

Nantong University ICPC Team Notebook

Simon

10/31/2019

目录

第一章	配置	5
1.1	VIM 配置	5
1.2	基础头文件	5
第二章	输入输出	7
2.1	取消同步	7
2.2	浮点数输出格式	7
2.3	整型快速输入	7
2.4	字符串快速输入	8
2.5	整型快速输出	8
2.6	字符串快速输出	9
2.7	python 输入	9
2.8	java 输入	9
2.9	int128 输入输出	9
第三章	搜索	11
3.1	双端队列 BFS	11
3.2	Astar 启发式搜索	12
第四章	动态规划	14
4.1	背包问题合集	14
4.2	树形依赖背包	15
4.3	最长单调子序列 (nlogn)	16
4.4	最长公共子序列	17
4.5	单调队列优化 DP	18
4.6	单调队列 DP	18
4.7	区间 DP	19
4.8	时间段 DP	19
4.9	数位 DP	19
4.10	数位 DP	21
4.11	二维上升子序列最大和 (nlogn)	21
第五章	数论	23
5.1	暴力判素数	23
5.2	反素数, 求满足因子数个数等于 n 的最小数	23
5.3	米勒罗宾素性检测	24
5.4	埃氏筛	27
5.5	欧拉筛	27
5.6	分解质因数	27
5.7	最大公约数	28
5.8	最小公倍数	28
5.9	扩展欧几里得	28
5.10	法雷分数 (辗转相除法)	29
5.11	中国剩余定理	29
5.12	扩展 CRT	30
5.13	欧拉函数	30
5.14	递归欧拉降幂	31
5.15	求逆元	32
5.16	快速乘法取模	33
5.17	快速幂取模	33
5.18	互质对数计数	34

5.19	原根	34
5.20	BSGS	35
5.21	离散对数问题 (高阶 BSGS)	36
5.22	二分分数树 (Stern-Brocot Tree)	39
5.23	二次剩余	40
5.24	计算莫比乌斯函数	41
5.25	杜教筛	42
5.26	Min25 筛	42
5.27	常用公式	44
5.28	数论公式	46
第六章	数学	48
6.1	$C(n, m) \bmod p$ (n 很大 p 可以很大)	48
6.2	Lucas 定理	48
6.3	计算从 $C(n, 0)$ 到 $C(n, p)$ 的值	48
6.4	求贝尔数 (模 p)	49
6.5	计算第一类斯特林数	50
6.6	自适应辛普森	51
6.7	博弈论	51
6.8	SG 函数	53
6.9	矩阵快速幂	53
6.10	高斯消元	54
6.11	异或线性基	56
6.12	java 大数开方	58
6.13	单纯形法	59
6.14	容斥	60
6.15	拉格朗日插值	61
6.16	多项式乘法/平方/取模	62
6.17	快速傅里叶变换	66
6.18	快速数论变换	67
6.19	快速沃尔什变换	67
6.20	分治 fft	68
6.21	polya 项链染色	69
6.22	染色多项式	69
6.23	错位排列递推公式	69
6.24	BBP 公式求 π 十六进制的第 k 位	69
第七章	图论	72
7.1	前向星	72
7.2	并查集	72
7.3	可撤销并查集 (按秩合并)	73
7.4	Kruskal 最小生成树	74
7.5	Prim 最小生成树	75
7.6	SPFA 最短路	76
7.7	dijkstra 最短路	76
7.8	Floyd 任意两点间最短路	77
7.9	2-SAT 问题	78
7.10	tarjan 强连通分量	79
7.11	Kosaraju 强连通分量	80
7.12	点双联通分量	81
7.13	边双联通分量	83
7.14	求桥	84
7.15	欧拉回路	85
7.16	k 短路	86
7.17	前 k 短路	88
7.18	全局 k 短路	91
7.19	k 小带权匹配	92
7.20	最小环	95
7.21	最小树形图	95
7.22	次小生成树 (Prim)	96
7.23	次小生成树 (Kruskal)	97
7.24	最小生成树计数	99
7.25	最小树形图计数	100

7.26	Dinic 最大流	102
7.27	ISAP 最大流	103
7.28	最小费用最大流	105
7.29	ZKW 费用流	106
7.30	上下界网络流	108
7.31	图匹配理论	115
7.32	二分图最大匹配匈牙利算法	117
7.33	二分图最大权匹配 KM 算法	117

第八章 数据结构

120

8.1	树状数组	120
8.2	二维树状数组	122
8.3	树状数组求逆序对	123
8.4	堆	124
8.5	单调栈和单调队列	125
8.6	RMQ	126
8.7	RMQ	126
8.8	线段树	127
8.9	ZKW 线段树	128
8.10	吉司机线段树	129
8.11	扫描线	132
8.12	固定大小矩形最大点覆盖	136
8.13	二维线段树 (单点更新区间最值)	137
8.14	二维线段树 (区间加值单点查询)	139
8.15	主席树	141
8.16	主席树动态 k 大	143
8.17	Treap 树	145
8.18	函数式 Treap	147
8.19	Splay 树	149
8.20	Splay 树	154
8.21	Splay 树	157
8.22	点分治	159
8.23	树上启发式合并	160
8.24	0-1trie 区间异或最大值	161
8.25	0-1trie 子树异或最大值	163
8.26	莫队算法	165
8.27	最近公共祖先 (在线)	166
8.28	最近公共祖先 (离线)	167
8.29	最近公共祖先	168
8.30	树链剖分	171

第九章 字符串

173

9.1	KMP	173
9.2	扩展 KMP	173
9.3	字符串的最小最大表示法	174
9.4	TRIE	175
9.5	AC 自动机	176
9.6	后缀数组 (倍增)	177
9.7	后缀数组 (sais)	179
9.8	后缀自动机	180
9.9	最长回文子串	181
9.10	回文树	182
9.11	字符串哈希算法	183
9.12	字符串哈希表	184

第十章 几何

186

10.1	平面几何公式	186
10.2	求凸包	187
10.3	四点共面	188
10.4	多边形重心	189
10.5	旋转卡壳	190
10.6	模拟退火	192
10.7	半平面交	193

10.8 计算几何	195
第十一章 类	205
11.1 点类	205
11.2 分数类	205
11.3 矩阵	206
11.4 01 矩阵	207
11.5 简单大数	208
11.6 大数	211
11.7 java 大数	215
第十二章 技巧与黑科技	216
12.1 离散化	216
12.2 快速枚举子集	216
12.3 bitset	217
12.4 位运算	217
12.5 跳舞链	218
12.6 K-D 树	219
12.7 随机	221
12.8 珂朵莉树 (Old Driver Tree)	221
12.9 CDQ 分治	222
12.10 内置位运算函数	223
12.11 0-1 分数规划	224
12.12 BM 线性递推	225

第一章 配置

1.1 VIM 配置

```
1 syntax on
2 set nu
3 set cindent
4 set tabstop=4
5 set shiftwidth=4
6 set noswapfile
7 set mouse=a
8
9 map <C-A> ggVG"+y
10
11 func Close(char)
12     if getline(".")[col('.')-1]==a:char
13         return "\<Right>"
14     else return a:char
15     endif
16 endfunc
17
18 inoremap { {}<Esc>i
19 inoremap } <c-r>=Close('}')<CR>
20 inoremap ( ()<Esc>i
21 inoremap ) <c-r>=Close('}')<CR>
22 inoremap [ []<Esc>i
23 inoremap ] <c-r>=Close(']')<CR>
24
25
26 map <F9> :call Run()<CR>
27 func! Run()
28     exec "w"
29     exec "!g++ -std=c++11 -O2 % -o %<"
30     exec "!time ./%<"
31 endfunc
```

1.2 基础头文件

```
1 #include<iostream>
2 #include<algorithm>
3 #include<cstdio>
4 #include<cstring>
5 using namespace std;
```

```
6  #define int long long
7  #define maxn 200000
8  #define INF 0x3f3f3f3f
9  const int mod=1e9+7;
10 signed main(){
11     ios::sync_with_stdio(false);
12     cin.tie(0);
13
14     return 0;
15 }
```

第二章 输入输出

2.1 取消同步

```
1 std::ios::sync_with_stdio(false);
2 std::cin.tie(0);
```

2.2 浮点数输出格式

```
1 //include <iomanip>
2
3 std::cout << std::fixed << std::setprecision(12) << ans << std::endl;
```

2.3 整型快速输入

```
1 // 整型
2 // 若读入不成功, 返回false
3 // ios::sync_with_stdio(true)
4 // #include <cctype>
5 bool quick_in(int &x) {
6     char c;
7     while((c = getchar()) != EOF && !isdigit(c));
8     if(c == EOF) {
9         return false;
10    }
11    x = 0;
12    do {
13        x *= 10;
14        x += c - '0';
15    } while((c = getchar()) != EOF && isdigit(c));
16    return true;
17 }
18
19 // 带符号整型
20 // 直接=返回值
21 // #include <cctype>
22 int read() {
23     int x = 0, l = 1; char ch = getchar();
24     while (!isdigit(ch)) {if (ch=='-') l=-1; ch=getchar();}
```



```
25     while (isdigit(ch)) x=x*10+(ch^48),ch=getchar();
26     return x*1;
27 }
28
29 template <class T>
30 inline bool Read(T &ret) {
31     char c; int sgn;
32     if(c=getchar(),c==EOF) return 0; //EOF
33     while(c!='-'&&(c<'0' || c>'9')) c=getchar();
34     sgn=(c=='-') ?-1:1 ;
35     ret=(c=='-') ?0:(c-'0');
36     while(c=getchar(),c>='0'&&c<='9')
37         ret=ret*10+(c-'0');
38     ret*=sgn;
39     return 1;
40 }
```

2.4 字符串快速输入

```
1 bool quick_in(char *p) {
2     char c;
3     while((c = getchar()) != EOF && (c == '\0' || c == '\n'));
4     if(c == EOF) {
5         return false;
6     }
7     do {
8         *p++ = c;
9     } while((c=getchar()) != EOF && c != '\0' && c != '\n');
10    *p = 0;
11    return true;
12 }
```

2.5 整型快速输出

```
1 void quick_out(int x) {
2     char str[13];
3     if(x) {
4         int i;
5         for(i = 0; x; ++i) {
6             str[i] = x % 10 + '0';
7             x /= 10;
8         }
9         while(i--) {
10            putchar(str[i]);
11        }
12    } else {
13        putchar('0');
14    }
15 }
```

2.6 字符串快速输出

```
1 void quick_out(char *p) {
2     while(*p) {
3         putchar(*p++);
4     }
5 }
```

2.7 python 输入

```
1 a, b, c =map(int,input().split(' '))
```

2.8 java 输入

```
1 Scanner cin=new Scanner(System.in);// 读入
```

2.9 int128 输入输出

```
1 std::ostream& operator<<(std::ostream& os, __int128 T) {
2     if (T<0) os<<"-";if (T>=10 ) os<<T/10;if (T<=-10) os<<(-(T/10));
3     return os<<( (int) (T%10) >0 ? (int) (T%10) : -(int) (T%10) ) ;
4 }
5
6 void scan(__int128 &x) {
7     x = 0;
8     int f = 1;
9     char ch;
10    if((ch = getchar()) == '-') f = -f;
11    else x = x*10 + ch-'0';
12    while((ch = getchar()) >= '0' && ch <= '9')
13        x = x*10 + ch-'0';
14    x *= f;
15 }
16
17 void print(__int128 x) {
18     if(x < 0) {
19         x = -x;
20         putchar('-');
21     }
22     if(x > 9) print(x/10);
23     putchar(x%10 + '0');
24 }
25 // by Simon
26 struct Istream {
```

```
27     template <class T>
28     Istream &operator >>(T &x) {
29         static char ch;static bool neg;
30         for(ch=neg=0;ch<'0' || '9'<ch;neg|=ch=='-',ch=getchar());
31         for(x=0;'0'<=ch && ch<='9';(x*=10)+=ch-'0',ch=getchar());
32         x=neg?-x:x;
33         return *this;
34     }
35 }fin;
36
37 struct Ostream {
38     template <class T>
39     Ostream &operator <<(T x) {
40         x<0 && (putchar('-'),x=-x);
41         static char stack[233];static int top;
42         for(top=0;x;stack[++top]=x%10+'0',x/=10);
43         for(top==0 && (stack[top=1]='0');top;putchar(stack[top--]));
44         return *this;
45     }
46
47     Ostream &operator <<(char ch) {
48         putchar(ch);
49         return *this;
50     }
51 }fout;
```

第三章 搜索

3.1 双端队列 BFS

```
1  /*
2   * Author: Simon
3   * 功能：双端队列BFS
4   * 适用范围：边权值为可能有，也可能没有（由于 BFS 适用于权值为 1 的图，所以一般是 0 or 1）
5   *           或者能够转化为这种边权值的最短路问题。例一束光线从(n,m)点向左射出，遇到#可以选择方向不
6   *           变
7   *           或者向上下左右反射光线，问最少选择反射几次能从第一行向左射出
8   */
9  char a[maxn][maxn];
10 struct node{
11     int x,y,step,d;
12 };
13 int dir[4][2]={1,0,-1,0,0,1,0,-1};
14 bool vis[maxn][maxn][4];
15 bool bfs(int n,int m){
16     deque<node>q;
17     q.push_front({n,m,0,3});
18     while(!q.empty()){
19         node now=q.front(),next;q.pop_front();
20         if(now.x==1&&now.d==3){cout<<now.step<<endl;return 1;}
21         if(vis[now.x][now.y][now.d]) continue;
22         vis[now.x][now.y][now.d]=1;
23         next=now;
24         next.x+=dir[now.d][0],next.y+=dir[now.d][1];
25         if(next.x>=1&&next.x<=n&&next.y>=1&&next.y<=m) q.push_front(next);
26         if(a[now.x][now.y]=='#'){
27             for(int i=0;i<4;i++){
28                 next=now;
29                 next.x+=dir[i][0];
30                 next.y+=dir[i][1];
31                 next.step++;next.d=i;
32                 if(next.x<1||next.x>n||next.y<1||next.y>m||vis[next.x][next.y][i]) continue;
33                 q.push_back(next);
34             }
35         }
36     }
37     return 0;
38 }
```

3.2 Astar 启发式搜索

```
1  /*
2   * Author: Simon
3   * 功能: A*启发式搜索 (例: 八数码问题)
4   */
5  int Hash[9]={1,1,2,6,24,120,720,5040,40320};
6  int dir[4][2]={-1,0,1,0,0,-1,0,1};
7  char d[5]="udlr";
8  int vis[maxn];
9  struct node{
10     int f[3][3];
11     int g,h,hashval,x,y;
12     bool operator <(const node a) const{
13         return a.g+a.h<g+h;
14     }
15 };
16 struct path{
17     int pre;
18     char ch;
19 }p[maxn];
20 int get_h(int f[][3]){
21     int ans=0;
22     for(int i=0;i<3;i++){
23         for(int j=0;j<3;j++){
24             if(f[i][j]){
25                 ans+=abs(i-(f[i][j]-1)/3)+abs(j-(f[i][j]-1)%3);
26             }
27         }
28     }
29     return ans;
30 }
31 bool checkedge(node next){
32     if(next.x>=0&&next.y>=0&&next.x<3&&next.y<3) return 1;
33     return 0;
34 }
35 void As_bfs(node e){
36     priority_queue<node>q;
37     node now,next;
38     for(int i=0;i<9;i++) now.f[i/3][i%3]=(i+1)%9;
39     int end_ans=get_hash(now);
40     e.h=get_h(e);e.g=0;
41     e.hashval=get_hash(e);
42     p[e.hashval].pre=-1;
43     q.push(e);
44     while(!q.empty()){
45         now=q.top(); q.pop();
46         if (now.hashval == end_ans) {
47             print(now.hashval);
48             cout << endl;
49             return;
50         }
51         if(vis[now.hashval]) continue;vis[now.hashval]=1;
52         for(int i=0;i<4;i++){
```

```
53         next=now;
54         next.x=now.x+dir[i][0];
55         next.y=now.y+dir[i][1];
56         if(checkedge(next)){
57             swap(next.f[now.x][now.y], next.f[next.x][next.y]);
58             next.hashval = get_hash(next);
59             if(vis[next.hashval]) continue;
60             next.g++; next.h = get_h(next);
61             p[next.hashval].pre=now.hashval;
62             p[next.hashval].ch=d[i];
63             q.push(next);
64         }
65     }
66 }
67 }
```

第四章 动态规划

4.1 背包问题合集

```
1  /*
2   * Author: Simon
3   * 复杂度:  $O(N \cdot V), O(N \cdot V), O(V \cdot \text{Sigma}(\log(C_i)))$ 
4   * 说明: 利用单调队列还可以优化时间复杂度
5   */
6  /*01背包 */
7  void ZeroOnepark(int V/*背包容量*/, int val/*物品价值*/, int vol/*物品体积*/) {
8      for (int j = V; j >= vol; j--) {
9          dp[j] = max(dp[j], dp[j - vol] + val);
10     }
11 }
12
13 /*完全背包 */
14 void Completemark(int V/*背包容量*/, int val/*物品价值*/, int vol/*物品体积*/) {
15     for (int j = vol; j <= V; j++) {
16         dp[j] = max(dp[j], dp[j - vol] + val);
17     }
18 }
19
20 /*多重背包 */
21 void Multiplepark(int val/*物品价值*/, int vol/*物品体积*/, int amount/*物品数量*/) {
22     if (vol * amount >= v) {
23         Completemark(val, vol);
24     } else {
25         int k = 1;
26         while (k < amount) {
27             ZeroOnepark(k * val, k * vol);
28             amount -= k; k <= 1;
29         }
30         if (amount > 0) {
31             ZeroOnepark(amount * val, amount * vol);
32         }
33     }
34 }
35 int main(){
36     memset(dp, 0, sizeof(dp));
37     for (int i = 1; i <= n; i++) {
38         Multiplepark(value[i], volume[i], num[i]);
39     }
40 }
41
42 /*
```

```

43  * Author: Simon
44  * 功能：单调队列优化多重背包
45  * 复杂度： $O(N \cdot V)$ 
46  */
47  int solve(int volume[], int value[], int n, int V){
48      pair<int, int> q[maxn]; int head=1, tail=0;
49      for (int i = 1; i <= n; i++) {
50          for (int j = 0; j < volume[i]; j++) {
51              q[head = tail = 1] = make_pair(dp[j], 0);
52              for (int k = j + volume[i]; k <= V; k += volume[i]) {
53                  int a = k / volume[i], t = dp[k] - a * value[i];
54                  while (head <= tail && q[tail].first <= t) tail--;
55                  q[++tail] = make_pair(t, a);
56                  while (head <= tail && q[head].second + num[i] < a) head++;
57                  dp[k] = max(dp[k], q[head].first + a * value[i]);
58              }
59          }
60      }
61  }

```

4.2 树形依赖背包

```

1  /*
2  * Author: Simon
3  * 功能：树形依赖背包问题
4  * 定义 $dp[u][i]$ 表示，以 $u$ 为根节点的子树中保留 $i$ 条树枝所获得的最大权值
5  * 则转移方程为 $dp[u][i] = \max(dp[u][i], dp[left[u]][i-j-1] + left[u].w + dp[right[u]][j-1] + right[u].w)$ 
6  * 表示 $u$ 的右儿子保留 $j-1$ 条边， $u$ 的左儿子保留剩下的 $i-j-1$ 条边，此时总共有 $i-2$ 条边，还要加上 $u-left[u]$ 
7  * 另外两种转移状态 $dp[u][i] = \max(dp[u][i], dp[u][i-j] + dp[v][j-1] + w)$ 
8  * 跟上面类似，只不过将 $u$ 与其中一个儿子节点的状态放在一起。此时需要倒序枚举 $i$ 来保证只选择一次
9  * 没有访问过的子树不会保存在 $dp[u][i]$ 中，所以不会出现重复计算的情况。
10 */
11 void dfs(int u, int p=-1){
12     sz[u]=1;
13     for(auto t:g[u]){
14         int &v=t.first, &w=t.second;
15         if(v==p) continue;
16         dfs(v, u); sz[u]+=sz[v];
17         for(int i=min(q, sz[u]); i>=1; i--){
18             for(int j=1; j<=min(sz[v], i); j++){
19                 dp[u][i]=max(dp[u][i], dp[u][i-j]+dp[v][j-1]+w);
20             }
21         }
22     }
23 }

```


4.3 最长单调子序列 (nlogn)

```

1  int arr[maxn], n;
2
3  template<class Cmp>
4  int LIS (Cmp cmp) {
5      static int m, end[maxn];
6      m = 0;
7      for (int i=0; i<n; i++) {
8          int pos = lower_bound(end, end+m, arr[i], cmp)-end;
9          end[pos] = arr[i], m += pos==m;
10     }
11     return m;
12 }
13
14 bool greater1(int value) {
15     return value >=1;
16 }
17
18 /*****
19     std::cout << LIS(std::less<int>()) << std::endl;           //严格上升
20     std::cout << LIS(std::less_equal<int>()) << std::endl;     //非严格上升
21     std::cout << LIS(std::greater<int>()) << std::endl;        //严格下降
22     std::cout << LIS(std::greater_equal<int>()) << std::endl;  //非严格下降
23     std::cout << count_if(a,a+7,std::greater1) << std::endl;  //计数
24 *****/
25 /*
26  * Author: Simon
27  * 功能：求最长不下降子序列的长度
28  * 复杂度：O(nLog(n))
29  */
30 int solve(int n,int a[]){
31     int dp[maxn]; memset(dp, INF, sizeof(dp));
32     for(int i=1;i<=n;i++){
33         *upper_bound(dp+1,dp+n+1,a[i])=a[i];
34     }
35     int ans=lower_bound(dp+1,dp+n+1,INF)-dp;
36     return ans;
37 }
38 /*
39  * Author: Simon
40  * 功能：求最长上升子序列
41  * 复杂度：O(nLog(n))
42  */
43 vector<int> solve(int n,int p[]){
44     vector<int> f;f.push_back(0);
45     int a[n+5]; /*记录每个数的前一个数是多少 */
46     for(int i=1;i<=n;i++){
47         auto it=lower_bound(f.begin(),f.end(),p[i]);
48         if(it==f.end()){
49             a[p[i]]=f.back();
50             f.push_back(p[i]);
51         }
52         else{

```

```

53         *it=p[i];it--;
54         a[p[i]]=*it;
55     }
56 }
57 vector<int>vis; /*最长上升子序列 */
58 for(int t=f.back();t;t=a[t]) vis.push_back(t);
59 reverse(vis.begin(),vis.end());
60 return vis;
61 }
62 /*
63  * Author: Simon
64  */
65 int a[maxn];/*原始数组 */
66 memset(f,INF,sizeof(int)*(n+2));
67 for(int i=1;i<=n;i++){ //nLog(n)求最长上升子序列
68     int j = lower_bound(f + 1, f + n + 1, a[i]) - f;
69     End[i] = j; //以i为终点的最长严格上升子序列长度为j
70     f[j] = a[i];
71 }
72 memset(g,0,sizeof(int)*(n+2));
73 for (int i = n; i >= 1; i--) {
74     int j=lower_bound(g+1,g+n+1,-a[i])-g;
75     Start1[i]=j; //以i为起点的最长严格上升子序列的长度为j
76     g[j]=-a[i];
77 }
78 memset(g,INF,sizeof(int)*(n+2));
79 reverse(a+1,a+n+1);
80 for(int i=1;i<=n;i++){ //nLog(n)求最长上升子序列
81     int j = lower_bound(g + 1, g + n + 1, a[i]) - g;
82     Start[n-i+1] = j; //以i为起点的最长严格下降子序列长度为j
83     g[j] = a[i];
84 }
85 memset(f,0,sizeof(int)*(n+2));
86 for (int i = n; i >= 1; i--) {
87     int j = lower_bound(f + 1, f + n + 1, -a[i]) - f;
88     End1[n-i+1] = j; //以i为终点的最长严格下降子序列的长度为j
89     f[j] = -a[i];
90 }

```

4.4 最长公共子序列

```

1  int dp[maxn][maxn];
2
3  void LCS(int n1, int n2, int A[], int B[]) {
4      for(int i=1; i<=n1; i++) {
5          for(int j=1; j<=n2; j++) {
6              dp[i][j] = dp[i-1][j];
7              if (dp[i][j-1] > dp[i][j]) {
8                  dp[i][j] = dp[i][j-1];
9              }
10             if (A[i] == B[j] && dp[i-1][j-1] + 1 > dp[i][j]) {
11                 dp[i][j] = dp[i-1][j-1] + 1;

```

```

12     }
13     }
14 }
15 }

```

4.5 单调队列优化 DP

```

1 //单调队列求区间最小值
2 int a[maxn], q[maxn], num[maxn] = {0};
3 int Fmin[maxn];
4 int k, n, head, tail;
5
6 void DPmin() {
7     head = 1, tail = 0;
8     for (int i = 1; i <= n; i++) {
9         while (num[head] < i-k+1 && head <= tail) head++;
10        while (a[i] <= q[tail] /*区间最大值此处改为>=*/ && head <= tail) tail--;
11        num[++tail] = i;
12        q[tail] = a[i];
13        Fmin[i] = q[head];
14    }
15 }

```

4.6 单调队列 DP

```

1 int head,tail;
2 struct node
3 {
4     int val,k;
5 }q[maxm*2];
6 void right(int i,int j) {
7     while(head<tail&&dp[i-1][j]-sum[i][j-1]>=q[tail-1].val) tail--;//dp[i-1][k]-sum[i][k]
8     q[tail].val=dp[i-1][j]-sum[i][j-1];
9     q[tail++].k=j;
10    while(head==-1|| (head<tail&&j-q[head].k>t)) head++;//保证步数不大于限制条件
11    dp[i][j]=max(dp[i][j],q[head].val+sum[i][j]);//dp[i][j]=max(dp[i][j],dp[i-1][k]-sum[i][k]-1+sum[i][j])
12 }
13 void left(int i,int j) {
14     while(head<tail&&dp[i-1][j]+sum[i][j]>=q[tail-1].val) tail--;//dp[i-1][k]+sum[i][k];
15     q[tail].val=dp[i-1][j]+sum[i][j];
16     q[tail++].k=j;
17     while(head==-1|| (head<tail&&q[head].k-j>t)) head++;//保证步数不大于限制条件
18     dp[i][j]=max(dp[i][j],q[head].val-sum[i][j-1]);//dp[i][j]=max(dp[i][j],dp[i-1][j]+sum[i][k]-sum[i][j-1])
19 }
20 void solve() {
21     for(int i=1;i<=n;i++) for(int j=1;j<=m;j++) dp[i][j]=-INF;
22     for(int i=1;i<=m;i++){//第一行初始化

```

```

23     if(abs(x-i)>t) continue;
24     if(i<=x) dp[1][i]=sum[1][x]-sum[1][i-1];
25     else dp[1][i]=sum[1][i]-sum[1][x-1];
26 }
27 for(int i=2;i<=n;i++) {
28     head=tail=0;
29     for(int j=m;j>=1;j--) left(i,j);
30     head=tail=0;
31     for(int j=1;j<=m;j++) right(i,j);
32 }
33 }

```

4.7 区间 DP

```

1  for (int x = 0; x < n; x++){//枚举长度
2      for (int i = 1; i + x <= n; i++){//枚举起点
3          dp[i][i] = 1;
4          int j = x + i;//终点
5          dp[i][j] = dp[i + 1][j] + 1;
6          for (int k = i + 1; k <= j; k++) {
7              if (a[i] == a[k])
8                  dp[i][j] = min(dp[i][j], dp[i][k - 1] + dp[k + 1][j]);
9          }
10     }
11 }

```

4.8 时间段 DP

```

1  int t=maxn-1;
2  memset(dp,-1,sizeof(dp));dp[0][0]=0;
3  for(int i=0;i<t;i++){//枚举每一个起点
4      for(int j=0;j<(1<n);j++) dp[i+1][j]=max(dp[i+1][j],dp[i][j]); //更新所有状态的值
5      for(int j=0;j<g[i].size();j++){//枚举当前起点的所有终点
6          for(int k=0;k<(1<n);k++){
7              if(dp[i][k]!=-1){
8                  dp[g[i][j].fin][k|(1<g[i][j].num)]=max(dp[g[i][j].fin][k|(1<g[i][j].num)],dp
9                      [i][k]+g[i][j].val);
10                 //枚举所有结束时间，结束时间由起始时间转移过来
11             }
12         }
13     }

```

4.9 数位 DP

```

1  typedef long long ll;
2  int a[20];

```

```

3 11 dp[20][state]; //不同题目状态不同
4 11 dfs(int pos, /*state变量*/, bool lead /*前导零*/, bool limit /*数位上界变量*/) //不是每个题都要判
   断前导零
5 {
6     //递归边界, 既然是按位枚举, 最低位是0, 那么pos==-1说明这个数我枚举完了
7     if(pos==-1) return 1; /*这里一般返回1, 表示你枚举的这个数是合法的, 那么这里就需要你在枚举时
   必须每一位都要满足题目条件, 也就是说当前枚举到pos位, 一定要保证前面已经枚举的数位是合
   法的。不过具体题目不同或者写法不同的话不一定要返回1 */
8     //第二个就是记忆化(在此前可能不同题目还能有一些剪枝)
9     if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];
10    /*常规写法都是在没有限制的条件记忆化, 这里与下面记录状态是对应, 具体为什么是有条件的记忆化
   后面会讲*/
11    int up=limit?a[pos]:9; //根据limit判断枚举的上界up; 这个的例子前面用213讲过了
12    ll ans=0;
13    //开始计数
14    for(int i=0; i<=up; i++) //枚举, 然后把不同情况的个数加到ans就可以了
15    {
16        if() ...
17        else if() ...
18        ans+=dfs(pos-1, /*状态转移*/, lead && i==0, limit && i==a[pos]) //最后两个变量传参都是这
   样写的
19        /*这里还算比较灵活, 不过做几个题就觉得这里也是套路了
20        大概就是说, 我当前数位枚举的数是i, 然后根据题目的约束条件分类讨论
21        去计算不同情况下的个数, 还有要根据state变量来保证i的合法性, 比如题目
22        要求数位上不能有62连续出现, 那么就是state就是要保存前一位pre, 然后分类,
23        前一位如果是6那么这意味就不能是2, 这里一定要保存枚举的这个数是合法*/
24    }
25    //计算完, 记录状态
26    if(!limit && !lead) dp[pos][state]=ans;
27    /*这里对应上面的记忆化, 在一定条件下时记录, 保证一致性, 当然如果约束条件不需要考虑Lead, 这
   里就是Lead就完全不用考虑了*/
28    return ans;
29 }
30
31 11 solve(ll x)
32 {
33     int pos=0;
34     while(x) //把数位都分解出来
35     {
36         a[pos++]=x%10; //个人老是喜欢编号为[0,pos), 看不惯的就按自己习惯来, 反正注意数位边界就行
37         x/=10;
38     }
39     return dfs(pos-1 /*从最高位开始枚举*/, /*一系列状态 */, true, true); //刚开始最高位都是有限制并
   且有前导零的, 显然比最高位还要高的一位视为0嘛
40 }
41
42 int main()
43 {
44     ll le, ri;
45     while(~scanf("%lld%lld", &le, &ri))
46     {
47         //初始化dp数组为-1, 这里还有更加优美的优化, 后面讲
48         printf("%lld\n", solve(ri)-solve(le-1));
49     }

```

50 }

4.10 数位 DP

```

1  int a[maxn], bit[maxn]; //a为分解整数数组, bit数组为10^(i-1)
2  pair<int, int> dp[maxn][2000]; //first=满足条件的数个数, second=满足条件的数的和
3  bool vis[maxn][2000];
4  pair<int, int> dfs(int pos, int sta, int num, bool lead, bool limit){ //求满足条件的所有数的和
5      if(pos==0) return make_pair(1, 0); //计数
6      if(!limit && !lead && vis[pos][sta]) return dp[pos][sta];
7      if(!limit && !lead) vis[pos][sta]=1;
8      int up=limit?a[pos]:9, t, tt; pair<int, int> tmp, ans;
9      for(int i=0; i<=up; i++){
10         if(num>=k && !(sta & (1<<i))) continue; //不满足条件, 跳出
11         if(lead && !i) t=0, tt=0;
12         else t=(sta|(1<<i)), tt=(sta & (1<<i))?num:num+1;
13         tmp=dfs(pos-1, t, tt, lead && !i, limit && i==up);
14         ans.first+=tmp.first; ans.first%=mod; //满足条件的数的个数
15         ans.second+=tmp.first*bit[pos]%mod*i%mod+tmp.second; ans.second%=mod; //满足条件的数的和
16         //10+11+...+19=9*(10*1)+45
17     }
18     if(!limit && !lead) dp[pos][sta]=ans;
19     return ans;
20 }

```

4.11 二维上升子序列最大和 (nlogn)

```

1  //二维带权最大上升子序列
2  //written by Simon
3
4  #include<bits/stdc++.h>
5  using namespace std;
6  typedef int Int;
7  #define int long long
8  #define INF 0x3f3f3f3f
9  #define maxn 100005
10 int bin[maxn]; //用于二分
11 struct cmp_key{ //map排序辅助结构体, 第一维升序, 第二维降序
12     bool operator()(const pair<int, int>&a, const pair<int, int>&b){
13         if(a.first==b.first){
14             return a.second>b.second;
15         }
16         else return a.first<b.first;
17     }
18 };
19 map<pair<int, int>, int, cmp_key>p;
20 int binary_search(int v, int n){
21     int left=0, right=n, mid;
22     while(left<=right){

```

```
23     mid=(left+right)>>1;
24     if(bin[mid]<v){
25         left=mid+1;
26     }
27     else right=mid-1;
28 }
29 return left;
30 }
31 int tree[maxn];
32 int lowbit(int x){
33     return (x&(-x));
34 }
35 void update(int x,int v,int len){
36     for(int i=x;i<=len;i+=lowbit(i)){
37         tree[i]=max(tree[i],v);
38     }
39 }
40 int get(int x,int len){
41     int sum=0;
42     for(int i=x;i>=1;i-=lowbit(i)){
43         sum=max(sum,tree[i]);
44     }
45     return sum;
46 }
47 Int main() {
48     ios::sync_with_stdio(false);
49     cin.tie(0);
50     int n;
51     scanf("%lld",&n);
52     for(int i=1;i<=n;i++){
53         int x,y,v;
54         scanf("%lld%lld%lld",&x,&y,&v);
55         p[make_pair(x,y)]+=v;//去重 (关键)
56         bin[i]=y;
57     }
58     sort(bin+1,bin+n+1);
59     auto len=unique(bin+1,bin+n+1)-(bin+1);//去重
60     for(auto v:p){
61         int x=binary_search(v.first.second,len);
62         int val=get(x-1,len);
63         update(x,val+v.second,len);
64     }
65     printf("%lld\n",get(len,len));
66     cin.get(),cin.get();
67     return 0;
68 }
```

第五章 数论

5.1 暴力判素数

```
1 bool is_prime(int u) {
2     if(u == 0 || u == 1) return false;
3     if(u == 2) return true;
4     if(u%2 == 0) return false;
5     for(int i=3; i <= sqrt(u) ; i+=2)
6         if(u%i==0) return false;
7     return true;
8 }
```

5.2 反素数，求满足因子数个数等于 n 的最小数

```
1  /*
2  Author: Simon
3  其实顾名思义，素数就是因子只有两个的数，那么反素数，就是因子最多的数（并且因子个数相同的时候值
   最小），所以反素数是相对于一个集合来说的。
4  反素数的特点。
5  1. 反素数肯定是从2开始的连续素数的幂次形式的乘积。
6  2. 数值小的素数的幂次大于等于数值大的素数的幂次，即
7      $e_1 \geq e_2 \geq e_3 \geq \dots \geq e_k$ 
8  */
9  int a[16]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53}
10 //给定因子数 $n$ ，求满足因子数个数等于 $n$ 的最小数。
11 void dfs(int dep/*第dep个素数*/,int sum/*当前的数*/,int num/*因子个数*/,int up/*上个素数的幂*/
12 ){
13     if(dep>=16||num>n) return ;
14     if(num==n){
15         ans=min(ans,sum);
16         return ;
17     }
18     for(int i=1;i<=up;i++){ //由性质2可知，数值大的素数的幂次小于等于数值小的素数的幂次
19         if(sum*fpow(a[dep],i)>ans) break;
20         dfs(dep+1,sum*fpow(a[dep],i),num*(i+1),i);
21     }
22 }
23 //求小于等于 $n$ 的因子数最多的数。与上面做法相同，只是dfs结束条件改一下。
```


5.3 米勒罗宾素性检测

```
1  #include<stdio.h>
2  #include<string.h>
3  #include<stdlib.h>
4  #include<time.h>
5  #include<iostream>
6  #include<string.h>
7  #include<math.h>
8  #include<algorithm>
9  using namespace std;
10
11  //*****
12  // Miller_Rabin 算法进行素数测试
13  //速度快, 而且可以判断 <2^63的数
14  //*****
15  const int S=20;//随机算法判定次数, S越大, 判错概率越小
16
17
18  //计算 (a*b)%c.  a,b都是Long Long的数, 直接相乘可能溢出的
19  // a,b,c <2^63
20  long long mult_mod(long long a,long long b,long long c)
21  {
22      a%=c;
23      b%=c;
24      long long ret=0;
25      while(b)
26      {
27          if(b&1){ret+=a;ret%=c;}
28          a<<=1;//别手残, 这里是a<<=1, 不是快速幂的a=a*a;
29          if(a>=c)a%=c;
30          b>>=1;
31      }
32      return ret;
33  }
34
35
36
37  //计算 x^n %c
38  long long pow_mod(long long x,long long n,long long mod)//x^n%c
39  {
40      if(n==1)return x%mod;
41      x%=mod;
42      long long tmp=x;
43      long long ret=1;
44      while(n)
45      {
46          if(n&1) ret=mult_mod(ret,tmp,mod);
47          tmp=mult_mod(tmp,tmp,mod);
48          n>>=1;
49      }
50      return ret;
51  }
52
```

```
53
54
55
56
57 //以a为基,  $n-1=x*2^t$   $a^{(n-1)}=1(mod\ n)$  验证n是不是合数
58 //一定是合数返回true, 不一定返回false
59 bool check(long long a, long long n, long long x, long long t)
60 {
61     long long ret=pow_mod(a,x,n);
62     long long last=ret;
63     for(int i=1;i<=t;i++)
64     {
65         ret=mult_mod(ret,ret,n);
66         if(ret==1&&last!=1&&last!=n-1) return true;//合数
67         last=ret;
68     }
69     if(ret!=1) return true;
70     return false;
71 }
72
73 // Miller_Rabin()算法素数判定
74 //是素数返回true.(可能是伪素数, 但概率极小)
75 //合数返回false;
76
77 bool Miller_Rabin(long long n)
78 {
79     if(n<2)return false;
80     if(n==2)return true;
81     if((n&1)==0) return false;//偶数
82     long long x=n-1;
83     long long t=0;
84     while((x&1)==0){x>>=1;t++;}
85     for(int i=0;i<S;i++)
86     {
87         long long a=rand()%(n-1)+1;//rand()需要stdlib.h头文件
88         if(check(a,n,x,t))
89             return false;//合数
90     }
91     return true;
92 }
93
94
95 //*****
96 //pollard_rho 算法进行质因数分解
97 //*****
98 long long factor[100];//质因数分解结果(刚返回时是无序的)
99 int tol;//质因数的个数。数组小标从0开始
100
101 long long gcd(long long a, long long b)
102 {
103     if(a==0)return 1;//???????
104     if(a<0) return gcd(-a,b);
105     while(b)
106     {
```

```
107     long long t=a%b;
108     a=b;
109     b=t;
110 }
111 return a;
112 }
113
114 long long Pollard_rho(long long x,long long c)
115 {
116     long long i=1,k=2;
117     long long x0=rand()%x;
118     long long y=x0;
119     while(1)
120     {
121         i++;
122         x0=(mult_mod(x0,x0,x)+c)%x;
123         long long d=gcd(y-x0,x);
124         if(d!=1&&d!=x) return d;
125         if(y==x0) return x;
126         if(i==k){y=x0;k+=k;}
127     }
128 }
129 //对n进行素因子分解
130 void findfac(long long n)
131 {
132     if(Miller_Rabin(n))//素数
133     {
134         factor[tol++]=n;
135         return;
136     }
137     long long p=n;
138     while(p>=n)p=Pollard_rho(p,rand()%(n-1)+1);
139     findfac(p);
140     findfac(n/p);
141 }
142 int main()
143 {
144     // srand(time(NULL));//需要time.h头文件 //POJ上G++要去掉这句话
145     int T;
146     long long n;
147     scanf("%d",&T);
148     while(T--)
149     {
150         scanf("%I64d",&n);
151         if(Miller_Rabin(n))
152         {
153             printf("Prime\n");
154             continue;
155         }
156         tol=0;
157         findfac(n);
158         long long ans=factor[0];
159         for(int i=1;i<tol;i++)
160             if(factor[i]<ans)
```

```
161         ans=factor[i];
162         printf("%I64d\n",ans);
163     }
164     return 0;
165 }
```

5.4 埃氏筛

```
1 bool prime_or_not[maxn];
2 for (int i = 2; i <= sqrt(maxn); i++) {
3     if (!prime_or_not[i]) {
4         for (int j = i * i; j <= maxn; j = j+i) {
5             prime_or_not[j] = 1;
6         }
7     }
8 }
```

5.5 欧拉筛

```
1 int prime[maxn], cnt = 0;
2 bool vis[maxn] = {1, 1};
3 void Euler(int n) {
4     cnt = 0;
5     for (int i = 2; i <= n; i++) {
6         if (!vis[i]) prime[++cnt] = i;
7         for (int j = 1; j <= cnt && i * prime[j] <= n; j++) {
8             vis[i * prime[j]] = true;
9             if (i % prime[j] == 0) break;
10        }
11    }
12 }
```

5.6 分解质因数

```
1 int cnt[maxn]; // 存储质因子是什么
2 int num[maxn]; // 该质因子的个数
3 int tot = 0; // 质因子的数量
4 void factorization(int x) // 输入x, 返回cnt数组和num数组
5 {
6     for(int i=2;i*i<=x;i++)
7     {
8         if(x%i==0)
9         {
10             cnt[tot]=i;
11             num[tot]=0;
12             while(x%i==0)
13             {
```

```
14         x/=i;
15         num[tot]++;
16     }
17     tot++;
18 }
19 }
20 if(x!=1)
21 {
22     cnt[tot]=x;
23     num[tot]=1;
24     tot++;
25 }
26 }
```

5.7 最大公约数

```
1 ll gcd(ll a, ll b) {
2     ll t;
3     while(b != 0) {
4         t=a%b;
5         a=b;
6         b=t;
7     }
8     return a;
9 }
```

5.8 最小公倍数

```
1 ll lcm(ll a, ll b) {
2     return a * b / gcd(a, b);
3 }
```

5.9 扩展欧几里得

```
1 /*
2  * Author: Simon
3  * 复杂度:  $O(\log(\max(a,b)))$ 
4  * 功能: 求解  $a*x+b*y=c$ 
5  */
6
7 /*解  $a*x+b*y=gcd(a,b)$ */
8 int exgcd(int a, int b, int &x, int &y) {
9     if (b == 0) {
10         x = 1, y = 0;
11         return a;
12     }
13     int g = exgcd(b, a % b, y, x);
```

```

14     y -= a / b * x;
15     return g;
16 }
17 /*
18  * 解 $a*x+b*y=c$ 
19  * 假设有一对特解 $x=n, y=m$ 
20  * 则其通解为:  $x=n-b*t$  or  $y=m+a*t$ 
21  */
22 void solve(int n, int A, int B, int C) {
23     int x, y;
24     int g = exgcd(A, B, x, y);
25     if (C % g != 0) {
26         cout << "-1" << endl;
27         return;
28     }
29     A /= g, B /= g, C /= g;
30     x *= C % B;
31     x = (x % B + B) % B; y = (C - A * x) / B; /*求最小非负整数 $x$ , 则 $y=(c-a*x)/b$  */
32     // y *= C % A;
33     // y = (y % A + A) % A; x = (C - B * y) / A; /*求最小非负整数 $y$ , 则 $x=(c-b*y)/a$  */
34     /*
35      * 具体题目
36      */
37 }

```

5.10 法雷分数 (辗转相除法)

```

1  /*
2  /*
3  * Author: Simon
4  * 功能: 求满足 $p1/x1 < b/y < p2/x2$ , 最小分母(且分子也最小)的分数 $b/y$ 
5  * 复杂度:  $O(\log(n))$ 
6  */
7  void solve(int p1, int x1, int p2, int x2, int &b, int &y){
8      int l=p1/x1, r=(p2+x2-1)/x2;
9      if(l+1<r){/*若两分数之间有整数, 直接返回 */
10         b=l+1, y=1;
11         return ;
12     }
13     solve(x2, p2-l*x2, x1, p1-l*x1, y, b);/*否则同时减去 $l$ 后, 分子分母互换 */
14     b+=l*y; /*回溯, 加回减去的值 */
15 }

```

5.11 中国剩余定理

```

1  /*
2  * Author: Simon
3  * 复杂度:  $O(n)$ 
4  * 功能: 求解 $x \equiv a_i \pmod{m_i}$ 同余方程组

```

```

5  * 某些计数问题或数论问题出于加长代码、增加难度、或者是一些其他不可告人的原因 给出的模数：不是
    质数！
6  * 但是对其质因数分解会发现它没有平方因子，也就是该模数是由一些不重复的质数相乘得到。
7  * 那么我们可以分别对这些模数进行计算，最后用 CRT 合并答案。
8  */
9  int crt(int ai[], int mi[], int len) {
10     int ans = 0, lcm = 1;
11     for (int i = 0; i < len; i++) lcm *= mi[i];
12     for (int i = 0; i < len; i++) {
13         int Mi = lcm / mi[i];
14         int inv = fpow(Mi, mi[i] - 2, mi[i]);
15         int x = fmul(fmul(inv, Mi, lcm), ai[i], lcm); //若lcm大于1e9需要用快速乘fmul
16         ans = (ans + x) % lcm;
17     }
18     return ans;
19 }

```

5.12 扩展 CRT

```

1  /*
2  * Author:Simon
3  * 功能：模数可以不互质情况下，求解同余方程组。
4  * 若返回-1则无解，否则返回最小非负整数解x，通解为x+i*M
5  * 复杂度：O(nLog)
6  */
7  int excrt(int mi[],int ai[],int n){ //扩展中国剩余定理
8      int M=mi[1],ans=ai[1]; //x=ans+i*M, 得到一个通解
9      for(int i=2;i<=n;i++){
10         int a=M,b=mi[i],c=((ai[i]-ans)%b+b)%b/*将c化为正数*/,x,y; //与第二个方程组成不定方程
11         int gcd=exgcd(a,b,x,y); //通过扩展欧几里得解的一组特解 (p,q)
12         if(c%gcd!=0) return -1;
13         a/=gcd,b/=gcd;
14         (x*=(c/gcd)%b)%=b; x=(x+b)%b;
15         ans+=x*M; //则 新同余方程的解x=ans+p*M
16         M*=b; //所有模数的最小公倍数 (M*b)/gcd=M*(b/gcd)
17         ans%=M; //最小整数解
18     }
19     return (ans+M)%M;
20 }

```

5.13 欧拉函数

```

1  int Phi(int n) {
2      int m = sqrt(n + 0.5);
3      int ans = n;
4      for (int i = 2; i <= m; ++i) {
5          if (n % i == 0) {
6              ans = ans - ans / i;
7              while (n % i == 0) n /= i;

```

```

8     }
9     }
10    if (n > 1) ans = ans - ans / n;
11    return ans;
12 }
13 /*
14  * Author: Simon
15  * 复杂度:  $O(n)$ 
16  * 功能: 求  $[1, n]$  小于  $i$  且与  $i$  互质的数的个数  $\phi(i)$ 
17  */
18 int prime[maxn], phi[maxn], cnt = 0;
19 bool vis[maxn];
20 void Euler(int n) {
21     phi[1] = 1;
22     for (int i = 2; i <= n; i++) {
23         if (!vis[i]) {
24             prime[++cnt] = i;
25             phi[i] = i - 1;
26         }
27         for (int j = 1; j <= cnt && i * prime[j] <= n; j++) {
28             vis[i * prime[j]] = 1;
29             if (i % prime[j] == 0) {
30                 phi[i * prime[j]] = phi[i] * prime[j];
31                 break;
32             }
33             phi[i * prime[j]] = phi[i] * (prime[j] - 1);
34         }
35     }
36 }

```

5.14 递归欧拉降幂

```

1  /*
2  * Author: Simon
3  * 功能: 求  $a^a^a^a^a \dots^a \bmod m$ , 总共  $b$  个
4  * 复杂度:  $O(\log(m))$ 
5  */
6  #include <bits/stdc++.h>
7  using namespace std;
8  typedef int Int;
9  #define int long long
10 #define INF 0x3f3f3f3f
11 #define maxn 1000005
12 int Phi(int x) {
13     int ans = x;
14     for (int i = 2; i * i <= x; i++) {
15         if (x % i == 0) {
16             ans = ans / i * (i - 1);
17             while (x % i == 0) x /= i;
18         }
19     }
20     if (x != 1) ans = ans / x * (x - 1);

```



```

21     return ans;
22 }
23 int fpow(int a,int b,int mod){
24     a%=mod;int ans=1;
25     while(b){
26         if(b&1) (ans*=a)%=mod;
27         (a*=a)%=mod;
28         b>>=1;
29     }
30     return ans%mod;
31 }
32 int gcd(int a,int b){
33     return b==0?a:gcd(b,a%b);
34 }
35 /*判断 $a^b \bmod p$  中 $b$ 是否小于 $p$  */
36 bool check(int a,int b,int p){
37     int ans=1;
38     if(a>=p) return 0;
39     for(int i=1;i<=b;i++){
40         if(ans>=20) return 0; /* $p$ 最大 $1e6$ , 所以 $ans \geq 20$ 则肯定大于 $p$  */
41         ans=fpow(a,ans,1e18);
42         if(ans>=p) return 0;
43     }
44     return 1;
45 }
46 /*递归欧拉降幂 */
47 int f(int a,int b,int m){
48     if(m==1) return 0;
49     if(b<=1) return fpow(a,b,m);
50     int p=Phi(m);
51     int t=f(a,b-1,p); /* $f(a,b,m)=a^{\{f(a,b-1,p) \bmod \text{待定}\} \bmod m}$  */
52     int g=gcd(a,m);
53     if(g==1/* $gcd(a,m)=1$  */ || check(a,b-1,p)/* $f(a,b-1,INF)<p$  */) return fpow(a,t,m); //扩展欧拉
        定理
54     else/* $f(a,b-1,INF)>p$  */ return fpow(a,t+p,m);
55 }
56 Int main(){
57     ios::sync_with_stdio(false);
58     cin.tie(0);
59     int T;cin>>T;
60     while(T--){
61         int a,b,m;cin>>a>>b>>m;
62         /* $f(a,b,m)=a^a^a^{\dots^a} \bmod m$ , 总共 $b$ 个 $a$  */
63         cout<<f(a,b,m)%m<<endl;
64     }
65     return 0;
66 }

```

5.15 求逆元

```

1 LL Inv(LL a, LL n){
2     return PowMod(a, EulerPhi(n) - 1, n);

