

华南师范大学
ACM-ICPC 集训队
DeDeRong



2019 年 11 月 3 日

目录

1 图论	4		
1.1 最短路径	4		
1.1.1 Dijkstra	4		
1.1.2 Dijkstra 优化	4		
1.1.3 Floyd	4		
1.1.4 Bellman-Ford	5		
1.1.5 SPFA	5		
1.2 次短路	6		
1.3 LCA	7		
1.3.1 倍增	7		
1.3.2 RMQ	8		
1.4 强连通分量	9		
1.5 割点	9		
1.6 桥	10		
1.7 最大流	11		
1.7.1 Dinic	11		
1.7.2 Dinic 优化	12		
1.8 二分图匹配	13		
1.8.1 匈牙利算法	13		
1.9 最小生成树	14		
1.9.1 Prim	14		
1.9.2 kruskal	14		
1.10 次小生成树-POJ1679	15		
1.11 拓扑排序	16		
1.12 Floyd 找最小环	17		
1.13 生成树计数 (基尔霍夫矩阵) BZOJ1002	18		
1.14 悬线法 BZOJ3039	19		
2 数据结构	20		
2.1 并查集	20		
2.2 带权并查集	20		
2.2.1 BZOJ1202	20		
2.3 ST 表	21		
2.4 树状数组	21		
2.5 线段树	22		
2.5.1 单点修改 + 区间求和 HDU1166	22		
2.5.2 区间修改 + 区间求和 POJ3465	23		
2.5.3 单点修改 + 区间最值 HDU1754	24		
2.5.4 区间染色 + 统计 + 离散化 POJ2528	25		
2.5.5 线段树 + 扫描线求矩阵覆盖周长 POJ1177	26		
2.5.6 线段树 + 扫描线求矩阵面积并 HDU1542	28		
2.5.7 状压 + 线段树维护区间 26 字母种类 CF1234D	29		
2.6 主席树	30		
2.6.1 区间第 K 大	30		
2.6.2 动态区间第 K 大 (主席树套树状数组) ZOJ2112	31		
3 DP	33		
3.1 背包	33		
3.2 数位 DP	34		
3.2.1 HDU 2089 不要 62	34		
3.2.2 SCOI 2009 Windy 数	35		
3.3 区间 DP	36		
3.3.1 51Nod 1021 石子合并	36		
3.4 最长公共递增子序列	36		
3.5 最长公共子序列	37		
3.6 最长有序子序列	37		
4 字符串	38		
4.1 KMP	38		
4.2 EXKMP	39		
4.3 字符串 Hash	40		
4.3.1 自然溢出	40		
5 数学	40		
5.1 欧拉函数	40		
5.2 枚举约数	41		
5.3 全错排	42		
5.4 唯一分解定理	42		
5.4.1 例题一	42		
5.4.2 例题二	43		
5.5 快速幂	44		
5.6 欧拉降幂和广义欧拉降幂	45		
5.7 逆序数	45		
5.8 无序序列变有序的最少交换次数	46		
5.8.1 相邻元素交换	46		
5.8.2 任意元素交换	46		
5.9 GCD	46		
5.9.1 非递归	46		

5.9.2	递归	46	8.5	C++ 大数	57
5.10	EXGCD	46	8.5.1	大数加法	57
5.10.1	例题	47	8.5.2	大数乘法	57
5.11	逆元	48	8.5.3	整数转 string	58
5.11.1	拓展欧几里得法	48	8.6	Java	58
5.11.2	费马小定理法	48			
5.12	中国剩余定理	48			
5.12.1	例题	49			
5.13	拓展中国剩余定理	49			
5.14	素数	50			
5.14.1	判断小于 MAXN 的数是否为素数	50			
5.14.2	埃式筛法	50			
5.14.3	线性筛法	50			
5.14.4	Miller Rabin	51			
5.15	组合数	51			
5.16	阶乘长度	51			
5.17	全排列	52			
5.18	求斐波那契第 N 项	52			
6	STL	53			
6.1	vector	53			
6.2	set	53			
6.3	pair	54			
6.4	stack	54			
6.5	queue	54			
6.6	map	55			
6.7	bitset	55			
6.8	algorithm	55			
7	计算几何	55			
7.1	三角形面积	55			
7.2	两圆面积交	55			
8	其它	56			
8.1	数据类型范围	56			
8.2	头文件	56			
8.3	Vim 配置	56			
8.4	输入挂	56			
8.4.1	关闭同步	56			
8.4.2	IO	57			

1 图论

1.1 最短路径

1.1.1 Dijkstra

```

1 const int maxn = 1e4;
2 const int inf = 0x3f3f3f3f;
3
4 //d数组用来记录源点s到顶点i的最短距离
5 //v表示该顶点是否在顶点集S中
6 //g邻接矩阵存图, g[i][j]表示i到j的边的权值, 无边时为inf
7 //n为顶点数量
8 int d[maxn], v[maxn];
9 int g[maxn][maxn];
10 int n;
11 void dij(int s)
12 {
13     memset(v, 0, sizeof(v));
14     for(int i=1; i<=n; i++)
15         d[i] = g[s][i];
16     v[s] = 1;
17     for(int i=1; i<=n; i++)
18     {
19         int u = 0;
20         for(int j=1; j<=n; j++)
21         {
22             if(!v[j] && (u==0 || d[j] < d[u]))
23                 u = j;
24         }
25         if(u==0) return ;
26         v[u] = 1;
27         for(int j=1; j<=n; j++)
28         {
29             d[j] = min(d[j], d[u]+g[u][j]);
30         }
31     }
32 }

```

1.1.2 Dijkstra 优化

```

1 const int maxn = 1e4;

```

```

2 const int inf = 0x3f3f3f3f;
3 typedef pair<int, int> P; //first表示最短距离, second表示顶点编号
4 //边: to表示这条边指向的顶点, 权值为w
5 struct Edge
6 {
7     int to, w;
8 };
9 //用vector实现邻接表
10 vector<Edge> g[maxn];
11 int d[maxn]; //记录源点到顶点i的最短距离
12 int n;
13
14 void dij(int s)
15 {
16     priority_queue<P, vector<P>, greater<P> > q;
17     memset(d, inf, sizeof(d));
18     d[s] = 0;
19     q.push(P(0, s));
20     while(!q.empty())
21     {
22         P p = q.top();
23         q.pop();
24         int u = p.second;
25         if(d[u] < p.first) continue;
26         for(int i=0; i<g[u].size(); i++)
27         {
28             Edge e = g[u][i];
29             if(d[e.to] > d[u] + e.w)
30             {
31                 d[e.to] = d[u] + e.w;
32                 q.push(P(d[e.to], e.to));
33             }
34         }
35     }
36 }

```

1.1.3 Floyd

```

1 int g[maxn][maxn];
2 int n;
3 void floyd()
4 {

```

```

5   for(int k=1;k<=n;k++)
6       for(int i=1;i<=n;i++)
7           for(int j=1;j<=n;j++)
8               g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
9   }

```

1.1.4 Bellman-Ford

```

1   const int maxn = 1e4;
2   const int inf = 0x3f3f3f3f; //常用于表示无穷大
3
4   //边结构体, 记录u→v的边, 权值为w
5   struct Edge
6   {
7       int u, v, w;
8       Edge(int uu, int vv, int ww) { u=uu; v=vv; w=ww; }
9       Edge(){}
10  }e[maxn];
11  int edgecnt; // 边的数量
12  //加边操作
13  void addEdge(int u, int v, int w)
14  {
15      e[edgecnt++] = Edge(u, v, w);
16  }
17
18  int n; //顶点总数
19  int d[maxn]; //记录最短距离的数组
20
21  //存在负权回路则返回true, 否则返回false
22  bool bellman_ford(int s)
23  {
24      memset(d, inf, sizeof(d));
25      d[s] = 0;
26      //进行n-1次松弛操作, 第n次检查是否含有负权回路
27      for(int i=1;i<=n;i++)
28      {
29          int flag = 0;
30          for(int j=0; j<edgecnt; j++)
31          {
32              Edge t = e[j];
33              int u, v, w;
34              u = t.u; v = t.v; w = t.w;

```

```

35          if(d[v] > d[u] + w)
36          {
37              d[v] = d[u] + w;
38              flag = 1;
39          }
40      }
41      if(!flag) return false;
42      if(i==n && flag) return true;
43  }
44  return false;
45  }

```

1.1.5 SPFA

```

1   const int maxn = 1e4;
2   const int inf = 0x3f3f3f3f; //常用于表示无穷大
3
4   //边结构体, to表示边指向的顶点编号, 权值为w
5   struct Edge
6   {
7       int to, w;
8       Edge(int tt, int ww) { to = tt; w = ww; }
9       Edge(){}
10  };
11  //vector实现的邻接表
12  vector<Edge> g[maxn];
13  int n; //顶点数
14  //d表示最短距离, inq[i]表示结点是否在队列中, 为1则在, cnt[i]记录i入队的次数
15  int d[maxn], inq[maxn], cnt[maxn];
16  //初始化
17  void init()
18  {
19      memset(d, inf, sizeof(d));
20      memset(inq, 0, sizeof(inq));
21      memset(cnt, 0, sizeof(cnt));
22  }
23  //返回true表示存在负权回路
24  bool spfa(int s)
25  {
26      init();
27      d[s] = 0;
28      inq[s] = 1;

```

```

29 cnt[s] = 1;
30 queue<int> q;
31 q.push(s);
32 while(!q.empty())
33 {
34     int u = q.front();
35     inq[u] = 0;
36     q.pop();
37     for(int i=0; i < g[u].size(); i++)
38     {
39         Edge e = g[u][i];
40         if(d[e.to] > d[u] + e.w)
41         {
42             d[e.to] = d[u] + e.w;
43             if(inq[e.to] == 0)
44             {
45                 inq[e.to] = 1;
46                 q.push(e.to);
47                 cnt[e.to]++;
48                 if(cnt[e.to] > n) return true;
49             }
50         }
51     }
52 }
53 return true;
54 }

```

1.2 次短路

```

1 #include <bits/stdc++.h>
2 #define INF 1e16+100
3 #define ms(x,y) memset(x,y,sizeof(x))
4 using namespace std;
5
6 typedef long long ll;
7 typedef pair<ll,ll> P;
8
9 const double pi = acos(-1.0);
10 const int mod = 1e9 + 7;
11 const int maxn = 1e5 + 5;
12
13 struct Edge{

```

```

14     ll to, cost;
15 };
16
17 ll n,m;
18 vector<Edge> a[maxn];
19 ll dist[maxn], dist2[maxn];
20
21 void addedge(ll u, ll v, ll w)
22 {
23     a[u].push_back(Edge{v,w});
24     a[v].push_back(Edge{u,w});
25 }
26
27 void solve()
28 {
29     priority_queue<P, vector<P>, greater<P>> que;
30     //ms(dist, INF);
31     //ms(dist2, INF);
32     fill(dist, dist+n, INF);
33     fill(dist2, dist2+n, INF);
34     dist[0]=0;
35     que.push(P(0,0));
36     while(que.size())
37     {
38         P u=que.top(); que.pop();
39         int v=u.second;
40         ll d=u.first;
41         if(dist2[v]<d) continue; //不是次短距离则抛弃
42         for(int i=0; i<a[v].size(); i++)
43         {
44             Edge e=a[v][i];
45             ll d2=d+e.cost;
46             if(dist[e.to]>d2) //更新最短
47             {
48                 swap(dist[e.to], d2);
49                 que.push(P(dist[e.to], e.to));
50             }
51             if(dist2[e.to]>d2&&dist[e.to]<d2) //更新次短
52             {
53                 dist2[e.to]=d2;
54                 que.push(P(dist2[e.to], e.to));
55             }
56         }
57     }

```

```

58 printf("%lld\n",dist2[n-1]);
59 }
60
61 int main()
62 {
63     //freopen("in.txt","r",stdin);
64     //freopen("out.txt","w",stdout);
65     int t;
66     scanf("%d",&t);
67     while(t--)
68     {
69         scanf("%lld%lld",&n,&m);
70         for(int i=0;i<n;i++) a[i].clear();
71         for(int i=0;i<m;i++)
72         {
73             ll p,q,w;
74             scanf("%lld%lld%lld",&p,&q,&w);
75             addedge(p-1,q-1,w);
76         }
77         solve();
78     }
79     return 0;
80 }

```

1.3 LCA

1.3.1 倍增

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define inf 0x3f3f3f3f
5 typedef pair<int, int> P;
6 const int maxn = 5e5+5;
7 const ll mod = 1e9+7;
8
9 vector<int> son[maxn]; // 存储儿子顶点
10 // dep[i]表示顶点i的深度, n个顶点, m个询问, rt为树根, fa数组用来预处理顶点i
   // 向上跳2^j步之后的顶点
11 int dep[maxn], n, m, rt, fa[maxn][20];
12 int v[maxn]={0}; // 是否访问标记
13
14 // pre是父顶点, rt是当前顶点

```

```

15 void dfs(int pre, int rt)
16 {
17     dep[rt] = dep[pre]+1; // 当前顶点的深度为父顶点加一
18     fa[rt][0] = pre; // 当前顶点向上跳一步为父顶点
19     v[rt] = 1; // 访问
20     // dp预处理
21     for(int i=1;i<=19;i++)
22         fa[rt][i] = fa[fa[rt][i-1]][i-1];
23     // 继续dfs
24     for(int i=0;i<son[rt].size();i++)
25         if(v[son[rt][i]]==0)
26             dfs(rt, son[rt][i]);
27 }
28
29 // 求解LCA(a, b)
30 int lca(int a, int b)
31 {
32     if(dep[a] < dep[b])
33         swap(a, b);
34     for(int i=19;i>=0;i--)
35     {
36         if(dep[a]-dep[b] >= (1<<i))
37         {
38             a = fa[a][i];
39         }
40     }
41     if(a==b)return a;
42     for(int i=19;i>=0;i--)
43     {
44         if(fa[a][i] != fa[b][i])
45         {
46             a = fa[a][i];
47             b = fa[b][i];
48         }
49     }
50     return fa[a][0];
51 }
52
53 int main()
54 {
55     scanf("%d%d%d", &n, &m, &rt);
56     for(int i=1;i<n;i++)
57     {
58         int a, b;

```

```

59     scanf("%d%d", &a, &b);
60     son[a].push_back(b);
61     son[b].push_back(a);
62 }
63 memset(fa, 0, sizeof(fa));
64 memset(dep, inf, sizeof(dep));
65 v[0]=1;
66 dep[0] = 0;
67 dfs(0, rt);
68 for(int i=1;i<=m;i++)
69 {
70     int a, b;
71     scanf("%d%d", &a, &b);
72     printf("%d\n", lca(a, b));
73 }
74 return 0;
75 }

```

1.3.2 RMQ

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define inf 0x3f3f3f3f
5 typedef pair<int, int> P;
6 const int maxn = 5e5+5;
7 const ll mod = 1e9+7;
8
9 vector<int> g[maxn]; // 存图
10 // dep记录DFS序中每一个顶点的深度, vis记录DFS序, id记录顶点i第一次在DFS序
   中的位置, st表
11 int dep[maxn<<1]={0}, vis[maxn<<1]={0}, id[maxn]={0}, st[maxn<<1][25];
12 // dfs序计数用, 看代码能理解
13 int dfs_c=1;
14
15 // 父顶点为pre, 当前顶点为now, 当前深度为d
16 void dfs(int pre, int now, int d)
17 {
18     id[now] = dfs_c; // now顶点在DFS序中第一次出现的位置是dfs_c
19     dep[dfs_c] = d; // 记录now的深度
20     vis[dfs_c++] = now; // DFS序中第dfs_c个顶点是now, 同时将dfs_c加一
21     for(int i=0;i<g[now].size();i++)

```

```

22 {
23     if(g[now][i]!=pre)
24     {
25         dfs(now, g[now][i], d+1);
26         vis[dfs_c] = now;
27         dep[dfs_c++] = d;
28     }
29 }
30 }
31
32 // 预处理st表
33 void getSt(int n)
34 {
35     for(int i=1;i<=n;i++)
36     st[i][0] = i;
37     for(int j=1; (1<<j)<=n; j++)
38     {
39         for(int i=1;i+(1<<j)<=n; i++)
40         {
41             int a = st[i][j-1], b = st[i+(1<<(j-1))][j-1];
42             if(dep[a] < dep[b])
43                 st[i][j] = a;
44             else st[i][j] = b;
45         }
46     }
47 }
48
49 // 查询DFS序中区间[l, r]深度最小的顶点在DFS序中的位置
50 int query(int l, int r)
51 {
52     int k = log2(r-l+1);
53     int a = st[l][k];
54     int b = st[r-(1<<k)+1][k];
55     // 返回深度较小的那一个顶点在DFS序中的位置
56     if(dep[a]<dep[b])return a;
57     else return b;
58 }
59
60 // 求LCA(a, b)
61 int lca(int a, int b)
62 {
63     int x, y;
64     x = id[a], y = id[b];
65     if(x>y)return vis[query(y, x)];

```



```

66 else return vis[query(x, y)];
67 }
68
69 // 检查用的
70 void check(int n)
71 {
72     for(int i=1;i<=dfs_c;i++)cout<<dep[i]<<" ";cout<<"\n\n";
73     for(int i=1;i<=dfs_c;i++)cout<<vis[i]<<" ";cout<<"\n\n";
74     for(int i=1;i<=n;i++)cout<<id[i]<<" ";cout<<"\n\n";
75 }
76
77 int main()
78 {
79     int n, m, rt;
80     scanf("%d%d%d", &n, &m, &rt);
81     for(int i=1;i<n;i++)
82     {
83         int a, b;
84         scanf("%d%d", &a, &b);
85         g[a].push_back(b);
86         g[b].push_back(a);
87     }
88     dfs(0, rt, 1);
89     getSt(dfs_c);
90     //check(n);
91     for(int i=1;i<=m;i++)
92     {
93         int a, b;
94         scanf("%d%d", &a, &b);
95         printf("%d\n", lca(a, b));
96     }
97     return 0;
98 }

