

```
lld _M1[MAXN]; map<lld, lld> _M2;

lld M(lld n) {
   if (n < MAXN) return _M1[n];
   if (_M2.count(n)) return _M2[n];
   lld ans = 1;
   for (lld L = 2, R = n/(n/2); L <= n; L = R+1, R = n/(n/L))
      ans -= (R - L + 1) * M(n / L);
   return _M2[n] = ans;
}
```

4.5.4 min25 筛

当积性函数 $f(n)$ 在 p 为素数时满足以下性质时可以使用 min25 筛。

- $f(p)$ 可以用关于 p 的多项式表示。
- $f(p^k)$ 可以快速计算答案。

我们需要预处理多项式的 x^k 相关的 $\sum_{i=2}^{n}i^k[i\in\text{prime}]$。记所有质数的集合为 P，第 j 个质数为 P_j，x 的最小质因子为 $\text{minp}(x)$，我们需要预处理

$$g(n,j)=\sum_{i=2}^{n}i^k\ [i\in P\,\text{or}\,\text{minp}(i)>P_j]$$

则转移方程为

$$g(n,j)=\begin{cases}g(n,j-1), & p_j^2>n\\ g(n,j-1)-p_j^k\left(g(\lfloor n/p_j\rfloor,j-1)-\sum_{x=1}^{j-1}p_x^k\right), & p_j^2\le n\end{cases}$$

预处理的时候经常性计算 $g(\lfloor n/x\rfloor)$ 的值，利用根号分块来预处理。$O(\frac{n^{\frac{3}{4}}}{\log n})$

刚才去掉了合数处的点值，现在需要根据积性函数的性质加回来。构造函数

$$S(n,j)=\sum_{x=2}^{n}f(x)\ [\text{minp}(x)\ge p_j]$$

整理完了即为

$$S(n,k)=\sum_{k\geq j}\sum_{e\geq 1} f(p_k^e)(1+S(\lfloor n/p_k^e\rfloor,k+1))$$
$$=g(n,|P|)-\sum_{i=1}^{j-1}f(p_i)+\sum_{k\geq j}\sum_{e\geq 1}\left(f(p_k^e)S(\lfloor n/p_k^e\rfloor,k+1)+f(p_k^{e+1})\right)$$

```
// euler sieve, spk for sum p^k from p_1 ~ p_pcn
// note that it should be pri[++pcn]!!
int pri[MAXN], pcn, spk[MAXN];
int m, Sqrt, id1[MAXN], id2[MAXN];
lld gk[MAXN], w[MAXN];
#define GID(t) ((t)<=Sqrt)?id1[(t)]:id2[n/(t)]
lld sum_ik(int i) { /* for i=2,3,4..n, sum i^k */ }

void init() {
   m = 0, Sqrt = sqrt(n);
   for (lld L = 1, R; L <= n; L = R+1)
      R = n / (n / L), w[++m] = n / L, gk[m] = sum_ik(w[m]),
         (w[m] <= Sqrt ? id1[w[m]] : id2[R]) = m;
   for (int j = 1; j <= pcn; j++) // p_pcn^2 should be greater than n
      for (int i = 1; i <= m && pri[j] <= w[i]/pri[j]; i++)
         gk[i] -= (spk[j] - spk[j-1]) * (gk[GID(w[i]/pri[j])] - spk[j-1]);
}

lld S(lld x, int y) {
   if (x <= 1 || pri[y] > x) return 0;
   int d = GID(x), e; lld ans = g(x) - g(pri[y]-1), p1;
   for (int i = y; i <= pcn && pri[i] <= x / pri[i]; i++)
      for (e = 1, p1 = pri[i]; p1 <= x / pri[i]; ++e, p1 *= pri[i])
         ans += S(x/p1, i+1) * f(i,e) + f(i,e+1);
   return ans;
} // 递归单点求值 F(n) = f(1) + f(2) + ... + f(n) = S(n,1) + 1

void solve(int S[MAXN], int sf[MAXN]) {
   for (int i = 1; i <= pcn; i++) sf[i] = sf[i-1] + f(pri[i]);
   for (int i = 1; i <= m; i++) S[i] = g[i]; // merge from gk[i]
   for (int i = pcn; i >= 1; i--)
      for (int j = 1, e = 1; j <= m; j++, e = 1)
         for (lld p1 = pri[i]; p1 <= w[j] / pri[i]; p1 *= pri[i], e++)
            S[j] += f(i,e+1) + f(i,e) * (S[GID(w[j]/p1)] - sf[i]);
} // 同时求出根号分块处的点值
```

若 $f(p^e)=C_{e+A-1}^{A-1}$,则 $f(p)=A$,预处理 `sp0, g0`,则 `g(x)-g(pri[y]-1)` 写为 `A*(g0[d]-(y-1))`,而 `f(i,e)` 写为 `C(e+A-1,A-1)`。

若 $f(p^e)=Ae+1$,则 $f(p)=A+1$,预处理 `sp0, g0`,则 `g(x)-g(pri[y]-1)` 写为 `(A+1)*(g0[d]-(y-1))`,而 `f(i,e)` 写为 `A*e+1`。

若 $f(p^e)=p\oplus e$,则 $f(2)=3, f(p)=p-1$,预处理 `sp0, sp1, g0, g1`,则 `g(x)-g(pri[y]-1)` 写为 `g1[d]-g0[d]-sp1[y-1]+y-1`,并特判 `if(y==1) ans+=2`,而 `f(i,e)` 写为 `pri[i]^e`。

4.5.5 6N 求素数

```
void add(int x) { ans++; }
void prime(int N) {
   static bitset<1000000001/3> p;
   add(2); add(3);
   for (int i=5, d=2; i<=N; i+=d, d=6-d) if(!p[i/3]) {
      add(i); if(i > N/i) continue;
      for (int j=i*i, v=d; j<=N; j+=i*v, v=6-v) p[j/3] = 1;
   }
} // fcynb
```

4.5.6 DIVCNT1

```
typedef __int128 i128; lld n;
struct pr { lld x, y; pr (lld x0 = 0, lld y0 = 0) : x(x0), y(y0) {}
   inline pr operator+(const pr &B) const { return pr(x+B.x, y+B.y); } };
inline bool inner(lld x, lld y) { return n < x * y; }
inline bool steep(lld x, pr v) { return (i128)n * v.x <= (i128)x * x * v.y; }
i128 S1() {
   static pr stack[1000005]; int top = 0; int i, crn = cbrt(n);
   lld srn = sqrt(n), x = n / srn, y = srn + 1;
   i128 ret = 0; pr L, R, M;
   stack[++top] = pr(1, 0); stack[++top] = pr(1, 1);
   while (true) {
      for (L = stack[top--]; inner(x + L.x, y - L.y); x += L.x, y -= L.y)
         ret += x * L.y + (L.y + 1) * (L.x - 1) / 2;
      if (y <= crn) break;
      for (R = stack[top]; !inner(x+R.x, y-R.y); R = stack[--top]) L = R;
      for (; M = L + R, 1; )
         if (inner(x+M.x, y-M.y)) stack[++top] = R = M;
         else { if (steep(x+M.x, R)) break; L = M; }
   }
   for (i = 1; i < y; ++i) ret += n / i;
   return ret * 2 - srn * srn;
} // 求d(1)+d(2)+...+d(n), n<=2^63
```

第五章　线性代数

5.1　多项式

5.1.1　拉格朗日插值

```
Poly Lagrange(const Poly &X, const Poly &FX) {
   Poly c; int n = X.size();
   for (int i = 0; i < n; i++) {
      Poly x({FX[i]});
      for (int j = 0; j < n; j++) if (j != i)
         x *= Poly({-X[j],1}), x *= fpow(X[i]-X[j], MOD-2);
      c += x;
   } // 求单点值只需要 O(n^2logMOD)，如果函数定义域连续可以预处理阶乘等达到 O(n)
   return c;
} // vector<lld> 实现 Poly *= Poly, Poly *= lld, Poly += Poly
```

5.1.2　快速傅立叶变换

```
const double PI = acos(-1.0);
typedef complex<double> cplx;

void dft(cplx a[], int DFT, int N) {
   static int rev[1<<20]; static cplx w[1<<20], ww[1<<20];
   // 小心浮点数运算的精度！有必要时上long double
   for (int i = 0; i < N; i++) {
      rev[i] = (rev[i>>1]>>1)|((i&1)?N>>1:0);
      if (i < rev[i]) swap(a[i], a[rev[i]]);
      w[i] = cplx(cos(PI*i/N), sin(PI*i/N));
      ww[i] = cplx(w[i].real(), -w[i].imag());
   }
   for (int d = 0, ctz = __builtin_ctz(N); (1<<d) < N; d++) {
      cplx u, t; int m2 = 1<<d, m = m2<<1;
      for (int k = 0; k < N; k += m)
         for (int j = 0; j < m2; j++)
            t = (DFT==1?w:ww)[j<<(ctz-d)] * a[k+j+m2], u = a[k+j],
            a[k+j] = u + t, a[k+j+m2] = u - t;
   }
   if (DFT == -1) for (int i = 0; i < N; i++) a[i] /= N;
}
```

5.1.3 快速数论变换

```
const int MOD = 998244353, G = 3;

void dft(int a[], int DFT, int N) {
   static int rev[1<<20], wmk[1<<20];
   for (int i = 0; i < N; i++)
      rev[i] = (rev[i>>1]>>1)|((i&1)?(N>>1):0);
   for (int i = 0; i < N; i++)
      if (i < rev[i]) swap(a[i], a[rev[i]]);
   for (int m = 2, m2 = 1; m <= N; m <<= 1, m2 <<= 1) {
      int wm = fpow(G, (MOD-1)/m*DFT+(DFT==-1?MOD-1:0));
      for (int j = 0; j < m2; j++)
         wmk[j] = j ? 1ll * wmk[j-1] * wm % MOD : 1;
      for (int k = 0; k < N; k += m)
         for (int j = 0, u, t; j < m2; j++)
            t = 1ll * wmk[j] * a[k+j+m2] % MOD, u = a[k+j],
            a[k+j] = (u+t)%MOD, a[k+j+m2] = (u-t+MOD)%MOD;
   }
   if (DFT == -1)
      for (int i = 0, invN = fpow(N,MOD-2); i < N; i++)
         a[i] = 1ll * a[i] * invN % MOD;
} // 调用次数过多可考虑在全局预处理所有N对应的rev和wmk
```

5.1.4 分治 FFT

```
// dp[i] = sum(a[j] * dp[i-j], where j = 1..i, a[0] = 0)
void cdqFFT(int L, int R) {
   if (L == R) { if (L == 0) dp[L] = 1; return; }
   int mid = (L+R)>>1, len=R-L+1;
   cdqFFT(L, mid);
   static int x[MAXN], y[MAXN];
   for (int i = 0; i < len<<1; i++)
      x[i] = (i < len) ? a[i] : 0,
      y[i] = (i <= mid-L) ? dp[i+L] : 0;
   dft(x, 1, len<<1); dft(y, 1, len<<1);
   for (int i = 0; i < len<<1; i++)
      x[i] = 1ll * x[i] * y[i] % MOD;
   dft(x, -1, len<<1);
   for (int i = mid+1; i <= R; i++)
      dp[i] = (dp[i] + x[i-L]) % MOD;
   cdqFFT(mid+1, R);
}
```

```
// dp[n] = inv[n] * sum(dp[i] * dp[j], where i+j = n-1)
void cdqFFT(int L, int R) {
   if (L == R) {
      if (L == 0) dp[L] = 0; else if (L == 1) dp[L] = 1;
```

```
        else dp[L] = (1ll * dp[L] * inv[L] % MOD + MOD) % MOD;
    } else {
        static int x[MAXN], y[MAXN];
        int mid = (L+R)>>1, len = R-L+1;
        cdqFFT(L, mid);
        for (int i = 0; i < len<<1; i++)
            x[i] = (i < len) ? dp[i] : 0,
            y[i] = (i <= mid-L) ? dp[i+L] : 0;
        dft(x, 1, len<<1); dft(y, 1, len<<1);
        for (int i = 0; i < len<<1; i++)
            x[i] = 1ll * x[i] * y[i] % MOD;
        dft(x, -1, len<<1);
        for (int i = mid+1; i <= R; i++)
            dp[i] = (dp[i] + x[i-L-1] * (L?2LL:1LL)) % MOD;
        cdqFFT(mid+1, R);
    }
}
```

```
// A[n+1] = INV6 * (sum(A[i] * A[j] * A[k], where i+j+k=n)
// + 3 * sum(A[i] * A[j], where i+2j=n) + 2 * sum(A[i], where 3i = n))
void cdqFFT(int L, int R) {
    if (L == R) {
        if (L == 0) A[L] = 6;
        else if (L % 3 == 1) A[L] = (A[L] + A[L/3] * 2ll) % MOD;
        A[L] = 1ll * A[L] * INV6 % MOD;
    } else {
        int mid = (L+R)>>1, len = R-L+1; cdqFFT(L, mid);
        static int tp[MAXN], tp2[MAXN], tp3[MAXN];
        int fftLen = len << 1; // can be smaller due to circular convolution
        for (int i = 0; i < fftLen; i++)
            tp[i] = (i <= mid-L) ? A[i+L] : 0,
            tp2[i] = (i < len) ? A[i] : 0,
            tp3[i] = ((i&1)==0 && i<=len) ? A[i/2] : 0;
        dft(tp, 1, fftLen); dft(tp2, 1, fftLen); dft(tp3, 1, fftLen);
        for (int i = 0; i < fftLen; i++)
            tp2[i] = 1ll * tp2[i] * tp2[i] % MOD * tp[i] % MOD;
        dft(tp2, -1, fftLen); // tp = A[i]A[j]A[n-i-j]
        for (int i = mid+1; i <= R; i++)
            A[i] = (A[i] + tp2[i-L-1] * (L?3LL:1LL)) % MOD;
        for (int i = 0; i < fftLen; i++)
            tp[i] = 1ll * tp[i] * tp3[i] % MOD;
        dft(tp, -1, fftLen); // tp-new = A[i]A[n-2i]
        for (int i = mid+1; i <= R; i++)
            A[i] = (A[i] + 3ll * tp[i-L-1]) % MOD;
        cdqFFT(mid+1, R);
    }
} // O(nlog^2n)
```

5.1.5 小多项式相乘

```
typedef long long lld;
const int MOD = 998244353, G = 3;
const int MAXN = 524288;
int fac[MAXN], inv[MAXN], invs[MAXN];
int revs[MAXN], wmks[MAXN];
lld fpow(lld a, lld k) {
   lld b = 1;
   for (; k; k>>=1, a=a*a%MOD)
      if (k&1) b=b*a%MOD;
   return b;
}
void dft_init() {
   for (int N = 2; N <= (1<<18); N <<= 1) {
      int *rev = revs + N, *wmk = wmks + N, m2 = N/2;
      for (int i = 0; i < N; i++)
         rev[i] = (rev[i>>1]>>1)|((i&1)?(N>>1):0);
      int wm = fpow(3, (MOD-1)/N);
      for (int j = 0; j < m2; j++)
         wmk[j] = j ? 1ll * wmk[j-1] * wm % MOD : 1;
      wm = fpow(wm, MOD-2);
      wmk = wmks + N + m2;
      for (int j = 0; j < m2; j++)
         wmk[j] = j ? 1ll * wmk[j-1] * wm % MOD : 1;
   }
}
void dft(int a[], int DFT, int N) {
   const int *rev = revs + N, *wmk;
   for (int i = 0; i < N; i++)
      if (i < rev[i]) swap(a[i], a[rev[i]]);
   for (int m = 2, m2 = 1; m <= N; m <<= 1, m2 <<= 1) {
      wmk = wmks + m; if (DFT == -1) wmk += m2;
      for (int k = 0; k < N; k += m)
         for (int j = 0, u, t; j < m2; j++)
            t = 1ll * wmk[j] * a[k+j+m2] % MOD, u = a[k+j],
            a[k+j] = (u+t)%MOD, a[k+j+m2] = (u-t+MOD)%MOD;
   }
   if (DFT == -1)
      for (int i = 0, invN = fpow(N,MOD-2); i < N; i++)
         a[i] = 1ll * a[i] * invN % MOD;
}


int t[262144], a;
int f[262144], B[131072], C[131072];
void solve(int l, int r, int L, int R)
{
```

```
47    if (R - L <= 15)
48    {
49       int len = R-L+1, sz = 1;
50       for (int i = 0; i < len; i++) t[L+i] = !i;
51       for (int w = l; w <= r; w++, sz++){
52          for (int i = sz; i > 0; i--)
53             (t[L+i] = (1ll * t[L+i] * C[w] + 1ll * t[L+i-1] * B[w]) % MOD +
                  MOD) %= MOD;
54          t[L] = (1ll * t[L] * C[w]) % MOD;
55       }
56    }
57    else
58    {
59       int mid = (L+R)>>1, m = (l+r)>>1, len=R-L+1;
60       solve(l, m, L, mid), solve(m+1, r, mid+1, R);
61       static int a[131072], b[131072];
62       for (int i = 0; i < len/2; i++)
63          a[i] = t[L+i], b[i] = t[mid+1+i],
64          a[len/2+i] = b[len/2+i] = 0;
65       dft(a, 1, len), dft(b, 1, len);
66       for (int i = 0; i < len; i++) a[i] = (1LL * a[i] * b[i]) % MOD;
67       dft(a, -1, len);
68       for (int i = 0; i < len; i++)
69          t[L+i] = a[i];
70    }
71 }
```

5.1.6 多项式求逆

```
1  void getInv(int a[], int b[], int len) {
2     if (len == 1) {
3        b[0] = fpow(a[0], MOD-2);
4     } else {
5        getInv(a, b, len>>1);
6        static int A[1<<18], B[1<<18];
7        for (int i = 0; i < len; i++)
8           A[i] = a[i], B[i] = b[i], A[i+len] = B[i+len] = 0;
9        dft(A, 1, len<<1); dft(B, 1, len<<1);
10       for (int i = 0; i < len<<1; i++)
11          A[i] = 1ll * A[i] * B[i] % MOD * B[i] % MOD;
12       dft(A, -1, len<<1);
13       for (int i = 0; i < len; i++)
14          b[i] = (2ll * b[i] - A[i] + MOD) % MOD;
15    } // B_{t+1}(x)=2B_t(x)-A(x)B_t^2(x) (\mod x^{2^{t+1}})
16 } // 注意传入前先给b清零
```

5.1.7 多项式带余除法

```
void getDivAndMod(int f[], int n, int g[], int m, int q[], int r[]) {
   static int gr[1<<18], tmp[1<<18], qq[1<<18];
   for (int i = 0; i <= n; i++) qq[n-i] = f[i];
   for (int i = 0; i <= m; i++) gr[m-i] = g[i];
   int len = 1; while (len <= n-m+1) len <<= 1;
   fill(gr + min(n-m+1,m+1), gr+len, 0), fill(tmp, tmp+len, 0);
   getInv(gr, tmp, len); len <<= 1;
   for (int i = 0; i <= n-m+1; i++) gr[i] = tmp[i];
   for (int i = n-m+2; i < len; i++) gr[i] = qq[i] = 0;
   dft(qq, 1, len), dft(gr, 1, len);
   for (int i = 0; i < len; i++) qq[i] = 1ll * qq[i] * gr[i] % MOD;
   dft(qq, -1, len); reverse(qq, qq + n-m+1); copy(qq, qq+n-m+1, q);
   while (len <= (n<<1)) len <<= 1;
   for (int i = n-m+1; i < len; i++) qq[i] = 0;
   for (int i = 0; i <= m; i++) tmp[i] = g[i];
   for (int i = m+1; i < len; i++) tmp[i] = 0;
   dft(qq, 1, len), dft(tmp, 1, len);
   for (int i = 0; i < len; i++) tmp[i] = 1ll * tmp[i] * qq[i] % MOD;
   dft(tmp, -1, len);
   for (int i = 0; i < m; i++) r[i] = (f[i] - tmp[i] + MOD) % MOD;
} // F = G * Q + R; degF = n, degG = m, degQ = n-m, degR <= m-1.
```

5.1.8 多项式多点求值

将需要求的点值分为两部分，构造 $P_0(x)=\prod_{i=1}^{\lfloor n/2\rfloor}(x-a_i)$ 和 $P_1(x)=\prod_{i=\lfloor n/2\rfloor+1}^{n}(x-a_i)$，则使用带余除法后得到 $F(x)\%P_0(x)=R_0(x)$，则对同一边有 $F(a_i)=R_0(a_i)$，然后分治计算即可。在实际操作时先预处理每一个 $P_i(x)$，此过程使用归并分治 FFT，然后再从上往下递归多项式取模使用选择分治 FFT。

5.1.9 拉格朗日反演

若函数 $f(x),g(x)$ 满足 $f(g(x))=g(f(x))=x$，则有

$$[x^n]f(x)=\frac{1}{n}[x^{n-1}]\left(\frac{x}{g(x)}\right)^n$$

5.1.10 多项式分析运算

求导 按求导定义来。

$$\left(\sum_{k=0}^{n}a_kx^k\right)'=\sum_{k=0}^{n-1}a_{k+1}(k+1)x^k$$

求积分 按不定积分定义来。

$$G(x)=\int\left(\sum_{k=0}^{n}a_kx^k\right)dx=\sum_{k=0}^{n}\frac{a_k}{k+1}x^{k+1}$$

取对数 先求逆和导数再积分。

$$G(x)=\ln A(x)\Rightarrow G'(x)=\frac{A'(x)}{A(x)}$$

指数函数 牛顿迭代一下。似乎也有分治 NTT 做法。

$$B(x) = e^{A(x)} \Rightarrow B_t(x) \equiv B_{t-1}(x)(1 - \ln B_{t-1}(x) + A(x)) \pmod{x^t}$$

多项式开根 套路倍增。

$$B_{t+1}(x) = \frac{2B_t(x)^2 + A(x)}{2B_t(x)} \pmod{x^{t+1}}$$

牛顿迭代 要求 $F(B(x)) \equiv 0$，则有倍增

$$B_{t+1}(x) = B_t(x) - \frac{F(B_t(x))}{F'(B_t(x))} \pmod{x^{t+1}}$$

5.1.11 多项式换元

```
Poly moveTo(const Poly &x, lld n) {
   Poly ans(x.size(), 0); static int jc[100];
   for (int i = jc[0] = 1; i < x.size(); i++)
      jc[i] = jc[i-1] * n % MOD;
   for (int i = 0; i < x.size(); i++)
      for (int j = 0; j < x.size(); j++)
         ans[i] = (ans[i] + 1ll * x[j] * jc[j-i] % MOD * C(j, i)) % MOD;
   return ans;
}
```

5.1.12 DFT 预处理

```
void dft_init() {
   for (int N = 2; N <= (1<<18); N <<= 1) {
      int *rev = revs + N, *wmk = wmks + N, m2 = N/2;
      for (int i = 0; i < N; i++)
         rev[i] = (rev[i>>1]>>1)|((i&1)?(N>>1):0);
      int wm = fpow(3, (MOD-1)/N);
      for (int j = 0; j < m2; j++)
         wmk[j] = j ? 1ll * wmk[j-1] * wm % MOD : 1;
      wm = fpow(wm, MOD-2);
      wmk = wmks + N + m2;
      for (int j = 0; j < m2; j++)
         wmk[j] = j ? 1ll * wmk[j-1] * wm % MOD : 1;
   }
}

void dft(int a[], int DFT, int N) {
   const int *rev = revs + N, *wmk;
   for (int m = 2, m2 = 1; m <= N; m <<= 1, m2 <<= 1) {
      wmk = wmks + m; if (DFT == -1) wmk += m2;
   }
}
```

5.2 矩阵与向量空间

5.2.1 高斯消元

```
int gauss(int n, int m) {
   int i = 0, j = 0;
   for (; i < n; i++) {
      while (j < m && abs(matrix[i][j]) > eps) {
         for (int s = i+1; s < n; s++) {
            if (abs(matrix[i][j]) < eps) continue;
            for (int x = j; x < m; x++)
               swap(matrix[i][x], matrix[s][x]);
            break;
         }
         if (abs(matrix[i][j]) < eps) j++;
      }
      if (j >= m) break;
      for (int y = 0; y < n; y++) if (i != y) // 将0改为i是非对角阵
         for (int x = m-1; x >= j; x--)
            matrix[y][x] -= matrix[y][j] * matrix[i][x] / matrix[i][j];
      j++;
   }
   return min(i, j); // 返回矩阵的秩
}
```

5.2.2 矩阵求逆

```
int calc(int n){
  int d=1; memset(e,0,sizeof(e));
  for (int i=0; i<n; i++) e[i][i]=1;
  for (int i=0; i<n; i++){
    int p=i; while (!x[p][i] && p<n) p++;
    if (p>=n) return 0;
    for (int j=0; j<n; j++) swap(x[i][j],x[p][j]),swap(e[i][j],e[p][j]);
    int rev=Pow(x[i][i],mod-2); d=1LL*d*x[i][i]%mod;
    for (int j=0; j<n; j++) x[i][j]=1LL*x[i][j]*rev%mod,e[i][j]=1LL*e[i][j]*
        rev%mod;
    for (int j=0; j<n; j++) if (j!=i && x[j][i]) {
      int t=x[j][i];
      for (int k=0; k<n; k++)
        x[j][k]=(x[j][k]-1LL*t*x[i][k]%mod+mod)%mod,e[j][k]=(e[j][k]-1LL*t*e[
            i][k]%mod+mod)%mod;
    }
  }
  return d; // 返回矩阵的行列式
}
```

5.2.3 异或线性基

```
struct LinearBasis {
   lld b[63], nb[63], tot;
   void init() { tot = 0; memset(b, 0, sizeof(b)); }
   bool insert(lld x) {
      for (int i = 62; i >= 0; i--) if (x & (1LL<<i)) {
         if (!b[i]) return b[i] = x, true; else x ^= b[i]; } return x > 0; }
   lld max(lld res) {
      for (int i = 62; i >= 0; i--) res = max(res, res ^ b[i]); return res; }
   lld min(lld res) {
      for (int i = 0; i <= 62; i++) if (b[i]) res ^= b[i]; return res; }

   void rebuild() {
      memset(nb, 0, sizeof(nb));
      for (int i = 62; i >= 0; i--) for (int j = i-1; j >= 0; j--)
         if (b[i] & (1LL<<j)) b[i] ^= b[j];
      for (int i = 0; i <= 62; i++) if (b[i]) nb[tot++] = b[i];
   }

   lld kth_max(lld k) {
      lld res = 0;
      for (int i = 62; i >= 0; i--) if (k & (1LL<<i)) res ^= nb[i];
      return res;
   }
};
```

合并 将两个线性基合并是将其中一个线性基中的向量往另一个线性基中插入。

可线性表示 无法插入原来的线性基。

5.2.4 实数线性基

```
template <int N>
struct LinearBasis {
   double b[N][N]; bool v[N];
   void init() { memset(b, 0, sizeof(b)); memset(v, 0, sizeof(v)); }

   bool insert(double x[]) {
      for (int i = 0; i < N; i++) if (fabs(x[i]) > 1e-5) {
         if (v[i]) {
            double t = x[i] / b[i][i];
            for (int j = 0; j < N; j++) x[j] -= t * b[i][j];
         } else {
            for (int j = 0; j < N; j++) b[i][j] = x[j];
            return v[i] = true;
         }
      }
```

```
16        return false;
17    }
18 };
```

5.3 递推数列

5.3.1 线性递推

```
1  struct LinearRec {
2     lld bin[LOG][MAXN], first[MAXN], trans[MAXN], n;
3
4     void add(lld _c[], const lld a[], const lld b[]) {
5        static lld c[MAXN<<1];
6        for (int i = 0; i <= 2*n; i++) c[i] = 0;
7        for (int i = 0; i <= n; i++)
8           for (int j = 0; j <= n; j++)
9              c[i+j] = (c[i+j] + a[i] * b[j]) % MOD;
10       for (int i = n+n; i > n; i--) if (c[i])
11             for (int j = 0; j < n; j++)
12                c[i-j-1] = (c[i-j-1] + c[i] * trans[j]) % MOD;
13       for (int i = 0; i <= n; i++) _c[i] = c[i];
14    } // T(n) = O(n^2), 多项式乘法对特征方程取模可以 O(nlogn)
15
16    void prep(vector<lld> &First, vector<lld> &Trans) {
17       assert(First.size() >= Trans.size());
18       n = Trans.size();
19       for (int i = 0; i < n; i++)
20          first[i] = First[i], trans[i] = Trans[i];
21       bin[0][1] = 1; // make sure bin[0][other] == 0
22       for (int i = 1; i < LOG; i++)
23          add(bin[i], bin[i-1], bin[i-1]);
24    } // O(T(n)LOG) prep({a1,a2}, {2,1}) -> a[n]=2a[n-1]+a[n-2]
25
26    lld calc(lld k) {
27       static lld a[MAXN]; a[0] = 1; lld ans = 0;
28       for (int i = 1; i <= n; i++) a[i] = 0;
29       for (int i = 0; i < LOG; i++) if (k >> i & 1) add(a, a, bin[i]);
30       for (int i = 0; i < n; i++) ans = (ans + a[i+1] * first[i]) % MOD;
31       return ans;
32    } // O(T(n)logk)
33 };
```