```

```

3 //return PowMod(a,n-2,n); //n为素数
4 }
5
6 int Inv(int a, int n) {
7     int d, x, y;
8     d = extended_euclid(a, n, x, y);
9     if(d == 1) return (x%n + n) % n;
10    else return -1; // no solution
11 }
12 // by Simon
13 int inv[maxn];
14 int Inv(int n,int p){ //线性求1~n的逆元
15     inv[1] = 1;
16     for (int i = 2; i <= n; ++i) inv[i] = (long long)(p - p / i) * inv[p % i] % p;
17 }
18 //by Simon
19 int a[maxn]/*n个数 */,s[maxn]/*前缀积 */,inv[maxn]/*第i个数的逆元 */;
20 int Inv(int n,int p){ //线性求任意n个数的逆元
21     s[0] = 1;
22     for (int i = 1; i <= n; ++i) s[i] = s[i - 1] * a[i] % p; //前缀积
23     sv[n] = fpow(s[n], p - 2); // 当然这里也可以用 exgcd 来求逆元,视个人喜好而定.
24     for (int i = n; i >= 1; --i) sv[i - 1] = sv[i] * a[i] % p; //前缀积的逆元
25     for (int i = 1; i <= n; ++i) inv[i] = sv[i] * s[i - 1] % p;
26 }

```

5.16 快速乘法取模

```

1 //by sevenkplus
2 #define ll long long
3 #define ld long double
4 ll mul(ll x,ll y,ll z){return (x*y-(ll)(x/(ld)z*y+1e-3)*z+z)%z;}
5
6 //by Lazer2001
7 inline long long mmul (long long a, long long b, const long long& Mod) {
8     long long lf = a * (b >> 25LL) % Mod * (1LL << 25) % Mod;
9     long long rg = a * ( b & ( ( 1LL << 25 ) - 1 ) ) % Mod ;
10    return (lf + rg) % Mod ;
11 }

```

5.17 快速幂取模

```

1 /*
2  * Author: Simon
3  * 复杂度:  $O(\log(n))$ 
4  * 功能: 快速幂
5  */
6 int fpow(int a,int b,const int &mod){
7     int ans=1;a%=mod;
8     while(b){

```

```

9      if(b&1) ans=(1LL*ans*a)%mod;
10     a=(1LL*a*a)%mod;
11     b>>=1;
12 }
13 return ans;
14 }

```

5.18 互质对数计数

```

1 //Written by Simon
2 //求r以内与n不互质的数的个数
3 int solve(int r) {
4     int sum=0;
5     for(int i=1;i<(1<<fac.size());i++) { //枚举质因数的每一种组合
6         int ans=1,num=0;
7         for(int j=0;j<fac.size();j++) { //求当前组和的积
8             if(i&(1<<j)) {
9                 ans *= fac[j];
10                num++;
11            }
12        }
13        if(num&1) sum+=r/ans; //如果当前组合个数为奇数个，加上r以内能被ans整除的数的个数
14        else sum-=r/ans; //否则减去r以内能被ans整除的数的个数
15    }
16    return sum;
17 }

```

5.19 原根

```

1 /*
2 Author: Simon
3 平均复杂度 $O(\log\log(p))$ 
4 若不存在原根则返回-1
5 对于所有素数 $p>2$ , 正整数 $e$ , 当前仅当 $n=1, 2, 4, p^e, 2p^e$ 有原根
6 若 $g$ 是 $p$ 的原根, 对于 $1\leq i<p$ ,  $g^i \bmod p$ , 互不相同, 即唯一。
7 */
8 int proot(int p){ //fac为 (p-1) 的所有质因子。
9     for(int a=2;a<p;a++){
10         bool flag=0;
11         for(int i=0;i<fac.size();i++){
12             int v=fac[i];
13             if(fpow(a,(p-1)/v,p)==1){ //如果存在 $d$ ,  $a^{\{p-1/d\}} \bmod p=1$ 则 $a$ 不是 $p$ 的原根。
14                 flag=1;break;
15             }
16         }
17         if(!flag) return a;
18     }
19     return -1;
20 }

```

5.20 BSGS

```
1 //Author: Simon
2 #include <algorithm>
3 #include <cmath>
4 #include <cstring>
5 using ll = long long;
6 const int maxn = 1000005;
7 const ll mod = 611977;
8
9 struct HashMap {
10     ll head[mod+5], key[maxn], value[maxn], nxt[maxn], tol;
11     inline void clear() {
12         tol=0;
13         memset(head,-1,sizeof(head));
14     }
15     HashMap() {
16         clear();
17     }
18     inline void insert(ll k,ll v) {
19         ll idx = k % mod;
20         for(ll i = head[idx]; ~i; i = nxt[i]) {
21             if(key[i] == k) {
22                 value[i] = std::min(value[i], v);
23                 return ;
24             }
25         }
26         key[tol] = k;
27         value[tol] = v;
28         nxt[tol] = head[idx];
29         head[idx] = tol++;
30     }
31     inline ll operator [] (const ll &k) const {
32         ll idx = k % mod;
33         for(ll i=head[idx]; ~i; i=nxt[i]) {
34             if(key[i]==k) return value[i];
35         }
36         return -1;
37     }
38 }mp;
39
40 inline ll fpow(ll a, ll b, ll mod) {
41     a %= mod;
42     ll ans = 1;
43     while (b) {
44         if(b&1) ans = ans * a % mod;
45         a = a * a % mod;
46         b >>= 1;
47     }
48     return ans;
49 }
```

```

50 inline ll exgcd(ll a,ll b,ll &x,ll &y) {
51     if (b==0) {
52         x=1, y=0;
53         return a;
54     }
55     ll ans = exgcd(b, a%b, y, x);
56     y -= a/b*x;
57     return ans;
58 }
59
60 inline ll Bsgs(ll a,ll b,ll mod) {
61     a %= mod, b %= mod; mp.clear();
62     if (b==1) return 0;
63     if (a == 0) return b == 0 ? 1 : -1;
64     ll m = ceil(sqrt(mod)), inv, y;
65     exgcd(fpow(a, m, mod), mod, inv, y);
66     inv = (inv % mod + mod) % mod;
67     mp.insert(1, 1);
68     for(ll i=1, e=1; i<m; i++) {
69         e = e * a % mod;
70         if(mp[e] == -1) mp.insert(e, i+1);
71     }
72     for(ll i = 0; i <= m; i++) {
73         if(mp[b] != -1) {
74             ll ans = mp[b]-1;
75             return ans + i * m;
76         }
77         b = b * inv % mod;
78     }
79     return -1;
80 }
81
82 inline ll gcd(ll a, ll b) {
83     return b==0 ? a : gcd(b, a%b);
84 }
85 inline int exBsgs(int a, int b, int p) {//扩展BSGS, 处理a, mod不互质的情况
86     if(b == 1) return 0;
87     int tb = b, tmp = 1;
88     for(int g = gcd(a, p), k=1; g != 1; g = gcd(a, p), k++) {
89         if(tb % g) return -1; //保证g为a,b,p的最大公约数
90         tb /= g; p /= g; tmp = tmp * a / g % p;
91         if(tmp == tb) return k;
92     }
93     return Bsgs(a, b, p);
94 }

```

5.21 离散对数问题 (高阶 BSGS)

```

1  /*
2   * Author: Simon
3   * 功能: 求解 $a^x \equiv b \pmod{p}$ , 其中 $p$ 可达 $1e18$ , 但 $p$ 的质因数个数很少。
4   * 复杂度: 复杂度 $O(k \cdot e_i \cdot \log(p_i^{e_i}) + k \cdot \log(p_i^{e_i}))$ 

```

```
5  * 其中 $k$ 为 $p-1$ 的质因子的个数,  $e_i$ 为 $p-1$ 的质因子的最高幂次。 $p_i$ 最高幂次对应的质因子。
6  */
7  #include <bits/stdc++.h>
8  using namespace std;
9  typedef int Int;
10 #define int __int128_t
11 #define INF 0x3f3f3f3f
12 #define maxn 200005
13 struct Istream {
14     template <class T>
15     Istream &operator >>(T &x) {
16         static char ch; static bool neg;
17         for(ch=neg=0; ch<'0' || '9'<ch; neg|=ch=='-', ch=getchar());
18         for(x=0; '0'<=ch && ch<='9'; (x*=10)+=ch-'0', ch=getchar());
19         x=neg?-x:x;
20         return *this;
21     }
22 }fin;
23 struct Ostream {
24     template <class T>
25     Ostream &operator <<(T x) {
26         x<0 && (putchar('-'), x=-x);
27         static char stack[233]; static int top;
28         for(top=0; x; stack[++top]=x%10+'0', x/=10);
29         for(top==0 && (stack[top=1]='0'); top; putchar(stack[top--]));
30         return *this;
31     }
32
33     Ostream &operator <<((char ch) {
34         putchar(ch);
35         return *this;
36     }
37 }fout;
38 vector<pair<int,int> >fac;
39 void solve(int n){ /*求解质因数 */
40     fac.clear();
41     for(int i=2; i*i<=n; i++){
42         int tmp=0;
43         if(n%i==0){
44             while(n%i==0) n/=i, tmp++;
45             fac.push_back({i,tmp});
46         }
47     }
48     if(n>1) fac.push_back({n,1});
49 }
50 int fpow(int a,int b,int mod){
51     int ans=1; a%=mod;
52     while(b){
53         if(b&1) (ans*=a)%=mod;
54         (a*=a)%=mod;
55         b>>=1;
56     }
57     return ans;
58 }
```

```

59 int proot(int p){ //求原根
60     for(int a=2;a<p;a++){
61         bool flag=0;
62         for(int i=0;i<fac.size();i++){
63             int v=fac[i].first;
64             if(fpow(a,(p-1)/v,p)==1){
65                 flag=1;break;
66             }
67         }
68         if(!flag) return a;
69     }
70     return -1;
71 }
72 int cal(int a,int b,int p,pair<int,int>fac){
73     int ans=0,t=fac.first;
74     map<int,int>mp;
75     for(int i=0;i<fac.first;i++) mp[fpow(a,i*(p-1)/t,p)]=i;
76     for(int i=1;i<=fac.second;i++){
77         int c=mp[fpow(b,(p-1)/t,p)];
78         (ans+=c*t/fac.first)%=p;
79         (b*=fpow(fpow(a,c*t/fac.first,p),p-2,p))%=p;
80         (t*=fac.first)%=p;
81     }
82     return ans;
83 }
84 int exgcd(int a,int b,int &x,int &y){
85     if(b==0){
86         x=1,y=0;
87         return a;
88     }
89     int g=exgcd(b,a%b,y,x);
90     y-=a/b*x;
91     return g;
92 }
93 int excrt(int mi[],int ai[],int n){ //扩展中国剩余定理
94     int M=mi[1],ans=ai[1];
95     for(int i=2;i<=n;i++){
96         int a=M,b=mi[i],c=((ai[i]-ans)%b+b)%b;
97         int x,y;
98         int gcd=exgcd(a,b,x,y);
99         if(c%gcd!=0) return -1;
100         a/=gcd,b/=gcd;
101         (x*=(c/gcd)%b)%=b; x=(x+b)%b;
102         ans+=x*M;
103         M*=b;
104         ans%=M;
105     }
106     return (ans+M)%M;
107 }
108 int pohlig_hellman(int a,int b,int p,vector<pair<int,int> >fac){ /*求解 $a^x=b \pmod p$ ,  $a$ 为 $p$ 的原根
    */
109     int mi[fac.size()+5]={0},ai[fac.size()+5]={0};
110     for(int i=0;i<fac.size();i++){
111         mi[i+1]=fpow(fac[i].first,fac[i].second,p); /* $p_i^{e_i}$  */

```

```

112     ai[i+1]=cal(a,b,p,fac[i]); /*求解 $x \equiv a \pmod{p^{e_i}}$  */
113 }
114     return excrt(mi,ai,fac.size()); /*扩展中国剩余定理合并同余方程 */
115 }
116 Int main(){
117     Int T;cin>>T;
118     while(T--){
119         int p,a,b;fin>>p>>a>>b; /*求 $a^x \equiv b \pmod{p}$  */
120         solve(p-1); /*求得 $p-1$ 的质因数, 及其幂次 */
121         int root=proot(p); /*求得 $p$ 的原根 */
122         int pa=pohlig_hellman(root,a,p,fac)/* $g^{pa} \equiv a \pmod{p}$  */,pb=pohlig_hellman(root,b,p,fac
            );/* $g^{pb} \equiv b \pmod{p}$  */
123         if(pa==0){ /*转换为求 $g^{pa \cdot x} \equiv g^{pb} \pmod{p}$ , 由欧拉定理得 $pa \cdot x \equiv pb \pmod{(p-1)}$  */
124             if(pb==0) fout<<1;
125             else fout<<-1;
126         }
127         else{ /*求解  $pa \cdot x \equiv pb \pmod{(p-1)}$  */
128             int x,y;
129             int gcd=exgcd(pa,p-1,x,y);
130             if(pb%gcd!=0){
131                 fout<<-1;
132                 cout<<endl;
133                 continue;
134             }
135             int B=(p-1)/gcd;pa/=gcd;
136             x*=pb/gcd;
137             fout<<((x%B+B)%B);
138         }
139         cout<<endl;
140     }
141     cin.get(),cin.get();
142     return 0;
143 }

```

5.22 二分分数树 (Stern-Brocot Tree)

```

1 //Author:CookieC
2 //未做模板调整, 请自行调整
3 #include <cmath>
4 #define LL long long
5 #define LD long double
6
7 void SternBrocot(LD X, LL &A, LL &B) {
8     A=X+0.5;
9     B=1;
10    if(A==X)
11        return;
12    LL la=X, lb=1, ra=X+1, rb=1;
13    long double C=A, a, b, c;
14    do {
15        a = la+ra;
16        b = lb+rb;

```



```

17     c = a/b;
18     if(std::abs(C-X) > std::abs(c-X)) {
19         A=a;
20         B=b;
21         C=c;
22         if(std::abs(X-C) < 1e-10) {
23             break;
24         }
25     }
26     if(X<c) {
27         ra=a;
28         rb=b;
29     } else {
30         la=a;
31         lb=b;
32     }
33     } while(lb+rb<=1e5);
34 }

```

5.23 二次剩余

```

1  /*
2   * Author: Simon
3   * 功能: 求解  $x^2 \equiv n \pmod{p}$ , 即  $x = \text{sqrt}(n) \pmod{p}$ 
4   * 复杂度:  $O(\text{sqrt}(p))$ 
5   */
6
7  /*类似复数 单位元为w(复数的单位元为-1)*/
8  struct Complex {
9      int x, y, w;
10     Complex() {}
11     Complex(int x, int y, int w) : x(x), y(y), w(w) {}
12 };
13 /*类复数乘法 */
14 Complex mul(Complex a, Complex b, int p) {
15     Complex ans;
16     ans.x = (a.x * b.x % p + a.y * b.y % p * a.w % p) % p;
17     ans.y = (a.x * b.y % p + a.y * b.x % p) % p;
18     ans.w = a.w;
19     return ans;
20 }
21 /*类复数快速幂 */
22 Complex Complexfpow(Complex a, int b, int mod) {
23     Complex ans = Complex(1, 0, a.w);
24     while (b) {
25         if (b & 1) ans = mul(ans, a, mod);
26         a = mul(a, a, mod);
27         b >>= 1;
28     }
29     return ans;
30 }
31 int fpow(int a, int b, int mod) {

```

```

32     int ans = 1;
33     a %= mod;
34     while (b) {
35         if (b & 1) (ans *= a) %= mod;
36         (a *= a) %= mod;
37         b >>= 1;
38     }
39     return ans;
40 }
41 /*求解 $x^2=n(mod\ p)$  */
42 int solve(int n, int p) {
43     n %= p;
44     if (n == 0) return 0;
45     if (p == 2) return n;
46     if (fpow(n, (p - 1) / 2, p) == p - 1) return -1; /*勒让德定理判断 $n$ 不是 $p$ 的二次剩余 */
47     mt19937 rnd(time(0));
48     int a, t, w;
49     do {
50         a = rnd() % p;
51         t = a * a - n;
52         w = (t % p + p) % p; /*构造 $w=a^2-n$  */
53     } while (fpow(w, (p - 1) / 2, p) != p - 1); /*找到一个 $w$ 不是 $p$ 的二次剩余 */
54     Complex ans = Complex(a, 1, w);
55     ans = Complexfpow(ans, (p + 1) / 2, p); /*答案为 $(a+w)^{(p+1)/2}$  */
56     return ans.x;
57 }
58 int exgcd(int a, int b, int &x, int &y) {
59     if (b == 0) {
60         x = 1, y = 0;
61         return a;
62     }
63     int g = exgcd(b, a % b, y, x);
64     y -= a / b * x;
65     return g;
66 }
67 /*求解 $x^2=n(mod\ p^k)$  */
68 int exsolve(int n, int p, int k, int pk) {
69     int r = solve(n, p); /*先求出 $x^2=n(mod\ p)$  */
70     if (r == -1) return -1;
71     Complex ans = Complex(r, 1, n);
72     ans = Complexfpow(ans, k, pk); /*求出 $(r+sqrt(n))^k=t+u*sqrt(n)$  */
73     int a = ans.y, b = pk, c = ans.x, x, y;
74     int g = exgcd(a, b, x, y); /*求解 $u*x=t(mod\ p^k)$  */
75     if (c % g != 0) return -1;
76     a /= g; b /= g;
77     x *= c / g;
78     return (x % b + b) % b;
79 }

```

5.24 计算莫比乌斯函数

```

1 int prime[maxn], mu[maxn], cnt = 0;

```

```

2  bool vis[maxn];
3  void Mobius(int n){
4      mu[1]=1;
5      for(int i=2;i<=n;i++){
6          if(!vis[i]){
7              prime[++cnt]=i;
8              mu[i]=-1;
9          }
10         for(int j=1;j<=cnt && prime[j]*i<=n;j++){
11             vis[prime[j]*i]=1;
12             if(i%prime[j]==0) break;
13             mu[i*prime[j]]=-mu[i];
14         }
15     }
16 }

```

5.25 杜教筛

```

1  int DuJiao(int n)// 杜教筛——欧拉函数之和
2  {
3      if(n<maxn) return Phi[n]; //欧拉函数前缀和
4      if(mp[n]!=-1) return mp[n];
5      int sum=0,z=n%mod;
6      // for(int l=2,r;l<=n;l=r+1) // #version 1
7      // {
8      //     r=n/(n/l);
9      //     sum+=DuJiao(n/l)*(r-l+1);
10     //     sum%=mod;
11     // }
12     for(int i=1;i*i<=n;i++) // #vsesion 2——对每一个i=[2...n]求sum[phi(1)+...+phi(n/i)]
13     {
14         sum+=DuJiao(i)*(n/i-n/(i+1));
15         sum%=mod;
16         int x=n/i; //x为值,枚举i求x;
17         if(x==i||i==1) continue;
18         sum+=DuJiao(x)*(n/x-n/(x+1));
19         sum%=mod;
20     }
21     sum=((z*(z+1)%mod*inv2%mod)%mod-sum%mod+mod)%mod; //等差数列前n项和-sum
22     mp.insert(n,sum); //加入HashMap
23     return sum%mod;
24 }

```

5.26 Min25 筛

```

1  /*
2   * Author: Simon
3   * 复杂度:  $O(n^{\{3/4\}}/\text{Log}n)$ 
4   * 功能: 解决一类积形函数前缀和问题

```

```

5  * 适用条件：在质数处表达式为多项式，在质数的高次幂处可以快速求值
6  */
7  #include <bits/stdc++.h>
8  using namespace std;
9  typedef int Int;
10 #define int long long
11 #define INF 0x3f3f3f3f
12 #define maxn 1000005
13 const int mod=1e9+7;
14 int prime[maxn],cnt=0,w[maxn],g[maxn],h[maxn],m=0;
15 int id1[maxn],id2[maxn],Sqr,sp[maxn];
16 bool vis[maxn]={1,1};
17 void Euler(int n){
18     for(int i=2;i<=n;i++){
19         if(!vis[i]) prime[++cnt]=i,sp[cnt]=sp[cnt-1]+i;
20         for(int j=1;j<=cnt&&i*prime[j]<=n;j++){
21             vis[i*prime[j]]=1;
22             if(i%prime[j]==0) break;
23         }
24     }
25 }
26 int getG(int p){
27     return p;
28 }
29 int getH(int p){
30     return 1;
31 }
32 int getSigmaG(int p){ /*素数和 */
33     return sp[p];
34 }
35 int getSigmaH(int p){
36     return p;
37 }
38 int getF(int p,int e){
39     return (p^e);
40 }
41 int fpow(int a,int b,int mod){
42     int ans=1;a%=mod;
43     while(b){
44         if(b&1) (ans*=a)%=mod;
45         (a*=a)%=mod;
46         b>>=1;
47     }
48     return ans;
49 }
50 int S(int x,int y,int n){
51     if (x<=1||prime[y]>x) return 0;
52     int k=(x<=Sqr)?id1[x]:id2[n/x];
53     int res=((1LL*g[k]-h[k])-(getSigmaG(y-1)-getSigmaH(y-1)))%mod; /*g(n,|P|)-sigma(f(P_i)) */
54     res=(res+mod)%mod;
55     if (y==1) res+=2; //特判。
56     for (int i=y;i<=cnt&&1LL*prime[i]*prime[i]<=x;++i){
57         int p1=prime[i],p2=1LL*prime[i]*prime[i];
58         for (int e=1;p2<=x;++e,p1=p2,p2*=prime[i])

```

```

59         (res+=(1LL*S(x/p1,i+1,n)*getF(prime[i],e)%mod+getF(prime[i],(e+1))%mod)%mod)%=mod;
60     }
61     return res;
62 }
63 Int main()
64 {
65     ios::sync_with_stdio(false);
66     cin.tie(0);
67     int n;cin>>n;
68     Sqr=sqrt(n);Euler(Sqr);
69     for(int i=1,j;i<=n;i=j+1){ /*f(i)=g(i)-h(i) */
70         j=n/(n/i);
71         w[++m]=n/i; /*预处理离散化 xk=n/i */
72         h[m]=(w[m]-1)%mod; /*h(m,0)即h函数的前缀和减去h(1) */
73         g[m]=((w[m]+1)%mod*w[m]%mod*fpow(2,mod-2,mod)%mod-1+mod)%mod; /*g(m,0)即g函数的前缀和
            减去g(1)*/
74         w[m]<=Sqr?id1[w[m]]=m:id2[n/w[m]]=m;
75     }
76     for(int j=1;j<=cnt;j++){
77         for(int i=1;i<=m&&prime[j]*prime[j]<=w[i];i++){
78             int k=w[i]/prime[j]<=Sqr?id1[w[i]/prime[j]]:id2[n/(w[i]/prime[j])];
79             (g[i]-=getG(prime[j])%mod*(g[k]-getSigmaG(j-1))%mod+mod)%=mod; /*根据转移方程即g(i,
                j)=-f(P_j)*g(n/P_j,j-1)-sigma(f(P_i)) */
80             (h[i]-=getH(prime[j])%mod*(h[k]-getSigmaH(j-1))%mod+mod)%=mod; /*h函数转移同上 */
81         }
82     }
83     cout<<((S(n,1,n)+1)%mod+mod)%mod<<endl;
84     cin.get(),cin.get();
85     return 0;
86 }

```

5.27 常用公式

- 约数定理：若 $n = \prod_{i=1}^k p_i^{a_i}$ ，则
 - 约数个数 $f(n) = \prod_{i=1}^k (a_i + 1)$
 - 约数和 $g(n) = \prod_{i=1}^k (\sum_{j=0}^{a_i} p_i^j)$
- 小于 n 且互素的数之和为 $n\varphi(n)/2$
- 若 $\gcd(n, i) = 1$ ，则 $\gcd(n, n-i) = 1 (1 \leq i \leq n)$
- 错排公式： $D(n) = (n-1)(D(n-2) + D(n-1)) = \sum_{i=2}^n \frac{(-1)^k n!}{k!} = \lfloor \frac{n!}{e} + 0.5 \rfloor$
- 威尔逊定理： $p \text{ is prime} \Rightarrow (p-1)! \equiv -1 \pmod{p}$
- 欧拉定理： $\gcd(a, n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1 \pmod{n}$
- 欧拉定理推广： $\gcd(n, p) = 1 \Rightarrow a^n \equiv a^{n \% \varphi(p)} \pmod{p}$
- 素数定理：对于不大于 n 的素数个数 $\pi(n)$ ， $\lim_{n \rightarrow \infty} \frac{\pi(n)}{n} = \frac{1}{\ln n}$
- 位数公式：正整数 x 的位数 $N = \log_{10}(n) + 1$

10. 斯特灵公式 $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$

11. 设 $a > 1, m, n > 0$, 则 $\gcd(a^m - 1, a^n - 1) = a^{\gcd(m, n)} - 1$

12. 设 $a > b, \gcd(a, b) = 1$, 则 $\gcd(a^m - b^m, a^n - b^n) = a^{\gcd(m, n)} - b^{\gcd(m, n)}$

$$G = \gcd(C_n^1, C_n^2, \dots, C_n^{n-1}) = \begin{cases} n, & n \text{ is prime} \\ 1, & n \text{ has multy prime factors} \\ p, & n \text{ has single prime factor } p \end{cases}$$

$$\gcd(\text{Fib}(m), \text{Fib}(n)) = \text{Fib}(\gcd(m, n))$$

13. 若 $\gcd(m, n) = 1$, 则:

(a) 最大不能组合的数为 $m * n - m - n$

(b) 不能组合数个数 $N = \frac{(m-1)(n-1)}{2}$

14. $(n+1)\text{lcm}(C_n^0, C_n^1, \dots, C_n^{n-1}, C_n^n) = \text{lcm}(1, 2, \dots, n+1)$

15. 若 p 为素数, 则 $(x + y + \dots + w)^p \equiv x^p + y^p + \dots + w^p \pmod{p}$

16. 卡特兰数: 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012

$$h(0) = h(1) = 1, h(n) = \frac{(4n-2)h(n-1)}{n+1} = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1}$$

17. 伯努利数: $B_n = -\frac{1}{n+1} \sum_{i=0}^{n-1} C_{n+1}^i B_i$

$$\sum_{i=1}^n i^k = \frac{1}{k+1} \sum_{i=1}^{k+1} C_{k+1}^i B_{k+1-i} (n+1)^i$$

18. FFT 常用素数

$r \cdot 2^k + 1$	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353	119	23	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

5.28 数论公式

1. 小于 n 的 i, j , $\gcd(i, j) = 1$ 的 i, j 的对数与欧拉函数的关系 $\sum_{i=1}^n \sum_{j=1}^n [\gcd(i, j) = 1] = 2 \sum_{i=1}^n \sum_{j=1}^i [\gcd(i, j) = 1] - \sum_{i=1}^n [\gcd(i, i) = 1] = (2 \sum_{i=1}^n \varphi(i)) - 1$

2. 小于 n 的 i, j , 且 $\gcd(i, j) = 1$ 时, 所有 $i \cdot j$ 的和与欧拉函数的关系 $\sum_{i=1}^n i \sum_{j=1}^n [\gcd(i, j) = 1] \cdot j = 2 \sum_{i=1}^n i \sum_{j=1}^i [\gcd(i, j) = 1] \cdot j - \sum_{i=1}^n [\gcd(i, i) = 1] \cdot i$
 $= \left(2 \sum_{i=1}^n i \frac{i \cdot \varphi(i) + [i=1]}{2} \right) - 1 = \left(\sum_{i=1}^n i^2 \cdot \varphi(i) + [i=1] \right) - 1$
 3. 约数, 倍数之间重要的变换 $\sum_{k=1}^n \sum_{d|k} d \cdot k = \sum_{k=1}^n \sum_{d=1}^{\frac{n}{k}} d \cdot k \cdot d = \sum_{d=1}^n \sum_{k|d} d \cdot \frac{d}{k} = \sum_{d=1}^n \sum_{k=1}^{\frac{n}{d}} d \cdot k \cdot d =$
 $\sum_{d=1}^n \sum_{d|k} d \cdot k = \sum_{k=1}^n \sum_{k|d} d \cdot \frac{d}{k}$
 4. $C_m^n = C_{m-1}^{n-1} + C_{m-1}^n$
 5. $C_n^m = \frac{n!}{m!(n-m)!}$
 6. $C_n^m = C_n^{n-m} = C_{n-1}^m + C_{n-1}^{m-1}$ (杨辉三角)
 7. $C_n^0 + C_n^1 + C_n^2 + \dots + C_n^n = \sum_{i=0}^n C_n^i = 2^n$
 8. $C_n^0 + C_n^2 + C_n^4 + \dots = C_n^1 + C_n^3 + C_n^5 + \dots = 2^{n-1}$
 9. $C_n^m + C_{n+1}^m + C_{n+2}^m + \dots + C_{n+m}^m = \sum_{i=0}^m C_{n+i}^m = C_{n+m+1}^{m+1}$
 10. $kC_n^k = nC_{n-1}^{k-1}, \frac{C_n^k}{k+1} = \frac{C_{n+1}^{k+1}}{n+1}$
 11. $\sum_{k=0}^n k * C_m^k = \sum_{k=0}^{n-1} m * C_{m-1}^k$
 12. $C_m^n \% p = C_{m/p}^{n/p} * C_{m \% p}^{n \% p} \% p$ (Lucas 定理)
 13. $C_{m+n}^k = \sum_{i=0}^k C_m^i C_n^{k-i}$ (Vandermonde 恒等式)
 14. $\frac{n!}{\prod_{i=1}^k a_i}$ (有重复元素的全排列公式, a_i 为第 i 种元素的个数, k 为元素种类数)
-

第六章 数学

6.1 $C(n,m) \bmod p$ (n 很大 p 可以很大)

```

1 LL C(const LL &n, const LL &m, const int &pr) {
2     LL ans = 1;
3     for (int i = 1; i <= m; i++) {
4         LL a = (n - m + i) % pr;
5         LL b = i % pr;
6         ans = (ans * (a * Inv(b, pr) % pr) % pr) % pr;
7     }
8     return ans;
9 }
10 /*
11  * Author: Simon
12  * 功能: 求组合数
13  * 复杂度:  $O(p+\log(p))$ 
14  */
15 int bit[maxn]; //阶乘数组, 对 $p$ 取模, 则只需初始化 $1\sim p$ 的阶乘即可。
16 int C(const int n, const int m, const int p){
17     if (n < m) return 0;
18     return bit[n] * fpow(bit[m], p - 2, p) % p * fpow(bit[n - m], p - 2, p) % p;
19 }

```

6.2 Lucas 定理

```

1 //C(n, m) mod p (n 很大 p 较小 (不知道能不能为非素数))
2 LL Lucas(LL n, LL m, const int &pr) {
3     if (m == 0) return 1;
4     return C(n % pr, m % pr, pr) * Lucas(n / pr, m / pr, pr) % pr;
5 }

```

6.3 计算从 $C(n, 0)$ 到 $C(n, p)$ 的值

```

1 //by Yuhao Du
2 int p;
3 std::vector<int> gao(int n) {
4     std::vector<int> ret(p+1, 0);
5     if (n==0) {
6         ret[0]=1;
7     } else if (n%2==0) {

```

```

8      std::vector<int> c = gao(n/2);
9      for(int i = 0; i <= p+1; i++) {
10         for(int j = 0; j <= p+1; j++) {
11            if (i+j<=p) ret[i+j]+=c[i]*c[j];
12        }
13    }
14 } else {
15     std::vector<int> c = gao(n-1);
16     for(int i = 0; i <= p+1; i++) {
17         for(int j = 0; j <= 2; j++) {
18            if (i+j<=p) ret[i+j]+=c[i];
19        }
20    }
21 }
22 return ret;
23 }

```

6.4 求贝尔数（模 p）

```

1  /*
2   * Author: Simon
3   * 功能：模p意义下用记忆化搜索快速求第n个贝尔数
4   * 描述：贝尔数为集合的划分数
5   */
6  #include<iostream>
7  #include<cstring>
8  #include<algorithm>
9  #include<cstdio>
10 using namespace std;
11 #define int long long
12 #define maxn 1000005
13 #define INF 0x3f3f3f3f
14 int n,p;
15 int a[maxn],fac[maxn],inv[maxn];
16 int dfs(int n){
17     if(a[n]) return a[n];
18     if(n-p>=0&& n+1-p>=0){ /*a[n]=(a[n-p]+a[n+1-p])%p */
19         return a[n]=(dfs(n-p)+dfs(n+1-p))%p;
20     }
21     int tmp=1,sum=0;
22     for(int k=0;k<n;k++){ /*如果n小于p，则暴力求a[n]*/
23         tmp=fac[n-1]%p*inv[k]%p*inv[n-1-k]%p;
24         sum=(sum+tmp*dfs(k)%p)%p;
25     }
26     return a[n]=sum;
27 }
28 int fpow(int a,int b,int mod){
29     int ans=1;a%=mod;
30     while(b){
31         if(b&1) ans=1LL*ans*a%mod;
32         a=1LL*a*a%mod;
33         b>>=1;

```

```

34     }
35     return ans;
36 }
37 signed main(){
38     ios::sync_with_stdio(false);
39     cin.tie(0);
40     int T;cin>>T;
41     while(T--){
42         cin>>n>>p;
43         memset(fac,0,sizeof(fac));
44         memset(inv,0,sizeof(inv));
45         memset(a,0,sizeof(a));
46         a[0]=1;a[1]=1; /*初始化前2个贝尔数 */
47         int t=min(n,p-1); /*模p意义下，预处理t个阶乘即可 */
48         fac[0]=1;for(int i=1;i<=t;i++) fac[i]=fac[i-1]*i%p;
49         inv[t]=fpow(fac[t],p-2,p);
50         for(int i=t;i>=1;i--){
51             inv[i-1]=inv[i]*i%p;
52         }
53         cout<<dfs(n)<<endl;
54     }
55     return 0;
56 }

```

6.5 计算第一类斯特林数

```

1  int seq[60][maxn << 1] , ptr = 0;
2  long long B[maxn << 1] , C[maxn << 1];
3
4  int DFS( int l , int r ){
5      if( l == r ){
6          int id = ptr ++ ;
7          seq[id][1] = 1 ;
8          seq[id][0] = 1 ;
9          return id;
10     } else {
11         int mid = l + r >> 1;
12         int lid = DFS( l , mid );
13         int rid = DFS( mid + 1 , r );
14         ptr -= 2;
15         int newid = ptr ++ ;
16         int len = 1;
17         while( len <= r - l + 1 ) len <<= 1;
18         for(int i = 0 ; i < len ; ++ i) B[i] = seq[lid][i] , C[i] = seq[rid][i] , seq[lid][i]
            = seq[rid][i] = 0;
19         ntt( B , len , 1 );
20         ntt( C , len , 1 );
21         for(int i = 0 ; i < len ; ++ i) B[i] = B[i] * C[i] % Mod;
22         ntt( B , len , -1 );
23         for(int i = 0 ; i < len ; ++ i) seq[newid][i] = B[i];
24         return newid;
25     }

```

```

26 }
27
28 //int id = DFS( 0 , N - 1 );
29 //for(int i = N ; i >= 0 ; — i) {
30 // printf( "f[%d] is %d \n" , N - i , seq[id][i] );
31 //}

```

6.6 自适应辛普森

```

1 double F(double x) {
2     //Simpson公式用到的函数
3 }
4 double simpson(double a, double b) { //三点Simpson法, 这里要求F是一个全局函数
5     double c = a + (b - a) / 2;
6     return (F(a) + 4 * F(c) + F(b)) * (b - a) / 6;
7 }
8 double asr(double a, double b, double eps, double A) { //自适应Simpson公式 (递归过程)。已知整
    个区间[a,b]上的三点Simpson值A
9     double c = a + (b - a) / 2;
10    double L = simpson(a, c), R = simpson(c, b);
11    if (fabs(L + R - A) <= 15 * eps) return L + R + (L + R - A) / 15.0;
12    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
13 }
14 double asr(double a, double b, double eps) { //自适应Simpson公式 (主过程)
15     return asr(a, b, eps, simpson(a, b));
16 }

```

6.7 博弈论

```

1 Nim Game
2     最经典最基础的博弈。
3     n堆石子, 双方轮流从任意一堆石子中取出至少一个, 不能取的人输。
4     对于一堆x个石子的情况, 容易用归纳法得到SG(x)=x。
5     所以所有石子个数的异或和为0是必败态, 否则为必胜态。
6
7 Bash Game
8     每人最多一次只能取m个石子, 其他规则同Nim Game。
9     依旧数学归纳... SG(x)=x mod (m+1)。
10
11 NimK Game
12     每人一次可以从最多K堆石子中取出任意多个, 其他规则同Nim Game。
13     结论: 在二进制下各位上各堆石子的数字之和均为(K+1)的倍数的话则为必败态, 否则为必胜态。
14     这个证明要回到原始的方法上去。
15     补: 这个游戏还可以推广, 即一个由n个子游戏组成的游戏, 每次可以在最多K个子游戏中进行操作。
16     然后只要把结论中各堆石子的个数改为各个子游戏的SG值即可, 证明也还是一样的。
17
18 Anti-Nim Game
19     似乎又叫做Misère Nim。
20     不能取的一方获胜, 其他规则同Nim Game。

```

关于所谓的”Anti-SG游戏”及”SJ定理”贾志鹏的论文上有详细说明,不过似乎遇到并不多.
结论是一个状态是必胜态当且仅当满足以下条件之一:
SG值不为0且至少有一堆石子数大于1;
SG值为0且不存在石子数大于1的石子堆.

Fibonacci Nim

有一堆个数为 $n(n \geq 2)$ 的石子, 游戏双方轮流取石子, 规则如下:
1) 先手不能在第一次把所有的石子取完, 至少取1颗;
2) 之后每次可以取的石子数至少为1, 至多为对手刚取的石子数的2倍。
约定取走最后一个石子的人为赢家。
结论: 当 n 为Fibonacci数的时候, 必败。

Staircase Nim

每人一次可以从第一堆石子中取走若干个, 或者从其他石子堆的一堆中取出若干个放到左边一堆里(没有石子的石子堆不会消失), 其他规则同Nim Game。
这个游戏的结论比较神奇:
当且仅当奇数编号堆的石子数异或和为0时为必败态。
简单的理解是从偶数编号堆中取石子对手又可以放回到奇数编号堆中, 而且不会让对手不能移动. 比较意识流, 然而可以归纳证明。

Wythoff Game

有两堆石子, 双方轮流从某一堆取走若干石子或者从两堆中取走相同数目的石子, 不能取的人输。
容易推理得出对任意自然数 k , 都存在唯一的一个必败态使得两堆石子数差为 k , 设其为 $P_k = (a_k, b_k)$, 表示石子数分别为 $a_k, b_k (a_k \leq b_k)$ 。
那么 a_k 为在 $P_k (k < k)$ 中未出现过的最小自然数, $b_k = a_k + k$ 。
数学班的说, 用Betty定理以及显然的单调性就可以推出神奇的结论:
 $a_k = \text{floor}(k * 5\sqrt{5} + 12), b_k = \text{floor}(k * 5\sqrt{5} + 32)$ 。

Take & Break

有 n 堆石子, 双方轮流取出一堆石子, 然后新增两堆规模更小的石子堆(可以没有石子), 无法操作者输。
这个游戏似乎只能暴力SG, 知道一下就好。

树上删边游戏

给出一个有 n 个结点的树, 有一个点作为树的根节点, 双方轮流从树中删去一条边, 之后不与根节点相连的部分将被移走, 无法操作者输。
结论是叶子结点的SG值为0, 其他结点SG值为其每个儿子结点SG值加1后的异或和, 证明也并不复杂。

翻硬币游戏

n 枚硬币排成一行, 有的正面朝上, 有的反面朝上。
游戏者根据某些约束翻硬币 (如: 每次只能翻一或两枚, 或者每次只能翻连续的几枚), 但他所翻动的硬币中, 最右边的必须是从正面翻到反面。
谁不能翻谁输。

需要先开动脑筋把游戏转化为其他的取石子游戏之类的, 然后用如下定理解决:
局面的 SG 值等于局面中每个正面朝上的棋子单一存在时的 SG 值的异或和。

无向图删边游戏

一个无向连通图, 有一个点作为图的根。
游戏者轮流从图中删去边, 删去一条边后, 不与根节点相连的部分将被移走。
谁无路可走谁输。

对于这个模型, 有一个著名的定理——Fusion Principle:

我们可以对无向图做如下改动: 将图中的任意一个偶环缩成一个新点, 任意一个奇环缩成一个新点加一个新边; 所有连到原先环上的边全部改为与新点相连。 这样的改动不会影响图的 SG 值。

6.8 SG 函数

```

1  /*
2   * author: Simon
3   * f[m]:可改变当前状态的方式, m为方式的种类, f[m]要在getSG之前先预处理
4   * sg[]:0~n的SG函数值
5   * mex[]:为x后继状态的集合
6   * 若sg值为正数, 则先手必赢, 否则若为0, 则先手必输。
7   */
8  int f[maxn],sg[maxn],mex[maxn];
9  void getSG(int n/*要求多少个sg值*/,int m/*有多少种操作方式*/){
10     memset(sg,0,sizeof(sg));
11     for(int i = 1; i <= n; i++){ /*因为SG[0]始终等于0, 所以i从1开始*/
12         memset(mex,0,sizeof(mex)); /*每一次都要将上一状态 的后继集合 重置*/
13         for(int j = 0; f[j] <= i && j < m; j++)
14             mex[sg[i-f[j]]] = 1; /*将后继状态的SG函数值进行标记*/
15         for(int j = 0;; j++) if(!mex[j]){ /*查询当前后继状态SG值中最小的非零值*/
16             sg[i] = j;
17             break;
18         }
19     }
20 }

```

6.9 矩阵快速幂

```

1  //Author: Simon
2  #define maxn 16
3  const int mod=1e9+7;
4  struct Matrix{ //矩阵类
5      int m[maxn][maxn];
6      Matrix(){
7          for(int i=0;i<maxn;i++) for(int j=0;j<maxn;j++) m[i][j]=0;
8      }
9      void init(){
10         for(int i=0;i<maxn;i++) m[i][i]=1;
11     }
12     void set(int len){ //构造矩阵, 根据题目变化
13         for(int i=0;i<len;i++){
14             for(int j=i-1;j<=i+1;j++){
15                 if(j<0||j>=len) continue;
16                 m[i][j]=1;
17             }
18         }
19     }
20     int *operator [](int x){
21         return m[x];
22     }
23 };

```

```

24 Matrix operator *(Matrix a, Matrix b){ //矩阵乘法,多组数据可以加个全局变量Len控制矩阵大小O(Len
    ^3)
25     Matrix c;
26     for(int i=0;i<maxn;i++){
27         for(int j=0;j<maxn;j++){
28             for(int k=0;k<maxn;k++){
29                 c[i][j]=(c[i][j]+a[i][k]*b[k][j])%mod;
30             }
31         }
32     }
33     return c;
34 }
35 Matrix fpow(Matrix a,int b){ //矩阵快速幂
36     Matrix c;c.init();
37     while(b){
38         if(b&1) c=c*a;
39         a=a*a;
40         b>>=1;
41     }
42     return c;
43 }
44 Matrix ans; //答案矩阵,仅第一列有用,ans[0][0]=f(n)
45 void init(int x){ //若题目类型为分段求和,则可能使用
46     for(int i=x;i<maxn;i++){
47         ans[i][0]=0;
48     }
49 }

```

6.10 高斯消元

```

1  /*
2  Author: Simon
3  返回自由变量个数, -1表示无解。
4  若矩阵的秩=增广矩阵的秩=变量个数, 则有唯一解
5  若矩阵的秩=增广矩阵的秩<变量个数, 则有无穷多解
6  若矩阵的秩<增广矩阵的秩, 则无解
7
8  注: 若用于开关问题, 则使用注释部分。
9  复杂度: O(n^3)
10 */
11 const double/*开关问题: int*/ eps = 1e-10;
12 // int n/*方程个数*/, m/*变量个数*/;
13 double/*开关问题: int*/ a[maxn][maxn]/*增广矩阵(n*(m+1)), 开关问题: a[i][j]表示与j关联的开关为i
    */, x[maxn]/*解*/;
14 bool free_x[maxn]; /*是否为自由变量*/
15 int free_xx[maxn]/*用于自由变量*/, free_num=0/*自由变量个数*/;
16
17 void printMatrix(double/*开关问题: int*/ a[maxn][maxn], int n, int m){
18     for(int i=0;i<n;i++){
19         for(int j=0;j<=m;j++){
20             cout<<left<<setw(20)<<a[i][j];
21         }

```

```

22     cout<<endl;
23 }
24 }
25
26 inline int sgn(double/*开关问题: int*/ x) { return (x > eps) - (x < -eps)/*开关问题: x?1:0*/;
    } //若x不接近0, 返回1, 否则返回0。
27
28 int Gauss(double/*开关问题: int*/ a[maxn][maxn]/*增广矩阵*/, int n/*方程个数*/, int m/*变量个
    数*/) {
29     memset(free_x, 1, sizeof(free_x)); memset(x, 0, sizeof(x));
30     int r = 0/*第r行*/, c = 0/*第c列*/;
31     while (r < n && c < m) {/*化为上三角矩阵*/
32         int m_r = r;
33         for(int i=r+1;i<n;i++) if (fabs(a[i][c]) > fabs(a[m_r][c])) m_r = i; /*从第r行开始, 找
            出第c列绝对值最大的 */
34         if (m_r != r){
35             for(int j=c;j<m+1;j++) swap(a[r][j], a[m_r][j]); /*将值最大的放到第r行*/
36         }
37         if (!sgn(a[r][c])) { /*判断a[r][c]是否为零*/
38             free_xx[free_num++]=c; /*记录自由变量*/
39             a[r][c] = 0; ++c;
40             continue;
41         }
42         for(int i=r+1;i<n;i++){ /*将第c列化为上三角*/
43             if (a[i][c]) {
44                 double/*开关问题: int*/ t = a[i][c] / a[r][c];/*开关问题: 删除*/
45                 for(int j=c;j<m+1;j++) a[i][j] -= a[r][j] * t/*开关问题: a[i][j]^=a[r][j]*/;
46             }
47         }
48         ++r; ++c;
49     }
50     for(int i=r;i<n;i++) if(sgn(a[i][m])) return -1;/*若xi=0,b!=0则无解*/
51     if (r < m) { /*此时r为矩阵的秩,也是增广矩阵的秩*/
52         for(int i=r-1;i>=0;i--){
53             int f_cnt = 0, k = -1;
54             for(int j=0;j<m;j++){
55                 if (sgn(a[i][j]) && free_x[j]) {
56                     ++f_cnt; k = j;
57                 }
58             }
59             if(f_cnt > 0) continue;
60             double/*开关问题: int*/ s = a[i][m];
61             for(int j=0;j<m;j++) if (j != k) s -= a[i][j] * x[j]/*开关问题: s^=(a[i][j]&x[j])
                */;
62             x[k] = s / a[i][k]/*开关问题: x[k]=s*/;/**/ free_x[k] = 0;
63         }
64         return m - r;/*有无穷多解, 且自由变量的个数为m-r*/
65     }
66     for(int i=m-1;i>=0;i--){/*回代解方程组*/
67         double/*开关问题: int*/ s = a[i][m];
68         for(int j=i+1;j<m;j++) s -= a[i][j] * x[j]/*开关问题: s^=(a[i][j]&x[j])*/;
69         x[i] = s / a[i][i]/*开关问题: x[i]=s*/;
70     }
71     return 0;/*有唯一解*/

```



```

72 }
73 /*
74     当线性方程组的解有无穷多时，可给自由变量任意合理的值，
75     然后通过自由变量获得的值，回代求得其他变量的值。
76     可用于开关问题中，用最少的操作使得所有开关达到一定状态
77
78     位运算开关问题模板：
79 */
80 int solve(int n,int m,int b/*自由变量的个数*/){
81     int ans=INF;
82     for(int i=0;i<(1<<b);i++){ /*枚举自由变量的 取值状态*/
83         int num=0;
84         for(int j=0;j<b;j++){
85             if(i&(1<<j)) x[free_xx[j]]=1,num++; /*根据状态将自由变量赋值*/
86             else x[free_xx[j]]=0;
87         }
88         for(int j=m-b-1;j>=0;j--){ /*根据自由变量的值回代求解其他变量*/
89             int id; for(id=j;id<m;id++) if(a[j][id]) break;
90
91             int s = a[j][m];
92             for(int k=id+1;k<m;k++) s^=(a[j][k]&x[k]);
93             x[id]=s;
94
95             num+=x[id];
96         }
97         ans=min(ans,num);
98     }
99     return ans;
100 }

```

6.11 异或线性基

```

1  /*
2   * Author: Simon
3   * 功能：线性基的插入
4   */
5  int a[maxn];
6  void insert(int val){
7      for(int i=60;i>=0;i--){
8          if(val&(1LL<<i)){
9              if(!a[i]){a[i]=val;break;}
10             else val^=a[i];
11         }
12     }
13 }
14 /*
15 * Author: Simon
16 * 功能：将上三角矩阵线性基化为对角矩阵形式
17 */
18 int p[maxn],cnt=0;
19 void rebuild(int n){
20     for(int i=60;i>=0;i--){

```

```
21     for(int j=i-1;j>=0;j--){
22         if(a[i]&(1LL<<j)) a[i]^=a[j];
23     }
24     cnt=0;
25     for(int i=60;~i;i--){
26         if(a[i]) p[cnt++]=a[i];
27     }
28 }
29 /*
30  * Author: Simon
31  * 功能：线性基的合并
32  */
33 void merge(int *g){
34     for(int i=0;i<=60;i++){
35         if(g[i]) insert(g[i]);
36     }
37 }
38 /*
39  * Author: Simon
40  * 功能：线性基查询最大值
41  */
42 int query(){
43     int ans=0;
44     for(int i=60;~i;i--){
45         if(ans^a[i]>ans) ans^=a[i];
46     }
47     return ans;
48 }
49 /*
50  * Author: Simon
51  * 功能：线性基查询最小值
52  */
53 int query(){
54     for(int i=0;i<=60;i++){
55         if(a[i]) return a[i];
56     }
57     return 0;
58 }
59 /*
60  * Author: Simon
61  * 功能：线性基化查询第k小值（无重复）
62  */
63 int query(int k,int cnt) /*需先化为对角矩阵 */
64     int ans=0;
65     for(int i=0;i<cnt;i++){
66         if(k>>i&1LL) ans^=p[i];
67     }
68 }
69 /*
70  * Author: Simon
71  * 功能：在线查询区间最大异或和
72  * 按右端点分类，构造n个线性基，并记录每个值插入的位置
73  * 同时保证插入时靠右的值具有优先插入权
74  */
```

```

75 int a[maxn],base[maxn][25]/*最大位置到i的线性基 */,pos[maxn][25];
76 void insert(int val,int p){
77     int k=p;
78     for(int i=0;i<=20;i++) base[p][i]=base[p-1][i],pos[p][i]=pos[p-1][i]; /*复制最大位置为p-1
        的线性基,在此基础上插入 */
79     for(int i=20;i>=0;i--){ if(val>>i&1){
80         if(!base[p][i]){
81             base[p][i]=val;
82             pos[p][i]=k;
83             break;
84         }
85         if (k > pos[p][i]) {
86             swap(pos[p][i], k); /*位置大的优先,注意交换的位置是k,不是原数p */
87             swap(base[p][i], val);
88         }
89         val ^= base[p][i];
90     }
91 }
92 int query(int l,int r){
93     int ans=0;
94     for(int i=20;i>=0;i--){
95         if((ans^base[r][i])>ans&&pos[r][i]>=1) ans^=base[r][i];
96     }
97     return ans;
98 }
99 /*
100 * Author: Simon
101 * 功能: 线性基查询q在n个数的任意组合的异或值的排名(有重复)
102 * n个数,它们随意组合的异或值有2^n个数,但是去重后只有2^r个数,r为线性基的个数。
103 * 所以对于每个数它们出现的次数一定相同,为2^(n-r)次。
104 */
105 void rebuild(int n){ /*线性基的重建,将上三角矩阵转化为对角矩阵 */
106     for(int i=60;i>=0;i--){
107         for(int j=i-1;j>=0;j--){
108             if(a[i]&(1LL<<j)) a[i]^=a[j];
109         }
110     }cnt=0;
111     for(int i=0;i<=60;i++){
112         if(a[i]) p[cnt++]=i;
113     }
114 }
115 int query(int q){
116     int ans=0;
117     for(int i=0;i<cnt;i++) if(q&(1LL<<p[i])) ans=(ans+(1<<i));
118     return ans;
119 }