```

1.4 强连通分量

```

1 vector<int> g[maxn];
2 int low[maxn], dfn[maxn], sta[maxn], ins[maxn], belong[maxn];
3 int cnt, ind, tot; //cnt: 强连通分量的数量, ind: 时间戳, tot: sta的top
4
5 void init()
6 {

```

```

7     memset(ins, 0, sizeof(ins));
8     memset(belong, 0, sizeof(belong));
9     memset(dfn, 0, sizeof(dfn));
10    cnt = ind = tot = 0;
11 }
12
13 void Tarjan(int u)
14 {
15     low[u] = dfn[u] = ++ind;
16     ins[u] = 1;
17     sta[++tot] = u;
18     for(int i=0;i<g[u].size();i++)
19     {
20         int v = g[u][i];
21         if(!dfn[v])
22         {
23             Tarjan(v);
24             low[u] = min(low[u], low[v]);
25         }
26         else if(ins[v])
27             low[u] = min(low[u], dfn[v]);
28     }
29     int p;
30     if(low[u] == dfn[u])
31     {
32         ++cnt;
33         do
34         {
35             p = sta[tot--];
36             belong[p] = cnt;
37             ins[p] = 0;
38         }while(p != u);
39     }
40 }

```

1.5 割点

```

1 vector<int> g[maxn];
2 // iscut[i]: 若顶点i是割点, 则为1, 反之为0
3 int low[maxn], dfn[maxn], iscut[maxn];
4 int ind;
5

```

```

6 void init()
7 {
8     memset(dfn, 0, sizeof(dfn));
9     memset(iscut, 0, sizeof(iscut));
10    ind = 0;
11 }
12
13 // pa为u的父节点, 初始时Tarjan(i, i)
14 void Tarjan(int u, int pa)
15 {
16     int cnt = 0; //用来记录子树的数量
17     low[u] = dfn[u] = ++ind;
18     for(int i=0; i<g[u].size(); i++)
19     {
20         int v = g[u][i];
21         if(!dfn[v])
22         {
23             Tarjan(v, u);
24             low[u] = min(low[u], low[v]);
25             // 若low[v]>=dfn[u], 并且u不是根节点, 则u是割点
26             if(low[v] >= dfn[u] && pa!=u)
27                 iscut[u] = 1;
28             // 若u是根节点, 则cnt++
29             if(u == pa)
30                 cnt++;
31         }
32         else if(v != pa) //若v不等于父节点
33             low[u] = min(low[u], dfn[v]);
34     }
35     if(cnt>=2 && u==pa) //根节点子树数量大于等于2, 则为割点
36         iscut[u] = 1;
37 }

```

1.6 桥

```

1 // 用链式前向星来存储边
2 struct Edge
3 {
4     // iscut表示是否为桥
5     int to, next, iscut;
6 }e[maxn*maxn*2];
7

```

```

8 int head[maxn], low[maxn], dfn[maxn];
9 int ind, tot; // tot是边的数量
10
11 void init()
12 {
13     memset(head, -1, sizeof(head));
14     memset(dfn, 0, sizeof(dfn));
15     ind = tot = 0;
16 }
17
18 void addedge(int u, int v)
19 {
20     e[tot].to = v;
21     e[tot].next = head[u];
22     e[tot].iscut = 0;
23     head[u] = tot++;
24 }
25
26 void Tarjan(int u, int pa)
27 {
28     low[u] = dfn[u] = ++ind;
29     for(int i=head[u]; ~i; i = e[i].next)
30     {
31         int v = e[i].to;
32         if(v == pa) continue;
33         if(!dfn[v])
34         {
35             Tarjan(v, u);
36             low[u] = min(low[u], low[v]);
37             // 是桥
38             if(low[v] > dfn[u])
39             {
40                 e[i].iscut = e[i^1].iscut = 1;
41             }
42         }
43         else
44         {
45             low[u] = min(low[u], dfn[v]);
46         }
47     }
48 }

```

1.7 最大流

1.7.1 Dinic

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef pair<int, int> P;
5 const int maxn = 1e6+5;
6 const int inf = 0x3f3f3f3f;
7 const int mod = 1e9+7;
8
9 // 用链式前向星来存储图
10 struct ed
11 {
12     int to, val, ne;
13 }edge[maxn<<1];
14 int head[maxn], dep[maxn];
15 // 顶点数n, 边数m, 源点s, 汇点e, 加边时的指针tot
16 int n, m, s, e, tot;
17
18 void init()
19 {
20     tot = -1;
21     memset(head, -1, sizeof(head));
22 }
23
24 void addEdge(int u, int v, int val)
25 {
26     edge[++tot].to = v;
27     edge[tot].val = val;
28     edge[tot].ne = head[u];
29     head[u] = tot;
30 }
31
32 // 就是最普通的bfs
33 int bfs()
34 {
35     memset(dep, -1, sizeof(dep));
36     dep[s] = 0;
37     queue<int> q;
38     q.push(s);
39     while(!q.empty())
40     {

```

```

41         int u = q.front();
42         q.pop();
43         for(int i=head[u]; ~i; i=edge[i].ne)
44         {
45             int v = edge[i].to;
46             if(dep[v]==-1 && edge[i].val>0)
47             {
48                 dep[v] = dep[u]+1;
49                 q.push(v);
50             }
51         }
52     }
53     return (dep[e] != -1); //若dep[e]==-1则表示没有可以到达e的增广路了，算法结束。
54 }
55
56 // 当前顶点u, 当前流量flow
57 // 初始时dfs(s, inf)
58 int dfs(int u, int flow)
59 {
60     if(u == e)return flow;
61     for(int i=head[u]; ~i; i=edge[i].ne)
62     {
63         int v = edge[i].to;
64         if(dep[v]==dep[u]+1 && edge[i].val)
65         {
66             int a = dfs(v, min(flow, edge[i].val));
67             if(a>0) //若找到增广路
68             {
69                 edge[i].val -= a;
70                 edge[i^1].val += a;
71                 return a;
72             }
73         }
74     }
75     return 0;
76 }
77
78 ll dinic()
79 {
80     ll ans = 0;
81     while(bfs())
82     {
83         int a = dfs(s, (1<<30));

```

```

84     ans += a;
85 }
86 return ans;
87 }
88
89 int main()
90 {
91     scanf("%d%d%d", &n, &m, &s, &e);
92     init();
93     for(int i=1;i<=m;i++)
94     {
95         int u, v, w;
96         scanf("%d%d", &u, &v, &w);
97         addEdge(u, v, w);
98         addEdge(v, u, 0); //反边
99     }
100     printf("%lld\n", dinic());
101     return 0;
102 }

```

1.7.2 Dinic 优化

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef pair<int, int> P;
5 const int maxn = 1e6+5;
6 const int inf = 0x3f3f3f3f;
7 const int mod = 1e9+7;
8
9 struct ed
10 {
11     int to, val, ne;
12 }edge[maxn<<1];
13 int head[maxn],dep[maxn], cur[maxn];
14 int n, m, s, e, tot;
15
16 void init()
17 {
18     tot = -1;
19     memset(head, -1, sizeof(head));
20 }

```

```

21
22 void addEdge(int u, int v, int val)
23 {
24     edge[++tot].to = v;
25     edge[tot].val = val;
26     edge[tot].ne = head[u];
27     head[u] = tot;
28 }
29
30 int bfs()
31 {
32     memset(dep, -1, sizeof(dep));
33     dep[s] = 0;
34     queue<int> q;
35     q.push(s);
36     while(!q.empty())
37     {
38         int u = q.front();
39         q.pop();
40         for(int i=head[u]; ~i; i=edge[i].ne)
41         {
42             int v = edge[i].to;
43             if(dep[v]==-1 && edge[i].val>0)
44             {
45                 dep[v] = dep[u]+1;
46                 q.push(v);
47             }
48         }
49     }
50     return (dep[e] != -1);
51 }
52
53 int dfs(int u, int flow)
54 {
55     if(u == e) return flow;
56     // rflow用于多路增广, 表示流入到顶点u的剩余未流出的流量
57     int rflow = flow;
58     // 当前弧优化, 通过引用, 可以改变cur[i]的值, 使得下次遍历到顶点u时, 会直接
    // 从上次增广的边开始遍历
59     for(int& i=cur[u]; ~i; i=edge[i].ne)
60     {
61         int v = edge[i].to;
62         if(dep[v]==dep[u]+1 && edge[i].val)
63         {

```

```

64     int a = dfs(v, min(rflow, edge[i].val));
65     edge[i].val -= a;
66     edge[i^1].val += a;
67     rflow -= a; // 剩余流量要减少
68     if(rflow<=0)break; // 若没有剩余流量了, 就break
69 }
70 }
71 // 若没有一丝流量流出, 则表示通过顶点u已经无法增广了, 于是炸点, dep可以设置
    为任何无意义值
72 if(rflow == flow)
73     dep[u] = -2;
74     return flow - rflow; // 返回流出的流量
75 }
76
77 ll dinic()
78 {
79     ll ans = 0;
80     while(bfs())
81     {
82         // 新一轮dfs之前要对cur进行初始化
83         for(int i=1;i<=n;i++)cur[i] = head[i];
84         int a = dfs(s, (1<<30));
85         ans += a;
86     }
87     return ans;
88 }
89
90 int main()
91 {
92     scanf("%d%d%d", &n, &m, &s, &e);
93     init();
94     for(int i=1;i<=m;i++)
95     {
96         int u, v, w;
97         scanf("%d%d", &u, &v, &w);
98         addEdge(u, v, w);
99         addEdge(v, u, 0);
100     }
101     printf("%lld\n", dinic());
102     return 0;
103 }

```

1.8 二分图匹配

1.8.1 匈牙利算法

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  typedef pair<int, int> P;
5  const int maxn = 1e3+5;
6  const int inf = 0x3f3f3f3f;
7  const int mod = 1e9+7;
8
9
10 int mp[maxn][maxn];
11 int use[maxn], link[maxn];
12 int n, m, e, ans;
13
14 int found(int u)
15 {
16     for(int i=1;i<=m;i++)
17     {
18         if(!use[i] && mp[u][i])
19         {
20             use[i] = 1;
21             if(!link[i] || found(link[i]))
22             {
23                 link[i] = u;
24                 return 1;
25             }
26         }
27     }
28     return 0;
29 }
30
31 int main()
32 {
33     scanf("%d%d", &n, &m, &e);
34     for(int i=1;i<=e;i++)
35     {
36         int u, v;
37         scanf("%d", &u, &v);
38         if(u<=n && v<=m)
39         {
40             mp[u][v] = 1;

```

```

41     }
42 }
43 for(int i=1;i<=n;i++)
44 {
45     memset(use, 0, sizeof(use));
46     if(found(i))ans++;
47 }
48 printf("%d\n", ans);
49 return 0;
50 }

```

1.9 最小生成树

1.9.1 Prim

```

1 const int MAX=10000007;
2 int dis[5002],map[5002][5002],mark[5002];
3 int prim(int n)
4 {
5     for(int i=1;i<=n;i++) //初始化每个点到生成树中点的距离
6     {
7         dis[i]=map[1][i];
8         mark[i]=0;
9     }
10    dis[1]=0;
11    mark[1]=1; //1这个点加入生成树中。
12    int sum=0;
13    for(int i=1;i<n;i++) //枚举n-1条边
14    {
15        int sta=-1,Min=MAX;
16        for(int j=1;j<=n;j++) //找不在生成树中的点中距离生成树中的点长度最小的
17        {
18            if(!mark[j]&&dis[j]<Min)
19            {
20                Min=dis[j];
21                sta=j;
22            }
23        }
24        if(sta==-1) return -1; //没找到可以可以联通的路
25        mark[sta]=1; //新找到的点加入生成树
26        sum+=Min;
27        for(int j=1;j<=n;j++) //更新树外的点到树中的点的距离
28        {

```

```

29            if(!mark[j]&&dis[j]>map[sta][j])
30                dis[j]=map[sta][j];
31        }
32    }
33    return sum;
34 }
35
36 int main()
37 {
38     int n,m;
39     cin>>n>>m;
40     for(int i=1;i<=n;i++)
41     {
42         for(int j=1;j<=n;j++)
43         {
44             map[i][j]=MAX;
45         }
46     }
47     for(int i=1;i<=m;i++)
48     {
49         int a,b,c;
50         cin>>a>>b>>c;
51         if(c<map[a][b])
52         {
53             map[a][b]=c;
54             map[b][a]=c;
55         }
56     }
57     int ans = prim(n);
58     if(ans==-1)
59         cout<<"orz"<<endl;
60     else
61         cout<<ans<<endl;
62     return 0;
63 }

```

1.9.2 kruskal

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define N 5005
4 int father[N];

```

```

5
6 int find(int x)
7 {
8     int k = x;
9     while(father[k]!=k)
10    {
11        k = father[k];
12    }
13    while(father[x]!=x)
14    {
15        int temp = x;
16        x = father[x];
17        father[temp] = k;
18    }
19    return k;
20 }
21
22 void join(int a, int b)
23 {
24     int f1, f2;
25     f1 = find(a);
26     f2 = find(b);
27     father[f1] = f2;
28 }
29
30 struct edge
31 {
32     int node1, node2;
33     int cost;
34 };
35
36 vector<edge> edges;
37
38 bool cmp(edge a, edge b)
39 {
40     return a.cost > b.cost;
41 }
42
43 int kruskal(int n)
44 {
45     sort(edges.begin(), edges.end(), cmp);
46     for(int i=1;i<=n;i++)
47         father[i] = i;
48     int sum=0;

```

```

49     while(n!=1 && !edges.empty())
50     {
51         edge temp = edges[edges.size()-1];
52         edges.pop_back();
53         if(find(temp.node1)!=find(temp.node2))
54         {
55             sum += temp.cost;
56             n--;
57             join(temp.node1, temp.node2);
58         }
59     }
60     if(n!=1 && edges.empty())
61         sum = -1;
62     return sum;
63 }
64
65 int main()
66 {
67     int n,m;
68     int result;
69     cin>>n>>m;
70     for(int i=1;i<=m;i++)
71     {
72         int a,b,c;
73         cin>>a>>b>>c;
74         edge t;
75         t.node1=a;t.node2=b;t.cost=c;
76         edges.push_back(t);
77     }
78     result = kruskal(n);
79     if(result == -1)
80         cout<<"orz"<<endl;
81     else
82         cout<<result<<endl;
83     return 0;
84 }

```

1.10 次小生成树-POJ1679

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;

```

```

4 #define inf 0x3f3f3f3f
5 typedef pair<int, int> P;
6 const int maxn = 110;
7 const ll mod = 1e9+7;
8
9 int n, m;
10 int g[maxn][maxn];
11 int d[maxn], v[maxn], maxd[maxn][maxn], pre[maxn], mst[maxn][maxn];
12 int ans = 0;
13
14 void prim()
15 {
16     for(int i=1;i<=n;i++)
17     {
18         v[i] = 0; d[i] = inf; pre[i] = 1;
19     }
20     memset(maxd, 0, sizeof(maxd));
21     memset(mst, 0, sizeof(mst));
22     ans = 0;
23     priority_queue<P, vector<P>, greater<P> > q;
24     d[1] = 0; q.push(P(0, 1));
25     while(!q.empty())
26     {
27         P p = q.top(); q.pop();
28         int u = p.second;
29         if(v[u]) continue;
30         v[u] = 1; ans += d[u];
31         mst[pre[u]][u] = mst[u][pre[u]] = 1;
32         for(int i=1; i<=n;i++)
33         {
34             if(v[i] && g[u][i] < inf)
35                 maxd[u][i] = maxd[i][u] = max(maxd[pre[u]][u], d[u]);
36             if(d[i] > g[u][i])
37             {
38                 d[i] = g[u][i];
39                 pre[i] = u;
40                 q.push(P(d[i], i));
41             }
42         }
43     }
44 }
45
46 int main()
47 {

```

```

48     int t;
49     cin>>t;
50     while(t--)
51     {
52         memset(g, inf, sizeof(g));
53         cin>>n>>m;
54         while(m--)
55         {
56             int a, b, c;
57             cin>>a>>b>>c;
58             g[a][b] = g[b][a] = c;
59         }
60         prim();
61         int flag = 0;
62         for(int i=1;i<=n&&!flag;i++)
63         {
64             for(int j=1;j<=n;j++)
65             {
66                 if(mst[i][j] || g[i][j]==inf)continue;
67                 if(g[i][j] == maxd[i][j])
68                 {
69                     flag = 1;
70                     break;
71                 }
72             }
73         }
74         if(flag) cout<<"Not Unique!"<<endl;
75         else cout<<ans<<endl;
76     }
77     return 0;
78 }

```

1.11 拓扑排序

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int maxn=30;
4
5 int head[maxn],ip,indegree[maxn];
6 int n,m,seq[maxn];
7
8 struct note

```



```

9 {
10     int v,next;
11 }edge[maxn*maxn];
12
13 void init()
14 {
15     memset(head,-1,sizeof(head));
16     ip=0;
17 }
18
19 void addedge(int u,int v)
20 {
21     edge[ip].v=v,edge[ip].next=head[u],head[u]=ip++;
22 }
23
24 int topo()
25 {
26     queue<int>q;
27     int indeg[maxn];
28     for(int i=0; i<n; i++)
29     {
30         indeg[i]=indegree[i];
31         if(indeg[i]==0)
32             q.push(i);
33     }
34     int k=0;
35     bool res=false;
36     while(!q.empty())
37     {
38         if(q.size()!=1)res=true;
39         int u=q.front();
40         q.pop();
41         seq[k++]=u;
42         for(int i=head[u]; i!=-1; i=edge[i].next)
43         {
44             int v=edge[i].v;
45             indeg[v]--;
46             if(indeg[v]==0)
47                 q.push(v);
48         }
49     }
50     if(k<n)return -1;// no
51     if(res)return 0;// more
52     return 1; // only