5.3.2 特征多项式

```
1  vector<int> BerlekampMassey(vector<int> s) {
2     vector<int> C(1, 1), B(1, 1);
3     int L = 0, m = 1, b = 1;
```

```
    for (int n = 0; n < s.size(); n++) {
        lld d = 0;
        for (int i = 0; i <= L; i++)
            d = (d + 1LL * C[i] * s[n-i]) % MOD;
        if (d == 0) {
            ++m;
        } else if (2 * L <= n) {
            vector<int> T = C; lld c = MOD - d * fpow(b, MOD-2) % MOD;
            while (C.size() < B.size() + m) C.push_back(0);
            for (int i = 0; i < B.size(); i++)
                C[i+m] = (C[i+m] + c * B[i]) % MOD;
            L = n+1-L; B = T; b = d; m = 1;
        } else {
            lld c = MOD - d * fpow(b, MOD-2) % MOD;
            while (C.size() < B.size() + m) C.push_back(0);
            for (int i = 0; i < B.size(); i++)
                C[i+m] = (C[i+m] + c * B[i]) % MOD;
            ++m;
        }
    }
    return C;
} // In: {1, 1, 2, 3} Out: {1, MOD-1, MOD-1}, O(n^2)

lld gao(vector<int> a, lld n) {
    vector<int> c = BerlekampMassey(a); c.erase(c.begin());
    for (int i = 0; i < c.size(); i++) c[i] = (MOD-c[i])%MOD;
    static LinearRec lr; lr.prep(a, c); return lr.calc(n);
} // gao({a1, a2, ..}, k) = ak
```

5.3.3 循环节

斐波那契 对于一个正整数 n，我们求循环节长度方法如下：

1. 将 n 唯一分解 $n = {p_1}^{a_1}{p_1}^{a_2}\cdots{p_r}^{a_r}$
2. 计算模 p_i 的循环节长度，假设长度为 h_i
3. 计算模 ${p_i}^{a_i}$ 的循环节长度，为 $x_i = h_i{p_i}^{a_i-1}$
4. 计算模 n 的循环节长度 $L = \mathrm{lcm}(x_1, x_2, \cdots, x_r)$

当 $p \le 5$ 时，我们特殊判断 $h_2 = 3, h_3 = 8, h_5 = 20$。如果 5 是 p 的二次剩余，那么循环节长度是 $p-1$ 的因子，否则为 $2(p+1)$ 的因子。

证明非二次剩余过程中将 $\sqrt{5}$ 扩入 $\mathbb{Z}_p$ 成为 $\mathbb{Z}_p[\sqrt{5}]$，注意 $\mathrm{char}\,\mathbb{Z}_p[\sqrt{5}] = p$，然后对特征值进行运算。

5.3.4 齐肯多夫定理

齐肯多夫定理表示任何正整数都可以表示成若干个不连续的斐波那契数（不包括第一个斐波那契数）之和。这种和式称为齐肯多夫表述法。对于任何正整数，其齐肯多夫表述法都可以由贪心算法（即每次选出最大可能的斐波那契数）得到。

Pinary 数是一个仅由 0 和 1 组成的数，这个数不包含前导 0，且相邻位置不同时为 1，大小由长度和字典序决定，长度不同时，长度越小值越小，长度相同时，字典序越小值越小。

5.4 计算方法

5.4.1 二分零点

检查型二分 在区间 $[L,R]$ 寻找第一个不满足 *check* 的位置，不存在返回 $R+1$。

```
inline int find(int L, int R) {
   int ans = L-1;
   while (L <= R) {
      int mid = (L+R)/2;
      if (check(mid)) ans = max(ans, mid), L = mid+1;
      else R = mid-1;
   }
   return ans;
}

inline double find(double l, double r, double eps) {
   double ans = l;
   while (r - l > eps) {
      double m = (l+r)/2;
      if (check(m)) ans = m, l = m;
      else r = m;
   }
   return ans;
}
```

5.4.2 三分最大值

单峰函数 称 $1,2,3,6,4$ 是凸性序列，$5,4,4,3,1,2,4,6$ 是凹性序列。以下对凸性序列成立。

三分套三分 适合二元函数且单峰。

```
inline int find(int l, int r) {
   while (l < r-1) {
      int mid = (l+r)/2, mmid = (mid+r)/2;
      if (f(mid) > f(mmid)) r = mmid;
      else l = mid;
   }
   return f(l) > f(r) ? l : r;
}

inline double find(double l, double r) {
   while (r - l > eps) {
      double m1 = l+(r-l)/3, m2 = up-(r-l)/3;
      if (f(m1) <= f(m2)) l = m1;
      else r = m2;
```

```
15    }
16    return (l+r)/2;
17 }
```

5.4.3 simpson 积分

```
1  double simpson(double a, double b) {
2     double c = a + (b-a)/2;
3     return (F(a) + 4*F(c) + F(b))*(b-a)/6;
4  }
5
6  double asr(double a, double b, double eps, double A) {
7     double c = a + (b-a)/2, L = simpson(a,c), R = simpson(c, b);
8     if (fabs(L+R-A) <= 15*eps) return L+R+(L+R-A)/15.0;
9     return asr(a, c, eps/2, L) + asr(c, b, eps/2, R);
10 }
11
12 double asr(double a, double b, double eps) {
13    return asr(a, b, eps, simpson(a,b));
14 }
```

5.4.4 线性规划

标准单纯形 有 n 个实数变量 $x_1, x_2, \cdots, x_n$ 和 m 条约束，其中第 i 条约束形如 $\sum_{j=1}^{n} a_{ij}x_j \le b_i$，且变量需要满足 $x_j \ge 0$，指定 x_j 取值使得 $F = \sum_{j=1}^{n} c_j x_j$ 值最大。

等号限制 转换为一个 $\le$ 和一个 $\ge$。

最小值 目标函数系数取反，答案取反。

变量自身限制 $x \le a \Rightarrow x' = a - x \ge 0$，$x \ge a \Rightarrow x' = x - a \ge 0$，$x \in R \Rightarrow x_1 - x_2 \ge 0$。

对偶 Maximize $c^T x \to Ax \le b, x \ge 0$ 转化为 Minimize $y^T b \to y^T A \ge c^T, y \ge 0$

```
1  const int NR = 110;
2  const double eps = 1e-8, inf = 1e9;
3  int n, m, type, id[NR], tp[NR];
4  double a[NR][NR];
5
6  void pivot(int r, int c) {
7     swap(id[r+n], id[c]);
8     double t = -a[r][c], a[r][c] = -1;
9     for (int i = 0; i <= n; i++) a[r][i] /= t;
10    for (int i = 0; i <= m; i++) if (a[i][c] && r != i) {
11       t = a[i][c], a[i][c] = 0;
12       for (int j = 0; j <= n; j++) a[i][j] += t * a[r][j];
13    }
```

```
}

void solve() {
   double t;
   for (int i = 0; i <= n; i++) id[i] = i;
   while (true) {
      int i = 0, j = 0; double _w = -eps;
      for (int k = 1; k <= m; k++)
         if (a[k][0] < _w) _w = a[i=k][0];
      if (!i) break;
      for (int k = 1; k <= n; k++)
         if (a[i][k] > eps) { j = k; break; }
      if (!j) { printf("Infeasible\n"); return; }
      pivot(i, j);
   }
   while (true) {
      int i = 0, j = 0; double _w = eps;
      for (int k = 1; k <= n; k++)
         if (a[0][k] > _w) _w = a[0][j=k];
      if (!j) break; _w = inf;
      for (int k = 1; k <= m; k++)
         if (a[k][j] < -eps && (t = -a[k][0] / a[k][j]) > _w)
            _w = t, i = k;
      if (!i) { printf("Unbounded\n"); return; }
      pivot(i, j);
   }
   printf("%.9f\n", a[0][0]);
   for (int i = n+1; i <= n+m; i++) tp[id[i]] = i-n;
   for (int i = 1; i <= n; i++) printf("%.9f ", tp[i] ? a[tp[i]][0] : 0);
}

int main() {
   scanf("%d %d", &n, &m);
   for (int i = 1; i <= n; i++) scanf("%lf", &a[0][i]);
   for (int i = 1; i <= m; i++) {
      for (int j = 1; j <= n; j++)
         scanf("%lf", &a[i][j]), a[i][j] *= -1;
      scanf("%lf", &a[i][0]);
   }
   solve();
   return 0;
}
```

第六章　组合数学

6.1　简单计数

6.1.1　Lucas 定理

$$\binom{sp+q}{tp+r} \equiv \binom{s}{t}\binom{q}{r} \pmod{p}$$

6.1.2　拓展 Lucas 定理

先计算 $n!$ 对 p^k 取模，利用以下方法，然后使用中国剩余定理合并。

```
struct exLucas {
   int p; lld pk, fac[MAXP];

   void init(int _p, lld _pk) {
      p = _p, pk = _pk; fac[0] = fac[1] = 1;
      for (int i = 2; i < p; i++) fac[i] = (__int128)i * fac[i-1] % pk;
   }

   pair<lld,lld> facMod(lld n) {
      if (n < p) {
         return make_pair(fac[n], 0ll);
      } else {
         auto last = facMod(n/p); last.second += n/p;
         last.first = (__int128)last.first * fac[n%p] % pk;
         last.first = (__int128)last.first * fpow(fac[p-1],n/p,pk) % pk;
         return last;
      }
   }
};
```

6.1.3　第一类斯特林数

n 个不同物品构成 k 个圆排列。

$$s(n,k) = s(n-1,k-1) + (n-1)s(n-1,k)$$

组合型生成函数。计算时可以使用倍增 +FFT。

$$s_1(x) = \sum_{k=0}^{n} s(n,k)x^k = \prod_{i=0}^{n-1}(x+i)$$

6.1.4 第二类斯特林数

n 个物品划分成 m 个非空无区别集合的方法数。

$$S(n,k)=S(n-1,k-1)+kS(n-1,k)$$

生成函数为

$$G_k(x)=\frac{x^k}{(1-x)(1-2x)\cdots(1-kx)}$$

可以通过卷积加速计算。

$$S(n,m)=\frac{1}{m!}\sum_{k=0}^{m}(-1)^kC(m,k)(m-k)^n=\sum_{k=0}^{m}\frac{(-1)^k(m-k)^n}{k!(m-k)!}=\sum_{k+t=m}\frac{(-1)^k}{k!}\times\frac{t^n}{t!}$$

6.1.5 贝尔数

B_n 表示把 n 个带标号物品划分为若干不相交可空集合的方案数，也是第二类斯特林数前缀和。

$$B_{n+1}=\sum_{k=0}^{n}C(n,k)B_k$$

6.1.6 自然数幂和

$\sum_{i=1}^{n}i^k$ 使用拉格朗日插值，注意到插值点是连续的，考虑预处理阶乘和其逆元以达到 $O(k)$ 计算。

6.1.7 整数划分

将 n 表示成多个最大值不超过 k 的正整数和的方案数，或者将 n 拆分成不超过 k 个数的和的方案数。

$$B(n,k)=B(n-1,k-1)+B(n-k,k)$$

```
// T(n) = O(n^1.5), splitting n.
int f[732], g[200001];

void init(int n) {
   f[1] = 1, f[2] = 2, f[3] = 5, f[4] = 7, g[0] = 1;
   for (int i = 5; i < 731; i++) f[i] = 3 + 2 * f[i-2] - f[i-4];
   for (int i = 1; i <= n; i++)
      for (int j = 1; f[j] <= i; j++)
         if ((j+1)>>1&1) g[i] = (g[i] + g[i-f[j]]) % MOD;
         else g[i] = (g[i] - g[i-f[j]]) % MOD;
}
```

n	3	4	5	6	7
$p(n)$	3	5	7	11	15
n	8	16	24	32	42
$p(n)$	22	231	1575	8349	147273
n	50	64	72	80	96
$p(n)$	204226	1741630	5392783	15796476	118114304

6.1.8 错位排列

n 个元素放在 n 个编号位置，元素编号与位置编号各不对应的方案数 $D(n)$。

考虑，第一步，将元素 n 放在位置 k，有 $n-1$ 种方法；第二步，放编号为 k 的元素，放到位置 n 或其他位置，则有 $D_n=(n-1)(D_{n-2}+D_{n-1})$，其中 $D_1=0,D_2=1$。由二项式反演也可以得到通项公式。

6.1.9 卡特兰数

$$H_{n+1} = H_0H_n + H_1H_{n-1} + \cdots + H_nH_0 = \frac{C_{2n}^n}{n+1}$$

常见应用：括号配对方式、出栈次序、凸多边形三角划分、给定节点组成二叉搜索树、从 $(0,0)$ 到 (n,n) 满足路径上 $y \geq x$ 的方案数。

6.1.10 非降路径

从 $(0,0)$ 走到 (m,n) 不接触 $y=x$ 的非降路径数，其中 $m<n$，相当于 $(0,1) \to (m,n)$ 减去一定接触的路径数 $(1,0) \to (m,n)$，即方案数为 $C(m+n-1,m) - C(m+n-1,n)$。

从 $(0,0)$ 走到 (m,n) 不走到 $y=x$ 下方的非降路径数，其中 $m<n$，考虑 $(0,0)$ 关于 $y=x-1$ 的对称点 $(1,-1)$，则相当于 $(0,0) \to (m,n)$ 减去一定接触的路径数 $(1,-1) \to (m,n)$，即方案数为 $C(m+n,m) - C(m+n,m-1)$。

6.1.11 N 个小球放进 M 个盒子

n 球	m 盒	空	方案数	备注
相同	相同	不可空	$B(n,m)$	整数划分
相同	相同	可空	$\sum B(n,i)$	上式前缀和
相同	不同	不可空	$C(n-1,m-1)$	隔板法
相同	不同	可空	$C(n+m-1,m-1)$	预留隔板
不同	相同	不可空	$S(n,m)$	第二类斯特林
不同	相同	可空	$\sum S(n,i)$	上式前缀和
不同	不同	不可空	$m!S(n,m)$	第二类斯特林
不同	不同	可空	m^n	上式前缀和

6.2 容斥原理

6.2.1 莫比乌斯反演

$$f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu\left(\frac{n}{d}\right) f(d)$$

$$f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) f(d)$$

例 通常考虑将 $[\gcd = d]$ 转化为 $[d|\gcd]$ 来设计反演的函数。

6.2.2 二项式反演

$$f(n) = \sum_{k=0}^{n} \binom{n}{k} g(k) \Leftrightarrow g(n) = \sum_{k=0}^{n} (-1)^{n-k} \binom{n}{k} f(k)$$

例 考虑可空集合与 k 元非空集合之间的组合关系可以得到前者的式子。

6.2.3 单位根反演

$$[n|k] = \frac{1}{n}\sum_{i=0}^{n-1}\omega_n^{ki}$$

例 已知 $f(x)=\sum_{k=0}^{n}a_kx^k$，求 $\sum a_{4i}$。可利用单位根反演变为 $\frac{1}{n}(f(\omega_4^0)+f(\omega_4^1)+f(\omega_4^2)+f(\omega_4^3))$。

例 求 $\sum_{i=0}^{n/k}C_n^{ik}F_{ik}$。设 $f(x)=(E+xA)^n$，其中 A 为斐波那契数列转移矩阵，则单位根反演即可。

6.2.4 斯特林反演

$$f(n)=\sum_{k=0}^{n}\left\{{n\atop k}\right\}g(k)\Leftrightarrow g(n)=\sum_{k=0}^{n}(-1)^{n-k}\left[{n\atop k}\right]f(k)$$

6.2.5 Min-Max 容斥

$$\max(S)=\sum_{T\subseteq S}(-1)^{|T|-1}\min(T)\Leftrightarrow\min(S)=\sum_{T\subseteq S}(-1)^{|T|-1}\max(T)$$

$$k^{th}\max(S)=\sum_{T\subseteq S}(-1)^{|T|-k}\binom{|T|-1}{k-1}\min(T)$$

例 在期望的意义下依旧成立。

6.2.6 子集反演

$$f(S)=\sum_{T\subseteq S}g(T)\Leftrightarrow g(S)=\sum_{T\subseteq S}(-1)^{|S|-|T|}f(T)$$

例 求 $c_r=\sum_{\gcd(p,q)=r}a_pb_q$，不妨构造 $c'_r=\sum_{d|r}c_d$ 的变换对。

高维前缀和 考虑每个维度对答案是如何产生贡献的，先由 g 变为 f，然后相乘，变回 g。

```
void FWT(int a[], int n, int DFT) {
   for (int d = 1; d < n; d <<= 1)
      for (int m = d<<1, i = 0; i < n; i += m)
         for (int j = 0; j < d; j++)
            // and: a[i+j] += DFT * a[i+j+d];
            // or: a[i+j+d] += DFT * a[i+j];
}

// 求数组gcd，像求数组FWT那样将r的值存入g[id[r]]
// 设 lcm=\prod pi^ei，则w为总个数，fc为ei，fp为pi
int qp[17], fc[17], dcn, w;
lld g[MAXN], fp[17], di[MAXN];
unordered_map<lld, int> did;

void prepare() {
   for (int i = qp[0] = 1; i < w; i++)
      qp[i] = qp[i-1] * (fc[i-1] + 1);
```

```
   dcn = qp[w-1] * (fc[w-1] + 1);
   genFac(0, 1, 0);
}

void genFac(int wi, lld fac, int ids) {
   if (wi == w) {
      di[ids] = fac; did[fac] = ids;
   } else {
      lld pr = 1;
      for (int i = 0; i <= fc[wi]; i++, pr *= fp[wi])
         genFac(wi+1, fac*pr, ids+i*qp[wi]);
   }
} // 给因数编码

void conv() {
   for (int i = 0; i < w; i++)
      for (int j = dcn-1; j >= 0; j--)
         if (j / qp[i] % (fc[i]+1) != fc[i])
            g[j] += g[j+qp[i]];
   for (int i = 0; i < dcn; i++)
      g[i] *= g[i];
   for (int i = 0; i < w; i++)
      for (int j = 0; j < dcn; j++)
         if (j / qp[i] % (fc[i]+1) != fc[i])
            g[j] -= g[j+qp[i]];
} // 卷积 g=g*g
```

6.2.7 K 进制异或卷积

考虑将坐标看为很多个 K 进制数字，对每一个数位进行模 K 加法 $\oplus$，则想求 $c_k = \sum_{i\oplus j=k} a_i b_j$ 可以进行卷积 $c'_i = a'_i b'_i$。对每一个数位进行 K 阶 DFT，即预处理出 $\omega_K^i (i = 0, 1, \cdots, K-1)$。$K = 2$ 的情况即为 bitwise-XOR。注意逆变换和 FFT 一样除上 N。

```
void FWT(int a[], int n, int DFT) {
   if (DFT == -1) DFT = 2;
   for (int d = 1; d < n; d <<= 1)
      for (int m = d<<1, i = 0; i < n; i += m)
         for (int j = 0, x, y; j < d; j++)
            x = a[i+j], y = a[i+j+d],
            a[i+j] = (x+y)/DFT, a[i+j+d] = (x-y)/DFT;
} // K = 2

const int MOD = 1e9+9, W1 = 115381398, W2 = 884618610;
void FWT(int A[], int MAXN, int inv) {
   for (int m = 3, m3 = 1; m <= MAXN; m *= 3, m3 *= 3)
      for (int k = 0; k < MAXN; k += m)
         for (int i = 0, a, b, c; i < m3; i++)
            a = A[k+i], b = A[k+i+m3], c = A[k+i+m3*2],
```

```
16              A[k+i] = (0ll + a + b + c) % MOD,
17              A[k+i+m3] = (a + 1ll * b * W1 + 1ll * c * W2) % MOD,
18              A[k+i+m3*2] = (a + 1ll * b * W2 + 1ll * c * W1) % MOD;
19      if (inv == -1)
20          for (int i = 0, INVN = fpow(MAXN, MOD-2); i < MAXN; i++)
21              A[i] = 1ll * A[i] * INVN % MOD;
22  } // K = 3
```

6.3 生成函数

6.3.1 组合型生成函数

$$f(x)=\sum_{k=0}^{\infty}a_kx^k$$

例 拆分 n，1 至多 3 次，2 至少 2 次，3 无限制，求方案数。

$$G(x)=\left(1+x+x^2+x^3\right)\left(x^4+x^6+x^8+\cdots\right)\left(1+x^3+x^6+\cdots\right)$$

结果中 x^n 系数即为所求组合数。

6.3.2 组合型生成函数

$$f(x)=\sum_{k=0}^{\infty}a_k\frac{x^k}{k!}$$

例 求 A,B,C,D 组成的长度为 N 的字符串的方案数，其中 A,C 为偶数个。构造函数

$$G(x)=\left(1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots\right)^2\left(1+\frac{x^2}{2!}+\frac{x^4}{4!}+\frac{x^6}{6!}+\cdots\right)^2=\frac{e^{4x}+2e^{2x}+1}{4}$$

则解为 $x^n/n!$ 的系数。可以得到答案为 $2^{n-1}+4^{n-1}$。

6.3.3 Burnside 引理

设 G 是 n 元集 $S=\{1,2,\cdots,n\}$ 上的置换群 $G=\{\sigma_1,\sigma_2,\cdots,\sigma_m\}$，把每个 σ_i 都表示成不相杂轮换的乘积，则 G 在 S 上诱导出的等价关系将 S 划分为不同等价类的个数为

$$L=\frac{1}{|G|}\sum_{j=1}^{m}\lambda_1(\sigma_j)$$

其中 $\lambda_1(\sigma)$ 为置换 σ 中的不动点个数。

注 S 是所有染颜色的方案，最后答案意思就是有几种不同的染色方案。

6.3.4 Polya 定理

设 H 是 n 个对象上的一个置换群，用 m 种颜色对 n 个对象涂色，一个对象可以任意涂色，则在置换群 H 作用下不等价的方案数为

$$L=\frac{1}{|H|}\sum_{\tau\in H}m^{\lambda(\tau)}$$

其中 $\lambda(\tau)$ 表示置换 τ 表示为不相杂轮换乘积的形式中轮换的个数。

例 置换群 $H'=\{(1)(2)(3),(123),(132),(1)(23),(2)(13),(3)(12)\}$ 和三个物品涂三种颜色的不等价方案数为 $L'=\frac{1}{6}\left(3^2+2\times 3^1+3\times 3^2\right)$。记颜色 $\{r,g,b\}$，则其生成函数 $L(r,g,b)=\frac{1}{6}[(r+g+b)^3+2(r^3+b^3+g^3)+3(r+g+b)(r^2+g^2+b^2)]$，其中 $r^xg^yb^z$ 前系数表示对应染色方案个数时的方案数。

例 n 个点的圆环对称和旋转是等价的。可以看出其置换群有 $2n$ 个元素。其中 n 个元素是由旋转得到的，由初始状态旋转 k 个单位，得到的不相杂轮换数为 $\gcd(k,n)$ 个。另外 n 个元素是轴对称得到的，当奇数时，不相杂轮换为 $\frac{n+1}{2}$ 个，当偶数时，经过珠子的 $\frac{n}{2}$ 个置换的不相杂轮换 $\frac{n}{2}$ 个，不经过珠子的 $\frac{n}{2}$ 个置换不相杂轮换有 $\frac{n}{2}+1$ 个。

例 三叉树个数生成函数 $A(x)$。考虑从根节点向下伸展节点：恒同变换对子树没有要求，其生成函数为 $A(x)^3$；一对置换时一个没有要求，另外两颗子树要相同，其生成函数为 $A(x)A(x^2)$；三轮换则三颗子树都相同，其生成函数为 $A(x^3)$。则最后有 $A(x)=1+x\frac{A(x)^3+3A(x)A(x^2)+2A(x^3)}{6}$。

6.3.5 棋盘多项式

棋盘 C 上棋子不能布置在同一行或者同一列，$r_k(C)$ 为 k 个棋子布到棋盘 C 上的方案数。

$$R(C)=\sum_{k=0}^{n}r_k(C)x^k$$

记 $C_{(i)}$ 为去掉某个棋子所在一行一列的棋盘，$C_{(e)}$ 为仅去掉该棋子的棋盘，则有

$$R(C)=xR(C_{(i)})+R(C_{(e)})$$

带禁区的排列 设整个棋盘为 $n\times n$，禁区棋盘的多项式为 $R(x)$，则可用排列数为

$$N=n!-r_1(n-1)!+r_2(n-2)!-\cdots+(-1)^nr_n$$

6.4 常见问题

6.4.1 约瑟夫环

```
lld josephus(lld n, lld m, lld k) {
   if (k == 1) return m;
   lld id = 0;
   for (lld i = n-m+1; i <= n; i++) {
      lld x = min((i-id-1)/(k-1),n-i-1);
      if (x > 0) i += x, id += x * k;
      id = (id + k-1) % i + 1;
   }
   return id;
} // n个人，报数到k出局，第m个出局人的位置

lld invJosephus(lld n, lld m, lld k) {
   if (k == 1) return m;
   lld id = n;
   for (lld pos = (m+n-k%n)%n+1; pos > 1; ) {
      if (pos <= k) --id, pos = ((pos-k)%id+id-1)%id+1;
      else id -= (pos-1)/k, pos -= (pos-1)/k*k;
   }
   return n - id + 1;
```

```
} // n个人，报数到k出局，第m个位置人的出局时间
```

6.4.2 星期计算

$$w = \left\lfloor \frac{c}{4} \right\rfloor - 2c + y + \left\lfloor \frac{y}{4} \right\rfloor + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + d - 1 \mod 7$$

w 是星期对 7 取模；c 是年份除以 100 向下取整；y 是年份模 100 的正余数；m 是月份，注意将某年的 1、2 月转换为上一年的 13、14 月，其他数字也要调整；d 是日。注意只适用于 1582 年 10 月 15 日之后。

6.4.3 康托展开

将某个排列 $(p_1, p_2, \cdots, p_n)$ 和其字典序 X 相互转换。a_i 表示第 p_i 后面小于 p_i 的数字个数。

$$X = a_n(n-1)! + a_{n-1}(n-2)! + \cdots + a_1 0!$$

6.4.4 Jacobi's Four Square Theorem

设 $a^2+b^2+c^2+d^2 = n$ 的自然数解个数为 $r4(n)$，$d(n)$ 为 n 的约数和，则 n 为奇数时，$r4(n) = 8d(n)$，否则 $r4(n) = 24d(k)$，k 是 n 去除所有 2 以后的结果。

6.4.5 15-谜

记 P_k 为编号 k 的牌在初始状态中的位置，P_{16} 为空格的位置，$L_i = \sum_{j=1}^{i-1}[P_j > P_i]$，$X = ((P_i-1)\%4 + (P_i-1)/4)\%2$，则当且仅当 $X + \sum_{i=1}^{16} L_i$ 是偶数的时候目标状态可由此状态到达。

6.4.6 切比雪夫多项式

我们称 $T_n(\cos x) = \cos nx$ 的多项式 $T(x)$ 为第一类切比雪夫多项式。有如下递推关系式成立。

$$T_0(x) = 1; \quad T_1(x) = x; \quad T_{n+1} = 2xT_n(x) - T_{n-1}(x)$$

有通项公式

$$T_n(x) = n\sum_{k=0}^{n}(-2)^k \frac{(n+k-1)!}{(n-k)!(2k)!}(1-x)^k$$

第七章　博弈论

7.1　常见博弈游戏

7.1.1　Bash Game

游戏规则　一堆石子共 n 个，两个玩家轮流拿，每次至少 1 个至多 m 个，最后取光者胜。

必胜策略　如果 $n=(m+1)k$ 则先手必败，否则先手必胜。

构造方案　每次都使得两轮数量和为 $m+1$ 即可构造必胜策略。

7.1.2　Nim Game

游戏规则　n 堆石子，两个玩家轮流操作，每次任选一非空堆拿走至少一颗石子，若无法操作则失败。

必胜策略　如果 n 堆石子个数异或和为 0 则先手必败，否则必胜。

构造方案　假设初始异或和为 0，先手选择一堆 x，拿完剩余 y，剩余堆的异或和变为 $x\oplus y$，那么一定可以找到一堆 a 个石子，a 二进制中最高位等于 $x\oplus y$ 的最高位，即 $a>a\oplus x\oplus y$，那么就可以将 a 变为 $a\oplus x\oplus y$，则异或和变为 0，所以先手必败。若初始异或和为 $t>0$，则可按上述后手操作将异或和转为 0，从而必胜。