```

6.12 java 大数开方

```

1 import java.math.BigInteger;
2
3 public class Main {

```

```

4    static BigInteger n,mod;
5    public static BigInteger Sqrt(BigInteger c) {
6        if(c.compareTo(BigInteger.ONE)<=0)
7            return c;
8        BigInteger temp=null,x;
9        x=c.shiftRight((c.bitLength()+1)/2);
10       while(true) {
11           temp=x;
12           x=x.add(c.divide(x)).shiftRight(1);
13           if(temp.equals(x)||x.add(BigInteger.ONE).equals(temp)) break;
14       }
15       return x;
16   }
17   public static boolean judge(BigInteger c) {
18       BigInteger x=Sqrt(c);
19       if(x.multiply(x).equals(c)) {
20           return true;
21       } else {
22           return false;
23       }
24   }
25 }

```

6.13 单纯形法

```

1  // 单纯形解线性规划 by zimpha
2  // 给出m个这样的约束条件:  $\sum(A[i]*X[i]) \leq B$ 
3  // 求出X的解, 在满足 $X[i] \geq 0$ 的情况下,  $\sum(C[i]*X[i])$ 达到最大
4  #include <cstdio>
5  #include <cstring>
6  #include <algorithm>
7  #define fo(i,a,b) for(int i=a;i<=b;i++)
8  using namespace std;
9  typedef long double db;
10 const int N=25;
11 db a[N][N], eps=1e-9;
12 int id[N*2],n,m,t,x;
13 double ans[N*2];
14 bool pd;
15 db abs(db x) {return x<0?-x:x;}
16 void pivot(int l,int e) {
17     swap(id[n+1],id[e]);
18     db t=a[l][e];a[l][e]=1;
19     fo(i,0,n) a[l][i]/=t;
20     fo(i,0,m)
21         if (i!=l&&abs(a[i][e])>eps) {
22             db t=a[i][e];a[i][e]=0;
23             fo(j,0,n) a[i][j]-=t*a[l][j];
24         }
25 }
26 void prepare() {
27     while (1) {

```

```

28     int l=0,e=0;
29     fo(i,1,m) if (a[i][0]<-eps&&(!l||(rand()&1))) l=i;
30     if (!l) break;
31     fo(i,1,n) if (a[l][i]<-eps&&(!e||(rand()&1))) e=i;
32     if (!e) {pd=1;return;}
33     pivot(l,e);
34 }
35 }
36 void solve() {
37     while (1) {
38         int l=0,e=0;db mn=1e18;
39         fo(i,1,n) if (a[0][i]>eps) {e=i;break;}
40         if (!e) break;
41         fo(i,1,m)
42             if (a[i][e]>eps&&a[i][0]/a[i][e]<mn) {
43                 mn=a[i][0]/a[i][e];
44                 l=i;
45             }
46         if (!l) {pd=1;return;}
47         pivot(l,e);
48     }
49 }
50 int main() {
51     srand(233);
52     scanf("%d%d%d",&n,&m,&t);
53     fo(i,1,n) scanf("%d",&x),a[0][i]=x;
54     fo(i,1,m) {
55         fo(j,1,n) scanf("%d",&x),a[i][j]=x;
56         scanf("%d",&x);
57         a[i][0]=x;
58     }
59     fo(i,1,n+m) id[i]=i;
60     prepare();
61     if (pd) { //不存在满足所有约束的解
62         printf("Infeasible\n");
63         return 0;
64     }
65     pd=0;
66     solve();
67     if (pd) { //对于任意的 M, 都存在一组解使得目标函数的值大于 M
68         printf("Unbounded\n");
69         return 0;
70     }
71     printf("%.15lf\n",-(double)a[0][0]);
72     if (t) {
73         fo(i,1,m) ans[id[i+n]]=a[i][0];
74         fo(i,1,n) printf("%.15lf",ans[i]);
75     }
76 }

```

6.14 容斥

```

1  /*
2  * Author: Simon
3  * 复杂度:  $O(2^k + \sqrt{n})$  其中  $k$  为  $n$  中不同的质因数的个数,  $n \leq 10^5$  时,  $k$  最大为 5
4  * 功能: 容斥求  $[1, n]$  中与  $p$  互质的数的和
5  */
6  /*求  $n$  的质因数*/
7  void solve(int n){ /*需预处理欧拉筛*/
8      fac.clear();
9      for(int i=1; i<=cnt&&prime[i]*prime[i]<=n; i++){
10         if(n%prime[i]==0){
11             fac.push_back(prime[i]);
12             while(n%prime[i]==0) n/=prime[i];
13         }
14     }
15     if(n>1) fac.push_back(n);
16 }
17 /*求和公式*/
18 int sum_1(int n){
19     return n*(n+1)/2;
20 }
21 /*容斥求  $[1, n]$  中与  $p$  互质的数的和*/
22 int cal(int n, int p){
23     solve(p); /*求出  $p$  的质因数*/
24     int ans=0;
25     for(int i=1; i<(1<<fac.size()); i++){ /*枚举子集*/
26         int num=0, lcm=1;
27         for(int j=0; j<fac.size(); j++){
28             if(i>>j&1){
29                 num++; lcm*=fac[j];
30             }
31         }
32         if(num&1) ans+=sum_1(n)-sum_1(n/lcm)*lcm; /*总和减去不互质的数的和*/
33         else ans-=sum_1(n)-sum_1(n/lcm)*lcm;
34     }
35     return ans;
36 }

```

6.15 拉格朗日插值

```

1  /*
2  * Author: Simon
3  * 复杂度:  $O(n)$ 
4  * 功能: 已知多项式前  $n+1$  项, 求第  $k$  项。
5  *  $n$  次多项式的前缀和是  $n+1$  次多项式。
6  */
7  int inv[maxn] /*阶乘逆元*/, bit[maxn] /*阶乘*/;
8  int ubit[maxn], subit[maxn];
9  void init(){
10     bit[0] = 1;
11     for (int i = 1; i < maxn; i++) bit[i] = 1LL * bit[i - 1] * i % mod;
12     inv[maxn - 1] = fpow(bit[maxn - 1], mod - 2, mod);
13     for (int i = maxn - 1; i >= 1; i--) inv[i - 1] = 1LL * inv[i] * i % mod;

```

```

14 }
15 int Lagrangian(int y[]/*值域*/,int n/*变量*/,int k/*待求y(k)*/) {
16     if(k<=n) return y[k];
17     ubit[0]=subit[n]=1;
18     for(int i=1;i<=n;i++){
19         ubit[i]=1LL*ubit[i-1]*((k-i+1))%mod; /* ubit[i]=prod_{j\in [0,i-1]}(n-j) */
20         subit[n-i]=1LL*subit[n-i+1]*((k-n+i-1))%mod; /*subit[i]=prod_{j\in [i+1,n]}(n-j) */
21     }
22     int ans=0;
23     for(int i=0;i<=n;i++){
24         int s1=1LL*y[i]%mod*ubit[i]%mod*subit[i]%mod; /*分子*/
25         int s2=1LL*inv[i]%mod*inv[n-i]%mod; /*分母 */
26         ans=(1LL*ans+1LL*((n-i)&1?-1:1)*s1%mod*s2%mod)%mod;
27     }
28     return (ans+mod)%mod;
29 }
30 /*
31  * Author: Simon
32  * 复杂度: O(n^2)
33  * 功能: 已知多项式任意n项, 求第k项
34  */
35 int Lagrangian(int y[]/*值域*/,int x[]/*变量*/,int k/*待求y_k*/,int mod){
36     int ans=0;
37     for(int i=1;i<=n;i++){
38         int s1=y[i]%mod,s2=1;
39         for(int j=1;j<=n;j++){
40             if(i==j) continue;
41             (s1*=((k-x[j])%mod))%=mod;
42             (s2*=((x[i]-x[j])%mod))%=mod;
43         }
44         (ans+=s1*fpow(s2,mod-2,mod)%mod)%=mod;
45     }
46     return (ans+mod)%mod;
47 }

```

6.16 多项式乘法/平方/取模

```

1 namespace fft {
2     typedef int type;
3     typedef double db;
4     struct cp {
5         db x, y;
6
7         cp() { x = y = 0; }
8
9         cp(db x, db y) : x(x), y(y) {}
10 };
11 inline cp operator+(cp a, cp b) { return cp(a.x + b.x, a.y + b.y); }
12 inline cp operator-(cp a, cp b) { return cp(a.x - b.x, a.y - b.y); }
13 inline cp operator*(cp a, cp b) { return cp(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
14 inline cp conj(cp a) { return cp(a.x, -a.y); }

```

```

15
16     type base = 1;
17     vector<cp> roots = {{0, 0},
18                        {1, 0}};
19     vector<type> rev = {0, 1};
20     const db PI = acosl(-1.0);
21     void ensure_base(type nbase) {
22         if (nbase <= base) {
23             return;
24         }
25         rev.resize(static_cast<unsigned long>(1 << nbase));
26         for (type i = 0; i < (1 << nbase); i++) {
27             rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
28         }
29         roots.resize(static_cast<unsigned long>(1 << nbase));
30         while (base < nbase) {
31             db angle = 2 * PI / (1 << (base + 1));
32             for (type i = 1 << (base - 1); i < (1 << base); i++) {
33                 roots[i << 1] = roots[i];
34                 db angle_i = angle * (2 * i + 1 - (1 << base));
35                 roots[(i << 1) + 1] = cp(cos(angle_i), sin(angle_i));
36             }
37             base++;
38         }
39     }
40     void fft(vector<cp> &a, type n = -1) {
41         if (n == -1) {
42             n = a.size();
43         }
44         assert((n & (n - 1)) == 0);
45         type zeros = __builtin_ctz(n);
46         ensure_base(zeros);
47         type shift = base - zeros;
48         for (type i = 0; i < n; i++) {
49             if (i < (rev[i] >> shift)) {
50                 swap(a[i], a[rev[i] >> shift]);
51             }
52         }
53         for (type k = 1; k < n; k <= 1) {
54             for (type i = 0; i < n; i += 2 * k) {
55                 for (type j = 0; j < k; j++) {
56                     cp z = a[i + j + k] * roots[j + k];
57                     a[i + j + k] = a[i + j] - z;
58                     a[i + j] = a[i + j] + z;
59                 }
60             }
61         }
62     }
63     vector<cp> fa, fb;
64     vector<type> multiply(vector<type> &a, vector<type> &b) {
65         type need = a.size() + b.size() - 1;
66         type nbase = 0;
67         while ((1 << nbase) < need) nbase++;
68         ensure_base(nbase);

```



```

69     type sz = 1 << nbase;
70     if (sz > (type) fa.size())
71         fa.resize(static_cast<unsigned long>(sz));
72     for (type i = 0; i < sz; i++) {
73         type x = (i < (type) a.size() ? a[i] : 0);
74         type y = (i < (type) b.size() ? b[i] : 0);
75         fa[i] = cp(x, y);
76     }
77     fft(fa, sz);
78     cp r(0, -0.25 / sz);
79     for (type i = 0; i <= (sz >> 1); i++) {
80         type j = (sz - i) & (sz - 1);
81         cp z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
82         if (i != j) {
83             fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
84         }
85         fa[i] = z;
86     }
87     fft(fa, sz);
88     vector<type> res(static_cast<unsigned long>(need));
89     for (type i = 0; i < need; i++) {
90         res[i] = fa[i].x + 0.5;
91     }
92     return res;
93 }
94 vector<type> multiply_mod(vector<type> &a, vector<type> &b, type m, type eq = 0) {
95     type need = a.size() + b.size() - 1;
96     type nbase = 0;
97     while ((1 << nbase) < need) nbase++;
98     ensure_base(nbase);
99     type sz = 1 << nbase;
100    if (sz > (type) fa.size()) {
101        fa.resize(static_cast<unsigned long>(sz));
102    }
103    for (type i = 0; i < (type) a.size(); i++) {
104        type x = (a[i] % m + m) % m;
105        fa[i] = cp(x & ((1 << 15) - 1), x >> 15);
106    }
107    fill(fa.begin() + a.size(), fa.begin() + sz, cp{0, 0});
108    fft(fa, sz);
109    if (sz > (type) fb.size()) {
110        fb.resize(static_cast<unsigned long>(sz));
111    }
112    if (eq) {
113        copy(fa.begin(), fa.begin() + sz, fb.begin());
114    } else {
115        for (type i = 0; i < (type) b.size(); i++) {
116            type x = (b[i] % m + m) % m;
117            fb[i] = cp(x & ((1 << 15) - 1), x >> 15);
118        }
119        fill(fb.begin() + b.size(), fb.begin() + sz, cp{0, 0});
120        fft(fb, sz);
121    }
122    db ratio = 0.25 / sz;

```

```

123     cp r2(0, -1);
124     cp r3(ratio, 0);
125     cp r4(0, -ratio);
126     cp r5(0, 1);
127     for (type i = 0; i <= (sz >> 1); i++) {
128         type j = (sz - i) & (sz - 1);
129         cp a1 = (fa[i] + conj(fa[j]));
130         cp a2 = (fa[i] - conj(fa[j])) * r2;
131         cp b1 = (fb[i] + conj(fb[j])) * r3;
132         cp b2 = (fb[i] - conj(fb[j])) * r4;
133         if (i != j) {
134             cp c1 = (fa[j] + conj(fa[i]));
135             cp c2 = (fa[j] - conj(fa[i])) * r2;
136             cp d1 = (fb[j] + conj(fb[i])) * r3;
137             cp d2 = (fb[j] - conj(fb[i])) * r4;
138             fa[i] = c1 * d1 + c2 * d2 * r5;
139             fb[i] = c1 * d2 + c2 * d1;
140         }
141         fa[j] = a1 * b1 + a2 * b2 * r5;
142         fb[j] = a1 * b2 + a2 * b1;
143     }
144     fft(fa, sz);
145     fft(fb, sz);
146     vector<type> res(static_cast<unsigned long>(need));
147     for (type i = 0; i < need; i++) {
148         long long aa = fa[i].x + 0.5;
149         long long bb = fb[i].x + 0.5;
150         long long cc = fa[i].y + 0.5;
151         res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
152     }
153     return res;
154 }
155 vector<type> square(vector<type> &a) {
156     return multiply(a, a);
157 }
158 vector<type> square_mod(vector<type> &a, type m) {
159     return multiply_mod(a, a, m, 1);
160 }
161 vector<type> kiss_me(vector<type>&b, long long k, type mod) {
162     vector<type> a = b;
163     vector<type> res(1, 1);
164     for (; k; k >>= 1, a = square_mod(a, mod)) {
165         if (k & 1) {
166             res = multiply_mod(res, a, mod);
167         }
168     }
169     return res;
170 }
171 pair<vector<type>, vector<type> > mul2(vector<type>&b, long long k) {
172     return make_pair(kiss_me(b, k, (type)1e9 + 7), kiss_me(b, k, (type)1e9 + 9));
173 }
174 vector<vector<type> > muln(vector<type>&b, long long k, vector<int> mod_list) {
175     vector< vector<type> > res(mod_list.size());
176     for (int i = 0; i < mod_list.size(); ++i) {

```

```

177         res[i] = kiss_me(b, k, mod_list[i]);
178     }
179     return res;
180 }
181 };

```

6.17 快速傅里叶变换

```

1  const double pi = acos(-1.0);
2  struct Complex { /*复数结构体*/
3      double x, y; /*实部和虚部 x+yi*/
4      Complex(double _x = 0.0, double _y = 0.0) { x = _x, y = _y; }
5      Complex operator-(const Complex& b) const { return Complex(x - b.x, y - b.y); }
6      Complex operator+(const Complex& b) const { return Complex(x + b.x, y + b.y); }
7      Complex operator*(const Complex& b) const { return Complex(x * b.x - y * b.y, x * b.y + y
          * b.x); }
8  };
9  /*
10 * 进行FFT和IFFT前的反转变换。
11 * 位置i和 (i二进制反转后位置) 互换
12 * len必须取2的幂
13 */
14 void change(Complex y[], int len) {
15     for (int i = 1, j = len >> 1; i < len - 1; i++) { /*i做正常的+1, j做反转类型的+1,始终保持i和
        j是反转的*/
16         if (i < j) std::swap(y[i], y[j]); /*交换互为小标反转的元素, i<j保证交换一次*/
17         int k = len >> 1;
18         while (j >= k) j -= k, k >>= 1;
19         if (j < k) j += k;
20     }
21 }
22
23 /*
24 * 做FFT
25 * len必须为2^k形式, 且不小于最高次幂。
26 * on==1时是DFT, on==-1时是IDFT
27 * DFT是将多项式系数表达转化为点值表达
28 * 复杂度O(nLog(n))
29 */
30 void fft(Complex y[], int len, int on) {
31     change(y, len); /*预处理分治区间长度为1时的位置顺序*/
32     for (int mid = 1; mid < len; mid <= 1) { /*当前分治区间的长度的一半*/
33         Complex Wn(cos(pi / mid), on * sin(pi / mid)); /*mid*2等分点, 单位根*/
34         for (int R = mid << 1, j = 0; j < len; j += R) { /*R是区间长度, j表示第j/R+1个区间的起
            始下标*/
35             Complex w(1, 0); /*幂*/
36             for (int k = 0; k < mid; k++, w = w * Wn) { /*枚举左半部分*/
37                 Complex u = y[j + k], v = w * y[j + mid + k]; /*蝴蝶效应*/
38                 y[j + k] = u + v;
39                 y[j + mid + k] = u - v;
40             }
41         }

```

```

42     }
43     if(on== -1) for(int i=0;i<len;i++) y[i].x/=len;
44 }

```

6.18 快速数论变换

```

1 // ——
2 // 模数 $p$ 为费马素数,  $g$ 为 $p$ 的原根。
3 //  $g^{\frac{p-1}{n}}$ 具有和 $w_n=e^{\frac{2i\pi}{n}}$ 相似的性质。
4 // 具体的 $p$ 和 $g$ 可参考1.11
5 // ——
6 const int mod = 119 << 23 | 1;
7 const int g = 3;
8 int wn[23];
9 void getwn() {
10     for (int i = 0; i < 20; i++) wn[i] = fpow(g, (mod - 1) / (1 << i), mod);
11 }
12 void change(int y[],int len){
13     for(int i=1,j=len>>1;i<len-1;i++){
14         if(i<j) swap(y[i],y[j]);
15         int k=len>>1;
16         while(j>=k) j-=k,k>>=1;
17         if(j<k) j+=k;
18     }
19 }
20 void ntt(int y[],int len,int on,int p){
21     change(y,len);
22     for(int mid=1;mid<len;mid<=1){
23         int wn=fpow(g,(p-1)/(mid<<1),p);
24         for(int R=mid<<1,j=0;j<len;j+=R){
25             int w=1;
26             for(int k=0;k<mid;k++,w=w*wn%p){
27                 int u=y[j+k],v=w*y[j+mid+k]%p;
28                 y[j+k]=(u+v)%p;
29                 y[j+mid+k]=(u-v+p)%p;
30             }
31         }
32     }
33     if(on== -1) {
34         int inv = fpow(len, p - 2, p);
35         for(int i=1;i<len/2;i++) swap(y[i],y[len-i]);
36         for(int i=0;i<len;i++) (y[i]*=inv)%=mod;
37     }
38 }

```

6.19 快速沃尔什变换

```

1 void fwt(int f[], int m) {
2     int n = __builtin_ctz(m);

```

```

3   for (int i = 0; i < n; ++i)
4       for (int j = 0; j < m; ++j)
5           if (j & (1 << i)) {
6               int l = f[j ^ (1 << i)], r = f[j];
7               f[j ^ (1 << i)] = l + r, f[j] = l - r;
8               // or: f[j] += f[j ^ (1 << i)];
9               // and: f[j ^ (1 << i)] += f[j];
10          }
11 }
12
13 void ifwt(int f[], int m) {
14     int n = __builtin_ctz(m);
15     for (int i = 0; i < n; ++i)
16         for (int j = 0; j < m; ++j)
17             if (j & (1 << i)) {
18                 int l = f[j ^ (1 << i)], r = f[j];
19                 f[j ^ (1 << i)] = (l + r) / 2, f[j] = (l - r) / 2;
20                 // 如果有取模需要使用逆元
21                 // or: f[j] -= f[j ^ (1 << i)];
22                 // and: f[j ^ (1 << i)] -= f[j];
23             }
24 }

```

6.20 分治 fft

```

1 //dp[i] = sigma(a[j] * dp[i-j]) (j < i);
2 const int maxn = "Edit";
3 int dp[maxn], a[maxn];
4 Complex x[maxn<<2], y[maxn<<2];
5 void solve(int L, int R){
6     if(L == R) return ;
7     int mid = (L + R) >> 1;
8     solve(L, mid);
9     int len = 1, len1 = R - L + 1;
10    while(len <= len1) len <<= 1;
11    for(int i = 0; i < len1; ++i) x[i] = Complex(a[i], 0);
12    for(int i = len1; i <= len; ++i) x[i] = Complex(0, 0);
13    for(int i = L; i <= mid; ++i)
14        y[i-L] = Complex(dp[i], 0);
15    for(int i = mid - L + 1; i <= len; ++i) y[i] = Complex(0, 0);
16    fft(x, len, 1);
17    fft(y, len, 1);
18    for(int i = 0; i < len; ++i) x[i] = x[i] * y[i];
19    fft(x, len, -1);
20    for(int i = mid + 1; i <= R; ++i){
21        dp[i] += x[i-L].x + 0.5;
22    }
23    solve(mid + 1, R);
24 }

```

6.21 polya 项链染色

```

1  /*
2   * c种颜色的珠子，组成长为s的项链，项链没有方向和起始位置
3   */
4  int gcd(int a, int b) {
5      return b ? gcd(b, a % b) : a;
6  }
7
8  int main(int argc, const char * argv[]) {
9      int c, s;
10     while (std::cin >> c >> s) {
11         int k;
12         long long p[64];
13         p[0] = 1; // power of c
14         for (k = 0; k < s; k++) {
15             p[k + 1] = p[k] * c;
16         }
17         // reflection part
18         long long count = s & 1 ? s * p[s / 2 + 1] : (s / 2) * (p[s / 2] + p[s / 2 + 1]);
19         // rotation part
20         for (k = 1; k <= s; k++) {
21             count += p[gcd(k, s)];
22             count /= 2 * s;
23         }
24         std::cout << count << '\n';
25     }
26     return 0;
27 }

```

6.22 染色多项式

```

1 完全图:  $t(t-1)(t-2)\dots(t-(n-1))$ 
2 有n个顶点的树:  $t(t-1)^{(n-1)}$ 
3 环:  $(t-1)^n + (-1)^n(t-1)$ 
4 加边减边:  $P(G)=P(G+e)+P(G\bullet e)$ 

```

6.23 错位排列递推公式

```

1 d[1] = 0;
2 d[2] = 1;
3 for (int i = 3; i < maxn; i++) {
4     d[i] = (i-1)*(d[i-1]+d[i-2]);
5 }

```

6.24 BBP 公式求 pi 十六进制的第 k 位

```

1 // BBP算法 询问十六进制下圆周率的第n位
2 // 时间复杂度  $O(n \log n)$ 
3
4 using ll = long long;
5
6 ll remain( ll m, ll n, ll k, ll extra) {
7     ll temp1=1,temp2=1;
8     if (n==0) return extra%k;
9     if (n==1) return (m*extra)%k;
10    while(n>1) {
11        temp1=m;
12        temp1*=temp1;
13        if(temp1>=k)temp1%=k;
14        if(n%2==1)temp2=m*temp2;
15        temp2%=k;
16        m=temp1;
17        n=n/2;
18    }
19    temp1=(temp1*temp2)%k;
20    return (temp1*extra)%k;
21 }
22
23 ll remain_nex( ll m, ll n, ll k) {
24     ll temp1 = 1, temp2 = 1;
25     if (n == 0) return 1;
26     if (n == 1) return m%k;
27     while (n>1) {
28         temp1 = m;
29         temp1 *= temp1;
30         if (temp1 >= k) temp1%=k;
31         if (n % 2 == 1) temp2=m*temp2;
32         temp2 %= k;
33         m = temp1;
34         n = n / 2;
35     }
36     return (temp1*temp2)%k;
37 }
38
39 char compute_n(int j) {
40     ll m;
41     long double sum=0,temp=1.0,temp1;
42     int i;
43     j--;
44     temp1=1.0;
45     for (i=0;i<=j;i++) sum=sum+remain(16,j-i,8*i+1,4)/(long double)(8.0*i+1);
46     for (i=0;i<=j;i++) sum=sum-remain(16,j-i,8*i+4,2)/(long double)(8.0*i+4);
47     for (i=0;i<=j;i++) sum=sum-remain_nex(16,j-i,8*i+5)/(long double)(8.0*i+5);
48     for (i=0;i<=j;i++) sum=sum-remain_nex(16,j-i,8*i+6)/(long double)(8.0*i+6);
49     temp=1.0;
50     for (;temp>0.000001;i++) {
51         temp=temp/16.0;sum=sum+(4.0/(8*i+1)-2.0/(8*i+4)-1.0/(8*i+5)-1.0/(8*i+6))*temp;
52     }
53     for (;sum<0;) sum=sum+16;
54     m=sum;

```

```
55     sum=sum-m;
56     sum=sum*16;
57     m=sum;
58     return (char)(m<10 ? m+48: m+55);
59 }
```

第七章 图论

7.1 前向星

```
1  const int maxn = 10005;    //点的最大个数
2
3  int head[maxn], cnt=0; //head用来表示以i为起点的第一条边存储的位置, cnt读入边的计数器
4
5  struct Edge {
6      int next; //同一起点的上一条边的储存位置
7      int to;   //第i条边的终点
8      int w;    //第i条边权重
9  };
10
11  Edge edge[maxn];
12
13  void addedge(int u, int v, int w) {
14      edge[cnt].w = w;
15      edge[cnt].to = v;
16      edge[cnt].next = head[u];
17      head[u] = cnt++;
18  }
19
20  void traverse() {
21      for(int i=0; i<=n; i++) {
22          for(int j=head[i]; j!=-1; j=edge[j].next) {
23              std::cout << i << " " << head[i].to << " " << head[i].w << '\n';
24          }
25      }
26  }
```

7.2 并查集

```
1  int fa[N];
2
3  void init(int n) {
4      for (int i = 1; i <= n; i++) fa[i] = i;
5  }
6
7  int find(int u) {
8      return fa[u] == u ? fa[u] : fa[u] = find(fa[u]);
9  }
```

```
10
11 void unin(int u, int v) {
12     fa[find(v)] = find(u);
13 }
```

7.3 可撤销并查集（按秩合并）

```
1  #include <iostream>
2  #include <stack>
3  #include <utility>
4
5  class UFS {
6      private:
7          int *fa, *rank;
8          std::stack <std::pair <int*, int> > stk ;
9      public:
10         UFS() {}
11         UFS(int n) {
12             fa = new int[(const int)n + 1];
13             rank = new int[(const int)n + 1];
14             memset (rank, 0, n+1);
15             for (int i = 1; i <= n; ++i) {
16                 fa [i] = i;
17             }
18         }
19         inline int find(int x) {
20             while (x ^ fa[x]) {
21                 x = fa[x];
22             }
23             return x ;
24         }
25         inline int Join (int x, int y) {
26             x = find(x), y = find(y);
27             if (x == y) {
28                 return 0;
29             }
30             if (rank[x] <= rank[y]) {
31                 stk.push(std::make_pair (fa + x, fa[x]));
32                 fa[x] = y;
33                 if (rank[x] == rank[y]) {
34                     stk.push(std::make_pair (rank + y, rank[y]));
35                     ++rank[y];
36                     return 2;
37                 }
38                 return 1 ;
39             }
40             stk.push(std::make_pair(fa + y, fa [y]));
41             return fa[y] = x, 1;
42         }
43         inline void Undo ( ) {
44             *stk.top( ).first = stk.top( ).second ;
45             stk.pop( ) ;
```

```

46     }
47 }T;

```

7.4 Kruskal 最小生成树

```

1  /*
2  * Author: Simon
3  * 复杂度:  $m\log(m)$ 
4  * 功能: 适用于稀疏图求MST
5  */
6  namespace Kruskal{
7      int Set[maxn], Rank[maxn], cnt=0; //并查集 rank
8      struct node{
9          int u,v,w;
10         node(){}
11         node(int u,int v,int w):u(u),v(v),w(w){}
12         bool operator <(const node&a)const{
13             return w<a.w;
14         }
15     }g[maxn]; //结构体储存
16     void init(int n) {
17         for (int i = 0; i <= n; i++) {
18             Rank[i] = 0; Set[i] = i;
19         }
20     }
21     void addedge(int u,int v,int w){
22         g[cnt++]=node(u,v,w);
23     }
24     int find(int x) {
25         int root = x;
26         while (root != Set[root]) root = Set[root];
27         while (x != root) {
28             int t = Set[x];
29             Set[x] = root;
30             x = t;
31         }
32         return root;
33     }
34     void unite(int x, int y) {
35         x = find(x); y = find(y);
36         if (Rank[x] < Rank[y]) {
37             Set[x] = y;
38         } else {
39             Set[y] = x;
40             if (Rank[x] == Rank[y]) Rank[x]++;
41         }
42     }
43     //n为边的数量, m为村庄的数量
44     int Run(int n, int m) {
45         int num = 0, res = 0; //将边按照权值从小到大排序
46         sort(g,g+cnt);
47         for (int i = 0; i < n && num != m - 1; i++) {

```

```

48         int u=g[i].u,v=g[i].v,w=g[i].w;
49         //判断当前这条边的两个端点是否属于同一棵树
50         if (find(u) != find(v)) {
51             unite(u, v);
52             res += w;
53             num++;
54         }
55     }
56     //如果加入边的数量小于 $m - 1$ ,则表明该无向图不连通,等价于不存在最小生成树
57     if (num < m - 1) res = -1;
58     return res;
59 }
60 }

```

7.5 Prim 最小生成树

```

1  /*
2   * Author: Simon
3   * 复杂度:  $n\log(m)$ 
4   * 功能: 适用于求稠密图求MST
5   */
6  namespace Prim{
7      int head[maxn],cnt=0;
8      bool vis[maxn];
9      struct node{
10         int v,w,next;
11         node(){}
12         node(int v,int w,int next):v(v),w(w),next(next){}
13         bool operator <(const node a)const{
14             return w>a.w;
15         }
16     }g[maxn];
17     void init(){
18         memset(vis,0,sizeof(vis));
19         memset(head,-1,sizeof(head));cnt=0;
20     }
21     void addedge(int u,int v,int w){
22         g[cnt]=node(v,w,head[u]);
23         head[u]=cnt++;
24     }
25     int Run(int s,int n){
26         priority_queue<node>q;
27         q.push(node(s,0));int ans=0;
28         while(!q.empty()){
29             node now=q.top();q.pop();
30             if(vis[now.v]) continue;
31             vis[now.v]=1;ans+=now.w;
32             for(int i=head[now.v];~i;i=g[i].next){
33                 if(!vis[g[i].v]) q.push(g[i]);
34             }
35         }
36         return ans;

```

```

37     }
38 }

```

7.6 SPFA 最短路

```

1  /*
2  * Author: Simon
3  * 复杂度:  $O(ne)$ 
4  */
5  bool SPFA(int n,int r){
6      memset(dis,INF,sizeof(dis));dis[r]=0;
7      memset(vis,0,sizeof(vis));vis[r]=1;
8      memset(ccnt,0,sizeof(ccnt));ccnt[r]=1;
9      queue<int>q;q.push(r);
10     while(!q.empty()){
11         int u=q.front();q.pop();vis[u]=0;
12         for(int i=head[u];~i;i=g[i].next){
13             int &v=g[i].v,&w=g[i].w;
14             if(dis[u]+w<dis[v]){
15                 dis[v]=dis[u]+w;
16                 if(!vis[v]){
17                     q.push(v);vis[v]=1;
18                     if(++ccnt[v]>n) return 1;
19                 }
20             }
21         }
22     }
23     return 0;
24 }

```

7.7 dijkstra 最短路

```

1  /*
2  * Author: Simon
3  * 复杂度:  $O(e \cdot \log(n))$ 
4  */
5  namespace Dijkstra{
6      struct node{
7          int v,w,next;
8          node(){}
9          node(int v,int w,int next=-1):v(v),w(w),next(next){}
10         bool operator <(const node&a)const{
11             return w>a.w;
12         }
13     }g[maxn];
14     int head[maxn],cnt=0,dis[maxn];
15     bool vis[maxn];
16     void addedge(int u,int v,int w){
17         g[cnt]=node(v,w,head[u]);

```

```

18     head[u]=cnt++;
19 }
20 void init(){
21     memset(head,-1,sizeof(head)); cnt=0;
22 }
23 void Run(int n,int r){
24     fill(dis,dis+n+5,INF);dis[r]=0;
25     memset(vis,0,sizeof(vis));
26     priority_queue<node>q;
27     q.push({r,0});
28     while(!q.empty()){
29         node now=q.top();q.pop();
30         int &u=now.v;
31         if(vis[u]) continue; vis[u]=1;
32         for(int i=head[u];~i;i=g[i].next){
33             int &v=g[i].v,&w=g[i].w;
34             if(!vis[v]&&dis[u]+w<dis[v]){
35                 dis[v]=dis[u]+w;
36                 q.push({v,dis[v]});
37             }
38         }
39     }
40 }
41 }

```

7.8 Floyd 任意两点间最短路

```

1  // #define inf maxn*maxw+10
2  for(int i = 0; i < n; i++) {
3      for(int j = 0; j < n; j++) {
4          d[i][j] = inf;
5      }
6  }
7  d[0][0] = 0;
8  for(int k = 0; k < n; k++) {
9      for(int i = 0; i < n; i++) {
10         for(int j = 0; j < n; j++) {
11             d[i][j] = std::min(d[i][j], d[i][k] + d[k][j]);
12         }
13     }
14 }
15 /*
16  * Author: Simon
17  * 功能：传递闭包
18  */
19 for(int k=1;k<=n;k++){
20     for(int i=1;i<=n;i++){
21         for(int j=1;j<=n;j++){
22             if(a[i][k]&&a[k][j]) a[i][j]=1; /*传递闭包 */
23         }
24     }
25 }

```

7.9 2-SAT 问题

```
1  class TwoSAT{
2      private:
3          const static int maxm=maxn*2;
4
5          int S[maxm],c;
6          vector<int> G[maxm];
7
8          bool DFS(int u){
9              if(vis[u^1])
10                 return false;
11              if(vis[u])
12                 return true;
13              vis[u]=1;
14              S[c++]=u;
15              for(auto &v:G[u])
16                 if(!DFS(v))
17                     return false;
18              return true;
19          }
20
21      public:
22          int N;
23          bool vis[maxm];
24
25          void Clear(){
26              for(int i=2;i<(N+1)*2;++i)
27                  G[i].clear();
28              memset(vis,0,sizeof(bool)*(N+1)*2);
29          }
30
31          void AddClause(int x,int xv,int y,int yv){
32              x=x*2+xv;
33              y=y*2+yv;
34              G[x].push_back(y);
35              G[y].push_back(x);
36          }
37
38          bool Solve(){
39              for(int i=2;i<(N+1)*2;i+=2)
40                  if(!vis[i]&&!vis[i+1]){
41                      c=0;
42                      if(!DFS(i)){
43                          while(c>0)
44                              vis[S[--c]]=0;
45                          if(!DFS(i+1))
46                              return false;
47                      }
48                  }
49              return true;
```

```

50     }
51 };

```

7.10 tarjan 强连通分量

```

1  /*
2  * Author: Simon
3  * 功能：求强连通分量
4  */
5  int dfn[maxn], low[maxn], Stack[maxn], Belong[maxn];
6  int index=0, scnt=0, bcnt=0;
7  void Tarjan(int u){
8      dfn[u]=low[u]=++index;
9      Stack[++scnt]=u; vis[u]=1;
10     for(int i=head[u]; ~i; i=g[i].next){
11         int &v=g[i].v;
12         if(!dfn[v]){
13             Tarjan(v);
14             low[u]=min(low[u], low[v]);
15         }
16         else if(vis[v]) low[u]=min(low[u], dfn[v]);
17     }
18     if(dfn[u]==low[u]){
19         bcnt++; int t;
20         do{
21             t=Stack[scnt--]; vis[t]=0;
22             Belong[t]=bcnt;
23         }while(t!=u);
24     }
25 }
26 void solve(int n){
27     memset(dfn, 0, sizeof(dfn));
28     index=bcnt=scnt=0;
29     for(int i=1; i<=n; i++){
30         if(!dfn[i]) Tarjan(i);
31     }
32 }
33 /*
34 * Author: Simon
35 * 功能：缩点建立新图
36 */
37 for(int u=1; u<=n; u++){
38     for(int i=head1[u]; ~i; i=g[i].next){
39         int &v=g[i].v;
40         if(Belong[u]!=Belong[v]){
41             addedge(g2, head2, cnt2, Belong[v], Belong[u]);
42             rdu[Belong[u]]++;
43         }
44     }
45 }
46 /*
47 * Author: Simon

```



```

48  * 功能：双联通算法
49  */
50  int dfn[maxn], low[maxn], Stack[maxn], Belong[maxn];
51  int index=0, scnt=0, bcnt=0;
52  int tarjan(int u, int fa)
53  {
54      dfn[u]=low[u]++index;
55      for(int i=head[u]; ~i; i=g[i].next){
56          int &v=g[i].v;
57          if(!dfn[v])
58          {
59              Stack[++scnt]=v; /*搜索到的点入栈 */
60              tarjan(edges[i].to, u);
61              low[u]=min(low[u], low[v]);
62              if(low[v]>=dfn[u]) /*是割点或根 */
63              {
64                  bcnt++; int t;
65                  do{
66                      t=Stack[scnt--];
67                      Belong[t]=bcnt;
68                  }while(t!=v);
69                  while(stack[top]!=edges[i].to) /*将点出栈直到目标点 */
70                      bcc[num].push_back(stack[top--]);
71                  bcc[num].push_back(stack[top--]); /*目标点出栈 */
72                  bcc[num].push_back(u); /*不要忘了将当前点存入bcc */
73              }
74          }
75          else if(edges[i].to!=fa)
76              lowu=min(lowu, dfn[edges[i].to]);
77      }
78  }

```

7.11 Kosaraju 强连通分量

```

1  #include <vector>
2  #include <algorithm>
3
4  const int maxn = "Edit";
5
6  class Kosaraju {
7  private:
8      std::vector<int> s[maxn], g[maxn];
9      bool vis[maxn]={0};
10 public:
11     int st[maxn], top=0, contract[maxn]={0};
12     int n, m;
13     void dfs(int x){
14         vis[x]=1;
15         for(int i=0; i<(int)s[x].size(); ++i){
16             if(!vis[s[x][i]]) dfs(s[x][i]);
17         }
18         st[top++]=x;

```

```

19     }
20     void dfs2(int x,int k){
21         if(contract[x])return;
22         contract[x]=k;/**x屬於第k個contract*/
23         for(int i=0;i<(int)g[x].size();++i){
24             dfs2(g[x][i],k);
25         }
26     }
27     void addedge(int a, int b) {
28         s[a].push_back(b);
29         g[b].push_back(a);
30     }
31     void kosaraju() {
32         for(int i=0;i<n;++i){
33             if(!vis[i]) {
34                 dfs(i);
35             }
36         }
37         for(int i=top-1,t=0;i>=0;--i){
38             if(!contract[st[i]]) {
39                 dfs2(st[i],++t);
40             }
41         }
42     }
43 };

```

7.12 点双联通分量

```

1  //Author: CookiC
2  #include <stack>
3  #include <vector>
4  #define maxn 1000
5  using namespace std;
6
7  class BCC{
8  private:
9      int clk, cnt;
10     int pre[maxn];
11     stack<int> s;
12
13     int DFS(int u,int f){
14         int lowu = pre[u] = ++clk;
15         int child = 0;
16         s.push(u);
17         for (auto it = G[u].begin(); it != G[u].end(); ++it){
18             int v = *it;
19             if (!pre[v]){
20                 s.push(u);
21                 ++child;
22                 int lowv = DFS(v, u);
23                 if (lowu > lowv)
24                     lowu = lowv;

```

```
25         if (lowv >= pre[u]){
26             iscut[u] = 1;
27             ++cnt;
28             int x;
29             do{
30                 x = s.top();
31                 s.pop();
32                 if (num[x] != cnt)
33                     num[x] = cnt;
34             }while (x != u);
35         }
36     }
37     else if (pre[v] < pre[u] && v != f){
38         if (lowu > pre[v])
39             lowu = pre[v];
40     }
41 }
42 if (f < 0 && child == 1)
43     iscut[u] = 0;
44 return lowu;
45 }
46 public:
47     vector<int> G[maxn];
48     bool iscut[maxn];
49     int num[maxn];
50
51     void Clear(int n){
52         for (int i = 0; i < n; ++i)
53             G[i].clear();
54     }
55
56     void AddEdge(int u,int v){
57         G[u].push_back(v);
58         G[v].push_back(u);
59     }
60
61     void Find(){
62         int i;
63         memset(pre, 0, sizeof(pre));
64         memset(iscut, 0, sizeof(iscut));
65         memset(num,0,sizeof(num));
66         clk = cnt = 0;
67         for (i = 0; i < n; ++i)
68             if (!pre[i]){
69                 while(!s.empty())
70                     s.pop();
71                 DFS(i,-1);
72             }
73     }
74 };
```

7.13 边双联通分量

```
1 //Author: XieNaoban
2 //在求桥的基础上修改
3 #include <algorithm>
4 #include <cstring>
5 #include <vector>
6 #include <cmath>
7 #include <set>
8
9 class CutEdge {
10 private:
11     int N;
12     int clk, pre[Maxn];
13
14     int DFS(int u, int f) {
15         int lowu = pre[u] = ++clk;
16         for (auto e = G[u].begin(); e != G[u].end(); ++e) {
17             int v = *e;
18             if (!pre[v]) {
19                 int lowv = DFS(v, u);
20                 lowu = min(lowu, lowv);
21                 if (lowv > pre[u]) {
22                     Cut[u].insert(v);
23                     Cut[v].insert(u);
24                 }
25             }
26             else if (pre[u] > pre[v]) {
27                 int cnt = 0; //重复边的处理
28                 for (const auto &e : G[u]) if (e == v) ++cnt;
29                 if (cnt > 1 || v != f) {
30                     lowu = min(lowu, pre[v]);
31                 }
32             }
33         }
34         return lowu;
35     }
36
37     void DFS2(int u, int id) {
38         ID[u] = id;
39         for (const auto &v : G[u]) if (!ID[v]) {
40             if (Cut[u].count(v)) {
41                 ++Num;
42                 G2[id].push_back(Num);
43                 G2[Num].push_back(id);
44                 DFS2(v, Num);
45             }
46             else DFS2(v, id);
47         }
48     }
49
50 public:
51     vector<int> G[Maxn];
52     set<int> Cut[Maxn];
```

```

53
54     vector<int> G2[Maxn]; //缩点后的图 (以ID为结点)
55     int ID[Maxn]; //每个点所在的子图
56     int Num; //ID个数
57
58     void Clear(int n) {
59         N = n;
60         memset(ID, 0, sizeof(ID));
61         memset(pre, 0, sizeof(pre));
62         for (int i = 1; i <= N; ++i) {
63             G[i].clear();
64             G2[i].clear();
65             Cut[i].clear();
66         }
67         clk = 0;
68         Num = 1;
69     }
70
71     void AddEdge(int u, int v) {
72         G[u].push_back(v);
73         G[v].push_back(u);
74     }
75
76     void Find() {
77         for (int i = 1; i <= N; ++i)
78             if (!pre[i])
79                 DFS(i, -1);
80     }
81
82     //求边双联通部分
83     int BCC() { //要求先运行Find
84         DFS2(1, Num);
85         return Num;
86     }
87 };

```

7.14 求桥

```

1  class bcc_bridges {
2      public:
3          struct edge {
4              int y;
5              edge * next;
6          };
7          edge e[M], *li[N];
8          int etop;
9          void init() {
10             memset(li, 0, sizeof(li));
11             etop = 0;
12         }
13         inline void add_edge(int u, int v) {
14             e[etop].y = v;

```

```

15         e[etop].next = li[u];
16         li[u] = &e[etop++];
17     }
18     std::vector<std::pair<int, int>> briges;
19     int dfn[N], low[N];
20     int clk;
21     void dfs(int u, int fa) {
22         dfn[u] = low[u] = ++clk;
23         int v;
24         for (edge * t = li[u]; t; t = t->next) {
25             v = t->y;
26             if (!dfn[v]) {
27                 dfs(v, u);
28                 low[u] = std::min(low[u], low[v]);
29                 if (low[v] > dfn[u])
30                     briges.emplace_back(u, v); // u <-> v is a bridge
31             }
32             else if (dfn[v] < dfn[u] && v != fa)
33                 low[u] = std::min(low[u], dfn[v]);
34         }
35     }
36     void find_bridge(int n) {
37         clk = 0;
38         std::fill(dfn + 1, dfn + n + 1, 0);
39         std::fill(low, low + n + 1, 0);
40         for (int i = 1; i <= n; ++i) {
41             if (!dfn[i])
42                 dfs(i, 0);
43         }
44     }
45 };

```

7.15 欧拉回路

```

1  const int maxn = 100;
2
3  int n;
4  int step;
5  int path[maxn];
6
7  void find_path_u(int now, int mat[][maxn]) {
8      for (int i = n - 1; i >= 0; i--) {
9          while (mat[now][i]) {
10             mat[now][i]--, mat[i][now]--;
11             find_path_u(i, mat);
12         }
13     }
14     path[step++] = now;
15 }
16
17 void find_path_d(int now, int mat[][maxn]) {
18     for (int i = n - 1; i >= 0; i--) {

```

```

19     while (mat[now][i]) {
20         mat[now][i]--;
21         find_path_d(i, mat);
22     }
23 }
24 path[step++] = now;
25 }
26
27
28 int euler_circuit(int start, int mat[][maxn]) {
29     step = 0;
30     find_path_u(start, mat);
31     // find_path_d(start, mat);
32     return step;
33 }
34
35 int main() {
36
37 }

```

7.16 k 短路

```

1  #include <cstdio>
2  #include <cstring>
3  #include <queue>
4  #include <vector>
5  #include <algorithm>
6  using namespace std;
7
8  const int maxn = 10000 + 5;
9  const int INF = 0x3f3f3f3f;
10 int s, t, k;
11
12 bool vis[maxn];
13 int dist[maxn];
14
15 struct Node {
16     int v, c;
17     Node (int _v = 0, int _c = 0) : v(_v), c(_c) {}
18     bool operator < (const Node &rhs) const {
19         return c + dist[v] > rhs.c + dist[rhs.v];
20     }
21 };
22
23 struct Edge {
24     int v, cost;
25     Edge (int _v = 0, int _cost = 0) : v(_v), cost(_cost) {}
26 };
27
28 vector<Edge>E[maxn], revE[maxn];
29
30 void Dijkstra(int n, int s) {

```

```

31  memset(vis, false, sizeof(vis));
32  for (int i = 1; i <= n; i++) dist[i] = INF;
33  priority_queue<Node> que;
34  dist[s] = 0;
35  que.push(Node(s, 0));
36  while (!que.empty()) {
37      Node tep = que.top(); que.pop();
38      int u = tep.v;
39      if (vis[u]) continue;
40      vis[u] = true;
41      for (int i = 0; i < (int)E[u].size(); i++) {
42          int v = E[u][i].v;
43          int cost = E[u][i].cost;
44          if (!vis[v] && dist[v] > dist[u] + cost) {
45              dist[v] = dist[u] + cost;
46              que.push(Node(v, dist[v]));
47          }
48      }
49  }
50 }
51
52 int astar(int s) {
53     priority_queue<Node> que;
54     que.push(Node(s, 0)); k--;
55     while (!que.empty()) {
56         Node pre = que.top(); que.pop();
57         int u = pre.v;
58         if (u == t) {
59             if (k) k--;
60             else return pre.c;
61         }
62         for (int i = 0; i < (int)revE[u].size(); i++) {
63             int v = revE[u][i].v;
64             int c = revE[u][i].cost;
65             que.push(Node(v, pre.c + c));
66         }
67     }
68     return -1;
69 }
70
71 void addedge(int u, int v, int w) {
72     revE[u].push_back(Edge(v, w));
73     E[v].push_back(Edge(u, w));
74 }
75
76 int main() {
77     int n, m, u, v, w;
78     while (scanf("%d%d", &n, &m) != EOF) {
79         for (int i = 0; i <= n; i++) {
80             E[i].clear();
81             revE[i].clear();
82         }
83         int aaa;
84         scanf("%d%d%d%d", &s, &t, &k, &aaa);

```



```

85     for (int i = 0; i < m; i++) {
86         scanf("%d%d%d", &u, &v, &w);
87         addedge(u, v, w);
88     }
89     Dijkstra(n, t);
90     if (dist[s] == INF) {
91         printf("No Solution\n");
92         continue;
93     }
94     if (s == t) k++;
95     ans = astar(s);
96 }
97 return 0;
98 }

```

7.17 前 k 短路

```

1  /*
2   * Author: Simon
3   * 功能：前k短路，dfs+倍增+Dijkstra+二分
4   * 复杂度：O(Log(Maxval) * k * Log(n))
5   */
6  #include <bits/stdc++.h>
7  using namespace std;
8  typedef int Int;
9  #define int long long
10 #define maxn 10005
11 #define maxm 50005
12 #define maxc 1005
13 #define maxk 10005
14 #define maxv maxn*1LL*maxc*maxk
15 #define INF maxv
16 namespace Dijkstra{
17     struct node{
18         int v,w,next;
19         node(){}
20         node(int v,int w,int next=-1):v(v),w(w),next(next){}
21         bool operator <(const node&a)const{
22             return w>a.w;
23         }
24     }g[maxm];
25     int head[maxn],cnt=0,dis[maxn];
26     bool vis[maxn];
27     void init(){
28         memset(head,-1,sizeof(head));cnt=0;
29     }
30     void addedge(int u,int v,int w){
31         g[cnt]=node(v,w,head[u]);
32         head[u]=cnt++;
33     }
34     void run(int r){
35         memset(dis,INF,sizeof(dis));dis[r]=0;