```

```

53 }

```

1.12 Floyd 找最小环

```

1 const int INF = 0x3f3f3f3f;
2 const int MAXN = 110;
3
4 int n, m;           // n: 节点个数, m: 边的个数
5 int g[MAXN][MAXN];  // 无向图
6 int dist[MAXN][MAXN]; // 最短路径
7 int r[MAXN][MAXN];   // r[i][j]: i到j的最短路径的第一步
8 int out[MAXN], ct;    // 记录最小环
9
10 int solve(int i, int j, int k)
11 { // 记录最小环
12     ct = 0;
13     while (j != i)
14     {
15         out[ct++] = j;
16         j = r[i][j];
17     }
18     out[ct++] = i;
19     out[ct++] = k;
20     return 0;
21 }
22
23 int main()
24 {
25     while (scanf("%d%d", &n, &m) != EOF)
26     {
27         int i, j, k;
28         for (i = 0; i < n; i++)
29         {
30             for (j = 0; j < n; j++)
31             {
32                 g[i][j] = INF;
33                 r[i][j] = i;
34             }
35         }
36         for (i = 0; i < m; i++)
37         {
38             int x, y, l;

```

```

39     scanf("%d%d%d", &x, &y, &l);
40     —x;
41     —y;
42     if (l < g[x][y])
43     {
44         g[x][y] = g[y][x] = l;
45     }
46 }
47 memmove(dist, g, sizeof(dist));
48 int Min = INF;           // 最小环
49 for (k = 0; k < n; k++)
50 {                         // Floyd
51     for (i = 0; i < k; i++) // 一个环中的最大结点为k(编号最大)
52     {
53         if (g[k][i] < INF)
54         {
55             for (j = i + 1; j < k; j++)
56             {
57                 if (dist[i][j] < INF && g[k][j] < INF && Min > dist[i][j] + g[k]
58                     [i] + g[k][j])
59                 {
60                     Min = dist[i][j] + g[k][i] + g[k][j];
61                     solve(i, j, k); // 记录最小环
62                 }
63             }
64         }
65     }
66     for (i = 0; i < n; i++)
67     {
68         if (dist[i][k] < INF)
69         {
70             for (j = 0; j < n; j++)
71             {
72                 if (dist[k][j] < INF && dist[i][j] > dist[i][k] + dist[k][j])
73                 {
74                     dist[i][j] = dist[i][k] + dist[k][j];
75                     r[i][j] = r[k][j];
76                 }
77             }
78         }
79     }
80     if (Min < INF)
81     {

```

```

82         for (ct—; ct >= 0; ct—)
83         {
84             printf("%d", out[ct] + 1);
85             if (ct)
86             {
87                 printf(" ");
88             }
89         }
90     }
91     else
92     {
93         printf("No solution.");
94     }
95     printf("\n");
96 }
97 return 0;
98 }

```

1.13 生成树计数 (基尔霍夫矩阵) BZOJ1002

```

1 // f[i] = 3*f[i-1] - f[i-2] + 2
2 // 需要高精度
3 #include<bits/stdc++.h>
4 using namespace std;
5 typedef long long ll;
6 typedef unsigned long long ull;
7 typedef pair<int, int> P;
8 const int maxn = 1e6+5;
9 const int inf = 0x3f3f3f3f;
10 const int mod = 1e9+7;
11 const double PI = acos(-1);
12
13
14 struct node
15 {
16     int a[101], len;
17 };
18 node f[101];
19 node mul(node a, int k)
20 {
21     for(int i=1;i<=a.len;i++)
22         a.a[i]*=k;

```

```

23   for(int i=1;i<=a.len;i++)
24   {
25       a.a[i+1]+=a.a[i]/10;
26       a.a[i]%=10;
27   }
28   while(a.a[a.len+1]!=0)a.len++;
29   return a;
30 }
31
32 node sub(node a, node b)
33 {
34     a.a[1] += 2;
35     int j = 1;
36     while(a.a[j]>=10)
37     {
38         a.a[j]%=10;
39         a.a[j+1]++;
40         j++;
41     }
42     for(int i=1;i<=a.len;i++)
43     {
44         a.a[i] -= b.a[i];
45         if(a.a[i]<0)
46         {
47             a.a[i]+=10;
48             a.a[i+1]--;
49         }
50     }
51     while(a.a[a.len]==0)a.len--;
52     return a;
53 }
54
55 int main()
56 {
57     f[1].a[1] = 1; f[2].a[1] = 5;
58     f[1].len = f[2].len = 1;
59     int n;
60     scanf("%d", &n);
61     for(int i=3;i<=n;i++)
62         f[i] = sub(mul(f[i-1], 3), f[i-2]);
63     for(int i=f[n].len;i>0;i--)
64         printf("%d", f[n].a[i]);
65     printf("\n");
66     return 0;

```

```

67 }

```

1.14 悬线法 BZOJ3039

```

1 // 悬线法求最大子矩阵
2 // 给定n*m的矩阵，含F和R，求最大的全F矩阵的面积*3
3
4 #include<bits/stdc++.h>
5 using namespace std;
6 typedef long long ll;
7 typedef unsigned long long ull;
8 typedef pair<int, int> P;
9 const int maxn = 1e3+5;
10 const int inf = 0x3f3f3f3f;
11 const ll INF = 0x3f3f3f3f3f3f3f3f;
12 const int mod = 1e9+7;
13 const double PI = acos(-1);
14
15
16 char mp[maxn][maxn];
17 int l[maxn][maxn], r[maxn][maxn], h[maxn][maxn];
18 int n, m;
19
20 int main()
21 {
22     ios::sync_with_stdio(0);
23     cin>>n>>m;
24     for(int i=1;i<=n;i++)
25     {
26         for(int j=1;j<=m;j++)
27         {
28             cin>>mp[i][j];
29         }
30     }
31     for(int i=1;i<=n;i++)
32     {
33         int tmp = 1;
34         for(int j=1;j<=m;j++)
35         {
36             if(mp[i][j] == 'F')
37             {
38                 l[i][j] = tmp;

```

```

39     if(mp[i-1][j]=='F')
40         l[i][j] = max(l[i][j], l[i-1][j]);
41     }
42     else tmp = j+1;
43     }
44 }
45 for(int i=1;i<=n;i++)
46 {
47     int tmp = m;
48     for(int j=m;j>=1;j--)
49     {
50         if(mp[i][j]=='F')
51         {
52             r[i][j] = tmp;
53             if(mp[i-1][j] == 'F')
54                 r[i][j] = min(r[i][j], r[i-1][j]);
55         }
56         else tmp = j-1;
57     }
58 }
59 for(int i=1;i<=n;i++)
60 {
61     for(int j=1;j<=m;j++)
62     {
63         if(mp[i][j] == 'F')
64             h[i][j] = h[i-1][j]+1;
65     }
66 }
67 int ans = 0;
68 for(int i=1;i<=n;i++)
69 {
70     for(int j=1;j<=m;j++)
71     {
72         if(mp[i][j]=='F')
73             ans = max(ans, h[i][j]*(r[i][j]-l[i][j]+1));
74     }
75 }
76 cout<<3*ans<<endl;
77 return 0;
78 }

```

2 数据结构

2.1 并查集

```

1 int pre[maxn];
2
3 int Find(int x)
4 {
5     int p,tmp;
6     p=x;
7     while(x!=pre[x])
8         x=pre[x];
9     while(p!=x)
10    {
11        tmp=pre[x];
12        pre[x]=x;
13        p=tmp;
14    }
15    return x;
16 }
17
18 void join(int x,int y)
19 {
20     int fx=Find(x);
21     int fy=Find(y);
22     if(fx!=fy)
23         pre[fx]=fy;
24 }

```

2.2 带权并查集

2.2.1 BZOJ1202

```

1 \\ 给出n个月份，每个月份都有营业额
2 \\ m个判断，表示区间[l, r]的和为w
3 \\ 判断是否有矛盾
4 #include<bits/stdc++.h>
5 using namespace std;
6 typedef long long ll;
7 typedef unsigned long long ull;
8 typedef pair<int, int> P;
9 const int maxn = 1e3+5;

```

```

10 const int inf = 0x3f3f3f3f;
11 const ll INF = 0x3f3f3f3f3f3f3f3f;
12 const int mod = 1e9+7;
13 const double PI = acos(-1);
14
15
16
17 int n, m;
18 int fa[105], d[105];
19 int find(int x)
20 {
21     if(fa[x]==x)return x;
22     int xx = find(fa[x]);
23     d[x] += d[fa[x]];
24     return fa[x] = xx;
25 }
26 int main()
27 {
28     int t;
29     scanf("%d", &t);
30     while(t--)
31     {
32         scanf("%d%d", &n, &m);
33         memset(d, 0, sizeof(d));
34         for(int i=0;i<=n;i++) fa[i] = i;
35         int f = 1;
36         while(m--)
37         {
38             int s, t, w;
39             scanf("%d%d%d", &s, &t, &w);
40             s--;
41             int fas = find(s), fat = find(t);
42             if(fas != fat)
43             {
44                 fa[fat] = fas;
45                 d[fat] = d[s]+w-d[t];
46             }
47             else if(d[t]-d[s]!=w)f=0;
48         }
49         if(f)printf("true\n");
50         else printf("false\n");
51     }
52     return 0;
53 }

```

2.3 ST 表

```

1 int st[maxn][20];
2 void st_init()
3 {
4     for(int i=1;i<=n;i++) st[i][0]=a[i]; // 长度为1的区间最小值党委就为自身
5     // 预处理从i开始, 长度为2^j的区间
6     for(int j=1;(1<<j)<=n;j++)
7         for(int i=1;i+(1<<j)-1<=n;i++)
8             st[i][j]=max(st[i][j-1],st[i+(1<<(j-1))][j-1]);
9 }
10
11 int query(int l,int r)
12 {
13     int k=log2(r-l+1);
14     return min(st[l][k],st[r-(1<<k)+1][k]);
15 }

```

2.4 树状数组

```

1 /*
2 * INIT: ar[]置为0;
3 * CALL: add(i, v): 将i点的值加v; sum(i): 求[1, i]的和;
4 */
5 #define typev int // type of res
6 const int N = 1010;
7 typev ar[N]; // index: 1 ~ N
8 int lowb(int t)
9 {
10     return t & (-t);
11 }
12
13 void add(int i, typev v)
14 {
15     for (; i < N; ar[i] += v, i += lowb(i));
16     return ;
17 }
18
19 typev sum(int i)

```

```

20 {
21     typev s = 0;
22     for (; i > 0; s += ar[i], i -= lowb(i));
23     return s;
24 }

```

2.5 线段树

2.5.1 单点修改 + 区间求和 HDU1166

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5 int num[50005];
6 ll tree[4 * 50000 + 5];
7
8 void build(int p, int l, int r)
9 {
10     if(l == r)
11     {
12         tree[p] = num[l];
13         return ;
14     }
15     else
16     {
17         int mid = (r+l) >> 1;
18         build(p<<1, l, mid);
19         build(p<<1|1, mid+1, r);
20         tree[p] = tree[p<<1] + tree[p<<1|1];
21     }
22 }
23
24 void add(int p, int l, int r, int ind, int v)
25 {
26     if(l == r)
27     {
28         tree[p] += v;
29         return ;
30     }
31     else
32     {
33         int mid = (r+l) >> 1;

```

```

34         if(ind <= mid) add(p<<1, l, mid, ind, v);
35         else add(p<<1|1, mid+1, r, ind, v);
36         tree[p] = tree[p<<1] + tree[p<<1|1];
37     }
38 }
39
40 ll query(int p, int l, int r, int x, int y)
41 {
42     if(x <= l && r <= y)
43     {
44         return tree[p];
45     }
46     else
47     {
48         int mid = (l+r) >> 1;
49         ll ans = 0;
50         if(x <= mid)
51             ans += query(p<<1, l, mid, x, y);
52         if(mid < y)
53             ans += query(p<<1|1, mid+1, r, x, y);
54         return ans;
55     }
56 }
57
58 int main()
59 {
60     int t;
61     scanf("%d", &t);
62     for(int i=1;i<=t;i++)
63     {
64         int n;
65         scanf("%d", &n);
66         for(int j=1;j<=n;j++)
67             scanf("%d", &num[j]);
68         build(1, 1, n);
69         string s;
70         printf("Case %d:\n", i);
71         while(cin>>s && s[0] != 'E')
72         {
73             if(s[0] == 'Q')
74             {
75                 int x, y;
76                 scanf("%d%d", &x, &y);
77                 printf("%lld\n", query(1, 1, n, x, y));

```

```

78     }
79     else if(s[0] == 'A')
80     {
81         int x, y;
82         scanf("%d%d", &x, &y);
83         add(1, 1, n, x, y);
84     }
85     else
86     {
87         int x, y;
88         scanf("%d%d", &x, &y);
89         add(1, 1, n, x, -y);
90     }
91 }
92 }
93 return 0;
94 }

```

2.5.2 区间修改 + 区间求和 POJ3465

```

1 #include<iostream>
2 using namespace std;
3 typedef long long ll;
4 ll num[100005];
5 ll tree[4 * 100000 + 5];
6 ll lazy[4 * 100000 + 5]={0};
7 int n,m;
8 void build(int p, int l, int r)
9 {
10     if(l == r)
11         tree[p] = num[l];
12     else
13     {
14         int mid = (l+r) >> 1;
15         build(p<<1, l, mid);
16         build(p<<1|1, mid+1, r);
17         tree[p] = tree[p<<1] + tree[p<<1|1];
18     }
19 }
20
21 void pushdown(int p, int l, int r)
22 {

```

```

23     if(lazy[p])
24     {
25         lazy[p<<1] += lazy[p];
26         lazy[p<<1|1] += lazy[p];
27         tree[p<<1] += lazy[p] * (((l+r)>>1) - 1 + 1);
28         tree[p<<1|1] += lazy[p] * (r - ((l+r)>>1));
29         lazy[p] = 0;
30     }
31 }
32
33 void add(int p, int l, int r, int x, int y, ll v)
34 {
35     if(x <= l && r <= y)
36     {
37         lazy[p] += v;
38         tree[p] += v * (r-l+1);
39         return ;
40     }
41     else
42     {
43         int mid = (l+r) >> 1;
44         pushdown(p, l, r);
45         if(x <= mid)
46             add(p<<1, l, mid, x, y, v);
47         if(y > mid)
48             add(p<<1|1, mid+1, r, x, y, v);
49         tree[p] = tree[p<<1] + tree[p<<1|1];
50     }
51 }
52
53 ll query(int p, int l, int r, int x, int y)
54 {
55     if(x <= l && r <= y)
56     {
57         return tree[p];
58     }
59     else
60     {
61         int mid = (l+r) >> 1;
62         ll ans = 0;
63         pushdown(p, l, r);
64         if(x <= mid)
65             ans += query(p<<1, l, mid, x, y);
66         if(y > mid)

```

```

67     ans += query(p<<1|1, mid+1, r, x, y);
68     return ans;
69 }
70 }
71
72 int main()
73 {
74     cin>>n>>m;
75     for(int i=1;i<=n;i++)
76         cin>>num[i];
77     build(1, 1, n);
78     for(int i=1;i<=m;i++)
79     {
80         char a;
81         cin>>a;
82         if(a == 'Q')
83         {
84             int x, y;
85             cin>>x>>y;
86             cout<<query(1, 1, n, x, y)<<endl;
87         }
88         else
89         {
90             int x, y;
91             ll c;
92             cin>>x>>y>>c;
93             add(1, 1, n, x, y, c);
94         }
95     }
96     return 0;
97 }

```

2.5.3 单点修改 + 区间最值 HDU1754

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 int n,m;
5
6 int num[200005];
7 int tree[200000 * 4 + 5];
8

```

```

9 void build(int p, int l, int r)
10 {
11     if(l == r)
12     {
13         tree[p] = num[l];
14         return ;
15     }
16     else
17     {
18         int mid = (l+r) >> 1;
19         build(p<<1, l, mid);
20         build(p<<1|1, mid+1, r);
21         tree[p] = max(tree[p<<1], tree[p<<1|1]);
22         return ;
23     }
24 }
25
26 void update(int p, int l, int r, int ind, int v)
27 {
28     if(l == r)
29     {
30         tree[p] = v;
31         return ;
32     }
33     else
34     {
35         int mid = (l+r) >> 1;
36         if(ind <= mid)
37             update(p<<1, l, mid, ind, v);
38         else
39             update(p<<1|1, mid+1, r, ind, v);
40         tree[p] = max(tree[p<<1], tree[p<<1|1]);
41     }
42 }
43
44 ll query(int p, int l, int r, int x, int y)
45 {
46     if(x <= l && r <= y)
47     {
48         return tree[p];
49     }
50     else
51     {
52         int mid = (l+r) >> 1;

```



```

53     ll ans = -1;
54     if(x <= mid)
55         ans = max(ans, query(p<<1, l, mid, x, y));
56     if(y > mid)
57         ans = max(ans, query(p<<1|1, mid+1, r, x, y));
58     return ans;
59 }
60 }
61
62 int main()
63 {
64     while(scanf("%d %d", &n, &m) != EOF)
65     {
66         for(int i=1;i<=n;i++)
67         {
68             scanf("%d", &num[i]);
69         }
70         build(1, 1, n);
71         for(int i=1;i<=m;i++)
72         {
73             char a;
74             int x, y;
75             scanf(" %c%d%d", &a, &x, &y);
76             if(a == 'Q')
77             {
78                 printf("%lld\n", query(1, 1, n, x, y));
79             }
80             else
81             {
82                 update(1, 1, n, x, y);
83             }
84         }
85     }
86     return 0;
87 }

```

2.5.4 区间染色 + 统计 + 离散化 POJ2528

```

1 #include<iostream>
2 #include<algorithm>
3 #include<string.h>
4 using namespace std;

```

```

5 typedef long long ll;
6
7 const int maxn = 20000 + 100;
8 int tree[maxn<<4];
9 int li[maxn],ri[maxn];
10 bool vis[maxn];
11 int lisan[maxn*3];
12 int ans = 0;
13
14 void init()
15 {
16     memset(tree, -1, sizeof(tree));
17     memset(vis, 0, sizeof(vis));
18     ans = 0;
19 }
20
21 void pushdown(int p)
22 {
23     tree[p<<1] = tree[p<<1|1] = tree[p];
24     tree[p] = -1;
25 }
26
27 void update(int p, int l, int r, int x, int y, int v)
28 {
29     if(x <= l && r <= y)
30     {
31         tree[p] = v;
32         return ;
33     }
34     if(tree[p]!=-1)
35         pushdown(p);
36     int mid = (l+r)>>1;
37     if(x <= mid)
38         update(p<<1, l, mid, x, y, v);
39     if(y > mid)
40         update(p<<1|1, mid+1, r, x, y, v);
41     tree[p] = -1;
42 }
43
44 void query(int p, int l, int r)
45 {
46     if(tree[p]!=-1)
47     {
48         if(vis[tree[p]]==0)