7.1.3　Wythoff Game

游戏规则　两堆石子分别 a,b 颗，两个玩家轮流操作，从一堆拿任意非 0 颗，或者从两堆都拿相同任意非 0 颗，拿走最后一颗获胜。

Betty Theorem　设 a,b 是正无理数且 $1/a+1/b=1$。记 $P=\{\lfloor na\rfloor\mid n\in\mathbb{Z}^+\}$，$Q=\{\lfloor nb\rfloor\mid n\in\mathbb{Z}^+\}$，则 P 与 Q 是 $\mathbb{N}^+$ 的一个划分，即 $P\bigcap Q=\emptyset$ 且 $P\bigcup Q=\mathbb{N}^+$。

必胜条件　先手奇异局势则必赢。判断 (a,b) 是否为奇异局势：$[\lfloor(a-b)\frac{\sqrt{5}+1}{2}\rfloor=b]$。

构造方案　奇异局势为 $P_0=(0,0),P_1=(1,2),P_2=(3,5),P_3=(4,7),P_4=(6,10),P_5=(8,13),P_6=(9,15),P_7=(11,18),P_8=(12,20)\cdots$ 其中 $P_{k,1}=\operatorname{mex}(P_0,\cdots,P_{k-1})=\lfloor k\phi\rfloor,P_{k,2}=P_{k,1}+k,\phi=\frac{\sqrt{5}+1}{2}$，$\phi$ 可由 Betty 定理解方程 $\frac{1}{\phi}+\frac{1}{\phi+1}=1$ 得到。可以发现每个自然数都出现在这个局势列表里，任意操作都可以将奇异局势变为非奇异局势，采用适当的方案可以将非奇异局势变为奇异局势。假设目前面对 (a,b)，若 $b=a$ 则变为 $(0,0)$；若 $a=P_{k,1},b>P_{k,2}$，则取走 $b-P_{k,2}$ 个物品；若 $a=P_{k,1},b<P_{k,2}$，则同时从两堆中取走 $a-P_{b-a,1}$ 个；若 $a>P_{k,1},b=P_{k,2}$，则从第一堆中拿走 $a-P_{k,1}$；若 $a<P_{k,1},b=P_{k,2}$，则在 $a=P_{n,1}(n<k)$ 时从第二堆中拿走 $b-P_{n,2}$ 个，在 $a=P_{n,2}(n<k)$ 时从第二堆中拿走 $b-P_{n,1}$ 即可。

```cpp
bool Wythoff(lld a, lld b) {
   const lld mod = 1e9, r0 = 618033988, r1 = 749894848, r2 = 204586834;
   if (a < b) swap(a, b); lld tmp = a - b, low = tmp % mod, hiw = tmp / mod;
   lld sum = low * r2; sum = hiw * r2 + low * r1 + sum / mod;
   sum = hiw * r1 + low * r0 + sum / mod; sum = tmp + hiw * r0 + sum / mod;
   return sum == b;
} // returns whether first wins
```

7.1.4 Eculid Game

游戏规则 两堆石子 (a,b)，不妨设 $a \geq b$，两个玩家轮流操作，每次只能 a 堆中拿去 b 的倍数个，有一个数为 0 时失败。

必胜条件 若 $a > 2b$ 或 $a\%b = 0$ 则先手必胜，否则胜负与 $(b, a-b)$ 相反。

构造方案 若 $a - b < b$ 时只有这种可能，听天由命；若 $a - b > b$，则从 a 中拿走 x 个 b，若 $(a - xb, b)$ 必败，则拿走 x 个 b，否则若 $(a - xb, b)$ 必胜，则拿走 $x - 1$ 个 b。

7.1.5 Imitate Game

游戏规则 n 个棋子围成一圈，两个玩家轮流操作，从中取走一个或两个棋子，不过取两个时必须是连续的棋子，棋子取走后留下空位，相隔空位的棋子不连续，拿走最后一个棋子的获胜。

必胜条件 当 $n \geq 3$ 时，先手必败。

构造方案 B 模仿 A 的选择，拿走对称位置相同个数，一定能保证自己拿走最后一个。

7.1.6 Fibbonacci Game

游戏规则 一堆 n 个石子，先手不能在第一次把所有石子取完，之后每次可以取的石子介于 1 和对手刚取的石子数的 2 倍之间。约定取走最后一个石子的人为赢家。

必胜条件 先手必胜，当且仅当 n 不为斐波那契数。

构造方案 先将 n 表示为齐肯多夫表示法，即 $n = F_{a_1} + F_{a_2} + \cdots + F_{a_p}$，先手先取完 F_{a_p}，则剩余部分被分解为若干子游戏，后手只能取 $F_{a_{p-1}}$ 这一堆，且不能一次取完，且先手一定可以取得 $F_{a_{p-1}}$ 的最后一颗石子。现在讨论 F_{k+1} 这一堆的先手必败做法。若先手取得了 $y \geq F_{k-1}$ 颗石子，则后手一定可以一次性拿完 F_k 颗石子；若先手取得了 $y \geq F_{k-1}/3$ 颗石子，则后手一定可以一次性拿完 $F_{k-1} - y$ 颗石子，并且可以证明 $\frac{2}{3}F_{k-1} < \frac{1}{2}F_k$，即先手无法一次性拿完 F_k 中的石子，重新转换成了原局面；否则又转换为了后手必胜。

7.1.7 Chocolate Game

游戏规则 有数 $1 - n$，每次选一个数划去它所有的约数，没有数则失败。

必胜条件 先手必胜，先手只要考虑拿不拿 1。

7.2 博弈定理

7.2.1 SG 定理

SG 函数 对于任意状态 x，定义 $SG(u) = \text{mex}_{(u,v)\in G}\{SG(v)\}$ 为此状态的 SG 函数。$SG(x) = 0$ 当且仅当 x 为必败点（出度为 0）时。例如，一堆取石子游戏有 $SG(x) = x$。

判定条件 整个游戏的 SG 函数是每个子游戏 SG 函数的异或和。先手必败当且仅当 SG 值为 0。

7.2.2 SJ 定理

Anti-SG Game DAG 上没有出度的点为胜利状态，其他定义与一般游戏相同。

判定条件 先手必胜，当且仅当以下两个条件同时成立或同时不成立：合并的 SG 值为 0，所有游戏的 SG 值不超过 1。

7.2.3 树上删边

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，不与根相连的部分将被移走。谁无法移动谁输。

叶节点的 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。

7.2.4 无向图删边

Fusion Principle 在无向图中，去掉所有偶环，将所有奇环变为长度为 1 的链，不会影响图的 SG 值。

第八章　计算几何

8.1　平面几何

8.1.1　基础运算函数

```
const double eps = 1e-8, inf = 1e20;
const double pi = acos(-1.0);
inline int sgn(double x) { return x > eps ? 1 : x < -eps ? -1 : 0; }
inline double sqr(double x) { return x * x; }

struct Point {
   double x, y;
   Point(double _x = 0, double _y = 0) { x = _x, y = _y; }
   bool operator==(Point b) const { return sgn(x-b.x)==0 && sgn(y-b.y)==0; }
   bool operator<(Point b) const { return sgn(x-b.x)==0?sgn(y-b.y)<0:x<b.x; }
   Point operator+(Point b) const { return Point(x+b.x, y+b.y); }
   Point operator-(Point b) const { return Point(x-b.x, y-b.y); }
   Point operator*(double k) const { return Point(x*k, y*k); }
   Point operator/(double k) const { return Point(x/k, y/k); }
   double operator^(Point b) const { return x * b.y - y * b.x; }
   double operator*(Point b) const { return x * b.x + y * b.y; }
   double length() const { return hypot(x, y); }
   double len2() const { return x * x + y * y; }
   Point rotateLeft() const { return Point(-y, x); }
   Point rotateRight() const { return Point(y, -x); }
   Point trunc(double r) const { double l = length(); if (!sgn(l)) return *
       this; return Point(x*r, y*r); }
};

struct Line {
   Point s, e;
   Line(Point _s, Point _e) { s = _s, e = _e; }
   Line(Point p, double dx, double dy) { s = p; e = s + Point(dx, dy); }
   double length() const { return (e-s).length(); }
   Line(double a, double b, double c) {
      if (sgn(a) == 0) s = Point(0,-c/b), e = Point(1,-c/b);
      else if (sgn(b) == 0) s = Point(-c/a, 0), e = Point(-c/a, 1);
      else s = Point(0, -c/b), e = Point(1, (-c-a)/b);
   } // Ax+By+C=0
};
```

```
35
36 struct Circle {
37    Point p; double r;
38    Circle(Point c, double rad) { p = c, r = rad; }
39    Circle(double _x, double _y, double _r) { p.x = _x, p.y = _y, r = _r; }
40 };
```

8.1.2 点与点关系

```
1 Point rotate(Point p, double arg) {
2    double c = cos(arg), s = sin(arg);
3    return Point(p.x * c - p.y * s, p.x * s + p.y * c);
4 } // 逆时针
5
6 double rad(Point a, Point p, Point b) {
7    return fabs(atan2(fabs((a-p)^(b-p)), (a-p)*(b-p)));
8 } // 角度 <APB
```

8.1.3 点与线关系

```
1  int pointOnLine(Point p, Line l) {
2     return sgn((p-l.s)^(l.e-l.s));
3  } // 点p与直线关系：-1左侧，0直线上，1右侧
4
5  bool pointOnSeg(Point p, Line s) {
6     return sgn((p-s.s)^(s.e-s.s)) == 0 && sgn((p-s.s)*(p-s.e)) <= 0;
7  } // 点p是否在线段s上
8
9  double disPointToLine(Point p, Line l) {
10    return fabs((p-l.s)^(l.e-l.s))/l.length();
11 } // 点p到直线l的距离
12
13 double disPointToSeg(Point p, Line s) {
14    if (sgn((p-s.s)*(s.e-s.s))<0 || sgn((p-s.e)*(s.s-s.e))<0)
15       return min((p-s.s).length(), (p-s.e).length());
16    return disPointToLine(p, s);
17 } // 点p到线段s的距离
18
19 Point project(Point p, Line l) {
20    return l.s + (((l.e-l.s)*((l.e-l.s)*(p-l.s)))/((l.e-l.s).len2()));
21 } // 将点p投影到直线l上
22
23 Point symmetryPoint(Point p, Line l) {
24    Point q = project(p, l);
25    return Point(2*q.x-p.x, 2*q.y-p.y);
26 } // 点p关于直线的对称点
```

8.1.4 线与线关系

```
bool parallel(Line u, Line v) {
   return sgn((u.e-u.s)^(v.e-v.s))==0;
} // 直线u与v是否平行

int lineCrossLine(Line u, Line v) {
   return parallel(u, v) ? pointOnLine(u.s, v) == 0 : 2;
} // 直线和直线的关系：0平行，1相同，2相交

Point crossPoint(Line u, Line v) {
   double a1 = (v.e-v.s)^(u.s-v.s), a2 = (v.e-v.s)^(u.e-v.s);
   return Point((u.s.x*a2-u.e.x*a1)/(a2-a1), (u.s.y*a2-u.e.y*a1)/(a2-a1));
} // 两条直线的交点，假定直线不平行

int segCrossSeg(Line u, Line v) {
   int d1 = sgn((u.e-u.s)^(v.s-u.s));
   int d2 = sgn((u.e-u.s)^(v.e-u.s));
   int d3 = sgn((v.e-v.s)^(u.s-v.s));
   int d4 = sgn((v.e-v.s)^(u.e-v.s));
   if ((d1^d2)==-2 && (d3^d4)==-2) return 2;
   return (d1==0 && sgn((v.s-u.s)*(v.s-u.e))<=0)
      || (d2==0 && sgn((v.e-u.s)*(v.e-u.e))<=0)
      || (d3==0 && sgn((u.s-v.s)*(u.s-v.e))<=0)
      || (d4==0 && sgn((u.e-v.s)*(u.e-v.e))<=0);
} // 线段和线段的关系：2规范相交，1有重合点，0不相交

double disSegToSeg(Line u, Line v) {
   return min(min(disPointToSeg(v.s, u), disPointToSeg(v.e, u)),
         min(disPointToSeg(u.s, v), disPointToSeg(u.e, v)));
} // 两条不相交线段间的距离
```

8.1.5 点与圆关系

```
Circle triangleOutter(Point a, Point b, Point c) {
   Line u((a+b)/2, ((a+b)/2)+((b-a).rotateLeft()));
   Line v((b+c)/2, ((b+c)/2)+((c-b).rotateLeft()));
   Point p(crossPoint(u, v)); return Circle(p, (a-p).length());
} // 三角形外接圆

Circle triangleInner(Point a, Point b, Point c) {
   double m = atan2(b.y-a.y,b.x-a.x), n = atan2(c.y-a.y,c.x-a.x);
   Line u(a, a + Point(cos((n+m)/2), sin(n+m)/2));
   m = atan2(a.y-b.y,a.x-b.x), n = atan2(c.y-b.y,c.x-b.x);
   Line v(b, b + Point(cos((n+m)/2), sin(n+m)/2));
   Point p(crossPoint(u, v)); return Circle(p, disPointToSeg(p, Line(a,b)));
} // 三角形内切圆

```

```
int pointWhereCircle(Point b, Circle c) {
   return sgn((b-c.p).length() - c.r);
} // 点和圆的关系：-1内部，0圆上，1圆外

int tangentLine(Circle c, Point q, Line &u, Line &v) {
   int x = pointWhereCircle(q, c);
   if (x == -1) return 0;
   if (x == 0) return u = Line(q, q + (q-c.p).rotateLeft()), v = u, 1;
   double d = (q-c.p).length(), l = c.r*c.r/d, h = sqrt(c.r*c.r-l*l);
   u = Line(q, c.p + (q-c.p).trunc(l) + (q-c.p).rotateLeft().trunc(h));
   v = Line(q, c.p + (q-c.p).trunc(l) + (q-c.p).rotateRight().trunc(h));
   return 2;
} // 求经过点q的切线，返回条数
```

8.1.6 线与圆关系

```
int circleRelateLine(Circle u, Line v) {
   return 1 - sgn(disPointToLine(u.p, v) - u.r);
} // 圆和直线的交点个数

int circleCrossLine(Circle u, Line v, Point &p1, Point &p2) {
   if (!circleRelateLine(u, v)) return 0;
   Point a = project(u.p, v);
   double d = sqrt(u.r * u.r - sqr(disPointToLine(u.p, v)));
   if (sgn(d) == 0) return p1 = p2 = a, 1;
   Point stdd = (v.e - v.s).trunc(d);
   return p1 = a + stdd, p2 = a - stdd, 2;
} // 圆和直线的交点，返回个数

double areaCircleTriangle(Circle u, Point a, Point b) {
   if (sgn((u.p-a)^(u.p-b)) == 0) return 0.0;
   Point q[5]; int len = 0; q[len++] = a;
   Line l(a, b); Point p1, p2;
   if (circleCrossLine(u, l, q[1], q[2]) == 2) {
      if (sgn((a - q[1]) * (b - q[1])) < 0) q[len++] = q[1];
      if (sgn((a - q[2]) * (b - q[2])) < 0) q[len++] = q[2];
   }
   q[len++] = b;
   if (len == 4 && sgn((q[0] - q[1]) * (q[2] - q[1])) > 0) swap(q[1], q[2]);
   double res = 0;
   for (int i = 0; i < len-1; i++)
      if (pointWhereCircle(q[i],u) == 1 || pointWhereCircle(q[i+1],u) == 1)
         res += u.r * u.r * fabs(atan2((q[i] - u.p) ^ (q[i+1] - u.p), (q[i] -
            u.p) * (q[i+1] - u.p))) / 2;
      else res += fabs((q[i] - u.p) ^ (q[i+1] - u.p))/2;
   return res;
} // 求圆与三角形PAB相交的面积
```

8.1.7 圆与圆关系

```
int circleRelateCircle(Circle u, Circle v) {
   double d = (u.p - v.p).length(); int s = sgn(d - u.r - v.r);
   return s >= 0 ? s + 4 : sgn(d - fabs(u.r - v.r)) + 2;
} // 圆与圆关系：5相离，4外切，3相交，2内切，1内含

int circleCrossCircle(Circle u, Circle v, Point &p1, Point &p2) {
   int rel = circleRelateCircle(u, v);
   if (rel == 1 || rel == 5) return 0;
   double d = (u.p-v.p).length(), l = (d*d+u.r*u.r-v.r*v.r)/(2*d);
   double h = sqrt(u.r * u.r - l * l);
   Point tmp = u.p + (v.p - u.p).trunc(l);
   p1 = tmp + ((v.p - u.p).rotateLeft().trunc(h));
   p2 = tmp + ((v.p - u.p).rotateRight().trunc(h));
   return rel == 3 ? 2 : 1;
} // 求两圆交点，返回个数

double areaCircleCircle(Circle u, Circle v) {
   int rel = circleRelateCircle(u, v);
   if (rel >= 4) return 0.0;
   if (rel <= 2) return min(u.area(), v.area());
   double d = (u.p-v.p).length(), hf = (u.r+v.r+d)/2.0;
   double ss = 2*sqrt(hf*(hf-u.r)*(hf-v.r)*(hf-d));
   double a1 = acos((u.r*u.r+d*d-v.r*v.r)/(2.0*u.r*d))*u.r*u.r;
   double a2 = acos((v.r*v.r+d*d-u.r*u.r)/(2.0*v.r*d))*v.r*v.r;
   return a1 + a2 - ss;
} // 求两圆相交的面积
```

8.2 平面凸包

8.2.1 多边形

```
struct Polygon {
   int n; Point p[MAXP];
   inline Point& operator[](int i) { return p[i]; }
   inline Point operator[](int i) const { return p[i]; }

   void norm() {
      Point mi = p[0];
      for (int i = 1; i < n; i++) mi = min(mi, p[i]);
      sort(p, p + n, [mi] (Point a, Point b) -> bool {
         int d = sgn((a - mi) ^ (b - mi));
         return d == 0 ? sgn((mi-a).length() - (mi-b).length()) < 0 : d > 0;
      }); // 极角排序
   }

```

```
15    bool isConvex() {
16       bool s[3] = { false, false, false };
17       for (int i = 0; i < n; i++) {
18          int j = (i+1)%n, k = (j+1)%n;
19          s[sgn((p[j]-p[i])^(p[k]-p[i]))+1] = true;
20          if (s[0] && s[2]) return false;
21       }
22       return true;
23    }
24 };
25 
26 void Graham(Polygon &dest, Polygon &src) {
27    int &top = dest.n, n = src.n;
28    Point *S = dest.p, *p = src.p;
29    src.norm(); top = 0;
30    if (n == 1) { top = 1, S[0] = p[0]; return; }
31    top = 2, S[0] = p[0], S[1] = p[1];
32    for (int i = 2; i < n; S[top++] = p[i], i++)
33       while (top > 1 && sgn((S[top-1]-S[top-2])^(p[i]-S[top-2])) <= 0) top--;
34    if (top == 2 && S[0] == S[1]) top--;
35 } // 获得凸包
36 
37 void Andrew(Polygon &dest, Polygon &src, int TT) {
38    int &top = dest.n, n = src.n; top = n;
39    Point *S = dest.p, *p = src.p; sort(p, p+n);
40    for (int i = 0; i < min(n,2); i++) S[i] = p[i];
41    if (top == 2 && S[0] == S[1]) top--;
42    if (n <= 2) return; top = 1;
43    for (int i = 2; i < n; S[++top] = p[i], i++)
44       while (top && sgn((S[top]-p[i])^(S[top-1]-p[i])) <= TT) top--;
45    int tmp = top; S[++top] = p[n-2];
46    for (int i = n-3; i >= 0; S[++top] = p[i], i--)
47       while (top != tmp && sgn((S[top]-p[i])^(S[top-1]-p[i])) <= TT) top--;
48    if (top == 2 && S[0] == S[1]) top--;
49    dest.norm();
50 } // 获得凸包，如果边上没有冗余点则TT=0，否则TT=-1
```

8.2.2 与点线圆的关系

```
1  int pointRelatePolygon(Point q, const Polygon &p) {
2     int cnt = 0;
3     for (int i = 0, j = 1; i < p.n; i++, j=(i+1)%p.n) {
4        if (p[i] == q) return 3;
5        if (pointOnSeg(q, Line(p[i], p[j]))) return 2;
6        int k = sgn((q-p[j])^(p[i]-p[j]));
7        int u = sgn(p[i].y-q.y), v = sgn(p[j].y-q.y);
8        if (k > 0 && u < 0 && v >= 0) cnt++;
```

```
        if (k < 0 && v < 0 && u >= 0) cnt--;
    }
    return cnt != 0 ? 1 : 0;
} // 点和多边形的关系：3顶点，2边上，1内部，0外部

void convexCut(const Polygon &p, Line u, Polygon &dest) {
    int &top = dest.n; top = 0;
    for (int i = 0; i < n; i++) {
        int d1 = sgn((u.e-u.s)^(p[i]-u.s));
        int d2 = sgn((u.e-u.s)^(p[(i+1)%p.n]-u.s));
        if (d1 >= 0) dest[top++] = p[i];
        if (d1 * d2 < 0) dest[top++] = crossPoint(u, Line(p[i], p[(i+1)%p.n]));
    }
} // 获得有向直线u和它左侧的凸包部分形成的凸包

double areaCirclePolygon(const Polygon &p, Circle c) {
    double ans = 0;
    for (int i = 0, j = 1%p.n; i < p.n; i++, j=(i+1)%p.n)
        if (sgn((p[j] - c.p) ^ (p[i] - c.p)) >= 0)
            ans += areaCircleTriangle(c, p[i], p[j]);
        else
            ans -= areaCircleTriangle(c, p[i], p[j]);
    return fabs(ans);
} // 圆与多边形的交面积

Point baryCenter(Polygon &p) {
    Point ret(0, 0); double area = 0;
    for (int i = 1; i < p.n-1; i++) {
        double tmp = (p[i]-p[0])^(p[i+1]-p[0]);
        if (sgn(tmp) == 0) continue;
        area += tmp, ret = ret + ((p[0]+p[i]+p[i+1])/3*tmp);
    }
    if (sgn(area)) ret = ret / area;
    return ret;
} // 获得多边形的重心
```

8.2.3 最小矩形面积覆盖

```
double minRectangleCover(Polygon &A) {
    #define dt(x) ((A[i+1]-A[i])*(A[x]-A[i]))
    #define cx(x) ((A[i+1]-A[i])^(A[x]-A[i]))
    int n = A.n; if (n < 3) return 0.0; A[n] = A[0];
    double ans = -1; int r = 1, p = 1, q;
    for (int i = 0; i < n; i++) {
        while (sgn(cs(r+1) - cs(r)) >= 0) r = (r+1)%n;
        while (sgn(dt(p+1) - dt(p)) >= 0) p = (p+1)%n;
        if (i == 0) q = p;
```

```
10      while (sgn(dt(q+1) - dt(q)) <= 0) q = (q+1)%n;
11      double tmp = cs(r) * (dt(p) - dt(q)) / (A[i]-A[i+1]).len2();
12      if (ans < 0 || ans > tmp) ans = tmp;
13   }
14   return ans;
15 }
```

8.2.4 旋转卡壳求直径

```
1  double rotateCalipers(Polygon &p) {
2     if (p.n == 2) return (p[0]-p[1]).length();
3     double ans = -inf; p[p.n] = p[0];
4     for (int i = 0, q = 1; i < n; i++) {
5        while (sgn((p[q] - p[i+1]) ^ (p[i] - p[i+1]) - (p[q+1] - p[i+1]) ^ (p[i]
             - p[i+1])) < 0) q = (q + 1) % p.n;
6        ans = max(ans, max((p[q]-p[i]).length(), (p[q+1]-p[i+1]).length()));
7     }
8     return ans;
9  } // 注意凸包上不能有三点共线
```

8.2.5 半平面交

```
1  struct HalfPlane : Line {
2     double angle;
3     HalfPlane() : Line(Point(0,0), Point(0,0)) {}
4     HalfPlane(Point _s, Point _e) : Line(_s, _e) {}
5     void init() { angle = atan2(e.y - s.y, e.x - s.x); }
6     bool operator<(const HalfPlane &b) const { return angle < b.angle; }
7  }; // 直线s->e左侧
8
9  struct HalfPlanes {
10    int n, Q[MAXP], F, R; HalfPlane hp[MAXP]; Point p[MAXP];
11    void push(HalfPlane tmp) { hp[n++] = tmp; }
12
13    void unique() {
14       int m = 1;
15       for (int i = 1; i < n; i++)
16          if (sgn(hp[i].angle - hp[i-1].angle) != 0)
17             hp[m++] = hp[i];
18          else if (sgn((hp[m-1].e-hp[m-1].s)^(hp[i].s-hp[m-1].s)) > 0)
19             hp[m-1] = hp[i];
20       n = m;
21    }
22
23    bool solve() {
24       for (int i = 0; i < n; i++) hp[i].init();
25       sort(hp, hp+n); unique(); Q[F=0] = 0, Q[R=1] = 1;
```

```
26      p[1] = crossPoint(hp[0], hp[1]);
27      for (int i = 2; i < n; i++) {
28         while (F<R && sgn((hp[i].e-hp[i].s)^(p[R]-hp[i].s))<0) R--;
29         while (F<R && sgn((hp[i].e-hp[i].s)^(p[F+1]-hp[i].s))<0) F++;
30         Q[++R] = i; if (parallel(hp[i], hp[Q[R-1]])) return false;
31         p[R] = crossPoint(hp[i], hp[Q[R-1]]);
32      }
33      while (F<R && sgn((hp[Q[F]].e-hp[Q[F]].s)^(p[R]-hp[Q[F]].s))<0) R--;
34      while (F<R && sgn((hp[Q[R]].e-hp[Q[R]].s)^(p[F+1]-hp[Q[R]].s))<0) F++;
35      return F+1 < R;
36   }
37
38   void getConvex(Polygon &con) {
39      p[F] = crossPoint(hp[Q[F]], hp[Q[R]]); con.n = R - F + 1;
40      for (int j = F, i = 0; j <= R; i++, j++) con[i] = p[j];
41   } // 先调用solve()并返回true
42 };
```

8.3 经典问题

8.3.1 圆面积并与 K 次圆交

```
1  const int MAXN = 1010;
2  bool operator==(Circle u, Circle v) { return u.p == v.p && !sgn(u.r-v.r); }
3  double areaarc(double th, double r) { return 0.5*r*r*(th-sin(th)); }
4  Point direct(double theta) { return Point(cos(theta), sin(theta)); }
5
6  bool inner(Circle x, Circle y) {
7     return circleRelateCircle(x, y) == 1 && sgn(x.r - y.r) <= 0;
8  } // 是否x在y内部
9
10 struct Circles {
11    Circle c[MAXN]; int n;
12    double ans[MAXN], pre[MAXN]; // ans[i]: i次覆盖面积
13    inline void add(Circle cc) { c[n++] = cc; }
14
15    void init_or() {
16       static bool mark[MAXN];
17       int i, j, k = 0;
18       for (i = 0; i < n; i++) {
19          for (j = 0; j < n; j++) if (i != j && !mark[j])
20             if ((c[i] == c[j]) || inner(c[i], c[j])) break;
21          if (j < n) mark[i] = true;
22       }
23       for (i = 0; i < n; i++) if (!mark[i]) c[k++] = c[i]; n = k;
24    } // k次覆盖不要调用，面积并调用
25
```

```
26    void getArea() {
27       memset(ans, 0, sizeof(ans));
28       vector<pair<double,int>> v;
29       for (int i = 0; i < n; i++) {
30          v.clear(), v.emplace_back(-pi, 1), v.emplace_back(pi, -1);
31          for (int j = 0; j < n; j++) if (i != j) {
32             Point q = c[j].p - c[i].p;
33             double ab = q.length(), ac = c[i].r, bc = c[j].r;
34             if (sgn(ab + ac - bc) <= 0) {
35                v.emplace_back(-pi, 1), v.emplace_back(pi, -1);
36                continue;
37             }
38             if (sgn(ab + bc - ac) <= 0) continue;
39             if (sgn(ab - ac - bc) > 0) continue;
40             double th = atan2(q.y, q.x);
41             double phi = acos((ac*ac+ab*ab-bc*bc)/(2*ac*ab));
42             double a0 = th - phi; if (sgn(a0 + pi) < 0) a0 += 2 * pi;
43             double a1 = th + phi; if (sgn(a1 - pi) > 0) a1 -= 2 * pi;
44             if (sgn(a0 - a1) > 0) {
45                v.emplace_back(a0, 1), v.emplace_back(pi, -1);
46                v.emplace_back(-pi, 1), v.emplace_back(a1, -1);
47             } else {
48                v.emplace_back(a0, 1), v.emplace_back(a1, -1);
49             }
50          }
51          sort(v.begin(), v.end());
52          int cur = 0;
53          for (auto vj : v) {
54             if (cur && sgn(vj.first - pre[cur])) {
55                ans[cur] += areaarc(vj.first - pre[cur], c[i].r);
56                ans[cur] += 0.5 * ((c[i].p + (direct(pre[cur]) * c[i].r)) ^
57                   (c[i].p + (direct(vj.first) * c[i].r)));
58             }
59             cur += vj.second, pre[cur] = vj.first;
60          }
61       }
62       for (int i = 1; i < n; i++) ans[i] -= ans[i+1];
63    }
64 };
```