```

```

36     memset(vis,0,sizeof(vis));
37     priority_queue<node>q;q.push({r,0});
38     while(!q.empty()){
39         node now=q.top();q.pop();
40         int u=now.v;
41         if(vis[u]) continue;vis[u]=1;
42         for(int i=head[u];~i;i=g[i].next){
43             int &v=g[i].v,&w=g[i].w;
44             if(!vis[v]&&dis[u]+w<dis[v]){
45                 dis[v]=dis[u]+w;
46                 q.push({v,dis[v]});
47             }
48         }
49     }
50 }
51 }
52 namespace kth{
53     int mid,num=0,ans[maxn];
54     /*
55     * 比较函数是为了把每个点的所有连出去的边按照  $v+dis[y]$  由小到大排序
56     * 其中  $v$  是该边的权重,  $dis[y]$  是与  $x$  相邻的点到终点的最短路径。
57     * 排完序后有两个好处:
58     * 1. 排序后每个点连出去的第一条边就当作最短路径上的边。
59     * 2. 依序枚举相邻边时, 当确定一条边若走上去后到达终点的距离会超过  $V$ ,
60     *    那之后的边也不必再枚举了
61     */
62     bool cmp(pair<int,int> a,pair<int,int> b){
63         return a.second+Dijkstra::dis[a.first]<b.second+Dijkstra::dis[b.first];
64     }
65     vector<pair<int,int> >g[maxn];
66     int nxt[maxn][21],/* $i$ 点的 $2^j$ 祖先是 $nxt[i][j]$  */
67     need[maxn][21],/*由 $i$ 出发并在接下来 $2^j$ 个祖先(不包含 $i$ )中选一个开始走最短路径树 的最少花
68     费为 $need[i][j]$  */
69     path_cost[maxn][21];/*从 $i$ 点到其第 $2^j$ 祖先的路径长度为 $path\_cost[i][j]$  */
70     void init(int n,int t){
71         for(int i=1;i<=n;i++){
72             if(g[i].size()){
73                 nxt[i][0] = g[i][0].first;
74                 path_cost[i][0] = g[i][0].second;
75                 int nt=(g[i][0].first!=t);
76                 if(g[g[i][0].first].size()>nt){ /*第一步走最短路径树上的节点, 之后一步不走最短
77                 路径树上的点 */
78                     int z=g[g[i][0].first][nt].first;
79                     need[i][0] = Dijkstra::dis[z]+g[i][0].second+g[g[i][0].first][nt].second;
80                 }
81                 else need[i][0]=maxv;
82             }
83             else{
84                 nxt[i][0]=i;
85                 path_cost[i][0]=need[i][0]=maxv;
86             }
87         }
88     }
89     for(int j=1;j<=20;j++){
90         for(int i=1;i<=n;i++){

```

```

88         nxt[i][j]=nxt[nxt[i][j-1]][j-1];
89         path_cost[i][j]=path_cost[nxt[i][j-1]][j-1]+path_cost[i][j-1];
90         need[i][j]=min(need[i][j-1],path_cost[i][j-1]+need[nxt[i][j-1]][j-1]);
91     }
92 }
93 }
94 void addedge(int u,int v,int w){
95     g[u].push_back({v,w});
96 }
97 void dfs(int u,int t,int now_cost,int k){
98     /*
99     * 这循环是在枚举不在最短路径树上的边。
100    * 当 x = t 时，连出去的所有边都不是最短路径树上的边，
101    * 否则从第二条开始才是。
102    * 其中 (x!=t) 等价于 (x!=t)?1:0
103    */
104    for(int i=(u!=t);i<g[u].size()&&num<k;i++){
105        int &v=g[u][i].first,&w=g[u][i].second;
106        if(now_cost+w+Dijkstra::dis[v]>mid) break;
107        ans[++num]=now_cost+w+Dijkstra::dis[v];
108        dfs(v,t,now_cost+w,k);
109    }
110    if(u!=t&&num<k&&now_cost+need[u][20]<=mid){/*保证了一定存在一个位置，使得从此位置开始
111        不走最短路径树上的边，也能在mid要求的花费内到达终点 */
112        for(int i=19;i>=0;i--){
113            /* 若接下来 2^i 节点都不可能在花费小于mid情况下 不走最短路径树且到达终点，不如
114            就先进到点 nxt[i][x] */
115            if(now_cost+need[u][i]>mid){
116                now_cost+=path_cost[u][i];
117                u=nxt[u][i];
118            }
119            dfs(g[u][0].first, t, now_cost + g[u][0].second, k);
120        }
121    }
122    int solve(int s,int t,int k){
123        int left=0,right=maxv;
124        while(left<=right){
125            mid=(left+right)>>1;
126            num=0; ans[++num]=Dijkstra::dis[s];/* 先把最短路径树上起点到终点的路径算入 */
127            dfs(s,t,0,k);
128            if(num>=k) right=mid-1;
129            else left=mid+1;
130        }
131        return left;
132    }
133 }
134 Int main()
135 {
136     ios::sync_with_stdio(false);
137     cin.tie(0);
138     int n,m,k;cin>>n>>m>>k;
139     int s,t;cin>>s>>t;
140     Dijkstra::init();

```

```

140     for(int i=1;i<=m;i++){
141         int u,v,w;cin>>u>>v>>w;
142         kth::addedge(u,v,w);
143         Dijkstra::addedge(v,u,w); //建立反向边
144     }
145     Dijkstra::run(t);
146     for(int i=1;i<=n;i++) sort(kth::g[i].begin(),kth::g[i].end(),kth::cmp);
147     kth::init(n, t);
148     int res=kth::solve(s,t,k);
149     /*
150      * 虽然已经知道第 k 短路径长度是 res, 但得再把门槛设为 res - 1 后再搜索一次,
151      * 不足 k 个的值再补上 res, 否则直接用 res 当门槛去搜索的话,
152      * 可能会搜出过多的长度为 Low 的路径。
153      */
154     kth::num=0;kth::mid=res-1;
155     kth::ans[++kth::num]=Dijkstra::dis[s];
156     kth::dfs(s,t,0,k);
157     sort(kth::ans+1,kth::ans+kth::num+1);
158     while(kth::num<k) kth::ans[++kth::num]=res;
159     for(int i=1;i<=kth::num;i++){
160         if(kth::ans[i]>=INF) cout<<"NO"<<endl;
161         else cout<<kth::ans[i]<<endl;
162     }
163     cin.get(),cin.get();
164     return 0;
165 }

```

7.18 全局 k 短路

```

1  /*
2   * Author: Simon
3   * 功能: 全局k短路
4   * 复杂度:  $m\log(m)+k\log(k)$ 
5   * 多次查询, k为最大的k
6   */
7  #include<bits/stdc++.h>
8  using namespace std;
9  typedef int Int;
10 #define int long long
11 #define INF 0x3f3f3f3f
12 #define maxn 50005
13 struct node{
14     int u,v,w,rnk/*从u出发的, 第rank短的边*/;
15     node(){}
16     node(int u,int v,int w,int rank):u(u),v(v),w(w),rank(rank){}
17     bool operator <(const node&a)const{
18         return w>a.w;
19     }
20 };
21 int a[maxn];
22 struct pi{
23     int u,v,w;

```

```

24     pi(){}
25     pi(int u,int v,int w):u(u),v(v),w(w){}
26     bool operator <(const pi&a) const{
27         return w<a.w;
28     }
29 };
30 vector<pi>g[maxn];
31 Int main(){
32     ios::sync_with_stdio(false);
33     cin.tie(0);
34     int T;cin>>T;
35     while(T--){
36         int n,m,qq;cin>>n>>m>>qq; /*n个点, m条边, qq次查询*/
37         priority_queue<node>q;
38         for(int i=1;i<=m;i++){
39             int u,v,w;
40             cin>>u>>v>>w;
41             g[u].push_back({u,v,w});
42         }
43         for(int i=1;i<=n;i++) sort(g[i].begin(),g[i].end()); //按边权从小到大排序
44         for(int i=1;i<=n;i++) if(g[i].size()) q.push({g[i][0].u,g[i][0].v,g[i][0].w,0}); //初
            始将所有点的最短出边入队列
45         int Max=0; for(int i=1;i<=qq;i++) cin>>a[i],Max=max(Max,a[i]); //最大要算到第Max小的路
            径
46         vector<int>ans;
47         for(int i=1;i<=Max;i++){
48             node now=q.top();q.pop();
49             ans.push_back(now.w); //第i次出队列的边权长度, 就是第i小的路径长度
50             if(g[now.v].size()){ //1、从v点出发的最短边
51                 int u=now.v,v=g[now.v][0].v,w=g[now.v][0].w;
52                 q.push({u,v,w+now.w,0});
53             }
54             if(g[now.u].size()>now.rank+1){ //2、由u-v转为u-v'
55                 int u=now.u,v=g[now.u][now.rank+1].v,w=g[now.u][now.rank+1].w;
56                 q.push({u,v,now.w+w-g[now.u][now.rank].w,now.rank+1});
57             }
58         }
59         for(int i=1;i<=qq;i++) cout<<ans[a[i]-1]<<endl;
60         for(int i=0;i<=n;i++) g[i].clear();
61     }
62     return 0;
63 }

```

7.19 k 小带权匹配

```

1  /*
2   * Author: Simon
3   * 功能: 第k小带权匹配(枚举点)
4   * 复杂度:  $O(k \cdot m \cdot \log(1000 \cdot n))$ 
5   */
6  #include<bits/stdc++.h>
7  using namespace std;

```

```

8  typedef int Int;
9  #define int long long
10 #define INF 0x3f3f3f3f
11 #define maxn 25
12 #define maxm 1005
13 struct node{
14     int v,w,next;
15     node(){}
16     node(int v,int w,int next):v(v),w(w),next(next){}
17 }g[maxm];
18 int num=0/*所有匹配中权值和小于期望值mid的匹配数 */,mid/*第k小匹配的期望值 */,head[maxn],cnt
    =0;
19 void init(){
20     memset(head,-1,sizeof(head));
21     cnt=0;
22 }
23 void addedge(int u,int v,int w){
24     g[cnt]=node(v,w,head[u]);
25     head[u]=cnt++;
26 }
27 bool vis[maxn];
28 void dfs(int u,int val,int n,int k){
29     if(val>mid) return ;
30     if(u>n){
31         if(val<=mid) num++;
32         return ;
33     }
34     dfs(u+1,val,n,k); /*可能不选 */
35     for(int i=head[u];~i&&num<k/*关键剪枝 */;i=g[i].next){
36         int &v=g[i].v,&w=g[i].w;
37         if(!vis[v]&&val+w<=mid){ /*v节点未匹配, 并且匹配后总权值小于等于 期望值mid */
38             vis[v]=1;
39             dfs(u+1,val+w,n,k);
40             vis[v]=0;
41         }
42     }
43 }
44 Int main()
45 {
46     ios::sync_with_stdio(false);
47     cin.tie(0);
48     init();
49     int n,m,k;cin>>n>>m>>k;
50     for(int i=1;i<=m;i++){
51         int u,v,w;cin>>u>>v>>w;
52         addedge(u,v,w);
53     }
54     int left=0,right=1000*n;/*每条边的最大权值为1000, 最多有n/2个点 */
55     while(left<=right){
56         mid=(left+right)>>1; /*假设一个第k小匹配和的期望 */
57         num=0; dfs(1,0,n,k); /*暴力枚举判断满足小于等于mid的匹配有没有k个 */
58         if(num>=k) right=mid-1;/*若有, 则减少期望值 */
59         else left=mid+1; /*否则增加期望值 */
60     }

```

```

61     cout<<left<<endl;
62     cin.get(),cin.get();
63     return 0;
64 }
65 /*
66  * Author: Simon
67  * 功能: 第k小带权匹配(枚举边)
68  * 复杂度:  $O(k \cdot \log(1000 \cdot n))$ 
69  */
70 #include<bits/stdc++.h>
71 using namespace std;
72 typedef int Int;
73 #define int long long
74 #define INF 0x3f3f3f3f
75 #define maxn 25
76 #define maxm 1005
77 struct node{
78     int b,w;
79     bool operator <(const node&a)const{
80         if(w==a.w) return b<a.b;
81         return w<a.w;
82     }
83 }g[maxm];
84 int mid,num=0;
85 void dfs(int i,int m,int k,int val,int mask){
86     if(i>m) return ;
87     if(num>=k) return ; /*剪枝 */
88     if(val+g[i].w>mid) return ; /*因为排过序, 所以以后的边肯定大于g[i].w */
89     if(!(mask&g[i].b)){ /*当前此条边的u,v节点未被选过 */
90         num++; /*因为不选的在选的后面, 所以这里可以直接计数 */
91         dfs(i+1,m,k,val+g[i].w,mask|g[i].b);
92     }
93     dfs(i+1,m,k,val,mask); /*不选此条边, 因为边权从小到大排好序, 所以不选放在选的后面 */
94 }
95 Int main()
96 {
97     ios::sync_with_stdio(false);
98     cin.tie(0);
99     int n,m,k;cin>>n>>m>>k;
100     for(int i=1;i<=m;i++){
101         int u,v,w;cin>>u>>v>>w;
102         g[i].b=(1LL<<(u+n))|(1LL<<v); /*将u~v这条边hash处理 */
103         g[i].w=w;
104     }
105     sort(g+1,g+m+1); /*从小到大排序 */
106     int left=0,right=1000*n;
107     while(left<=right){
108         mid=(left+right)>>1;
109         num=1; /*全不选的方案 */dfs(1,m,k,0,0);
110         if(num>=k) right=mid-1;
111         else left=mid+1;
112     }
113     cout<<left<<endl;
114     cin.get(),cin.get();

```

```

115     return 0;
116 }

```

7.20 最小环

```

1  int min=INT_MAX;
2
3  for(k=1;k<=n;k++) {
4      for(i=1;i<=n;i++) {
5          for(j=1;j<=n;j++) {
6              if(dist[i][j]!=INF&&map[j][k]!=INF&&map[k][i]!=INF&&dist[i][j]+dist[j][k]+map[k][i]
                <mindis) {
7                  mindis=min(mindis,dist[i][j]+map[j][k]+map[k][i]);
8              }
9          }
10     }
11     for(i=1;i<=n;i++) {
12         for(j=1;j<=n;j++) {
13             if(dist[i][k]!=INF&&dist[k][j]!=INF&&dist[i][k]+dist[k][j]<dist[i][j]) {
14                 dist[i][j]=dist[i][k]+dist[k][j];
15                 pre[i][j]=pre[k][j];
16             }
17         }
18     }
19 }

```

7.21 最小树形图

```

1  #include <cstdio>
2  #include <cmath>
3  #define type int
4
5  type c[mm], in[mn], w[mn], ans;
6  int s[mm], t[mm], id[mn], pre[mn], q[mn], vis[mn];
7
8  type Directed_MST(int root,int NV,int NE) {
9      type ret=0, sum=0, tmp;
10     int i, j, u, v, r;
11     bool huan=1;
12     for (i=0;i<=NV;++i) in[i]=0, id[i]=i, pre[i]=-1;
13     while (huan) {
14         for(i=0;i<=NV;++i)
15             if(pre[j=id[i]]>=0) {
16                 if(pre[i]<0)in[i]+=w[j],id[i]=id[j];
17                 else in[i]+=w[i],ret+=w[i];
18             }
19         for(i=0;i<=NV;++i)pre[i]=-1,vis[i]=0;
20         for(i=0;i<NE;++i)
21             if((u=id[s[i]])!=(v=id[t[i]])&&(w[v]>(tmp=c[i]-in[t[i]]))||pre[v]<0))

```



```

22         pre[v]=u,w[v]=tmp;
23     for(i=1;i<=NV;++i)
24         if(i!=root&&id[i]==i&&pre[i]<0) return -1;
25     for(pre[root]=-1,sum=i=0;i<=NV;++i)
26         if(pre[i]>=0) sum+=w[i];
27     for(i=huan=0;i<=NV;++i)
28         if(!vis[i]) {
29             r=0,j=i;
30             while(j>=0&&vis[j]>=0) {
31                 if(vis[j]>0) {
32                     while(q[---r]!=j) id[q[r]]=j,vis[q[r]]=-1;
33                     huan=1,vis[j]=-1;
34                 }
35                 else vis[q[r++]]=j=1,j=pre[j];
36             }
37             while(r-->0) vis[q[r]]=pre[q[r]]=-1;
38         }
39     }
40     return ret+sum;
41 }
42
43 int main() {
44     int n,m,e,T,cas=0;
45     scanf("%d",&T);
46     while(T-->0) {
47         scanf("%d%d",&n,&m),--n;
48         e=0;
49         while(m-->0) scanf("%d%d%d",&s[e],&t[e],&c[e]),e+=(s[e]!=t[e]);
50         ans=Directed_MST(0,n,e);
51         if(ans<0) printf("Case#%d: Possums!\n",++cas);
52         else printf("Case#%d: %d\n",++cas,ans);
53     }
54     return 0;
55 }

```

7.22 次小生成树 (Prim)

```

1  // 0-indexed
2  bool vis[maxn];
3  int d[maxn][maxn];
4  int lowc[maxn];
5  int pre[maxn];
6  int Max[maxn][maxn];    // Max[i][j]表示i到j的路径上的最大边权
7  bool used[maxn][maxn];
8  int Prim(int n) {
9      int ans = 0;
10     memset(vis, false, sizeof(vis));
11     memset(Max, 0, sizeof(Max));
12     memset(used, false, sizeof(used));
13     vis[0] = true;
14     pre[0] = -1;
15     for (int i = 1; i < n; i++) {

```

```

16     lowc[i] = d[0][i];
17     pre[i] = 0;
18 }
19 lowc[0] = 0;
20 for (int i = 1; i < n; i++) {
21     int minc = INF;
22     int p = -1;
23     for (int j = 0; j < n; j++)
24         if (!vis[j] && minc > lowc[j]) {
25             minc = lowc[j];
26             p = j;
27         }
28     if (minc == INF) return -1;
29     ans += minc;
30     vis[p] = true;
31     used[p][pre[p]] = used[pre[p]][p] = true;
32     for (int j = 0; j < n; j++) {
33         if (vis[j]) Max[j][p] = Max[p][j] = max(Max[j][pre[p]], lowc[p]);
34         if (!vis[j] && lowc[j] > d[p][j]) {
35             lowc[j] = d[p][j];
36             pre[j] = p;
37         }
38     }
39 }
40 return ans;
41 }
42 int SMST(int n, int ans) {
43     int Min = INF;
44     for (int i = 0; i < n; i++)
45         for (int j = i + 1; j < n; j++)
46             if (d[i][j] != INF && !used[i][j])
47                 Min = min(Min, ans + d[i][j] - Max[i][j]);
48     if (Min == INF) return -1;
49     return Min;
50 }

```

7.23 次小生成树 (Kruskal)

```

1  //1-indexed
2  struct edge {
3      int s, t, w;    //从s到t 权值w
4      bool vis;
5      edge() {}
6      edge(int s, int t, int w) :s(s), t(t), w(w) {}
7      bool operator < (const edge e) const {
8          return w < e.w;
9      }
10 }e[maxm];
11
12 int pre[maxn];
13 int Max[maxn][maxn];    // Max[i][j]表示从i到j路径上的最大边权
14

```

```
15 int find(int x) {
16     int r = x, i = x, j;
17     while (pre[r] != r)
18         r = pre[r];
19     while (i != r) {    // 状态压缩
20         j = pre[i];
21         pre[i] = r;
22         i = j;
23     }
24     return r;
25 }
26
27 int kruskal(int n, int m) { // n为边数 m为点数
28     int lef = m - 1, ans = 0;
29     memset(Max, 0, sizeof(Max));
30     vector<int>v[maxn];
31     for (int i = 1; i <= m; i++) {
32         pre[i] = i;
33         v[i].push_back(i);
34     }
35     sort(e + 1, e + n + 1);
36     for (int i = 1; i <= n; i++) {
37         int fs = find(e[i].s), ft = find(e[i].t), len1, len2;
38         if (fs != ft) {
39             pre[fs] = ft;
40             ans += e[i].w;
41             lef--; e[i].vis = true;
42             len1 = v[fs].size(), len2 = v[ft].size();
43             for (int j = 0; j < len1; j++)
44                 for (int k = 0; k < len2; k++)
45                     Max[v[fs][j]][v[ft][k]] = Max[v[ft][k]][v[fs][j]] = e[i].w;
46             int tmp[maxn];
47             for (int j = 0; j < len1; j++)
48                 tmp[j] = v[fs][j];
49             for (int j = 0; j < len2; j++)
50                 v[fs].push_back(v[ft][j]);
51             for (int j = 0; j < len1; j++)
52                 v[ft].push_back(tmp[j]);
53         }
54         if (!lef) break;
55     }
56     if (lef) ans = -1; // 图不连通
57     return ans;
58 }
59
60 int SMST(int n, int ans) { // n为边数, ans为最小生成树权值
61     int ret = INF;
62     for (int i = 1; i <= n; i++)
63         if (!e[i].vis)
64             ret = min(ret, ans + e[i].w - Max[e[i].s][e[i].t]);
65     if (ret == INF) return -1;
66     return ret;
67 }
```

7.24 最小生成树计数

```
1 // 无向图，求生成树个数 Determinant算法
2 ll A[maxn][maxn], B[maxn][maxn];
3 ll determinant(int n) {
4     ll res = 1;
5     for (int i = 1; i <= n; i++) {
6         if (!B[i][i]) {
7             bool flag = false;
8             for (int j = i + 1; j <= n; j++) {
9                 if (B[j][i]) {
10                     flag = true;
11                     for (int k = i; k < n; k++)
12                         swap(B[i][k], B[j][k]);
13                     res = -res;
14                     break;
15                 }
16             }
17             if (!flag) return 0;
18         }
19         for (int j = i + 1; j <= n; j++) {
20             while (B[j][i]) {
21                 ll t = B[i][i] / B[j][i];
22                 for (int k = i; k <= n; k++) {
23                     B[i][k] = B[i][k] - t * B[j][k];
24                     swap(B[i][k], B[j][k]);
25                 }
26                 res = -res;
27             }
28         }
29         res *= B[i][i];
30     }
31     return res;
32 }
33 int main()
34 {
35     int n, m, k;
36     while (~scanf("%d%d%d", &n, &m, &k)) {
37         memset(A, 0, sizeof(A));
38         memset(B, 0, sizeof(B));
39         for (int i = 1; i <= m; i++) {
40             int a, b;
41             scanf("%d%d", &a, &b);
42             A[a][b] = A[b][a] = 1;
43         }
44         for (int i = 1; i <= n; i++) {
45             for (int j = 1; j <= n; j++) {
46                 if (i != j && !A[i][j]) {
47                     B[i][i]++;
48                     B[i][j] = -1;
49                 }
50             }
51         }
52         n--;
```

```
53     ll ans = determinant(n);
54     printf("%lld\n", ans);
55 }
56 }
```

7.25 最小树形图计数

```
1 // 有向图最小生成树计数
2 struct node {
3     int a, b, cost;
4 }edge[maxm];
5
6 int n, m, o;
7 ll ans, mod;
8 int pre[maxn], ka[maxn];
9 ll G[maxn][maxn], B[maxn][maxn];
10 bitset<maxn> vis;
11 vector<int> v[maxn];
12
13 bool cmp(node a, node b) { return a.cost < b.cost; }
14 int find(int x) { return pre[x] == x ? pre[x] : pre[x] = find(pre[x]); }
15
16 ll det(ll a[][maxn], int n) { //Matrix-Tree定理求Kirchhoff矩阵
17     for (int i = 0; i < n; i++)
18         for (int j = 0; j < n; j++) a[i][j] %= mod;
19     ll ret = 1;
20     for (int i = 1; i < n; i++) {
21         for (int j = i + 1; j < n; j++)
22             while (a[j][i]) {
23                 ll t = a[i][i] / a[j][i];
24                 for (int k = i; k < n; k++) a[i][k] = (a[i][k] - a[j][k] * t) % mod;
25                 for (int k = i; k < n; k++) swap(a[i][k], a[j][k]);
26                 ret = -ret;
27             }
28         if (a[i][i] == 0) return 0;
29         ret = ret * a[i][i] % mod;
30     }
31     return (ret + mod) % mod;
32 }
33
34 void Matrix_Tree() {
35     for (int i = 1; i <= n; i++) { //根据访问标记找出连通分量
36         if (vis[i]) {
37             v[find(i)].push_back(i);
38             vis[i] = 0;
39         }
40     }
41     for (int i = 1; i <= n; i++) {
42         if (v[i].size() > 1) { //枚举连通分量
43             memset(B, 0, sizeof(B));
44             int len = v[i].size();
45             for (int a = 0; a < len; a++) {
```

```

46         for (int b = a + 1; b < len; b++) {
47             int la = v[i][a], lb = v[i][b];
48             B[b][a] -= G[la][lb];
49             B[a][b] = B[b][a];
50             B[a][a] += G[la][lb];
51             B[b][b] += G[la][lb];
52         } //构造矩阵
53     }
54     ll ret = det(B, len) % mod;
55     ans = ans * ret % mod;
56     for (int j = 0; j < len; j++)
57         pre[v[i][j]] = i;
58 }
59 }
60 for (int i = 1; i <= n; i++) { //连通图缩点+初始化
61     pre[i] = find(i);
62     ka[i] = pre[i];
63     v[i].clear();
64 }
65 }
66
67 int main()
68 {
69     while (scanf("%d%d%lld", &n, &m, &mod), n || m || mod) {
70         for (int i = 1; i <= m; i++)
71             scanf("%d%d%d", &edge[i].a, &edge[i].b, &edge[i].cost);
72         sort(edge + 1, edge + m + 1, cmp);
73         for (int i = 1; i <= n; i++)
74             v[i].clear();
75         for (int i = 1; i <= n; i++)
76             pre[i] = ka[i] = i;
77         vis.reset();
78         memset(G, 0, sizeof(G));
79         ans = 1;
80         o = edge[1].cost;
81         for (int i = 1; i <= m; i++) {
82             int pa = find(edge[i].a), pb = find(edge[i].b);
83             if (pa != pb) {
84                 vis[pa] = 1;
85                 vis[pb] = 1;
86                 ka[find(pa)] = find(pb);
87                 G[pa][pb]++;
88                 G[pb][pa]++;
89             }
90             if (i == m || edge[i + 1].cost != o) { //所有相同的边并成一组
91                 Matrix_Tree();
92                 o = edge[i + 1].cost;
93             }
94         }
95         bool done = true;
96         for (int i = 2; i <= n; i++) {
97             if (ka[i] != ka[i - 1]) {
98                 done = false;
99                 break;

```

```

100     }
101     }
102     if (!done) printf("0\n");
103     else {
104         ans %= mod;
105         printf("%lld\n", ans);
106     }
107 }
108 return 0;
109 }

```

7.26 Dinic 最大流

```

1  /*
2   * Author: Simon,Ttmq36
3   * 复杂度:  $O(V^2 \cdot E)$ 
4   */
5  namespace Dinic{
6      struct node{
7          int v,w,next;
8          node(){}
9          node(int v,int w,int next):v(v),w(w),next(next){}
10     }g[maxn<<1];
11     int head[maxn],cur[maxn],cnt=0;
12     int dep[maxn];
13     void init(){
14         memset(head,-1,sizeof(head));cnt=0;
15     }
16     void addedge(int u,int v,int w){
17         g[cnt]=node(v,w,head[u]);
18         head[u]=cnt++;
19         g[cnt]=node(u,0,head[v]);
20         head[v]=cnt++;
21     }
22     bool bfs(int s,int t){
23         memset(dep,0,sizeof(dep));
24         queue<int>q;q.push(s);
25         dep[s]=1; cur[s]=head[s];
26         while(!q.empty()){
27             int u=q.front();q.pop();
28             for(int i=head[u];~i;i=g[i].next){
29                 int &v=g[i].v,&w=g[i].w;
30                 if(!dep[v]&&w>0){
31                     dep[v] = dep[u] + 1;
32                     cur[v] = head[v];
33                     q.push(v);
34                 }
35             }
36         }
37         if(!dep[t]) return 0;
38         return 1;
39     }

```

```

40 int dfs(int s,int t,int f){
41     if(s==t) return f;
42     int flow=0,u=s;
43     for(int &i=cur[u];~i;i=g[i].next){
44         int &v=g[i].v,&w=g[i].w;
45         if(dep[u]+1==dep[v]&&w>0){
46             int d=dfs(v,t,min(f,w));
47             if(d>0){
48                 flow+=d; f-=d;
49                 g[i].w-=d,g[i^1].w+=d;
50                 if(f<=0) break;
51             }
52         }
53     }
54     if(!flow) dep[u]=-INF;
55     return flow;
56 }
57 int maxFlow(int s,int t){
58     int ans=0;
59     while(bfs(s,t)){
60         ans+=dfs(s,t,INF);
61     }
62     return ans;
63 }
64 }

```

7.27 ISAP 最大流

```

1  const int maxn = "Edit";
2  struct ISAP {
3      int n, m, s, t;          //结点数，边数（包括反向弧），源点编号和汇点编号
4      vector<Edge> edges;      //边表。edges[e]和edges[e^1]互为反向弧
5      vector<int> G[maxn];    //邻接表，G[i][j]表示结点i的第j条边在e数组中的序号
6      bool vis[maxn];         //BFS使用
7      int d[maxn];            //起点到i的距离
8      int cur[maxn];          //当前弧下标
9      int p[maxn];            //可增广路上的一条弧
10     int num[maxn];           //距离标号计数
11     void init(int n) {
12         this->n = n;
13         for (int i = 0; i < n; i++) G[i].clear();
14         edges.clear();
15     }
16     void AddEdge(int from, int to, int cap) {
17         edges.pb(Edge(from, to, cap, 0));
18         edges.pb(Edge(to, from, 0, 0));
19         int m = edges.size();
20         G[from].pb(m - 2);
21         G[to].pb(m - 1);
22     }
23     int Augument() {
24         int x = t, a = INF;

```



```

25     while (x != s) {
26         Edge& e = edges[p[x]];
27         a = min(a, e.cap - e.flow);
28         x = edges[p[x]].from;
29     }
30     x = t;
31     while (x != s) {
32         edges[p[x]].flow += a;
33         edges[p[x] ^ 1].flow -= a;
34         x = edges[p[x]].from;
35     }
36     return a;
37 }
38 void BFS() {
39     clr(vis, 0);
40     clr(d, 0);
41     queue<int> q;
42     q.push(t);
43     d[t] = 0;
44     vis[t] = 1;
45     while (!q.empty()) {
46         int x = q.front();
47         q.pop();
48         int len = G[x].size();
49         for (int i = 0; i < len; i++) {
50             Edge& e = edges[G[x][i]];
51             if (!vis[e.from] && e.cap > e.flow) {
52                 vis[e.from] = 1;
53                 d[e.from] = d[x] + 1;
54                 q.push(e.from);
55             }
56         }
57     }
58 }
59 int Maxflow(int s, int t) {
60     this->s = s;
61     this->t = t;
62     int flow = 0;
63     BFS();
64     clr(num, 0);
65     for (int i = 0; i < n; i++)
66         if (d[i] < INF) num[d[i]]++;
67     int x = s;
68     clr(cur, 0);
69     while (d[s] < n) {
70         if (x == t) {
71             flow += Augument();
72             x = s;
73         }
74         int ok = 0;
75         for (int i = cur[x]; i < G[x].size(); i++) {
76             Edge& e = edges[G[x][i]];
77             if (e.cap > e.flow && d[x] == d[e.to] + 1) {
78                 ok = 1;

```

```

79         p[e.to] = G[x][i];
80         cur[x] = i;
81         x = e.to;
82         break;
83     }
84 }
85 if (!ok) { //Retreat
86     int m = n - 1;
87     for (int i = 0; i < G[x].size(); i++) {
88         Edge& e = edges[G[x][i]];
89         if (e.cap > e.flow) m = min(m, d[e.to]);
90     }
91     if (--num[d[x]] == 0) break; //gap优化
92     num[d[x] = m + 1]++;
93     cur[x] = 0;
94     if (x != s) x = edges[p[x]].from;
95 }
96 }
97 return flow;
98 }
99 };

```

7.28 最小费用最大流

```

1  #include <iostream>
2  #include <vector>
3  #include <queue>
4
5  const int MAXE = 1000;
6  const int MAXN = 1000;
7  const int INF = 1000000;
8
9  using ii = std::pair<int, int>;
10
11 struct edge {
12     int u, v, cost, cap, flow;
13 } E[MAXE], * pred[MAXN];
14
15 std::vector<edge *> g[MAXN];
16 int N, M, EE, dist[MAXN], phi[MAXN];
17
18 inline edge * opp(edge * e) {
19     return E + ((e - E) ^ 1);
20 }
21
22 void inti() {
23     for (int i = 0; i <= N; i++) {
24         g[i].clear();
25     }
26     EE = 0;
27 }
28

```

```

29 void add_edge(int u, int v, int cost, int cap) {
30     E[EE] = { u, v, cost, cap, 0 };
31     g[u].emplace_back(E + (EE++));
32     E[EE] = { v, u, -cost, 0, 0 };
33     g[v].emplace_back(E + (EE++));
34 }
35
36 bool dijkstra(int S, int T) {
37     std::fill(dist, dist + N, INF);
38     std::fill(pred, pred + N, nullptr);
39     std::priority_queue<ii, std::vector<ii>, std::greater<ii>> pq;
40     dist[S] = 0;
41     for (pq.emplace(dist[S], S); !pq.empty(); ) {
42         int u = pq.top().second;
43         pq.pop();
44         for (auto e : g[u]) {
45             if (e->cap - e->flow > 0 && dist[e->v] > dist[e->u] + e->cost + phi[e->u] - phi[e->v]) {
46                 dist[e->v] = dist[e->u] + e->cost + phi[e->u] - phi[e->v];
47                 pred[e->v] = e;
48                 pq.emplace(dist[e->v], e->v);
49             }
50         }
51     }
52     for (int i = 0; i < N; i++) {
53         phi[i] = std::min(INF, phi[i] + dist[i]);
54     }
55     return dist[T] != INF;
56 }
57
58 std::pair<int, int> mincost_maxflow(int S, int T) {
59     int mincost = 0, maxflow = 0;
60     std::fill(phi, phi + N, 0);
61     while (dijkstra(S, T)) {
62         int flow = INF;
63         for (edge * e = pred[T]; e; e = pred[e->u])
64             flow = std::min(flow, e->cap - e->flow);
65         for (edge * e = pred[T]; e; e = pred[e->u]) {
66             mincost += e->cost * flow;
67             e->flow += flow;
68             opp(e)->flow -= flow;
69         }
70         maxflow += flow;
71     }
72     return std::make_pair(mincost, maxflow);
73 }

```

7.29 ZKW 费用流

```

1 const int inf = ~0U>>1;
2 const int N = "Edit";
3

```

```
4 typedef struct seg{
5     int to,op,cost,nxt,f;
6 }seg;
7
8 seg v[N*40];
9
10 int ans =0,tot,dis[N],base[N],vis[N],ttf = 0;
11
12 int S,T; int cur[N];
13
14 void inti() {
15     memset(base,0,sizeof(base));
16     memset(dis,0,sizeof(dis));
17     tot = 0; ans = 0; ttf = 0;
18     memset(vis,0,sizeof(vis));
19 }
20
21 int aug(int u,int flow){
22     if (u == T){
23         ans += flow * dis[S];
24         ttf += flow;
25         return flow;
26     }
27     vis[u] = 1;
28     int now = 0;
29     for (int i = base[u];i;i = v[i].nxt){
30         int x = v[i].to;
31         if (vis[x] || v[i].f <= 0 || dis[u] != dis[x] + v[i].cost)
32             continue;
33         int tmp = aug(x,std::min(flow - now,v[i].f));
34         v[i].f -= tmp; v[v[i].op].f += tmp;
35         now += tmp;
36         if (now == flow) return flow;
37     }
38     return now;
39 }
40
41
42 int modlabel() {
43     int del = inf;
44     for (int i = S; i <= T; i++) {
45         if (vis[i]) for (int j = base[i];j;j = v[j].nxt) {
46             if (v[j].f){
47                 int x = v[j].to;
48                 if (!vis[x]) del = std::min(del,dis[x] + v[j].cost - dis[i]);
49             }
50         }
51     }
52     if (del == inf) {
53         return 0;
54     }
55     for (int i = S;i <= T;i++) {
56         if (vis[i]) {
57             vis[i] = 0,dis[i] += del,cur[i] = base[i];
```

```

58     }
59 }
60 return 1;
61 }
62
63
64 int zkw() {
65     for (int i = S; i <= T; i++) cur[i] = base[i];
66     int fl, t = 0;
67     do {
68         t = 0;
69         while((t = aug(S, inf))) memset(vis, 0, sizeof(vis));
70     } while(modlabel());
71     return ans;
72 }
73
74 void add(int x, int y, int c, int f){
75     v[++tot].to = y; v[tot].op = tot + 1;
76     v[tot].f = f; v[tot].cost = c;
77     v[tot].nxt = base[x]; base[x] = tot;
78     v[++tot].to = x; v[tot].op = tot - 1;
79     v[tot].f = 0; v[tot].cost = -c;
80     v[tot].nxt = base[y]; base[y] = tot;
81 }

```

7.30 上下界网络流

```

1  /*
2      首先建立一个源 $S$ 和一个汇 $T$ ，一般称为附加源和附加汇。
3      对于图中的每条弧 $\langle u, v \rangle$ ，假设它容量上界为 $c$ ，下界 $b$ ，那么把这条边拆为三条只有上界的弧。
4      一条为 $\langle S, v \rangle$ ，容量为 $b$ ；
5      一条为 $\langle u, T \rangle$ ，容量为 $b$ ；
6      一条为 $\langle u, v \rangle$ ，容量为 $c-b$ 。
7      其中前两条弧一般称为附加弧。
8      然后对这张图跑最大流，以 $S$ 为源，以 $T$ 为汇，如果所有的附加弧都满流，则原图有可行流；否则就
9      是无解。
10     这时，每条非附加弧的流量加上它的容量下界，就是原图中这条弧应该有的流量。
11
12     对于原图中的每条弧，我们把 $c-b$ 称为它的自由流量，意思就是只要它流满了下界，这些流多少都
13     没问题。
14     既然如此，对于每条弧 $\langle u, v \rangle$ ，我们强制给 $v$ 提供 $b$ 单位的流量，并且强制从 $u$ 那里拿走 $b$ 单位的流量，这
15     一步对应着两条附加弧。
16     如果这一系列强制操作能完成的话，也就是有一组可行流了。
17     注意：这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。
18 */
19
20 using namespace std;
21 const int oo = (1LL << 31) - 1;
22 const int LEN = 1e5 + 5;
23 struct node {
24     int x, y, l, r;
25 } a[LEN];
26 namespace ISAP {

```

```
23 int flow, tot, n, m, src, tar, qh, qt, cnt, ans;
24 struct edge {
25     int vet, next, len;
26 } E[LEN * 2];
27 int dis[LEN], gap[LEN], head[LEN], cur[LEN], q[LEN], vis[LEN], IN[LEN];
28 void add(int u, int v, int c) {
29     E[++tot] = (edge){v, head[u], c};
30     head[u] = tot;
31 }
32 void join(int u, int v, int c) {
33     add(u, v, c);
34     add(v, u, 0);
35 }
36 void bfs(int s) {
37     qh = qt = 0;
38     q[++qt] = s;
39     dis[s] = 0;
40     vis[s] = 1;
41     while (qh < qt) {
42         int u = q[++qh];
43         gap[dis[u]]++;
44         for (int e = head[u]; e != -1; e = E[e].next) {
45             int v = E[e].vet;
46             if (E[e ^ 1].len && !vis[v]) {
47                 dis[v] = dis[u] + 1;
48                 vis[v] = 1;
49                 q[++qt] = v;
50             }
51         }
52     }
53 }
54 int isap(int u, int aug) {
55     if (u == tar) return aug;
56     int flow = 0;
57     for (int e = head[u]; e != -1; e = E[e].next) {
58         int v = E[e].vet;
59         if (E[e].len && dis[v] == dis[u] - 1) {
60             int tmp = isap(v, min(aug - flow, E[e].len));
61             E[e].len -= tmp;
62             E[e ^ 1].len += tmp;
63             flow += tmp;
64             head[u] = e;
65             if (flow == aug || dis[src] == cnt) return flow;
66         }
67     }
68     if (!--gap[dis[u]]) dis[src] = cnt;
69     ++gap[dis[u]];
70     head[u] = cur[u];
71     return flow;
72 }
73 void init() {
74     tot = -1, gap[0] = 0;
75     for (int i = 1; i <= cnt; i++) {
76         dis[i] = gap[i] = vis[i] = IN[i] = 0;
```

```

77         head[i] = -1;
78     }
79 }
80 int maxflow(int s, int t) {
81     src = s, tar = t;
82     int res = 0;
83     for (int i = 1; i <= cnt; i++) cur[i] = head[i];
84     bfs(tar);
85     while (dis[src] < cnt) res += isap(src, oo);
86     return res;
87 }
88 }
89 using namespace ISAP;
90 int main() {
91     scanf("%d%d", &n, &m);
92     cnt = n;
93     src = ++cnt, tar = ++cnt;
94     init();
95     for (int i = 1; i <= m; i++) {
96         int x, y, l, r;
97         scanf("%d%d%d%d", &x, &y, &l, &r);
98         a[i] = (node){x, y, l, r};
99         join(x, y, r - 1);
100         IN[y] += l, IN[x] -= l;
101     }
102     for (int i = 1; i <= n; i++) {
103         if (IN[i] < 0) join(i, tar, -IN[i]);
104         else {
105             join(src, i, IN[i]);
106             flow += IN[i];
107         }
108     }
109     int ans = maxflow(src, tar);
110     if (flow == ans) {
111         puts("YES");
112         for (int i = 1; i <= m; i++) printf("%d\n", a[i].l + E[i * 2 - 1].len);
113     } else puts("NO");
114     return 0;
115 }

```

```
117 /*
```

```
118     先来看有源汇可行流
```

```
119     建模方法:
```

```
120     建立弧 $\langle t, s \rangle$ , 容量下界为0, 上界为 $oo$ 。
```

```
121     然后对这个新图 (实际上只是比原图多了一条边) 按照无源汇可行流的方法建模,
```

```
122     如果所有附加弧满流, 则存在可行流。
```

```
123     求原图中每条边对应的实际流量的方法, 同无源汇可行流, 只是忽略掉弧 $\langle t, s \rangle$ 就好。
```

```
124     而且这时候弧 $\langle t, s \rangle$ 的流量就是原图的总流量。
```

```
125     理解方法:
```

```
126     有源汇相比无源汇的不同就在于, 源和汇是不满足流量平衡的, 那么连接 $\langle t, s \rangle$ 之后,
```

```
127     源和汇也满足了流量平衡, 就可以直接按照无源汇的方式建模。
```

```
128     注意: 这张图的最大流只是对应着原图的一组可行流, 而不是原图的最大或最小流。
```

```
129
```

```
130     有源汇最大流
```

建模方法：

首先按照有源汇可行流的方法建模，如果不存在可行流，更别提什么最大流了。

如果存在可行流，那么在运行过有源汇可行流的图上（就是已经存在流量的那张图，流量不要清零），跑一遍从 s 到 t 的最大流（这里的 s 和 t 是原图的源和汇，不是附加源和附加汇），就是原图的最大流。

理解方法：

为什么要在那个已经有了流量的图上跑最大流？因为那张图保证了每条弧的容量下界，在这张图上跑最大流，

实际上就是在容量下界全部满足的前提下尽量多得获得“自由流量”。

注意，在这张已经存在流量的图上，弧 $\langle t, s \rangle$ 也是存在流量的，千万不要忽略这条弧。

因为它的相反弧 $\langle s, t \rangle$ 的流量为 $\langle t, s \rangle$ 的流量的相反数，且 $\langle s, t \rangle$ 的容量为 0 ，所以这部分的流量也是会被算上的。

```
140 */
141 using namespace std;
142 typedef long long ll;
143 const int LEN = 1e5 + 5;
144 const int oo = (1LL << 31) - 1;
145 namespace DINIC {
146     int tot, n, m, src, tar, qh, qt, cnt, s, t, S, T;
147     int ans, flow;
148     struct edge {
149         int vet, next, len;
150     } E[LEN * 2];
151     int dis[LEN], gap[LEN], head[LEN], cur[LEN], q[LEN], vis[LEN], IN[LEN];
152     void add(int u, int v, int c) {
153         E[++tot] = (edge){v, head[u], c};
154         head[u] = tot;
155     }
156     void join(int u, int v, int c) {
157         add(u, v, c);
158         add(v, u, 0);
159     }
160     void init() {
161         tot = -1;
162         for (int i = 1; i <= cnt; i++) head[i] = -1;
163     }
164     bool bfs() {
165         for (int i = 1; i <= cnt; i++) dis[i] = 0;
166         qh = qt = 0;
167         q[++qt] = src;
168         dis[src] = 1;
169         while (qh < qt) {
170             int u = q[++qh];
171             for (int e = head[u]; e != -1; e = E[e].next) {
172                 int v = E[e].vet;
173                 if (E[e].len && !dis[v]) {
174                     dis[v] = dis[u] + 1;
175                     if (v == tar) return 1;
176                     q[++qt] = v;
177                 }
178             }
179         }
180         return dis[tar];
181     }
```



```

182     int dfs(int u, int aug) {
183         if (u == tar || !aug) return aug;
184         int tmp = 0;
185         for (int &e = cur[u]; e != -1; e = E[e].next) {
186             int v = E[e].vet;
187             if (dis[v] == dis[u] + 1) {
188                 if (tmp = dfs(v, min(aug, E[e].len))) {
189                     E[e].len -= tmp;
190                     E[e ^ 1].len += tmp;
191                     return tmp;
192                 }
193             }
194         }
195         return 0;
196     }
197     int maxflow(int s, int t) {
198         src = s, tar = t;
199         int res = 0, flow = 0;
200         while (bfs()) {
201             for (int i = 1; i <= cnt; i++) cur[i] = head[i];
202             while (flow = dfs(src, oo)) res += flow;
203         }
204         return res;
205     }
206 }
207 using namespace DINIC;
208 int main() {
209     scanf("%d%d%d%d", &n, &m, &s, &t);
210     cnt = n;
211     S = ++cnt, T = ++cnt;
212     init();
213     for (int i = 1; i <= m; i++) {
214         int x, y, l, r;
215         scanf("%d%d%d%d", &x, &y, &l, &r);
216         join(x, y, r - 1);
217         IN[y] += l, IN[x] -= l;
218     }
219     for (int i = 1; i <= n; i++) {
220         if (IN[i] < 0) join(i, T, -IN[i]);
221         else if (IN[i] > 0) {
222             flow += IN[i];
223             join(S, i, IN[i]);
224         }
225     }
226     join(t, s, oo);
227     ans = maxflow(S, T);
228     if (ans != flow) puts("please go home to sleep");
229     else {
230         ans = maxflow(s, t);
231         printf("%lld\n", ans);
232     }
233     return 0;
234 }
235

```

```

236  /*
237      先来看有源汇可行流
238      建模方法：
239      建立弧<t,s>，容量下界为0，上界为oo。
240      然后对这个新图（实际上只是比原图多了一条边）按照无源汇可行流的方法建模，
241      如果所有附加弧满流，则存在可行流。
242      求原图中每条边对应的实际流量的方法，同无源汇可行流，只是忽略掉弧<t,s>就好。
243      而且这时候弧<t,s>的流量就是原图的总流量。
244      理解方法：
245      有源汇相比无源汇的不同就在于，源和汇是不满足流量平衡的，那么连接<t,s>之后，
246      源和汇也满足了流量平衡，就可以直接按照无源汇的方式建模。
247      注意：这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。
248
249      有源汇最小流
250      有源汇最小流的常见建模方法比较多，我就只说我常用的一种。
251      建模方法：
252      首先按照有源汇可行流的方法建模，但是不要建立<t,s>这条弧。
253      然后在这个图上，跑从附加源ss到附加汇tt的最大流。
254      这时候再添加弧<t,s>，下界为0，上界oo。
255      在现在的这张图上，从ss到tt的最大流，就是原图的最小流。
256      理解方法：
257      我们前面提到过，有源汇可行流的流量只是对应一组可行流，并不是最大或者最小流。
258      并且在跑完有源汇可行流之后，弧<t,s>的流量就是原图的流量。
259      从这个角度入手，我们想让弧<t,s>的流量尽量小，就要尽量多的消耗掉那些“本来不需要经过<t,s>”
        的流量。
260      于是我们在添加<t,s>之前，跑一遍从ss到tt的最大流，就能尽量多的消耗那些流量啦QwQ。
261  */
262  using namespace std;
263  typedef long long ll;
264  const int LEN = 2e5 + 5;
265  const int oo = (1LL << 31) - 1;
266  namespace DINIC {
267      int tot, n, m, src, tar, qh, qt, cnt, s, t, S, T, ans, flow;
268      struct edge {
269          int vet, next, len;
270      } E[LEN * 2];
271      int dis[LEN], gap[LEN], head[LEN], cur[LEN], q[LEN], vis[LEN], IN[LEN];
272      void add(int u, int v, int c) {
273          E[++tot] = (edge){v, head[u], c};
274          head[u] = tot;
275      }
276      void join(int u, int v, int c) {
277          add(u, v, c);
278          add(v, u, 0);
279      }
280      void init() {
281          tot = -1;
282          for (int i = 1; i <= cnt; i++) head[i] = -1;
283      }
284      bool bfs() {
285          for (int i = 1; i <= cnt; i++) dis[i] = 0;
286          qh = qt = 0;
287          q[++qt] = src;
288          dis[src] = 1;

```

```

289         while (qh < qt) {
290             int u = q[++qh];
291             for (int e = head[u]; e != -1; e = E[e].next) {
292                 int v = E[e].vet;
293                 if (E[e].len && !dis[v]) {
294                     dis[v] = dis[u] + 1;
295                     if (v == tar) return 1;
296                     q[++qt] = v;
297                 }
298             }
299         }
300         return dis[tar];
301     }
302     int dfs(int u, int aug) {
303         if (u == tar || !aug) return aug;
304         int tmp = 0;
305         for (int &e = cur[u]; e != -1; e = E[e].next) {
306             int v = E[e].vet;
307             if (dis[v] == dis[u] + 1) {
308                 if (tmp = dfs(v, min(aug, E[e].len))) {
309                     E[e].len -= tmp;
310                     E[e ^ 1].len += tmp;
311                     return tmp;
312                 }
313             }
314         }
315         return 0;
316     }
317     int maxflow(int s, int t) {
318         src = s, tar = t;
319         int res = 0, flow = 0;
320         while (bfs()) {
321             for (int i = 1; i <= cnt; i++) cur[i] = head[i];
322             while (flow = dfs(src, oo)) res += flow;
323         }
324         return res;
325     }
326 }
327 using namespace DINIC;
328 int main() {
329     scanf("%d%d%d%d", &n, &m, &s, &t);
330     cnt = n;
331     S = ++cnt, T = ++cnt;
332     init();
333     for (int i = 1; i <= m; i++) {
334         int x, y, l, r;
335         scanf("%d%d%d%d", &x, &y, &l, &r);
336         join(x, y, r - 1);
337         IN[y] += 1, IN[x] -= 1;
338     }
339     for (int i = 1; i <= n; i++) {
340         if (IN[i] < 0) join(i, T, -IN[i]);
341         else if (IN[i] > 0) {
342             flow += IN[i];

```

```

343         join(S, i, IN[i]);
344     }
345 }
346 ans = maxflow(S, T);
347 flow -= ans;
348 join(t, s, oo);
349 ans = maxflow(S, T);
350 if (ans != flow) puts("please go home to sleep");
351 else printf("%d\n", ans);
352 return 0;
353 }