```

```

49     {
50         vis[tree[p]] = 1;
51         ans++;
52     }
53     return;
54 }
55 if(l==r)return;
56 int mid = (l+r)>>1;
57 query(p<<1, l, mid);
58 query(p<<1|1, mid+1, r);
59 }
60
61 int main()
62 {
63     int t;
64     cin>>t;
65     while(t--)
66     {
67         init();
68         int n;
69         cin>>n;
70         int tot = 0;
71         for(int i=0;i<n;i++)
72         {
73             cin>>li[i]>>ri[i];
74             lisan[tot++] = li[i];
75             lisan[tot++] = ri[i];
76         }
77         sort(lisan, lisan+tot);
78         int m = unique(lisan, lisan+tot) - lisan;
79         int t = m;
80         for(int i=1;i<t;i++)
81         {
82             if(lisan[i]-lisan[i-1]>1)
83             {
84                 lisan[m++] = lisan[i-1]+1;
85             }
86         }
87         sort(lisan, lisan+m);
88         for(int i=0;i<n;i++)
89         {
90             int x,y;
91             x = lower_bound(lisan, lisan+m, li[i]) - lisan;
92             y = lower_bound(lisan, lisan+m, ri[i]) - lisan;

```

```

93         update(1, 0, m-1, x, y, i);
94     }
95     query(1, 0, m-1);
96     cout<<ans<<endl;
97 }
98 return 0;
99 }

```

2.5.5 线段树 + 扫描线求矩阵覆盖周长 POJ1177

```

1  #include<iostream>
2  #include<vector>
3  #include<algorithm>
4  #include<cmath>
5  using namespace std;
6  typedef long long ll;
7
8  const int maxn = 10005;
9  vector<int> x;
10 int getID(int v)
11 {
12     return lower_bound(x.begin(), x.end(), v) - x.begin();
13 }
14
15 struct Segment
16 {
17     int l, r;
18     int h;
19     int flag;
20 }segment[maxn];
21 bool cmp(Segment a, Segment b)
22 {
23     return a.h < b.h;
24 }
25
26 struct Node
27 {
28     int l,r;
29     int lr, rr;
30     int len;
31     int line;
32     int s;

```

```

33 }node[maxn<<2];
34
35 void build(int p, int l, int r)
36 {
37     node[p].l = l; node[p].r = r;
38     node[p].line = node[p].len = node[p].s = 0;
39     node[p].lr = node[p].rr = 0;
40     if(l==r)return;
41     int mid = (l+r)>>1;
42     build(p<<1, l, mid);
43     build(p<<1|1, mid+1, r);
44 }
45
46 void pushup(int p)
47 {
48     if(node[p].s)
49     {
50         node[p].line = 1;
51         node[p].rr = node[p].lr = 1;
52         node[p].len = x[node[p].r+1] - x[node[p].l];
53         return;
54     }
55     else if(node[p].l == node[p].r)
56     {
57         node[p].lr = node[p].rr = node[p].line = node[p].len = 0;
58     }
59     else
60     {
61         node[p].lr = node[p<<1].lr;
62         node[p].rr = node[p<<1|1].rr;
63         node[p].len = node[p<<1].len + node[p<<1|1].len;
64         node[p].line = node[p<<1].line + node[p<<1|1].line - (node[p<<1].rr&&
        node[p<<1|1].lr);
65     }
66 }
67
68 void update(int p, int l, int r, int v)
69 {
70     if(node[p].r < l || node[p].l > r)return;
71     if(l <= node[p].l && node[p].r <= r)
72     {
73         node[p].s += v;
74         pushup(p);
75         return;

```

```

76     }
77     update(p<<1, l, r, v);
78     update(p<<1|1, l, r, v);
79     pushup(p);
80 }
81
82 int main()
83 {
84     int n;
85     cin>>n;
86     for(int i=1;i<=n;i++)
87     {
88         int x1,x2,y1,y2;
89         cin>>x1>>y1>>x2>>y2;
90         Segment &s1 = segment[2*i-1];
91         Segment &s2 = segment[i<<1];
92         s1.l = s2.l = x1;
93         s1.r = s2.r = x2;
94         s1.h = y1; s2.h = y2;
95         s1.flag = 1; s2.flag = -1;
96         x.push_back(x1);
97         x.push_back(x2);
98     }
99     sort(segment+1, segment+2*n+1, cmp);
100
101     sort(x.begin(), x.end());
102     x.erase(unique(x.begin(), x.end()), x.end());
103
104     build(1, 0, x.size()-1);
105
106     ll ans = 0;
107     int last = 0;
108     for(int i=1;i<=2*n;i++)
109     {
110         int l = getID(segment[i].l);
111         int r = getID(segment[i].r);
112         update(1, l, r-1, segment[i].flag);
113         ans += abs(node[1].len - last);
114         if(i!=2*n)
115             ans += node[1].line * 2 * (segment[i+1].h - segment[i].h);
116         last = node[1].len;
117     }
118     cout<<ans<<endl;
119     return 0;

```

120 }

2.5.6 线段树 + 扫描线求矩阵面积并 HDU1542

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5 const int maxn = 210;
6 int n;
7 vector<double> x;
8 inline int getID(double v)
9 {
10     return lower_bound(x.begin(), x.end(), v) - x.begin();
11 }
12
13 struct Segment
14 {
15     double l, r;
16     double h;
17     int flag;
18 }segment[maxn];
19 bool cmp(Segment a, Segment b)
20 {
21     return a.h < b.h;
22 }
23
24 struct Node
25 {
26     int l, r;
27     int s;
28     double len;
29 }node[maxn<<2];
30
31 void pushup(int p)
32 {
33     if(node[p].s)
34         node[p].len = x[node[p].r+1] - x[node[p].l];
35     else if(node[p].l == node[p].r)
36         node[p].len = 0;
37     else
38         node[p].len = node[p<<1].len + node[p<<1|1].len;

```

```

39 }
40
41 void build(int p, int l, int r)
42 {
43     if(l>r)return;
44     node[p].l = l; node[p].r = r;
45     node[p].s = 0; node[p].len = 0;
46     if(l==r) return;
47     int mid = (l+r)>>1;
48     build(p<<1, l, mid);
49     build(p<<1|1, mid+1, r);
50     pushup(p);
51 }
52
53 void update(int p, int l, int r, int v)
54 {
55     if(l>node[p].r || r<node[p].l) return;
56     if(l <= node[p].l && node[p].r <= r)
57     {
58         node[p].s += v;
59         pushup(p);
60         return;
61     }
62     update(p<<1, l, r, v);
63     update(p<<1|1, l, r, v);
64     pushup(p);
65 }
66
67 int main()
68 {
69     int cas = 0;
70     while(scanf("%d", &n) && n)
71     {
72         x.clear();
73         for(int i=1;i<=n;i++)
74         {
75             double x1,x2,y1,y2;
76             scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
77             Segment &s1 = segment[2*i-1];
78             Segment &s2 = segment[i<<1];
79             s1.l=s2.l=x1;
80             s1.r=s2.r=x2;
81             s1.h=y1;
82             s2.h=y2;

```

```

83     s1.flag=1;
84     s2.flag=-1;
85     x.push_back(x1);
86     x.push_back(x2);
87 }
88 sort(segment+1, segment+2*n+1, cmp);
89
90 sort(x.begin(), x.end());
91 x.erase(unique(x.begin(), x.end()), x.end());
92
93 build(1, 0, x.size()-1);
94 double ans = 0;
95 for(int i=1;i<=2*n;i++)
96 {
97     int l=getID(segment[i].l);
98     int r=getID(segment[i].r);
99     update(1, l, r-1, segment[i].flag);
100    ans+=node[1].len*(segment[i+1].h - segment[i].h);
101 }
102 printf("Test case #%d\n", ++cas);
103 printf("Total explored area: %.2f\n\n", ans);
104 }
105 return 0;
106 }

```

2.5.7 状压 + 线段树维护区间 26 字母种类 CF1234D

```

1  /*
2  * 操作1: 将原pos位置的字母改为字母c
3  * 操作2: 查询区间[l, r]的字母种类数
4  */
5
6  #include<bits/stdc++.h>
7  using namespace std;
8  typedef long long ll;
9  typedef pair<int, int> P;
10 const int maxn = 1e5+5;
11 const int inf = 0x3f3f3f3f;
12 const int mod = 1e9+7;
13
14 char s[maxn];
15 const int N=1e5+5;

```

```

16 struct fun{
17     bitset<270>bit;
18 }tree[N<<2],v[N<<2];
19
20
21
22 void sett(int l,int r,int rt){ // 建树
23     if(l==r){
24         tree[rt].bit=0;
25         tree[rt].bit.set(s[l]-'a');
26         return ;
27     }
28     int mid=(l+r)>>1;
29     sett(l,mid,rt<<1);
30     sett(mid+1,r,rt<<1|1);
31     tree[rt].bit=(tree[rt<<1].bit|tree[rt<<1|1].bit);
32 }
33
34 void fun(int l,int r,int rt){ // 下推
35     if(v[rt].bit.count()){ // 统计 bit 中 1 的个数
36         v[rt<<1].bit=v[rt].bit;
37         v[rt<<1|1].bit=v[rt].bit;
38         tree[rt<<1].bit=tree[rt].bit;
39         tree[rt<<1|1].bit=tree[rt].bit;
40         v[rt].bit=0; // 记得清零
41     }
42 }
43
44 void upset(int x,int y,int vel,int l,int r,int rt){
45     if(x<=l&&y>=r){
46         v[rt].bit.set(vel);
47         tree[rt].bit=0;
48         tree[rt].bit.set(vel);
49         return ;
50     }
51     fun(l,r,rt);
52     int mid=l+r>>1;
53     if(x<=mid) upset(x,y,vel,l,mid,rt<<1);
54     if(y>mid) upset(x,y,vel,mid+1,r,rt<<1|1);
55     tree[rt].bit=(tree[rt<<1].bit|tree[rt<<1|1].bit);
56 }
57 bitset<270>bb(0);
58 void findd(int x,int y,int l,int r,int rt){
59     if(x<=l&&y>=r){

```

```

60     bb|=tree[rt].bit;
61     return ;
62 }
63
64 fun(l,r,rt);
65 int mid=l+r>>1;
66 if(x<=mid)
67     findd(x,y,l,mid,rt<<1);
68 if(y>mid)
69     findd(x,y,mid+1,r,rt<<1|1);
70 }
71
72
73 int main()
74 {
75     int q;
76     int op;
77     int l, r;
78     char ch;
79     int ind;
80     int n;
81     scanf("%s", s+1);
82     n = strlen(s+1);
83     scanf("%d", &q);
84     sett(1, n, 1);
85     while(q--)
86     {
87         scanf("%d", &op);
88         if(op==1)
89         {
90             scanf("%d ", &ind);
91             ch = getchar();
92             upset(ind, ind, ch-'a', 1, n, 1);
93         }
94         else
95         {
96             scanf("%d%d", &l, &r);
97             bb = 0;
98             findd(1, r, 1, n, 1);
99             printf("%d\n", bb.count());
100         }
101     }
102     return 0;
103 }

```

2.6 主席树

2.6.1 区间第 K 大

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define inf 0x3f3f3f3f
5 typedef pair<int, int> P;
6 const int maxn = 1e6+5;
7 const ll mod = 1e9+7;
8
9 // 顶点，代表区间[l, r]中有v个数字
10 struct node
11 {
12     int l, r, v;
13 }tree[maxn*20]; // 空间开大一点，因为要动态开点
14
15 // edit[i]存的是第i颗权值线段树的根节点在tree数组中的位置
16 // a是存放原数据的数组， b是离散化后的数组， tot表示顶点的个数
17 int edit[maxn], a[maxn], b[maxn], tot=0;
18
19 // 建树
20 int build(int l, int r)
21 {
22     // 这里就是动态开辟新的结点，就是将tot加一
23     int pos = ++tot;
24     tree[pos].v = 0; //初始化为0
25     if(l==r)return pos; //到根节点了，返回
26     // 二分建树没什么好说的
27     int mid = (l+r)>>1;
28     tree[pos].l = build(l, mid);
29     tree[pos].r = build(mid+1, r);
30     //要返回pos的位置，因为edit数组要存新根的位置
31     return pos;
32 }
33
34 // 就是插入操作，插入新的元素
35 // ed是前一版本的结点在tree的位置， 更新区间[l, r], 位置为v
36 int update(int ed, int l, int r, int v)
37 {

```

```

38 // 动态开点
39 int pos = ++tot;
40 //先将新一版本的当前结点复制为上一个版本的对应结点
41 tree[pos] = tree[ed]; tree[pos].v++; //新一版本的结点的v要加一，因为对应区
   间插入了一个数
42 if(l==r) return pos; //到叶子节点了，返回
43 // 二分
44 int mid = (l+r)>>1;
45 // 如果更新位置v在左子树中，递归更新即可，在右子树中同理，最后要返回pos
46 if(v<=mid) tree[pos].l = update(tree[ed].l, l, mid, v);
47 else tree[pos].r = update(tree[ed].r, mid+1, r, v);
48 return pos;
49 }
50
51 // 查询区间[l, r]第k大的数字，pre对应l-1版本的权值线段树的节点位置， ed代表r
   版本的权值线段树的节点位置
52 int query(int pre, int ed, int l, int r, int k)
53 {
54     // 到叶子节点就返回
55     if(l==r) return l;
56     // 二分
57     int mid = (l+r)>>1;
58     // 先计算左子树的数字个数
59     int x = tree[tree[ed].l].v - tree[tree[pre].l].v;
60     // 若左子树的数字个数大于等于k，说明我们要找的数字在左子树中，递归走到左子
   树继续寻找
61     if(x>=k) return query(tree[pre].l, tree[ed].l, l, mid, k);
62     // 否则就在右子树中，我们要在右子树中寻找k-x大的数字，递归寻找就好
63     else return query(tree[pre].r, tree[ed].r, mid+1, r, k-x);
64 }
65
66 int main()
67 {
68     int n, q; // n个数字， q次询问
69     scanf("%d%d", &n, &q); // 这一题cin/cout会被卡
70     // 输入数据，并copy到b数组中
71     for(int i=1;i<=n;i++)
72     {
73         scanf("%d", &a[i]);
74         b[i] = a[i];
75     }
76     // 离散化
77     sort(b+1, b+1+n);
78     int m = unique(b+1, b+1+n) - b-1;

```

```

79 // 构建一颗空的权值线段树，edit[0]存放的就是这颗空树的根节点的位置
80 edit[0] = build(1, m);
81 // 插入n个数据
82 for(int i=1;i<=m;i++)
83 {
84     // 找到a[i]离散化后对应的位置
85     a[i] = lower_bound(b+1, b+m+1, a[i]) - b;
86     // edit[i]存放第i版本的权值线段树的根节点位置
87     edit[i] = update(edit[i-1], 1, m, a[i]);
88 }
89 // 处理q次询问
90 while(q--)
91 {
92     int x, y, k;
93     scanf("%d%d%d", &x, &y, &k);
94     // pos对应的是离散化后的位置，所以最后输出b[pos]即可
95     int pos = query(edit[x-1], edit[y], 1, m, k);
96     printf("%d\n", b[pos]);
97 }
98 return 0;
99 }

```

2.6.2 动态区间第 K 大 (主席树套树状数组) ZOJ2112

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define inf 0x3f3f3f3f
5 typedef pair<int, int> P;
6 const int maxn = 60010;
7 const ll mod = 1e9+7;
8 const int M = 2500010;
9
10 int n, m, q, tot;
11 struct node
12 {
13     int l, r, v;
14 }tree[M];
15
16 // T是主席树，与上面代码的edit作用一样， S数组就是树状数组， use数组是树状数
   组求和时用的，记录的是树状数组中哪一些权值线段树要被用来求和
17 int T[maxn], S[maxn], use[maxn], a[maxn], b[maxn];

```

```

18
19 // 记录询问，因为要将修改后的值一起构建主席树，所以将在线转为离线
20 struct Q
21 {
22     // 对于查询区间[l, r]第k大的询问来说，flag为1
23     //若是修改操作，l记录修改的位置，r记录新值，flag为0
24     int l, r, k, flag;
25 }query[10010];
26
27 // 快速找出x在离散化后的位置
28 int HASH(int x)
29 {
30     return lower_bound(b+1, b+m+1, x) - b;
31 }
32
33 // 建静态主席树，和之前的一样
34 int build(int l, int r)
35 {
36     int pos = ++tot;
37     tree[pos].v = 0;
38     if(l==r) return pos;
39     int mid = (l+r)>>1;
40     tree[pos].l = build(l, mid);
41     tree[pos].r = build(mid+1, r);
42     return pos;
43 }
44
45 // 和之前的静态主席树update差不多，只不过不是直接将tree[pos].v+1,而是加参数v
46 //消除影响v就为-1，添加影响v就为1，其他没什么不同
47 int update(int ed, int l, int r, int p, int v)
48 {
49     int pos = ++tot;
50     tree[pos] = tree[ed];
51     tree[pos].v += v;
52     if(l==r) return pos;
53     int mid = (l+r)>>1;
54     if(p<=mid) tree[pos].l = update(tree[ed].l, l, mid, p, v);
55     else tree[pos].r = update(tree[ed].r, mid+1, r, p, v);
56     return pos;
57 }
58
59 // 树状数组的lowbit
60 int lowbit(int x) { return x&(-x); }
61

```