8.3.2 平面最近点对

```
1  Point p[MAXN], tmpt[MAXN];
2
3  double closestPair(int L, int R) {
4     if (L == R) return INF;
5     if (L+1 == R) return (p[L]-p[R]).length();
```

```
   int mid = (L+R)/2, cnt = 0;
   double d1 = closestPair(L,mid), d2 = closestPair(mid+1,R);
   double d = min(d1,d2);
   for (int i = left; i <= right; i++)
      if (fabs(p[mid].x - p[i].x) <= d)
         tmpt[cnt++] = p[i];
   sort(tmpt, tmpt+cnt, [] (Point a, Point b) -> bool {
      return a.y < b.y || (a.y == b.y && a.x < b.x); });
   for (int i = 0; i < cnt; i++)
      for (int j = i+1; j < cnt && tmpt[j].y - tmpt[i].y < d; j++)
         d = min(d, (tmpt[i] - tmpt[j]).length());
   return d;
} // O(nlogn)
```

8.3.3 最小圆覆盖（随机增量）

```
Circle a[MAXN], ans;
inline bool inCircle(Point p) { return sgn((p-ans.p).length()-ans.r) < 0; }

void minCoverCircle(int n) {
   random_shuffle(a, a+n); ans.p.x = ans.p.y = ans.r = 0;
   for (int i = 0; i < n; i++) if (!inCircle(a[i])) {
      ans.p = a[i], ans.r = 0;
      for (int j = 0; j < i; j++) if (!inCircle(a[j])) {
         ans.p = (a[i]+a[j])/2, ans.r = (ans.p-ans[i]).length();
         for (int k = 0; k < j; k++) if (!inCircle(a[k])) {
            double a = a[i].x - a[j].x, b = a[i].y - a[j].y;
            double c = a[j].len2() - a[i].len2();
            double d = a[i].x - a[k].x, e = a[i].y - a[k].y;
            double f = a[k].len2() - a[i].len2();
            ans.p = Point((f*b-c*e)/(a*e-b*d), (f*a-c*d)/(b*d-e*a));
            ans.r = (b[i] - ans.p).length();
         }
      }
   }
} // random, O(n)
```

8.3.4 整数格点

皮克定理 给定顶点坐标均是整点的简单多边形，则其面积 A 和内部格点数 i、边上格点数 b 的关系为

$$A = i + \frac{b}{2} - 1$$

```
// 注意这部分中的结构体的数字必须为整数类型

int onSegment(Line l) {
   return abs(gcd(l.s.x - l.e.x, l.s.y - l.e.y)) + 1;
} // 线段上整点个数
```

```

int onEdge(const Polygon &p) {
   int ret = 0;
   for (int i = 0; i < p.n; i++)
      ret += abs(gcd(p[i].x-p[(i+1)%p.n].x, p[i].y-p[(i+1)%p.n].y));
   return ret;
} // 边上整点个数

int insidePolygon(const Polygon &p) {
   int area = 0, edge = 0;
   for (int i = 0; i < p.n; i++)
      area += p[i] ^ p[(i+1)%p.n],
      ret += abs(gcd(p[i].x-p[(i+1)%p.n].x, p[i].y-p[(i+1)%p.n].y));
   if (area < 0) area = -area;
   return (area - onEdge) / 2 + 1;
} // 多边形内部整点个数
```

8.4 立体几何

8.4.1 基础运算函数

```
const double eps = 1e-8;
int sgn(double x) { return x < -eps ? -1 : x > eps ? 1 : 0; }

struct Point3 {
   double x, y, z;
   Point3(double _x = 0, double _y = 0, double _z = 0) { x = _x, y = _y, z = _z; }
   bool operator==(Point3 b) const { return sgn(x-b.x)==0 && sgn(y-b.y)==0 && sgn(z-b.z)==0; }
   bool operator<(Point3 b) const { return sgn(x-b.x)==0 ? (sgn(y-b.y)==0 ? sgn(z-b.z) < 0 : y<b.y) : x<b.x; }
   double len() const { return sqrt(x*x+y*y+z*z); }
   double len2() const { return x*x+y*y+z*z; }
   Point3 operator-(Point3 b) const { return Point3(x-b.x, y-b.y, z-b.z); }
   Point3 operator+(Point3 b) const { return Point3(x+b.x, y+b.y, z+b.z); }
   Point3 operator*(double k) const { return Point3(k*x, k*y, k*z); }
   Point3 operator/(double k) const { return Point3(x/k, y/k, z/k); }
   double operator*(Point3 b) const { return x*b.x+y*b.y+z*b.z; }
   Point3 operator^(Point3 b) const { return Point3(y*b.z-z*b.y, z*b.x-x*b.z, x*b.y-y*b.x); }
   double rad(Point3 a, Point3 b) const { Point3 aa(a-*this), bb(b-*this); return acos(aa * bb / (aa.len() * bb.len())); }
   Point3 trunc(double r) const { double l = len(); if (!sgn(l)) return *this; r /= l; return Point3(x*r, y*r, z*r); }
};

```

```
struct Line3 {
   Point3 s, e;
   Line3(Point3 _s, Point3 _e) { s = _s, e = _e; }
   bool operator==(Line3 v) const { return s==v.s && e==v.e; };
   double length() const { return (e-s).len(); }
};

struct Plane3 {
   Point3 a, o;
   Plane3(Point3 _a, Point3 b, Point3 c) { a = _a, o = (b-_a)^(c-_a); }

   Plane3(double A, double B, double C, double D) {
      o = Point3(A, B, C);
      if (sgn(A)) a = Point3((-D-C-B)/A, 1, 1);
      else if (sgn(B)) a = Point3(1, (-D-C-A)/B, 1);
      else if (sgn(C)) a = Point3(1, 1, (-D-A-B)/C);
   }
};
```

8.4.2 点线面关系

```
double disPointToLine(Point3 p, Line3 u) {
   return ((u.e-u.s)^(p-u.s)).len() / (u.s-u.e).len();
} // 点到直线的距离

double disPointToSeg(Point3 p, Line3 u) {
   if (sgn((p-u.s)*(u.e-u.s)) < 0 || sgn((p-u.e)*(u.s-u.e)) < 0)
      return min((p-u.s).len(), (p-u.e).len());
   return disPointToLine(p, u);
} // 点到线段的距离

bool pointOnSeg(Point3 p, Line3 u) {
   return sgn(((u.s-p)^(u.e-p)).len()) == 0 && sgn((u.s-p)*(u.e-p)) == 0;
} // 点是否在线段上

Point3 projectToLine(Point3 p, Line3 v) {
   return v.s + (((v.e-v.s)*((v.e-v.s)*(p-v.s)))/((v.e-v.s).len2()));
} // 将点投影至直线上

bool pointOnPlane(Point3 p, Plane3 u) {
   return sgn((p-u.a)*u.o) == 0;
} // 点是否在平面上

Point3 projectToPlane(Point3 p, Plane3 v) {
   Line3 u(p, p+v.o); planeCrossLine(v, u, p); return p;
} // 将点投影至平面上

```

```
int planeCrossPlane(Plane3 u, Plane3 f, Line3 &qwq) {
   Point3 oo = u.o ^ f.o, v = u.o ^ oo;
   double d = fabs(f.o * v); if (sgn(d) == 0) return 0;
   Point3 q = u.a + (v*(f.o*(f.a-u.a))/d);
   return qwq = Line3(q, q+oo), 1;
} // 获取两个平面点交线

double anglePlane(Plane3 u, Plane3 f) {
   return acos((u.o*f.o)/(u.o.len()*f.o.len()));
} // 两个平面间的二面角

int planeCrossLine(Plane3 v, Line3 u, Point3 &p) {
   double x = v.o*(u.e-v.a), y = v.o*(u.s-v.a), d = x-y;
   if (sgn(d) == 0) return 0;
   return p = ((u.s*x)-(u.e*y))/d, 1;
} // 直线和平面的交点，返回交点个数
```

8.4.3 最小球覆盖（三分）

```
namespace BallCover {
   const double EPS = 1e-7;
   const int MAXP = 1e2+5;
   inline double sqr(double x) { return x * x; }
   struct Point { double P[3]; } a[MAXP]; int n;

   double calc(Point now) {
      double ans = 0.0;
      for (int i = 0; i < n; i++)
         ans = max(ans, sqrt(sqr(now.P[0] - a[i].P[0])
            + sqr(now.P[1] - a[i].P[1]) + sqr(now.P[2] - a[i].P[2])));
      return ans;
   }

   pair<Point, double> detect(Point now, int cnt) {
      if (cnt >= 3) return make_pair(now, calc(now));
      double r = 100000, l = -100000, dr, dl;
      pair<Point, double> ans, ans1, ans2;
      Point tp1, tp2; tp1 = tp2 = now;
      while (r - l > EPS) {
         tp1.P[cnt] = dl = (2*l+r)/3, tp2.P[cnt] = dr = (2*r+l)/3;
         ans1 = detect(tp1, cnt+1), ans2 = detect(tp2, cnt+1);
         if (ans1.second > ans2.second) l = dl, ans = ans1;
         else r = dr, ans = ans2;
      }
      return ans;
   }
}
```

8.4.4 凸包

```
#define cross(a,b,c) ((b-a)^(c-a))
#define area(a,b,c) (cross(a,b,c).len())
#define volume(a,b,c,d) (cross(a,b,c)*(d-a))

struct ConvexHull3D {
    struct face { int a, b, c; bool ok; };

    int n, num; // n: 初始点个数, num: 面三角形个数
    Point3 P[MAXN]; face F[8*MAXN]; int g[MAXN][MAXN];

    double dblcmp(Point3 p, face f) const {
        return ((P[f.b]-P[f.a])^(P[f.c]-P[f.a]))*(p-P[f.a]);
    }

    void deal(int p, int a, int b) {
        int f = g[a][b]; face add;
        if (!F[f].ok) return;
        if (dblcmp(P[p], F[f]) > eps) {
            dfs(p, f);
        } else {
            add.a = b, add.b = a, add.c = p, add.ok = true;
            g[p][b] = g[a][p] = g[b][a] = num;
            F[num++] = add;
        }
    }

    void dfs(int p, int now) {
        F[now].ok = false;
        deal(p, F[now].b, F[now].a);
        deal(p, F[now].c, F[now].b);
        deal(p, F[now].a, F[now].c);
    }

    bool same(int s, int t) const {
        const Point3 &a = P[F[s].a], &b = P[F[s].b], &c = P[F[s].c];
        return fabs(volume(a, b, c, P[F[t].a])) < eps &&
            fabs(volume(a, b, c, P[F[t].b])) < eps &&
            fabs(volume(a, b, c, P[F[t].c])) < eps;
    }

    void create() {
        num = 0; face add;

        // 保证前四点不共面
        bool flag = true;
        for (int i = 1; i < n && flag; i++)
```

```
47          if (!(P[0] == P[i]))
48              swap(P[1],P[i]), flag = false;
49      if (flag) return; flag = true;
50      for (int i = 2; i < n && flag; i++)
51          if (sgn(area(P[0], P[1], P[i])))
52              swap(P[2], P[i]), flag = false;
53      if (flag) return; flag = true;
54      for (int i = 3; i < n && flag; i++)
55          if (sgn(volume(P[0], P[1], P[2], P[i])))
56              swap(P[3], P[i]), flag = false;
57      if (flag) return;
58      // **********
59
60      for (int i = 0; i < 4; i++) {
61          add.a = (i+1)%4, add.b = (i+2)%4, add.c = (i+3)%4, add.ok = true;
62          if (dblcmp(P[i], add) > 0) swap(add.b, add.c);
63          g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] = num;
64          F[num++] = add;
65      }
66
67      for (int i = 4; i < n; i++) for (int j = 0; j < num; j++)
68          if (F[j].ok && dblcmp(P[i], F[j]) > eps) { dfs(i, j); break; }
69      int tmp = num; num = 0;
70      for (int i = 0; i < tmp; i++) if (F[i].ok) F[num++] = F[i];
71  } // 获得凸包
72
73  double surfaceArea() const {
74      double res = 0;
75      if (n == 3) return area(P[0], P[1], P[2]) / 2;
76      for (int i = 0; i < num; i++)
77          res += area(P[F[i].a], P[F[i].b], P[F[i].c]);
78      return res / 2.0;
79  } // 凸包表面积
80
81  double chVolume() const {
82      double res = 0; Point3 tmp(0, 0, 0);
83      for (int i = 0; i < num; i++)
84          res += volume(tmp, P[F[i].a], P[F[i].b], P[F[i].c]);
85      return fabs(res / 6);
86  } // 凸包体积
87
88  int surfaceTriangleCount() const {
89      return num;
90  } // 表面三角形个数
91
92  int surfacePolygonCount() const {
93      int res = 0;
94      for (int i = 0; i < num; i++) {
```

```
95          bool flag = true;
96          for (int j = 0; j < i && flag; j++)
97             if (same(i, j)) flag = false;
98          res += flag;
99       }
100      return res;
101   } // 表面多边形个数
102
103   Point3 baryCenter() const {
104      Point3 ans(0,0,0), o(0,0,0); double all = 0;
105      for (int i = 0; i < num; i++) {
106         double vol = volume(o, P[F[i].a], P[F[i].b], P[F[i].c]);
107         ans = ans + (((o+P[F[i].a]+P[F[i].b]+P[F[i].c])/4.0)*vol);
108         all += vol;
109      }
110      return ans/all;
111   } // 凸球重心
112
113   double disPointToFace(Point3 p, int i) const {
114      return fabs(volume(P[F[i].a], P[F[i].b], P[F[i].c], p))
115         / area(P[F[i].a], P[F[i].b], P[F[i].c]);
116   } // 点到表面的距离
117};
```

第九章　字符串

9.1　匹配

9.1.1　KMP

- 给定模板串 p 和目标串 t，检查 p 是否是 t 的子串。
- $fail[i] = k$ 表示 $p[0..k-1] = p[i-k..i-1]$

```
void get_fail(char p[], int lenP, int fail[]) {
   fail[0] = fail[1] = 0;
   for (int i = 1; i < lenP; i++) {
      int j = fail[i];
      while (j && p[i] != p[j]) j = fail[j];
      fail[i+1] = p[i]==p[j] ? j+1 : 0;
   }
} // p从0开始，ababcab -> 0 0 0 1 2 0 1 2

void find(char t[], char p[], int f[]) {
   int n = strlen(t), m = strlen(p); get_fail(p, m, f);
   for (int i = 0, j = 0; i < n; i++) {
      while (j && p[j] != t[i]) j = f[j];
      if (p[j] == t[i]) j++;
      if (j == m) printf("%d\n", i-m+1);
   }
} // target串，parent串，fail数组
```

9.1.2　AC 自动机

- 给定若干模板串 p_i，统计目标串 t 中模板串的出现情况
- $insert(p_i) \to build() \to find(t)$
- 模板串与其子串被编码为 $1..size$，其 $fail$ 指向 Trie 中最大的可用后缀

```
int son[MAXN][26], val[MAXN], last[MAXN], fail[MAXN], size;

void insert(const char s[]) {
   int len = strlen(s), u = 0;
   for (int i = 0, v; i < len; i++, u = son[u][v])
      if (!son[u][v = s[i]-'a']) son[u][v] = ++size;
   val[u] = 1;
```

```
}

void build() {
   static int Q[MAXN]; int head = 0, tail = 0;
   for (int c = fail[0] = 0; c < 26; c++)
      if (son[0][c]) Q[tail++] = son[0][c];
   while (head < tail) {
      int u, v = Q[head++];
      for (int c = 0; c < 26; c++) {
         if (!(u = son[v][c])) {
            son[v][c] = son[fail[v]][c];
         } else {
            Q[tail++] = u,
            fail[u] = son[fail[v]][c],
            last[u] = val[fail[u]] ? fail[u] : last[fail[u]];
         }
      }
   }
}

void print(int j) {
   for (int p = j; p; p = last[p]) ans++;
} // 统计模板串出现次数

void find(const char s[]) {
   int v = 0, len = strlen(s);
   for (int i = 0; i < len; i++) {
      v = son[v][s[i]-'a'];
      if (val[v]) print(v);
      else if (last[v]) print(last[v]);
   }
}
```

9.1.3 exKMP

- 给定文本串 S 和模板串 T，求出 S 每个后缀与 T 的 lcp
- 前半段得到 $nex[i]=z$ 意义 $t[0..z-1]=t[i..i+z-1]$
- 后半段得到 $extend[i]=z$ 意义 $s[i..i+z-1]=t[0..z-1]$

```
int nex[MAXN], extend[MAXN], lenS, lenT;

void exkmp(const char s[], const char t[]) {
   nex[0] = lenT = strlen(t); lenS = strlen(s);

   for (int i = 1, j = -1, a, p; i < lenT; i++, j--) {
      if (j < 0 || i + nex[i-a] >= p) {
         if (j < 0) j = 0, p = i;
```

```
9           while (p < lenT && t[p] == t[j]) p++, j++;
10          nex[i] = j;
11          a = i;
12       } else {
13          nex[i] = nex[i-a];
14       }
15    }
16
17    for (int i = 0, j = -1, a, p; i < lenS; i++, j--) {
18       if (j < 0 || i + nex[i-a] >= p) {
19          if (j < 0) j = 0, p = i;
20          while (p < lenS && j <= lenT && s[p] == t[j]) p++, j++;
21          extend[i] = j;
22          a = i;
23       } else {
24          extend[i] = nex[i-a];
25       }
26    }
27 }
```

9.2 后缀

9.2.1 后缀数组

- n：串长，m：字符集大小，$s[1..n]$：字符串
- $sa[1..n]$：字典序第 i 小的是哪个后缀
- $rank[1..n]$：后缀 i 的小到大排名
- $height[i]$：$sa[i]$ 与 $sa[i-1]$ 的最长公共前缀

```
1  #define rep(i,a,b) for (int i = a; i <= b; i++)
2  #define pre(i,a,b) for (int i = a; i >= b; i--)
3  int Rank[MAXN], sa[MAXN], sum[MAXN], tmp[MAXN], height[MAXN];
4
5  void DA(char s[], int n, int m) {
6     int *x=Rank, *y=tmp;
7     rep(i,0,m) sum[i]=0;
8     rep(i,1,n) sum[Rank[i]=(s[i]-'a'+1)]++;
9     rep(i,1,m) sum[i]+=sum[i-1];
10    pre(i,n,1) sa[sum[Rank[i]]--]=i;
11    for(int j=1,p=1; p<n; j<<=1,m=p) {
12       p=1; pre(i,n,n-j+1) y[p++]=i;
13       rep(i,1,n) if(sa[i]>j) y[p++]=sa[i]-j;
14       rep(i,1,m) sum[i]=0;
15       rep(i,1,n) sum[x[y[i]]]++;
16       rep(i,1,m) sum[i]+=sum[i-1];
17       pre(i,n,1) sa[sum[x[y[i]]]--]=y[i];
```

```
18      p=1; swap(x,y); x[sa[1]]=1; int a,b=sa[1];
19      rep(i,2,n) a=b,b=sa[i],x[b]=(y[a]==y[b]&&y[a+j]==y[b+j])?p:++p;
20   }
21   rep(i,1,n) Rank[sa[i]]=i;
22   for(int i=1,j,k=0;i<=n;height[Rank[i++]]=k)
23      for(k?k--:0,j=sa[Rank[i]-1]; s[i+k]==s[j+k]; k++);
24 }
```

9.2.2 后缀自动机

```
1  int fa[MAXN*2], son[MAXN*2][26], deep[MAXN*2], cnt, root, last;
2  inline int New(int _deep) { deep[++cnt] = _deep; return cnt; }
3  inline void pre() { root = last = New(0); }
4
5  inline void SAM(int alp) {
6     int np = New(deep[last]+1), u = last;
7     while (u && !son[u][alp]) son[u][alp] = np, u = fa[u];
8     if (!u) fa[np]=root;
9     else {
10       int v = son[u][alp];
11       if (deep[v] == deep[u]+1) fa[np] = v;
12       else {
13          int nv = New(deep[u]+1);
14          memcpy(son[nv], son[v], sizeof(son[v]));
15          fa[nv] = fa[v], fa[v] = fa[np] = nv;
16          while (u && son[u][alp] == v) son[u][alp] = nv, u = fa[u];
17       }
18    }
19    last = np;
20 }
```

9.2.3 最小表示法

- 获得使 $S[i..n]+S[1..i-1]$ 字典序最小的 i

```
1  int MinimumRepresentation(const int s[], int l) {
2     int i = 0, j = 1, k;
3     while (i < l && j < l) {
4        k = 0;
5        while (s[i+k] == s[j+k] && k < l) k++;
6        if (k == l) return i;
7        if (s[i+k] > s[j+k]) i = max(i+k+1, j+1);
8        else if (j+k+1 > i) j = j+k+1;
9        else j = i+1;
10    }
11    return i < l ? i : j;
12 }
```

9.2.4 离线后缀树

```
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
const int maxn=100010;
int cnt,last,fa[maxn][20],f[maxn],dep[maxn],suffix[maxn],Len[maxn],to[maxn
   ][30];
int son[maxn],nex[maxn],lnk[maxn],e;
char str[maxn];
void add(int u,int v){
   son[++e]=v; nex[e]=lnk[u]; lnk[u]=e;
}
void ins(int x,int z){
   int p=last,np=++cnt;
   Len[np]=Len[p]+1; suffix[z]=np;
   while (p && !to[p][x]) to[p][x]=np,p=f[p];
   if (!p) f[np]=1;
   else{
      int q=to[p][x];
      if (Len[q]==Len[p]+1) f[np]=q;
      else{
         int nq=++cnt;
         memcpy(to[nq],to[q],sizeof(to[nq]));
         Len[nq]=Len[p]+1;
         f[nq]=f[q];
         f[q]=f[np]=nq;
         while (p && to[p][x]==q) to[p][x]=nq,p=f[p];
      }
   }
   last=np;
}
void dfs(int u){
   fa[u][0]=f[u];
   dep[u]=dep[f[u]]+1;
   for (int i=1; i<20; i++) fa[u][i]=fa[fa[u][i-1]][i-1];
   for (int i=lnk[u]; i; i=nex[i]) dfs(son[i]);
}
int lca(int u,int v){
   if (dep[u]<dep[v]) swap(u,v);
   int delta=dep[u]-dep[v];
   for (int i=0; i<20; i++) if ((delta>>i)&1) u=fa[u][i];
   if (u==v) return u;
   for (int i=19; i>=0; i--)
      if (fa[u][i]!=fa[v][i]) u=fa[u][i],v=fa[v][i];
   return f[u];
}
```

```
int main(){
   scanf("%s",str);
   int len=strlen(str);
   cnt=last=1;
   for (int i=len-1; i>=0; i--) ins(str[i]-'a',i);
   for (int i=2; i<=cnt; i++) add(f[i],i);
   dfs(1);
   while (1){
      int u,v;
      scanf("%d%d",&u,&v);
      printf("%d\n",Len[lca(suffix[u],suffix[v])]);
   }
   return 0;
}
```

9.3 回文串

9.3.1 Manacher

- $s[i]$：加入分割符号以后的字符串
- $f[i]-1$：以此为中心在原串中的长度

```
char s[MAXN*2]; int f[MAXN*2];

void Manacher(const char Str[], int len) {
   int l = 0, r = 0, m = 0; s[l++] = '#';
   for (int i = 0; i < len; i++)
      s[l++] = Str[i], s[l++] = '#';
   s[l] = '\0'; f[0] = 1;
   for (int i = 1; i < l; i++) {
      f[i] = i <= r ? min(r-i+1, f[m-(i-m)]) : 1;
      while (i-f[i]>=0 && s[i+f[i]]==s[i-f[i]]) f[i]++;
      if (i+f[i]-1 > r) r = i+f[i]-1, m = i;
   }
}
```

9.3.2 回文自动机

- N：串长，tot：本质不同回文串数量
- $text[1..N]$：原字符串
- $son[x][y]$：第 x 个点所代表的回文串两边加上字符 y 后的回文串
- $fail[x]$：第 x 个点所代表的回文串的最长回文后缀
- $len[x]$：第 x 个点所代表的回文串的长度
- $cnt[x]$：第 x 个点所代表的回文串的出现次数（需建完树后 $count()$ 一遍）

```cpp
int fail[MAXN], text[MAXN], len[MAXN];
int cnt[MAXN][2], son[MAXN][26], tot, n, last;

inline int find(int x) {
   while (text[n-len[x]-1] != text[n]) x = fail[x];
   return x;
}

inline void init() {
   for (int i = 0; i <= tot; i++)
      cnt[i][0] = cnt[i][1] = 0, memset(son[i], 0, sizeof(son[i]));
   n = last = 0, text[0] = len[1] = -1, fail[0] = tot = 1;
}

void add(int x, int tp) {
   text[++n] = x; int cur = find(last);
   if (!son[cur][x]) {
      tot++; len[tot] = len[cur] + 2;
      fail[tot] = son[find(fail[cur])][x];
      son[cur][x] = tot;
   }
   cnt[last = son[cur][x]][tp]++;
}

void count(int tp) {
   for (int i = tot; i > 2; i--)
      cnt[fail[i]][tp] += cnt[i][tp];
}

// 公共回文子串对数
lld commonPalindromeCount(const char a[], const char b[]) {
   init(); int L = strlen(a);
   for (int i = 0; i < L; i++) add(a[i]-'a', 0);
   n = last = 0; L = strlen(b);
   for (int i = 0; i < L; i++) add(b[i]-'a', 1);
   count(0), count(1);
   lld ans = 0;
   for (int i = 2; i <= tot; i++) ans += 1ll * cnt[i][0] * cnt[i][1];
   return ans;
}
```

第十章　数据结构

10.1　ST 表

```cpp
const int MAXN = 1048576, LOG = 20;
int ST[LOG][MAXN+5];

inline void initST(const int num[], int n) {
   for (int i = 1; i <= n; i++) ST[0][i] = num[i];
   for (int j = 1; (1<<j) <= n; j++)
      for (int i = 1; i + (1<<j) - 1 <= n; i++)
         ST[j][i] = max(ST[j-1][i], ST[j-1][i+(1<<(j-1))]);
} // 预处理，下标从1开始，O(nlogn)

inline int queryST(int l, int r) {
   int k = int(log2(r-l+1.0));
   return max(st[k][l], st[k][r-(1<<k)+1]);
} // 查询最大值/最小值/坐标，O(1)
```

```cpp
const int MAXN = 512, LOG = 10;
int ST[LOG][LOG][MAXN][MAXN], lg2[MAXN];
#define rep(i, a, b) for (int i = a; i <= b; i++)

inline void initST(const int num[MAXN][MAXN], int n, int m) {
   rep (i, 2, MAXN-1) lg2[i] = lg2[i>>1]+1;
   rep (i, 1, n) rep (j, 1, m) ST[0][0][i][j] = num[i][j];
   rep (ii, 0, lg2[n]) rep (jj, 0, lg2[m]) if (ii+jj)
      rep (i, 1, n-(1<<ii)+1) rep (j, 1, m-(1<<jj)+1)
         ST[ii][jj][i][j] = ii
            ? max(ST[ii-1][jj][i][j], ST[ii-1][jj][i+(1<<(ii-1))][j])
            : max(ST[ii][jj-1][i][j], ST[ii][jj-1][i][j+(1<<(jj-1))]);
} // 预处理，下标从1开始，O(nm logn logm)

inline int queryST(int x1, int y1, int x2, int y2) {
   int k1 = lg2[x2-x1+1], k2 = lg2[y2-y1+1];
   x2 -= (1<<k1)-1, y2 -= (1<<k2)-1;
   return max( max(ST[k1][k2][x1][y1], ST[k1][k2][x1][y2]),
      max(ST[k1][k2][x2][y1], ST[k1][k2][x2][y2]));
} // x1 <= i <= x2, y1 <= j <= y2
```