```

7.31 图匹配理论

- 1 二分图匹配：
- 2 点覆盖、最小点覆盖
- 3 点覆盖集即一个点集，使得所有边至少有一个端点在集合里。或者说是“点”覆盖了所有“边”。。极小点覆盖(minimal vertex covering)：本身为点覆盖，其真子集都不是。最小点覆盖(minimum vertex covering)：点最少的点覆盖。点覆盖数(vertex covering number)：最小点覆盖的点数。
- 4
- 5 边覆盖、极小边覆盖
- 6 边覆盖集即一个边集，使得所有点都与集合里的边邻接。或者说是“边”覆盖了所有“点”。极小边覆盖(minimal edge covering)：本身是边覆盖，其真子集都不是。最小边覆盖(minimum edge covering)：边最少的边覆盖。边覆盖数(edge covering number)：最小边覆盖的边数。
- 7
- 8 独立集、极大独立集
- 9 独立集即一个点集，集合中任两个结点不相邻，则称V为独立集。或者说是导出的子图是零图（没有边）的点集。极大独立集(maximal independent set)：本身为独立集，再加入任何点都不是。最大独立集(maximum independent set)：点最多的独立集。独立数(independent number)：最大独立集的点数。
- 10
- 11 团
- 12 团即一个点集，集合中任两个结点相邻。或者说是导出的子图是完全图的点集。极大团(maximal clique)：本身为团，再加入任何点都不是。最大团(maximum clique)：点最多的团。团数(clique number)：最大团的点数。
- 13
- 14 边独立集、极大边独立集
- 15 边独立集即一个边集，满足边集中的任两边不邻接。极大边独立集(maximal edge independent set)：本身为边独立集，再加入任何边都不是。最大边独立集(maximum edge independent set)：边最多的边独立集。边独立数(edge independent number)：最大边独立集的边数。
- 16
- 17 边独立集又称匹配(matching)，相应的有极大匹配(maximal matching)，最大匹配(maximum matching)，匹配数(matching number)。
- 18
- 19 支配集、极小支配集
- 20 支配集即一个点集，使得所有其他点至少有一个相邻点在集合里。或者说是一部分的“点”支配了所有“点”。极小支配集(minimal dominating set)：本身为支配集，其真子集都不是。最小支配集(minimum dominating set)：点最少的支配集。支配数(dominating number)：最小支配集的点数。
- 21
- 22 边支配集、极小边支配集
- 23 边支配集即一个边集，使得所有边至少有一条邻接边在集合里。或者说是一部分的“边”支配了所有“边”。极小边支配集(minimal edge dominating set)：本身是边支配集，其真子集都不是。最小边支配集(minimum edge dominating set)：边最少的边支配集。边支配数(edge dominating number)：

最小边支配集的边数。

24

25 最小路径覆盖

26 最小路径覆盖(path covering): 是“路径”覆盖“点”，即用尽量少的不相交简单路径覆盖有向无环图G的所有顶点，即每个顶点严格属于一条路径。路径的长度可能为0(单个点)。

27 最小路径覆盖数=G的点数-最小路径覆盖中的边数。应该使得最小路径覆盖中的边数尽量多，但是又不能让两条边在同一个顶点相交。拆点：将每一个顶点i拆成两个顶点Xi和Yi。然后根据原图中边的信息，从X部往Y部引边。所有边的方向都是由X部到Y部。因此，所转化出的二分图的最大匹配数则是原图G中最小路径覆盖上的边数。因此由最小路径覆盖数=原图G的顶点数-二分图的最大匹配数便可以得解。

28

29 匹配

30 匹配(matching)是一个边集，满足边集中的边两两不邻接。匹配又称边独立集(edge independent set)。

31 在匹配中的点称为匹配点(matched vertex)或饱和点；反之，称为未匹配点(unmatched vertex)或未饱和点。

32 交错轨(alternating path)是图的一条简单路径，满足任意相邻的两条边，一条在匹配内，一条不在匹配内。

33 增广轨(augmenting path): 是一个始点与终点都为未匹配点的交错轨。

34 最大匹配(maximum matching)是具有最多边的匹配。

35 匹配数(matching number)是最大匹配的大小。

36 完美匹配(perfect matching)是匹配了所有点的匹配。

37 完备匹配(complete matching)是匹配了二分图较小集合(二分图X, Y中小的那个)的所有点的匹配。

38 增广轨定理：一个匹配是最大匹配当且仅当没有增广轨。

39 所有匹配算法都是基于增广轨定理：一个匹配是最大匹配当且仅当没有增广轨。这个定理适用于任意图。

40

41 二分图的性质

42 二分图中，点覆盖数是匹配数。

43 (1) 二分图的最大匹配数等于最小覆盖数，即求最少的点使得每条边都至少和其中的一个点相关联，很显然直接取最大匹配的一段节点即可。

44 (2) 二分图的独立数等于顶点数减去最大匹配数，很显然的把最大匹配两端的点都从顶点集中去掉这个时候剩余的点是独立集，这是 $|V|-2|M|$ ，同时必然可以从每条匹配边的两端取一个点加入独立集并且保持其独立集性质。

45 (3) DAG的最小路径覆盖，将每个点拆点后作最大匹配，结果为 $n-m$ ，求具体路径的时候顺着匹配边走就可以，匹配边 $i \rightarrow j', j \rightarrow k', k \rightarrow l' \dots$ 构成一条有向路径。

46 (4) 最大匹配数=左边匹配点+右边未匹配点。因为在最大匹配集中的任意一条边，如果他的左边没标记，右边被标记了，那么我们就可找到一条新的增广路，所以每一条边都至少被一个点覆盖。

47 (5) 最小边覆盖=图中点的个数-最大匹配数=最大独立集。

48

49 定理1：最小覆盖数 = 最大匹配数

50 定理2：最大独立集S 与 最小覆盖集T 互补

51

52 有向无环图最小不相交路径覆盖

53 定义：用最少的不相交路径覆盖所有顶点。

54 定理：把原图中的每个点V拆成Vx和Vy，如果有一条有向边A→B，那么就加边Ax-By。这样就得到了一个二分图，最小路径覆盖=原图的节点数-新图最大匹配。

55

56 有向无环图最小可相交路径覆盖

57 定义：用最小的可相交路径覆盖所有顶点。

58 算法：先用floyd求出原图的传递闭包，即如果a到b有路，那么就加边a→b。然后就转化成了最小不相交路径覆盖问题。

59

60 Kuhn-Munkers算法的几种变形应用

61 1.Kuhn-Munkers算法是求最大权完备匹配，如果要求最小权完备匹配怎么办？方法很简单，只需将所有的边权值取其相反数，求最大权完备匹配，匹配的值再取相反数即可。

- 62 2.Kuhn–Munkers算法的运行要求是必须存在一个完备匹配，如果求一个最大权匹配(不一定完备)该如何办？
依然很简单，把不存在的边权值赋为0。
- 63 3.Kuhn–Munkers算法求得的最大权匹配是边权值和最大，如果我想要边权之积最大，又怎样转化？还是不难办到，每条边权取自然对数，然后求最大和权匹配，求得的结果a再算出 e^a 就是最大积匹配。

7.32 二分图最大匹配匈牙利算法

```

1  int n, m;
2  int g[maxn][maxn]; //0-labeled
3  int linker[maxn];
4  bool used[maxn];
5
6  bool dfs(int u) {
7      int v;
8      for(v = 0; v < m; v++) {
9          if(g[u][v] && !used[v]) {
10             used[v] = true;
11             if(linker[v] == -1 || dfs(linker[v])) {
12                 linker[v] = u;
13                 return true;
14             }
15         }
16     }
17     return false;
18 }
19
20 int hungary() {
21     int res = 0;
22     int u;
23     memset(linker, -1, sizeof(linker));
24     for(u = 0; u < n; u++) {
25         memset(used, 0, sizeof(used));
26         if(dfs(u)) {
27             res++;
28         }
29     }
30     return res;
31 }

```

7.33 二分图最大权匹配 KM 算法

```

1  const int maxn = "Edit";
2  const int inf = 2e9;
3
4  int n, cost[maxn][maxn];
5  int lx[maxn], ly[maxn], match[maxn], slack[maxn], prev[maxn];
6  bool vy[maxn];
7

```

```

8 void augment(int root) {
9     std::fill(vy + 1, vy + n + 1, false);
10    std::fill(slack + 1, slack + n + 1, inf);
11    int py;
12    match[py = 0] = root;
13    do {
14        vy[py] = true;
15        int x = match[py], delta = inf, yy;
16        for (int y = 1; y <= n; y++) {
17            if (!vy[y]) {
18                if (lx[x] + ly[y] - cost[x][y] < slack[y]) {
19                    slack[y] = lx[x] + ly[y] - cost[x][y];
20                    prev[y] = py;
21                }
22                if (slack[y] < delta) {
23                    delta = slack[y];
24                    yy = y;
25                }
26            }
27        }
28        for (int y = 0; y <= n; y++) {
29            if (vy[y]) {
30                lx[match[y]] -= delta;
31                ly[y] += delta;
32            } else {
33                slack[y] -= delta;
34            }
35        }
36        py = yy;
37    } while (match[py] != -1);
38    do {
39        int pre = prev[py];
40        match[py] = match[pre];
41        py = pre;
42    } while (py);
43 }
44
45 int KM() {
46     for (int i = 1; i <= n; i++) {
47         lx[i] = ly[i] = 0;
48         match[i] = -1;
49         for (int j = 1; j <= n; j++) {
50             lx[i] = std::max(lx[i], cost[i][j]);
51         }
52     }
53     int answer = 0;
54     for (int root = 1; root <= n; root++) {
55         augment(root);
56     }
57     for (int i = 1; i <= n; i++) {
58         answer += lx[i];
59         answer += ly[i];
60         //printf("%d %d\n", match[i], i);
61     }

```

```
62     return answer;  
63 }
```

第八章 数据结构

8.1 树状数组

```
1  /*
2   * Author: Simon
3   * 功能：单点修改, 区间查询
4   */
5  int tree[maxn];
6  inline int lowbit(int x){
7      return x&(-x);
8  }
9  inline void update(int x,int val){ /*将x处的值增加val */
10     for(int i=x;i<maxn;i+=lowbit(i)){
11         tree[i]+=val;
12     }
13 }
14 inline int query(int x){ /*求[1,x]的所有值的和 */
15     int ans=0;
16     for(int i=x;i>0;i-=lowbit(i)){
17         ans+=tree[i];
18     }
19     return ans;
20 }
21 /*
22 * Author: Simon
23 * 功能：区间修改, 单点查询
24 */
25 int a[maxn];
26 int main(){
27     int n;scanf("%d",&n);
28     for(int i=1;i<=n;i++) scanf("%d",a+i),update(i,a[i]-a[i-1]); /*初始化 */
29     update(2,10);update(8,-10); /*将区间[2,7]的所有数的值增加10 */
30     printf("%d\n",query(4)); /*查询[4]处的值 */
31 }
32 /*
33 * Author: Simon
34 * 功能：单点修改, 区间最值
35 */
36 int tree[maxn],a[maxn];
37 inline int lowbit(int x){
38     return x&(-x);
39 }
40 inline void update(int x,int val,int n){
41     for(int i=x;i<=n;i+=lowbit(i)){
42         tree[i]=val;
```

```

43     for(int j=1;j<lowbit(i);j<=1){
44         tree[i]=max(tree[i],tree[i-j]);
45     }
46 }
47 }
48 inline int query(int x,int y){
49     int ans=0;
50     while(x<=y){
51         ans=max(ans,a[y]);
52         for(y-=1;y-lowbit(y)>=x;y-=lowbit(y)){
53             ans=max(ans,tree[y]);
54         }
55     }
56     return ans;
57 }
58 /*
59  * Author: Simon
60  * 功能：区间修改，区间查询
61  * 若要修改的区间为[x,y]，则要更新的为update(x,val);update(y+1,-val);
62  * 若要查询的区间为[x,y]，则查询的为query(y)-query(x-1);
63  */
64 int tree1[maxn],tree2[maxn];
65 inline int lowbit(int x){
66     return x&(-x);
67 }
68 inline void update(int x,int val){
69     for(int i=x;i<maxn;i+=lowbit(i)){
70         tree1[i]+=val;tree2[i]+=x*val;
71     }
72 }
73 inline int query(int x){
74     int ans=0;
75     for(int i=x;i>0;i-=lowbit(i)){
76         ans+=(x+1)*tree1[i]-tree2[i];
77     }
78     return ans;
79 }
80 int a[maxn];
81 int main(){
82     int n;scanf("%d",&n);
83     for(int i=1;i<=n;i++) scanf("%d",a+i),update(i,a[i]-a[i-1]); /*初始化 */
84     update(2,10);update(8,-10); /*将区间[2,7]所有的值增加10 */
85     printf("%d\n",query(6)-query(4-1)); /*查询区间[4,6]的和 */
86 }
87 /*
88  * Author: Simon
89  * 功能：求第k小值
90  */
91 int tree[maxn];
92 inline int lowbit(int x){
93     return x&(-x);
94 }
95 inline void update(int x,int val){ /*将x处的值增加val，此处val一般为1 */
96     for(int i=x;i<maxn;i+=lowbit(i)){

```

```

97         tree[i]+=val;
98     }
99 }
100 inline int query(int x){ /*求[1,x]的所有值的和 */
101     int ans=0;
102     for(int i=x;i>0;i-=lowbit(i)){
103         ans+=tree[i];
104     }
105     return ans;
106 }
107 inline get_kth(int k,int n){ /*求第k小的数的值, n为最大的数的值, 一般比最大数的值大1 */
108     int ans=0,cnt=0; /*ans为当前是哪个数, cnt为上个数排名为多少 */
109     for(int i=log2(n);i>=0;i--){
110         ans+=1<<i;
111         if(ans>=n||cnt+tree[ans]>=k) ans-=1<<i; /*若当前数大于最大的数, 或者排名大于k, 回退 */
112         else cnt+=tree[ans];
113     }
114     return ans+1;
115 }

```

8.2 二维树状数组

```

1  int N;
2  int c[maxn][maxn];
3
4  inline int lowbit(int t) {
5      return t&(-t);
6  }
7
8  void update(int x, int y, int v) {
9      for (int i=x; i<=N; i+=lowbit(i)) {
10         for (int j=y; j<=N; j+=lowbit(j)) {
11             c[i][j]+=v;
12         }
13     }
14 }
15
16 int query(int x, int y) {
17     int s = 0;
18     for (int i=x; i>0; i-=lowbit(i)) {
19         for (int j=y; j>0; j-=lowbit(j)) {
20             s += c[i][j];
21         }
22     }
23     return s;
24 }
25
26 int sum(int x, int y, int xx, int yy) {
27     x--, y--;
28     return query(xx, yy) - query(xx, y) - query(x, yy) + query(x, y);
29 }
30 /*

```

```

31  * Author: Simon
32  * 功能：二维区间修改，区间查询
33  * 若要修改的矩形区间为  $[x1, y1] \sim [x2, y2]$ ，则更新的点为
34  *  $update(x1, y1, val); update(x2+1, y2+1, val); update(x1, y2+1, -val); update(x2+1, y1, -val);$ 
35  * 若要查询的举行区间为  $[x1, y1] \sim [x2, y2]$ ，则查询的值为
36  *  $query(x2, y2) - query(x1-1, y2) - query(x2, y1-1) + query(x1-1, y1-1)$ 
37  */
38  int tree1[maxn][maxn], tree2[maxn][maxn], tree3[maxn][maxn], tree4[maxn][maxn];
39  inline int lowbit(int x){
40      return x & (-x);
41  }
42  inline void update(int x, int y, int val){
43      for(int i=x; i<maxn; i+=lowbit(i)){
44          for(int j=y; j<maxn; j+=lowbit(j)){
45              tree1[i][j] += val; tree2[i][j] += (x-1)*val;
46              tree3[i][j] += (y-1)*val; tree4[i][j] += (x-1)*(y-1)*val;
47          }
48      }
49  }
50  inline int query(int x, int y){
51      int ans=0;
52      for(int i=x; i>0; i-=lowbit(i)){
53          for(int j=y; j>0; j-=lowbit(j)){
54              ans += (x*y)*tree1[i][j] - y*tree2[i][j] - x*tree3[i][j] + tree4[i][j];
55          }
56      }
57      return ans;
58  }
59  int a[maxn][maxn];
60  int main(){
61      int n, m; scanf("%d%d", &n, &m);
62      for(int i=1; i<=n; i++){
63          for(int j=1; j<=m; j++){
64              scanf("%d", &a[i][j]);
65              update(i, j, a[i][j] - a[i-1][j] - a[i][j-1] + a[i-1][j-1]);
66          }
67      }
68      update(2, 3, 10); update(7, 6, 10); update(2, 6, -10); update(7, 3, -10);
69      /*将矩形区间[2,3]~[6,5]所有的值增加10 */
70      printf("%d\n", query(4, 5) + query(2, 1) - query(2, 5) - query(4, 2));
71      /*查询矩形区间[3,2]~[4,5]的和 */
72  }

```

8.3 树状数组求逆序对

```

1  //树状数组求逆序对
2  const int maxn = "Edit";
3
4  int lowbit(int x) {
5      return (x & -x);
6  }
7

```

```
8  bool cmp(std::pair<int, int> no1, std::pair<int, int> no2) {
9      return no1.first < no2.first;
10 }
11
12 int d[maxn], p[maxn], n;
13 std::pair<int, int> start[maxn];
14
15 void add(int x) {
16     while (x <= n) {
17         d[x]++;
18         x += lowbit(x);
19     }
20 }
21
22 long long sum(int x) {
23     long long sum = 0;
24     while (x) {
25         sum += d[x];
26         x -= lowbit(x);
27     }
28     return sum;
29 }
30
31 int main(int argc, char *argv[]) {
32     long long ans;
33     std::cin >> n;
34     memset(d, 0, sizeof(d));
35     ans = 0;
36     for (int i = 1; i <= n; i++) {
37         std::cin >> start[i].first;
38         start[i].second = i;
39     }
40     std::sort(start + 1, start + n + 1, cmp);
41     int id = 1;
42     p[start[1].second] = 1;
43     for (int i = 2; i <= n; ++i) {
44         if (start[i].first == start[i - 1].first) {
45             p[start[i].second] = id;
46         } else {
47             p[start[i].second] = ++id;
48         }
49     }
50     for (int i = 1; i <= n; i++) {
51         add(p[i]);
52         ans += i - sum(p[i]);
53     }
54     std::cout << ans << std::endl;
55     return 0;
56 }
```

```

1  const int N = 1000;
2
3  template <class T>
4  class Heap {
5      private:
6          T h[N];
7          int len;
8      public:
9          Heap() {
10             len = 0;
11         }
12         inline void push(const T& x) {
13             h[++len] = x;
14             std::push_heap(h+1, h+1+len, std::greater<T>());
15         }
16         inline T pop() {
17             std::pop_heap(h+1, h+1+len, std::greater<T>());
18             return h[len--];
19         }
20         inline T& top() {
21             return h[1];
22         }
23         inline bool empty() {
24             return len == 0;
25         }
26 };

```

8.5 单调栈和单调队列

```

1  /*
2   * Author: Simon
3   * 功能：单调栈求某子序列中的最小值乘以子序列所有元素和最大
4   * 最基础的应用就是给定一组数，针对每个数，寻找它和它右边第一个比它大的数之间有多少个数。
5   * 给定一序列，寻找某一子序列，使得子序列中的最小值乘以子序列的长度最大。
6   * 给定一序列，寻找某一子序列，使得子序列中的最小值乘以子序列所有元素和最大。
7   * 给定一序列，在限定每个字母出现次数的情况下，求其字典序最小的k长子序列。可求后缀和，
8   *     当一个字母出栈时，判断此后位置当前字母的个数是否满足限制条件，若满足出栈，否则不出栈。
9   * 复杂度：O(n)
10 */
11 int Stack[maxn], lft[maxn], top=0, ans=0, a[maxn];
12 a[n+1]=INF;
13 for(int i=1; i<=n+1; i++){
14     int t=i; lft[i]=i;
15     while(top&&a[i]<a[Stack[top]]){
16         t=Stack[top--];
17         ans=max(ans, (i-lft[t])*a[t]);
18     }
19     Stack[++top]=i;
20     lft[i]=lft[t];
21 }
22 /*
23  * Author: Simon

```

```

24  * 功能：求区间长度小于 $k$ 的区间最小值
25  * 复杂度： $O(n)$ 
26  */
27  int q[maxn], l=1, r=0, a[maxn];
28  for(int i=1; i<=n; i++){
29      while(l<=r&&a[i]<=a[q[r]]) r--; //维护单调递增区间
30      q[++r]=i;
31      while(l<=r&&i-q[l]>=k) l++; //维护不大于 $k$ 的区间长度
32      if(i-k>=0) return a[q[l]];
33  }

```

8.6 RMQ

```

1  //A为原始数组，d[i][j]表示从i开始，长度为(1<<j)的区间最小值
2
3  int A[maxn];
4  int d[maxn][30];
5
6  void init(int A[], int len) {
7      for (int i = 0; i < len; i++) d[i][0] = A[i];
8      for (int j = 1; (1 << j) <= len; j++) {
9          for (int i = 0; i + (1 << j) - 1 < len; i++) {
10             d[i][j] = min(d[i][j-1], d[i + (1 << (j-1))][j-1]);
11         }
12     }
13 }
14
15 int query(int l, int r) {
16     int p = 0;
17     while ((1 << (p+1)) <= r-l+1) p++;
18     return min(d[l][p], d[r-(1 << p)+1][p]);
19 }

```

8.7 RMQ

```

1  //author: wavator
2  #include <algorithm>
3  #include <vector>
4
5  template <class T>
6  struct RMQ {
7      std::vector<std::vector<T>> > rmq;
8      // vector<T> rmq[20]; or T[100002][20] if need speed
9      // T kInf = numeric_limits<T>::max(); // if need return a value when the interval fake
10     void init(const std::vector<T>& a) { // 0 base
11         int n = (int)a.size(), base = 1, depth = 1;
12         while (base < n)
13             base <<= 1, ++depth;
14         rmq.assign((unsigned)depth, a);

```

```

15     for (int i = 0; i < depth - 1; ++i)
16         for (int j = 0; j < n; ++j) {
17             rmq[i + 1][j] = std::min(rmq[i][j], rmq[i][std::min(n - 1, j + (1 << i))]);
18         }
19     }
20     T q(int l, int r) { // [L, r)
21         if(l>r)return 0x3f3f3f3f;
22         int dep = 31 - __builtin_clz(r - l); // Log(b - a)
23         return min(rmq[dep][l], rmq[dep][r - (1 << dep)]);
24     }
25 };

```

8.8 线段树

```

1 //A为原始数组, sum记录区间和, Add为懒惰标记
2
3 int A[maxn], sum[maxn << 2], Add[maxn << 2];
4
5 void pushup(int rt) {
6     sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
7 }
8
9 void pushdown(int rt, int l, int r) {
10     if (Add[rt]) {
11         int mid = (l + r) >> 1;
12         Add[rt << 1] += Add[rt];
13         Add[rt << 1 | 1] += Add[rt];
14         sum[rt << 1] += (mid - l + 1)*Add[rt];
15         sum[rt << 1 | 1] += (r - mid)*Add[rt];
16         Add[rt] = 0;
17     }
18 }
19
20 void build(int l, int r, int rt) {
21     if (l == r) {
22         sum[rt] = A[l];
23         return;
24     }
25     int mid = (l + r) >> 1;
26     build(l, mid, rt << 1);
27     build(mid + 1, r, rt << 1 | 1);
28     pushup(rt);
29 }
30
31 // 区间加值
32 void update(int L, int R, int val, int l, int r, int rt) {
33     if (L <= l && R >= r) {
34         Add[rt] += val;
35         sum[rt] += (r - l + 1)*val;
36         return;
37     }
38     pushdown(rt, l, r);

```



```

39     int mid = (l + r) >> 1;
40     if (L <= mid)update(L, R, val, l, mid, rt << 1);
41     if (R > mid)update(L, R, val, mid + 1, r, rt << 1 | 1);
42     pushup(rt);
43 }
44
45 //点修改
46 void update(int index, int val, int l, int r, int rt) {
47     if (l == r) {
48         sum[rt] = val;
49         return;
50     }
51     int mid = (l + r) >> 1;
52     if (index <= mid)update(index, val, l, mid, rt << 1);
53     else update(index, val, mid + 1, r, rt << 1 | 1);
54     pushup(rt);
55 }
56
57 //区间查询
58 int query(int L, int R, int l, int r, int rt) {
59     if (L <= l && R >= r) {
60         return sum[rt];
61     }
62     pushdown(rt, l, r);
63     int mid = (l + r) >> 1;
64     int ret = 0;
65     if (L <= mid)ret += query(L, R, l, mid, rt << 1);
66     if (R > mid)ret += query(L, R, mid + 1, r, rt << 1 | 1);
67     return ret;
68 }

```

8.9 ZKW 线段树

```

1  const int maxn = 50009;
2  using ll = long long;
3
4  ll T[maxn*4];
5  int M,n;
6  void build() {
7      for(M=1;M<=n+1;M<=1);
8      for(int i=1;i<=n;i++)
9          std::cin >> T[i+M];
10     for(int i=M-1;i;i--)
11         T[i]=T[i<<1]+T[i<<1|1];
12 }
13
14 void update(int x,int val) {
15     T[x+=M]=val;    //修改
16     // T[x+=M]+=val; //加值
17     for(x>>=1;x>=1;x>>=1) {
18         T[x]=T[x<<1]+T[x<<1|1];
19     }

```

```

20 }
21
22 ll query(int l, int r) {
23     l=l+M-1, r=r+M+1;
24     ll ans=0;
25     for(; l^r^1; l>>=1, r>>=1) {
26         if(~l&1) ans+=T[l^1];
27         if(r&1) ans+=T[r^1];
28     }
29     return ans;
30 }

```

8.10 吉司机线段树

```

1 //使用方法
2 //Build(1, 1, n) 建树
3 //读入ql, qr, qt 调用函数 XXX(1, 1, n)
4 using ll = long long;
5
6 const int N = "Edit";
7 const int M = N<<2;
8
9 int mx[M], sx[M], cx[M], mn[M], sn[M], cn[M];
10 ll sum[M];
11 int ta[M];
12
13 inline void update(int x){
14     int l = x<<1, r = x<<1|1;
15     sum[x] = sum[l] + sum[r];
16     if (mx[l] == mx[r]) {
17         mx[x] = mx[l], cx[x] = cx[l] + cx[r], sx[x] = std::max(sx[l], sx[r]);
18     } else { // r>l
19         if (mx[l] > mx[r]) std::swap(l, r);
20         mx[x] = mx[r];
21         cx[x] = cx[r];
22         sx[x] = std::max(sx[r], mx[l]);
23     }
24     if (mn[l] == mn[r]) {
25         mn[x] = mn[l], cn[x] = cn[l] + cn[r], sn[x] = std::min(sn[l], sn[r]);
26     } else { // r<l
27         if (mn[l] < mn[r]) std::swap(l, r);
28         mn[x] = mn[r];
29         cn[x] = cn[r];
30         sn[x] = std::min(sn[r], mn[l]);
31     }
32 }
33
34 //建树
35 inline void Build(int x, int l, int r){
36     if (l == r) {
37         int a;
38         std::cin >> a;

```

```

39     sum[x] = mx[x] = mn[x] = a; cx[x] = cn[x] = 1;
40     sx[x] = -(1<<30); sn[x]=1<<30; ta[x]=0;
41     return;
42 }
43 int mid=(l+r)>>1;
44 Build(x<<1,l,mid);
45 Build(x<<1|1,mid+1,r);
46 update(x);
47 }
48
49 inline void _add(int x, int l, int r, int t) {
50     sum[x] += (ll)(r-l+1)*t;
51     mn[x]+=t; sn[x]+=t; mx[x]+=t; sx[x]+=t;
52     ta[x]+=t;
53 }
54
55 inline void _min(int x,int l,int r,int t){
56     sum[x] -= (ll)cx[x]*(mx[x]-t);
57     mx[x]=t; mn[x]=std::min(mn[x],t);
58     if (mn[x] == mx[x]) {
59         sum[x] = (ll)(r-l+1)*t; cx[x] = cn[x] = r-l+1; sx[x] = -(1<<30); sn[x] = 1<<30;
60     } else {
61         sn[x]=std::min(sn[x],t);
62     }
63 }
64
65 inline void _max(int x,int l,int r,int t){
66     sum[x] += (ll)cn[x]*(t-mn[x]);
67     mn[x] = t; mx[x] = std::max(mx[x], t);
68     if (mn[x] == mx[x]) {
69         sum[x]=(ll)(r-l+1)*t; cx[x] = cn[x] = r-l+1; sx[x] = -(1<<30); sn[x] = 1<<30;
70     } else {
71         sx[x] = std::max(sx[x], t);
72     }
73 }
74
75 inline void push(int x, int l, int r){
76     int mid = (l+r)>>1;
77     if (ta[x]) {
78         _add(x<<1, l, mid, ta[x]);
79         _add(x<<1|1, mid+1, r, ta[x]);
80         ta[x] = 0;
81     }
82     if (mx[x<<1]>mx[x] && sx[x<<1]<mx[x]) _min(x<<1, l, mid, mx[x]);
83     if (mx[x<<1|1]>mx[x] && sx[x<<1|1]<mx[x]) _min(x<<1|1, mid+1, r, mx[x]);
84     if (mn[x<<1]<mn[x] && sn[x<<1]>mn[x]) _max(x<<1, l, mid, mn[x]);
85     if (mn[x<<1|1]<mn[x] && sn[x<<1|1]>mn[x]) _max(x<<1|1, mid+1, r, mn[x]);
86 }
87
88 int ql, qr, qt;
89 int n;
90
91 //把一个区间[L,R] 里小于x 的数变成x
92 inline void Mmax(int x, int l, int r){

```

```

93     if (mn[x] >= qt) return;
94     if (ql<=l && r<=qr && qt<sn[x]){
95         _max(x, l, r, qt); return;
96     }
97     push(x, l, r); int mid = (l+r)>>1;
98     if (ql<=mid) Mmax(x<<1, l, mid);
99     if (qr>mid) Mmax(x<<1|1, mid+1, r);
100    update(x);
101 }
102
103 //把一个区间[L,R] 里大于x 的数变成x
104 inline void Mmin(int x,int l,int r) {
105     if (mx[x]<=qt) return;
106     if (ql<=l && r<=qr && qt>sx[x]) {
107         _min(x,l,r,qt); return;
108     }
109     push(x,l,r); int mid=(l+r)>>1;
110     if (ql<=mid) Mmin(x<<1, l, mid);
111     if (qr>mid) Mmin(x<<1|1, mid+1, r);
112     update(x);
113 }
114
115 //区间加值
116 inline void Add(int x, int l, int r) {
117     if (ql<=l && r<=qr) {
118         _add(x, l, r, qt); return;
119     }
120     push(x, l, r); int mid=(l+r)>>1;
121     if (ql<=mid) Add(x<<1, l, mid);
122     if (qr>mid) Add(x<<1|1, mid+1, r);
123     update(x);
124 }
125
126 //区间最大值
127 inline int Max(int x, int l, int r) {
128     if (ql<=l && r<=qr) return mx[x];
129     push(x, l, r);
130     int ret=-(1<<30); int mid=(l+r)>>1;
131     if (ql<=mid) ret=std::max(ret, Max(x<<1, l, mid));
132     if (qr>mid) ret=std::max(ret, Max(x<<1|1, mid+1, r));
133     return ret;
134 }
135
136 //区间最小值
137 inline int Min(int x, int l, int r) {
138     if (ql<=l && r<=qr) return mn[x];
139     push(x, l, r) ;
140     int ret=1<<30; int mid=(l+r) >>1;
141     if (ql<=mid) ret=std::min(ret, Min(x<<1, l, mid) );
142     if (qr>mid) ret=std::min(ret, Min(x<<1|1, mid+1, r) );
143     return ret;
144 }
145
146 //区间求和

```

```

147 inline ll Sum(int x, int l, int r) {
148     if (ql<=l && r<=qr) return sum[x];
149     push(x, l, r);
150     ll ret=0; int mid=(l+r) >>1;
151     if (ql<=mid) ret+=Sum(x<<1, l, mid);
152     if (qr>mid) ret+=Sum(x<<1|1, mid+1, r);
153     return ret;
154 }

```

8.11 扫描线

```

1 // 矩形面积并（交） 求并FLAG=0，求交FLAG=1
2 struct Line {
3     double l, r, h;
4     int d;
5     Line() {}
6     Line(double l, double r, double h, int d) : l(l), r(r), h(h), d(d) {}
7     bool operator < (const Line L) const {
8         return h < L.h;
9     }
10 }line[maxn << 1];
11
12 int FLAG; // 求矩形面积并 FLAG = 0，求矩形面积交 FLAG = 1
13 int Cover[maxn << 3];
14 double A[maxn << 1];
15 double Sum[maxn << 3];
16 double X1[maxn << 1], X2[maxn << 1], Y1[maxn << 1], Y2[maxn << 1];
17
18 void pushdown(int rt, int l, int r) {
19     int mid = (l + r) >> 1;
20     if (Cover[rt] != -1) {
21         Cover[rt << 1] = Cover[rt << 1 | 1] = Cover[rt];
22         Sum[rt << 1] = (Cover[rt] > FLAG ? (A[mid + 1] - A[l]) : 0);
23         Sum[rt << 1 | 1] = (Cover[rt] > FLAG ? (A[r + 1] - A[mid + 1]) : 0);
24     }
25 }
26
27 void pushup(int rt, int l, int r) {
28     if (Cover[rt << 1] == -1 || Cover[rt << 1 | 1] == -1) Cover[rt] = -1;
29     else if (Cover[rt << 1] != Cover[rt << 1 | 1]) Cover[rt] = -1;
30     else Cover[rt] = Cover[rt << 1];
31     Sum[rt] = Sum[rt << 1] + Sum[rt << 1 | 1];
32 }
33
34 void build(int l, int r, int rt) {
35     if (l == r) {
36         Cover[rt] = 0;
37         Sum[rt] = 0;
38         return;
39     }
40     int mid = (l + r) >> 1;
41     build(l, mid, rt << 1);

```

```

42     build(mid + 1, r, rt << 1 | 1);
43     pushup(rt, l, r);
44 }
45
46 void update(int L, int R, int v, int l, int r, int rt) {
47     if (L <= l && r <= R) {
48         if (Cover[rt] != -1) {
49             Cover[rt] += v;
50             Sum[rt] = (Cover[rt] > FLAG ? (A[r + 1] - A[l]) : 0);
51             return;
52         }
53     }
54     pushdown(rt, l, r);
55     int mid = (l + r) >> 1;
56     if (L <= mid) update(L, R, v, l, mid, rt << 1);
57     if (mid < R) update(L, R, v, mid + 1, r, rt << 1 | 1);
58     pushup(rt, l, r);
59 }
60
61 int find(double key, int n, double d[]) {
62     int l = 1, r = n;
63     while (r >= l) {
64         int mid = (r + l) >> 1;
65         if (d[mid] == key) return mid;
66         else if (d[mid] > key) r = mid - 1;
67         else l = mid + 1;
68     }
69     return -1;
70 }
71
72 int init(int n) {
73     int N = 0;
74     for (int i = 1; i <= n; i++) {
75         A[++N] = X1[i];
76         line[N] = Line(X1[i], X2[i], Y1[i], 1);
77         A[++N] = X2[i];
78         line[N] = Line(X1[i], X2[i], Y2[i], -1);
79     }
80     sort(A + 1, A + N + 1);
81     sort(line + 1, line + N + 1);
82     int k = 1;
83     for (int i = 2; i <= N; i++)
84         if (A[i] != A[i - 1])
85             A[++k] = A[i];
86     build(1, k - 1, 1);
87     return k;
88 }
89
90 double query(int n, int k) {
91     double ret = 0;
92     for (int i = 1; i < n; i++) {
93         int l = find(line[i].l, k, A);
94         int r = find(line[i].r, k, A) - 1;
95         if (l <= r) update(l, r, line[i].d, 1, k - 1, 1);

```

```

96         ret += Sum[1] * (line[i + 1].h - line[i].h);
97     }
98     return ret;
99 }
100 /*
101 int main()
102 {
103     int n, T;
104     scanf("%d", &T);
105     while (T--) {
106         scanf("%d", &n);
107         for (int i = 1; i <= n; i++)
108             scanf("%lf%lf%lf%lf", &X1[i], &Y1[i], &X2[i], &Y2[i]);
109         int k = init(n);
110         double ans = query(n << 1, k);
111         printf("%.2Lf\n", ans);
112     }
113 }
114 */
115
116 //=====
117
118 // 矩形周长并
119 int Sum[maxn << 3], cnt[maxn << 3], vNum[maxn << 3];
120 bool lbd[maxn << 3], rbd[maxn << 3];
121 double X1[maxn << 1], X2[maxn << 1], Y1[maxn << 1], Y2[maxn << 1];
122 double A[maxn << 1];
123
124 struct Line {
125     double l, r, h;
126     int label;
127     Line() {}
128     Line(double l, double r, double h, int label) :l(l), r(r), h(h), label(label) {}
129     bool operator < (const Line L) const {
130         return h < L.h;
131     }
132 }line[maxn << 1];
133
134 void pushup(int l, int r, int rt) {
135     if (cnt[rt]) {
136         lbd[rt] = rbd[rt] = true;
137         Sum[rt] = A[r + 1] - A[l];
138         vNum[rt] = 2;
139     }
140     else if (l == r) Sum[rt] = vNum[rt] = lbd[rt] = rbd[rt] = 0;
141     else {
142         lbd[rt] = lbd[rt << 1];
143         rbd[rt] = rbd[rt << 1] | 1;
144         Sum[rt] = Sum[rt << 1] + Sum[rt << 1 | 1];
145         vNum[rt] = vNum[rt << 1] + vNum[rt << 1 | 1];
146         if (rbd[rt << 1] && lbd[rt << 1 | 1]) vNum[rt] -= 2;
147     }
148 }
149

```

```
150 void update(int L, int R, int v, int l, int r, int rt) {
151     if (L <= l && r <= R) {
152         cnt[rt] += v;
153         pushup(l, r, rt);
154         return;
155     }
156     int mid = (l + r) >> 1;
157     if (L <= mid) update(L, R, v, l, mid, rt << 1);
158     if (R > mid) update(L, R, v, mid + 1, r, rt << 1 | 1);
159     pushup(l, r, rt);
160 }
161
162 int find(double key, int n, double d[]) {
163     int l = 1, r = n;
164     while (r >= l) {
165         int mid = (r + l) >> 1;
166         if (d[mid] == key) return mid;
167         else if (d[mid] > key) r = mid - 1;
168         else l = mid + 1;
169     }
170     return -1;
171 }
172
173 int init(int n) {
174     for (int i = 1; i <= n; i++) {
175         A[i] = X1[i]; A[i + n] = X2[i];
176         line[i].l = X1[i]; line[i].r = X2[i];
177         line[i].h = Y1[i]; line[i].label = 1;
178         line[i + n].l = X1[i]; line[i + n].r = X2[i];
179         line[i + n].h = Y2[i]; line[i + n].label = -1;
180     }
181     n <<= 1;
182     int k = 1;
183     sort(A + 1, A + n + 1);
184     sort(line + 1, line + n + 1);
185     for (int i = 2; i <= n; i++)
186         if (A[i] != A[i - 1])
187             A[++k] = A[i];
188     return k;
189 }
190
191 double query(int n, int k) {
192     double ret = 0, lst = 0;
193     for (int i = 1; i <= n; i++) {
194         if (line[i].l < line[i].r) {
195             int l = find(line[i].l, k, A);
196             int r = find(line[i].r, k, A);
197             update(l, r - 1, line[i].label, 1, k - 1, 1);
198         }
199         ret += vNum[1] * (line[i + 1].h - line[i].h);
200         ret += abs(Sum[1] - lst);
201         lst = Sum[1];
202     }
203     return ret;
```



```

204 }
205 /*
206 int main()
207 {
208     int n;
209     while (~scanf("%d", &n)) {
210         for (int i = 1; i <= n; i++)
211             scanf("%Lf%Lf%Lf%Lf", &X1[i], &Y1[i], &X2[i], &Y2[i]);
212         int k = init(n);
213         double ans = query(n << 1, k);
214         printf("%Lf\n", ans);
215     }
216     return 0;
217 }
218 */

```

8.12 固定大小矩形最大点覆盖

```

1 // 扫描线 求 矩形最大点覆盖
2 struct Line {
3     ll x, y1, y2, k;    // k 为矩形权值
4     bool operator < (const Line nod) const {
5         return x < nod.x || (x == nod.x && k < nod.k);
6     }
7 }line[maxn];
8 struct segTree {
9     ll ma, l, r, lazy;
10 }tree[maxn << 2];
11 ll yy[maxn];
12 int cnt, ycnt;
13 void pushup(int rt) {
14     tree[rt].ma = max(tree[rt << 1].ma, tree[rt << 1 | 1].ma) + tree[rt].lazy;
15 }
16 void build(int l, int r, int rt) {
17     tree[rt].ma = tree[rt].lazy = 0;
18     tree[rt].l = yy[l], tree[rt].r = yy[r];
19     if (r - l == 1) return;
20     int mid = (l + r) >> 1;
21     build(l, mid, rt << 1);
22     build(mid, r, rt << 1 | 1);
23     pushup(rt);
24 }
25 void update(ll L, ll R, ll w, int rt) {
26     if (tree[rt].l >= L && tree[rt].r <= R) {
27         tree[rt].lazy += w;
28         tree[rt].ma += w;
29         return;
30     }
31     if (L < tree[rt << 1].r)
32         update(L, min(R, tree[rt << 1].r), w, rt << 1);
33     if (R > tree[rt << 1 | 1].l)
34         update(max(tree[rt << 1 | 1].l, L), R, w, rt << 1 | 1);

```

```

35     pushup(rt);
36 }
37 int main()
38 {
39     ll n, W, H, x, y, w, ma;
40     while (~scanf("%lld%lld%lld", &n, &W, &H)) {
41         cnt = 0; ycnt = 1; ma = 0;
42         for (int i = 1; i <= n; i++) {
43             scanf("%lld%lld%lld", &x, &y, &w);
44             line[cnt].x = x; line[cnt].y1 = y; line[cnt].y2 = y + H;
45             line[cnt++].k = w;
46             line[cnt].x = x + W; line[cnt].y1 = y; line[cnt].y2 = y + H;
47             line[cnt++].k = -w;
48             yy[ycnt++] = y;
49             yy[ycnt++] = y + H;
50         }
51         sort(yy + 1, yy + ycnt);
52         ycnt = unique(yy + 1, yy + ycnt) - (yy + 1);
53         sort(line, line + cnt);
54         build(1, ycnt, 1);
55         for (int i = 0; i < cnt; i++) {
56             update(line[i].y1, line[i].y2, line[i].k, 1);
57             if (line[i].k > 0) ma = max(ma, tree[1].ma);
58         }
59         printf("%lld\n", ma);
60     }
61     return 0;
62 }

```

8.13 二维线段树 (单点更新区间最值)

```

1 // 二维线段树单点更新+区间最值 树套树实现
2 int n;
3 int y2rt[maxn], x2rt[maxn];
4
5 struct Nodey {
6     int l, r;
7     int Max, Min;
8 };
9
10 struct Nodex {
11     int l, r;
12     Nodey nodey[maxn << 2];
13
14     void build(int l, int r, int rt) {
15         nodey[rt].l = l;
16         nodey[rt].r = r;
17         nodey[rt].Max = -inf;
18         nodey[rt].Min = inf;
19         if (l == r) {
20             y2rt[l] = rt;
21             return;

```

```
22     }
23     int mid = (l + r) >> 1;
24     build(l, mid, rt << 1);
25     build(mid + 1, r, rt << 1 | 1);
26 }
27
28 int queryMin(int rt, int L, int R) {
29     if (nodey[rt].l == L && nodey[rt].r == R)
30         return nodey[rt].Min;
31     int mid = (nodey[rt].l + nodey[rt].r) >> 1;
32     if (R <= mid) return queryMin(rt << 1, L, R);
33     else if (L > mid) return queryMin(rt << 1 | 1, L, R);
34     else return min(queryMin(rt << 1, L, mid), queryMin(rt << 1 | 1, mid + 1, R));
35 }
36
37 int queryMax(int rt, int L, int R) {
38     if (nodey[rt].l == L && nodey[rt].r == R)
39         return nodey[rt].Max;
40     int mid = (nodey[rt].l + nodey[rt].r) >> 1;
41     if (R <= mid) return queryMax(rt << 1, L, R);
42     else if (L > mid) return queryMax((rt << 1) | 1, L, R);
43     else return max(queryMax(rt << 1, L, mid), queryMax((rt << 1) | 1, mid + 1, R));
44 }
45 }nodex[maxn << 2];
46
47 void build(int l, int r, int rt) {
48     nodex[rt].l = l;
49     nodex[rt].r = r;
50     nodex[rt].build(1, n, 1);
51     if (l == r) {
52         x2rt[l] = rt;
53         return;
54     }
55     int mid = (l + r) >> 1;
56     build(l, mid, rt << 1);
57     build(mid + 1, r, rt << 1 | 1);
58 }
59
60 // 点修改
61 void update(int x, int y, int val) {
62     int rtx = x2rt[x];
63     int rty = y2rt[y];
64     nodex[rtx].nodey[rty].Min = nodex[rtx].nodey[rty].Max = val;
65     for (int i = rtx; i; i >>= 1) {
66         for (int j = rty; j; j >>= 1) {
67             if (i == rtx && j == rty) continue;
68             if (j == rty) {
69                 nodex[i].nodey[j].Min = min(nodex[i << 1].nodey[j].Min, nodex[(i << 1) | 1].
                    nodey[j].Min);
70                 nodex[i].nodey[j].Max = max(nodex[i << 1].nodey[j].Max, nodex[(i << 1) | 1].
                    nodey[j].Max);
71             }
72             else {
```

```

73         nodex[i].nodey[j].Min = min(nodex[i].nodey[j << 1].Min, nodex[i].nodey[(j <<
74         1) | 1].Min);
75         nodex[i].nodey[j].Max = max(nodex[i].nodey[j << 1].Max, nodex[i].nodey[(j <<
76         1) | 1].Max);
77     }
78 }
79
80 int queryMin(int rt, int x1, int x2, int y1, int y2) {
81     if (nodex[rt].l == x1 && nodex[rt].r == x2)
82         return nodex[rt].queryMin(1, y1, y2);
83     int mid = (nodex[rt].l + nodex[rt].r) >> 1;
84     if (x2 <= mid) return queryMin(rt << 1, x1, x2, y1, y2);
85     else if (x1 > mid) return queryMin(rt << 1 | 1, x1, x2, y1, y2);
86     else return min(queryMin(rt << 1, x1, mid, y1, y2), queryMin(rt << 1 | 1, mid + 1, x2, y1,
87     y2));
88 }
89
90 int queryMax(int rt, int x1, int x2, int y1, int y2) {
91     if (nodex[rt].l == x1 && nodex[rt].r == x2)
92         return nodex[rt].queryMax(1, y1, y2);
93     int mid = (nodex[rt].l + nodex[rt].r) >> 1;
94     if (x2 <= mid) return queryMax(rt << 1, x1, x2, y1, y2);
95     else if (x1 > mid) return queryMax(rt << 1 | 1, x1, x2, y1, y2);
96     else return max(queryMax(rt << 1, x1, mid, y1, y2), queryMax(rt << 1 | 1, mid + 1, x2, y1,
97     y2));
98 }

```

8.14 二维线段树 (区间加值单点查询)

```

1 // 二维线段树区间加值+单点查询 树套树实现
2 int n;
3 int x2rt[maxn], y2rt[maxn];
4
5 struct Nodey {
6     int l, r;
7     int val;
8 };
9
10 struct Nodex {
11     int l, r;
12     Nodey nodey[maxn << 2];
13
14     void build(int l, int r, int rt) {
15         nodey[rt].l = l;
16         nodey[rt].r = r;
17         nodey[rt].val = 0;
18         if (l == r) {
19             y2rt[l] = rt;
20             return;
21         }

```

```

22     int mid = (l + r) >> 1;
23     build(l, mid, rt << 1);
24     build(mid + 1, r, rt << 1 | 1);
25 }
26
27 void addVal(int rt, int L, int R, int val) {
28     if (nodey[rt].l == L && nodey[rt].r == R) {
29         nodey[rt].val += val;
30         return;
31     }
32     int mid = (nodey[rt].l + nodey[rt].r) >> 1;
33     if (R <= mid) addVal(rt << 1, L, R, val);
34     else if (L > mid) addVal(rt << 1 | 1, L, R, val);
35     else {
36         addVal(rt << 1, L, mid, val);
37         addVal(rt << 1 | 1, mid + 1, R, val);
38     }
39 }
40 }nodex[maxn << 2];
41
42 void build(int l, int r, int rt) {
43     nodex[rt].l = l;
44     nodex[rt].r = r;
45     nodex[rt].build(1, n, 1);
46     if (l == r) {
47         x2rt[l] = rt;
48         return;
49     }
50     int mid = (l + r) >> 1;
51     build(l, mid, rt << 1);
52     build(mid + 1, r, rt << 1 | 1);
53 }
54
55 void addVal(int rt, int x1, int x2, int y1, int y2, int val) {
56     if (nodex[rt].l == x1 && nodex[rt].r == x2) {
57         nodex[rt].addVal(1, y1, y2, val);
58         return;
59     }
60     int mid = (nodex[rt].l + nodex[rt].r) >> 1;
61     if (x2 <= mid) addVal(rt << 1, x1, x2, y1, y2, val);
62     else if (x1 > mid) addVal(rt << 1 | 1, x1, x2, y1, y2, val);
63     else {
64         addVal(rt << 1, x1, mid, y1, y2, val);
65         addVal(rt << 1 | 1, mid + 1, x2, y1, y2, val);
66     }
67 }
68
69 int getVal(int x, int y) {
70     int ret = 0;
71     for (int i = x2rt[x]; i; i >>= 1)
72         for (int j = y2rt[y]; j; j >>= 1)
73             ret += nodex[i].nodey[j].val;
74     return ret;
75 }