```

62 // 修改操作，修改位置x的影响
63 int add(int x, int v)
64 {
65     // 找出a[x]在离散化后的位置p
66     int p = HASH(a[x]);
67     while(x<=n)
68     {
69         // 修改操作，对树状数组中相应的权值线段树进行修改，消除影响：v=-1，添加
        // 影响：v=1
70         // 因为树状数组中的权值线段树不需要可持久化，所以直接在原版本上修改就可以
        // 了
71         S[x] = update(S[x], 1, m, p, v);
72         x+=lowbit(x);
73     }
74 }
75
76 // 树状数组求和，求左子树包含的数字个数，和静态主席树一样的思想，都是先求左
    // 子树
77 int sum(int x)
78 {
79     int ret = 0;
80     while(x)
81     {
82         // use[i]记录的就是树状数组中相应的权值线段树的结点位置
83         // 似乎这一句有一点点难以理解，结合整体代码多看几遍吧
84         ret += tree[tree[use[x]].l].v;
85         x-=lowbit(x);
86     }
87     return ret;
88 }
89
90 // 询问操作。树状数组求[pre, ed]的和，tpre和ted是静态主席树的区间左右顶点的
    // 位置，区间[l, r]第k大
91 int Query(int pre, int ed, int tpre, int ted, int l, int r, int k)
92 {
93     if(l==r)return l;
94     int mid = (l+r)>>1;
95     // sum就是树状数组的求和，相对于静态主席树，多了sum求修改操作的影响
96     // tmp为当前左子树的数字个数
97     int tmp = sum(ed) - sum(pre) + tree[tree[tet].l].v - tree[tree[tpre].l].v;
98     // 若左子树的数字个数大于等于k就往左子树继续走，这里和主席树没什么区别
99     if(tmp >= k)
100     {
101         // 两个for循环是更新左子树需要用到的树状数组中的权值线段树的位置，有那么

```



```

    一丢丢难以理解，多看几遍？
102 for(int i=ed; i; i-=lowbit(i)) use[i] = tree[use[i]].l;
103 for(int i=pre; i; i-=lowbit(i)) use[i] = tree[use[i]].l;
104 return Query(pre, ed, tree[tpre].l, tree[tet].l, l, mid, k);
105 }
106 else
107 {
108     // 走右子树同理
109     for(int i=ed; i; i-=lowbit(i)) use[i] = tree[use[i]].r;
110     for(int i=pre; i; i-=lowbit(i)) use[i] = tree[use[i]].r;
111     return Query(pre, ed, tree[tpre].r, tree[tet].r, mid+1, r, k-tmp);
112 }
113 }
114
115 int main()
116 {
117     int t; // t个case
118     scanf("%d", &t);
119     while(t--)
120     {
121         scanf("%d%d", &n, &q);
122         m = tot = 0; // 记得初始化
123         for(int i=1; i<=n; i++)
124         {
125             scanf("%d", &a[i]);
126             b[++m] = a[i];
127         }
128         char op[5];
129         for(int i=1; i<=q; i++)
130         {
131             scanf("%s", op);
132             // 查询操作
133             if(op[0]=='Q')
134             {
135                 scanf("%d%d%d", &query[i].l, &query[i].r, &query[i].k);
136                 query[i].flag = 1;
137             }
138             else
139             {
140                 scanf("%d%d", &query[i].l, &query[i].r);
141                 b[++m] = query[i].r; // 注意要将修改后的新值加入到待离散化的b数组
142                 query[i].flag = 0;
143             }
144         }

```

```

145     // 离散化
146     sort(b+1, b+m+1);
147     m = unique(b+1, b+m+1) - b-1;
148     // 构建主席树
149     T[0] = build(1, m);
150     for(int i=1; i<=n; i++)
151         T[i] = update(T[i-1], 1, m, HASH(a[i]), 1);
152     // 构建树状数组，每一个节点都是一颗空的权值线段树
153     for(int i=1; i<=n; i++)
154         S[i] = T[0];
155     // 离线处理q个询问
156     for(int i=1; i<=q; i++)
157     {
158         if(query[i].flag) // 查询
159         {
160             // 两个for循环标记区间[l, r]要使用的树状数组中的权值线段树的位置
161             for(int j=query[i].r; j; j-=lowbit(j)) use[j] = S[j];
162             for(int j=query[i].l-1; j; j-=lowbit(j)) use[j] = S[j];
163             printf("%d\n", b[Query(query[i].l-1, query[i].r, T[query[i].l-1], T[
                query[i].r], 1, m, query[i].k)]);
164         }
165         else
166         {
167             // 先消除影响
168             add(query[i].l, -1);
169             // 在原数组中更新值
170             a[query[i].l] = query[i].r;
171             // 添加新值的影响
172             add(query[i].l, 1);
173         }
174     }
175 }
176 return 0;
177 }

```

3 DP

3.1 背包

```

1 const int MAXN = 10000;
2 const int SIZE = 100000;
3

```

```

4 int dp[SIZE];
5 int volume[MAXN], value[MAXN], c[MAXN];
6 int n, v;          // 总物品数, 背包容量
7
8 // 01背包
9 void ZeroOnepark(int val, int vol)
10 {
11     for (int j = v ; j >= vol; j--)
12     {
13         dp[j] = max(dp[j], dp[j - vol] + val);
14     }
15 }
16
17 // 完全背包
18 void Completepark(int val, int vol)
19 {
20     for (int j = vol; j <= v; j++)
21     {
22         dp[j] = max(dp[j], dp[j - vol] + val);
23     }
24 }
25
26 // 多重背包
27 void Multiplepark(int val, int vol, int amount)
28 {
29     if (vol * amount >= v)
30     {
31         Completepark(val, vol);
32     }
33     else
34     {
35         int k = 1;
36         while (k < amount)
37         {
38             ZeroOnepark(k * val, k * vol);
39             amount -= k;
40             k <<= 1;
41         }
42         if (amount > 0)
43         {
44             ZeroOnepark(amount * val, amount * vol);
45         }
46     }
47 }

```

```

48
49 int main()
50 {
51     while (cin >> n >> v)
52     {
53         for (int i = 1 ; i <= n ; i++)
54         {
55             cin >> volume[i] >> value[i] >> c[i];          // 费用, 价值, 数量
56         }
57         memset(dp, 0, sizeof(dp));
58         for (int i = 1; i <= n; i++)
59         {
60             Multiplepark(value[i], volume[i], c[i]);
61         }
62         cout << dp[v] << endl;
63     }
64     return 0;
65 }

```

3.2 数位 DP

3.2.1 HDU 2089 不要 62

```

1 // 求区间[n, m]中不含4和62的数字个数
2 // dp[pos][sta]:在第pos位中, 前一位是否为6的满足条件的数字个数。
3 // sta: 0表示前一位不为6, 1:表示前一位为6
4
5 #include<bits/stdc++.h>
6 using namespace std;
7 typedef long long ll;
8 typedef unsigned long long ull;
9 typedef pair<int, int> P;
10 const int maxn = 1e6+5;
11 const int inf = 0x3f3f3f3f;
12 const int mod = 1e9+7;
13 const double PI = acos(-1);
14
15
16 int n, m;
17 int dp[10][2];
18 int num[10];
19
20 int dfs(int pos, int pre, int sta, bool limit)

```

```

21 {
22     if(pos== -1) return 1;
23     if(!limit && dp[pos][sta] != -1) return dp[pos][sta];
24     int up = limit ? num[pos] : 9;
25     int tmp = 0;
26     for(int i=0; i<=up; i++)
27     {
28         if(pre==6 && i==2) continue;
29         if(i==4) continue;
30         tmp += dfs(pos-1, i, i==6, limit && i==num[pos]);
31     }
32     if(!limit) dp[pos][sta] = tmp;
33     return tmp;
34 }
35
36 int solve(int x)
37 {
38     int pos = 0;
39     while(x)
40     {
41         num[pos++] = x%10;
42         x/=10;
43     }
44     return dfs(pos-1, -1, 0, true);
45 }
46
47 int main()
48 {
49     while(~scanf("%d%d", &n, &m) && n && m)
50     {
51         memset(dp, -1, sizeof(dp));
52         printf("%d\n", solve(m) - solve(n-1));
53     }
54     return 0;
55 }

```

3.2.2 SCOI 2009 Windy 数

```

1 // 求区间[1, r]中不含前导零且相邻位之差的绝对值大于等于2的数的个数
2
3 #include<bits/stdc++.h>
4 using namespace std;

```

```

5 typedef long long ll;
6 typedef unsigned long long ull;
7 typedef pair<int, int> P;
8 const int maxn = 1e6+5;
9 const int inf = 0x3f3f3f3f;
10 const int mod = 1e9+7;
11 const double PI = acos(-1);
12
13
14 ll l, r;
15 ll a[20], dp[20][20];
16
17 ll dfs(int pos, int pre, int sta, bool limit)
18 {
19     if(pos== -1) return 1;
20     if(!limit && dp[pos][pre] != -1) return dp[pos][pre];
21     int up = limit ? a[pos] : 9;
22     ll ans = 0;
23     for(int i=0; i<=up; i++)
24     {
25         if(sta!=0)
26         {
27             ans += dfs(pos-1, i, sta && (i==0), limit && (i==up));
28         }
29         else
30         {
31             if(abs(pre-i)>=2)
32             {
33                 ans += dfs(pos-1, i, sta && (i==0), limit && (i==up));
34             }
35         }
36     }
37     if(!limit && pre!=0) dp[pos][pre] = ans;
38     return ans;
39 }
40
41 ll solve(int x)
42 {
43     int pos = 0;
44     while(x)
45     {
46         a[pos++] = x%10;
47         x/=10;
48     }

```

```

49 return dfs(pos-1, 0, 1, true);
50 }
51
52 int main()
53 {
54     memset(dp, -1, sizeof(dp));
55     scanf("%lld%lld", &l, &r);
56     printf("%lld\n", solve(r) - solve(l-1));
57     return 0;
58 }

```

3.3 区间 DP

3.3.1 51Nod 1021 石子合并

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef pair<int, int> P;
5 const int maxn = 1e5+5;
6 const int inf = 0x3f3f3f3f;
7 const int mod = 1e9+7;
8 const double PI = acos(-1);
9
10 int n;
11 int num[105];
12 int sum[105];
13 int dp[105][105];
14 int main()
15 {
16     memset(dp, inf, sizeof(dp));
17     cin>>n;
18     for(int i=1;i<=n;i++)
19     {
20         cin>>num[i];
21         sum[i] = sum[i-1]+num[i];
22         dp[i][i] = 0;
23     }
24
25     for(int len=1;len<=n;len++)
26     {
27         for(int i=1;i<=n-len+1;i++)
28         {

```

```

29             int end = i+len-1;
30             for(int j=i;j<end;j++)
31             {
32                 dp[i][end] = min(dp[i][end], dp[i][j] + dp[j+1][end]+sum[end]-sum[i-1]);
33             }
34         }
35     }
36     cout<<dp[1][n]<<endl;
37     return 0;
38 }

```

3.4 最长公共递增子序列

```

1 /*
2 * 最长公共递增子序列 O(n^2)
3 * f记录路径,DP记录长度,用a对b扫描,逐步最优化。
4 */
5 const int N = 1010;
6
7 int f[N][N], dp[N];
8
9 int gcis(int a[], int la, int b[], int lb, int ans[])
10 { // a[1...la], b[1...lb]
11     int i, j, k, mx;
12     memset(f, 0, sizeof(f));
13     memset(dp, 0, sizeof(dp));
14     for (i = 1; i <= la; i++)
15     {
16         memcpy(f[i], f[i-1], sizeof(f[0]));
17         for (k = 0, j = 1; j <= lb; j++)
18         {
19             if (b[j-1] < a[i-1] && dp[j] > dp[k])
20             {
21                 k = j;
22             }
23             if (b[j-1] == a[i-1] && dp[k] + 1 > dp[j])
24             {
25                 dp[j] = dp[k] + 1,
26                 f[i][j] = i * (lb + 1) + k;
27             }
28         }

```

```

29 }
30 for (mx = 0, i = 1; i <= lb; i++)
31 {
32     if (dp[i] > dp[mx])
33     {
34         mx = i;
35     }
36 }
37 for (i = la * lb + la + mx, j = dp[mx]; j; i = f[i / (lb + 1)][i % (lb +
38     1)], j--)
39 {
40     ans[j - 1] = b[i % (lb + 1) - 1];
41 }
42 return dp[mx];
43 }

```

3.5 最长公共子序列

```

1 const int N = 1010;
2
3 int a[N][N];
4
5 int LCS(const char *s1, const char *s2)
6 { // s1:0...m, s2:0...n
7     int m = (int)strlen(s1), n = (int)strlen(s2);
8     int i, j;
9     a[0][0] = 0;
10    for (i = 1; i <= m; ++i)
11    {
12        a[i][0] = 0;
13    }
14    for (i = 1; i <= n; ++i)
15    {
16        a[0][i] = 0;
17    }
18    for (i = 1; i <= m; ++i)
19    {
20        for (j = 1; j <= n; ++j)
21        {
22            if (s1[i - 1] == s2[j - 1])
23            {
24                a[i][j] = a[i - 1][j - 1] + 1;

```

```

25     }
26     else if (a[i - 1][j] > a[i][j - 1])
27     {
28         a[i][j] = a[i - 1][j];
29     }
30     else
31     {
32         a[i][j] = a[i][j - 1];
33     }
34 }
35 }
36 return a[m][n];
37 }

```

3.6 最长有序子序列

```

1 int n;
2 int a[maxn]; // 原数组
3 int s[maxn]; // 记录LIS
4 int LIS()
5 {
6     int top = 0;
7     for(int i=0;i<n;i++)
8     {
9         int pos = upper_bound(s, s+top, a[i])-s;
10        s[pos] = a[i];
11        top = max(top, pos+1);
12    }
13    return top;
14 }

```

```

1 /*
2 * 递增（默认）
3 * 递减
4 * 非递增
5 * 非递减 (1)>= && < (2)< (3)>=
6 */
7 const int MAXN = 1001;
8
9 int a[MAXN], f[MAXN], d[MAXN]; // d[i] 用于记录 a[0...i] 以 a[i] 结尾的最
10 大长度

```

```

11 int bsearch(const int *f, int size, const int &a)
12 {
13     int l = 0, r = size - 1;
14     while (l <= r)
15     {
16         int mid = (l + r) / 2;
17         if (a > f[mid - 1] && a <= f[mid]) // (1)
18         {
19             return mid;
20         }
21         else if (a < f[mid])
22         {
23             r = mid - 1;
24         }
25         else
26         {
27             l = mid + 1;
28         }
29     }
30     return -1;
31 }
32
33 int LIS(const int *a, const int &n)
34 {
35     int i, j, size = 1;
36     f[0] = a[0];
37     d[0] = 1;
38     for (i = 1; i < n; ++i)
39     {
40         if (a[i] <= f[0]) // (2)
41         {
42             j = 0;
43         }
44         else if (a[i] > f[size - 1]) // (3)
45         {
46             j = size++;
47         }
48         else
49         {
50             j = bsearch(f, size, a[i]);
51         }
52         f[j] = a[i];
53         d[i] = j + 1;
54     }

```

```

55     return size;
56 }
57
58 int main()
59 {
60     int i, n;
61     while (scanf("%d", &n) != EOF)
62     {
63         for (i = 0; i < n; ++i)
64         {
65             scanf("%d", &a[i]);
66         }
67         printf("%d\n", LIS(a, n)); // 求最大递增 / 上升子序列(如果为最大非
68                                     降子序列,只需把上面的注释部分给与替换)
69     }
69     return 0;
70 }

```

4 字符串

4.1 KMP

```

1 // 能够获得t在s中出现的所有位置
2
3 #include<bits/stdc++.h>
4 using namespace std;
5 typedef long long ll;
6 typedef unsigned long long ull;
7 typedef pair<int, int> P;
8 const int maxn = 1e6+5;
9 const int inf = 0x3f3f3f3f;
10 const int mod = 1e9+7;
11 const double PI = acos(-1);
12
13
14 char s[maxn], t[maxn];
15 int ne[maxn];
16 int ls, lt;
17
18 void init()
19 {
20     scanf("%s", s+1);

```

```

21 scanf("%s", t+1);
22 ls = strlen(s+1);
23 lt = strlen(t+1);
24 }
25
26 void get_ne()
27 {
28     int j = 0;
29     for(int i=2;i<=lt;i++)
30     {
31         while(j && t[i]!=t[j+1])
32             j = ne[j];
33         if(t[i]==t[j+1])j++;
34         ne[i] = j;
35     }
36 }
37
38 void kmp()
39 {
40     int j=0;
41     vector<int> v;
42     for(int i=1;i<=ls;i++)
43     {
44         while(j>0 && s[i]!=t[j+1])j = ne[j];
45         if(s[i]==t[j+1])j++;
46         if(j==lt)
47         {
48             v.push_back(i-lt+1);
49             j = ne[j];
50         }
51     }
52     for(auto x : v)printf("%d\n", x);
53 }
54
55 int main()
56 {
57     init();
58     get_ne();
59     kmp();
60     return 0;
61 }

```

4.2 EXKMP

```

1 // 求t与s的所有后缀的最大前缀长度， 存于ex数组
2 #include<bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 typedef unsigned long long ull;
6 typedef pair<int, int> P;
7 const int maxn = 1e6+5;
8 const int inf = 0x3f3f3f3f;
9 const int mod = 1e9+7;
10 const double PI = acos(-1);
11
12
13 char s[maxn], t[maxn];
14 int ne[maxn], ex[maxn];
15
16 void get_ne(char *s)
17 {
18     int i=0, j, po, len=strlen(s);
19     ne[0] = len;
20     while(s[i]==s[i+1] && i+1<len)i++;
21     ne[1] = i;
22     po = 1;
23     for(int i=2;i<len;i++)
24     {
25         if(ne[i-po]+i < ne[po]+po)
26             ne[i] = ne[i-po];
27         else
28         {
29             j = ne[po]+po-i;
30             if(j<0)j=0;
31             while(i+j < len && s[j] == s[j+i])j++;
32             ne[i] = j;
33             po = i;
34         }
35     }
36 }
37
38 void exkmp(char *s, char *t)
39 {
40     int i=0, j, po, ls=strlen(s), lt=strlen(t);
41     get_ne(t);
42     while(s[i]==t[i] && i<lt && i<ls)i++;