10.2 树状数组

10.2.1 一维树状数组

```
int BIT[1048576], n;
inline int lowbit(int x) { return x & (-x); }

inline void update(int x, int k) {
   while (x <= n) BIT[x] += k, x += lowbit(x);
} // 给 A[x] 处加上 k

inline int query(int x) {
   int res = 0;
   while (x > 0) res += BIT[x], x -= lowbit(x);
   return res;
} // 查询 A[1..x] 的和
```

区间修改 单点查询 维护差分数组，区间修改 L 处加，R+1 处减，单点直接查询前缀和。

区间修改 区间查询 维护 $D[i]$ 和 $iD[i]$，修改 L 处加，R+1 处减，查询 $(n+1)S_1(n) - S_2(n)$。

10.2.2 二维树状数组

```
int c[MAXN][MAXM], n, m;
inline int lowbit(int x) { return x & (-x); }

void update(int x, int y, int z) {
   for (int i = x; i <= n; i += lowbit(i))
      for (int j = y; j <= m; j += lowbit(j))
         c[i][j] += z;
} // 二维前缀和，给 (x,y) 处加 z

int query(int x, int y) {
   int ret = 0;
   for (int i = x; i >= 1; i -= lowbit(i))
      for (int j = y; j >= 1; j -= lowbit(j))
         ret += c[i][j];
   return ret;
} // 二维前缀和，求 1<=i<=x, 1<=j<=y 的值

// 区间修改单点查询，修改a[x1..x2][y1..y2]+=w，查询单点值直接 query
void add1(int x1, int y1, int x2, int y2, int w) {
   update(x1, y1, w), update(x2+1, y2+1, w);
   update(x2+1, y1, -w), update(x1, y2+1, -w);
}

// 区间修改区间查询，维护 D[i][j], i*D[i][j], j*D[i][j], i*j*D[i][j]
void add2(int x1, int y1, int x2, int y2, int w) {
```

```
26     update(c1,x1,y1,w), update(c1,x2+1,y1,-w);
27     update(c1,x1,y2+1,-w), update(c1,x2+1,y2+1,w);
28     update(c2,x1,y1,w*x1), update(c2,x2+1,y1,-w*(x2+1));
29     update(c2,x1,y2+1,-w*x1), update(c2,x2+1,y2+1,w*(x2+1));
30     update(c3,x1,y1,w*y1), update(c3,x2+1,y1,-w*y1);
31     update(c3,x1,y2+1,-w*(y2+1)), update(c3,x2+1,y2+1,w*(y2+1));
32     update(c4,x1,y1,w*x1*y1), update(c4,x2+1,y1,-w*(x2+1)*y1);
33     update(c4,x1,y2+1,-w*x1*(y2+1)), update(c4,x2+1,y2+1,w*(x2+1)*(y2+1));
34 }
35
36 int get2(int x, int y) {
37     return query(c1,x,y)*(x+1)*(y+1) - query(c2,x,y)*(y+1)
38         - query(c3,x,y)*(x+1) + query(c4,x,y);
39 }
40
41 int sum2(int x1, int y1, int x2, int y2) {
42     return get2(x2,y2) - get2(x2,y1-1) - get2(x1-1,y2) + get2(x1-1,y1-1);
43 }
```

10.3 并查集

10.3.1 普通并查集

```
1  int fa[MAXN];
2
3  void initDSU(int n) { for (int i = 0; i <= n; i++) fa[i] = i; }
4  int find(int x) { return fa[x] = (fa[x] == x ? x : find(fa[x])); }
5
6  bool merge(int x, int y) {
7      int fx = find(x), fy = find(y);
8      if (fx == fy) return false;
9      if (fx < fy) fa[fx] = fy; else fa[fy] = fx;
10     return true;
11 }
```

10.3.2 种类并查集

10.3.3 带权并查集

10.3.4 可持久化并查集

```
1  #include<cstdio>
2  #include<algorithm>
3  using namespace std;
4  struct node{
5      int ch[2],f,deep;
6  }lib[10000000];
```

```
7  int n,m,size=-1,l,r,k,p,last,root[200010];
8  void build(int &rot,int l,int r){
9     rot=++size;
10    if (l==r){lib[rot].f=l;return;}
11    int m=(l+r)>>1;
12    build(lib[rot].ch[0],l,m);
13    build(lib[rot].ch[1],m+1,r);
14 }
15 void add(int rot,int l,int r,int x){
16    if (l==r){lib[rot].deep++;return;}
17    int m=(l+r)>>1;
18    if (x<=m) add(lib[rot].ch[0],l,m,x);
19    else add(lib[rot].ch[1],m+1,r,x);
20 }
21 void change(int &rot,int l,int r,int pos,int val){
22    lib[++size]=lib[rot]; rot=size;
23    if (l==r){lib[rot].f=val; return;}
24    int m=(l+r)>>1;
25    if (pos<=m) change(lib[rot].ch[0],l,m,pos,val);
26    else change(lib[rot].ch[1],m+1,r,pos,val);
27 }
28 int get(int rot,int l,int r,int x){
29    if (l==r) return rot;
30    int m=(l+r)>>1;
31    if (x<=m) return get(lib[rot].ch[0],l,m,x);
32    else return get(lib[rot].ch[1],m+1,r,x);
33 }
34 int find(int rot,int x){
35    int p=get(rot,1,n,x);
36    if (lib[p].f==x) return p;
37    return find(rot,lib[p].f);
38 }
39
40 int main(){
41    scanf("%d%d",&n,&m);
42    build(root[0],1,n);
43    for (int i=1; i<=m; i++){
44       scanf("%d",&p);
45       if (p==1){
46          scanf("%d%d",&l,&r); l^=last; r^=last;
47          root[i]=root[i-1];
48          int fl=find(root[i],l),fr=find(root[i],r);
49          if (lib[fl].f!=lib[fr].f){
50             if (lib[fl].deep>lib[fr].deep) swap(fl,fr);
51             change(root[i],1,n,lib[fl].f,lib[fr].f);
52             if (lib[fl].deep==lib[fr].deep) add(root[i],1,n,lib[fr].f);
53          }
54       }else if (p==2){
```

```
55          scanf("%d",&k); k^=last;
56          root[i]=root[k];
57       }else{
58          scanf("%d%d",&l,&r); l^=last; r^=last;
59          root[i]=root[i-1];
60          int fl=find(root[i],l),fr=find(root[i],r);
61          if (lib[fl].f==lib[fr].f) printf("1\n"),last=1;
62          else printf("0\n"),last=0;
63       }
64    }
65    return 0;
66 }
```

10.4 平衡树

10.4.1 Treap

```
1  class Treap {
2  private:
3     struct node { int v, key, lc, rc, cnt, sz; } p[MAXN];
4     int seed, root, pn;
5     int rnd() { return seed = seed * 23333 % 998244353; }
6
7     void update(int fxy) {
8        p[fxy].sz = p[p[fxy].lc].sz + p[p[fxy].rc].sz + p[fxy].cnt;
9     }
10
11    void rotate(int &fxy, int dir) {
12       if (dir) {
13          int v = p[fxy].lc; p[fxy].lc = p[v].rc; p[v].rc = fxy;
14          update(fxy); update(fxy = v);
15       } else {
16          int v = p[fxy].rc; p[fxy].rc = p[v].lc; p[v].lc = fxy;
17          update(fxy); update(fxy = v);
18       }
19    }
20
21    void insert(int v, int &fxy) {
22       if (!fxy) {
23          p[fxy = ++pn] = (node){ v, rnd(), 0, 0, 1, 1 };
24       } else if (v == p[fxy].v) {
25          p[fxy].cnt++; p[fxy].sz++;
26       } else if (v < p[fxy].v) {
27          insert(v, p[fxy].lc); p[fxy].sz++;
28          if (p[fxy].key < p[p[fxy].lc].key) rotate(fxy, 1);
29       } else if (v > p[fxy].v) {
30          insert(v, p[fxy].rc); p[fxy].sz++;
```

```
31          if (p[fxy].key < p[p[fxy].rc].key) rotate(fxy, 0);
32       }
33    }
34
35    bool remove(int v, int &fxy) {
36       if (!v) return 0;
37       if (v == p[fxy].v) {
38          if (p[fxy].cnt > 1) {
39             p[fxy].cnt--; p[fxy].sz--; return 1;
40          } else if (!p[fxy].lc || !p[fxy].rc) {
41             fxy = p[fxy].lc ? p[fxy].lc : p[fxy].rc; return 1;
42          } else {
43             rotate(fxy, p[p[fxy].lc].key > p[p[fxy].rc].key ? 1 : 0);
44             return remove(v, fxy);
45          }
46       }
47       if (!remove(v, p[fxy].v > v ? p[fxy].lc : p[fxy].rc)) return 0;
48       p[fxy].sz--; return 1;
49    }
50
51 public:
52    Treap() { seed = 1; pn = 0; }
53    void work_ins(int v) { insert(v, root); }
54    void work_del(int v) { remove(v, root); }
55
56    int ask_rk(int v) {
57       int fxy = root, tans = 1;
58       while(fxy) {
59          if (p[fxy].v == v) return tans + p[p[fxy].lc].sz;
60          else if (p[fxy].v > v) fxy = p[fxy].lc;
61          else tans += p[p[fxy].lc].sz + p[fxy].cnt, fxy = p[fxy].rc;
62       }
63       return 0;
64    }
65
66    int ask_v(int v) {
67       int fxy = root;
68       while (fxy) {
69          if (p[p[fxy].lc].sz + p[fxy].cnt < v)
70             v -= p[p[fxy].lc].sz + p[fxy].cnt, fxy = p[fxy].rc;
71          else if (p[p[fxy].lc].sz < v) return p[fxy].v;
72          else fxy = p[fxy].lc;
73       }
74       return 0;
75    }
76
77    int ask_pre(int v) {
78       int fxy = root, tans = 0;
```

```
79        while (fxy) {
80           if (p[fxy].v < v) tans = p[fxy].v, fxy = p[fxy].rc;
81           else fxy = p[fxy].lc;
82        }
83        return tans;
84     }
85
86     int ask_suf(int v) {
87        int fxy = root, tans = 0;
88        while (fxy) {
89           if (p[fxy].v > v) tans = p[fxy].v, fxy = p[fxy].lc;
90           else fxy = p[fxy].rc;
91        }
92        return tans;
93     }
94  }bltree;
95  int main()
96  {
97     scanf("%d",&n);
98     int opt,x;
99     while(n--)
100    {
101      scanf("%d%d",&opt,&x);
102      if(opt==1)bltree.work_ins(x);
103      if(opt==2)bltree.work_del(x);
104      if(opt==3)printf("%d\n",bltree.ask_rk(x));
105      if(opt==4)printf("%d\n",bltree.ask_v(x));
106      if(opt==5)printf("%d\n",bltree.ask_pre(x));
107      if(opt==6)printf("%d\n",bltree.ask_suf(x));
108    }
109    return 0;
110 }
```

10.4.2 左偏树

```
1  #include<cstdio>
2  #include<cstring>
3  #include<algorithm>
4  #define MAXN 100005
5  using namespace std;
6  int n,m,p,q,f[MAXN];
7  struct node{
8     int l,r,v,d;
9  }t[MAXN];
10 int find(int x){
11    return f[x]==x?x:f[x]=find(f[x]);
12 }
```

```
int merge(int x,int y){
   if (x==0) return y; if (y==0) return x;
   if (t[x].v<t[y].v) swap(x,y);
   t[x].r=t[x].r?merge(t[x].r,y):y;
   if (t[t[x].l].d<t[t[x].r].d) swap(t[x].l,t[x].r);
   t[x].d=t[x].r?t[t[x].r].d+1:0;
   return x;
}
int fight(int x,int y){
   int fx=find(x),fy=find(y);
   if (fx==fy) return -1;
   t[fx].v/=2; t[fy].v/=2;
   int roota=merge(t[fx].l,t[fx].r); t[fx].l=t[fx].r=0;
   int rootb=merge(t[fy].l,t[fy].r); t[fy].l=t[fy].r=0;
   roota=merge(roota,fx); rootb=merge(rootb,fy);
   int tmp=merge(roota,rootb);
   f[tmp]=tmp; f[fx]=tmp; f[fy]=tmp;
   return t[tmp].v;
}
int main(){
   while (scanf("%d",&n)==1){
      memset(t,0,sizeof(t));
      for (int i=1; i<=n; i++) scanf("%d",&t[i].v),f[i]=i;
      scanf("%d",&m);
      for (int i=1; i<=m; i++){
         scanf("%d%d",&p,&q);
         printf("%d\n",fight(p,q));
      }
   }
   return 0;
}
```

10.4.3 笛卡尔树

```
#include<cstdio>
#include<algorithm>
using namespace std;
struct point{
   int key,val,s;
   bool operator <(const point b)const{
      return key<b.key;
   }
}p[50010];
int n,stack[50010],l[50010],r[50010],f[50010];
int main(){
   scanf("%d",&n);
   for (int i=1; i<=n; i++) scanf("%d%d",&p[i].key,&p[i].val),p[i].s=i;
```

```
    sort(p+1,p+1+n);
    stack[0]=1; stack[1]=1;
    for (int i=2; i<=n; i++){
        if (p[stack[stack[0]]].val<p[i].val){
            r[p[stack[stack[0]]].s]=p[i].s;
            f[p[i].s]=p[stack[stack[0]]].s;
            stack[++stack[0]]=i;
            continue;
        }
        while (stack[0]>0 && p[stack[stack[0]]].val>p[i].val) stack[0]--;
        l[p[i].s]=p[stack[stack[0]+1]].s;
        f[p[stack[stack[0]+1]].s]=p[i].s;
        if (stack[0]!=0) r[p[stack[stack[0]]].s]=p[i].s,f[p[i].s]=p[stack[stack
            [0]]].s;
        stack[++stack[0]]=i;
    }
    printf("YES\n");
    for (int i=1; i<=n; i++) printf("%d %d %d\n",f[i],l[i],r[i]);
    return 0;
}
```

10.5 KD 树

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 1000010;
const int Dimension = 3;
const int inf = 1e9;
const long long infl = 1e18;
struct point
{
    int x[3];
    point operator - (const point &b) const{
        point c;
        for (int i = 0; i < 3; i++) c.x[i] = x[i] - b.x[i];
        return c;
    }
    long long operator * (const point & b)const{
        long long d = 0;
        for (int i = 0; i < 3; i++) d += 1LL * x[i] * b.x[i];
        return d;
    }
};
int chl[maxn], chr[maxn];
point rg[maxn][2], p[maxn];
int build(int l, int r, int _d)
{
```

```
25     int mid = l + r >> 1;
26     nth_element(p + l, p + mid, p + r + 1, [_d](const point &a, const point &b)
           {return a.x[_d] < b.x[_d];});
27     for (int i = 0; i < 3; i++)
28     {
29        rg[mid][0].x[i] = inf;
30        rg[mid][1].x[i] = -inf;
31     }
32     for (int i = l; i <= r; i++)
33     {
34        for (int j = 0; j < 3; j++)
35        {
36           rg[mid][0].x[j] = min(rg[mid][0].x[j], p[i].x[j]);
37           rg[mid][1].x[j] = max(rg[mid][1].x[j], p[i].x[j]);
38        }
39     }
40     if (l < mid)
41        chl[mid] = build(l, mid - 1, (_d + 1) % 3);
42     if (mid < r)
43        chr[mid] = build(mid + 1, r, (_d + 1) % 3);
44     return mid;
45  }
46  long long sqr(int x)
47  {
48     return 1ll * x * x;
49  }
50  long long ans = infl;
51  int ansl, ansr;
52  void contri(int pl, int pr)
53  {
54     long long t = (p[pl] - p[pr]) * (p[pl] - p[pr]);
55     if (t < ans)
56        ans = t, ansl = pl, ansr = pr;
57  }
58  void query(int l, int r, const int &tar, int _d)
59  {
60     int mid = l + r >> 1; int _x = p[mid].x[_d], _X = p[tar].x[_d];
61     if (mid != tar) contri(mid, tar);
62     if (_X < _x)
63     {
64        if (chl[mid])
65           query(l, mid - 1, tar, (_d + 1) % 3);
66        if (chr[mid])
67        {
68           long long sdis = sqr(_X - rg[chr[mid]][0].x[_d]);
69           if (sdis < ans)
70              query(mid + 1, r, tar, (_d + 1) % 3);
71        }
```

```
72      }else{
73          if (chr[mid])
74              query(mid + 1, r, tar, (_d + 1) % 3);
75          if (chl[mid])
76          {
77              long long sdis = sqr(_X - rg[chl[mid]][1].x[_d]);
78              if (sdis < ans)
79                  query(l, mid - 1, tar, (_d + 1) % 3);
80          }
81      }
82  }
83  int main()
84  {
85      // read n and p[1]...p[n]
86      int rt = build(1, n, 0);
87      for (int i = 1; i <= n; i++)
88          query(1, n, i, 0);
89      // minimum distance of pairs
90      printf("Dist = %.10lf\n", sqrt(ans));
91      return 0;
92  }
```

10.6 线段树

10.6.1 主席树

```
1   #include<cstdio>
2   #include<algorithm>
3   #include<map>
4   #define MAXN 1000000
5   using namespace std;
6   struct node{
7       int ch[2],size;
8   }lib[MAXN*20];
9   int top,cnt,n,m,a[MAXN],b[MAXN],root[MAXN];
10  map<int,int>w,iw;
11  int NEWNODE(int p){
12      lib[++top]=lib[p];return top;
13  }
14  void insert(int &p,int l,int r,int v){
15      p=NEWNODE(p);
16      if (l==r){lib[p].size++; return;}
17      int mid=(l+r)>>1;
18      if (v<=mid) insert(lib[p].ch[0],l,mid,v);
19      else insert(lib[p].ch[1],mid+1,r,v);
20      lib[p].size=lib[lib[p].ch[0]].size+lib[lib[p].ch[1]].size;
21  }
```

```
22 int query(int pl,int pr,int l,int r,int k){
23    if (l==r) return l;
24    int cnt=lib[lib[pr].ch[0]].size-lib[lib[pl].ch[0]].size,mid=(l+r)>>1;
25    if (k<=cnt) return query(lib[pl].ch[0],lib[pr].ch[0],l,mid,k);
26    else return query(lib[pl].ch[1],lib[pr].ch[1],mid+1,r,k-cnt);
27 }
28 int main(){
29    scanf("%d%d",&n,&m);
30    for (int i=1; i<=n; i++) scanf("%d",&a[i]),b[i]=a[i];
31    sort(b+1,b+1+n); cnt=0;
32    for (int i=1; i<=n; i++) if (b[i]!=b[i-1]) cnt++,w[b[i]]=cnt,iw[cnt]=b[i];
33    root[0]=0;
34    for (int i=1; i<=n; i++){
35       root[i]=root[i-1];
36       insert(root[i],1,n,w[a[i]]);
37    }
38    for (int i=1; i<=m; i++){
39       int l,r,k;
40       scanf("%d%d%d",&l,&r,&k);
41       printf("%d\n",iw[query(root[l-1],root[r],1,n,k)]);
42    }
43    return 0;
44 }
```

第十一章　图论

11.1　最短路

11.1.1　最短路树

Maintain the shortest path if an adjoining edge is prohibited.

11.1.2　单源 Dijkstra

```cpp
const int INF = 0x3f3f3f3f;
int W[1010][1010], dis[1010], vis[1010];

void dijkstra(int n, int t) {
   fill(vis, vis+n+1, 0), fill(dis, vis+n+1, INF), dis[t] = 0;
   for (int j = 0; j <= n; j++) {
      int k = -1, Min = INF;
      for (int i = 0; i <= n; i++)
         if (!vis[i] && dis[i] < Min) Min = dis[i], k = i;
      if (k == -1) break; vis[k] = true;
      for (int i = 0; i <= n; i++) if (!vis[i])
         dis[i] = min(dis[i], dis[k] + W[k][i]);
   }
} // O(V^2) 适用于稠密非负权图
```

```cpp
const int MAXN = 1000010, INF = 0x3f3f3f3f;
typedef pair<int,int> Edge; vector<Edge> G[MAXN];
bool vis[MAXN]; int dis[MAXN];
#define V second

void addedge(int u, int v, int cost) {
   G[u].emplace_back(cost, v);
}

void dijkstra(int n, int start) {
   fill(vis, vis+n+1, false), fill(dis, dis+n+1, INF);
   priority_queue<Edge,vector<Edge>,greater<Edge>> que;
   dis[start] = 0, que.emplace(0, start);
   while (!que.empty()) {
      Edge tmp = que.top(); que.pop(); int u = tmp.V;
      if (vis[u]) continue; vis[u] = true;
```

```
17        for (auto e : G[u]) if (!vis[e.V] && dis[e.V] > dis[u] + e.first)
18           dis[e.V] = dis[u] + e.first, que.emplace(dis[e.V], e.V);
19     }
20  } // 堆优化 O(ElogE) 适用于稀疏非负权图
```

11.1.3 单源 SPFA

```
1   const int INF = 0x3f3f3f3f, MAXV = 505, MAXE = 5210;
2   int cnt, dist[MAXV], head[MAXV], num[MAXV], vis[MAXV];
3   struct edge { int v, w, next; } E[MAXE];
4
5   void addedge(int u, int v, int w) {
6      E[cnt] = (edge){ v, w, head[u] }; head[u] = cnt++;
7   }
8
9   bool spfa(int n, int st) {
10     memset(vis, 0, sizeof(vis));
11     memset(num, 0, sizeof(num));
12     static int Q[MAXE]; int front = 0, rear = 0;
13     vis[st] = 1, dist[st] = 0, num[st]++, Q[rear++] = st;
14     while (front < rear) {
15        int u = Q[front++]; vis[u] = 0;
16        for (int i = head[u], v; ~i; i = E[i].next) {
17           if (dist[v = E[i].v] <= dist[u] + E[i].w) continue;
18           dist[v] = dist[u] + E[i].w;
19           if (!vis[v]) {
20              vis[v] = 1, Q[rear++] = v;
21              if (++num[v] > n) return false;
22           }
23        }
24     }
25     return true;
26  } // O(kE) 玄学复杂度，判负环再用吧，有负环返回false
```

11.1.4 全源 Floyd

```
1   void floyd(int n) {
2      for (int k = 1; k <= n; k++)
3         for (int i = 1; i <= n; i++)
4            for (int j = 1; j <= n; j++)
5               dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
6   } // O(V^3) 适用于稠密图
```

11.1.5 全源 Johnson

```
1   using Graph::spfa; // bool spfa(int n, int start, int dis[]);
```

```
using Graph::dijkstra; // dijkstra(int n, int start, int dis[]);

bool johnson(int n) {
   static int h[MAXV]; fill(h, h+n+1, 0);
   for (int i = 1; i <= n; i++) addedge(0, i, 0);
   if (!Graph::spfa(n+1, 0, h)) return false;
   for (int u = 1; u <= n; u++)
      for (int i = head[u]; ~i; i = E[i].next)
         E[i].w += h[u] - h[E[i].v];
   for (int u = 1; u <= n; u++) {
      Graph::dijkstra(n, u, dist[u]);
      for (int v = 1; v <= n; v++)
         dist[u][v] -= h[u] - h[v];
   }
} // O(V^2logV + VE) 适用于稀疏图，建图注意留出点0
```

11.1.6 Dijkstra 求 K 短路

当 K 不大时使用 Dijkstra 求 K 短路。

```
typedef long long ll;
const int INF = 0x3f3f3f3f;
const int MAXN = 5e5 + 100;
const int MAXM = 3e3 + 10;
int n, m, k, v;
ll ans, dist[110];
int lin[MAXN], tot = 0;
struct edge {
   int y, v, next;
}e[MAXN];

inline void add(int xx, int yy, int vv) {
   e[++tot].y = yy;
   e[tot].v = vv;
   e[tot].next = lin[xx];
   lin[xx] = tot;
}

void Dijkstra() {
   priority_queue < pair < int , int > > q;
   q.push(make_pair(0, 1));
   while(!q.empty()) {
      int x = q.top().second;
      int d = -q.top().first;
      q.pop();
      if(x == n) {
         dist[++v] = d;
         if(v == k + 1) return ;
```

```
        }
        for(int i = lin[x], y; i; i = e[i].next) {
            y = e[i].y;
            ans = d + e[i].v;
            q.push(make_pair(-ans, y));
        }
    }
}

int main() {
    memset(dist, -1, sizeof(dist));
    n = read(); m = read(); k = read();
    for(int i = 1; i <= m; ++i) {
        int x,y,v;
        x = read(); y = read(); v = read();
        add(x, y, v);
    }
    Dijkstra();
    for(int i = 1; i <= k; ++i) {
        write(dist[i]);
        putchar('\n');
    }
    return 0;
}
```

11.1.7 A* 求 K 短路

```
typedef pair<int,int> P;
const int N = 1010, M = 10010, INF = 1000000010;
int g[N], h[N], v[M<<1], w[M<<1], nxt[M<<1], ed;
int d[N], vis[N], ans[N];
priority_queue<P, vector<P>, greater<P>> Q;

void add(int x, int y, int z) {
    v[++ed] = x; w[ed] = z; nxt[ed] = g[y]; g[y] = ed;
    v[++ed] = y; w[ed] = z; nxt[ed] = h[x]; h[x] = ed;
} // fill g[], h[] with default 0.

void astar(int n, int k) {
    for (int i = 1; i <= k; i++) ans[i] = -1;
    for (int i = 1; i <= n; i++) d[i] = INF;
    int S = n, T = 1, x; Q.push(P(d[T] = 0, T));
    while (!Q.empty()) {
        P t = Q.top(); Q.pop();
        if (d[t.second] < t.first) continue;
        for (int i = g[x=t.second]; i; i = nxt[i])
            if (d[x]+w[i]<d[v[i]]) Q.push(P(d[v[i]]=d[x]+w[i],v[i]));
```

```
    }
    if (d[S] < INF) Q.push(P(d[S], S));
    while (!Q.empty()) {
        P t = Q.top(); Q.pop(); vis[x=t.second]++;
        if (x == T && vis[T] <= k) ans[vis[T]] = t.first;
        if (vis[x] <= k) for (int i = h[x]; i; i = nxt[i])
            Q.push(P(t.first-d[x]+d[v[i]]+w[i], v[i]));
    }
    for (int i = 1; i <= k; i++) printf("%d\n", ans[i]);
}
```

11.1.8 差分约束系统

a 向 b 连一条权值为 c 的有向边表示 $b-a \leq c$，用 SPFA 判断是否存在负环，存在即无解。

11.1.9 无向图最小环

```
const int MAXV = 110, INF = 0x2a2a2a2a;
int C[MAXV][MAXV], S[MAXV][MAXV]; // C为真实图邻接矩阵

int minimal_cylce(int N) {
    memcpy(S, C, sizeof(S)); int ans = INF;
    for (int k = 1; k <= N; k++) {
        for (int i = 1; i <= N; i++)
            for (int j = 1; j <= N; j++)
                if (j != i && j != k && i != k)
                    ans = min(ans, S[i][j] + C[j][k] + C[k][i]);
        for (int i = 1; i <= N; i++)
            for (int j = 1; j <= N; j++)
                S[i][j] = min(S[i][j], S[i][k] + S[k][j]);
    }
    return ans;
}
```

11.1.10 最短哈密顿回路

```
const int INF = 0x3f3f3f3f;
int G[17][17], dp[131072][17];

int hamiton() {
    memset(G, 0x3f, sizeof(G)); memset(dp, 0x3f, sizeof(dp));
    read_graph(); if (n == 1) return 0; Graph::floyd(G, n);
    dp[1][1] = 0; for (int i = 1; i <= n; i++) G[i][i] = 0;
    for (int s = 1; s < (1<<n); s++) for (int i = 1; i <= n; i++) {
        if (s & (1<<(i-1))) for (int j = 1; j <= n; j++) {
            if (s & (1<<(j-1))) continue; int tt = s|(1<<(j-1));
            dp[tt][j] = min(dp[tt][j], dp[s][i] + G[i][j]);
```