```

8.15 主席树

```
1 // 主席树 支持查询 $[L, r]$ 区间第 $k$ 大, 以及区间内不重复数字个数
2 //  $M = \text{maxn} * 30$ ;
3 int n, q, m, tot; //  $n$ 为数组大小,  $m$ 为离散化后数组大小
4 int A[maxn], T[maxn]; //  $A$ 为原数组,  $T$ 为离散化数组
5 int tree[M], lson[M], rson[M], Cnt[M]; //  $\text{Cnt}[i]$ 表示节点 $i$ 的子树包含数字的总数
6
7 void Init_hash() {
8     for (int i = 1; i <= n; i++) T[i] = A[i];
9     sort(T + 1, T + n + 1);
10    m = unique(T + 1, T + n + 1) - T - 1;
11 }
12
13 inline int Hash(int x) { return lower_bound(T + 1, T + m + 1, x) - T; }
14
15 int build(int l, int r) {
16     int root = tot++;
17     Cnt[root] = 0;
18     if (l != r) {
19         int mid = (l + r) >> 1;
20         lson[root] = build(l, mid);
21         rson[root] = build(mid + 1, r);
22     }
23     return root;
24 }
25
26 int update(int root, int pos, int val) {
27     int newroot = tot++, tmp = newroot;
28     Cnt[newroot] = Cnt[root] + val;
29     int l = 1, r = m;
30     while (l < r) {
31         int mid = (l + r) >> 1;
32         if (pos <= mid) {
33             lson[newroot] = tot++; rson[newroot] = rson[root];
34             newroot = lson[newroot]; root = lson[root];
35             r = mid;
36         }
37         else {
38             rson[newroot] = tot++; lson[newroot] = lson[root];
39             newroot = rson[newroot]; root = rson[root];
40             l = mid + 1;
41         }
42     }
43     Cnt[newroot] = Cnt[root] + val;
44     return tmp;
45 }
46
47 void init() { // 查询 $L \sim r$ 第 $k$ 大
48     Init_hash();
49     tree[0] = build(1, m);
```

```

50     for (int i = 1; i <= n; i++) {
51         int pos = Hash(A[i]);
52         tree[i] = update(tree[i - 1], pos, 1);
53     }
54 }
55
56 int query(int lrt, int rrt, int k) {    // 查询L~r第k大: T[query(tree[l - 1], tree[r], k)]
57     int l = 1, r = m;
58     while (l < r) {
59         int mid = (l + r) >> 1;
60         if (Cnt[lson[rrt]] - Cnt[lson[lrt]] >= k) {
61             r = mid;
62             lrt = lson[lrt];
63             rrt = lson[rrt];
64         }
65         else {
66             l = mid + 1;
67             k -= Cnt[lson[rrt]] - Cnt[lson[lrt]];
68             lrt = rson[lrt];
69             rrt = rson[rrt];
70         }
71     }
72     return l;
73 }
74
75 void init() {    // 查询L~r内不重复数字个数
76     tree[0] = build(1, n);
77     map<int, int>mp;
78     for (int i = 1; i <= n; i++) {
79         if (mp.find(A[i]) == mp.end())
80             tree[i] = update(tree[i - 1], i, 1);
81         else {
82             int tmp = update(tree[i - 1], mp[A[i]], -1);
83             tree[i] = update(tmp, i, 1);
84         }
85         mp[A[i]] = i;
86     }
87 }
88
89 int query(int root, int pos) {    // 查询L~r内不重复数字个数: query(tree[r], L)
90     int ret = 0;
91     int l = 1, r = n;
92     while (pos > 1) {
93         int mid = (l + r) >> 1;
94         if (pos <= mid) {
95             ret += Cnt[rson[root]];
96             root = lson[root];
97             r = mid;
98         }
99         else {
100             root = rson[root];
101             l = mid + 1;
102         }
103     }

```

```
104     return ret + Cnt[root];
105 }
```

8.16 主席树动态 k 大

```
1 // 主席树求  $[l, r]$  第  $k$  大, 可单点修改 使用树状数组套主席树在线操作, 树状数组维护改变量
2 //  $M = \text{maxn} * 40$ ;
3 int n, q, m, tot;
4 int A[maxn], T[maxn];
5 int tree[maxn], lson[M], rson[M], Cnt[M];
6 int Ntree[maxn], use[maxn]; //  $Ntree[i]$  表示动态第  $i$  棵树的树根,  $use[i]$  表示第  $i$  个树根是谁在使用
7
8 struct Query {
9     int kind;
10    int l, r, k;
11 } query[10005];
12
13 void Init_hash(int k) {
14     sort(T, T + k);
15     m = unique(T, T + k) - T;
16 }
17
18 int Hash(int x) { return lower_bound(T, T + m, x) - T; }
19
20 int build(int l, int r) {
21     int root = tot++;
22     Cnt[root] = 0;
23     if (l != r) {
24         int mid = (l + r) >> 1;
25         lson[root] = build(l, mid);
26         rson[root] = build(mid + 1, r);
27     }
28     return root;
29 }
30
31 int update(int root, int pos, int val) {
32     int newroot = tot++, tmp = newroot;
33     int l = 0, r = m - 1;
34     Cnt[newroot] = Cnt[root] + val;
35     while (l < r) {
36         int mid = (l + r) >> 1;
37         if (pos <= mid) {
38             lson[newroot] = tot++; rson[newroot] = rson[root];
39             newroot = lson[newroot]; root = lson[root];
40             r = mid;
41         }
42         else {
43             rson[newroot] = tot++; lson[newroot] = lson[root];
44             newroot = rson[newroot]; root = rson[root];
45             l = mid + 1;
46         }
47     }
48     Cnt[newroot] = Cnt[root] + val;
```



```
48     }
49     return tmp;
50 }
51
52 inline int lowbit(int x) { return x & (-x); }
53
54 int sum(int x) {
55     int ret = 0;
56     while (x > 0) {
57         ret += Cnt[lson[use[x]]];
58         x -= lowbit(x);
59     }
60     return ret;
61 }
62
63 void Modify(int x, int pos, int val) {
64     while (x <= n) {
65         Ntree[x] = update(Ntree[x], pos, val);
66         x += lowbit(x);
67     }
68 }
69
70 int Query(int left, int right, int k) {
71     int lrt = tree[left - 1];
72     int rrt = tree[right];
73     int l = 0, r = m - 1;
74     for (int i = left - 1; i; i -= lowbit(i)) use[i] = Ntree[i];
75     for (int i = right; i; i -= lowbit(i)) use[i] = Ntree[i];
76     while (l < r) {
77         int mid = (l + r) >> 1;
78         // sum(right) - sum(left - 1)为改变量, Cnt[lson[rrt]] - Cnt[lson[lrt]]为基础差值
79         int tmp = sum(right) - sum(left - 1) + Cnt[lson[rrt]] - Cnt[lson[lrt]];
80         if (tmp >= k) {
81             r = mid;
82             for (int i = left - 1; i; i -= lowbit(i))
83                 use[i] = lson[use[i]];
84             for (int i = right; i; i -= lowbit(i))
85                 use[i] = lson[use[i]];
86             lrt = lson[lrt];
87             rrt = lson[rrt];
88         }
89         else {
90             l = mid + 1;
91             k -= tmp;
92             for (int i = left - 1; i; i -= lowbit(i))
93                 use[i] = rson[use[i]];
94             for (int i = right; i; i -= lowbit(i))
95                 use[i] = rson[use[i]];
96             lrt = rson[lrt];
97             rrt = rson[rrt];
98         }
99     }
100     return l;
101 }
```

```

102
103 int main()
104 {
105     int Tcase;
106     char op[10];
107     scanf("%d", &Tcase);
108     while (Tcase--) {
109         scanf("%d%d", &n, &q);
110         tot = 0; m = 0;
111         for (int i = 1; i <= n; i++) {
112             scanf("%d", &A[i]);
113             T[m++] = A[i];
114         }
115         for (int i = 0; i < q; i++) {
116             scanf("%s", op);
117             if (op[0] == 'Q') {
118                 query[i].kind = 0;
119                 scanf("%d%d%d", &query[i].l, &query[i].r, &query[i].k);
120             }
121             else {
122                 query[i].kind = 1;
123                 scanf("%d%d", &query[i].l, &query[i].r);
124                 T[m++] = query[i].r;
125             }
126         }
127         Init_hash(m);
128         tree[0] = build(0, m - 1);
129         for (int i = 1; i <= n; i++)
130             tree[i] = update(tree[i - 1], Hash(A[i]), 1);
131         for (int i = 1; i <= n; i++) Ntree[i] = tree[0];
132         for (int i = 0; i < q; i++) {
133             if (query[i].kind == 0)
134                 printf("%d\n", T[Query(query[i].l, query[i].r, query[i].k)]);
135             else {
136                 Modify(query[i].l, Hash(A[query[i].l]), -1);
137                 Modify(query[i].l, Hash(query[i].r), 1);
138                 A[query[i].l] = query[i].r;
139             }
140         }
141     }
142     return 0;
143 }

```

8.17 Treap 树

```

1 typedef int value;
2
3 enum { LEFT, RIGHT };
4 struct node {
5     int size, priority;
6     value x, subtree;
7     node *child[2];

```

```

8     node(const value &x): size(1), x(x), subtree(x) {
9         priority = rand();
10        child[0] = child[1] = nullptr;
11    }
12 };
13
14 inline int size(const node *a) { return a == nullptr ? 0 : a->size; }
15
16 inline void update(node *a) {
17     if (a == nullptr) return;
18     a->size = size(a->child[0]) + size(a->child[1]) + 1;
19     a->subtree = a->x;
20     if (a->child[LEFT] != nullptr) a->subtree = a->child[LEFT]->subtree + a->subtree;
21     if (a->child[RIGHT] != nullptr) a->subtree = a->subtree + a->child[RIGHT]->subtree;
22 }
23
24 node *rotate(node *a, bool d) {
25     node *b = a->child[d];
26     a->child[d] = b->child[!d];
27     b->child[!d] = a;
28     update(a); update(b);
29     return b;
30 }
31
32 node *insert(node *a, int index, const value &x) {
33     if (a == nullptr && index == 0) return new node(x);
34     int middle = size(a->child[LEFT]);
35     bool dir = index > middle;
36     if (!dir) a->child[LEFT] = insert(a->child[LEFT], index, x);
37     else      a->child[RIGHT] = insert(a->child[RIGHT], index - middle - 1, x);
38     update(a);
39     if (a->priority > a->child[dir]->priority) a = rotate(a, dir);
40     return a;
41 }
42
43 node *erase(node *a, int index) {
44     assert(a != nullptr);
45     int middle = size(a->child[LEFT]);
46     if (index == middle) {
47         if (a->child[LEFT] == nullptr && a->child[RIGHT] == nullptr) {
48             delete a;
49             return nullptr;
50         } else if (a->child[LEFT] == nullptr) a = rotate(a, RIGHT);
51         else if (a->child[RIGHT] == nullptr) a = rotate(a, LEFT);
52         else a = rotate(a, a->child[LEFT]->priority < a->child[RIGHT]->priority);
53         a = erase(a, index);
54     } else {
55         bool dir = index > middle;
56         if (!dir) a->child[LEFT] = erase(a->child[LEFT], index);
57         else      a->child[RIGHT] = erase(a->child[RIGHT], index - middle - 1);
58     }
59     update(a);
60     return a;
61 }

```

```

62
63 void modify(node *a, int index, const value &x) {
64     assert(a != nullptr);
65     int middle = size(a->child[LEFT]);
66     if (index == middle) a->x = x;
67     else {
68         bool dir = index > middle;
69         if (!dir) modify(a->child[LEFT], index, x);
70         else      modify(a->child[RIGHT], index - middle - 1, x);
71     }
72     update(a);
73 }
74
75 value query(node *a, int l, int r) {
76     assert(a != nullptr);
77     if (l <= 0 && size(a) - 1 <= r) return a->subtree;
78     int middle = size(a->child[LEFT]);
79     if (r < middle) return query(a->child[LEFT], l, r);
80     if (middle < l) return query(a->child[RIGHT], l - middle - 1, r - middle - 1);
81     value res = a->x;
82     if (l < middle && a->child[LEFT] != nullptr)
83         res = query(a->child[LEFT], l, r) + res;
84     if (middle < r && a->child[RIGHT] != nullptr)
85         res = res + query(a->child[RIGHT], l - middle - 1, r - middle - 1);
86     return res;
87 }

```

8.18 函数式 Treap

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <cmath>
5  #include <algorithm>
6  #include <cstdlib>
7  #include <ctime>
8  using namespace std;
9  const int MAXN=100001;
10 static void read(int &n) {
11     char c='+';int x=0;bool flag=0;
12     while(c<'0' || c>'9'){c=getchar();if(c=='-')flag=1;}
13     while(c>='0'&&c<='9'){x=(x<<1)+(x<<3)+(c-48);c=getchar();}
14     flag==1?n=-x:n=x;
15 }
16 int ch[MAXN][3];// 0左孩子 1右孩子
17 int val[MAXN];// 每一个点的权值
18 int pri[MAXN];// 随机生成的附件权值
19 int siz[MAXN];// 以i为节点的树的节点数量
20 int sz;// 总结点的数量
21 void update(int x) {
22     siz[x]=1+siz[ch[x][0]]+siz[ch[x][1]];
23 }

```

```
24 int new_node(int v) {
25     siz[++sz]=1; // 新开辟一个节点
26     val[sz]=v;
27     pri[sz]=rand();
28     return sz;
29 }
30 int merge(int x,int y) { // 合并
31     if(!x||!y) return x+y; // x和y中必定有一个是0
32     if(pri[x]<pri[y]) // 把x加到左边的树上
33     {
34         ch[x][1]=merge(ch[x][1],y); // 不懂的看GIF图
35         update(x);
36         return x;
37     }
38     else
39     {
40         ch[y][0]=merge(x,ch[y][0]);
41         update(y);
42         return y;
43     }
44 }
45 void split(int now,int k,int &x,int &y) {
46     if(!now) x=y=0; // 到达叶子节点
47     else {
48         if(val[now]<=k) // 分离右子树
49             x=now,split(ch[now][1],k,ch[now][1],y);
50         else
51             y=now,split(ch[now][0],k,x,ch[now][0]);
52         update(now);
53     }
54 }
55 int kth(int now,int k) { // 查询排名
56     while(1) {
57         if(k<=siz[ch[now][0]])
58             now=ch[now][0]; // 在左子树中, 且数量小于左子树的大小, 迭代寻找
59         else if(k==siz[ch[now][0]]+1)
60             return now; // 找到了
61         else
62             k-=siz[ch[now][0]]+1,now=ch[now][1]; // 去右子树找
63     }
64 }
65 int main() {
66     srand((unsigned)time(NULL));
67     int n;
68     read(n);
69     int root=0,x,y,z;
70     for(int i=1;i<=n;i++) {
71         int how,a;
72         read(how);read(a);
73         if(how==1) { // 插入
74             split(root,a,x,y);
75             root=merge(merge(x,new_node(a)),y);
76         }
77         else if(how==2) { // 删除x
```

```

78     {
79         split(root,a,x,z);
80         split(x,a-1,x,y);
81         y=merge(ch[y][0],ch[y][1]);
82         root=merge(merge(x,y),z);
83     }
84     else if(how==3) { // 查询x的排名
85         split(root,a-1,x,y);
86         printf("%d\n",siz[x]+1);
87         root=merge(x,y);
88     }
89     else if(how==4) { // 查询排名为x的数
90         printf("%d\n",val[kth(root,a)]);
91     }
92     else if(how==5) { // 求x的前驱
93         split(root,a-1,x,y);
94         printf("%d\n",val[kth(x,siz[x])]);
95         root=merge(x,y);
96     }
97     else if(how==6) { // 求x的后继
98         split(root,a,x,y);
99         printf("%d\n",val[kth(y,1)]);
100        root=merge(x,y);
101    }
102 }
103 return 0;
104 }

```

8.19 Splay 树

```

1  // splay tree. HDU 3726: 插入、删除、合并
2
3  const int MAXN = 20010;
4  struct Node;
5  Node* null;
6  struct Node {
7      Node *ch[2], *fa; // 指向儿子和父亲结点
8      int size, key;
9      Node() {
10         ch[0] = ch[1] = fa = null;
11     }
12     inline void setc(Node* p, int d) {
13         ch[d] = p;
14         p->fa = this;
15     }
16     inline bool d() {
17         return fa->ch[1] == this;
18     }
19     void push_up() {
20         size = ch[0]->size + ch[1]->size + 1;
21     }
22     void clear() {

```

```
23     size = 1;
24     ch[0] = ch[1] = fa = null;
25 }
26 inline bool isroot() {
27     return fa == null || this != fa->ch[0] && this != fa->ch[1];
28 }
29 };
30
31 inline void rotate(Node* x) {
32     Node *f = x->fa, *ff = x->fa->fa;
33     int c = x->d(), cc = f->d();
34     f->setc(x->ch[!c], c);
35     x->setc(f, !c);
36     if (ff->ch[cc] == f) ff->setc(x, cc);
37     else x->fa = ff;
38     f->push_up();
39 }
40
41 inline void splay(Node* &root, Node* x, Node* goal) {
42     while (x->fa != goal) {
43         if (x->fa->fa == goal) rotate(x);
44         else {
45             bool f = x->fa->d();
46             x->d() == f ? rotate(x->fa) : rotate(x);
47             rotate(x);
48         }
49     }
50     x->push_up();
51     if (goal == null) root = x;
52 }
53
54 //找到 r 子树里面的第 k 个
55 Node* get_kth(Node* r, int k) {
56     Node* x = r;
57     while (x->ch[0]->size + 1 != k) {
58         if (k < x->ch[0]->size + 1) x = x->ch[0];
59         else {
60             k -= x->ch[0]->size + 1;
61             x = x->ch[1];
62         }
63     }
64     return x;
65 }
66
67
68 void erase(Node* &root, Node* x) {
69     splay(root, x, null);
70     Node* t = root;
71     if (t->ch[1] != null) {
72         root = t->ch[1];
73         splay(root, get_kth(root, 1), null);
74         root->setc(t->ch[0], 0);
75     }
76     else {
```

```
77     root = root->ch[0];
78 }
79 root->fa = null;
80 if (root != null) root->push_up();
81 }
82
83 void insert(Node* &root, Node* x) {
84     if (root == null) {
85         root = x;
86         return;
87     }
88     Node* now = root;
89     Node* pre = root->fa;
90     while (now != null) {
91         pre = now;
92         now = now->ch[x->key >= now->key];
93     }
94     x->clear();
95     pre->setc(x, x->key >= pre->key);
96     splay(root, x, null);
97 }
98
99 void merge(Node* &A, Node* B) {
100     if (A->size <= B->size) swap(A, B);
101     queue<Node*> Q;
102     Q.push(B);
103     while (!Q.empty()) {
104         Node* fr = Q.front();
105         Q.pop();
106         if (fr->ch[0] != null) Q.push(fr->ch[0]);
107         if (fr->ch[1] != null) Q.push(fr->ch[1]);
108         fr->clear();
109         insert(A, fr);
110     }
111 }
112
113 Node pool[MAXN], *tail;
114
115 struct Edge {
116     int u, v;
117 } edge[60010];
118 int a[MAXN];
119 bool del[60010];
120 struct QUERY {
121     char op[10];
122     int u, v;
123 } query[500010];
124 int y[500010];
125
126 Node* node[MAXN];
127 Node* root[MAXN];
128 int F[MAXN];
129 int find(int x) {
130     if (F[x] == -1) return x;
```



```
131     return F[x] = find(F[x]);
132 }
133
134 void debug(Node *root) {
135     if (root == null) return;
136     debug(root->ch[0]);
137     printf("size:%d, key=%d\n", root->size, root->key);
138     debug(root->ch[1]);
139 }
140
141 int main()
142 {
143     int n, m;
144     int iCase = 0;
145     while (scanf("%d%d", &n, &m) == 2) {
146         if (n == 0 && m == 0) break;
147         iCase++;
148         memset(F, -1, sizeof(F));
149         tail = pool;
150         null = tail++;
151         null->size = 0; null->ch[0] = null->ch[1] = null->fa = null;
152         null->key = 0;
153         for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
154         for (int i = 0; i < m; i++) {
155             scanf("%d%d", &edge[i].u, &edge[i].v);
156             del[i] = false;
157         }
158         int Q = 0;
159         while (1) {
160             scanf("%s", &query[Q].op);
161             if (query[Q].op[0] == 'E') break;
162             if (query[Q].op[0] == 'D') {
163                 scanf("%d", &query[Q].u);
164                 query[Q].u--;
165                 del[query[Q].u] = true;
166             }
167             else if (query[Q].op[0] == 'Q') {
168                 scanf("%d%d", &query[Q].u, &query[Q].v);
169             }
170             else {
171                 scanf("%d%d", &query[Q].u, &query[Q].v);
172                 y[Q] = a[query[Q].u];
173                 a[query[Q].u] = query[Q].v;
174             }
175             Q++;
176         }
177         for (int i = 1; i <= n; i++) {
178             node[i] = tail++;
179             node[i]->clear();
180             node[i]->key = a[i];
181             root[i] = node[i];
182         }
183         for (int i = 0; i < m; i++)
184             if (!del[i]) {
```

```

185         int u = edge[i].u;
186         int v = edge[i].v;
187         int t1 = find(u);
188         int t2 = find(v);
189         if (t1 == t2) continue;
190         F[t2] = t1;
191         merge(root[t1], root[t2]);
192     }
193     vector<int> ans;
194     for (int i = Q - 1; i >= 0; i--) {
195         if (query[i].op[0] == 'D') {
196             int u = edge[query[i].u].u;
197             int v = edge[query[i].u].v;
198             int t1 = find(u);
199             int t2 = find(v);
200             if (t1 == t2) continue;
201             F[t2] = t1;
202             merge(root[t1], root[t2]);
203         }
204         else if (query[i].op[0] == 'Q') {
205             int u = query[i].u;
206             int k = query[i].v;
207             u = find(u);
208             if (k <= 0 || k > root[u]->size) {
209                 ans.push_back(0);
210             }
211             else {
212                 k = root[u]->size - k + 1;
213                 Node* p = get_kth(root[u], k);
214                 ans.push_back(p->key);
215             }
216         }
217         else {
218             int u = query[i].u;
219             int t1 = find(u);
220             Node* p = node[u];
221             erase(root[t1], p);
222             p->clear();
223             p->key = y[i];
224             a[u] = y[i];
225             insert(root[t1], p);
226         }
227     }
228     double ret = 0;
229     int sz = ans.size();
230     for (int i = 0; i < sz; i++) ret += ans[i];
231     if (sz) ret /= sz;
232     printf("Case%d: %.6lf\n", iCase, ret);
233 }
234 return 0;
235 }

```

8.20 Splay 树

```
1 // splay tree: 仅伸展操作
2 #include<cstdio>
3 #include<iostream>
4 #include<algorithm>
5 #include<cstring>
6 #include<queue>
7 using namespace std;
8
9 const int maxn = 100005;
10 struct Node;
11 Node* null;
12 struct Node {
13     Node *ch[2], *fa;
14     int size, rev, key;
15     Node() { ch[0] = ch[1] = fa = null; rev = 0; }
16     inline void push_up() {
17         if (this == null) return;
18         size = ch[0] -> size + ch[1] -> size + 1;
19     }
20     inline void setc(Node* p, int d) {
21         ch[d] = p;
22         p -> fa = this;
23     }
24     inline bool d() {
25         return fa -> ch[1] == this;
26     }
27     void clear() {
28         size = 1;
29         ch[0] = ch[1] = fa = null;
30         rev = 0;
31     }
32     void Update_Rev() {
33         if (this == null) return;
34         swap(ch[0], ch[1]);
35         rev ^= 1;
36     }
37     inline void push_down() {
38         if (this == null) return;
39         if (rev) {
40             ch[0] -> Update_Rev();
41             ch[1] -> Update_Rev();
42             rev = 0;
43         }
44     }
45     inline bool isroot() {
46         return fa == null || this != fa -> ch[0] && this != fa -> ch[1];
47     }
48 };
49 Node pool[maxn], *tail;
50 Node *node[maxn], *root;
51
52 inline void rotate(Node* x) {
```

```

53     Node *f = x->fa, *ff = x->fa->fa;
54     f->push_down();
55     x->push_down();
56     int c = x->d(), cc = f->d();
57     f->setc(x->ch[!c], c);
58     x->setc(f, !c);
59     if (ff->ch[cc] == f) ff->setc(x, cc);
60     else x->fa = ff;
61     f->push_up();
62 }
63
64 inline void splay(Node* &root, Node* x, Node* goal) {
65     while (x->fa != goal) {
66         if (x->fa->fa == goal) rotate(x);
67         else {
68             x->fa->fa->push_down();
69             x->fa->push_down();
70             x->push_down();
71             bool f = x->fa->d();
72             x->d() == f ? rotate(x->fa) : rotate(x);
73             rotate(x);
74         }
75     }
76     x->push_up();
77     if (goal == null) root = x;
78 }
79
80 Node* get_kth(Node* r, int k) {
81     Node* x = r;
82     x->push_down();
83     while (x->ch[0]->size + 1 != k) {
84         if (k < x->ch[0]->size + 1) x = x->ch[0];
85         else {
86             k -= x->ch[0]->size + 1;
87             x = x->ch[1];
88         }
89     }
90     x->push_down();
91     return x;
92 }
93
94 Node* get_next(Node* p) {
95     p->push_down();
96     p = p->ch[1];
97     p->push_down();
98     while (p->ch[0] != null) {
99         p = p->ch[0];
100         p->push_down();
101     }
102     return p;
103 }
104
105 void build(Node* &x, int l, int r, Node* fa) {
106     if (l > r) return;

```

```
107     int mid = (l + r) >> 1;
108     x = tail++;
109     x->clear();
110     x->fa = fa;
111     node[mid] = x;
112     build(x->ch[0], l, mid - 1, x);
113     build(x->ch[1], mid + 1, r, x);
114     x->push_up();
115 }
116
117 void init(int n) {
118     tail = pool;
119     null = tail++;
120     null->fa = null->ch[0] = null->ch[1] = null;
121     null->size = 0; null->rev = 0;
122     Node *p = tail++;
123     p->clear();
124     root = p;
125     p = tail++;
126     p->clear();
127     root->setc(p, 1);
128     build(root->ch[1]->ch[0], 1, n, root->ch[1]);
129     root->ch[1]->push_up();
130     root->push_up();
131 }
132
133 int a[maxn], b[maxn];
134 bool cmp(int i, int j) { return a[i] < a[j] || (a[i] == a[j] && i < j); }
135
136 int main() {
137     int n;
138     while (scanf("%d", &n), n) {
139         for (int i = 1; i <= n; i++) {
140             scanf("%d", &a[i]);
141             b[i] = i;
142         }
143         init(n);
144         sort(b + 1, b + n + 1, cmp);
145         for (int i = 1; i <= n; i++) {
146             splay(root, node[b[i]], null);
147             int sz = root->ch[0]->size;
148             printf("%d", root->ch[0]->size);
149             if (i == n) printf("\n");
150             else printf(" ");
151             splay(root, get_kth(root, i), null);
152             splay(root, get_kth(root, sz + 2), root);
153             root->ch[1]->ch[0]->Update_Rev();
154         }
155     }
156     return 0;
157 }
```

8.21 Splay 树

```

1  typedef int value;
2
3  enum { LEFT, RIGHT };
4  struct node {
5      node * child[2], * parent;
6      value v, subtree;
7      int size;
8  } pool[MAXN], * pool_next = pool;
9
10 node * allocate(const value & v) {
11     node * x = pool_next++;
12     x->parent = x->child[LEFT] = x->child[RIGHT] = nullptr;
13     x->subtree = x->v = v;
14     x->size = 1;
15     return x;
16 }
17
18 struct tree {
19     node * root;
20     tree(): root(allocate(0)) {}
21
22     bool child_dir(const node * x, const node * y) { return (x->child[LEFT] == y) ? LEFT :
        RIGHT; }
23     bool is_child(const node * x, const node * y) { return x->child[LEFT] == y || x->child[
        RIGHT] == y; }
24
25     void update(node * x) {
26         x->size = 1;
27         x->subtree = x->v;
28         FOR (d, 2) if (x->child[d] != nullptr) {
29             x->size += x->child[d]->size;
30             if (d == LEFT) x->subtree = x->child[LEFT]->subtree + x->subtree;
31             else x->subtree = x->subtree + x->child[RIGHT]->subtree;
32         }
33     }
34
35     void set_child(node * x, bool dir, node * y) {
36         if ((x->child[dir] = y) != nullptr) y->parent = x;
37         update(x);
38     }
39
40     node * rotate(node * x, bool dir) {
41         node * parent = x->parent, * y = x->child[dir];
42         set_child(x, dir, y->child[!dir]);
43         set_child(y, !dir, x);
44         set_child(parent, child_dir(parent, x), y);
45         return y;
46     }
47
48     node * splay(node * x) {
49         node * old_p = nullptr;
50         while (x->parent != nullptr) {

```

```

51     node * p = x->parent;
52     x = rotate(p, child_dir(p, x));
53     if (old_p != nullptr && is_child(p, old_p)) rotate(p, child_dir(p, old_p));
54     old_p = p;
55 }
56 return x;
57 }
58
59 node * insert(int order, const value & v) { // order is 0-indexed
60     bool dir = LEFT;
61     node * parent = root, * x = parent->child[LEFT];
62     while (x != nullptr) {
63         int left_size = (x->child[LEFT] == nullptr) ? 0 : x->child[LEFT]->size;
64         parent = x;
65         if (order <= left_size) x = x->child[dir = LEFT];
66         else {
67             order -= left_size + 1;
68             x = x->child[dir = RIGHT];
69         }
70     }
71     set_child(parent, dir, x = allocate(v));
72     return splay(x);
73 }
74
75 node * find(int order) {
76     node * x = root->child[LEFT];
77     while (true) {
78         int left_size = (x->child[LEFT] == nullptr) ? 0 : x->child[LEFT]->size;
79         if (order < left_size) x = x->child[LEFT];
80         else if (order == left_size) break;
81         else {
82             order -= left_size + 1;
83             x = x->child[RIGHT];
84         }
85     }
86     return splay(x);
87 }
88
89 void erase(const int& order) {
90     node * x = find(order);
91     if (x->child[LEFT] == nullptr) set_child(root, LEFT, x->child[RIGHT]);
92     else if (x->child[RIGHT] == nullptr) set_child(root, LEFT, x->child[LEFT]);
93     else {
94         node * y = x->child[RIGHT];
95         while (y->child[LEFT] != nullptr) y = y->child[LEFT];
96         y = splay(y);
97         set_child(y, LEFT, x->child[LEFT]);
98         set_child(root, LEFT, y);
99     }
100 }
101
102 value query(int e) { // e is the prefix length desired.
103     node * x = root->child[LEFT];
104     if (e <= 0) return 0;

```

```

105     if (e >= x->size) return x->subtree;
106     x = find(e - 1);
107     if (x->child[LEFT] != nullptr) return x->child[LEFT]->subtree * x->v;
108     else return x->v;
109 }
110 };

```

8.22 点分治

```

1  /*
2  * Author: Simon
3  * 复杂度:  $O(n\log(n))$ 
4  * 功能: 点分治求树上满足边长小于 $k$ 的  $(u, v)$  的对数
5  */
6  namespace PDC {
7      struct node {
8          int v, w, next;
9          node() {}
10         node(int v, int w, int next) : v(v), w(w), next(next) {}
11     } g[maxn << 1];
12     int head[maxn], cnt = 0;
13     int rt /*树重心*/, son[maxn] /*不包含当前节点的所有子树的最大size*/, sz[maxn] /*当前节点为
14         根的子树的size*/;
15     int q[maxn] /*子树所有节点到根节点的距离*/, len, ans = 0;
16     bool vis[maxn];
17     void init() {
18         memset(head, -1, sizeof(head));
19         memset(vis, 0, sizeof(vis));
20         ans = 0; son[0] = INF; cnt = 0;
21     }
22     void addedge(int u, int v, int w) {
23         g[cnt] = node(v, w, head[u]);
24         head[u] = cnt++;
25     }
26     void getrt(int u, int p, int tot) { /*获得树的重心*/
27         sz[u] = 1; son[u] = 0;
28         for (int i = head[u]; ~i; i = g[i].next) {
29             int v = g[i].v;
30             if (v == p || vis[v]) continue;
31             getrt(v, u, tot);
32             sz[u] += sz[v];
33             son[u] = max(son[u], sz[v]);
34         }
35         son[u] = max(son[u], tot - sz[u]);
36         if (son[u] < son[rt]) rt = u;
37     }
38     void getdis(int u, int p, int pw) { /*求出以u为根节点的所有子节点到fa[u]的距离*/
39         q[++len] = pw; sz[u] = 1;
40         for (int i = head[u]; ~i; i = g[i].next) {
41             int v = g[i].v, w = g[i].w;
42             if (v == p || vis[v]) continue;
43             getdis(v, u, pw + w);

```



```

43         sz[u] += sz[v];
44     }
45 }
46 int solve(int u, int w) { /*根据具体题目变化*/
47     len = 0; getdis(u, -1, w);
48     sort(q + 1, q + r + 1); /*按到根节点的距离排序*/
49
50     int ans = 0, l = 1, r = len;
51     while (l < r) {
52         if (q[l] + q[r] <= k) ans += r - l, l++; /*若 $q[l]+q[r]\leq k$ , 则 $q[l]+q[l+1], q[l]+q[l+2], \dots, q[l]+q[r]$ 都满足 $\leq k$ */
53         else r--;
54     }
55     /* 求有多少条长度为 $k$ 的路径 */
56     int l = 1, r = 1, ans = 0;
57     for (int i = 1; i <= len; i++) {
58         l = lower_bound(q + i + 1, q + len + 1, k - q[i]) - q;
59         r = upper_bound(q + i + 1, q + len + 1, k - q[i]) - q;
60         ans += r - l; //求得到根节点长度为 $k-q[i]$ 的路径个数
61     }
62     /*
63     return ans;
64 }
65 void dfs(int u) {
66     vis[u] = 1; ans += solve(u, 0); /*先算上以 $u$ 为根的子树的贡献*/
67     for (int i = head[u]; ~i; i = g[i].next) {
68         int v = g[i].v, w = g[i].w;
69         if (vis[v]) continue;
70         ans -= solve(v, w); /*减去以 $u\sim v$ 这条边为重边而多计算的部分*/
71         rt = 0; getrt(v, -1, sz[v]); /*求以 $v$ 的子树的重心*/
72         dfs(rt);
73     }
74 }
75 void Run(int u, int n) {
76     getrt(1, -1, n);
77     dfs(rt);
78 }
79 }

```

8.23 树上启发式合并

```

1 // 树上启发式合并: dsu on tree
2 int n, x, y, Son, Max;
3 int sz[maxn], son[maxn];
4 ll sum, ans[maxn];
5 vector<int> v[maxn];
6
7 void getson(int u, int fa) {
8     sz[u] = 1;
9     for (int i = 0; i < v[u].size(); i++) {
10         int to = v[u][i];
11         if (to != fa) {

```

```

12         getson(to, u);
13         sz[u] += sz[to];
14         if (sz[to] > sz[son[u]])
15             son[u] = to;
16     }
17 }
18 }
19
20 void add(int u, int fa, int val) {
21     // 更新节点数据
22     // cnt[attr[u]] += val;
23     for (int i = 0; i < v[u].size(); i++) {
24         int to = v[u][i];
25         if (to != fa && to != Son)
26             add(to, u, val);
27     }
28 }
29
30 void dfs(int u, int fa, int k) {
31     for (int i = 0; i < v[u].size(); i++) {
32         int to = v[u][i];
33         if (to != fa && to != son[u])
34             dfs(to, u, 0);
35     }
36     if (son[u]) dfs(son[u], u, 1), Son = son[u];
37     add(u, fa, 1); Son = 0;
38     // 此处统计u节点处答案
39     // ans[u] = sum;
40     if (!k) add(u, fa, -1), Max = sum = 0;
41 }
42
43 // getson(1, 0);
44 // dfs(1, 0, 0);

```

8.24 0-1trie 区间异或最大值

```

1
2 // written by calabash_boy
3 // 01Trie求区间异或和的最大值
4
5 #include <cstdio>
6 #include <cstring>
7 #include <algorithm>
8 using namespace std;
9 const int MAX = 1e6+100;
10 int bas[35];
11 const int INF = 2147483645;
12
13 struct Trie {
14     int nxt[MAX<<2][2]; int l[MAX<<2];
15     int cnt; int ansl,ansr,ansv;
16     void init() {

```

```
17     cnt = 0;
18     memset(nxt[0],0,sizeof (nxt[0]));
19     memset(1,0x3f3f3f3f,sizeof(1));
20     ansv = 0;
21 }
22 int create() {
23     cnt++;
24     memset(nxt[cnt],0,sizeof (nxt[cnt]));
25     return cnt;
26 }
27 void insert(int id,int x) {
28     int y = 0;
29     for (int i=30;i>=0;i--) {
30         int t = x&bas[i];
31         t>>=i;
32         if (!nxt[y][t]) {
33             nxt[y][t] = create();
34         }
35         y = nxt[y][t];
36     }
37     l[y] = min(l[y],id);
38 }
39 void query(int id,int x) {
40     int y=0; int res =0;
41     for (int i=30;i>=0;i--) {
42         int t = x&bas[i];
43         t>>=i;
44         if (nxt[y][!t]) {
45             y =nxt[y][!t];
46             res+=bas[i];
47         } else{
48             y = nxt[y][t];
49         }
50     }
51     if (res==ansv) {
52         if (l[y]<ansl) {
53             ansl = l[y];  ansr = id;
54         }
55     } else if (res>ansv) {
56         ansv = res;
57         ansl = l[y];
58         ansr = id;
59     }
60 }
61 void print(int id) {
62     printf("Case#%d:\n%d%d\n",id,ansl+1,ansr);
63 }
64 }trie;
65
66 void init() {
67     bas[0] = 1;
68     for (int i=1;i<=30;i++) {
69         bas[i] = bas[i-1]<<1;
70     }
```

```
71 }
72 int main() {
73     init();
74     int n, Cas;
75     scanf("%d",&Cas);
76     for (int i=1;i<=Cas;i++) {
77         trie.init();  trie.insert(0,0);
78         scanf("%d",&n);
79         int sum=0;
80         for (int j=1;j<=n;j++) {
81             int ai;
82             scanf("%d",&ai);  sum^=ai;
83             trie.query(j,sum);  trie.insert(j,sum);
84         }
85         trie.print(i);
86     }
87     return 0;
88 }
```

8.25 0-1trie 子树异或最大值

```
1 // 可持久化01Trie+DFS序 子树上的点抑或最大值:
2 // written by calabash_boy
3
4 #include <iostream>
5 #include <cstdio>
6 using namespace std;
7 const int MAX = 1e5+100;
8 int bas[35];  int nxt[MAX<<5][2];
9 int root[MAX];  int sum[MAX<<5];
10 int n,q;  vector<int>E[MAX];
11 int st[MAX],en[MAX],rk[MAX];
12 int a[MAX];  int cnt;  int tot;
13 void sheet(){
14     bas[0]=1;
15     for (int i=1;i<=30;i++){
16         bas[i] = bas[i-1]<<1;
17     }
18 }
19 void init(){
20     for (int i=0;i<=n;i++){  E[i].clear(); }
21     cnt =tot=0;
22     memset(nxt[0],0,sizeof nxt[0]);
23 }
24 void input(){
25     for (int i=1;i<=n;i++){  scanf("%d",a+i);  }
26     for (int u=2;u<=n;u++){
27         int v;  scanf("%d",&v);
28         E[u].push_back(v);  E[v].push_back(u);
29     }
30 }
31 void dfs(int node ,int father ){
```

```
32     st[node] = ++tot; rk[tot] = node;
33     for (int des:E[node]){
34         if(des==father){ continue; }
35         dfs(des,node);
36     }
37     en[node] = tot;
38 }
39 int create(){
40     cnt++;
41     memset(nxt[cnt],0,sizeof nxt[cnt]);
42     return cnt;
43 }
44 int insert(int rt,int val){
45     int y = ++cnt; int x = rt; int res = y;
46     for (int i=30;i>=0;i--){
47         sum[y] = sum[x]+1;
48         nxt[y][0] = nxt[x][0]; nxt[y][1] = nxt[x][1];
49         int t = val&bas[i];
50         t>>=i;
51         nxt[y][t] = create();
52         y = nxt[y][t]; x = nxt[x][t];
53     }
54     sum[y] = sum[x]+1;
55     return res;
56 }
57 int query(int l,int r,int val){
58     int res =0; int x = l; int y = r;
59     for (int i=30;i>=0;i--){
60         int t = val&bas[i];
61         t>>=i;
62         if (sum[nxt[y][!t]]-sum[nxt[x][!t]]){
63             y = nxt[y][!t]; x = nxt[x][!t];
64             res+=bas[i];
65         }else{
66             y = nxt[y][t]; x = nxt[x][t];
67         }
68     }
69     return res;
70 }
71 void solve(){
72     dfs(1,0);
73     for (int i=1;i<=n;i++){
74         root[i] = insert(root[i-1],a[rk[i]]);
75     }
76     while (q--){
77         int nod,x;
78         scanf("%d%d",&nod,&x);
79         printf("%d\n",query(root[st[nod]-1],root[en[nod]],x));
80     }
81 }
82 int main(){
83     sheet();
84     while (scanf("%d%d",&n,&q)!=EOF){
85         init();
```

```

86     input();
87     solve();
88 }
89 return 0;
90 }

```

8.26 莫队算法

```

1  //Author:marszed
2  /*
3   *离线区间处理问题。
4   *从区间 $[l,r]$ 得到区间 $[l+1,r+1]$   $[l-1,r-1]$ 信息的转移复杂度为 $O(1)$ 。
5   *siz为块大小。
6   *cnt为位于第几个块。
7   *modify()函数为转移函数。
8   */
9
10 #include <iostream>
11 #include <algorithm>
12 #include <cmath>
13
14 const int maxn = 2e5 + 10;
15
16 int n, siz, q;
17 int a[maxn];
18
19 struct Node {
20     int id, l, r, val, cnt;
21
22     int operator< (const Node& b) {
23         return cnt == b.cnt ? r < b.r : cnt < b.cnt;
24     }
25 } nod[maxn];
26
27 void modify(int i, int flag) {
28
29 }
30
31 void mo() {
32     std::cin >> n >> q;
33     siz = sqrt(n);
34     for (int i = 1; i <= n; i++) {
35         std::cin >> a[i];
36     }
37     for (int i = 1; i <= q; i++) {
38         std::cin >> nod[i].l >> nod[i].r;
39         nod[i].id = i;
40         nod[i].cnt = nod[i].l / siz;
41     }
42     std::sort(nod + 1, nod + q + 1);
43     int l = 0, r = 0;
44     for (int i = 1; i <= q; i++) {

```

```

45     while (l < nod[i].l - 1)    modify(++l, 1);
46     while (l >= nod[i].l)      modify(l--, 1);
47     while (r < nod[i].r)      modify(++r, 1);
48     while (r > nod[i].r)      modify(r--, 1);
49     ans[nod[i].id] = Ans;
50 }
51 }
52
53 int main() {}

```

8.27 最近公共祖先 (在线)

```

1  // 时间复杂度  $O(n\log n+q)$ 
2  // By CSL
3
4  const int maxn = "Edit";
5  std::vector<int> G[maxn], sp;
6  int dep[maxn], dfn[maxn];
7
8  std::pair<int, int> dp[21][maxn << 1];
9
10 void init(int n) {
11     for (int i = 0; i < n; i++) G[i].clear();
12     sp.clear();
13 }
14
15 void dfs(int u, int fa) {
16     dep[u] = dep[fa] + 1;
17     dfn[u] = sp.size();
18     sp.push_back(u);
19     for (auto& v : G[u]) {
20         if (v == fa) continue;
21         dfs(v, u);
22         sp.push_back(u);
23     }
24 }
25
26 void initrmq() {
27     int n = sp.size();
28     for (int i = 0; i < n; i++) dp[0][i] = {dfn[sp[i]], sp[i]};
29     for (int i = 1; (1 << i) <= n; i++)
30         for (int j = 0; j + (1 << i) - 1 < n; j++)
31             dp[i][j] = std::min(dp[i - 1][j], dp[i - 1][j + (1 << (i - 1))]);
32 }
33
34 int lca(int u, int v) {
35     int l = dfn[u], r = dfn[v];
36     if (l > r) std::swap(l, r);
37     int k = 31 - __builtin_clz(r - l + 1);
38     return std::min(dp[k][l], dp[k][r - (1 << k) + 1]).second;
39 }

```

8.28 最近公共祖先 (离线)

```
1 // 时间复杂度  $O(n+q)$ 
2 // By CSL
3
4 #include <iostream>
5 #include <algorithm>
6 #include <vector>
7
8 const int maxn = "Edit";
9 int par[maxn]; // 并查集
10 int ans[maxn]; // 存储答案
11 std::vector<int> G[maxn]; // 邻接表
12 std::vector<std::pair<int, int>> query[maxn]; // 存储查询信息
13 bool vis[maxn]; // 是否被遍历
14
15 inline void init(int n) {
16     for (int i = 1; i <= n; i++) {
17         G[i].clear(), query[i].clear();
18         par[i] = i, vis[i] = 0;
19     }
20 }
21
22 int find(int u) {
23     return par[u] == u ? par[u] : par[u] = find(par[u]);
24 }
25
26 void unite(int u, int v) {
27     par[find(v)] = find(u);
28 }
29
30 inline void add_edge(int u, int v) {
31     G[u].push_back(v);
32 }
33
34 inline void add_query(int id, int u, int v) {
35     query[u].push_back(std::make_pair(v, id));
36     query[v].push_back(std::make_pair(u, id));
37 }
38
39 void tarjan(int u) {
40     vis[u] = 1;
41     for (auto& v : G[u]) {
42         if (vis[v]) continue;
43         tarjan(v);
44         unite(u, v);
45     }
46     for (auto& q : query[u]) {
47         int &v = q.first, &id = q.second;
48         if (!vis[v]) continue;
49         ans[id] = find(v);
```



```
50     }  
51 }
```

8.29 最近公共祖先

```
1  // LCA ST算法  
2  int n, top, root;  
3  int a[maxn << 1], d[maxn], st[maxn];  
4  int f[maxn << 1][18], loc[maxn << 1][18];  
5  vector<int> v[maxn];  
6  
7  int log2(int x) {  
8      int k = 0;  
9      while (x > 1) {  
10         x /= 2;  
11         k++;  
12     }  
13     return k;  
14 }  
15  
16 void dfs(int u, int dep) {  
17     d[u] = dep;  
18     a[++top] = u;  
19     for (int i = 0; i <= v[u].size(); i++) {  
20         int to = v[u][i];  
21         dfs(to, dep + 1);  
22         a[++top] = u;  
23     }  
24 }  
25  
26 void init() {  
27     int s = log2(top);  
28     for (int i = 1; i <= top; i++) {  
29         f[i][0] = d[a[i]];  
30         loc[i][0] = a[i];  
31     }  
32     for (int j = 1; j <= s; j++) {  
33         int k = top - (1 << j) + 1;  
34         for (int i = 1; i <= k; i++) {  
35             int x = i + (1 << (j - 1));  
36             if (f[i][j - 1] <= f[x][j - 1]) {  
37                 f[i][j] = f[i][j - 1];  
38                 loc[i][j] = loc[i][j - 1];  
39             }  
40             else {  
41                 f[i][j] = f[x][j - 1];  
42                 loc[i][j] = loc[x][j - 1];  
43             }  
44         }  
45     }  
46 }  
47 }
```

```
48 int query(int x, int y) {
49     x = st[x], y = st[y];
50     if (x > y) swap(x, y);
51     int i = log2(y - x);
52     int k = y - (1 << i) + 1;
53     return f[x][i] < f[k][i] ? loc[x][i] : loc[k][i];
54 }
55
56 //=====
57
58 // LCA Tarjan算法
59 int n, root, cnt;
60 int pre[maxn], ans[maxn];
61 vector<int> v[maxn], s[maxn], num[maxn];
62
63 int find(int x) { return pre[x] == x ? x : pre[x] = find(pre[x]); }
64
65 void dfs(int u) {
66     pre[u] = u;
67     for (int i = 0; i < v[u].size(); i++) {
68         int to = v[u][i];
69         dfs(to);
70         pre[find(pre[to])] = find(pre[u]);
71     }
72     for (int i = 0; i < s[u].size(); i++) {
73         int to = s[u][i];
74         if (pre[to] != to)
75             ans[num[u][i]] = find(pre[to]);
76     }
77 }
78
79 /*
80 for (int i = 1; i <= q; i++) {
81     scanf("%d%d", &x, &y);
82     if (x == y) ans[i] = x;
83     s[x].push_back(y);
84     s[y].push_back(x);
85     num[x].push_back(i);
86     num[y].push_back(i);
87 }
88 dfs(root);
89 */
90
91 //=====
92
93 // LCA 倍增算法
94 int n, ma, root;
95 int d[maxn], f[maxn][20];
96 vector<int> v[maxn];
97 inline void dfs(int u, int dep, int fa) {
98     d[u] = dep;
99     f[u][0] = fa;
100     ma = max(ma, dep);
101     for (int i = 0; i < v[u].size(); i++)
```

```

102         if (v[u][i] != fa) dfs(v[u][i], dep + 1, u);
103     }
104     inline int log2(int x) {
105         int k = 0;
106         while (x > 1) {
107             x >>= 1;
108             k++;
109         }
110         return k;
111     }
112     inline void init() {
113         dfs(root, 0, 0);
114         int s = log2(ma);
115         for (int j = 1; j <= s; j++)
116             for (int i = 1; i <= n; i++)
117                 f[i][j] = f[f[i][j - 1]][j - 1];
118     }
119     // 求x与y的LCA
120     inline int query(int x, int y) {
121         if (d[x] < d[y]) swap(x, y);
122         int s = log2(d[x] - d[y]);
123         while (d[x] > d[y]) {
124             if (d[x] - (1 << s) >= d[y])
125                 x = f[x][s];
126             s--;
127         }
128         s = log2(d[x]);
129         while (s > -1) {
130             if (f[x][s] != f[y][s]) {
131                 x = f[x][s];
132                 y = f[y][s];
133             }
134             s--;
135         }
136         return x == y ? x : f[x][0];
137     }
138     // 判断a与p是否在同一树边上 (p在a上方)
139     inline bool check(int a, int p) {
140         if (d[a] < d[p]) return false;
141         if (d[a] == d[p]) return a == p;
142         int s = log2(d[a] - d[p]);
143         while (d[a] > d[p]) {
144             if (d[a] - (1 << s) >= d[p])
145                 a = f[a][s];
146             s--;
147         }
148         return a == p;
149     }
150     // 求一条树边上x到y的距离
151     inline int getlen(int x, int y) {
152         int ret = 0;
153         if (d[x] < d[y]) swap(x, y);
154         int s = log2(d[x] - d[y]);
155         while (d[x] > d[y]) {

```

```

156     if (d[x] - (1 << s) >= d[y]) {
157         ret += (1 << s);
158         x = f[x][s];
159     }
160     s--;
161 }
162 return ret;
163 }

```

8.30 树链剖分

```

1 // 树链剖分 点权
2 /**
3  * top[v] 表示v所在的重链的顶端节点
4  * fa[v] 表示v的父节点
5  * deep[v] 表示v的深度(根的深度为1)
6  * snum[v] 表示以v为根的子树的节点数
7  * p[v] 表示v所在(线段树中)的位置
8  * fp[v] 与p[v]相反, 表示对应位置的节点
9  * son[v] 表示v的重儿子
10 * Edge 存树边
11 */
12
13 struct Edge {
14     int to, next;
15 }edge[maxn << 1];
16
17 int pos, n, m, tot; // n 为节点数
18 int head[maxn], top[maxn], fa[maxn], deep[maxn], num[maxn], p[maxn], fp[maxn], son[maxn];
19
20 void init() {
21     tot = 0;
22     pos = 1;
23     memset(head, -1, sizeof(head));
24     memset(son, -1, sizeof(son));
25     for (int i = 0; i <= n; i++)
26         v[i].clear();
27 }
28
29 void addedge(int u, int v) {
30     edge[tot].to = v;
31     edge[tot].next = head[u];
32     head[u] = tot++;
33 }
34
35 void dfs1(int u, int pre, int d) {
36     deep[u] = d;
37     fa[u] = pre;
38     num[u] = 1;
39     for (int i = head[u]; i != -1; i = edge[i].next) {
40         int to = edge[i].to;
41         if (to != pre) {

```