```

```

43 ex[0] = i;
44 po = 0;
45 for(i=1;i<ls;i++)
46 {
47     if(ne[i-po]+i < ex[po]+po)
48         ex[i] = ne[i-po];
49     else
50     {
51         j = ex[po] + po - i;
52         if(j<0) j=0;
53         while(i+j<ls && j<lt && s[j+i]==t[j])j++;
54         ex[i] = j;
55         po = i;
56     }
57 }
58 }
59
60 int main()
61 {
62     scanf("%s", s);
63     scanf("%s", t);
64     exkmp(s, t);
65     for(int i=0;i<strlen(s);i++)
66     {
67         if(i==0)printf("%d", ex[i]);
68         else printf(" %d", ex[i]);
69     }
70     printf("\n");
71     return 0;
72 }

```

4.3 字符串 Hash

4.3.1 自然溢出

```

1 typedef long long ll;
2 typedef unsigned long long ull;
3 #define maxn 1005
4 struct My_Hash
5 {
6     ull base=131;
7     ull p[maxn],ha[maxn];
8     void Insert(char s[])

```

```

9 {
10     int len=strlen(s+1);
11     p[0]=1,ha[0]=0;
12     for(int i=1;i<=len;i++)
13     {
14         p[i]=p[i-1]*base;
15         ha[i]=ha[i-1]*base+(ull)s[i];
16     }
17 }
18
19 ull gethash(int l,int r)
20 {
21     return (ull)ha[r]-ha[l-1]*p[r-l+1];
22 }
23 }S;
24 int main()
25 {
26     char s[maxn];
27     cin>>s+1;
28     S.Insert(s);
29     int T;
30     cin>>T;
31     while(T--)
32     {
33         int l,r;
34         cin>>l>>r;
35         cout<<S.gethash(l,r)<<endl;
36     }
37     return 0;
38 }

```

5 数学

5.1 欧拉函数

定义：欧拉函数 $\phi(x)$ 代表小于等于 x 的数中和 x 互质的数的个数（小于显然只对 1 成立），比如说小于等于 9 的数中与 9 互质的有 1,2,4,5,7,8，则 $\phi(9)=6$ 。以 $\phi(x)$ 表示小于等于 x 的数中与 x 互质的数的个数可以这样得到： $\phi(x) = x * (1 - 1/p_1) * (1 - 1/p_2) * (1 - 1/p_3) \cdots (1 - 1/p_n)$ 其中 $p_1, p_2, p_3 \cdots p_n$ 是 x 的所有质因数，每个质因数只使用一次，使用上述公式，则 $\phi(9)=9*(1-1/3)=6$ 。

特性：1. 若 a 为质数， $\phi[a] = a - 1$ ；

2. 若 a 为质数, $b\%a = 0, \phi[a * b] = \phi[b] * a$
3. 若 a, b 互质, $\phi[a * b] = \phi[a] * \phi[b]$ (当 a 为质数时, $\phi[b\%a] = 0, \phi[a * b] = \phi[a] * \phi[b]$)
4. 若 p 为质数, $\phi(p^k) = p^k - p^{k-1} = (p-1) * p^{k-1}$
5. 当 $n > 2$ 时, $\phi(n)$ 是偶数
6. $\phi(1) = 1$; (具体情况看题目, 有时题目会要求为 0)

欧拉筛法: 欧拉筛法的原理如下, 观察 $\phi(x)$ 的求解式子, 首先令 $\phi(x) = x$, 然后仿照素数筛法, 如果 x 能被 $2, 3, 5, 7 \dots$ 这些素数筛到 (即素数 $prime$ 是 x 的因子), 则执行 $\phi(x) = \phi(x) * (1 - 1/prime)$

```

1 // 埃拉托斯特尼筛求欧拉函数值
2 const int maxn = 1000000;
3 int phi[maxn+1];
4 bool isPrime[maxn+1];
5 void Euler()
6 {
7     for(int i=1; i<=maxn; i++) phi[i]=i;
8     memset(isPrime, true, sizeof(isPrime));
9     isPrime[0]=isPrime[1]=false;
10    phi[1]=1;
11    for(int i=2; i<=maxn; i++)
12    {
13        // i是质数
14        if(isPrime[i])
15        {
16            for(int j=i; j<=maxn; j+=i)
17            {
18                // i与i的倍数都更新一遍
19                isPrime[j]=false;
20                phi[j] -= phi[j]/i; // phi(x)=phi(x)*(1-1/prime) = phi(x) - phi(x)/prime
21            }
22        }
23    }
24 }
25
26
27 // 欧拉筛求欧拉函数, 时间复杂度接近O(n)
28 const int maxn = 1100000;
29 int num = 0;

```

```

30 int phi[maxn+1];
31 int prime[maxn+1];
32 int flag[maxn+1];
33
34 void euler()
35 {
36     memset(flag, 0, sizeof(flag));
37     phi[1]=1; // 1要特判
38     for (int i=2; i<=maxn; i++)
39     {
40         if (flag[i]==0) // 这代表i是质数
41         {
42             prime[++num]=i;
43             phi[i]=i-1;
44         }
45         for (int j=1; j<=num && prime[j]*i<=maxn; j++) // 经典的欧拉筛写法
46         {
47             flag[i*prime[j]]=1; // 先把这个合数标记掉
48             if(i%prime[j]==0)
49             {
50                 phi[i*prime[j]]=phi[i]*prime[j]; // 若prime[j]是i的质因子, 则根据计算公式, i已经包括i*prime[j]的所有质因子
51                 break; // 经典欧拉筛的核心语句, 这样能保证每个数只会被自己最小的因子筛掉一次
52             }
53             else phi[i*prime[j]]=phi[i]*phi[prime[j]]; // 利用了欧拉函数是个积性函数的性质
54         }
55     }
56 }

```

5.2 枚举约数

```

1 // 枚举约数(普通)
2 vector<int> factor(int n)
3 {
4     vector<int> a;
5     for(int i = 2; i*i <= n; i++){
6         if((n%i)==0){
7             a.push_back(i);
8             if((n/i)!=i)a.push_back(n/i); // 根号n的情况不要重复添加
9         }

```

```

10 }
11 return a;
12 }
13
14 特殊例题: f(n)为n所有约数的和, 给你一个数n, 让你求从1到n中f(n)为偶数的数有多少个
15 #include <bits/stdc++.h>
16 #define ll long long
17 #define INF 0x3f3f3f3f
18
19 using namespace std;
20
21 int main()
22 {
23     int t, cas = 1;
24     long long n, sum;
25     cin >> t;
26     while(t--)
27     {
28         scanf("%lld", &n);
29         sum = n;
30         sum -= (int)sqrt(n);
31         sum -= (int)sqrt(n/2);
32         printf("Case %d: %lld\n", cas++, sum);
33     }
34     return 0;
35 }

```

5.3 全错排

```

1 问题: n封信给n个人, 问全都送错的可能性有多少种
2 #include<iostream>
3 #include<stdio.h>
4 using namespace std;
5 int main()
6 {
7     long long int arr[21];
8     int num,i;
9     arr[1]=0; arr[2]=1;
10    for(i=3;i<21;i++)
11        arr[i]=(i-1)*(arr[i-1]+arr[i-2]);    // d[n]= (n-1)*( d[n-1] + d[n-2])
12    while(scanf("%d",&num)!=EOF)

```

```

13 {
14     cout<<arr[num]<<endl;
15 }
16 return 0;
17 }

```

5.4 唯一分解定理

算术基本定理可表述为: 任何一个大于 1 的自然数 N , 如果 N 不为质数, 那么 N 可以唯一分解成有限个质数的乘积 $N = P_1^{a_1} * P_2^{a_2} * P_3^{a_3} \cdots P_n^{a_n}$, 这里 $P_1 < P_2 < P_3 \cdots < P_n$ 均为质数, 其中指数 a_i 是正整数。

(1) 一个大于 1 的正整数 N , 如果它的标准分解式为: $N = P_1^{a_1} P_2^{a_2} \cdots P_n^{a_n}$, 那么它的正因数个数为 $\sigma_0(N) = (1 + a_1)(1 + a_2) \cdots (1 + a_n)$

(2) 它的全体正因数之和为

$$\sigma_1(N) = (1 + p_1 + p_1^2 + \cdots + p_1^{a_1})(1 + p_2 + p_2^2 + \cdots + p_2^{a_2}) \cdots (1 + p_n + p_n^2 + \cdots + p_n^{a_n})$$

当 $\sigma_1(N) = 2N$ 时就称 N 为完全数

5.4.1 例题一

求出有几种边长为整数, 面积等于 a 的矩形, 且矩形的短边不小于 b

```

1 //考点: 素数筛选 + 唯一分解定理
2 #include <bits/stdc++.h>
3 #define ll long long
4 #define INF 0x3f3f3f3f
5
6 using namespace std;
7
8 const int MAX = 1e6 + 10; // 边长
9 int prime[MAX], k; // k 为全局变量, 指向最后一个素数的后一个单元
10 bool isPrime[MAX];
11
12 // 素数筛选
13 void Prime()
14 {
15     k = 0;
16     memset(isPrime, true, sizeof(isPrime));
17     isPrime[1] = false; // 1不是素数
18     for(int i = 2; i < MAX; i++)

```

```

19 {
20     if(isPrime[i])
21     {
22         prime[k++] = i;
23         for(int j = 2 ; i * j < MAX ; j++)
24         {
25             isPrime[i * j] = false;
26         }
27     }
28 }
29 }
30
31 // 唯一分解定理找n的因子个数(N的因子个数 M = (1 + a1)*(1 + a2)*(1 + a3)
   ...*(1 + an))
32 long long solve(long long n)
33 {
34     long long ans = 0, sum = 1;
35     for(long long i = 0; i < k && prime[i]*prime[i] <= n; i++)
36     {
37         if(n % prime[i] == 0) // 从小到大寻找素数因子
38         {
39             ans = 0;
40             while(n % prime[i] == 0) // 计算素数因子次数
41             {
42                 ans++;
43                 n /= prime[i];
44             }
45             sum *= (1 + ans);
46         }
47     }
48     if(n > 1) sum *= 2; // 如果处理到最后n还不是1, 则最后剩下的一定是个素数
49     return sum;
50 }
51
52 int main()
53 {
54     Prime();
55     int t, x = 0;
56     long long ab, a, num;
57     scanf("%d", &t);
58     while(t--)
59     {
60         x++;
61         scanf("%lld%lld", &ab, &a);

```

```

62     if(ab < a * a)
63     {
64         printf("Case %d: 0\n", x);
65         continue;
66     }
67     num = solve(ab); // 获得因子数(一个数的因子包括他本身)
68     num /= 2;        // 因子对数
69     for(long long i = 1 ; i < a ; i++)
70     {
71         if(ab % i == 0) num--; //将边小于a的情况减去
72     }
73     printf("Case %d: %lld\n", x, num);
74 }
75 return 0;
76 }

```

5.4.2 例题二

给你一个数 $x = b^p$, 求 p 的最大值

$$x = p_1^{x_1} * P_2^{x_2} * \dots * P_s^{x_s}$$

题目要求 $x = b^p$, x 只有一个因子的 p 次幂构成如果 $x = 12 = 2^2 * 3^1$, 要让 $x = b^p$, 即 12 应该是 $12 = 12^1$ 所以 $p = \gcd(x_1, x_2, x_3, \dots, x_s)$; 比如: $24 = 2^3 * 3^1$, p 应该是 $\gcd(3, 1)=1$, 即 $24 = 24^1$ $324 = 3^4 * 2^2$, p 应该是 $\gcd(4, 2)=2$, 即 $324 = 18^2$ 本题有一个坑, 就是 x 可能为负数, 如果 x 为负数的话, $x = b^q$, q 必须是奇数, 所以将 x 转化为正数求得的解如果是偶数的话必须将其循环除 2 直到出现奇数

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int maxn=1e6+5; // 卡内存
5 bool isprime[maxn];
6 int prime[maxn], psize=0;
7
8 // 素数筛选
9 void getPrimes()
10 {
11     memset(isprime, true, sizeof(isprime));
12     isprime[0] = isprime[1] = false;
13     for (int i = 2; i < maxn; i++) if (isprime[i])
14     {

```

```

15     prime[psize++] = i;
16     for(int j = 2; i * j < maxn; j++)
17         isprime[i*j] = false;
18 }
19 }
20
21 // 辗转相除
22 int gcd(int a, int b)
23 {
24     return b == 0 ? a : gcd(b, a%b);
25 }
26
27 int main()
28 {
29     int t, cas = 1;
30     long long n;
31     getPrimes();
32     scanf("%d", &t);
33     while (t--)
34     {
35         scanf("%lld", &n);
36         int f = 0;
37         if(n < 0)
38         {
39             n = - n;
40             f = 1;
41         }
42         int res, ans = 0;
43         for(int i = 0 ; i < psize && prime[i] * prime[i] <= n ; i++)
44         {
45             if(n % prime[i] == 0)
46             {
47                 res = 0;
48                 while(n % prime[i] == 0)
49                 {
50                     res++;
51                     n /= prime[i];
52                 }
53                 if(ans == 0)
54                     ans = res;
55                 else
56                     ans = gcd(ans, res);
57             }
58         }

```

```

59         if(n > 1) ans = gcd(ans, 1); // 如果分解完后剩下的n大于1，则剩下的一定是
           一个素数
60         if(f == 1)
61         {
62             while(ans % 2 == 0)
63                 ans /= 2; // 如果n为负数且最后算出来指数为偶数，则循环除2直到变为奇
           数
64         }
65         printf("Case %d: %d\n", cas++, ans);
66     }
67     return 0;
68 }

```

5.5 快速幂

```

1 typedef long long ll;
2 ll pow(ll x,ll n,ll mod)
3 {
4     ll res=1;
5     while(n>0)
6     {
7         if(n&1)
8         {
9             res=res*x;
10            res=res%mod;
11        }
12        x=x*x;
13        x=x%mod;
14        n>>=1;
15    }
16    return res;
17 }
18
19 例题：给你一个n，一个k，求n^k的结果的前三位与后三位
20 //考点：快速幂取模(可获得后三位) + 数论运算(取前三位)
21
22 /**
23 * 设10^w=n^k，两边同时取log10，那么w=k*log10(n)。
24 * 再设x=(int)w (w的整数部分)，y=w-x (w的小数部分)，那么10^w=10^(x+y)=10^x
   * 10^y=n^k;
25 * 由于10^x是10的倍数，那么10^y=n^k/10^x，因为0<y<1,所以10^y为一位数，即最前
   边那一位，所以求出y来

```

```

26 */
27
28 #include <bits/stdc++.h>
29 #define ll long long
30 #define INF 0x3f3f3f3f
31
32 using namespace std;
33
34 // 快速幂取模
35 ll pow_mod(ll a, ll b, ll mod)
36 {
37     int ans = 1;
38     int base = a % mod;
39     while(b){
40         if(b & 1) ans = (ans*base) % mod;
41         base = (base*base) % mod;
42         b >>= 1;
43     }
44     return ans;
45 }
46
47 int main()
48 {
49     ll n, k;
50     ll ans1, ans2;
51     int t, cas = 1;
52     scanf("%d", &t);
53     while(t--)
54     {
55         scanf("%lld%lld", &n, &k);
56         ans2 = pow_mod(n, k, 1000); // 取后三位
57         double w = k*log10(n);
58         w = w-(ll)w;
59         ll ans1=(ll)(pow(10,w)*100); // 取前三位
60         printf("Case %d: %lld %03lld\n", cas++, ans1, ans2); // 后三位要补前导
           零
61     }
62     return 0;
63 }

```

5.6 欧拉降幂和广义欧拉降幂

5.7 逆序数

```

1 /*
2 * 也可以用树状数组做
3 * a[0...n-1] cnt=0; call: MergeSort(0, n)
4 */
5 const int N = 1010;
6 int a[N];
7 int c[N];
8 int cnt = 0;
9
10 void MergeSort(int l, int r)
11 {
12     int mid, i, j, tmp;
13     if (r > l + 1)
14     {
15         mid = (l + r) / 2;
16         MergeSort(l, mid);
17         MergeSort(mid, r);
18         tmp = l;
19         for (i = l, j = mid; i < mid && j < r;)
20         {
21             if (a[i] > a[j])
22             {
23                 c[tmp++] = a[j++];
24                 cnt += mid - i;
25             }
26             else
27             {
28                 c[tmp++] = a[i++];
29             }
30         }
31         if (j < r)
32         {
33             for (; j < r; ++j)
34             {
35                 c[tmp++] = a[j];
36             }
37         }
38         else
39         {
40             for (; i < mid; ++i)
41             {
42                 c[tmp++] = a[i];

```

```

43     }
44 }
45 for (i = 1; i < r; ++i)
46 {
47     a[i] = c[i];
48 }
49 }
50 return ;
51 }

```

5.8 无序序列变有序的最少交换次数

5.8.1 相邻元素交换

```
1 \\ 等于逆序数个数
```

5.8.2 任意元素交换

```

1 *
2 * 交换任意两数的本质是改变了元素位置，
3 * 故建立元素与其目标状态应放置位置的映射关系
4 */
5 int getMinSwaps(vector<int> &A)
6 {
7     // 排序
8     vector<int> B(A);
9     sort(B.begin(), B.end());
10    map<int, int> m;
11    int len = (int)A.size();
12    for (int i = 0; i < len; i++)
13    {
14        m[B[i]] = i;    // 建立每个元素与其应放位置的映射关系
15    }
16
17    int loops = 0;    // 循环节个数
18    vector<bool> flag(len, false);
19    // 找出循环节的个数
20    for (int i = 0; i < len; i++)
21    {
22        if (!flag[i])
23        {

```

```

24        int j = i;
25        while (!flag[j])
26        {
27            flag[j] = true;
28            j = m[A[j]];    // 原序列中j位置的元素在有序序列中的位置
29        }
30        loops++;
31    }
32 }
33 return len - loops;
34 }

```

5.9 GCD

5.9.1 非递归

```

1 int gcd(int x, int y)
2 {
3     if (!x || !y)
4     {
5         return x > y ? x : y;
6     }
7
8     for (int t; t = x % y, x = y, y = t);
9
10    return y;
11 }

```

5.9.2 递归