```
12      }
13   }
14   int ans = INF;
15   for (int i = 2; i <= n; i++)
16      ans = min(ans, dp[(1<<n)-1][i] + G[i][1]);
17   return ans;
18}
```

11.1.11 枚举三元环

给定一张 n 个点 m 条边的完全图，在 $O(m\sqrt{m})$ 的时间内枚举所有三元环。

```
1  const int N = 100010, M = 200010, Base = (1<<21)-1;
2  struct edge { int v, w; edge *nxt; } epool[M], *ecur = epool, *g[N];
3  struct Edge { int x, y, w; Edge *nxt; } Epool[M], *Ecur = Epool, *G[Base+1];
4  struct Elist { int x, y, w; } e[M]; int d[N], lim;
5
6  bool cmp(const Elist &a, const Elist &b) {
7     return a.x == b.x ? a.y < b.y : a.x < b.x;
8  }
9
10 int vis(int x, int y) {
11    for (Edge *l = G[(x<<8|y)&Base]; l; l = l->nxt)
12       if (l->x == x && l-> y == y) return l->w;
13    return 0;
14 }
15
16 void solve(int n, m) {
17    while (lim * lim < m) lim++;
18    for (int i = 1, x, y; i <= m; i++) {
19       scanf("%d %d", &x, &y); if (x < y) swap(x, y);
20       e[i].x = x, e[i].y = y;
21    }
22    sort(e+1, e+m+1, cmp);
23    for (int i = 1, x, y, Hash; i <= m; i++) {
24       d[x=e[i].x]++;
25       ecur->v = y = e[i].y; ecur->w = i; ecur->nxt = g[x]; g[x] = ecur++;
26       Ecur->x = x; Ecur->y = y; Ecur->w = i;
27       Ecur->nxt = G[Hash=(x<<8|y)&Base]; G[Hash] = Ecur++;
28    }
29
30    for (int i = 3, x, y; i <= n; i++) for (edge *j = g[i]; j; j = j->nxt) {
31       if (d[x=j->v] <= lim) {
32          for (edge *k = g[x]; k; k = k->nxt) if (y = vis(i, k->v)) {
33             // 三条边分别为 e[j->w] e[k->w] e[y]
34             // 与 x 点相连的边为 e[j->w] e[k->w]
35             // 与 i 点相连的边为 e[j->w] e[y]
36             // 与 k->v 点相连点边为 e[k->w] e[y]
```

```
37          }
38       } else {
39          for (edge *k = j->nxt; k; k = k->nxt) if (y = vis(x, k->v)) {
40             // 三条边分别为 e[j->w] e[k->w] e[y]
41             // 与 i 点相连的边为 e[j->w] e[k->w]
42             // 与 x 点相连的边为 e[j->w] e[y]
43             // 与 k->v 点相连点边为 e[k->w] e[y]
44          }
45       }
46    }
47
48    lim = 0, ecur = epool, Ecur = Epool;
49    for (int i = 1; i <= n; i++) d[i] = 0, g[i] = NULL;
50    for (int i = 1; i <= m; i++) G[(e[i].x<<8|e[i].y)&Base] = NULL;
51 }
```

11.1.12 欧拉路

欧拉回路 每条边只经过一次，而且回到起点。

欧拉路径 每条边只经过一次，不要求回到起点。

回路判断 无向图每个顶点度数为偶数；有向图的无向基图连通，且每个顶点出度等于入度。

BEST theorem 设 $ec(G)$ 为欧拉图 G 中以 w 为起点的欧拉回路的数量，$t_w(G)$ 为固定 w 为根有向图的生成树数量（用矩阵树定理计算），则有

$$ec(G) = t_w(G) \prod_{v \in V} (\deg(v) - 1)!$$

11.2 生成树

11.2.1 Kruskal

```
1  struct edge { int u, v, w; } G[100050];
2
3  int kruskal(int n, int m) {
4     for (int i = 1; i <= n; i++) fa[i] = i;
5     sort(G, G+m, [] (edge a, edge b) { return a.w < b.w; });
6     int ans = 0, left = n-1;
7     for (int i = 0; i < m; i++)
8        if (DSU::merge(G[i].u, G[i].v)) left--, ans += G[i].w;
9     return ans;
10 } // O(ElogE)，适用于稀疏图
```

11.2.2 Prim

```cpp
const int MAXV = 110, INF = 0x3f3f3f3f;
double c[MAXV][MAXV], dis[MAXV]; bool vis[MAXV];

int prim(int n) {
   memset(vis, 0, sizeof(vis)); memset(dis, 0x3f, sizeof(dis));
   vis[1] = true, dis[1] = 0; int ans = 0;
   for (int i = 1; i < n; i++) {
      int Min = INF, p = -1;
      for (int j = 1; j <= n; j++)
         if (!vis[j] && Min > dis[j]) Min = dis[j], p = j;
      if (p == -1) return -1; vis[p] = true; ans += Min;
      for (int j = 1; j <= n; j++) if (!vis[j]) dis[j] = min(dis[j], c[p][j]);
   }
   return ans;
} // O(V^2), 注意不要堆优化，别把斯坦纳树当作堆优化Prim
```

11.2.3 次小生成树

枚举每条不在最小生成树上的边，并把这条边放到最小生成树上面，一定会形成环，再从环中取出一条最长的边（不是新加入的边），即为次小生成树的权值。

11.2.4 矩阵树定理

G 的出度矩阵 $D(G)$, 邻接矩阵（存在 $i \to j$ 有向边则 $A_{i,j} = 1$, 否则为 0) $A(G)$, 则 $C(G) = D(G) - A(G)$ 中划去其中一行一列求行列式绝对值为原图的生成树个数。

实际上，划去 i^{th} 行和列后的行列式值是以 i（i 能达到所有点）为根的所有可能的生成树的边权乘积之和，其中 $D_{i,i}$ 为 i 出边边权之和，$A_{i,j}$ 为有向图边权，若不存在 $i \to j$ 则 $A_{i,j} = 0$。

11.2.5 曼哈顿距离最小生成树

考虑以每个点当作坐标原点，那么以 x 轴、y 轴、$x + y = 0$、$x - y = 0$ 四条直线可以将平面分割为 8 个部分，则将这八个部分里离原点曼哈顿距离最近的点添加边，考虑到对称性只枚举其中四个象限，则为 $O(n)$ 条边，进行最小生成树即可。寻找点的时候，建立抽象数据结构，每个元素对应一个点 (x, y)，第一关键字为 $y - x$，第二关键字为 $y + x$，便划归为：从 T 第一关键字大于某常数的所有元素中寻找第二关键字最小的元素。

```cpp
const int MAXN = 50010, INF = 0x3f3f3f3f;
struct Point { int x, y, id; } p[MAXN], pp[MAXN];
struct Edge { int u, v, d; } edge[MAXN<<2]; int fa[MAXN], tol;
#define rep(i, n) for (int i = 0; i < n; i++)
bool operator<(Point a, Point b) { return a.x!=b.x?a.x<b.x:a.y<b.y; }
bool operator<(Edge a, Edge b) { return a.d < b.d; }
struct BIT { int v, p; void init() { v = INF, p = -1; } } bit[MAXN];
int find(int x) { return fa[x] == -1 ? x : (fa[x] = find(fa[x])); }
void __addedge(int u, int v, int d) { edge[tol++] = Edge { u, v, d }; }
inline int dist(Point a, Point b) { return abs(a.x-b.x)+abs(a.y-b.y); }

inline bool merge(int a, int b) {
   int A = find(a), B = find(b);
```

```
14    if (A == B) return false; else return fa[A] = B, true;
15 }
16
17 void update(int i, int val, int pos) {
18    for (; i; i -= (i & (-i)))
19       if (val < bit[i].v) bit[i].v = val, bit[i].p = pos;
20 }
21
22 int ask(int i, int m) {
23    int min_val = INF, pos = -1;
24    for (; i <= m; i += (i & (-i)))
25       if (bit[i].v < min_val) min_val = bit[i].v, pos = bit[i].p;
26    return pos;
27 }
28
29 void solve(int n) {
30    int a[MAXN], b[MAXN]; tol = 0;
31    for (int dir = 0; dir < 4; dir++) {
32       if (dir & 1) rep(i, n) swap(p[i].x, p[i].y);
33       else if (dir == 2) rep(i, n) p[i].x = -p[i].x;
34       sort(p, p+n);
35       rep(i, n) a[i] = b[i] = p[i].y - p[i].x;
36       sort(b, b+n); int m = unique(b, b+n) - b;
37       for (int i = 1; i <= m; i++) bit[i].init();
38       for (int i = n-1; i >= 0; i--) {
39          int pos = lower_bound(b, b+m, a[i])-b+1, ans = ask(pos, m);
40          if (ans != -1) __addedge(p[i].id, p[ans].id, dist(p[i], p[ans]));
41          update(pos, p[i].x + p[i].y, i);
42       }
43    }
44    memset(fa, -1, sizeof(fa));
45    sort(edge, edge+tol);
46    for (int i = 0; i < tol; i++)
47       if (merge(edge[i].u, edge[i].v))
48          add_tree_edge(edge[i].u, edge[i].v);
49 }
```

11.2.6 欧拉距离最小生成树

求出三角剖分后，只有三角剖分上的边需要用到。

一个更简单的方式去理解：对于每个点把平面以每 60 度的形式分成 6 个部分，只需要考虑每个部分的最近点。注意需要用到维诺图（Voronoi diagram），这是三角剖分（Delaunay triangulation）的对偶图。

11.2.7 随机情况下欧拉距离最小生成树

- Method 1: Do it as if no random at all.
- Method 2: Choose some random direction and sort points by projection of point to it. For each point, enumerate and maintain p closest points until the current point’s projection distance is longer than

all chosen points’ s actual distance.

- Method 3: Divide the plane into several square regions, enumerate regions around it in a suitable distance for each point.(Usually in sqrt(MAXX))
- Method 4: Use KD-Tree to find the closest few points for each one. Then apply Kruskal.
- Method 5: Use KD-Tree to maintain the closest points for each one. Then apply Prim.

11.2.8 最小方差生成树

新权值为 $v_i = (w_i - avg)^2$。从最小值到最大值枚举 avg 计算 v_i。

11.2.9 最小乘积生成树

设每条边有 (x, y) 两个权值。求一棵生成树使得 $(\sum x)(\sum y)$ 最小。

设每棵生成树 $(\sum x, \sum y)$ 为坐标系上的一个点，则问题转化为求一个点使得 $xy = k$ 最小。则可以先分别求出距离 x 轴和 y 轴最近的点，即对单一权值求最小生成树。然后寻找一个在 AB 靠近原点侧寻找生成树 C。最小化叉积 $\vec{CA} \times \vec{CB}$，即最大化 $(B.x - A.x)C.y + (A.y - B.y)C.x - A.y\ B.x + A.x\ B.y$，修改 kruskal 所需要的边权为对应形式即可找到此 C。然后以 AC 和 BC 为边界分别递归寻找。可推广至高维。

11.2.10 最小直径生成树

Find the Absolute Center C of G: minimize $max(dist(C, i)), i \in G$. And then find the Shortest Path Tree from C.

寻找绝对重心：枚举每条边，考虑寻找这条边上的一个点作为重心。以重心在这条边上的位置作为参数，到其他点的距离的最大值作为函数值，求这个函数的最小值。则可以分别考虑到每个点的距离函数，寻找找它们的交点。

11.2.11 度数限制生成树

不同问题的时间复杂度

- Minimum Degree Spanning Tree: NP hard.
- Determine maximum degree <= k: NP-completeness.
- Only restrict the root node degree <= k: A classic problem.

对于第三个问题的算法

- Remove all edges of the root and find the spanning forest.
- If there are p components, p <= k must hold. Then connect each component to the root, which build a p-degree solution.
- Try to add an used edge (root,y) and delete the largest edge on the path (root,y) which can maintain by a DFS each time. Find the minimum way and build a (p+1)-degree solution.
- Repeat 3 to get a k-degree solution.

11.2.12 最小树形图

```
const int MAXN = 10050, MAXM = 50050;
const double INF = 1e9, EPS = 1e-8;
inline int sgn(double x) { return x < -EPS ? -1 : x > EPS; }
struct E { int u, v, nxt; double w; } e[MAXM];
int head[MAXN], tol;

void addedge(int u, int v, double w) {
   ++tol; e[tol].nxt = head[u]; head[u] = tol;
   e[tol].u = u, e[tol].v = v, e[tol].w = w;
}

int idx[MAXN], vis[MAXN], pre[MAXN];
double in[MAXN];

double dmst(int root, int n) {
   double ans = 0;
   while (true) {
      for (int i = 0; i <= n; i++) in[i] = INF;
      memset(vis, -1, sizeof(vis));
      memset(idx, -1, sizeof(idx));

      for (int i = 1, u, v; i <= tol; i++) {
         if ((u = e[i].u) == (v = e[i].v)) continue;
         if (e[i].w <= in[v]) in[v] = e[i].w, pre[v] = u;
      }

      for (int i = 1; i <= n; i++) {
         if (i == root) continue;
         if (sgn(in[i] - INF) == 0) return -1;
      }

      int cnt = 0; in[root] = 0;
      for (int i = 1; i <= n; i++) {
         ans += in[i]; int v = i;
         while (vis[v] != i && idx[v] == -1 && v != root)
            vis[v] = i, v = pre[v];
         if (v != root && idx[v] == -1) {
            ++cnt;
            for (int u = pre[v]; u != v; u = pre[u]) idx[u] = cnt;
            idx[v] = cnt;
         }
      }

      if (cnt == 0) break;
      for (int i = 1; i <= n; i++) if (idx[i] == -1) idx[i] = ++cnt;
      for (int i = 1, u, v; i <= tol; i++) {
```

```
47          u = e[i].u, v = e[i].v, e[i].u = idx[u], e[i].v = idx[v];
48          if (idx[u] != idx[v]) e[i].w -= in[v];
49       }
50
51       n = cnt, root = idx[root];
52    }
53
54    return ans;
55 }
```

11.2.13 斯坦纳树

在一张图中选择一些边，使得给定点集连通并且边权最小。设 $dp[u][S]$ 为以 u 为根，连通 S 集合点的最小权值，则有

- 枚举子树的形态 $dp[u][S]=\min_{s\subseteq S}\{dp[u][s]+dp[u][S-s]\}$
- 按照边进行松弛 $dp[u][S]=\min_{(u,v)\in G}\{w[u][v]+dp[v][S]\}$

```
1  const int MAXE = 20005, INF = 0x3f3f3f3f;
2  int f[MAXE][256], nxt[MAXE], head[MAXE], to[MAXE], w[MAXE], tol, ALL;
3  int Q[10000005], front, rear, wants[10], d; bool vis[MAXE];
4
5  void addedge(int u, int v, int z) {
6     nxt[++tol] = head[u]; head[u] = tol; to[tol] = v; w[tol] = z;
7     nxt[++tol] = head[v]; head[v] = tol; to[tol] = u; w[tol] = z;
8  } // 普通无向图。注意关键点编号储存在wants[0..d-1]中，d为关键点个数
9
10 void spfa(int opt) {
11    while (front < rear) {
12       int u = Q[front++], v;
13       for (int i = head[u]; i; i = nxt[i]) {
14          if (f[v=to[i]][opt] > f[u][opt] + w[i]) {
15             f[v][opt] = f[u][opt] + w[i];
16             if (!vis[v]) vis[v] = true, Q[rear++] = v;
17          }
18       }
19       vis[u] = false;
20    }
21    front = rear = 0;
22 } // 也可以使用dijkstra来松弛
23
24 void solve() {
25    memset(f, 0x3f, sizeof(f));
26    for (int i = 0; i < d; i++) f[wants[i]][1<<i] = 0;
27    ALL = (1<<d) - 1;
28    for (int opt = 0; opt <= ALL; opt++) {
29       for (int i = 1; i <= n; i++) {
30          for (int s = opt; s; s = (s-1) & opt)
```

```
            f[i][opt] = min(f[i][opt], f[i][s] + f[i][opt^s]);
         if (f[i][opt] != INF) Q[rear++] = i, vis[i] = true;
      }
      spfa(opt);
   }
}
```

如果是关键点全部都连通，则直接输出 $\min f[i][ALL]$；如果是要求某些点对互相连通，则把合法 opt 的最小值存进 $ans[opt]$，然后使用类似于枚举子集和补集的方式来合并最后答案。时间复杂度约为 $O(n3^d)$。

11.3 网络流

11.3.1 Dinic 最大流

```
const int INF = 0x3f3f3f3f, MAXN = 15000, MAXM = 100000;
struct Edge { int to, nex, cap; } e[MAXM];
int tol, head[MAXN], dep[MAXN]; // 初始化tol=2, head[0..N]=-1

void addedge(int u, int v, int cap, int rev = 0) {
   e[tol] = Edge { v, head[u], cap }; head[u] = tol++;
   e[tol] = Edge { u, head[v], rev }; head[v] = tol++;
}

bool dinic_bfs(int s, int t) {
   static int Q[MAXN]; int front = 0, rear = 0;
   memset(dep, 0, sizeof(dep)); Q[rear++] = s, dep[s] = 1;
   while (front != rear) {
      int u = Q[front++], v;
      for (int i = head[u]; ~i; i = e[i].nex) {
         if (e[i].cap > 0 && dep[v = e[i].to] == 0) {
            dep[v] = dep[u] + 1;
            if (v == t) return true;
            Q[rear++] = v;
         }
      }
   }
   return false;
}

int dinic_dfs(int u, int t, int f) {
   if (u == t) return f; int d, v, c = 0;
   for (int i = head[u]; ~i; i = e[i].nex) {
      if (e[i].cap > 0 && dep[u] + 1 == dep[v = e[i].to]) {
         d = dinic_dfs(v, t, min(f - c, e[i].cap));
         if (d > 0) {
            e[i].cap -= d, e[i^1].cap += d, c += d;
            if (f == c) break;
         } else dep[v] = -1;
```

```
35         }
36      }
37      return c;
38  }
39
40  int dinic(int s, int t) {
41      int maxflow = 0;
42      while (dinic_bfs(s, t))
43          maxflow += dinic_dfs(s, t, INF);
44      return maxflow;
45  }
```

11.3.2 ISAP 最大流

```
1   const int INF = 0x3f3f3f3f, MAXN = 100010, MAXE = 220000;
2   struct Edge { int to, nex, cap; } e[MAXE];
3   int tol, head[MAXN], gap[MAXN], dep[MAXN], cur[MAXN]; // tol=2, head[]=-1
4
5   void addedge(int u, int v, int cap, int rev = 0) {
6       e[tol] = Edge { v, head[u], cap }; head[u] = tol++;
7       e[tol] = Edge { u, head[v], rev }; head[v] = tol++;
8   }
9
10  void isap_bfs(int s, int t) {
11      memset(dep, -1, sizeof(dep)); memset(gap, 0, sizeof(gap));
12      static int Q[MAXN*10]; int front = 0, rear = 0;
13      gap[0] = 1; dep[t] = 0; Q[rear++] = t;
14      while (front != rear) {
15          int u = Q[front++], v;
16          for (int i = head[u]; ~i; i = e[i].nex) {
17              if (dep[v = e[i].to] != -1) continue;
18              Q[rear++] = v; dep[v] = dep[u] + 1; gap[dep[v]]++;
19          }
20      }
21  }
22
23  int isap(int s, int t, int N) {
24      isap_bfs(s, t); memcpy(cur, head, sizeof(head));
25      static int S[MAXN*10]; int top = 0, u = s, ans = 0;
26      while (dep[s] < N) {
27          if (u == t) {
28              int Min = INF, inser = 0;
29              for (int i = 0; i < top; i++)
30                  if (Min > e[S[i]].cap) Min = e[S[i]].cap, inser = i;
31              for (int i = 0; i < top; i++)
32                  e[S[i]].cap -= Min, e[S[i]^1].cap += Min;
33              ans += Min, top = inser, u = e[S[top]^1].to;
```

```
34        } else {
35           bool flag = false; int v, Min = N;
36           for (int i = cur[u]; ~i; i = e[i].nex)
37              if (e[i].cap > 0 && dep[v = e[i].to]+1 == dep[u]) {
38                 flag = true; cur[u] = i; break;
39           }
40
41           if (flag) { S[top++] = cur[u]; u = v; continue; }
42           for (int i = head[u]; ~i; i = e[i].nex)
43              if (e[i].cap > 0 && dep[e[i].to] < Min)
44                 Min = dep[e[i].to], cur[u] = i;
45           if (!(--gap[dep[u]])) return ans;
46           dep[u] = Min + 1; gap[dep[u]]++;
47           if (u != s) u = e[S[--top]^1].to;
48        }
49     }
50     return ans;
51  }
```

11.3.3 HLPP 最大流

```
1   const int MAXN = 2e5+5, MAXM = 4e5+5, INF = 0x3f3f3f3f;
2   struct Edge { int to, cap, nex; } G[MAXM<<1];
3   int head[MAXN], hi[MAXN], gap[MAXN<<1], tol, inq[MAXN], e[MAXN];
4   struct cmp { bool operator()(int a, int b) { return hi[a] < hi[b]; } };
5   priority_queue<int,vector<int>,cmp> heap;
6   // head[..] = -1, gap[..] = inq[..] = e[..] = 0
7
8   void addedge(int u, int v, int cap, int rev = 0) {
9      G[tol] = Edge { v, cap, head[u] }; head[u] = tol++;
10     G[tol] = Edge { u, rev, head[v] }; head[v] = tol++;
11  }
12
13  bool hlpp_bfs(int s, int t) {
14     static int Q[MAXN]; int front = 0, rear = 0;
15     memset(hi, 0x3f, sizeof(hi));
16     hi[t] = 0, Q[rear++] = t;
17     while (front < rear) {
18        int u = Q[front++];
19        for (int i = head[u]; ~i; i = G[i].nex)
20           if (G[i^1].cap && hi[G[i].to] > hi[u] + 1)
21              hi[G[i].to] = hi[u]+1, Q[rear++] = G[i].to;
22     }
23     return hi[s] != INF;
24  }
25
26  void push(int u, int s, int t) {
```

```
27     for (int i = head[u]; ~i; i = G[i].nex) {
28        if (!G[i].cap || hi[G[i].to]+1 != hi[u]) continue;
29        int d = min(e[u], G[i].cap);
30        G[i].cap -= d, G[i^1].cap += d, e[u] -= d, e[G[i].to] += d;
31        if (G[i].to != s && G[i].to != t && !inq[G[i].to])
32           heap.push(G[i].to), inq[G[i].to] = 1;
33        if (!e[u]) break;
34     }
35  }
36
37  void relabel(int u) {
38     hi[u] = INF;
39     for (int i = head[u]; ~i; i = G[i].nex)
40        if (G[i].cap && hi[G[i].to]+1 < hi[u]) hi[u] = hi[G[i].to]+1;
41  }
42
43  int hlpp(int s, int t, int n) {
44     if (!hlpp_bfs(s, t)) return 0;
45     hi[s] = n; memset(gap, 0, sizeof(gap));
46     for (int i = 1; i <= n; i++) if (hi[i] < INF) ++gap[hi[i]];
47     for (int i = head[s], d; ~i; i = G[i].nex) if ((d = G[i].cap)) {
48        G[i].cap -= d, G[i^1].cap += d, e[s] -= d, e[G[i].to] += d;
49        if (G[i].to != s && G[i].to != t && !inq[G[i].to])
50           heap.push(G[i].to), inq[G[i].to] = 1;
51     }
52     while (!heap.empty()) {
53        int u = heap.top(); heap.pop(); inq[u] = 0; push(u, s, t);
54        if (!e[u]) continue;
55        if (!--gap[hi[u]]) for (int i = 1; i <= n; i++)
56           if (i != s && i != t && hi[i] > hi[u] && hi[i] < n+1) hi[i] = n+1;
57        relabel(u); ++gap[hi[u]]; heap.push(u); inq[u] = 1;
58     }
59     return e[t];
60  }
```

11.3.4 SPFA 费用流

```
1  const int MAXN = 10010, MAXM = 100010, INF = 0x3f3f3f3f;
2  struct Edge { int to, next, cap, flow, cost; } edge[MAXM];
3  int head[MAXN], tol, pre[MAXN], dis[MAXN]; bool vis[MAXN];
4
5  void init() {
6     tol = 0; memset(head, -1, sizeof(head));
7  }
8
9  void addedge(int u, int v, int cap, int cost) {
10    edge[tol] = (Edge){ v, head[u], cap, 0, cost }; head[u] = tol++;
```

```
    edge[tol] = (Edge){ u, head[v], 0, 0, -cost }; head[v] = tol++;
}

bool spfa(int s, int t, int N) {
    for (int i = 0; i <= N; i++)
        dis[i] = INF, vis[i] = false, pre[i] = -1;
    static int Q[MAXM]; int front = 0, rear = 0;
    dis[s] = 0, vis[s] = true, Q[rear++] = s;
    while (front < rear) {
        int u = Q[front++]; vis[u] = false;
        for (int i = head[u]; ~i; i = edge[i].next) {
            int v = edge[i].to;
            if (edge[i].cap > edge[i].flow && dis[v] > dis[u] + edge[i].cost) {
                dis[v] = dis[u] + edge[i].cost, pre[v] = i;
                if (!vis[v]) vis[v] = true, Q[rear++] = v;
            }
        }
    }
    return pre[t] != -1;
}

pair<int,int> minCostMaxFlow(int s, int t, int N) {
    int flow = 0, cost = 0;
    while (spfa(s, t, N)) {
        int Min = INF;
        for (int i = pre[t]; ~i; i = pre[edge[i^1].to])
            Min = min(Min, edge[i].cap - edge[i].flow);
        for (int i = pre[t]; ~i; i = pre[edge[i^1].to])
            edge[i].flow += Min, edge[i^1].flow -= Min,
            cost += edge[i].cost * Min;
        flow += Min;
    }
    return make_pair(flow, cost);
}
```

11.3.5 Dijkstra 费用流

```
const int MAXN = 50010, MAXM = 1000010, INF = 0x3f3f3f3f;
struct Edge { int to, next, cap, flow, cost; } edge[MAXM];
int head[MAXN], tol, pre[MAXN], dis[MAXN], h[MAXN];
struct ND { int x, dis; bool operator<(ND b)const { return b.dis < dis; } };
priority_queue<ND> Q;

void init() {
    tol = 0; memset(head, -1, sizeof(head));
}

```

```
void addedge(int u, int v, int cap, int cost) {
   edge[tol] = (Edge){ v, head[u], cap, 0, cost }; head[u] = tol++;
   edge[tol] = (Edge){ u, head[v], 0, 0, -cost }; head[v] = tol++;
}

bool dijkstra(int s, int t, int N) {
   for (int i = 0; i <= N; i++)
      h[i] = min(h[i] + dis[i], INF), dis[i] = INF, pre[i] = -1;
   dis[s] = 0, Q.push(ND { s, 0 });
   while (!Q.empty()) {
      int u = Q.top().x, dist = Q.top().dis; Q.pop();
      if (dist > dis[u]) continue;
      for (int i = head[u]; ~i; i = edge[i].next) {
         Edge& e = edge[i]; int v = edge[i].to;
         if (e.cap > e.flow && dis[v] > dis[u] + e.cost + h[u] - h[v]) {
            dis[v] = dis[u] + e.cost + h[u] - h[v], pre[v] = i;
            Q.push(ND { v, dis[v] });
         }
      }
   }
   return dis[t] < INF;
}

pair<int,int> minCostMaxFlow(int s, int t, int N) {
   int flow = 0, cost = 0;
   while (dijkstra(s, t, N)) {
      int Min = INF, Fee = dis[t] + h[t] - h[s];
      for (int i = pre[t]; ~i; i = pre[edge[i^1].to])
         Min = min(Min, edge[i].cap - edge[i].flow);
      for (int i = pre[t]; ~i; i = pre[edge[i^1].to])
         edge[i].flow += Min, edge[i^1].flow -= Min;
      flow += Min, cost += Fee * Min;
   }
   return make_pair(flow, cost);
}
```

11.3.6 zkw 费用流

适用范围 二分图类型、多路增广。但在流量不大、费用不小、增广路较长的网络优势不大。