```
42         dfs1(to, u, d + 1);
43         num[u] += num[to];
44         if (son[u] == -1 || num[to] > num[son[u]])
45             son[u] = to;
46     }
47 }
48 }
49
50 void dfs2(int u, int sp) {
51     top[u] = sp;
52     p[u] = pos++;
53     fp[p[u]] = u;
54     if (son[u] == -1) return;
55     dfs2(son[u], sp);
56     for (int i = head[u]; i != -1; i = edge[i].next) {
57         int to = edge[i].to;
58         if (to != son[u] && to != fa[u])
59             dfs2(to, to);
60     }
61 }
62 /*
63 // 使用范例
64 int getsum(int a, int b) {
65     int f1 = top[a], f2 = top[b];
66     int ret = 0;
67     while (f1 != f2) {
68         if (deep[f1] < deep[f2]) {
69             swap(f1, f2);
70             swap(a, b);
71         }
72         ret += query(p[f1], p[a], 1, n - 1, 1);
73         a = fa[f1]; f1 = top[a];
74     }
75     if (a == b) return ret;
76     if (deep[a] > deep[b]) swap(a, b);
77     return ret + query(p[son[a]], p[b], 1, n - 1, 1);
78 }
79 */
```

第九章 字符串

9.1 KMP

```
1  /*
2   * Author: Simon
3   * 复杂度:  $O(n)$ 
4   */
5  int Next[maxn]; /* $i$ 之前相同前缀后缀的长度, 例ababc, Next[5]=2; */
6  void getNext(int m, char p[]){
7      memset(Next, 0, sizeof(int)*(m+5));
8      int k=-1, j=0;
9      Next[0]=-1;
10     while(j<m){
11         if(k==-1 || p[k]==p[j]){
12             k++, j++;
13             if(p[k]!=p[j]) Next[j]=k;
14             else Next[j]=Next[k];
15         }
16         else k=Next[k];
17     }
18 }
19 int KMP(int n, int m, char s[], char p[]){
20     int i=0, j=0, ans=0;
21     while(i<n){
22         if(j==-1 || s[i]==p[j]) i++, j++;
23         else j=Next[j];
24         if(j==m) ans++; /*计数 (可重叠) */
25         //if(j==m) ans++, j=0; /*计数 (不可重叠) */
26         //if(j==m) return i-m+1; /*返回第一个匹配的位置 */
27     }
28     //return j; /*返回s后缀与p前缀匹配的最长长度 */
29     return ans;
30 }
```

9.2 扩展 KMP

```
1  /*
2   * Author: nuchenghao
3   * 复杂度:  $O(n+m)$ 
4   * 功能: 解决如下问题: 定义母串 $s$ 和子串 $p$ , 设 $s$ 的长度为 $n$ ,  $p$ 的长度为 $m$ , 求 $p$ 与 $s$ 的每一个后缀的最长公共前缀
5   *       $extend[i]$ 表示 $p$ 与 $s[i, n-1]$ 的最长公共前缀。 ( $0 \leq i < n$ )
```

```

6  *      一个辅助工具为nxt[i]表示p[i,m-1]和p本身的最长公共前缀长度
7  *      下标都从0开始
8  */
9
10 int nxt[maxn], ex[maxn]; /*ex数组即为extend数组 */
11 /*预处理计算next数组 */
12 void getnxt(char *s) {
13     memset(nxt,0,sizeof(nxt));
14     int i = 0, j, po, len = strlen(s);
15     nxt[0] = len; /*初始化next[0] */
16     while (s[i] == s[i + 1] && i + 1 < len) i++; /*计算next[1] */
17     nxt[1] = i; po = 1; /*初始化po的位置 */
18     for (i = 2; i < len; i++) {
19         if (nxt[i - po] + i < nxt[po] + po) nxt[i] = nxt[i - po]; /*第一种情况，可以直接得到
20             next[i]的值 */
21         else { /*第二种情况，要继续匹配才能得到next[i]的值 */
22             j = nxt[po] + po - i;
23             if (j < 0) j = 0; /*如果i>po+nxt[po],则要从头开始匹配 */
24             while (i + j < len && s[j] == s[j + i]) j++; /*计算next[i] */
25             nxt[i] = j;
26             po = i; /*更新po的位置 */
27         }
28     }
29 /*计算extend数组 */
30 void exkmp(char *s, char *p) {
31     memset(ex,0,sizeof(ex));
32     int i = 0, j, po, len = strlen(s), l2 = strlen(p);
33     getnxt(p); /*计算子串的next数组 */
34     while (s[i] == p[i] && i < l2 && i < len) i++; /*计算ex[0] */
35     ex[0] = i; po = 0; //初始化po的位置
36     for (i = 1; i < len; i++) {
37         if (nxt[i - po] + i < ex[po] + po) ex[i] = nxt[i - po]; /*第一种情况，直接可以得到ex[i]
38             ]的值 */
39         else { /*第二种情况，要继续匹配才能得到ex[i]的值 */
40             j = ex[po] + po - i;
41             if (j < 0) j = 0; /*如果i>ex[po]+po则要从头开始匹配 */
42             while (i + j < len && j < l2 && s[j + i] == p[j]) j++; /*计算ex[i] */
43             ex[i] = j; po = i; //更新po的位置
44         }
45     }
46 }

```

9.3 字符串的最小最大表示法

```

1  /*
2  * Author: Simon
3  * 复杂度: O(n)
4  * 功能: 求字符串最小，最大表示法的起始下标
5  *      最小表示法指一个循环串以某一位开始时对应的串的字典序最小 这个串就是该循环串的最小表示
6  *      法
7  */

```

```
7
8  /*最小表示法 */
9  int get_minstring(char *s)
10 {
11     int len = strlen(s);
12     int i = 0, j = 1, k = 0;
13     while(i<len && j<len && k<len) {
14         int t=s[(i+k)%len]-s[(j+k)%len];
15         if(t==0) k++;
16         else {
17             if(t > 0) i+=k+1;
18             else j+=k+1;
19             if(i==j) j++;
20             k=0;
21         }
22     }
23     return min(i,j);
24 }
25
26 /*最大表示法 */
27 int get_maxstring(char *s)
28 {
29     int len = strlen(s);
30     int i = 0, j = 1, k = 0;
31     while(i<len && j<len && k<len) {
32         int t=s[(i+k)%len]-s[(j+k)%len];
33         if(t==0) k++;
34         else {
35             if(t > 0) j+=k+1;
36             else i+=k+1;
37             if(i==j) j++;
38             k=0;
39         }
40     }
41     return min(i,j);
42 }
```

9.4 TRIE

```
1  int tree[maxn][26];
2  int sum[maxn];
3  int tot;
4  void Insert(char *str) {
5      int len = strlen(str);
6      int root = 0;
7      for (int i = 0; i < len; i++) {
8          int id = str[i] - 'a';
9          if (!tree[root][id]) tree[root][id] = ++tot;
10         root = tree[root][id];
11         sum[root]++;
12     }
13 }
```



```
14
15 int Find(char *str) {
16     int len = strlen(str);
17     int root = 0;
18     for (int i = 0; i < len; i++) {
19         int id = str[i] - 'a';
20         if (!tree[root][id]) return 0;
21         root = tree[root][id];
22     }
23     return sum[root];
24 }
```

9.5 AC 自动机

```
1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #include <queue>
6 using namespace std;
7
8 struct Trie {
9     int next[500010][26], fail[500010], end[500010];
10    int root, L;
11    int newnode() {
12        for (int i = 0; i < 26; i++)
13            next[L][i] = -1;
14        end[L++] = 0;
15        return L - 1;
16    }
17    void init() {
18        L = 0;
19        root = newnode();
20    }
21    void insert(char buf[]) {
22        int len = strlen(buf);
23        int now = root;
24        for (int i = 0; i < len; i++) {
25            if (next[now][buf[i] - 'a'] == -1)
26                next[now][buf[i] - 'a'] = newnode();
27            now = next[now][buf[i] - 'a'];
28        }
29        end[now]++;
30    }
31    void build() {
32        queue<int>Q;
33        fail[root] = root;
34        for (int i = 0; i < 26; i++)
35            if (next[root][i] == -1)
36                next[root][i] = root;
37            else {
38                fail[next[root][i]] = root;
```

```

39         Q.push(next[root][i]);
40     }
41     while (!Q.empty()) {
42         int now = Q.front();
43         Q.pop();
44         for (int i = 0; i < 26; i++)
45             if (next[now][i] == -1)
46                 next[now][i] = next[fail[now]][i];
47             else
48                 {
49                     fail[next[now][i]] = next[fail[now]][i];
50                     Q.push(next[now][i]);
51                 }
52     }
53 }
54 // 查询buf字符串包含的模板串
55 int query(char buf[]) {
56     int len = strlen(buf);
57     int now = root;
58     int res = 0;
59     for (int i = 0; i < len; i++) {
60         now = next[now][buf[i] - 'a'];
61         int temp = now;
62         while (temp != root) {
63             res += end[temp];
64             end[temp] = 0;
65             temp = fail[temp];
66         }
67     }
68     return res;
69 }
70 };
71 char buf[1000010];

```

9.6 后缀数组 (倍增)

```

1  //author: Menci
2  #include <algorithm>
3  #include <string>
4  #include <iostream>
5
6  const int maxn = 1000;
7
8  char s[maxn];
9  int n, ht[maxn], rk[maxn], sa[maxn];
10
11 inline void suffixArray() {
12     static int set[maxn + 1], a[maxn + 1];
13     std::copy(s, s + n, set + 1);
14     std::sort(set + 1, set + n + 1);
15     int *end = std::unique(set + 1, set + n + 1);
16     for (int i = 1; i <= n; i++) a[i] = std::lower_bound(set + 1, end, s[i]) - set;

```

```

17
18 static int fir[maxn + 1], sec[maxn + 1], tmp[maxn + 1], buc[maxn + 1];
19 for (int i = 1; i <= n; i++) buc[a[i]]++;
20 for (int i = 1; i <= n; i++) buc[i] += buc[i - 1];
21 for (int i = 1; i <= n; i++) rk[i] = buc[a[i] - 1] + 1;
22
23 for (int t = 1; t <= n; t *= 2) {
24     for (int i = 1; i <= n; i++) fir[i] = rk[i];
25     for (int i = 1; i <= n; i++) sec[i] = i + t > n ? 0 : rk[i + t];
26
27     std::fill(buc, buc + n + 1, 0);
28     for (int i = 1; i <= n; i++) buc[sec[i]]++;
29     for (int i = 1; i <= n; i++) buc[i] += buc[i - 1];
30     for (int i = 1; i <= n; i++) tmp[n - buc[sec[i]]] = i;
31
32     std::fill(buc, buc + n + 1, 0);
33     for (int i = 1; i <= n; i++) buc[fir[i]]++;
34     for (int i = 1; i <= n; i++) buc[i] += buc[i - 1];
35     for (int j = 1, i; j <= n; j++) i = tmp[j], sa[buc[fir[i]] - 1] = i;
36
37     bool unique = true;
38     for (int j = 1, i, last = 0; j <= n; j++) {
39         i = sa[j];
40         if (!last) rk[i] = 1;
41         else if (fir[i] == fir[last] && sec[i] == sec[last]) rk[i] = rk[last], unique = false;
42         else rk[i] = rk[last] + 1;
43
44         last = i;
45     }
46
47     if (unique) break;
48 }
49
50 for (int i = 1, k = 0; i <= n; i++) {
51     if (rk[i] == 1) k = 0;
52     else {
53         if (k > 0) k--;
54         int j = sa[rk[i] - 1];
55         while (i + k <= n && j + k <= n && a[i + k] == a[j + k]) k++;
56     }
57     ht[rk[i]] = k;
58 }
59 }
60
61 int main() {
62     std::cin >> n >> s;
63     suffixArray();
64     for (int i = 1; i <= n; i++) {
65         std::cout << sa[i] << " ";
66     }
67 }

```

9.7 后缀数组 (sais)

```

1 namespace SA {
2     int sa[N], rk[N], ht[N], s[N<<1], t[N<<1], p[N], cnt[N], cur[N];
3     #define pushS(x) sa[cur[s[x]]++] = x
4     #define pushL(x) sa[cur[s[x]]++] = x
5     #define inducedSort(v) std::fill_n(sa, n, -1); std::fill_n(cnt, m, 0); \
6         for (int i = 0; i < n; i++) cnt[s[i]]++; \
7         for (int i = 1; i < m; i++) cnt[i] += cnt[i-1]; \
8         for (int i = 0; i < m; i++) cur[i] = cnt[i]-1; \
9         for (int i = n1-1; ~i; i--) pushS(v[i]); \
10        for (int i = 1; i < m; i++) cur[i] = cnt[i-1]; \
11        for (int i = 0; i < n; i++) if (sa[i] > 0 && t[sa[i]-1]) pushL(sa[i]-1); \
12        for (int i = 0; i < m; i++) cur[i] = cnt[i]-1; \
13        for (int i = n-1; ~i; i--) if (sa[i] > 0 && !t[sa[i]-1]) pushS(sa[i]-1)
14 void sais(int n, int m, int *s, int *t, int *p) {
15     int n1 = t[n-1] = 0, ch = rk[0] = -1, *s1 = s+n;
16     for (int i = n-2; ~i; i--) t[i] = s[i] == s[i+1] ? t[i+1] : s[i] > s[i+1];
17     for (int i = 1; i < n; i++) rk[i] = t[i-1] && !t[i] ? (p[n1] = i, n1++) : -1;
18     inducedSort(p);
19     for (int i = 0, x, y; i < n; i++) if (~x = rk[sa[i]]) {
20         if (ch < 1 || p[x+1] - p[x] != p[y+1] - p[y]) ch++;
21         else for (int j = p[x], k = p[y]; j <= p[x+1]; j++, k++)
22             if ((s[j]<<1|t[j]) != (s[k]<<1|t[k])) {ch++; break;}
23         s1[y = x] = ch;
24     }
25     if (ch+1 < n1) sais(n1, ch+1, s1, t+n, p+n1);
26     else for (int i = 0; i < n1; i++) sa[s1[i]] = i;
27     for (int i = 0; i < n1; i++) s1[i] = p[sa[i]];
28     inducedSort(s1);
29 }
30 template<typename T>
31 int mapCharToInt(int n, const T *str) {
32     int m = *max_element(str, str+n);
33     std::fill_n(rk, m+1, 0);
34     for (int i = 0; i < n; i++) rk[str[i]] = 1;
35     for (int i = 0; i < m; i++) rk[i+1] += rk[i];
36     for (int i = 0; i < n; i++) s[i] = rk[str[i]] - 1;
37     return rk[m];
38 }
39 // Ensure that str[n] is the unique lexicographically smallest character in str.
40 template<typename T>
41 void suffixArray(int n, const T *str) {
42     int m = mapCharToInt(++n, str);
43     sais(n, m, s, t, p);
44     for (int i = 0; i < n; i++) rk[sa[i]] = i;
45     for (int i = 0, h = ht[0] = 0; i < n-1; i++) {
46         int j = sa[rk[i]-1];
47         while (i+h < n && j+h < n && s[i+h] == s[j+h]) h++;
48         ht[rk[i]] = h;
49     }
50 }
51 };

```

9.8 后缀自动机

```
1 //Author:CookieC
2 #include<cstring>
3 #define MAXN 10000
4
5 struct State{
6     State *f,*c[26];
7     int len;
8 };
9
10 State *root,*last,*cur;
11 State StatePool[MAXN];
12
13 State* NewState(int len){
14     cur->len=len;
15     cur->f=0;
16     memset(cur->c,0,sizeof(cur->c));
17     return cur++;
18 }
19
20 void Init(){
21     cur=StatePool;
22     last=StatePool;
23     root=NewState(0);
24 }
25
26 void Extend(int w){
27     State *p = last;
28     State *np = NewState(p->len+1);
29     while(p&&!p->c[w]) {
30         p->c[w] = np;
31         p = p->f;
32     }
33     if(!p) {
34         np->f=root;
35     } else {
36         State *q=p->c[w];
37         if(p->len+1==q->len) {
38             np->f=q;
39         } else {
40             State *nq = NewState(p->len+1);
41             memcpy(nq->c, q->c, sizeof(q->c));
42             nq->f = q->f;
43             q->f = nq;
44             np->f = nq;
45             while(p&&p->c[w]==q) {
46                 p->c[w]=nq;
47                 p=p->f;
48             }
49         }
50     }
```

```

50     }
51     last=np;
52 }
53
54 bool Find(char *s,int len) {
55     int i;
56     State *p=root;
57     for(i=0;i<len;++i) {
58         if(p->c[s[i]-'a']) {
59             p=p->c[s[i]-'a'];
60         } else {
61             return false;
62         }
63     }
64     return true;
65 }

```

9.9 最长回文子串

```

1  /*
2   * Author: Simon
3   * 功能：求最长回文子串或回文子串的个数
4   * 复杂度：O(n)
5   */
6  string Manacher(string s) {
7      /*Insert '#'*/
8      string t = "$#";
9      for (int i = 0; i < s.size(); ++i) { /*预处理*/
10         t += s[i]; t += "#";
11     }
12     /*Process t*/
13     vector<int> p(t.size(), 0); /*p[i]表示以t[i]为中心的最长回文字串的半径长度*/
14     int mx = 0, id = 0, resLen = 0, resCenter = 0;
15     /*其中id为能延伸到最右端的位置的那个回文子串的中心点位置，mx是回文串能延伸到的最右端的位置，最大半径，中心位置*/
16     int ans=0;
17     for (int i = 1; i < t.size(); ++i) {
18         p[i] = mx > i ? min(p[2 * id - i], mx - i) : 1;
19         while (t[i + p[i]] == t[i - p[i]]) ++p[i]; //枚举
20         if (mx < i + p[i]) {
21             mx = i + p[i];
22             id = i;
23         }
24         if (resLen < p[i]) {
25             resLen = p[i];
26             resCenter = i;
27         }
28         ans+=p[i]/2;
29     }
30     return s.substr((resCenter - resLen) / 2, resLen - 1);
31     /*return ans; /*返回回文子串的个数*/
32 }

```

9.10 回文树

```

1  /*
2  * Author: Simon
3  * 复杂度:  $O(n \cdot \log(n))$ 
4  * 1. 求串S前缀0~i内本质不同回文串的个数 (两个串长度不同或者长度相同且至少有一个字符不同便是本质不同)
5  * 2. 求串S内每一个本质不同回文串出现的次数
6  * 3. 求串S内回文串的个数 (其实就是1和2结合起来)
7  * 4. 求以下标i结尾的回文串的个数
8  */
9  struct PAM{
10 /*Int */int tree[maxn][30] /*和字典树类似, 指向的串为当前串两端加上同一个字符构成 */;
11     fail[maxn] /*失配后跳转到fail指针指向的节点, 其为最长回文后缀 */;
12     cnt[maxn] /*表示节点i表示的回文串的个数 (建树时求出的不是完全的, 最后count()函数跑一遍以后才是正确的) */;
13     num[maxn] /*表示以节点i表示的最长回文串的最右端点为回文串结尾的回文串个数, num[idx+2]表示以str[idx]为结尾的回文串的种类数 */;
14     len[maxn] /*len[i]表示节点i表示的回文串的长度 (一个节点表示一个回文串) */;
15     s[maxn] /*存放添加的字符 */; lst/*指向新添加一个字母后所形成的最长回文串表示的节点。 */;
16     n /*表示添加的字符个数。 */; p/*表示添加的节点个数。 */;
17 int newNode(int l){ /*新增一个节点, 其最长回文串长度为l */
18     for(int i=0;i<26;i++) tree[p][i]=0;
19     cnt[p]=num[p]=0, len[p]=l;
20     return p++;
21 }
22 void init(){
23     n=p=lst=0;
24     newNode(0)/*偶节点 */; newNode(-1)/*奇节点 */;
25     s[0]=-1, fail[0]=1/*偶节点失配指针指向奇节点 */;
26 }
27 int getFail(int x){
28     while(s[n-len[x]-1]!=s[n]) x=fail[x];
29     return x;
30 }
31 void add(int c){
32     c-='a'; s[++n]=c;
33     int now=getFail(lst);/*找到最长的回文子串, 并且前后添加c字符后还是回文子串 */
34     if(!tree[now][c]){/*如果树中未存在此回文串 */
35         int next=newNode(len[now]+2);/*为此串建立新节点 */
36         fail[next]=tree[getFail(fail[now])][c];/*为新节点添加失配指针的指向 */
37         tree[now][c]=next;/*记录新串指向的节点 */
38         num[next]=num[fail[next]]+1;/*更新num数组 (num数组含义在上面) */
39     }
40     lst=tree[now][c];/*c字母所形成的最长回文子串所在的节点为lst */
41     cnt[lst]++;/*统计此回文串出现的次数 */
42     //return num[lst] /*正序插入表示以str[i]为结尾的回文串的个数, 倒序插入表示以str[i]为起点的回文串的个数*/
43 }
44 void count(){

```

```

45     for(int i=p-1;i>=0;i--) cnt[fail[i]]+=cnt[i];/*节点i表示的回文串的个数要加上包含此回文
46         串的串的个数, cnt[aa]+=cnt[baab] */
47 }pam;

```

9.11 字符串哈希算法

```

1  // RS Hash Function
2  unsigned int RSHash(char *str) {
3      unsigned int b = 378551;
4      unsigned int a = 63689;
5      unsigned int hash = 0;
6      while (*str) {
7          hash = hash * a + (*str++);
8          a *= b;
9      }
10     return (hash & 0x7FFFFFFF);
11 }
12
13 // JS Hash Function
14 unsigned int JSHash(char *str) {
15     unsigned int hash = 1315423911;
16     while (*str) {
17         hash ^= ((hash << 5) + (*str++) + (hash >> 2));
18     }
19     return (hash & 0x7FFFFFFF);
20 }
21
22 // P. J. Weinberger Hash Function
23 unsigned int PJWHash(char *str) {
24     unsigned int BitsInUnsignedInt = (unsigned int)(sizeof(unsigned int) * 8);
25     unsigned int ThreeQuarters = (unsigned int)((BitsInUnsignedInt * 3) / 4);
26     unsigned int OneEighth = (unsigned int)(BitsInUnsignedInt / 8);
27     unsigned int HighBits = (unsigned int)(0xFFFFFFFF) << (BitsInUnsignedInt -
28         OneEighth);
29     unsigned int hash = 0;
30     unsigned int test = 0;
31     while (*str) {
32         hash = (hash << OneEighth) + (*str++);
33         if ((test = hash & HighBits) != 0) {
34             hash = ((hash ^ (test >> ThreeQuarters)) & (~HighBits));
35         }
36     }
37     return (hash & 0x7FFFFFFF);
38 }
39 // ELF Hash Function
40 unsigned int ELFHash(char *str) {
41     unsigned int hash = 0;
42     unsigned int x = 0;
43     while (*str) {
44         hash = (hash << 4) + (*str++);

```



```
45     if ((x = hash & 0xF0000000L) != 0) {
46         hash ^= (x >> 24);
47         hash &= ~x;
48     }
49 }
50 return (hash & 0x7FFFFFFF);
51 }
52
53 // BKDR Hash Function
54 unsigned int BKDRHash(char *str) {
55     unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
56     unsigned int hash = 0;
57     while (*str) {
58         hash = hash * seed + (*str++);
59     }
60     return (hash & 0x7FFFFFFF);
61 }
62
63 // SDBM Hash Function
64 unsigned int SDBMHash(char *str) {
65     unsigned int hash = 0;
66     while (*str) {
67         hash = (*str++) + (hash << 6) + (hash << 16) - hash;
68     }
69     return (hash & 0x7FFFFFFF);
70 }
71
72 // DJB Hash Function
73 unsigned int DJBHash(char *str) {
74     unsigned int hash = 5381;
75     while (*str) {
76         hash += (hash << 5) + (*str++);
77     }
78     return (hash & 0x7FFFFFFF);
79 }
80
81 // AP Hash Function
82 unsigned int APHash(char *str) {
83     unsigned int hash = 0;
84     int i;
85     for (i=0; *str; i++) {
86         if ((i & 1) == 0) {
87             hash ^= ((hash << 7) ^ (*str++) ^ (hash >> 3));
88         } else {
89             hash ^= (~(hash << 11) ^ (*str++) ^ (hash >> 5));
90         }
91     }
92     return (hash & 0x7FFFFFFF);
93 }
```

9.12 字符串哈希表

```
1 typedef unsigned long long ull;
2 const ull base = 163;
3 char s[maxn];
4 ull hash[maxn];
5
6 void init() {
7     p[0] = 1;
8     hash[0] = 0;
9     int n = strlen(s + 1);
10    for(int i = 1; i <= 100000; i ++ ) p[i] = p[i-1] * base;
11    for(int i = 1; i <= n; i ++ ) hash[i] = hash[i-1] * base + (s[i] - 'a');
12 }
13
14 ull get(int l, int r, ull g[]) {
15     return g[r] - g[l-1] * p[r-l+1];
16 }
17
18 struct HASHMAP {
19     int size;
20     int head[maxh], next[maxn], f[maxn];    // maxh 为 hash 链表最大长度
21     ull state[maxn];
22     void init() {
23         size = 0;
24         memset(head, -1, sizeof(head));
25     }
26     int insert(ull val, int id) {
27         int h = val % maxh;
28         for (int i = head[h]; i != -1; i = next[i])
29             if (val == state[i]) return f[i];
30         f[size] = id;
31         state[size] = val;
32         next[size] = head[h];
33         head[h] = size;
34         return f[size++];
35     }
36 };
```

第十章 几何

10.1 平面几何公式

1 三角形：

- 2 1. 半周长 $P=(a+b+c)/2$
- 3 2. 面积 $S=aHa/2=ab\sin(C)/2=\sqrt{P(P-a)(P-b)(P-c)}$
- 4 3. 中线 $Ma=\sqrt{2(b^2+c^2)-a^2}/2=\sqrt{b^2+c^2+2bccos(A)}/2$
- 5 4. 角平分线 $Ta=\sqrt{bc((b+c)^2-a^2)}/(b+c)=2bccos(A/2)/(b+c)$
- 6 5. 高线 $Ha=bsin(C)=csin(B)=\sqrt{b^2-((a^2+b^2-c^2)/(2a))^2}$
- 7 6. 内切圆半径 $r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)$
 $=4Rsin(A/2)sin(B/2)sin(C/2)=\sqrt{(P-a)(P-b)(P-c)/P}$
 $=Ptan(A/2)tan(B/2)tan(C/2)$
- 10 7. 外接圆半径 $R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))$

11
12

13 四边形：

14 $D1, D2$ 为对角线, M 为对角线中点连线, A 为对角线夹角

- 15 1. $a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2$
- 16 2. $S=D1D2sin(A)/2$
- 17 (以下对圆的内接四边形)
- 18 3. $ac+bd=D1D2$
- 19 4. $S=\sqrt{(P-a)(P-b)(P-c)(P-d)}$, P 为半周长

20
21

22 正 n 边形：

23 R 为外接圆半径, r 为内切圆半径

- 24 1. 中心角 $A=2PI/n$
- 25 2. 内角 $C=(n-2)PI/n$
- 26 3. 边长 $a=2\sqrt{R^2-r^2}=2Rsin(A/2)=2rtan(A/2)$
- 27 4. 面积 $S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))$

28
29

30 圆：

- 31 1. 弧长 $l=rA$
- 32 2. 弦长 $a=2\sqrt{2hr-h^2}=2rsin(A/2)$
- 33 3. 弓形高 $h=r-\sqrt{r^2-a^2/4}=r(1-cos(A/2))=atan(A/4)/2$
- 34 4. 扇形面积 $S1=r1/2=r^2A/2$
- 35 5. 弓形面积 $S2=(r1-a(r-h))/2=r^2(A-sin(A))/2$

36
37

38 棱柱：

- 39 1. 体积 $V=Ah$, A 为底面积, h 为高
- 40 2. 侧面积 $S=lp$, l 为棱长, p 为直截面周长
- 41 3. 全面积 $T=S+2A$

42

43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91

棱锥：

1. 体积 $V=Ah/3$, A 为底面积, h 为高
(以下对正棱锥)
2. 侧面积 $S=lp/2$, l 为斜高, p 为底面周长
3. 全面积 $T=S+A$

棱台：

1. 体积 $V=(A_1+A_2+\sqrt{A_1A_2})h/3$, A_1, A_2 为上下底面积, h 为高
(以下为正棱台)
2. 侧面积 $S=(p_1+p_2)l/2$, p_1, p_2 为上下底面周长, l 为斜高
3. 全面积 $T=S+A_1+A_2$

圆柱：

1. 侧面积 $S=2\pi rh$
2. 全面积 $T=2\pi r(h+r)$
3. 体积 $V=\pi r^2h$

圆锥：

1. 母线 $l=\sqrt{h^2+r^2}$
2. 侧面积 $S=\pi rl$
3. 全面积 $T=\pi r(l+r)$
4. 体积 $V=\pi r^2h/3$

圆台：

1. 母线 $l=\sqrt{h^2+(r_1-r_2)^2}$
2. 侧面积 $S=\pi(r_1+r_2)l$
3. 全面积 $T=\pi r_1(l+r_1)+\pi r_2(l+r_2)$
4. 体积 $V=\pi(r_1^2+r_2^2+r_1r_2)h/3$

球：

1. 全面积 $T=4\pi r^2$
2. 体积 $V=4\pi r^3/3$

球台：

1. 侧面积 $S=2\pi rh$
2. 全面积 $T=\pi(2rh+r_1^2+r_2^2)$
3. 体积 $V=\pi h(3(r_1^2+r_2^2)+h^2)/6$

球扇形：

1. 全面积 $T=\pi r(2h+r_0)$, h 为球冠高, r_0 为球冠底面半径
2. 体积 $V=2\pi r^2h/3$

```

1  /*
2  *  Graham 求凸包  $O(N * \log N)$ 
3  *  CALL:  $nr = \text{graham}(\text{pnt}, \text{int } n, \text{res});$   $\text{res}[]$ 为凸包点集;
4  */
5  struct point {
6      double x, y;
7  };
8
9  bool mult(point sp, point ep, point op) {
10     return (sp.x - op.x) * (ep.y - op.y) >= (ep.x - op.x) * (sp.y - op.y);
11 }
12
13 inline bool operator < (const point &l, const point &r) {
14     return l.y < r.y || (l.y == r.y && l.x < r.x);
15 }
16
17 int graham(point pnt[], int n, point res[]) {
18     int i, len, top = 1;
19     sort(pnt, pnt + n);
20     if (n == 0) {
21         return 0;
22     }
23     res[0] = pnt[0];
24     if (n == 1) {
25         return 1;
26     }
27     res[1] = pnt[1];
28     if (n == 2) {
29         return 2;
30     }
31     res[2] = pnt[2];
32     for (i = 2; i < n; i++) {
33         while (top && mult(pnt[i], res[top], res[top - 1])) {
34             top--;
35         }
36         res[++top] = pnt[i];
37     }
38     len = top;
39     res[++top] = pnt[n - 2];
40     for (i = n - 3; i >= 0; i--) {
41         while (top != len && mult(pnt[i], res[top], res[top - 1])) {
42             top--;
43         }
44         res[++top] = pnt[i];
45     }
46     return top; // 返回凸包中点的个数
47 }

```

10.3 四点共面

```

1  struct point {
2      double x, y, z;

```

```

3     point operator - (point &o) {
4         point ans;
5         ans.x = this->x - o.x;
6         ans.y = this->y - o.y;
7         ans.z = this->z - o.z;
8         return ans;
9     }
10 };
11
12 double dot_product(const point &a, const point &b) {
13     return a.x * b.x + a.y * b.y + a.z * b.z;
14 }
15
16 point cross_product(const point &a, const point &b) {
17     point ans;
18     ans.x = a.y * b.z - a.z * b.y;
19     ans.y = a.z * b.x - a.x * b.z;
20     ans.z = a.x * b.y - a.y * b.x;
21     return ans;
22 }
23
24 int main() {
25     point p[4];
26     int T;
27     for (scanf("%d", &T); T--;) {
28         for (int i = 0; i < 4; ++i) {
29             scanf("%lf%lf%lf", &p[i].x, &p[i].y, &p[i].z);
30         }
31         puts(dot_product(p[3] - p[0], cross_product(p[2] - p[0], p[1] - p[0])) == 0.0 ? "Yes\n"
32             : "No\n");
33     }
34     return 0;
35 }

```

10.4 多边形重心

```

1  /*
2  * 求多边形重心
3  * INIT: pnt[] 已按顺时针(或逆时针)排好序; / CALL: res = bcenter(pnt, n);
4  */
5  struct point {
6      double x, y;
7  };
8
9  point bcenter(point pnt[], int n) {
10     point p, s;
11     double tp, area = 0, tpx = 0, tpy = 0;
12     p.x = pnt[0].x;
13     p.y = pnt[0].y;
14     for (int i = 1; i <= n; ++i) { // point: 0 ~ n - 1
15         s.x = pnt[(i == n) ? 0 : i].x;
16         s.y = pnt[(i == n) ? 0 : i].y;

```

```
17     tp = (p.x * s.y - s.x * p.y);
18     area += tp / 2;
19     tpx += (p.x + s.x) * tp;
20     tpy += (p.y + s.y) * tp;
21     p.x = s.x;
22     p.y = s.y;
23 }
24 s.x = tpx / (6 * area);
25 s.y = tpy / (6 * area);
26 return s;
27 }
```

10.5 旋转卡壳

```
1 struct Point {
2     int x, y;
3     Point(int _x = 0, int _y = 0) {
4         x = _x;
5         y = _y;
6     }
7     Point operator - (const Point &b) const {
8         return Point(x - b.x, y - b.y);
9     }
10    int operator ^(const Point &b) const {
11        return x * b.y - y * b.x;
12    }
13    int operator *(const Point &b) const {
14        return x * b.x + y * b.y;
15    }
16    void input() {
17        scanf("%d%d", &x, &y);
18        return ;
19    }
20 };
21
22 // 距离的平方
23 int dist2(Point a, Point b) {
24     return (a - b) * (a - b);
25 }
26
27 // 二维凸包, int
28 const int MAXN = 50010;
29 Point list[MAXN];
30 int Stack[MAXN], top;
31 bool _cmp(Point p1, Point p2) {
32     int tmp = (p1 - list[0]) ^ (p2 - list[0]);
33     if (tmp > 0) {
34         return true;
35     }
36     else if (tmp == 0 && dist2(p1, list[0]) <= dist2(p2, list[0])) {
37         return true;
38     } else {
```

```
39     return false;
40 }
41 }
42
43 void Graham(int n) {
44     Point p0;
45     int k = 0;
46     p0 = list[0];
47     for (int i = 1; i < n; i++) {
48         if (p0.y > list[i].y || (p0.y == list[i].y && p0.x > list[i].x)) {
49             p0 = list[i];
50             k = i;
51         }
52     }
53     swap(list[k], list[0]);
54     sort(list + 1, list + n, _cmp);
55     if (n == 1) {
56         top = 1;
57         Stack[0] = 0;
58         return ;
59     }
60     if (n == 2) {
61         top = 2;
62         Stack[0] = 0;
63         Stack[1] = 1;
64         return ;
65     }
66     Stack[0] = 0;
67     Stack[1] = 1;
68     top = 2;
69     for (int i = 2; i < n; i++) {
70         while (top > 1 && ((list[Stack[top - 1]] - list[Stack[top - 2]]) ^ (list[i] - list[
71             Stack[top - 2]])) <= 0) {
72             top--;
73         }
74         Stack[top++] = i;
75     }
76     return ;
77 }
78 // 旋转卡壳, 求两点间距离平方的最大值
79 int rotating_calipers(Point p[], int n) {
80     int ans = 0;
81     Point v;
82     int cur = 1;
83     for (int i = 0; i < n; i++) {
84         v = p[i] - p[(i + 1) % n];
85         while ((v ^ (p[(cur + 1) % n] - p[cur])) < 0) {
86             cur = (cur + 1) % n;
87         }
88         ans = max(ans, max(dist2(p[i], p[cur]), dist2(p[(i + 1) % n], p[(cur + 1) % n])));
89     }
90     return ans;
91 }
```



```
92
93 Point p[MAXN];
94
95 int main() {
96     int n;
97     while (scanf("%d", &n) == 1) {
98         for (int i = 0; i < n; i++) {
99             list[i].input();
100         }
101         Graham(n);
102         for (int i = 0; i < top; i++) {
103             p[i] = list[Stack[i]];
104         }
105         printf("%d\n", rotating_calipers(p, top));
106     }
107     return 0;
108 }
```

10.6 模拟退火

```
1 //模拟退火
2 //平面上找一个点 使得  $\sigma(1..N)dist(a, i)*w_i$  最小
3
4 #include <cstdlib>
5 #include <cstdio>
6 #include <cstring>
7 #include <cmath>
8
9 #define INF (1e17)
10 #define EPS (1e-3)
11 #define PI (acos(-1.0))
12 #define FIRE(x) (x *= 0.99)
13 using namespace std;
14 const int MAXN = 10000 + 10;
15 int N;
16 double total = INF;
17 struct Point {
18     double x, y, w;
19     Point (double _x, double _y) : x(_x), y(_y) {}
20     Point (void) {}
21     void Read(void) {
22         scanf("%lf%lf%lf", &x, &y, &w);
23     }
24     void operator += (Point t) {
25         x += t.x; y += t.y;
26     }
27     void operator /= (int N) {
28         x /= N, y /= N;
29     }
30 };
31 Point now, ans, point[MAXN];
32 inline double Dist(Point a, Point b) {
```

```

33     return sqrt((a.x - b.x) * (a.x - b.x) +
34                (a.y - b.y) * (a.y - b.y));
35 }
36 inline double Statistic(Point p) {
37     double res = 0.0;
38     for (int i = 0; i < N; i++) res += Dist(p, point[i]) * point[i].w;
39     if (res < total) total = res, ans = p;
40     return res;
41 }
42 inline double Rand(void) {
43     return (rand() % 1000 + 1) / 1000.0;
44 }
45 int main(void) {
46     srand(10086);
47     scanf("%d", &N);
48     register int i;
49     for (i = 0; i < N; i++) point[i].Read(), now += point[i];
50     now /= N;
51     double T = 100000.0, alpha, sub;
52     while (T > EPS) {
53         alpha = 2.0 * PI * Rand();
54         Point tmp(now.x + T * cos(alpha), now.y + T * sin(alpha));
55         sub = Statistic(now) - Statistic(tmp);
56         if(sub >= 0 || exp(sub / T) >= Rand()) now = tmp;
57         FIRE(T);
58     }
59     T = 0.001;
60     for (i = 1; i <= 1000; ++i) {
61         alpha = 2.0 * PI * Rand();
62         Point tmp(ans.x + T * cos(alpha) * Rand(), ans.y + T * sin(alpha) * Rand());
63         Statistic(tmp);
64     }
65     printf("%.3lf%.3lf\n", ans.x, ans.y);
66     return 0;
67 }

```

10.7 半平面交

```

1  #include <cstdio>
2  #include <algorithm>
3  #include <queue>
4  #include <cmath>
5  using namespace std;
6  const double eps = 1e-8;
7  struct Point{
8      double x,y;
9      Point(double xx=0.0,double yy=0.0) :x(xx) ,y(yy) {}
10     Point operator - (const Point &b) const {
11         return Point(x-b.x,y-b.y) ;
12     }
13     Point operator +(const Point &b) const {
14         return Point(x+b.x,y+b.y) ;

```

```

15     }
16     Point operator /(const double &b) const {
17         return Point(x/b,y/b) ;
18     }
19     Point operator *(const double &b) const {
20         return Point(x*b,y*b) ;
21     }
22     double operator ^(const Point &b) const {
23         return x*b.y-y*b.x;
24     }
25 };
26 typedef Point myvec;
27 double cross(myvec a,myvec b) {
28     return a^b;
29 }
30 struct Line{
31     Point p;
32     myvec v;
33     double ang;
34     Line() {}
35     Line( Point pp,myvec vv) :p(pp) ,v(vv) {}
36     bool operator < (const Line &l) const {
37         return ang < l.ang;
38     }
39
40 };
41 //点p在有向直线L的左边 (线上不算)
42 bool on_left( Line l,Point p) {
43     return cross(l.v,p-l.p) >0;
44 }
45 //直线交点 假设交点唯一存在
46 Point get_inter_section(Line a,Line b) {
47     myvec u = a.p - b.p;
48     double t = cross(b.v,u) /cross(a.v,b.v) ;
49     return a.p+a.v*t;
50
51 }
52 int half_plane_inter_section(Line *L,int n,Point *poly) {
53     sort(L,L+n) ;//级角排序
54     int fir,lst;//双向队列的第一个元素和最后一个元素的下标
55     Point *p = new Point[n];//p[i] 为q[i]和q[i+1]的交点
56     Line *q = new Line[n];//双端队列
57     q[ fir = lst = 0 ] = L[0];//双端队列初始化为只有一个半平面的L[0]
58     for( int i =1; i <n ; ++i)
59     {
60         while( fir < lst && !on_left(L[i],p[lst-1]) )
61             lst--;
62         while( fir<lst && !on_left(L[i],p[fir]) )
63             fir++;
64         q[++lst] = L[i];
65         if( fabs( cross(q[lst].v,q[lst-1].v) ) < eps ) { //两向量平行且同向 取内侧一个
66             lst--;
67             if( on_left(q[lst],L[i].p) )
68                 q[lst] = L[i];

```

```

69     }
70     if( fir < lst )
71         p[lst-1] = get_inter_section(q[lst-1],q[lst]) ;
72 }
73 while( fir< lst && !on_left(q[fir],p[lst-1]) )
74     lst--; //删除无用的平面
75 if(lst - fir <=1 )
76     return 0; //空集
77 p[lst] = get_inter_section(q[lst],q[fir]) ; //计算首尾两个半平面的交点
78 //从 deque 复制到输出中
79 int m = 0 ;
80 for( int i = fir;i<=lst;++i)
81     poly[m++] = p[i];
82 return m;
83 }

```

10.8 计算几何

```

1 //上交计算几何算法
2 /*****
3  * COMPUTATIONAL GEOMETRY ROUTINES
4  * WRITTEN BY : LIU Yu (C) 2003
5  *****/
6 // 叉乘
7 // 两个点的距离
8 // 点到直线距离
9 // 返回直线  $Ax + By + C = 0$  的系数
10 // 线段
11 // 圆
12 // 两个圆的公共面积
13 // 矩形
14 // 根据下标返回多边形的边
15 // 两个矩形的公共面积
16 // 多边形 , 逆时针或顺时针给出x,y
17 // 多边形顶点
18 // 多边形的边
19 // 多边形的周长
20 // 判断点是否在线段上
21 // 判断两条线断是否相交, 端点重合算相交
22 // 判断两条线断是否平行
23 // 判断两条直线断是否相交
24 // 直线相交的交点
25 // 判断是否简单多边形
26 // 求多边形面积
27 // 判断是否在多边形上
28 // 判断是否在多边形内部
29 // 点阵的凸包, 返回一个多边形
30 // 最近点对的距离
31 #include <cmath>
32 #include <cstdio>
33 #include <memory>
34 #include <algorithm>

```

```

35 #include <iostream>
36 using namespace std;
37 typedef double TYPE;
38 #define Abs(x) (((x)>0)?(x):(-(x)))
39 #define Sgn(x) (((x)<0)?(-1):(1))
40 #define Max(a,b) (((a)>(b))?(a):(b))
41 #define Min(a,b) (((a)<(b))?(a):(b))
42 #define Epsilon 1e-10
43 #define Infinity 1e+10
44 #define Pi 3.14159265358979323846
45 TYPE Deg2Rad(TYPE deg) {
46     return (deg * Pi / 180.0);
47 }
48 TYPE Rad2Deg(TYPE rad) {
49     return (rad * 180.0 / Pi);
50 }
51 TYPE Sin(TYPE deg) {
52     return sin(Deg2Rad(deg));
53 }
54 TYPE Cos(TYPE deg) {
55     return cos(Deg2Rad(deg));
56 }
57 TYPE ArcSin(TYPE val) {
58     return Rad2Deg(asin(val));
59 }
60 TYPE ArcCos(TYPE val) {
61     return Rad2Deg(acos(val));
62 }
63 TYPE Sqrt(TYPE val) {
64     return sqrt(val);
65 }
66 struct POINT {
67     TYPE x;
68     TYPE y;
69     TYPE z;
70     POINT() : x(0), y(0), z(0) {};
71     POINT(TYPE _x_, TYPE _y_, TYPE _z_ = 0) : x(_x_), y(_y_), z(_z_) {};
72 };
73 // cross product of (o→a) and (o→b)// 叉乘
74 TYPE Cross(const POINT & a, const POINT & b, const POINT & o) {
75     return (a.x - o.x) * (b.y - o.y) - (b.x - o.x) * (a.y - o.y);
76 }
77 // planar points' distance// 两个点的距离
78 TYPE Distance(const POINT & a, const POINT & b) {
79     return Sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y) + (a.z - b.z) * (a.z
80         - b.z));
81 }
82 struct LINE {
83     POINT a;
84     POINT b;
85     LINE() {};
86     LINE(POINT _a_, POINT _b_) : a(_a_), b(_b_) {};
87 //点到直线距离
88 double PointToLine(POINT p0 ,POINT p1 ,POINT p2 ,POINT &cp) {

```

```

89     double d = Distance(p1 ,p2);
90     double s = Cross(p1 ,p2 ,p0) / d;
91     cp.x = p0.x + s*( p2.y-p1.y) / d;
92     cp.y = p0.y - s*( p2.x-p1.x) / d;
93     return Abs(s);
94 }
95 // 返回直线 Ax + By + C =0 的系数
96 void Coefficient(const LINE & L, TYPE & A, TYPE & B, TYPE & C) {
97     A = L.b.y - L.a.y;
98     B = L.a.x - L.b.x;
99     C = L.b.x * L.a.y - L.a.x * L.b.y;
100 }
101 void Coefficient(const POINT & p,const TYPE a,TYPE & A,TYPE & B,TYPE & C) {
102     A = Cos(a);    // 线段
103     B = Sin(a);
104     C = - (p.y * B + p.x * A);
105 }
106 struct SEG {
107     POINT a;
108     POINT b;
109     SEG() {} ;
110     SEG(POINT _a_, POINT _b_):a(_a_),b(_b_) {} ;
111 };
112 // 圆
113 struct CIRCLE {
114     TYPE x;
115     TYPE y;
116     TYPE r;
117     CIRCLE() {}
118     CIRCLE(TYPE _x_, TYPE _y_, TYPE _r_) : x(_x_), y(_y_), r(_r_) {} ;
119 POINT Center(const CIRCLE & circle) {
120     return POINT(circle.x, circle.y);
121 }
122 TYPE Area(const CIRCLE & circle) {
123     return Pi * circle.r * circle.r;    //两个圆的公共面积
124 }
125 TYPE CommonArea(const CIRCLE & A, const CIRCLE & B) {
126     TYPE area = 0.0;
127     const CIRCLE & M = (A.r > B.r) ? A : B;
128     const CIRCLE & N = (A.r > B.r) ? B : A;
129     TYPE D = Distance(Center(M), Center(N));
130     if ((D < M.r + N.r) && (D > M.r - N.r)) {
131         TYPE cosM = (M.r * M.r + D * D - N.r * N.r) / (2.0 * M.r * D);
132         TYPE cosN = (N.r * N.r + D * D - M.r * M.r) / (2.0 * N.r * D);
133         TYPE alpha = 2.0 * ArcCos(cosM);
134         TYPE beta  = 2.0 * ArcCos(cosN);
135         TYPE TM = 0.5*M.r*M.r*Sin(alpha);
136         TYPE TN=0.5 * N.r * N.r * Sin(beta);
137         TYPE FM = (alpha / 360.0) * Area(M);
138         TYPE FN = (beta / 360.0) * Area(N);
139         area = FM + FN - TM - TN;
140     } else if (D <= M.r - N.r) {
141         area = Area(N);
142     }

```

```

143     return area;
144 }
145 // 矩形
146 // 矩形的线段
147 //      2
148 //      _____ b
149 //      |           |
150 //  3 |           | 1
151 //  a _____
152 //      0
153 struct RECT {
154     POINT a; // 左下点 POINT b; // 右上点
155     RECT() {};
156     RECT(const POINT & _a_, const POINT & _b_) {
157         a = _a_;
158         b = _b_;
159     }
160 };
161 //根据下标返回多边形的边
162 SEG Edge(const RECT & rect, int idx) {
163     SEG edge;
164     while (idx < 0) idx += 4;
165     switch (idx % 4) {
166     case 0:
167         edge.a = rect.a;
168         edge.b = POINT(rect.b.x, rect.a.y);
169         break;
170     case 1:
171         edge.a = POINT(rect.b.x, rect.a.y);
172         edge.b = rect.b;
173         break;
174     case 2:
175         edge.a = rect.b;
176         edge.b = POINT(rect.a.x, rect.b.y);
177         break;
178     case 3:
179         edge.a = POINT(rect.a.x, rect.b.y);
180         edge.b = rect.a;
181         break;
182     default:
183         break;
184     }
185     return edge;
186 }
187 TYPE Area(const RECT & rect) {
188     return (rect.b.x - rect.a.x) * (rect.b.y - rect.a.y);
189 }
190 // 两个矩形的公共面积
191 TYPE CommonArea(const RECT & A, const RECT & B) {
192     TYPE area = 0.0;
193     POINT LL(Max(A.a.x, B.a.x), Max(A.a.y, B.a.y));
194     POINT UR(Min(A.b.x, B.b.x), Min(A.b.y, B.b.y));
195     if ((LL.x <= UR.x) && (LL.y <= UR.y)) {
196         area = Area(RECT(LL, UR));

```

```

197     }
198     return area;
199 }// 多边形 , 逆时针或顺时针给出x,y
200 struct POLY {
201     int n; //n个点  TYPE * x;  //x,y为点的指针, 首尾必须重合  TYPE * y;
202     POLY() : n(0), x(NULL), y(NULL) {};
203     POLY(int _n_, const TYPE * _x_, const TYPE * _y_) {
204         n = _n_;
205         x = new TYPE[n + 1];
206         memcpy(x, _x_, n*sizeof(TYPE));
207         x[n] = _x_[0];
208         y = new TYPE[n + 1];
209         memcpy(y, _y_, n*sizeof(TYPE));
210         y[n] = _y_[0];
211     }
212 };//多边形顶点
213 POINT Vertex(const POLY & poly, int idx) {
214     idx %= poly.n;    //多边形的边
215     return POINT(poly.x[idx], poly.y[idx]);
216 }
217 SEG Edge(const POLY & poly, int idx) {
218     idx %= poly.n;
219     return SEG(POINT(poly.x[idx], poly.y[idx]),
220               POINT(poly.x[idx + 1], poly.y[idx + 1]));
221 } //多边形的周长
222 TYPE Perimeter(const POLY & poly) {
223     TYPE p = 0.0;
224     for (int i = 0; i < poly.n; i++)
225         p = p + Distance(Vertex(poly, i), Vertex(poly, i + 1));
226     return p;
227 }
228 bool IsEqual(TYPE a, TYPE b) {
229     return (Abs(a - b) < Epsilon);
230 }
231 bool IsEqual(const POINT & a, const POINT & b) {
232     return (IsEqual(a.x, b.x) && IsEqual(a.y, b.y));
233 }
234 bool IsEqual(const LINE & A, const LINE & B) {
235     TYPE A1, B1, C1;
236     TYPE A2, B2, C2;
237     Coefficient(A, A1, B1, C1);
238     Coefficient(B, A2, B2, C2);
239     return IsEqual(A1 * B2, A2 * B1) &&
240           IsEqual(A1 * C2, A2 * C1) && IsEqual(B1 * C2, B2 * C1);
241 }
242 // 判断点是否在线段上
243 bool IsOnSeg(const SEG & seg, const POINT & p) {
244     return (IsEqual(p, seg.a) || IsEqual(p, seg.b)) ||
245           (((p.x - seg.a.x) * (p.x - seg.b.x) < 0 || (p.y - seg.a.y) * (p.y - seg.b.y) < 0)
246            &&
247            (IsEqual(Cross(seg.b, p, seg.a), 0)));
248 }
249 //判断两条线断是否相交, 端点重合算相交
250 bool IsIntersect(const SEG & u, const SEG & v) {

```