```

1 int gcd(int a, int b)
2 {
3     return b==0 ? a : gcd(b, a%b);
4 }

```

5.10 EXGCD

欧几里得是用来求 a, b 的最大公约数，那么扩展欧几里得不仅能求出 a, b 的最大公约数，还能求出满足 $ax+by=\text{gcd}(a,b)$ 的一组可行解。

```

1  /*
2  * 求x, y使得gcd(a, b) = a * x + b * y;
3  */
4  int extgcd(int a, int b, int &x, int &y)
5  {
6      if (b == 0)
7      {
8          x = 1;
9          y = 0;
10         return a;
11     }
12     int d = extgcd(b, a % b, x, y);
13     int t = x;
14     x = y;
15     y = t - a / b * y;
16     return d;
17 }

```

5.10.1 例题

例题：给出 $a, b, c, x_1, x_2, y_1, y_2$ ，求满足 $ax+by+c=0$ ，且 $x \in [x_1, x_2], y \in [y_1, y_2]$ 的整数解个数

先处理无解情况：

1、当 $a=0$ 并且 $b=0$ ，而 $c \neq 0$ 时，显然无解；当 $a=0, b=0$ ，而 $c=0$ 时， $[x_1, x_2], [y_1, y_2]$ 都为可行解，根据乘法原理，可行解的个数为 $(x_2-x_1+1)*(y_2-y_1+1)$ ；

2、当 $a=0, b \neq 0$ 时：此时即为求解 $by=c$ ，则 $y=c/b$ ，如果 c/b 不是整数或 c/b 不在 $[y_1, y_2]$ 的范围内，无解否则 $[x_1, x_2]$ 内全部整数都为可行解。

3、当 $b=0, a \neq 0$ 时，同上。

4、若 c 不是 $\gcd(a, b)$ 的倍数，方程显然无解。

```

1 #include <iostream>
2 #include <cmath>
3 #include <cstdio>
4 typedef ll longlong
5 using namespace std;
6
7 ll a, b, c, x1, x2, y1, y2, x0, y0; // 全局变量
8 inline ll cmin(const ll &x, const ll &y) {return x < y ? x : y;}

```

```

9 inline ll cmax(const ll &x, const ll &y) {return x > y ? x : y;}
10
11 ll gcd(ll a, ll b)
12 {
13     if (b == 0) return a;
14     return gcd(b, a % b);
15 }
16
17 void exgcd(ll a, ll b)
18 {
19     if (b == 0) {x0 = 1; y0 = 0; return;}
20     exgcd(b, a % b);
21     ll t = x0; x0 = y0; y0 = t - a / b * y0;
22     return;
23 }
24
25 int main()
26 {
27     cin >> a >> b >> c;
28     cin >> x1 >> x2;
29     cin >> y1 >> y2;
30     c = -c;
31     if (c < 0) {a = -a; b = -b; c = -c;}
32     if (a < 0) {a = -a; ll t = x1; x1 = -x2; x2 = -t;}
33     if (b < 0) {b = -b; ll t = y1; y1 = -y2; y2 = -t;}
34     if (a == 0 && b == 0)
35     {
36         if (c == 0)
37         {
38             cout << (x2 - x1 + 1) * (y2 - y1 + 1) << endl;
39             return 0;
40         }
41         cout << "0" << endl; return 0;
42     }
43     else if (a == 0)
44     {
45         if (c % b == 0)
46             if (c / b <= y2 && c / b >= y1) {cout << x2 - x1 + 1 << endl; return 0;}
47         cout << "0" << endl; return 0;
48     }
49     else if (b == 0)
50     {
51         if (c % a == 0)
52             if (c / a <= x2 && c / a >= x1) {cout << y2 - y1 + 1 << endl; return 0;}

```

```

53 cout << "0" << endl; return 0;
54 }
55
56 ll d=gcd(a,b);
57 if (c%d!=0){cout << "0" << endl; return 0;}
58
59 a=a/d;b=b/d;c=c/d;
60 exgcd(a,b);
61 x0=x0*c;yy0=yy0*c;
62
63 double tx2=x2,tx1=x1,tx0=x0,ta=a,tb=b,tc=c,ty1=yy1,ty2=y2,ty0=yy0;
64 ll down1=floor(((tx2-tx0)/tb)),down2=floor(((ty0-ty1)/ta));
65 ll r=cmin(down1,down2);
66 ll up1=ceil(((tx1-tx0)/tb)),up2=ceil(((ty0-ty2)/ta));
67 ll l=cmax(up1,up2);
68 if (r<l) cout << "0" << endl;
69 else cout << r-l+1 << endl;
70 return 0;
71 }

```

5.11 逆元

对于正整数 a 和 m , 如果有 $ax \equiv 1 \pmod{m}$, 那么把这个同余方程中 x 的最小正整数解叫做 a 模 m 的逆元。

5.11.1 拓展欧几里得法

```

1 // 扩展欧几里得(求a对于mod的逆元, 要求a与mod互素)
2 ll exgcd(ll a, ll b, ll &x, ll &y)
3 {
4     //求2对于1e9+7的逆元就是 exgcd(2, 1e9+7, x, y),其中x的值就是inv2,
5     if (b == 0)
6     {
7         x = 1;
8         y = 0;
9         return a;
10    }
11    ll r = exgcd(b, a % b, x, y);
12    ll t = x % mod;
13    x = y % mod;

```

```

14 y = ((t - a / b * y) % mod + mod) % mod;
15 return r;
16 }

```

5.11.2 费马小定理法

```

1 // 费马小定理(求a对于mod的逆元, 要求mod为素数)
2 ll power_mod(ll a, ll b, ll mod)
3 {
4     ll ans = 1;
5     while (b)
6     {
7         if (b & 1) ans = ans * a % mod;
8         a = a * a % mod;
9         b >>= 1;
10    }
11    return ans;
12 }
13 inv2 = power_mod(a, mod - 2, mod);

```

5.12 中国剩余定理

中国剩余定理给出了以下的一元线性同余方程:

$$(S): \begin{cases} x \equiv a_1 & (\text{mod } m_1) \\ x \equiv a_2 & (\text{mod } m_2) \\ \vdots \\ x \equiv a_n & (\text{mod } m_n) \end{cases}$$

中国剩余定理说明: 假设整数 m_1, m_2, \dots, m_n 两两互质, 则对任意的整数: a_1, a_2, \dots, a_n 方程组 (S) 有解, 并且通解可以通过如下方式构造得到:

```

1 //n个方程: x=a[i](mod m[i]) (0<=i<n)
2 ll china(int n, ll *a, ll *m){
3     ll M = 1, ret = 0;
4     for(int i = 0; i < n; i++) M *= m[i];
5     for(int i = 0; i < n; i++){
6         ll w = M / m[i];
7         ret = (ret + w * inv(w, m[i]) * a[i]) % M;
8     }

```



```

9   return (ret + M) % M;
10  }

```

5.12.1 例题

人自出生起就有体力, 情感和智力三个生理周期, 分别为 23, 28 和 33 天。一个周期内有一天为峰值, 在这一天, 人在对应的方面 (体力, 情感或智力) 表现最好。通常这三个周期的峰值不会是同一天。现在给出三个日期, 分别对应于体力, 情感, 智力出现峰值的日期。然后再给出一个起始日期, 要求从这一天开始, 算出最少再过多少天后三个峰值同时出现。

分析: 假设 x 为这个天数, n_1 为第一个周期出现的日期, p_1 为其周期, 以此类推, 则如果同时出现峰值, 则:

$$\begin{aligned}
 x &= n_1 + k_1 * p_1 \\
 x &= n_2 + k_2 * p_2 \\
 x &= n_3 + k_3 * p_3
 \end{aligned}$$

两侧同时取余 p 有:

$$\begin{aligned}
 x \% p_1 &= n_1 \% p_1 \text{ 即 } x \% p_1 = a_1 \\
 x \% p_2 &= n_2 \% p_2 \text{ 即 } x \% p_2 = a_2 \\
 x \% p_3 &= n_3 \% p_3 \text{ 即 } x \% p_3 = a_3
 \end{aligned}$$

$p_1 = 23, p_2 = 27, p_3 = 33$ 两两互质

```

1  #include<cstdio>
2  typedef long long ll;
3  const int N = 100000 + 5;
4  void ex_gcd(ll a, ll b, ll &x, ll &y, ll &d){
5      if (!b) {d = a, x = 1, y = 0;}
6      else{
7          ex_gcd(b, a % b, y, x, d);
8          y -= x * (a / b);
9      }
10 }
11
12 ll inv(ll t, ll p){//如果不存在, 返回-1
13     ll d, x, y;
14     ex_gcd(t, p, x, y, d);
15     return d == 1 ? (x % p + p) % p : -1;
16 }
17

```

```

18 ll china(int n, ll *a, ll *m){//中国剩余定理
19     ll M = 1, ret = 0;
20     for(int i = 0; i < n; i++) M *= m[i];
21     for(int i = 0; i < n; i++){
22         ll w = M / m[i];
23         ret = (ret + w * inv(w, m[i]) * a[i]) % M;
24     }
25     return (ret + M) % M;
26 }
27
28 int main(){
29     ll p[3], r[3], d, ans, MOD = 21252;
30     int cas = 0;
31     p[0] = 23; p[1] = 28; p[2] = 33;
32     while(~scanf("%lld%lld%lld%lld", &r[0], &r[1], &r[2], &d) && (~r[0] || ~r[1] || ~r[2] || ~d)){
33         ans = ((china(3, r, p) - d) % MOD + MOD) % MOD;
34         printf("Case %d: the next triple peak occurs in %I64d days.\n", ++cas, ans ? ans : 21252);
35     }
36     return 0;
37 }

```

5.13 拓展中国剩余定理

```

1  //m1,m2, ... ,mn两两不保证互质
2  #include<cstdio>
3  #include<algorithm>
4  using namespace std;
5  typedef long long ll;
6  typedef pair<ll, ll> PLL;
7  PLL linear(ll A[], ll B[], ll M[], int n) {//求解A[i]x = B[i] (mod M[i]),总
    共n个线性方程组
8      ll x = 0, m = 1;
9      for(int i = 0; i < n; i++) {
10         ll a = A[i] * m, b = B[i] - A[i]*x, d = gcd(M[i], a);
11         if(b % d != 0) return PLL(0, -1);//答案不存在, 返回-1
12         ll t = b/d * inv(a/d, M[i]/d)% (M[i]/d);
13         x = x + m*t;
14         m *= M[i]/d;
15     }
16     x = (x % m + m) % m;

```

```

17 return PLL(x, m); //返回的x就是答案, m是最后的lcm值
18 }

```

5.14 素数

5.14.1 判断小于 MAXN 的数是否为素数

```

1  /*
2  * 素数筛选, 判断小于MAXN的数是不是素数
3  *  notprime是一张表, false表示是素数, true表示不是
4  */
5  const int MAXN = 1000010;
6  bool notprime[MAXN];
7
8  void init()
9  {
10     memset(notprime, false, sizeof(notprime));
11     notprime[0] = notprime[1] = true;
12     for (int i = 2; i < MAXN; i++)
13     {
14         if (!notprime[i])
15         {
16             if (i > MAXN / i) // 阻止后边i * i溢出 (或者i,j用long long)
17             {
18                 continue;
19             }
20             // 直接从i * i开始就可以, 小于i倍的已经筛选过了
21             for (int j = i * i; j < MAXN; j += i)
22             {
23                 notprime[j] = true;
24             }
25         }
26     }
27 }

```

5.14.2 埃式筛法

```

1  const int MAX = 1e6 + 10;
2  int prime[MAX], k; // k为全局变量, 指向最后一个素数的后一个单元
3  bool isPrime[MAX];
4  void Prime()

```

```

5  {
6      k = 0;
7      memset(isPrime, true, sizeof(isPrime));
8      isPrime[1] = false; // 1不是素数
9      for (int i = 2; i < MAX; i++)
10     {
11         if (isPrime[i])
12         {
13             prime[k++] = i;
14             for (int j = 2; i * j < MAX; j++)
15             {
16                 isPrime[i * j] = false;
17             }
18         }
19     }
20 }

```

5.14.3 线性筛法

```

1  /*
2  * 素数筛选, 查找出小于等于MAXN的素数
3  *  prime[0]存素数的个数
4  */
5
6  const int MAXN = 100000;
7  int prime[MAXN + 1];
8
9  void getPrime()
10 {
11     memset(prime, 0, sizeof(prime));
12     for (int i = 2; i <= MAXN; i++)
13     {
14         if (!prime[i])
15         {
16             prime[++prime[0]] = i;
17         }
18         for (int j = 1; j <= prime[0] && prime[j] <= MAXN / i; j++)
19         {
20             prime[prime[j] * i] = 1;
21             if (i % prime[j] == 0)
22             {
23                 break;

```

```

24     }
25     }
26 }
27 }

```

5.14.4 Miller Rabin

```

1  /*
2  * 随机素数测试（伪素数原理）
3  *  CALL: bool res = miller(n);
4  *  快速测试n是否满足素数的“必要”条件，出错概率极低
5  *  对于任意奇数n > 2和正整数s，算法出错概率 2-(s)
6  */
7
8  int witness(int a, int n)
9  {
10     int x, d = 1;
11     int i = ceil(log(n - 1.0) / log(2.0)) - 1;
12     for (; i >= 0; i--)
13     {
14         x = d;
15         d = (d * d) % n;
16         if (d == 1 && x != 1 && x != n - 1)
17         {
18             return 1;
19         }
20         if (((n - 1) & (1 << i)) > 0)
21         {
22             d = (d * a) % n;
23         }
24     }
25     return (d == 1 ? 0 : 1);
26 }
27
28 int miller(int n, int s = 50)
29 {
30     if (n == 2) // 质数返回1
31         return 1;
32     if (n % 2 == 0) // 偶数返回0
33         return 0;
34     int j, a;
35     for (j = 0; j < a; j++)

```

```

36 {
37     a = rand() * (n - 2) / RAND_MAX + 1;
38     // rand()只能随机产生[0, RAND_MAX)内的整数
39     // 而且这个RAND_MAX只有32768直接%n的话是永远
40     // 也产生不了[RAND_MAX, n)之间的数
41     if (witness(a, n))
42     {
43         return 0;
44     }
45 }
46 return 1;
47 }

```

5.15 组合数

```

1  int com(int n, int r) // return C(n, r)
2  {
3     if (n - r > r)
4     {
5         r = n - r; // C(n, r) = C(n, n - r)
6     }
7     int i, j, s = 1;
8     for (i = 0, j = 1; i < r; ++i)
9     {
10         s *= (n - i);
11         for (; j <= r && s % j == 0; ++j)
12         {
13             s /= j;
14         }
15     }
16     return s;
17 }

```

5.16 阶乘长度

```

1  #define PI 3.1415926
2
3  int main()
4  {
5     int n, a;
6     while (~scanf("%d", &n))

```

```

7 {
8     a = (int)((0.5 * log(2 * PI * n) + n * log(n) - n) / log(10));
9     printf("%d\n", a + 1);
10 }
11 return 0;
12 }

```

5.17 全排列

```

1 #define MAX_N 10
2 int n; // 共n个数
3 int rcd[MAX_N]; // 记录每个位置填的数
4 int used[MAX_N]; // 标记数是否用过
5 int num[MAX_N]; // 存放输入的n个数
6
7 void full_permutation(int l)
8 {
9     int i;
10    if (l == n)
11    {
12        for (i = 0; i < n; i++)
13        {
14            printf("%d", rcd[i]);
15            if (i < n-1)
16            {
17                printf(" ");
18            }
19        }
20        printf("\n");
21        return ;
22    }
23    for (i = 0; i < n; i++) // 枚举所有的数(n个),循环从开始
24    if (!used[i])
25    {
26        used[i] = 1; // 若num[i]没有使用过,则标记为已使用
27        rcd[l] = num[i]; // 在l位置放上该数
28        full_permutation(l+1); // 填下一个位置
29        used[i] = 0; // 清标记
30    }
31 }

```

5.18 求斐波那契第 N 项

```

1 /*
2 * 求斐波那契数列第N项, 模MOD
3 */
4 #define mod(a, m) ((a) % (m) + (m)) % (m)
5 const int MOD = 1e9 + 9;
6 struct MATRIX
7 {
8     long long a[2][2];
9 };
10
11 MATRIX a;
12 long long f[2];
13
14 void ANS_Cf(MATRIX a)
15 {
16     f[0] = mod(a.a[0][0] + a.a[1][0], MOD);
17     f[1] = mod(a.a[0][1] + a.a[1][1], MOD);
18     return ;
19 }
20
21 MATRIX MATRIX_Cf(MATRIX a, MATRIX b)
22 {
23     MATRIX ans;
24     int k;
25     for (int i = 0; i < 2; i++)
26     {
27         for (int j = 0; j < 2; j++)
28         {
29             ans.a[i][j] = 0;
30             k = 0;
31             while (k < 2)
32             {
33                 ans.a[i][j] += a.a[k][i] * b.a[j][k];
34                 ans.a[i][j] = mod(ans.a[i][j], MOD);
35                 ++k;
36             }
37         }
38     }
39     return ans;
40 }
41
42 MATRIX MATRIX_Pow(MATRIX a, long long n)

```

```

43 {
44     MATRIX ans;
45     ans.a[0][0] = 1;
46     ans.a[1][1] = 1;
47     ans.a[0][1] = 0;
48     ans.a[1][0] = 0;
49     while (n)
50     {
51         if (n & 1)
52         {
53             ans = MATRIX_Cf(ans, a);
54         }
55         n = n >> 1;
56         a = MATRIX_Cf(a, a);
57     }
58     return ans;
59 }
60
61 int main()
62 {
63     long long n;
64     while (cin >> n)
65     {
66         if (n == 1)
67         {
68             cout << '1' << '\n';
69             continue;
70         }
71         a.a[0][0] = a.a[0][1] = a.a[1][0] = 1;
72         a.a[1][1] = 0;
73         a = MATRIX_Pow(a, n - 2);
74         ANS_Cf(a);
75         cout << f[0] << '\n';
76     }
77     return 0;
78 }

```

6 STL

6.1 vector