```
const int MAXV = 2200, MAXE = 200020, INF = 0x3f3f3f3f;
struct EDGE { int to, next, cap, flow, cost; } edge[MAXE];
int head[MAXV], tot, cur[MAXV], dis[MAXV], ss, tt, N; bool vis[MAXV];
void init() { tot = 0; memset(head, -1, sizeof(head)); }

void addedge(int u, int v, int cap, int cost) {
   edge[tot] = (EDGE){ v, head[u], cap, 0, cost }; head[u] = tot++;
   edge[tot] = (EDGE){ u, head[v], 0, 0, -cost }; head[v] = tot++;
```

```
}

int aug(int u, int flow) {
   if (u == tt) return flow; vis[u] = true;
   for (int i = cur[u]; ~i; i = edge[i].next) {
      int v = edge[i].to;
      if (edge[i].cap > edge[i].flow && !vis[v]
            && dis[u] == dis[v] + edge[i].cost) {
         int tmp = aug(v, min(flow, edge[i].cap - edge[i].flow));
         edge[i].flow += tmp, edge[i^1].flow -= tmp;
         cur[u] = i; if (tmp) return tmp;
      }
   }
   return 0;
}

bool modify_label() {
   int d = INF;
   for (int u = 0; u < N; u++) if (vis[u])
      for (int i = head[u]; ~i; i = edge[i].next) {
         int v = edge[i].to;
         if (edge[i].cap > edge[i].flow && !vis[v])
            d = min(d, dis[v] + edge[i].cost - dis[u]);
   }
   if (d == INF) return false;
   for (int i = 0; i < N; i++) if (vis[i])
      vis[i] = false, dis[i] += d;
   return true;
}

pair<int,int> zkw(int start, int end, int n) {
   ss = start, tt = end, N = n;
   int min_cost = 0, max_flow = 0;
   for (int i = 0; i < n; i++) dis[i] = 0;
   while (true) {
      for (int i = 0; i < n; i++) cur[i] = head[i];
      while (true) {
         for (int i = 0; i < n; i++) vis[i] = false;
         int tmp = aug(ss, INF); if (tmp == 0) break;
         max_flow += tmp, min_cost += tmp * dis[ss];
      }
      if (!modify_label()) break;
   }
   return make_pair(max_flow, min_cost);
}
```

11.4 匹配

11.4.1 匈牙利算法

```
const int MAXV = 1001, MAXE = 10001;
int to[MAXE], nex[MAXE], head[MAXV], tol;
int link[MAXV]; bool fl[MAXV];
void init() { memset(head, 0xff, sizeof(head)); }

void addedge(int u, int v) {
   ++tol; to[tol] = v, nex[tol] = head[u], head[u] = tol;
}

bool dfs(int u) {
   for (int i = head[u], v; ~i; i = nex[i]) if (!fl[v = to[i]]) {
      fl[v] = true;
      if (link[v] == -1 || dfs(link[v])) return link[v] = u, true;
   }
   return false;
}

int hungary(int n) {
   int tans = 0;
   memset(link, -1, sizeof(link));
   for (int i = 1; i <= n; i++) {
      memset(fl, 0, sizeof(fl));
      if (dfs(i)) tans++;
   }
   return tans;
} // O(VE)
```

11.4.2 Hopcroft-Karp 算法

```
const int MAXN = 100050, INF = 0x3f3f3f3f;
vector<int> G[MAXN];
int uN, Mx[MAXN], My[MAXN], dx[MAXN], dy[MAXN], dis;
bool used[MAXN];

bool searchP() {
   queue<int> Q; dis = INF;
   memset(dx, -1, sizeof(dx)); memset(dy, -1, sizeof(dy));
   for (int i = 0; i < uN; i++)
      if (Mx[i] == -1) Q.push(i), dx[i] = 0;
   while (!Q.empty()) {
      int u = Q.front(); Q.pop();
      if (dx[u] > dis) break;
      for (auto v : G[u]) {
         if (dy[v] == -1) {
```

```
            dy[v] = dx[u] + 1;
            if (My[v] == -1) dis = dy[v];
            else dx[My[v]] = dy[v] + 1, Q.push(My[v]);
         }
      }
   }
   return dis != INF;
}

bool DFS(int u) {
   for (auto v : G[u]) {
      if (!used[v] && dy[v] == dx[u] + 1) {
         used[v] = true;
         if (My[v] != -1 && dy[v] == dis) continue;
         if (My[v] == -1 || DFS(My[v])) return My[v] = u, Mx[u] = v, true;
      }
   }
   return false;
}

int MaxMatch() {
   int res = 0;
   memset(Mx, -1, sizeof(Mx)); memset(My, -1, sizeof(My));
   while (searchP()) {
      memset(used, false, sizeof(used));
      for (int i = 0; i < uN; i++) if (Mx[i] == -1 && DFS(i)) res++;
   }
   return res;
} // O(sqrt(V)E)，先初始化uN为出发边集合的个数
```

11.4.3 二分图多重匹配

尽量转换为网络流拆点、费用流拆点求解。当 MAXM 较小时可以考虑用下面这个。

```
const int MAXN = 100010, MAXM = 11;
int uN, vN, g[MAXN][MAXM], linker[MAXM][MAXN], num[MAXM];
bool used[MAXM];

bool dfs(int u) {
   for (int v = 0; v < vN; v++) if (g[u][v] && !used[v]) {
      used[v] = true;
      if (linker[v][0] < num[v])
         return linker[v][++linker[v][0]] = u, true;
      for (int i = 1; i <= num[v]; i++)
         if (dfs(linker[v][i])) return linker[v][i] = u, true;
   }
   return false;
}
```

```

int hungary() {
   int res = 0;
   for (int i = 0; i < vN; i++) linker[i][0] = 0;
   for (int u = 0; u < uN; u++) {
      memset(used, false, sizeof(used));
      if (dfs(u)) res++;
   }
   return res;
}
```

11.4.4 二分图最大权匹配 KM 算法

```
const int N = 410, INF = 0x3f3f3f3f;
int linker[N], nx, ny; bool vis[N];
int g[N][N], lx[N], ly[N], pre[N], slack[N];

void bfs(int k) {
   int px, py = 0, yy = 0, d;
   for (int i = 1; i <= ny; i++)
      pre[i] = 0, slack[i] = INF;
   linker[py] = k;

   while (linker[py]) {
      px = linker[py], d = INF, vis[py] = 1;
      for (int i = 1; i <= ny; i++) if (!vis[i]) {
         if (slack[i] > lx[px] + ly[i] - g[px][i])
            slack[i] = lx[px] + ly[i] - g[px][i], pre[i] = py;
         if (slack[i] < d) d = slack[i], yy = i;
      }
      for (int i = 0; i <= ny; i++) {
         if (vis[i]) lx[linker[i]] -= d, ly[i] += d;
         else slack[i] -= d;
      }
      py = yy;
   }

   while (py) linker[py] = linker[pre[py]], py = pre[py];
}

int km() {
   for (int x = 1; x <= nx; x++) lx[x] = 0;
   for (int y = 1; y <= ny; y++) ly[y] = linker[y] = 0;
   for (int x = 1; x <= nx; x++)
      fill(vis+1, vis+1+ny, false), bfs(x);
   int ans = 0;
   for (int y = 1; y <= ny; y++)
```

```
35      if (linker[y]) ans += g[linker[y]][y];
36   return ans;
37 } // O(nx*nx*ny)，若求最小权匹配可以权值取反结果取反
```

11.4.5 KM 算法解决一类不等式问题

二分图中的两部分顶点组成的集合分别为 X,Y，每个顶点存在一个变量 x_u，给定一些形如以下的限制条件，$x_v + x_u >= wi$，其中 v 属于 X，u 属于 Y，求变量之和的最小值。对于每个条件连权值为 wi 的边，KM 算法结束后的 lx，ly 数组的和即为答案。

11.4.6 Hall 定理

二分图中的两部分顶点组成的集合分别为 X,Y，则有一组无公共点的边，一端恰好为组成 X 点的充分必要条件是：X 中的任意 k 个点至少与 Y 中的 k 个点相邻。对于区间图只需要考虑计算情况，线段树维护。

11.4.7 一般图匹配带花树

```
1  const int MAXN = 250; // N: 点的个数，点的编号从1到N
2  int N, Q[MAXN], head, tail, match[MAXN], fa[MAXN], base[MAXN];
3  bool G[MAXN][MAXN], inq[MAXN], inb[MAXN], inp[MAXN];
4  void push(int u) { Q[tail++] = u, inq[u] = true; }
5
6  int findca(int u, int v, int low) {
7     memset(inp, false, sizeof(inp));
8     for (; u = base[u], inp[u] = true, u != low; u = fa[match[u]]);
9     for (v = base[v]; !inp[v]; v = base[fa[match[v]]]);
10    return v;
11 }
12
13 void rst(int u, int nb) {
14    while (base[u] != nb) {
15       int v = match[u]; inb[base[u]] = inb[base[v]] = true;
16       u = fa[v]; if (base[u] != nb) fa[u] = v;
17    }
18 }
19
20 int aug(int start) {
21    fill(inq, inq+N+1, false), fill(fa, fa+N+1, 0);
22    for (int i = 1; i <= N; i++) base[i] = i;
23    head = tail = 1; push(start);
24    while (head < tail) for (int u = Q[head++], v = 1; v <= N; v++) {
25       if (!G[u][v] || base[u] == base[v] || match[u] == v) continue;
26       if ((v == start) || ((match[v] > 0) && fa[match[v]] > 0)) {
27          int newbase = findca(u, v, start);
28          memset(inb, false, sizeof(inb));
29          rst(u, newbase), rst(v, newbase);
30          if (base[u] != newbase) fa[u] = v;
31          if (base[v] != newbase) fa[v] = u;
```

```
32          for (int tu = 1; tu <= N; tu++) if(inb[base[tu]]) {
33              base[tu] = newbase; if(!inq[tu]) push(tu);
34          }
35       } else if(fa[v] == 0) {
36          fa[v] = u;
37          if (match[v]) push(match[v]);
38          else return v;
39       }
40    }
41    return 0;
42 }
43
44 void Edmonds() {
45    memset(match, 0, sizeof(match));
46    for (int u = 1; u <= N; u++) if (match[u] == 0)
47       for (int x = aug(u), v, w; x > 0; x = w)
48          v = fa[x], w = match[v], match[v] = x, match[x] = v;
49    int cnt = 0;
50    for (int u = 1; u <= N; u++) if (match[u] > 0) cnt++;
51    printf("%d\n", cnt); // 匹配数，匹配对数为cnt/2
52    for (int u = 1; u <= N; u++)
53       if (u < match[u]) printf("%d %d\n", u, match[u]);
54 }
```

11.4.8 一般图最大权匹配

```
1  const int MAXN = 110, INF = 0x3f3f3f3f;
2  int G[MAXN][MAXN], N, dis[MAXN], match[MAXN]; // 先凑N为偶数
3  bool vis[MAXN]; int sta[MAXN], top, P[MAXN];
4  inline void links(int u, int v) { match[u] = v, match[v] = u; }
5
6  bool dfs(int u) {
7     sta[top++] = u;
8     if (vis[u]) return true; vis[u] = true;
9     for (int i = 0; i < N; i++) {
10       if (i == u || i == match[u] || vis[i]) continue;
11       int t = match[i];
12       if (dis[t] < dis[u] + G[u][i] - G[i][t]) {
13          dis[t] = dis[u] + G[u][i] - G[i][t];
14          if (dfs(t)) return true;
15       }
16    }
17    return top--, vis[u] = false;
18 }
19
20 int getMatch() {
21    for (int i = 0; i < N; i++) P[i] = i;
```

```
    for (int i = 0; i < N; i+=2) links(i, i+1);
    int cnt = 0;
    while (true) {
        memset(dis, 0, sizeof(dis));
        memset(vis, false, sizeof(vis));
        top = 0; bool update = false;
        for (int i = 0; i < N; i++) if (dfs(P[i])) {
            update = true;
            int tmp = match[sta[top-1]], j = top-2;
            while (sta[j] != sta[top-1])
                match[tmp] = sta[j], swap(tmp, match[sta[j]]), j--;
            links(sta[j], tmp);
            break;
        }
        if (!update) {
            if ((++cnt) >= 3) break;
            random_shuffle(P, P+N);
        }
    }
    int ans = 0;
    for (int i = 0, v; i < N; i++)
        if (i < (v = match[i])) ans += G[i][v];
    return ans;
}
```

11.5 连通性

11.5.1 割点与桥

```
#include<cstdio>
#include<cstring>
#define MAXN 100010
#define MAXM 200010
using namespace std;
struct Tarjan{
    int head[MAXN],node[MAXM*2],nex[MAXM*2],top,n;

    void clr(){
        top=0;
        memset(head,-1,sizeof(head));
        memset(nex,0,sizeof(nex));
    }
    void addedge(int u,int v){
        node[top]=v; nex[top]=head[u]; head[u]=top++;
        node[top]=u; nex[top]=head[v]; head[v]=top++;
    }

```

```
19    int cut[MAXN],birl[MAXM],birr[MAXM],low[MAXN],dfn[MAXN],vis[MAXN],times; //
          vis=0 not visited vis=1 visiting vis=2 visited
20
21    int dfs(int u,int p,int root){
22       low[u]=dfn[u]=++times; vis[u]=1;
23       int rc=0;
24       for (int tmp=head[u]; tmp!=-1; tmp=nex[tmp]){
25          int v=node[tmp];
26          if (!dfn[v]) {
27             dfs(v,tmp,root); rc++;
28             if (u!=root && low[v]>=dfn[u]) cut[++cut[0]]=u;
29             if (low[v]>dfn[u]) birl[++birl[0]]=u,birr[++birr[0]]=v;
30             if (low[v]<low[u]) low[u]=low[v];
31          }else if (vis[v]==1 && dfn[v]<low[u] && (tmp^1)!=p) low[u]=dfn[v];
32       }
33       vis[u]=2;
34       return rc;
35    }
36
37    void get_cut_bri(){
38       times=0; cut[0]=0; birl[0]=0; birr[0]=0;
39       memset(low,0,sizeof(low));
40       memset(dfn,0,sizeof(dfn));
41       memset(vis,0,sizeof(vis));
42       for (int i=1; i<=n; i++){
43          int cnt;
44          if (!dfn[i]){
45             cnt=dfs(i,-1,i);
46             if (cnt>=2) cut[++cut[0]]=i;
47          }
48       }
49    }
50
51 }G;
52 int n,m;
53 int main(){
54    scanf("%d%d",&n,&m);
55    G.n=n; G.clr();
56    for (int i=1; i<=m; i++){
57       int u,v;
58       scanf("%d%d",&u,&v);
59       G.addedge(u,v);
60    }
61    G.get_cut_bri();
62    for (int i=1; i<=G.cut[0]; i++) printf("%d\n",G.cut[i]);
63    for (int i=1; i<=G.birl[0]; i++) printf("%d %d\n",G.birl[i],G.birr[i]);
64    return 0;
65 }
```

11.5.2 Tarjan 缩点

```
const int MAXE = 50010, MAXV = 10010;
int to[MAXE], nex[MAXE], head[MAXV], tot;
int dfn[MAXV], low[MAXV], Stack[MAXV], top, deep;
int cnt[MAXV], color[MAXV], scc; bool vis[MAXV];
// cnt[1~scc]表示每个SCC点个数, color[1..V]表示属于哪个SCC

void init() {
   memset(head, 0xff, sizeof(head));
   memset(dfn, 0, sizeof(dfn));
   scc = 0;
}

void addedge(int u, int v) {
   nex[++tot] = head[u]; head[u] = tot; to[tot] = v;
}

void tarjanDFS(int u) {
   dfn[u] = ++deep, low[u] = deep, vis[u] = true, Stack[++top] = u;

   for (int i = head[u], v; ~i; i = nex[i])
      if (!dfn[v = to[i]]) tarjanDFS(v), low[u] = min(low[u], low[v]);
      else if (vis[v]) low[u] = min(low[u], low[v]);

   if (dfn[u] == low[u]) {
      color[u] = ++scc, cnt[scc] = 1, vis[u] = false;
      while (Stack[top] != u)
         color[Stack[top]] = scc, cnt[scc]++,
         vis[Stack[top--]] = false;
      top--;
   }
}

void tarjan(int n) {
   for (int i = 1; i <= n; i++)
      if (!dfn[i]) tarjanDFS(i);
}
```

11.5.3 Kosaraju 缩点

```
const int MAXN = 20010, MAXM = 50010;
struct Edge { int to, next; } E1[MAXM], E2[MAXM];
int head1[MAXN], head2[MAXN], tot1, tot2, cnt1, cnt2;
bool mark1[MAXN], mark2[MAXN];
```

```
int st[MAXN], color[MAXN], num[MAXN], __num;
// SCC count: cnt2, 编号0~cnt2-1

void addedge(int u, int v) {
   E1[tot1].to = v, E1[tot1].next = head1[u], head1[u] = tot1++;
   E2[tot2].to = u, E2[tot2].next = head2[v], head2[u] = tot2++;
}

void DFS1(int u) {
   mark1[u] = true;
   for (int i = head1[u]; ~i; i = E1[i].next)
      if (!mark1[E1[i].to]) DFS1(E1[i].to);
   st[cnt1++] = u;
}

void DFS2(int u) {
   mark2[u] = true; __num++; color[u] = cnt2;
   for (int i = head2[u]; ~i; i = E2[i].next)
      if (!mark2[E2[i].to]) DFS2(E2[i].to);
}

void kosaraju(int n) {
   memset(mark1, 0, sizeof(mark1)), cnt1 = 0;
   memset(mark2, 0, sizeof(mark2)), cnt2 = 0;
   for (int i = 1; i <= n; i++) if (!mark1[i]) DFS1(i);
   for (int i = cnt1-1; i >= 0; i--) if (!mark2[st[i]])
      __num = 0, DFS2(st[i]), num[cnt2++] = __num;
}
```

11.5.4 2-SAT 强连通分量缩点法

```
// poj3678
#include<cstdio>
#include<vector>
#include<algorithm>
using namespace std;
const int N=2010;
const int M=4000010;
char op[5];
int times,cnt,n,m,e,top,sta[N],nd[M],nex[M],lnk[N],scc[N],dfn[N],low[N],val[N
   ],vis[N];
vector<int>ct[N];
void add(int u,int v){
   nd[++e]=v; nex[e]=lnk[u]; lnk[u]=e;
}
void dfs(int u){
   dfn[u]=low[u]=++times; sta[++top]=u;
```

```
16     for (int i=lnk[u]; i; i=nex[i])
17        if (!dfn[nd[i]]) dfs(nd[i]),low[u]=std::min(low[u],low[nd[i]]);
18        else if (!scc[nd[i]]) low[u]=std::min(low[u],dfn[nd[i]]);
19     if (dfn[u]==low[u]){
20        int v=sta[top--]; scc[v]=++cnt; ct[cnt].push_back(v);
21        while (v!=u) v=sta[top--],scc[v]=cnt,ct[cnt].push_back(v);
22     }
23  }
24  void solve(){
25     for (int i=1; i<=cnt; i++)
26        if (!vis[i]) for (vector<int>::iterator it =ct[i].begin(); it!=ct[i].end
              (); it++) val[*it]=1,vis[scc[*it^1]]=1;
27  }
28  int main(){
29     scanf("%d%d",&n,&m);
30     for (int i=1,a,b,c; i<=m; i++){
31        scanf("%d%d%d%s",&a,&b,&c,op);
32        if (op[0]=='A') c?(add(a<<1,a<<1|1),add(b<<1,b<<1|1)):
33                        (add(a<<1|1,b<<1),add(b<<1|1,a<<1));
34        if (op[0]=='O') c?(add(a<<1,b<<1|1),add(b<<1,a<<1|1)):
35                        (add(a<<1|1,a<<1),add(b<<1|1,b<<1));
36        if (op[0]=='X') c?(add(a<<1,b<<1|1),add(b<<1,a<<1|1),add(a<<1|1,b<<1),
              add(b<<1|1,a<<1)):
37                        (add(a<<1|1,b<<1|1),add(b<<1|1,a<<1|1),add(a<<1,b<<1),add(
                             b<<1,a<<1));
38     }
39     fclose(stdin);
40     for (int i=0; i<(n<<1); i++) if (!dfn[i]) dfs(i);
41     for (int i=0; i<n; i++)
42        if (scc[i<<1]==scc[i<<1|1]){
43           puts("NO");
44           return 0;
45        }
46     puts("YES");
47  // solve();
48  // for (int i=0; i<(n<<1); i++) if (val[i]) printf("%d",i&1);
49     return 0;
50  }
```

第十二章　树论

12.1　最近公共祖先

12.1.1　RMQ 求 LCA

```
int to[MAXN<<1], nex[MAXN<<1], head[MAXN], tol, tot;
int idx[MAXN<<1], st[18][MAXN<<1], dfn[MAXN], dep[MAXN], lg2[MAXN<<1];

void addedge(int u,int v) {
   to[++tol] = v, nex[tol] = head[u], head[u] = tol;
   to[++tol] = u, nex[tol] = head[v], head[v] = tol;
}

void dfs(int u, int fa) {
   idx[++tot] = u, dfn[u] = tot;
   for (int i = head[u], v; i; i = nex[i]) if ((v = to[i]) != fa)
      dep[v] = dep[u]+1, dfs(v, u), idx[++tot] = u;
}

void init_lca_rmq(int rt) {
   dfs(rt, 0); st[0][1] = rt;
   for (int i = 2; i <= tot; i++)
      st[0][i] = idx[i], lg2[i] = lg2[i>>1]+1;
   for (int i = 1; i < 18; i++)
      for (int j = 1, k = 1+(1<<(i-1)); j <= tot; j++, k++)
         st[i][j] = (k>tot || dep[st[i-1][j]]<dep[st[i-1][k]])
               ? st[i-1][j] : st[i-1][k];
}

int lca(int u, int v) {
   if (dfn[u] > dfn[v]) swap(u, v);
   int l = lg2[dfn[v]-dfn[u]+1];
   int x = st[l][dfn[u]], y = st[l][dfn[v]-(1<<l)+1];
   return dep[x] < dep[y] ? x : y;
}

int dist(int u, int v) {
   return dep[u]+dep[v]-(dep[lca(u,v)]<<1);
}
```

12.1.2 倍增 LCA

```
int to[MAXN<<1], nex[MAXN<<1], head[MAXN], tol;
int fa[MAXN][LOG], dep[MAXN],n,m;

void addedge(int u, int v) {
   to[++tol] = v, nex[tol] = head[u], head[u] = tol;
   to[++tol] = u, nex[tol] = head[v], head[v] = tol;
}

void dfs(int u) {
   for (int i = head[u], v; i; i = nex[i]) {
      if ((v = to[i]) == fa[u][0]) continue;
      fa[v][0] = u;
      for (int j = 1; j < LOG; j++)
         fa[v][j] = fa[fa[v][j-1]][j-1];
      dep[v] = dep[u]+1;
      dfs(v);
   }
}

int lca(int u, int v) {
   if (dep[u] < dep[v]) swap(u, v);
   int delta = dep[u] - dep[v];
   for (int i = 0; i < LOG; i++) if ((delta>>i)&1) u = fa[u][i];
   if (u == v) return u;
   for (int i = LOG-1; ~i; i--)
      if (fa[u][i] != fa[v][i]) u = fa[u][i], v = fa[v][i];
   return fa[u][0];
}
```

12.1.3 树上路径交

```
pair<int,int> intersect(int a, int b, int c, int d) {
   int x = lca(a, b), y1 = lca(a, c), z1 = lca(b, c);
   int y2 = lca(a, d), z2 = lca(b, d);
   return make_pair(x^y1^z1, x^y2^z2);
}
```

12.2 树链剖分

12.2.1 Claris 树链剖分

```
void dfs(int x) {
   size[x] = 1;
   for (int i = g[x]; i; i = nxt[i]) if (v[i] != f[x]) {
      f[v[i]] = x, d[v[i]] = d[x] + 1;
```

```
5       dfs(v[i]), size[x] += size[v[i]];
6       if (size[v[i]] > size[son[x]]) son[x] = v[i];
7    }
8 }
9
10 void dfs2(int x, int y) {
11    st[x] = ++dfn; top[x] = y;
12    if (son[x]) dfs2(son[x], y);
13    for (int i = g[x]; i; i = nxt[i]) if (v[i] != son[x] && v[i] != f[x]) dfs2(
         v[i], v[i]);
14    en[x] = dfn;
15 }
16
17 // 查询x,y两点的lca
18 int lca(int x, int y) {
19    for (; top[x] != top[y]; x = f[top[x]])
20       if (d[top[x]] < d[top[u]]) swap(x, y);
21    return d[x] < d[y] ? x : y;
22 }
23
24 // x是y的祖先，查询x到y方向的第一个点
25 int lca2(int x, int y) {
26    int t; while (top[x] != top[y]) t = top[y], y = f[top[y]];
27    return x == y ? t : son[x];
28 }
29
30 // 以root为根对x的子树操作
31 void subtree(int x) {
32    if (x == root) { change(1,n); return; }
33    if (st[x] > st[root] || en[x] < en[root]) { change(st[x], en[x]); return; }
34    int y = lca2(x, root); change(1, st[y]-1), change(en[y]+1, n);
35 }
36
37 // 对x到y路径上的点进行操作
38 void chain(int x, int y) {
39    for (; top[x] != top[y]; x = f[top[x]]) {
40       if (d[top[x]] < d[top[y]]) swap(x, y);
41       change(st[top[x]], st[x]);
42    }
43    if (d[x] < d[y]) swap(x, y);
44    change(st[y], st[x]);
45 }
```

12.2.2 Link Cut Tree

```
1 #include<bits/stdc++.h>
2 #define rep(i,a,b) for(int i=(a);i<=(b);i++)
```

```
3  #define per(i,a,b) for(int i=(a);i>=(b);i--)
4  using namespace std;
5  typedef long long int64;
6  pair<int,int> pin;
7  const int maxn=200005;
8  inline void read(int &x){
9     x=0;int f=1;char ch=getchar();
10    while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}
11    while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}
12    x*=f;
13 }
14 inline void judge(){
15    freopen("in.txt","r",stdin);
16    freopen("out.txt","w",stdout);
17 }
18 struct lct{
19    int rev[maxn],ch[maxn][2],fa[maxn],S[maxn],top,size[maxn];
20    inline bool isroot(int x){return ch[fa[x]][0]!=x&&ch[fa[x]][1]!=x;}
21    inline void update(int x){size[x]=size[ch[x][0]]+size[ch[x][1]]+1;}
22    inline void pushdown(int x){if(rev[x])rev[x]^=1,rev[ch[x][1]]^=1,rev[ch[x
         ][0]]^=1,swap(ch[x][0],ch[x][1]);}
23    inline void rotate(int x){
24       int y=fa[x],z=fa[y],d=(ch[y][1]==x);
25       if(!isroot(y))ch[z][ch[z][1]==y]=x;fa[x]=z;fa[y]=x;ch[y][d]=ch[x][!d];fa
            [ch[x][!d]]=y;ch[x][!d]=y;
26       update(y);update(x);
27    }
28    inline void splay(int x){
29       S[top=1]=x;update(x);for(int i=x;!isroot(i);i=fa[i])S[++top]=fa[i],
            update(fa[i]);per(i,top,1)pushdown(S[i]);
30       for(int y=fa[x],z=fa[y];!isroot(x);y=fa[x],z=fa[y]){if(!isroot(y))((ch[y
            ][0]==x)^(ch[z][0]==y))?rotate(x):rotate(y);rotate(x);}
31    }
32    inline void access(int x){int t=0;while(x){splay(x);ch[x][1]=t;t=x;x=fa[x
         ];}}
33    inline void makeroot(int x){access(x);splay(x);rev[x]^=1;}
34    inline void link(int x,int y){makeroot(x);fa[x]=y;}
35    inline void cut(int x,int y){makeroot(x);access(y);splay(x);ch[x][1]=fa[y
         ]=0;}
36    inline int find(int x){access(x);splay(x);for(;ch[x][0];x=ch[x][0]);return
         x;}
37 }lct;
38 int t[maxn];
39 int main(){
40 // judge();
41    int n;read(n);
42    rep(i,1,n){int x;read(x);t[i]=min(n+1,i+x);lct.link(i,t[i]);}
43    int m;read(m);lct.makeroot(n+1);
```

```
    while(m--){
        int x,y;read(x);read(y);y++;lct.makeroot(n+1);
        if (x==1){
            lct.access(y);lct.splay(y);printf("%d\n",lct.size[y]-1);
        }
        if (x==2){
            int k;read(k);
            lct.cut(y,t[y]);t[y]=min(n+1,y+k);
            lct.link(y,t[y]);
        }
    }
    return 0;
}
```

12.2.3 LCT2