```

250     return (Cross(v.a, u.b, u.a) * Cross(u.b, v.b, u.a) >= 0) &&
251             (Cross(u.a, v.b, v.a) * Cross(v.b, u.b, v.a) >= 0) &&(Max(u.a.x, u.b.x)>=Min(v.
252                 a.x, v.b.x))&&(Max(v.a.x, v.b.x)>= Min(u.a.x,u.b.x))&&(Max(u.a.y, u.b.y)>=
                Min(
253                     v.a.y, v.b.y))&&(Max(v.a.y, v.b.y)>=Min(u.a.y, u.b.y));
254 }
255 //判断两条线断是否平行
256 bool IsParallel(const LINE & A, const LINE & B) {
257     TYPE A1, B1, C1;
258     TYPE A2, B2, C2;
259     Coefficient(A, A1, B1, C1);
260     Coefficient(B, A2, B2, C2);
261     return (A1*B2== A2*B1) &&((A1 * C2 != A2 * C1) || (B1 * C2 != B2 * C1));
262 }
263 //判断两条直线断是否相交
264 bool IsIntersect(const LINE & A, const LINE & B) {
265     return !IsParallel(A, B);    //直线相交的交点
266 }
267
268 POINT Intersection(const LINE & A, const LINE & B) {
269     TYPE A1, B1, C1;
270     TYPE A2, B2, C2;
271     Coefficient(A, A1, B1, C1);
272     Coefficient(B, A2, B2, C2);
273     POINT I(0, 0);
274     I.x = - (B2 * C1 - B1 * C2) / (A1 * B2 - A2 * B1);
275     I.y =  (A2 * C1 - A1 * C2) / (A1 * B2 - A2 * B1);
276     return I;
277 }
278
279 bool IsInCircle(const CIRCLE & circle, const RECT & rect) {
280     return (circle.x - circle.r >= rect.a.x) &&
281            (circle.x + circle.r <= rect.b.x) &&(circle.y - circle.r >= rect.a.y) &&
282            (circle.y + circle.r <= rect.b.y);
283 }
284 //判断是否简单多边形
285 bool IsSimple(const POLY & poly) {
286     if (poly.n < 3) return false;
287     SEG L1, L2;
288     for (int i = 0; i < poly.n - 1; i++) {
289         L1 = Edge(poly, i);
290         for (int j = i + 1; j < poly.n; j++) {
291             L2 = Edge(poly, j);
292             if (j == i+1) {
293                 if (IsOnSeg(L1, L2.b)||IsOnSeg(L2, L1.a)) return false;
294             } else if (j == poly.n - i - 1) {
295                 if (IsOnSeg(L1, L2.a) || IsOnSeg(L2, L1.b)) return false;
296             } else {
297                 if (IsIntersect(L1, L2)) return false;    // for i
298             }
299         } // for j
300     }
301     return true;
302 }

```

```
303 //求多边形面积
304 TYPE Area(const POLY & poly) {
305     if (poly.n < 3) return TYPE(0);
306     double s = poly.y[0] * (poly.x[poly.n - 1] - poly.x[1]);
307     for (int i = 1; i < poly.n; i++) {
308         s += poly.y[i] * (poly.x[i - 1] - poly.x[(i + 1) % poly.n]);
309     }
310     return s/2;
311 }
312 //判断是否在多边形上
313 bool IsOnPoly(const POLY & poly, const POINT & p) {
314     for (int i = 0; i < poly.n; i++) {
315         if (IsOnSeg(Edge(poly, i), p)) return true;
316     }
317     return false;
318 }
319 //判断是否在多边形内部
320 bool IsInPoly(const POLY & poly, const POINT & p) {
321     SEG L(p, POINT(Infinity, p.y));
322     int count = 0;
323     for (int i = 0; i < poly.n; i++) {
324         SEG S = Edge(poly, i);
325         if (IsOnSeg(S, p)) {
326             return false; //如果想让 在poly上则返回 true,则改为 true
327         }
328         if (!IsEqual(S.a.y, S.b.y)) {
329             POINT & q = (S.a.y > S.b.y)?(S.a):(S.b);
330             if (IsOnSeg(L, q)) {
331                 ++count;
332             }
333             else if (!IsOnSeg(L, S.a)&&!IsOnSeg(L, S.b)&&IsIntersect(S, L)) {
334                 ++count;
335             }
336         }
337     }
338     return (count % 2 != 0);
339 }
340 // 点阵的凸包, 返回一个多边形
341 POLY ConvexHull(const POINT * set, int n) {
342     POINT * points = new POINT[n];
343     memcpy(points, set, n * sizeof(POINT));
344     TYPE * X = new TYPE[n];
345     TYPE * Y = new TYPE[n];
346     int i, j, k = 0, top = 2;
347     for(i = 1; i < n; i++) {
348         if((points[i].y<points[k].y)||((points[i].y==points[k].y)&&
349                                     (points[i].x<points[k].x))) {
350             k = i;
351         }
352     }
353     std::swap(points[0], points[k]);
354     for (i = 1; i < n - 1; i++) {
355         k = i;
356         for (j = i + 1; j < n; j++) {
```

```

357         if ((Cross(points[j], points[k], points[0]) > 0) || ((Cross(points[j], points[k],
358             points[0]) == 0) && (Distance(points[0], points[j]) < Distance(points[0],
359                 points[k]
360                     ))) {
361             k = j;
362         }
363         std::swap(points[i], points[k]);
364     }
365     X[0] = points[0].x;
366     Y[0] = points[0].y;
367     X[1] = points[1].x;
368     Y[1] = points[1].y;
369     X[2] = points[2].x;
370     Y[2] = points[2].y;
371     for (i = 3; i < n; i++) {
372         while (Cross(points[i], POINT(X[top], Y[top]), POINT(X[top
373             - 1], Y[top - 1])) >= 0) {
374             top--;
375         }
376         ++top;
377         X[top] = points[i].x;
378         Y[top] = points[i].y;
379     }
380     delete [] points;
381     POLY poly(++top, X, Y);
382     delete [] X;
383     delete [] Y;
384     return poly;
385 }
386 //最近点对的距离, Written By PrincessSnow
387 #define MAXN 100000
388 POINT pt[MAXN];
389 bool cmp(POINT n1, POINT n2) {
390     return (n1.x < n2.x || n1.x == n2.x && n1.y < n2.y);
391 }
392 double Get(double dis, int mid, int start, int end) {
393     int s = mid, e = mid, i, j;
394     double t;
395     while (s > start && pt[mid].x - pt[s].x <= dis) s--;
396     while (e < end && pt[e].x - pt[mid].x <= dis) e++;
397     for (i = s; i <= e; i++)
398         for (j = i + 1; j <= e && j <= i + 7; j++) {
399             t = Distance(pt[i], pt[j]);
400             if (t < dis) dis = t;
401         }
402     return dis;
403 }
404 double ClosestPairDistance(int start, int end) {
405     int m = end - start + 1, mid, i;
406     double t1, t2, dis = -1, t;
407     if (m <= 3) {
408         for (i = start; i < end; i++) {
409             t = Distance(pt[i], pt[i + 1]);

```

```

410         if(t < dis || dis == -1)      dis = t;
411     }
412     t = Distance(pt[start] , pt[end]);
413     if(t < dis) dis=t;
414     return dis;
415 }
416 if(m%2 == 0)      mid = start + m/2 - 1;
417 else              mid = start + m/2;
418 if(m%2 == 0) {
419     t1 = ClosestPairDistance(start, mid);
420     t2 = ClosestPairDistance(mid+1, end);
421 } else {
422     t1 =ClosestPairDistance(start,mid);
423     t2=ClosestPairDistance(mid+1,end);
424 }
425 if(t1 < t2)      dis = t1;
426 else              dis = t2;
427 dis = Get(dis, mid, start, end);
428 return dis;
429 }
430
431
432 //1. 球面上两点最短距离
433 // 计算圆心角Lat 表示纬度, -90 <= w <= 90, Lng 表示经度
434 // 返回两点所在大圆劣弧对应圆心角, 0 <= angle <= pi
435 double angle(double lng1, double lat1, double lng2, double lat2) {
436     double dlng = fabs(lng1 - lng2) * pi / 180;
437     while(dlng >= pi+pi)      dlng -= pi+pi;
438     if(dlng > pi)      dlng = pi + pi - dlng;
439     lat1 *= pi / 180,      lat2 *= pi / 180;
440     return acos( cos(lat1)*cos(lat2)*cos(dlng) + sin(lat1)*sin(lat2) );
441 }
442 // 计算距离, r为球半径
443 double line_dist(double r, double lng1, double lat1, double lng2, double lat2) {
444     double dlng = fabs(lng1 - lng2) * pi / 180;
445     while(dlng >= pi+pi)      dlng -= pi+pi;
446     if(dlng > pi)      dlng = pi + pi - dlng;
447     lat1 *= pi / 180,      lat2 *= pi / 180;
448     return r*sqrt(2-2*( cos(lat1)*cos(lat2)*cos(dlng)+ sin(lat1)*sin(lat2)) );
449 }
450 // 计算球面距离, r为球半径
451 double sphere_dist(double r, double lng1, double lat1, double lng2, double lat2) {
452     return r * angle(lng1, lat1, lng2, lat2);
453 }
454
455
456 //2. 三点求圆心坐标
457 double GetRadiusBy3Points(double x1, double y1,
458                             double x2, double y2, double x3, double y3, double &x, double &y) {
459     // 由 ( x - x1 )^2 + ( y - y1 )^2 = ( x - x2 )^2 + ( y - y2 )^2 得
460     // 2*( x2 - x1 )*x + 2*( y2 - y1 )*y = x2^2 - x1^2 + y2^2 - y1^2
461     // 同理得
462     // 2*( x3 - x2 )*x + 2*( y3 - y2 )*y = x3^2 - x2^2 + y3^2 - y2^2
463     // 由行列式解方程得 x , y

```

```
464     double a11, a12, a21, a22, b1, b2;
465     double d, d1, d2 ;
466     a11 = 2 * ( x3 - x2 );
467     a12 = 2 * ( y3 - y2 );
468     a21 = 2 * ( x2 - x1 );
469     a22 = 2 * ( y2 - y1 );
470     b1 = x3*x3 - x2*x2 + y3*y3 - y2*y2;
471     b2 = x2*x2 - x1*x1 + y2*y2 - y1*y1;
472     d = a11*a22 - a12*a21;
473     d1 = b1*a22 - a12*b2;
474     d2 = a11*b2 - b1*a21;
475     // x , y 是圆心坐标
476     x = d1 / d;
477     y = d2 / d;
478     return (x1 - x)*(x1 - x) + (y1 - y)*(y1 - y);
479 }
480 //
481 //
482 //3. 三角形几个重要的点
483 //设三角形的三条边为a, b, c, 且不妨假设a <= b <= c
484 //三角形的面积可以根据海伦公式算得, 如下:
485 //s = sqrt(p * (p - a) * (p - b) * (p - c)), p = (a + b + c) / 2
486 //1. 费马点(该点到三角形三个顶点的距离之和最小)
487 //有个有趣的结论: 若三角形的三个内角均小于120度,
488 //那么该点连接三个顶点形成的三个角均为120度; 若三角形存在一个内角
489 //大于120度, 则该顶点就是费马点)
490 //计算公式如下:
491 //若有一个内角大于120度(这里假设为角C), 则距离为a + b
492 //若三个内角均小于120度, 则距离为
493 //sqrt((a * a + b * b + c * c + 4 * sqrt(3.0) * s) / 2), 其中
494 //2. 内心——角平分线的交点
495 //令x = (a + b - c) / 2, y = (a - b + c) / 2, z = (-a + b + c) / 2, h = s / p
496 // 计算公式为 sqrt(x * x + h * h) + sqrt(y * y + h * h) + sqrt(z * z + h * h)
497 //3. 重心——中线的交点
498 //ACM算法模板集
499 // - 46 -
500 //计算公式如下:
501 //2.0 / 3 * (sqrt((2 * (a * a + b * b) - c * c) / 4)
502 //          + sqrt((2 * (a * a + c * c) - b * b) / 4)
503 //          + sqrt((2 * (b * b + c * c) - a * a) / 4))
504 //4. 垂心——垂线的交点
505 //计算公式如下:
506 //3 * (c / 2 / sqrt(1 - cosC * cosC))
```

第十一章 类

11.1 点类

```
1 struct point {
2     double x, y;
3     point() { };
4     point(double x, double y) :x(x), y(y) { }
5     point operator - (const point &b) const {
6         return point(x - b.x, y - b.y);
7     }
8     point operator + (const point &b) const {
9         return point(x + b.x, y + b.y);
10    }
11    point operator * (const double k) const {
12        return point(k * x, k * y);
13    }
14    point operator / (const double k) const {
15        return point(x / k, y / k);
16    }
17    double slope() {
18        return y / x;
19    }
20 };
```

11.2 分数类

```
1 struct Fraction {
2     long long num;
3     long long den;
4     Fraction(long long num=0, long long den=1) {
5         if(den<0) {
6             num=-num;
7             den=-den;
8         }
9         assert(den!=0);
10        long long g=gcd(abs(num),den);
11        this->num=num/g;
12        this->den=den/g;
13    }
14    Fraction operator +(const Fraction &o) const {
15        return Fraction(num*o.den+o.num,den*o.den);
16    }
```

```

17     Fraction operator -(const Fraction &o) const {
18         return Fraction(num*o.den-den*o.num,den*o.den);
19     }
20     Fraction operator *(const Fraction &o) const {
21         return Fraction(num*o.num,den*o.den);
22     }
23     Fraction operator /(const Fraction &o) const {
24         return Fraction(num*o.den,den*o.num);
25     }
26     bool operator <(const Fraction &o) const {
27         return num*o.den< den*o.num;
28     }
29     bool operator ==(const Fraction &o) const {
30         return num*o.den==den*o.num;
31     }
32 };

```

11.3 矩阵

```

1  #define maxm 10
2  typedef long long LL;
3
4  const LL Mod=1e9+7;
5  struct Matrix {
6      int n, m;
7      LL mat[maxm][maxm];
8      void clear() {
9          memset(mat, 0, sizeof(mat));
10     }
11
12     Matrix(int n, int m) :n(n), m(m) {
13         //不要设置默认构造函数，让编译器检查初始化遗漏
14         clear();
15     }
16
17     Matrix operator +(const Matrix &M) const {
18         Matrix res(n, m);
19         for (LL i = 0; i < n; ++i) for (LL j = 0; j < m; ++j) {
20             res.mat[i][j] = (mat[i][j] + M.mat[i][j]) % Mod;
21         }
22         return res;
23     }
24
25     Matrix operator *(const Matrix &M) const {
26         if (m != M.n){
27             std::cout << "Wrong!" << std::endl;
28             return Matrix(-1, -1);
29         }
30         Matrix res(n, M.m);
31         res.clear();
32         int i,j,k;
33         for (i = 0; i < n; ++i)

```

```

34         for (j = 0; j < M.m; ++j)
35             for (k = 0; k < m; ++k) {
36                 res.mat[i][j] += mat[i][k] * M.mat[k][j]%Mod;
37                 res.mat[i][j] %= Mod;
38             }
39         return res;
40     }
41     Matrix operator *(const LL &x) const {
42         Matrix res(n,m);
43         int i,j;
44         std::cout << n << ' ' << m << std::endl;
45         for (i = 0; i < n; ++i)
46             for (j = 0; j < m; ++j)
47                 res[i][j] = mat[i][j] * x % Mod;
48         return res;
49     }
50
51     Matrix operator ^(LL b) const { // 矩阵快速幂 , 取余Mod
52         if (n != m)
53             return Matrix(-1, -1);
54         Matrix a(*this);
55         Matrix res(n, n);
56         res.clear();
57         for (LL i = 0; i < n; ++i)
58             res.mat[i][i] = 1;
59         for (; b; b >>= 1) {
60             if (b & 1) {
61                 res = a * res;
62             }
63             a = a * a;
64         }
65         return res;
66     }
67
68     LL* operator [] (int i) {
69         return mat[i];
70     }
71
72     void Print() const {
73         for (int i = 0; i < n; ++i) {
74             for (int j = 0; j < m; ++j)
75                 std::cout << mat[i][j] << ' ';
76             std::cout << '\n';
77         }
78     }
79 };

```

11.4 01 矩阵

```

1 #include <bitset>
2 #define maxn 1000
3 struct Matrix01{

```



```

4     int n,m;
5     std::bitset<maxn> a[maxn];
6     void Resize(int x,int y){
7         n=x;
8         m=y;
9     }
10    std::bitset<maxn>& operator [] (int n) {
11        return a[n];
12    }
13    void print(){
14        for(int i = 0; i < n; ++i)
15            std::cout << a[i] << std::endl;
16    }
17 };
18
19 Matrix01 operator & (Matrix01 &a,Matrix01 &b){ int i,j,k;
20     Matrix01 c;
21     c.Resize(a.n,b.m);
22     for(i = 0; i < a.n; ++i) {
23         c[i].reset();
24         for(j = 0; j < b.m; ++j)
25             if(a[i][j])
26                 c[i]|=b[j];
27     }
28     return c;
29 }

```

11.5 简单大数

```

1  /*
2   * Author: Simon
3   * 功能：基础大数运算，结构体排序，减法
4   */
5  class Bignum{
6  public:
7      static const maxn=9999;
8      static const int maxsize=10000;
9      static const int dlen=4;
10     int a[maxsize],len;
11     Bignum(){len=1;memset(a,0,sizeof(a));};
12     Bignum(const int);
13     Bignum(const Bignum&);
14     Bignum &operator =(const Bignum&);
15
16     Bignum operator -(const Bignum &)const;
17     bool operator >(const Bignum&T)const;
18     bool operator >(const int &t)const;
19     friend istream&operator >>(istream &in,Bignum &b){
20         char ch[maxsize*4];
21         int i=-1;
22         in>>ch;

```

```

23     int l=strlen(ch);
24     int count=0,sum=0;
25     for(int i=l-1;i>=0;){
26         sum=0;
27         int t=1;
28         for(int j=0;j<4&& i>=0;j++,i--,t*=10){
29             sum+=(ch[i]-'0')*t;
30         }
31         b.a[count]=sum;
32         count++;
33     }
34     b.len=count++;
35     return in;
36 }
37 friend ostream& operator <<(ostream& out,Bignum &b){
38     int i;
39     out<<b.a[b.len-1];
40     for(int i=b.len-2;i>=0;i--){
41         out.width(dlen);
42         out.fill('0');
43         out<<b.a[i];
44     }
45     return out;
46 }
47 };
48 Bignum::Bignum(const int b){
49     int c,d=b;
50     len=0;memset(a,0,sizeof(a));
51     while(d>maxn){
52         c=d-(d/(maxn+1))*(maxn+1);
53         d=d/(maxn+1);
54         a[len++]=c;
55     }
56     a[len++]=d;
57 }
58 Bignum::Bignum(const Bignum&T):len(T.len){
59     int i;memset(a,0,sizeof(a));
60     for(int i=0;i<len;i++) a[i]=T.a[i];
61 }
62 Bignum &Bignum::operator=(const Bignum&n){
63     int i;
64     len=n.len;
65     memset(a,0,sizeof(a));
66     for(int i=0;i<len;i++) a[i]=n.a[i];
67     return *this;
68 }
69 Bignum Bignum::operator-(const Bignum&T)const{
70     int i,j,big; bool flag;
71     Bignum t1,t2;
72     if(*this>T){
73         t1=*this;t2=T;
74         flag=0;
75     } else{
76         t1=T;t2=*this;

```

```

77         flag=1;
78     }
79     big=t1.len;
80     for(int i=0;i<big;i++){
81         if(t1.a[i]<t2.a[i]){
82             j=i+1;
83             while(t1.a[j]==0) j++;
84             t1.a[j--]--;
85             while(j>i) t1.a[j--]+=maxn;
86             t1.a[i]+=maxn+1-t2.a[i];
87         }
88         else t1.a[i]-=t2.a[i];
89     }
90     t1.len=big;
91     while(t1.a[t1.len-1]==0&& t1.len>1){
92         t1.len--;big--;
93     }
94     if(flag) t1.a[big-1]=0-t1.a[big-1];
95     return t1;
96 }
97 bool Bignum::operator>(const Bignum&T)const{
98     int ln;
99     if(len>T.len) return 1;
100    else if(len==T.len){
101        ln=len-1;
102        while(a[ln]==T.a[ln]&&ln>=0) ln--;
103        return ln>=0&&a[ln]>T.a[ln];
104    }else return 0;
105 }
106 bool Bignum::operator >(const int &t)const{
107     Bignum b(t);
108     return *this>b;
109 }
110 struct node{
111     string name;
112     Bignum val;
113     bool operator <(const node&a){
114         return val>a.val;
115     }
116 }a[1005];
117 vector<string>ans;
118 signed main(){
119     int n;Bignum m;cin>>n>>m;
120     for(int i=1;i<=n;i++){
121         cin>>a[i].name>>a[i].val;
122     }
123     sort(a+1,a+n+1);
124     for(int i=1;i<=n;i++){
125         if(!(a[i].val>m)){ /*只重载了>运算符*/
126             ans.push_back(a[i].name);
127             m=m-a[i].val;
128         }
129     }
130     if(m>0) cout<<"0"<<endl;

```

```

131     else{
132         cout<<ans.size()<<endl;
133         for(auto v:ans) cout<<v<<endl;
134     }
135     return 0;
136 }

```

11.6 大数

```

1  class Bignum {
2  public:
3      static const int maxm = 9999; // MAXN 是一位中最大的数字,必须是99……的形式
4      static const int maxsize = 100; // MAXSIZE 是大数的位数
5      static const int dlen = 4; // DLEN 记录压几位
6      int a[maxsize]; //可以控制大数的位数,存储的最大数字为maxm^maxsize
7      int len; //大数目前长度
8      Bignum(){ len = 1;memset(a,0,sizeof(a)); } //构造函数
9      Bignum(const int); //将一个int类型的变量转化为大数
10     Bignum(const char*); //将一个字符串类型的变量转化为大数
11     Bignum(const Bignum &); //拷贝构造函数
12     Bignum &operator=(const Bignum &); //重载赋值运算符,大数之间进行赋值运算
13
14     Bignum operator+(const Bignum &) const; //重载加法运算符,两个大数之间的相加运算
15     Bignum operator-(const Bignum &) const; //重载减法运算符,两个大数之间的相减运算
16     Bignum operator*(const Bignum &) const; //重载乘法运算符,两个大数之间的相乘运算
17     Bignum operator/(const int &) const; //重载除法运算符,大数对一个整数进行相除运算
18
19     Bignum operator^(const int &) const; //大数的n次方运算
20     int operator%(const int &) const; //大数对一个int类型的变量进行取模运算
21     bool operator>(const Bignum & T) const; //大数和另一个大数的大小比较
22     bool operator>(const int & t) const; //大数和一个int类型的变量的大小比较
23
24     void print(); //输出大数
25
26     friend istream&operator >>(istream&in, Bignum &b) {
27         char ch[maxsize*4];
28         int i = -1;
29         in>>ch;
30         int l=strlen(ch);
31         int count=0,sum=0;
32         for(i=l-1;i>=0;) {
33             sum = 0;
34             int t=1;
35             for(int j=0;j<4&&ch[i-j]>='0';j++,i--,t*=10) {
36                 sum+=(ch[i-j]-'0')*t;
37             }
38             b.a[count]=sum;
39             count++;
40         }
41         b.len =count++;
42         return in;
43     }

```

```
44     friend ostream& operator<<(ostream& out, Bignum& b) { //重载输出运算符
45         int i;
46         out << b.a[b.len - 1];
47         for(i = b.len - 2 ; i >= 0 ; i--) {
48             out.width(dlen);
49             out.fill('0');
50             out << b.a[i];
51         }
52         return out;
53     }
54
55 };
56
57 Bignum::Bignum(const int b) { //将一个int类型的变量转化为大数
58     int c,d = b;
59     len = 0; memset(a,0,sizeof(a));
60     while(d > maxm) {
61         c = d - (d / (maxm + 1)) * (maxm + 1);
62         d = d / (maxm + 1);
63         a[len++] = c;
64     }
65     a[len++] = d;
66 }
67
68 Bignum::Bignum(const char*s) { //将一个字符串类型的变量转化为大数
69     int t,k,index,l,i;
70     memset(a,0,sizeof(a));
71     l=strlen(s); len=l/dlen;
72     if(l%dlen) len++;
73     index=0;
74     for(i=l-1;i>=0;i-=dlen) {
75         t=0; k=i-dlen+1;
76         if(k<0) k=0;
77         for(int j=k;j<=i;j++) t=t*10+s[j]-'0';
78         a[index++]=t;
79     }
80 }
81
82 Bignum::Bignum(const Bignum & T) : len(T.len) { //拷贝构造函数
83     int i; memset(a,0,sizeof(a));
84     for(i = 0 ; i < len ; i++) a[i] = T.a[i];
85 }
86
87 Bignum & Bignum::operator=(const Bignum & n) { //重载赋值运算符，大数之间进行赋值运算
88     len = n.len;int i;
89     memset(a,0,sizeof(a));
90     for(i = 0 ; i < len ; i++) a[i] = n.a[i];
91     return *this;
92 }
93
94
95 Bignum Bignum::operator+(const Bignum & T) const { //两个大数之间的相加运算
96     Bignum t(*this);
97     int i,big; //位数
```

```

98     big = T.len > len ? T.len : len;
99     for(i = 0 ; i < big ; i++) {
100         t.a[i] +=T.a[i];
101         if(t.a[i] > maxm) {
102             t.a[i + 1]++;
103             t.a[i] -=maxm+1;
104         }
105     }
106     if(t.a[big] != 0) t.len = big + 1;
107     else t.len = big;
108     return t;
109 }
110 Bignum Bignum::operator-(const Bignum & T) const { //两个大数之间的相减运算
111     int i,j,big; bool flag;
112     Bignum t1,t2;
113     if(*this>T) {
114         t1=*this; t2=T;
115         flag=0;
116     } else {
117         t1=T; t2=*this;
118         flag=1;
119     }
120     big=t1.len;
121     for(i = 0 ; i < big ; i++) {
122         if(t1.a[i] < t2.a[i]) {
123             j = i + 1;
124             while(t1.a[j] == 0) j++;
125             t1.a[j--]--;
126             while(j > i) t1.a[j--] += maxm;
127             t1.a[i] += maxm + 1 - t2.a[i];
128         }
129         else t1.a[i] -= t2.a[i];
130     }
131     t1.len = big;
132     while(t1.a[t1.len - 1] == 0 && t1.len > 1) {
133         t1.len--; big--;
134     }
135     if(flag) t1.a[big-1]=0-t1.a[big-1];
136     return t1;
137 }
138
139 Bignum Bignum::operator*(const Bignum & T) const { //两个大数之间的相乘运算
140     Bignum ret;
141     int i,j,up;
142     int temp,temp1;
143     for(i = 0 ; i < len ; i++) {
144         up = 0;
145         for(j = 0 ; j < T.len ; j++) {
146             temp = a[i] * T.a[j] + ret.a[i + j] + up;
147             if(temp > maxm) {
148                 temp1 = temp - temp / (maxm + 1) * (maxm + 1);
149                 up = temp / (maxm + 1);
150                 ret.a[i + j] = temp1;
151             } else {

```

```

152         up = 0;
153         ret.a[i + j] = temp;
154     }
155 }
156 if(up != 0) ret.a[i + j] = up;
157 }
158 ret.len = i + j;
159 while(ret.a[ret.len - 1] == 0 && ret.len > 1) ret.len--;
160 return ret;
161 }
162 Bignum Bignum::operator/(const int & b) const { //大数对一个整数进行相除运算
163     Bignum ret;
164     int i, down = 0;
165     for(i = len - 1; i >= 0; i--) {
166         ret.a[i] = (a[i] + down * (maxm + 1)) / b;
167         down = a[i] + down * (maxm + 1) - ret.a[i] * b;
168     }
169     ret.len = len;
170     while(ret.a[ret.len - 1] == 0 && ret.len > 1) ret.len--;
171     return ret;
172 }
173 int Bignum::operator%(const int & b) const { //大数对一个int类型的变量进行取模运算
174     int i, d = 0;
175     for(i = len - 1; i >= 0; i--) d = ((d * (maxm + 1)) % b + a[i]) % b;
176     return d;
177 }
178 Bignum Bignum::operator^(const int & n) const { //大数的n次方运算
179     Bignum t, ret(1);
180     int i;
181     if(n < 0) exit(-1);
182     if(n == 0) return 1;
183     if(n == 1) return *this;
184     int m = n;
185     while(m > 1) {
186         t = *this;
187         for(i = 1; i <= m; i++) t = t * t;
188         m = i; ret = ret * t;
189         if(m == 1) ret = ret * (*this);
190     }
191     return ret;
192 }
193
194 bool Bignum::operator>(const Bignum & T) const { //大数和另一个大数的大小比较
195     int ln;
196     if(len > T.len) return 1;
197     else if(len == T.len) {
198         ln = len - 1;
199         while(a[ln] == T.a[ln] && ln >= 0) ln--;
200         return ln >= 0 && a[ln] > T.a[ln];
201     } else return 0;
202 }
203
204 bool Bignum::operator>(const int & t) const { //大数和一个int类型的变量的大小比较
205     Bignum b(t);

```

```

206     return *this>b;
207 }
208
209 void Bignum::print() {    //输出大数
210     int i;
211     cout << a[len - 1];
212     for(i = len - 2 ; i >= 0 ; i--) {
213         cout.width(dlen);
214         cout.fill('0');
215         cout << a[i];
216     }
217     cout << endl;
218 }

```

11.7 java 大数

1. valueOf(parameter); 将参数转换为制定的类型。如: String s=" 12345" ;BigInteger c=BigInteger.valueOf(s); 则 c=12345;
2. add(); 大整数相加;
3. subtract(); 相减;
4. multiply(); 相乘;
5. divide(); 相除取整;
6. remainder(); 取余;
7. pow(); $a.pow(b)=a^b$;
8. gcd(); 最大公约数;
9. abs(); 绝对值;
10. negate(); 取反数;
11. mod(); $a.mod(b)=a\%b=a remainder(b)$;
12. max(); min();
13. public int compareTo();
14. boolean equals(); 是否相等;
15. BigInteger(String val); 将指定字符串转换为十进制表示形式;
16. BigInteger(String val,int radix); 将指定基数的 BigInteger 的字符串表示形式转换为 BigInteger。

A=BigInteger.ONE 1

B=BigInteger.TEN 10

C=BigInteger.ZERO 0

第十二章 技巧与黑科技

12.1 离散化

```
1 //数组离散化 含重复元素
2 std::sort(sub_a, sub_a+n);
3 int size = std::unique(sub_a, sub_a+n) - sub_a; //size为离散化后元素个数
4 for (i = 0; i < n; i++) {
5     a[i] = std::lower_bound(sub_a, sub_a+size, a[i]) - sub_a + 1; //k为b[i]经离散化后对应的值
6 }
7
8 //坐标离散化
9 int compress(int *x1, int *x2, int w){
10     std::vector<int> xs;
11     for (int i = 0; i < N; i++) {
12         for (int d = -1; d <= 1; d++) {
13             int tx1 = x1[i] + d, tx2 = x2[i] + d;
14             if (1 <= tx1 && tx1 <= w) xs.push_back(tx1);
15             if (1 <= tx2 && tx2 <= w) xs.push_back(tx2);
16         }
17     }
18     std::sort(xs.begin(), xs.end());
19     xs.erase(unique(xs.begin(), xs.end()), xs.end());
20     for (int i = 0; i < N; i++) {
21         x1[i] = find(xs.begin(), xs.end(), x1[i]) - xs.begin();
22         x2[i] = find(xs.begin(), xs.end(), x2[i]) - xs.begin();
23     }
24     return xs.size();
25 }
```

12.2 快速枚举子集

```
1 void print_subset(int n, int s) {
2     for (int i = 0; i < n; i++) {
3         if (s & (1 << i)) {
4             std::cout << i << " ";
5         }
6         std::cout << '\n';
7     }
8 }
9 int main(int argc, char *argv[]) {
10     int n;
11     std::cin >> n;
```

```

12     for (int i = 0; i < (1 << n); i++) print_subset(n, i);
13 }
14
15 //当x代表集合 x的子集: for (int i = x; i; i=(i-1)&x) {}

```

12.3 bitset

1	#include <bitset>	
2	b.any()	b中是否存在置为1的二进制位?
3	b.none()	b中不存在置位1的二进制位吗?
4	b.count()	b中置为1的二进制位的个数
5	b.size()	b中二进制位的个数
6	b[pos]	访问b中在pos处的二进制位
7	b.test(pos)	b中在pos处的二进制位是否为1?
8	b.set()	把b中所有二进制位都置为1
9	b.set(pos)	把b中在pos处的二进制位置为1
10	b.reset()	把b中所有二进制位置为0
11	b.reset(pos)	把b中在pos处的二进制位置为0
12	b.flip()	把b中所有二进制位逐位取反
13	b.flip(pos)	把b中在pos处的二进制位取反
14	b.to_ulong()	用b中同样的二进制位返回一个unsigned long值
15	os << b	把b中的位集中输出到os流

12.4 位运算

```

1 //去掉最后一位
2 x >> 1
3 //在最后加一个0
4 x << 1
5 //在最后加一个1
6 x << 1 + 1
7 //把最后一位变成1
8 x | 1
9 //把最后一位变成0
10 x | 1 - 1
11 //最后一位取反
12 x ^ 1
13 //把右数第k位变成1
14 x | (1 << (k-1))
15 //把右数第k位变成0
16 x & ~ (1 << (k-1))
17 //右数第k位取反
18 x ^ (1 << (k-1))
19 //取末三位
20 x & 7
21 //取末k位
22 x & (1 << k-1)
23 //取右数第k位
24 x >>(k-1) & 1

```

```

25 //把末k位变成1
26 x |(1 << k-1)
27 //末k位取反
28 x ^ (1 << k-1)
29 //把右边连续的1变成0
30 x & (x+1)
31 //x个1
32 ((1<<x)-1)
33 //二进制里1的数量
34 (x>>16)+(x&((1<<16)-1))

```

12.5 跳舞链

```

1 struct DLX{
2     const static int maxn=20010;
3     #define FF(i,A,s) for(int i = A[s];i != s;i = A[i])
4     int L[maxn],R[maxn],U[maxn],D[maxn];
5     int size,col[maxn],row[maxn],s[maxn],H[maxn];
6     bool vis[70];
7     int ans[maxn],cnt;
8     void init(int m){
9         for(int i=0;i<=m;i++){
10             L[i]=i-1;R[i]=i+1;U[i]=D[i]=i;s[i]=0;
11         }
12         memset(H,-1,sizeof(H));
13         L[0]=m;R[m]=0;size=m+1;
14     }
15     void link(int r,int c){
16         U[size]=c;D[size]=D[c];U[D[c]]=size;D[c]=size;
17         if(H[r]<0)H[r]=L[size]=R[size]=size;
18         else {
19             L[size]=H[r];R[size]=R[H[r]];
20             L[R[H[r]]]=size;R[H[r]]=size;
21         }
22         s[c]++;col[size]=c;row[size]=r;size++;
23     }
24     void del(int c){//精确覆盖
25         L[R[c]]=L[c];R[L[c]]=R[c];
26         FF(i,D,c)FF(j,R,i)U[D[j]]=U[j],D[U[j]]=D[j],--s[col[j]];
27     }
28     void add(int c){ //精确覆盖
29         R[L[c]]=L[R[c]]=c;
30         FF(i,U,c)FF(j,L,i)++s[col[U[D[j]]=D[U[j]]=j]];
31     }
32     bool dfs(int k){//精确覆盖
33         if(!R[0]){
34             cnt=k;return 1;
35         }
36         int c=R[0];FF(i,R,0)if(s[c]>s[i])c=i;
37         del(c);
38         FF(i,D,c){
39             FF(j,R,i)del(col[j]);

```

```

40         ans[k]=row[i];if(dfs(k+1))return true;
41         FF(j,L,i)add(col[j]);
42     }
43     add(c);
44     return 0;
45 }
46 void remove(int c){//重复覆盖
47     FF(i,D,c)L[R[i]]=L[i],R[L[i]]=R[i];
48 }
49 void resume(int c){//重复覆盖
50     FF(i,U,c)L[R[i]]=R[L[i]]=i;
51 }
52 int A(){//估价函数
53     int res=0;
54     memset(vis,0,sizeof(vis));
55     FF(i,R,0)if(!vis[i]){
56         res++;vis[i]=1;
57         FF(j,D,i)FF(k,R,j)vis[col[k]]=1;
58     }
59     return res;
60 }
61 void dfs(int now,int &ans){//重复覆盖
62     if(R[0]==0)ans=min(ans,now);
63     else if(now+A()<ans){
64         int temp=INF,c;
65         FF(i,R,0)if(temp>s[i])temp=s[i],c=i;
66         FF(i,D,c){
67             remove(i);FF(j,R,i)remove(j);
68             dfs(now+1,ans);
69             FF(j,L,i)resume(j);resume(i);
70         }
71     }
72 }
73 }dlx;

```

12.6 K-D 树

```

1  #include <queue>
2  #include <cstdio>
3  #include <cstring>
4  #include <algorithm>
5  using namespace std;
6  const int N = 55555, K = 5;
7  const int inf = 0x3f3f3f3f;
8
9  #define sqr(x) (x)*(x)
10 int k,n,idx;    //k为维数,n为点数
11 struct point {
12     int x[K];
13     bool operator < (const point &u) const {
14         return x[idx]<u.x[idx];
15     }

```

```

16 }po[N];
17
18 typedef pair<double,point>tp;
19 priority_queue<tp>nq;
20
21 struct kdTree {
22     point pt[N<<2];
23     int son[N<<2];
24
25     void build(int l,int r,int rt=1,int dep=0) {
26         if(l>r) return;
27         son[rt]=r-1;
28         son[rt*2]=son[rt*2+1]=-1;
29         idx=dep%k;
30         int mid=(l+r)/2;
31         nth_element(po+l,po+mid,po+r+1);
32         pt[rt]=po[mid];
33         build(l,mid-1,rt*2,dep+1);
34         build(mid+1,r,rt*2+1,dep+1);
35     }
36     void query(point p,int m,int rt=1,int dep=0) {
37         if(son[rt]==-1) return;
38         tp nd(0,pt[rt]);
39         for(int i=0;i<k;i++) nd.first+=sqr(nd.second.x[i]-p.x[i]);
40         int dim=dep%k,x=rt*2,y=rt*2+1,fg=0;
41         if(p.x[dim]>=pt[rt].x[dim]) swap(x,y);
42         if(~son[x]) query(p,m,x,dep+1);
43         if(nq.size()<m) nq.push(nd),fg=1;
44         else {
45             if(nd.first<nq.top().first) nq.pop(),nq.push(nd);
46             if(sqr(p.x[dim]-pt[rt].x[dim])<nq.top().first) fg=1;
47         }
48         if(~son[y]&&fg) query(p,m,y,dep+1);
49     }
50 }kd;
51
52 void print(point &p) {
53     for(int j=0;j<k;j++) printf("%d%c",p.x[j],j==k-1?'\\n':' ');
54 }
55
56 int main() {
57     while(scanf("%d%d",&n,&k)!=EOF) {
58         for(int i=0;i<n;i++) for(int j=0;j<k;j++) scanf("%d",&po[i].x[j]);
59         kd.build(0,n-1);
60         int t,m;
61         for(scanf("%d",&t);t--;) {
62             point ask;
63             for(int j=0;j<k;j++) scanf("%d",&ask.x[j]);
64             scanf("%d",&m); kd.query(ask,m);
65             printf("the %d closest %d points are:\\n", m);
66             point pt[20];
67             for(int j=0;!nq.empty();j++) pt[j]=nq.top().second,nq.pop();
68             for(int j=m-1;j>=0;j--) print(pt[j]);
69         }

```

```
70     }
71     return 0;
72 }
```

12.7 随机

```
1  // #include <iostream>
2  // #include <random>
3
4  std::vector<int> permutation(100);
5  for (int i = 0; i < 100; i++) {
6      permutation[i] = i+1;
7  }
8  std::mt19937_64 mt1(1); // 64位
9  std::mt19937 mt2(2); // 32位
10 shuffle(permutation.begin(), permutation.end(), mt2); // 打乱序列
11 for (auto it: permutation) {
12     std::cout << it << " ";
13 }
```

12.8 珂朵莉树 (Old Driver Tree)

```
1  #include <set>
2  #include <algorithm>
3
4  using ll = long long;
5
6  struct node {
7      int l, r;
8      mutable ll v;
9      node(int L, int R = -1, ll V = 0) : l(L), r(R), v(V) {}
10     bool operator < (const node& o) const {
11         return l < o.l;
12     }
13 };
14
15 std::set<node> s;
16
17 // 分割SET 返回一个pos位置的迭代器
18 std::set<node>::iterator split(int pos) {
19     auto it = s.lower_bound(node(pos));
20     if (it != s.end() && it->l == pos) return it;
21     --it;
22     if (pos > it->r) return s.end();
23     int L = it->l, R = it->r;
24     ll V = it->v;
25     s.erase(it);
26     s.insert(node(L, pos - 1, V));
27     return s.insert(node(pos, R, V)).first;
```

```

28 }
29
30 //区间加值
31 void add(int l, int r, ll val=1) {
32     split(l);
33     auto itr = split(r+1), itl = split(l);
34     for (; itl != itr; ++itl) itl->v += val;
35 }
36
37 //区间赋值
38 void assign(int l, int r, ll val = 0) {
39     split(l);
40     auto itr = split(r+1), itl = split(l);
41     s.erase(itl, itr);
42     s.insert(node(l, r, val));
43 }

```

12.9 CDQ 分治

```

1 //Author:marsed
2 /*
3 *将区间分成左右两部分 递归处理
4 一层递归计算当前左区间的修改操作对右区间的查询操作的影响
5 当flag为1代表修改操作 为0代表查询操作
6 */
7 #include <algorithm>
8 #define mid (l + r)/2
9
10 const int maxn = "Edit";
11
12 struct Node {
13     int id, x1,x2;
14     int operator<(const Node &b) { //按照参数的优先级排序
15         return ;
16     }
17 };
18
19 Node nod[maxn], tmp[maxn];
20
21 void cdq(int l, int r) {
22     if (l == r) return;
23     cdq(l, mid); cdq(mid + 1, r);
24     int p = l, q = mid + 1, cnt = 0;
25     while (p <= mid&&q <= r) {
26         if (nod[p] < nod[q]) {
27             if (nod[p].flag); //左区间里的修改操作会对右区间的查询操作有影响 计算影响
28             tmp[cnt++] = nod[p++];
29         } else {
30             if (!nod[q].flag); //计算右区间的查询操作的值
31             tmp[cnt++] = nod[q++];
32         }
33     }

```

```

34     while (p <= mid) tmp[cnt++] = nod[p++];
35     while (q <= r) {
36         if (!nod[q].flag) ;
37         tmp[cnt++] = nod[q++];
38     }
39     for (int i = 1; i <= r; i++)
40         nod[i] = tmp[i - 1];
41 }
42
43 int main()
44 {
45     cdq(1, q);
46     return 0;
47 }

```

12.10 内置位运算函数

```

1  — Built-in Function: int __builtin_ffs (unsigned int x)
2  Returns one plus the index of the least significant 1-bit of x, or if x is zero, returns zero.
3  返回右起第一个 ‘1’ 的位置。
4
5  — Built-in Function: int __builtin_clz (unsigned int x)
6  Returns the number of leading 0-bits in x, starting at the most significant bit position. If x
   is 0, the result is undefined.
7  返回左起第一个 ‘1’ 之前0的个数。
8
9  — Built-in Function: int __builtin_ctz (unsigned int x)
10 Returns the number of trailing 0-bits in x, starting at the least significant bit position. If
   x is 0, the result is undefined.
11 返回右起第一个 ‘1’ 之后的0的个数。
12
13 — Built-in Function: int __builtin_popcount (unsigned int x)
14 Returns the number of 1-bits in x.
15 返回 ‘1’ 的个数。
16
17 — Built-in Function: int __builtin_parity (unsigned int x)
18 Returns the parity of x, i.e. the number of 1-bits in x modulo 2.
19 返回 ‘1’ 的个数的奇偶性。
20
21 — Built-in Function: int __builtin_ffsl (unsigned long)
22 Similar to __builtin_ffs, except the argument type is unsigned long.
23
24 — Built-in Function: int __builtin_clzl (unsigned long)
25 Similar to __builtin_clz, except the argument type is unsigned long.
26
27 — Built-in Function: int __builtin_ctzl (unsigned long)
28 Similar to __builtin_ctz, except the argument type is unsigned long.
29
30 — Built-in Function: int __builtin_popcountl (unsigned long)
31 Similar to __builtin_popcount, except the argument type is unsigned long.
32
33 — Built-in Function: int __builtin_parityl (unsigned long)

```



```

34 Similar to __builtin_parity, except the argument type is unsigned long.
35
36 — Built-in Function: int __builtin_ffsll (unsigned long long)
37 Similar to __builtin_ffs, except the argument type is unsigned long long.
38
39 — Built-in Function: int __builtin_clzll (unsigned long long)
40 Similar to __builtin_clz, except the argument type is unsigned long long.
41
42 — Built-in Function: int __builtin_ctzll (unsigned long long)
43 Similar to __builtin_ctz, except the argument type is unsigned long long.
44
45 — Built-in Function: int __builtin_popcountll (unsigned long long)
46 Similar to __builtin_popcount, except the argument type is unsigned long long.
47
48 — Built-in Function: int __builtin_parityll (unsigned long long)
49 Similar to __builtin_parity, except the argument type is unsigned long long.
50
51
52
53 随机数函数:
54 default_random_engine: 随机非负数（不建议单独使用）。
55 uniform_int_distribution: 指定范围的随机非负数。
56 uniform_real_distribution: 指定范围的随机实数。
57 bernoulli_distribution: 指定概率的随机布尔值。
58
59 示例: default_random_engine e;
60     uniform_int_distribution<int>u(0,9);
61     cout<<u(e)<<endl;

```

12.11 0-1 分数规划

```

1  template <size_t N, typename T, typename Z = double>
2  struct zero_one_plan {
3      Z f[N];
4      Z solve(T *c, T *s, int n, int k) { // max-> sigma(c[i])/sigma(s[i])
5          Z l=0, r=*max_element(c, c+n);
6          while(fabs(r-l)>eps){
7              Z mid=(l+r)/2.;
8              rep(i,0,n)f[i]=1.*c[i]-mid*s[i];
9              nth_element(f, f+k, f+n, greater<Z>());
10             Z sm=0;
11             rep(i,0,k)sm+=f[i];
12             if(sm>-eps)l=mid;
13             else r=mid;
14         }
15         return l;
16     }
17 };

```

12.12 BM 线性递推

```

1  //author: xudyh
2
3  namespace linear_seq {
4      const int N = 10010;
5      typedef long long ll;
6      constexpr ll mod = (ll) 1e9 + 7;
7
8      ll pow_mod(ll a, ll b) {
9          ll r = 1;
10         for (a %= mod; b; b >>= 1, a = a * a % mod) {
11             if (b & 1) r = r * a % mod;
12         }
13         return r;
14     }
15
16     ll res[N], base[N], _c[N], _md[N];
17     vector<int> Md;
18
19     void mul(ll *a, ll *b, int k) {
20         k <= 1;
21         for (int i = 0; i < k; ++i) _c[i] = 0;
22         k >>= 1;
23         for (int i = 0; i < k; ++i) {
24             if (a[i]) {
25                 for (int j = 0; j < k; ++j) {
26                     _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
27                 }
28             }
29         }
30         for (int i = k + k - 1; i >= k; i--) {
31             if (_c[i]) {
32                 for (const int md: Md) {
33                     _c[i - k + md] = (_c[i - k + md] - _c[i] * _md[md]) % mod;
34                 }
35             }
36         }
37         for (int i = 0; i < k; ++i) {
38             a[i] = _c[i];
39         }
40     }
41
42     int solve(ll n, vector<int> a, vector<int> b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
43     //     printf("SIZE %d\n", SZ(b));
44         ll ans = 0, pnt = 0;
45         int k = (int) a.size();
46         assert(a.size() == b.size());
47         for (int i = 0; i < k; ++i) {
48             _md[k - 1 - i] = -a[i];
49         }
50         _md[k] = 1;
51         Md.clear();
52         for (int i = 0; i < k; ++i) {

```

```

53     if (_md[i] != 0) {
54         Md.push_back(i);
55     }
56 }
57 for (int i = 0; i < k; ++i) {
58     res[i] = base[i] = 0;
59 }
60 res[0] = 1;
61 while ((1ll << pnt) <= n) {
62     pnt++;
63 }
64 for (int p = pnt; p >= 0; p--) {
65     mul(res, res, k);
66     if ((n >> p) & 1) {
67         for (int i = k - 1; i >= 0; i--) {
68             res[i + 1] = res[i];
69         }
70         res[0] = 0;
71         for (const int md: Md) {
72             res[md] = (res[md] - res[k] * _md[md]) % mod;
73         }
74     }
75 }
76 for (int i = 0; i < k; ++i) {
77     ans = (ans + res[i] * b[i]) % mod;
78 }
79 if (ans < 0) ans += mod;
80 return ans;
81 }
82
83 vector<int> BM(vector<int> s) {
84     vector<int> C(1, 1), B(1, 1);
85     int L = 0, m = 1, b = 1;
86     for (int n = 0; n < (int) s.size(); ++n) {
87         ll d = 0;
88         for (int i = 0; i <= L; ++i) {
89             d = (d + (ll) C[i] * s[n - i]) % mod;
90         }
91         if (d == 0) {
92             ++m;
93         }
94         else if (2 * L <= n) {
95             vector<int> T = C;
96             ll c = mod - d * pow_mod(b, mod - 2) % mod;
97             while (C.size() < B.size() + m) {
98                 C.push_back(0);
99             }
100             for (int i = 0; i < (int) B.size(); ++i) {
101                 C[i + m] = (C[i + m] + c * B[i]) % mod;
102             }
103             L = n + 1 - L;
104             B = T;
105             b = d;
106             m = 1;

```

```
107         } else {
108             ll c = mod - d * pow_mod(b, mod - 2) % mod;
109             while (C.size() < B.size() + m) {
110                 C.push_back(0);
111             }
112             for (int i = 0; i < (int) B.size(); ++i) {
113                 C[i + m] = (C[i + m] + c * B[i]) % mod;
114             }
115             ++m;
116         }
117     }
118     return C;
119 }
120
121 int gao(vector<int> a, ll n) {
122     vector<int> c = BM(a);
123     c.erase(c.begin());
124     for (int &x:c) {
125         x = (mod - x) % mod;
126     }
127     return solve(n, c, vector<int>(a.begin(), a.begin() + c.size()));
128 }
129 }
```