```
1 vector<int> s;
```

```

2 // 定义一个空的vector对象, 存储的是int类型的元素
3 vector<int> s(n);
4 // 定义一个含有n个int元素的vector对象
5 vector<int> s(first, last);
6 // 定义一个vector对象, 并从由迭代器first和last定义的序列[first, last)中复制
   初值
7
8 s[i]           // 直接以下标方式访问容器中的元素
9 s.front()      // 返回首元素
10 s.back()       // 返回尾元素
11 s.push_back(x) // 向表尾插入元素x
12 s.size()       // 返回表长
13 s.empty()      // 表为空时, 返回真, 否则返回假
14 s.pop_back()   // 删除表尾元素
15 s.begin()      // 返回指向首元素的随机存取迭代器
16 s.end()        // 返回指向尾元素的下一个位置的随机存取迭代器
17 s.insert(it, val) // 向迭代器it指向的元素前插入新元素val
18 s.insert(it, n, val) // 向迭代器it指向的元素前插入n个新元素val
19 s.insert(it, first, last)
20 // 将由迭代器first和last所指定的序列[first, last)插入到迭代器it指向的元素前
   面
21 s.erase(it)    // 删除由迭代器it所指向的元素
22 s.erase(first, last) // 删除由迭代器first和last所指定的序列[first, last)
23 s.reserve(n)   // 预分配缓冲空间, 使存储空间至少可容纳n个元素
24 s.resize(n)    // 改变序列长度, 超出的元素将会全部被删除, 如果序列需要
   扩展 (原空间小于n), 元素默认值将填满扩展出的空间
25 s.resize(n, val) // 改变序列长度, 超出的元素将会全部被删除, 如果序列需要
   扩展 (原空间小于n), val将填满扩展出的空间
26 s.clear()      // 删除容器中的所有元素
27 s.swap(v)      // 将s与另一个vector对象进行交换
28 s.assign(first, last) // 将序列替换成由迭代器first和last所指定的序列[first,
   last), [first, last)不能是原序列中的一部分
29
30 // 要注意的是, resize操作和clear操作都是对表的有效元素进行的操作, 但并不一
   定会改变缓冲空间的大小
31 // vector上还定义了序列之间的比较操作运算符 (>, <, >=, <=, ==, !=), 可以按
   照字典序比较两个序列。

```

6.2 set

```

1 // 有序
2 set<int> s;

```

```

3 s.begin()      // 返回指向第一个元素的迭代器
4 s.clear()      // 清除所有元素
5 s.count()      // 返回某个值元素的个数
6 s.empty()      // 如果集合为空, 返回true(真)
7 s.end()        // 返回指向最后一个元素之后的迭代器, 不是最后一个元素
8 s.equal_range() // 返回集合中与给定值相等的上下限的两个迭代器
9 s.erase()      // 删除集合中的元素
10 s.find()       // 返回一个指向被查找元素的迭代器
11 s.get_allocator() // 返回集合的分配器
12 s.insert()     // 在集合中插入元素
13 s.lower_bound() // 返回指向大于(或等于)某值的第一个元素的迭代器
14 s.key_comp()   // 返回一个用于元素间值比较的函数
15 s.max_size()   // 返回集合能容纳的元素的最大限值
16 s.rbegin()     // 返回指向集合中最后一个元素的反向迭代器
17 s.rend()       // 返回指向集合中第一个元素的反向迭代器
18 s.size()       // 集合中元素的数目
19 s.swap()       // 交换两个集合变量
20 s.upper_bound() // 返回大于某个值元素的迭代器
21 s.value_comp() // 返回一个用于比较元素间的值的函数
22
23 // 多重集合
24 multiset<int> s;
25 // 操作类似

```

6.3 pair

```

1 pair<T1, T2> p1;
2 pair<T1, T2> p1(v1, v2);
3 p1.first;
4 p1.second;
5 p1 = make_pair(v1, v2);
6
7 vector<pair<int, int> > v;
8 sort(v.begin(), v.end()); //根据pair的first排序, 从小到大

```

6.4 stack

```

1 stack<int> s;
2 s.push(x); // 入栈
3 s.pop();   // 出栈
4 s.top();   // 访问栈顶

```

```

5 s.empty(); // 当栈空时, 返回true
6 s.size();  // 访问栈中元素个数

```

6.5 queue

```

1 queue<int> q;
2 q.push(x); // 入队列
3 q.pop();   // 出队列
4 q.front(); // 访问队首元素
5 q.back();  // 访问队尾元素
6 q.empty(); // 判断队列是否为空
7 q.size();  // 访问队列中的元素个数
8
9 //priority_queue(优先队列)。优先队列与队列的差别在于优先队列不是按照入队的
  顺序出队, 而是按照队列中元素的优先权出队列(默认为大者优先, 也可以通过指定
  算子来指定自己的优先顺序)。
10
11 priority_queue模版类有三个模版参数, 第一个是元素类型, 第二个是容器类型, 第三
  个是比较算子。其中后两者都可以忽略, 默认容器为vector, 默认算子为less, 即小
  的往前排, 大的往后排(出队列时列尾元素先出队)。
12
13 priority_queue<int> q;
14 priority_queue<pair<int, int> > qq; // 注意在两个尖括号之间一定要留空格, 防
  止误判
15 priority_queue<int, vector<int>, greater<int> > qqq; // 定义小的先出队列
16
17 q.empty()    // 如果队列为空, 则返回true, 否则返回false
18 q.size()     // 返回队列中元素的个数
19 q.pop()      // 删除队首元素, 但不返回其值
20 q.top()      // 返回具有最高优先级的元素值, 但不删除该元素
21 q.push(item) // 在基于优先级的适当位置插入新元素
22
23 //deque双端队列
24 #include<deque>
25 deque<int> dep;
26
27 deq.push_front(const T& x); //头插
28 deq.push_back(const T& x);  //尾插
29 deq.insert(iterator it, const T& x); //任意位置
30
31 deq.pop_front(); //删除头部
32 deq.pop_back();  //删除尾部

```

```

33 deq.erase(iterator it); //删除任意
34
35 deq[1]; // 并不会检查是否越界
36 deq.at(1); // 以上两者的区别就是 at 会检查是否越界，是则抛出 out of range 异常
37 deq.front(); //访问头部
38 deq.back(); //访问尾部

```

6.6 map

```

1 map<T1, T2> mp;
2 mp[key] = value;
3 T2 value = mp[key];
4 mp.size(); // 返回元素个数
5 mp.empty(); // 判断是否为空
6 mp.clear(); // 清空所有元素

```

6.7 bitset

```

1 const int MAXN = 32;
2
3 bitset<MAXN> bt; // bt 包括 MAXN 位，下标 0 ~ MAXN - 1，默认初始化为 0
4 bitset<MAXN> bt1(0xf); // 0xf 表示十六进制数 f，对应二进制 1111，将 bt1 低 4 位初始化为 1
5 bitset<MAXN> bt2(012); // 012 表示八进制数 12，对应二进制 1010，即将 bt2 低 4 位初始化为 1010
6 bitset<MAXN> bt3("1010"); // 将 bt3 低 4 位初始化为 1010
7
8 bt.any() // bt 中是否存在置为 1 的二进制位？
9 bt.none() // bt 中不存在置为 1 的二进制位吗？
10 bt.count() // bt 中置为 1 的二进制位的个数
11 bt.size() // bt 中二进制位的个数
12 bt[pos] // 访问 bt 中在 pos 处的二进制位
13 bt.test(pos) // bt 中在 pos 处的二进制位是否为 1
14 bt.set() // 把 bt 中所有二进制位都置为 1
15 bt.set(pos) // 把 bt 中在 pos 处的二进制位置为 1
16 bt.reset() // 把 bt 中所有二进制位都置为 0
17 bt.reset(pos) // 把 bt 中在 pos 处的二进制位置为 0
18 bt.flip() // 把 bt 中所有二进制位逐位取反
19 bt.flip(pos) // 把 bt 中在 pos 处的二进制位取反

```

```

20 bt[pos].flip() // 同上
21 bt.to_ulong() // 用 bt 中同样的二进制位返回一个 unsigned long 值

```

6.8 algorithm

```

1 reverse(begin, end) // 反转
2
3 unique(begin, end) // 需排序，去除重复的相邻元素，常用于求不同元素个数
4 int n=unique(a, a+10)-a;
5
6 lower_bound(begin, end, value) //返回指向第一个不小于给定值的元素的迭代器
7
8 upper_bound(begin, end, value) //返回指向第一个大于给定值的元素的迭代器
9
10 next_permutation(array) //一种排列的下一排列

```

7 计算几何

7.1 三角形面积

```

1 \\ 海伦公式
2 int p = (a+b+c)/2;
3 int s = sqrt(p * (p - a) * (p - b) * (p - c));
4
5 \\ 两边和夹角
6 \\ a, b为边, x为a和b的夹角
7 s = 0.5 * a * b * sin(x/90.0*acos(0));

```

7.2 两圆面积交

```

1 const double PI = acos(-1);
2
3 struct circle
4 {
5     double x, y, r;
6 };
7
8 // 计算圆心距

```

```

9 double dist(circle a, circle b)
10 {
11     return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
12 }
13
14 double area(circle a, circle b)
15 {
16     if((dist(a, b) + min(a.r, b.r)) <= max(a.r, b.r)) // 内含或重合
17     {
18         if(a.r < b.r)
19             return PI*a.r*a.r;
20         else
21             return PI*b.r*b.r;
22     }
23     else if(dist(a, b) >= (a.r + b.r)) // 相切
24     {
25         return 0.0;
26     }
27     else
28     {
29         double length = dist(a, b);
30         // 利用三角形余弦定理求圆心角
31         double d1 = 2*acos((a.r*a.r+length*length-b.r*b.r)/(2*a.r*length));
32         double d2 = 2*acos((b.r*b.r+length*length-a.r*a.r)/(2*b.r*length));
33         // 利用圆心角求得扇形面积再减去三角形面积后两部分相加就是相交面积
34         double area1 = a.r*a.r*d1/2 - a.r*a.r*sin(d1)/2;
35         double area2 = b.r*b.r*d2/2 - b.r*b.r*sin(d2)/2;
36         return area1 + area2;
37     }
38 }

```

8 其它

8.1 数据类型范围

数据类型	范围
char	-128 - 127
int	2147483648 - 2147483647(十位)
long long	-9223372036854775808 - 9223372036854775807(大约十九位)
double	1.7 * 10308

8.2 头文件

```

1 //include<stdio.h>
2 //include<iostream>
3 //include<queue>
4 //include<algorithm>
5 //include<cstring>
6 //include<vector>
7 //include<cmath>
8 //include<string>
9 //include<map>
10 //include<set>
11 #include<bits/stdc++.h>
12 using namespace std;
13 typedef long long ll;
14 #define inf 0x3f3f3f3f
15 typedef pair<int, int> P;
16 const int maxn = 5e4+5;
17 const ll mod = 1e9+7;

```

8.3 Vim 配置

```

1 // open ~/.vimrc
2 syntax on
3 set nu ts=4 sw=4 mouse=a cin
4 colo desert
5
6 map <C-A> ggVG"+y
7 map <F5> :call CR()<CR>
8 func! CR()
9     exec "w"
10     exec "!g++ -O2 -g -std=c++11 -Wall % -o %<"
11     exec "! ./%<"
12 endfunc

```

8.4 输入挂

8.4.1 关闭同步

```

1 #define endl '\n'
2

```



```

3 ios::sync_with_stdio(0);
4 cin.tie(0);

```

8.4.2 IO

```

1 #include<cstdio>
2
3 inline void read(int &x) //看情况可去掉负数部分
4 {
5     int t = 1;
6     char ch = getchar();
7     while(ch < '0' || ch > '9') { if(ch == '-') t = -1; ch = getchar(); }
8     x = 0;
9     while(ch >= '0' && ch <= '9'){ x=x*10+ch-'0'; ch = getchar(); }
10    x*=t;
11 }
12
13 void print(int i){
14     if(i<10){
15         putchar('0'+i);
16         return ;
17     }
18     print(i/10);
19     putchar('0'+i%10);
20 }

```

8.5 C++ 大数

8.5.1 大数加法

```

1 string add(string a,string b)
2 { //两数相加
3     string res="";
4     int i=1;
5     string first="0";
6     while(true) {
7         int tai=a.size()-i;
8         int tbi=b.size()-i;
9         if(tai<0 && tbi<0)
10            break; //从两数最右边开始模拟加法运算直到两数都遍历完
11         int ta,tb;

```

```

12         if(tai<0)
13             ta=0; //如果没数则至为0
14         else
15             ta=a[tai]-'0';
16         if(tbi<0)
17             tb=0; //如果没数则至为0
18         else
19             tb=b[tbi]-'0';
20         int temp=ta+tb+first[0]-'0'; //相加 first保存上一个的进位信息
21         first[0]=temp%10+'0'; //当前位是对10取余
22         res=first+res;
23         first[0]=temp/10+'0'; //进位是除10
24         i++;
25     }
26     if(first!="0")
27         res=first+res; //如果进位还有则添加
28     if(res[0]=='0' && res.size(>1) //去除前导0
29         res.erase(res.begin());
30     return res;
31 }

```

8.5.2 大数乘法

```

1 // 还需要配合大数加法
2 string mul(string a,string b) {
3     if(b.size()==1) { //如果b只有一位则使其分别相乘
4         string res="";
5         string first="0";
6         int mb=b[0]-'0';
7         for(int i=a.size()-1; i>=0; i--) { //从最后开始依次计算
8             int temp=(a[i]-'0') * mb + (first[0]-'0');
9             first[0]=temp%10+'0'; //当前位
10            res=first+res;
11            first[0]=temp/10+'0'; //进位
12        }
13        if(first!="0") {
14            res=first+res; //处理进位
15        }
16        if(res[0]=='0' && res.size(>1)
17            res.erase(res.begin()); //去除前导0
18        return res;
19    }

```

```

20 //否则则把b拆分为一位
21 string res="0";
22 string zero="";
23 for(int i=b.size()-1; i>=0; i--) { //从b的最后一位开始
24     string temp=mul(a,b.substr(i,1)); //计算当前为与a相乘
25     res=add(res,temp+zero); //在其后添加适当的0再与结果相加
26     zero=zero+"0";
27 }
28 return res;
29 }

```

8.5.3 整数转 string

```

1 string inttostring(int m) { //将整数转为string
2     string res="";
3     string temp="0";
4     while(m) {
5         temp[0]=m%10+'0';
6         res=temp+res;
7         m/=10;
8     }
9     if(res=="")
10         res="0";
11     return res;
12 }

```

8.6 Java

```

1 valueOf(paramant); 将参数转换为制定的类型
2
3 比如 int a=3;
4
5 BigInteger b=BigInteger.valueOf(a);
6
7 则b=3;
8
9 String s=" 12345" ;
10
11 BigInteger c=BigInteger.valueOf(s);
12
13 则c=12345;

```

```

14
15 // 常用函数
16 1.赋值:
17 BigInteger a=new BigInteger("1");
18 BigInteger b=BigInteger.valueOf(1);
19
20 2.运算:
21     add(); 大整数相加
22 BigInteger a=new BigInteger( "23" );
23 BigInteger b=new BigInteger( "34" );
24 a.add(b);
25
26 subtract(); 相减
27 multiply(); 相乘
28 divide(); 相除取整
29 remainder(); 取余
30 pow(); a.pow(b)=a^b
31 gcd(); 最大公约数
32 abs(); 绝对值
33 negate(); 取反数
34 mod(); a.mod(b)=a%b=a.remainder(b);
35
36 3.BigInteger构造函数:
37 一般用到以下两种:
38 BigInteger(String val);
39 将指定字符串转换为十进制表示形式;
40 BigInteger(String val,int radix);
41 将指定基数的 BigInteger 的字符串表示形式转换为 BigInteger
42
43 4.基本常量:
44 A=BigInteger.ONE 1
45 B=BigInteger.TEN 10
46 C=BigInteger.ZERO 0
47
48 5.n.compareTo(BigInteger.ZERO)==0 //相当于n==0
49
50 6.if(a[i].compareTo(n)>=0 && a[i].compareTo(m)<=0) // a[i]>=n && a[i]<=m
51
52 // 模板
53 import java.math.BigInteger;
54 import java.math.BigDecimal;
55 import java.util.Scanner;
56 import java.util.*;
57 import java.io.*;

```

```
58 public class Main {
59     public static void main(String [] args){
60         Scanner cin = new Scanner(System.in);
61         BigInteger a, b;
62         while(cin.hasNext())//相当于c语言中的scanf("%d", &n) != EOF
63         {
64             a = cin.nextBigInteger();
65             b = cin.nextBigInteger();
66             System.out.println(a.add(b));//大整数加法
67             System.out.println(a.subtract(b));//大整数减法
68             System.out.println(a.multiply(b));//大整数乘法
69             System.out.println(a.divide(b));//大整数除法, 取整
70             System.out.println(a.remainder(b));//大整数取模
71             System.out.println(a.abs());//对大整数a取绝对值
72             int x = 0;
73             System.out.println(a.pow(x));//大整数a的x次幂
74             int y = 8;
75             System.out.println(a.toString(y));//返回大整数a的p进制用字符串表现的形
            式
76             System.out.println(a.toString());//返回大整数a的十进制用字符串表现的形
            式
77             //大整数之间的比较
78             if( a.compareTo(b) == 0 ) System.out.println("a == b"); //大整数a==b
79             else if( a.compareTo(b) > 0 ) System.out.println("a > b"); //大整数a>b
80             else if( a.compareTo(b) < 0 ) System.out.println("a < b"); //大整数a<b
            )
81
82             BigDecimal c, d;
83             c = cin.nextBigDecimal();
84             d = cin.nextBigDecimal();
85             System.out.println(c.add(d));//浮点数相加
86             System.out.println(c.subtract(d));//浮点数相减
87             System.out.println(c.multiply(d));//浮点数相乘
88         }
89     }
90 }
91
92 // 输入方式
93 int a = cin.nextInt();
94 String s = cin.nextLine();
95 double b = cin.nextDouble();
```