```
#include<cstdio>
#include<algorithm>
using namespace std;
const int N=300010;
struct lct{
    int ch[N][2],f[N],rev[N],val[N],s[N],S[N],top;
    bool isroot(int x){
        return ch[f[x]][0]!=x && ch[f[x]][1]!=x;
    }
    void pushdown(int x){
        if (rev[x]){
            rev[x]=0; rev[ch[x][0]]^=1; rev[ch[x][1]]^=1; swap(ch[x][0],ch[x][1])
                ;
        }
    }
    void update(int x){
        s[x]=val[x]^s[ch[x][0]]^s[ch[x][1]];
    }
    void rotate(int x){
        int y=f[x],z=f[y]; int d=(ch[y][1]==x);
        if (!isroot(y)) ch[z][ch[z][1]==y]=x;
        f[x]=z; f[y]=x; f[ch[x][d^1]]=y;
        ch[y][d]=ch[x][d^1]; ch[x][d^1]=y;
        update(y); update(x);
    }
    void splay(int x){
        S[top=1]=x; int t=x; while (!isroot(t)) t=f[t],S[++top]=t;
        for (int i=top; i>=1; i--) pushdown(S[i]);
        for (int y=f[x],z=f[y]; !isroot(x); y=f[x],z=f[y]){
            if (!isroot(y)) (ch[z][1]==y ^ ch[y][1]==x)?rotate(x):rotate(y);
            rotate(x);
```

```
31          }
32      }
33      void access(int x){
34          int t=0; while (x){ splay(x); ch[x][1]=t;update(x); t=x; x=f[x]; }
35      }
36      void makeroot(int x){
37          access(x); splay(x); rev[x]^=1;
38      }
39      int findrt(int x){
40          access(x); splay(x); while(ch[x][0]) x=ch[x][0],pushdown(x);
41          return x;
42      }
43      void lnk(int x,int y){
44          makeroot(x); f[x]=y;
45      }
46      void cut(int x,int y){
47          makeroot(x); access(y); splay(x);
48          if (ch[x][1]==y) ch[x][1]=f[y]=0;
49          update(x);
50      }
51  }lct;
52  int main(){
53      int n,m;
54      scanf("%d%d",&n,&m);
55      for (int i=1; i<=n; i++) scanf("%d",&lct.val[i]);
56      for (int i=1,opt,x,y; i<=m; i++){
57          scanf("%d%d%d",&opt,&x,&y);
58          if (opt==0){
59              lct.makeroot(x); lct.access(y); lct.splay(x); printf("%d\n",lct.s[x])
                    ;
60          }
61          if (opt==1){
62              if (lct.findrt(x) != lct.findrt(y)) lct.lnk(x,y);
63          }
64          if (opt==2) lct.cut(x,y);
65          if (opt==3) lct.splay(x),lct.val[x]=y,lct.update(x);
66      }
67      return 0;
68  }
```

12.2.4 线段树维护直径

```
1  const int N = 2e5+9;
2  struct edge { int to,nex; lld w; } e[N<<1];
3  int h[N], f[N][25], dep[N], st2[N], ed2[N];
4  lld tr2[N], dis[N];
5  int rnk[N<<3], st1[N<<3], ed1[N<<3];
```

```
6  int cnt = 1, n, m, tot1 = 0, tot2 = 0;
7
8  struct node {
9     lld mx, mi, lmx, rmx, val, lazy;
10    int idl, idr, idmx, u, v;
11
12    void pd(lld tem) {
13       lazy += tem, mx += tem, mi += tem;
14       lmx -= tem, rmx -= tem;
15    }
16 } tr[N<<2];
17
18 void add(int u, int v, lld wi) {
19    e[++cnt] = edge { v, h[u], wi }; h[u] = cnt;
20    e[++cnt] = edge { u, h[v], wi }; h[v] = cnt;
21 }
22
23 void dfs(int u, int fa) {
24    st1[u] = ++tot1; st2[u] = ++tot2; rnk[tot1] = u;
25    f[u][0] = fa; dep[u] = dep[fa]+1;
26    for (int i = 1; i <= 20; ++i) f[u][i] = f[f[u][i-1]][i-1];
27    for (int i = h[u], v; i; i = e[i].nex) {
28       if ((v = e[i].to) == fa) continue;
29       dis[v] = dis[u] + e[i].w; dfs(v,u); rnk[++tot1] = u;
30    }
31    ed1[u] = tot1; ed2[u] = tot2;
32 }
33
34 void push_up(int o) {
35    tr[o].mx = max(tr[o<<1].mx, tr[o<<1|1].mx);
36    tr[o].mi = min(tr[o<<1].mi, tr[o<<1|1].mi);
37    tr[o].lmx = max(max(tr[o<<1].lmx, tr[o<<1|1].lmx),
38       tr[o<<1].mx - 2 * tr[o<<1|1].mi);
39    tr[o].rmx=max(max(tr[o<<1].rmx, tr[o<<1|1].rmx),
40       tr[o<<1|1].mx - 2 * tr[o<<1].mi);
41    tr[o].val = max(max(tr[o<<1].val, tr[o<<1|1].val),
42       max(tr[o<<1].lmx + tr[o<<1|1].mx, tr[o<<1].mx + tr[o<<1|1].rmx));
43
44    if (tr[o].lmx == tr[o<<1].lmx) tr[o].idl = tr[o<<1].idl;
45    else if (tr[o].lmx == tr[o<<1|1].lmx) tr[o].idl = tr[o<<1|1].idl;
46    else tr[o].idl = tr[o<<1].idmx;
47
48    if (tr[o].rmx == tr[o<<1].rmx) tr[o].idr = tr[o<<1].idr;
49    else if (tr[o].rmx == tr[o<<1|1].rmx) tr[o].idr = tr[o<<1|1].idr;
50    else tr[o].idr = tr[o<<1|1].idmx;
51
52    if (tr[o].mx == tr[o<<1].mx) tr[o].idmx = tr[o<<1].idmx;
53    else tr[o].idmx = tr[o<<1|1].idmx;
```

```
54
55    if (tr[o].val == tr[o<<1].val)
56       tr[o].u = tr[o<<1].u, tr[o].v = tr[o<<1].v;
57    else if (tr[o].val == tr[o<<1|1].val)
58       tr[o].u = tr[o<<1|1].u, tr[o].v = tr[o<<1|1].v;
59    else if (tr[o].val == tr[o<<1].lmx + tr[o<<1|1].mx)
60       tr[o].u = tr[o<<1].idl, tr[o].v = tr[o<<1|1].idmx;
61    else tr[o].u = tr[o<<1].idmx, tr[o].v = tr[o<<1|1].idr;
62 }
63
64 void push_down(int o) {
65    if (tr[o].lazy)
66       tr[o<<1].pd(tr[o].lazy), tr[o<<1|1].pd(tr[o].lazy);
67    tr[o].lazy = 0;
68 }
69
70 void build(int o, int l, int r) {
71    if (l == r) {
72       tr[o].idl = tr[o].idr = tr[o].idmx = rnk[l];
73       lld d = dis[rnk[l]];
74       tr[o].mx = tr[o].mi = d;
75       tr[o].lmx = tr[o].rmx = -d;
76       tr[o].lazy = tr[o].val = 0;
77       tr[o].u = tr[o].v = rnk[l];
78    } else {
79       tr[o].lazy = 0;
80       int mid = (l+r)>>1;
81       build(o<<1, l, mid), build(o<<1|1, mid+1, r);
82       push_up(o);
83    }
84 }
85
86 void change(int o, int l, int r, int x, int y, lld w) {
87    if (x <= l && r <= y) { tr[o].pd(w); return; }
88    push_down(o); int mid = (l+r)>>1;
89    if (x <= mid) change(o<<1, l, mid, x, y, w);
90    if (y > mid) change(o<<1|1, mid+1, r, x, y, w);
91    push_up(o);
92 }
93
94 void add2(int x, lld v) { for (;x<=n;x+=x&(-x)) tr2[x]+=v; }
95 lld sum2(int x) { lld res=0; for(;x;x-=x&(-x)) res+=tr2[x]; return res; }
96
97 int lca(int u, int v) {
98    if (dep[u] < dep[v]) swap(u, v);
99    for (int i = 20; i >= 0; --i)
100      if (dep[f[u][i]] >= dep[v]) u = f[u][i];
101   if (u == v) return u;
```

```
102    for (int i = 20; i >= 0; --i)
103       if (f[u][i] != f[v][i]) u = f[u][i], v = f[v][i];
104    return f[u][0];
105 }
106
107 lld cal(int u, int v) {
108    return sum2(st2[u]) + sum2(st2[v]) - 2 * sum2(st2[lca(u,v)]);
109 }
110
111 int main() {
112    int u, v, x; lld w, y; scanf("%d", &n);
113    for (int i = 1; i < n; ++i)
114       scanf("%d %d %lld", &u, &v, &w), add(u, v, w);
115    dfs(1,0);
116    for (int i = 1; i <= n; ++i)
117       add2(st2[i], dis[i]), add2(st2[i]+1, -dis[i]);
118    build(1, 1, tot1);
119
120    scanf("%d", &m); while (m--) {
121       char s[3]; scanf("%s", s);
122       if (s[0]=='Q') {
123          scanf("%d", &x);
124          u = tr[1].u, v=tr[1].v;
125          printf("%lld\n", max(cal(u,x), cal(v,x)));
126       }
127       else if (s[0] == 'C') {
128          scanf("%d %lld", &x, &y);
129          x *= 2; u = e[x].to, v = e[x|1].to;
130          u= dep[u] > dep[v] ? u : v;
131          change(1, 1, tot1, st1[u], ed1[u], y-e[x].w);
132          add2(st2[u], y-e[x].w), add2(ed2[u]+1, e[x].w-y);
133          e[x].w = e[x|1].w = y;
134       }
135    }
136    return 0;
137 }
```

12.3 树分治

12.3.1 点分治

```
1 #include<cstdio>
2 #include<algorithm>
3 #include<queue>
4 using namespace std;
5 const int N=100010;
6 const int M=500010;
```

```
7  const int INF=100000000;
8  int n,m,T;
9  void r(int &x){char ch=getchar(); x=0; while (ch<'0' || ch>'9')ch=getchar();
       while (ch>='0' && ch<='9') x=x*10+ch-'0',ch=getchar();}
10 void rc(char &ch){ch=getchar(); while (ch!='C' && ch!='G')ch=getchar();}
11 struct heap{
12    priority_queue<int> A,B;
13    void push(int x){A.push(x);}
14    void erase(int x){B.push(x);}
15    void pop(){while(B.size()&&A.top()==B.top()) A.pop(),B.pop();A.pop();}
16    int top(){while(B.size()&&A.top()==B.top())A.pop(),B.pop();if(!A.size())
          return 0;return A.top();}
17    int size(){return A.size()-B.size();}
18    int stop(){if(size()<2)return 0;int x=top();pop();int y=top();push(x);
          return y;}
19 }Ans,A[100005],B[100005];
20 int e,lnk[N],nex[N<<1],node[N<<1];
21 int dep[N],sz[N],F[N],U[N],f[N][20],Max[N],root,sum;
22 bool vis[N],rev[N];
23 void Add(int u,int v){node[++e]=v; nex[e]=lnk[u]; lnk[u]=e; node[++e]=u; nex[e
       ]=lnk[v]; lnk[v]=e;}
24 void Init(int u,int fa){
25    for (int i=lnk[u]; i; i=nex[i]){
26       int v=node[i]; if (v==fa) continue;
27       dep[v]=dep[u]+1; f[v][0]=u;
28       for (int j=1; j<=18; j++) f[v][j]=f[f[v][j-1]][j-1];
29       Init(v,u);
30    }
31 }
32 int Lca(int u,int v){
33    if (dep[u]<dep[v]) swap(u,v); int delta=dep[u]-dep[v];
34    for (int j=0; j<=18; j++) if ((delta>>j)&1) u=f[u][j];
35    if (u==v) return u;
36    for (int j=18; j>=0; j--)
37    if (f[u][j]!=f[v][j])u=f[u][j],v=f[v][j];
38    return f[u][0];
39 }
40 int dist(int u,int v){return dep[u]+dep[v]-(dep[Lca(u,v)]<<1);}
41 void center(int u,int fa){
42    Max[u]=0; sz[u]=1;
43    for (int i=lnk[u]; i; i=nex[i]){
44       if (vis[node[i]] || node[i]==fa) continue;
45       center(node[i],u); sz[u]+=sz[node[i]];
46       if (sz[node[i]]>Max[u]) Max[u]=sz[node[i]];
47    }
48    Max[u]=max(Max[u],sum-sz[u]);
49    if (Max[u]<Max[root]) root=u;
50 }
```

```
void _Init(int u){
   vis[u]=1;
   for (int i=lnk[u]; i; i=nex[i]){
      if (vis[node[i]]) continue;
      sum=sz[node[i]]; root=0;
      center(node[i],u); F[root]=u; U[root]=node[i];
      _Init(root);
   }
}
void ins(int u,int d){
   int t=u,z=F[t]; d?T+=1:T-=1;
   while (z){
      if (B[z].size()>1) Ans.erase(B[z].top()+B[z].stop());
      if (A[t].size()) B[z].erase(A[t].top()+1);
      d?A[t].push(dist(U[t],u)):A[t].erase(dist(U[t],u));
      if (A[t].size()) B[z].push(A[t].top()+1);
      if (B[z].size()>1) Ans.push(B[z].top()+B[z].stop());
      t=z; z=F[t];
   }
}
int main(){
   r(n); for (int i=1,u,v; i<n; i++) r(u),r(v),Add(u,v);
   Init(1,0); sum=n; Max[0]=INF;
   center(1,0); U[root]=1; _Init(root);
   for (int i=1; i<=n; i++) ins(i,rev[i]^=1);
   r(m); char ch;
   for (int i=1,x; i<=m; i++){
      rc(ch);
      if (ch=='G'){
         if (T<=1) printf("%d\n",T-1);
         else printf("%d\n",Ans.top());
      }
      else r(x),ins(x,rev[x]^=1);
   }
   return 0;
}
```

12.3.2 树上莫队

COTII 询问链上不同颜色数量。

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e5+10;
vector<pair<int,int> >h[maxn];
vector<int> g[maxn];
int w[maxn],b[maxn],l[maxn],r[maxn],f[maxn],Lca[maxn],vis[maxn],flag[maxn],blk
   [maxn],n,m;
```

```
int cnt[maxn],lis[maxn],ans,dfn[maxn],Dfn[maxn],top,Ans[maxn];
struct point{
   int x,y,z;
   bool operator<(const point &b)const{
      return blk[x]<blk[b.x] || (blk[x]==blk[b.x] && y<b.y);
   }
}q[maxn];
int find(int x){
   return f[x]==x?x:f[x]=find(f[x]);
}
void ins(int p){
   ans-=(cnt[w[lis[p]]]>0);
   cnt[w[lis[p]]]+=vis[lis[p]];
   ans+=(cnt[w[lis[p]]]>0);
   vis[lis[p]]*=-1;
}
void dfs(int u,int fa){
   lis[++top]=u; dfn[u]=top;
   for (auto v:g[u]) if (v!=fa) dfs(v,u);
   vis[u]=1;
   for (auto p:h[u]) if (vis[p.first]) Lca[p.second]=find(p.first);
   f[u]=fa;
   lis[++top]=u; Dfn[u]=top;
}
int main(){
   scanf("%d%d",&n,&m);
   for (int i=1; i<=n; i++) scanf("%d",&w[i]),b[i]=w[i];
   for (int i=1,u,v; i<n; i++) scanf("%d%d",&u,&v),g[u].push_back(v),g[v].
      push_back(u);
   sort(b+1,b+1+n);
   for (int i=1; i<=n; i++) w[i]=lower_bound(b+1,b+1+n,w[i])-b;
   for (int i=1; i<=m; i++) scanf("%d%d",&l[i],&r[i]),h[l[i]].push_back(
      make_pair(r[i],i)),h[r[i]].push_back(make_pair(l[i],i));
   for (int i=1; i<=n; i++) f[i]=i;
   dfs(1,0);
   for (int i=1; i<=m; i++){
      if (Lca[i]==l[i]) q[i]=point{dfn[l[i]],dfn[r[i]],i};
      else if (Lca[i]==r[i]) q[i]=point{dfn[r[i]],dfn[l[i]],i};
      else if (dfn[l[i]]<dfn[r[i]]) q[i]=point{Dfn[l[i]],dfn[r[i]],i},flag[i
         ]=1;
      else q[i]=point{Dfn[r[i]],dfn[l[i]],i},flag[i]=1;
   }
   int s=sqrt(top);
   for (int i=1; i<=top; i++) blk[i]=i/s;
   sort(q+1,q+1+m);
   int pl=1,pr=0;
   for (int i=1; i<=m; i++){
      while (pl<q[i].x) ins(pl),pl++;
```

```
52        while (pl>q[i].x) pl--,ins(pl);
53        while (pr<q[i].y) pr++,ins(pr);
54        while (pr>q[i].y) ins(pr),pr--;
55        if (flag[q[i].z]) Ans[q[i].z]=ans+(cnt[w[Lca[q[i].z]]]==0); else Ans[q[i
              ].z]=ans;
56    }
57    for (int i=1; i<=m; i++) printf("%d\n",Ans[i]);
58    return 0;
59 }
```

12.3.3 DSU on tree

```
1  #pragma GCC optimize("Ofast")
2  #include<cstdio>
3  #include<vector>
4  #include<cstring>
5  using namespace std;
6  const int N=100010;
7  int lnk[N],nd[N<<1],nex[N<<1],f[N],sz[N],son[N],c[N],s[N],g[N],Ans[N],n,m,e;
8  vector<pair<int,int> >lis[N];
9  void r(int &x){
10    char c=getchar(); x=0;
11    while (c<'0' || c>'9') c=getchar();
12    while (c>='0' && c<='9') x=x*10+c-'0',c=getchar();
13 }
14 void add(int u,int v){
15    nd[++e]=v; nex[e]=lnk[u]; lnk[u]=e;
16 }
17 void dfs(int u){
18    sz[u]=1;
19    for (int i=lnk[u],v=nd[i]; i; i=nex[i],v=nd[i])
20       if (v!=f[u]){
21          f[v]=u; dfs(v);
22          sz[u]+=sz[v];
23          if (sz[v]>sz[son[u]]) son[u]=v;
24       }
25 }
26 void Ins(int u){
27    g[++s[c[u]]]++; for (int i=lnk[u]; i; i=nex[i]) if (nd[i]!=f[u]) Ins(nd[i])
          ;
28 }
29 void Clr(int u){
30    g[s[c[u]]--]--; for (int i=lnk[u]; i; i=nex[i]) if (nd[i]!=f[u]) Clr(nd[i])
          ;
31 }
32 void Dfs(int u,int top){
33    for (int i=lnk[u]; i; i=nex[i]) if (nd[i]!=son[u] && nd[i]!=f[u]) Dfs(nd[i
```

```
        ],nd[i]);
34    if (son[u]) Dfs(son[u],top);
35    g[++s[c[u]]]++;
36    for (int i=lnk[u]; i; i=nex[i]) if (nd[i]!=son[u] && nd[i]!=f[u]) Ins(nd[i
        ]);
37    for (int i=0; i<lis[u].size(); i++) Ans[lis[u][i].second]=g[lis[u][i].first
        ];
38    if (u==top) Clr(u);
39  }
40
41  int main(){
42    freopen("color.in","r",stdin);
43    freopen("color.out","w",stdout);
44    r(n); r(m);
45    for (int i=1; i<=n; i++) r(c[i]);
46    for (int i=1,u,v; i<n; i++){
47      r(u); r(v);
48      add(u,v);
49      add(v,u);
50    }
51    dfs(1);
52    for (int i=1,u,k; i<=m; i++){
53      r(u); r(k);
54      lis[u].push_back(make_pair(k,i));
55    }
56    Dfs(1,1);
57    for (int i=1; i<=m; i++) printf("%d\n",Ans[i]);
58    return 0;
59  }
```

12.4 虚树

12.4.1 倍增预处理

```
1   const int MAXN = 1e6+5;
2   int f[MAXN][21], dfn[MAXN], dep[MAXN], times, k;
3   pair<int,int> sec[MAXN]; bool res[MAXN];
4   #define X second
5
6   struct graph {
7     int lnk[MAXN], nex[MAXN<<1], node[MAXN<<1], tol;
8     void clear() { tol = 0; }
9     void addedge(int u,int v) {
10      node[++tol] = v, nex[tol] = lnk[u], lnk[u] = tol;
11      node[++tol] = u, nex[tol] = lnk[v], lnk[v] = tol;
12    }
13  } P, G;
```

```
14
15 void init(int u) {
16    dfn[u] = ++times;
17    for (int i = G.lnk[u]; i; i = G.nex[i]) if (G.node[i] != f[u][0]) {
18       int v = G.node[i]; dep[v] = dep[u]+1; f[v][0] = u;
19       for (int j = 1; j <= 20; j++) f[v][j] = f[f[v][j-1]][j-1];
20       init(v);
21    }
22 }
23
24 int lca(int u, int v) {
25    if (dep[u] < dep[v]) swap(u, v); int delta = dep[u] - dep[v];
26    for (int i = 0; i <= 20; i++) if ((delta>>i)&1) u = f[u][i];
27    if (u == v) return v;
28    for (int i = 20; i >= 0; i--)
29       if (f[u][i] != f[v][i]) u = f[u][i], v = f[v][i];
30    return f[u][0];
31 }
32
33 int markup(int x) { sec[++k] = make_pair(dfn[x], x); } // 标注关键点
34
35 int build_vt() {
36    static int stk[MAXN]; int top = 0; P.tol = 0;
37    sort(sec+1, sec+1+k); stk[++top] = sec[1].X;
38    for (int j = 2; j <= k; j++) {
39       int Lca = lca(stk[top], sec[j].X);
40       if (Lca != stk[top]) {
41          while (top > 1 && lca(stk[top-1], sec[j].X) != stk[top-1])
42             P.addedge(stk[top-1], stk[top]), top--;
43          Lca = lca(stk[top], sec[j].X);
44          P.addedge(Lca, stk[top]), top--;
45          if (stk[top] != Lca) stk[++top] = Lca, res[Lca] = 1;
46       }
47       stk[++top] = sec[j].X;
48    }
49    while (top > 1) P.addedge(stk[top-1], stk[top]), top--;
50    return stk[top];
51 } // sec中为虚树关键点，返回虚树根节点
52
53 int sz[MAXN], Max[MAXN], Min[MAXN], Mi, Ma; lld s[MAXN], Sum;
54
55 void dfs(int u, int fa) {
56    Min[u] = 1<<30, Max[u] = -(1<<30), s[u] = sz[u] = 0;
57    if (!res[u]) sz[u] = 1, Min[u] = 0, Max[u] = 0; else res[u] = 0;
58    for (int i = P.lnk[u]; i; i = P.nex[i]) if (P.node[i] != fa) {
59       int v = P.node[i]; dfs(v, u); int dist = dep[v] - dep[u];
60       Ma = max(Ma, Max[v] + dist + Max[u]);
61       Mi = min(Mi, Min[v] + dist + Min[u]);
```

```
62        Sum += (s[v] + 1LL * dist * sz[v]) * sz[u] + s[u] * sz[v];
63        s[u] += s[v] + 1LL * dist * sz[v]; sz[u] += sz[v];
64        Max[u] = max(Max[u], Max[v] + dist);
65        Min[u] = min(Min[u], Min[v] + dist);
66     }
67     P.lnk[u] = 0; // 这里清除了虚树
68  }
69
70  int main() {
71     int n, q; scanf("%d", &n);
72     for (int i = 1, u, v; i < n; i++)
73        scanf("%d %d", &u, &v), G.addedge(u,v);
74     init(1); scanf("%d", &q);
75     for (int i = 1, vv; i <= q; i++) {
76        scanf("%d", &vv); k = 0;
77        for (int j = 1, u; j <= vv; j++)
78           scanf("%d", &u), markup(u);
79        Ma = 0, Mi = 1<<30, Sum = 0;
80        int rt = build_vt(); dfs(rt, 0);
81        printf("%lld %d %d\n", Sum, Mi, Ma);
82     }
83     return 0;
84  }
```

12.4.2 线性预处理

```
1  struct virtual_tree {
2     int nxt[MAXN*2], to[MAXN*2], head[MAXN], tol, dep[MAXN]; bool mark[MAXN];
3     int nex[MAXN], son[MAXN], lnk[MAXN], fa[MAXN], tot, depV[MAXN], rt;
4     typedef function<void(int,int)> opr; // opr(int x, int cnt)
5
6     inline void init(int n) {
7        tol = tot = 0;
8        for (int i = 1; i <= n; i++)
9           mark[i] = head[i] = lnk[i] = fa[i] = dep[i] = depV[i] = 0;
10    } // 清空之前树的状态
11
12    inline void addedge(int u, int v) {
13       nxt[++tol] = head[u], head[u] = tol, to[tol] = v;
14       nxt[++tol] = head[v], head[v] = tol, to[tol] = u;
15    } // 添加实际树上的边
16
17    inline void addedge2(int u, int v) {
18       nex[++tot] = lnk[u], lnk[u] = tot, son[tot] = v;
19       // fprintf(stderr, "%d -> %d\n", u, v);
20    } // 添加虚树上的边
21
```

```
22    int dfs(int u, int p) {
23       dep[u] = dep[p]+1; int last = 0;
24       for (int i = head[u], g; i; i = nxt[i]) {
25          if (to[i] == p) continue;
26          if (!(g = dfs(to[i], u))) continue;
27          if (last == 0) last = g;
28          else if (last == u) addedge2(u, g);
29          else addedge2(u, last), addedge2(u, g), last = u;
30       }
31       if (mark[u] && last != 0 && last != u) addedge2(u, last);
32       return mark[u] ? u : last;
33    } // DFS建立虚树
34
35    void dfs2(int u, int d) {
36       depV[u] = d;
37       for (int i = lnk[u]; i; i = nex[i])
38          fa[son[i]] = u, dfs2(son[i], d+1);
39    } // 处理虚树中的深度关系
40
41    inline void markup(int u) { mark[u] = true; } // 标记虚树关键点
42    inline void process() { dfs2(rt = dfs(1, 0), 1); } // 预处理虚树
43
44    inline void links(int u, int v, const opr &rnode, const opr &vnode) {
45       #define up(x) (rnode(x,1), vnode(x,dep[x]-dep[fa[x]]-1), x=fa[x])
46       while (depV[u] > depV[v]) up(u); while (depV[v] > depV[u]) up(v);
47       while (u != v) up(u), up(v); rnode(u,1);
48    } // 处理 u->v 链上的内容
49 };
```

12.5 Prufer 序列

12.5.1 Prufer 编码

在一棵 n 个节点带标号树中，我们认为度数为 1 的点为叶子。n 个点的树都 Prufer 序列是经过下面流程得到的一个长度为 $n-2$ 的序列。

1. 若当前树中只剩下两个点，退出，否则执行 2。
2. 找到树中编号最小的叶节点，将与它相连的那个点的编号加入 Prufer 序列的末尾，并将这个叶子删除。返回 1。

显然，每棵树都唯一对应一个 Prufer 序列，而每个 Prufer 序列也唯一对应一棵树。可以通过以下流程得到这棵树。

1. 令 $A=\{1,2,\cdots,n\}$，不断重复 2 直到 Prufer 序列为空。
2. 找到 A 中最小的不在 Prufer 序列中的点，将其与 Prufer 序列首元素连边，然后同时删除这个点与序列首元素。
3. 此时 A 中还剩下两个点，将这两个点连边。

根据以上流程，不难发现：若点 i 在树中的度数为 a_i，则它会在 Prufer 序列中出现 a_i-1 次。

12.5.2 Cayley 公式

Prufer 序列中每个元素可以从 1 取到 n，且每种方案会唯一对应一棵带标号无根树。所以，由于 Prufer 序列共有 n^{n-2} 个，n 个点带标号无根树就有 n^{n-2} 种。

当树中每个点点度数 a_i 都已经确定后，由 Prufer 序列得满足条件的树共有 $\frac{(n-2)!}{\prod(a_i-1)!}$ 种。

n 个点组成共有 m 棵树的森林，且 $1,2,\cdots,m$ 分别属于不同的树，方案数为 $m\cdot n^{n-m-1}$。

对于 n 个点，m 个连通块的图，假设每个连通块有 a_i 个点，那么用 $s-1$ 条边把它连通的方案数为 $n^{s-2}a_1a_2\cdots a_m$。