bestsort 常用算法模板

生命不息，

代码不止

2018 年 10 月

By Bestsort

一、数论.....3
1.筛选法素数打表.....3
2.欧拉函数.....3
3.快速幂取余.....4
4.快乘取余.....4
5.扩展欧几里德.....5
6.中国剩余定理.....5
7.逆元.....6
8.费马小定理（求逆元）.....6
9.母函数.....6
10.求线性递推式.....7
二、图.....10
1.链式前向星.....10
2.最短路.....10
dijkstra(非负权单源最短路(O(V^2)))).....10
bellman-ford(可判负权回路,O(VE)).....11
floyd(多源最短路,O(V^3)).....12
Dijkstra+堆优化((V+E)logV).....13
3.k 短路.....14
1.朴素 A*.....14
4.二分图的最大匹配(O(VE)).....16
5.拓扑排序(O(V+E)).....16
三.树.....17
1.最小生成树.....17
prim 算法(O(V^2)).....17
kruskal (O(ElogE)).....18
2.最小树形图 (朱刘算法).....19
3.LCA(最近公共祖先).....20
Tarjan 离线(O(V+E)).....20
在线 ST+RMQ(VlogV).....22
四、计算几何.....22
1.两圆相交面积.....22
2.叉积.....23
3.点积.....23
4.pick 定理.....24
五、动态规划.....24
1.背包.....24
2.LCS(最长公共子序列).....25
3.LIS(最长上升子序列).....25
4.数位 DP.....27
六、串.....28
1.最长回文串(Manacher 算法 O(n)).....28
2.模式匹配(KMP 算法(O(n+m))).....29
3.字典树(O(logn)).....30

3.AC 自动机(O(strlen)) 31
七、博弈....32
1.常用博弈....32
八.组合数....33
康托展开....34
平面划分问题....35
错排....35
基姆拉尔森公式(计算日期) 35
九.数据结构....36
1.并查集....36
2.线段树....36
3.一维树状数组....38
4.二维树状数组....39
5.划分树....42
6.RMQ....43
十.数学 43
1.解方程....43
2.矩阵快速幂....47
十一.其他....49
1.常用函数及语句....49
2.数据类型取值范围....49
3.c++大数....50

一、数论

1. 筛选法素数打表

```
int a[N]={1,1,0};
void isPrime()
{
    for(int i=2;i<N;i++)
        if(!a[i])
            for(int j=i+i;j<N;j+=i)
                a[j]=1;
}
```

2. 欧拉函数

朴素算法

```
int Eular(int n)
{
    int s=n;
    for(int i=2;i*i<=n;i++){
        if(n%i==0)
            s=s/i*(i-1);
        while(n%i==0)
            n/=i;
    }
    if(n>1)
        s=s/n*(n-1);
    return s;
}
```

3. 快速幂取余

```
LL powerMod(LL a,LL b,LL mod)
{
    LL ans=1;
    a%=mod;
    while(b){
        if(b&1)
            ans=ans*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return ans;
}
```

4. 快乘取余

```
LL mulMod(LL x,LL y,LL mod)
{
    LL ans=0;
    while(y){
        if(y&1)
            ans=(ans+x)%mod;
        x=(x+x)%mod;
        y>>=1;
    }
    return ans;
}
```

5. 扩展欧几里德

```
1  //a*x+b*y=gcd(a,b)
2  LL exgcd(LL a,LL b,LL &x,LL &y)
3  {
4      if(b==0){
5          x=1;
6          y=0;
7          return a;
8      }
9      LL d=exgcd(b,a%b,x,y);
10     LL t=x;
11     x=y;
12     y=t-a/b*y;
13     return d;
14 }
```

定理：设 a,b,c 为任意整数

若方程 ax+by=c 的一组整数解为（x0,y0）则它的任意整数解都可以写成(x0+kb’,y0-ka’),其中 a’=a/gcd(a,b),b’=b/gcd(a,b),k 为任意整数

若 ax+by=g (g=gcd(a,b),即 g 是 a,b 的最大公约数),有整数解(x1,y1),则 ax+by=c（c 是 g 的倍数）有整数解(c*x1/g,c*y1/g)

6. 中国剩余定理

令 m1,m2,…,mnm1,m2,…,mn 为两两互素的正整数，则同余方程组

x≡a1(modm1)x≡a2(modm2)⋮

x≡an(modmn)[x≡a1(modm1)x≡a2(modm2)⋮

x≡an(modmn)

有唯一的模 m=m1m2…mnm=m1m2…mn 解。（即有一个解 x，使 0≤x≤m，且所有其他的解均与此解模 m 同余。）

```
1  LL china(int n)
2  {
3      LL M=1,x=0;
4      for(int i=0;i<n;i++)
5          M*=m[i];
6      for(int i=0;i<n;i++){
7          LL w=M/m[i];
8          LL d,y;
9          d=exgcd(m[i],w,d,y);
10         x=(x+y*w*a[i])%M;
11     }
12     return (x+M)%M;
13 }
```

7. 逆元

对于正整数 a,n,满足 a*x≡1(mod n)的 x 值就是 a 关于 n 的乘法逆元。当且仅当 a 和 n 互质即 gcd(a,n)=1 时有解

扩展欧几里德求逆元

```
1 //模 n 下 a 的逆，如果不存在返回-1
2 LL inv(LL a,LL n)
3 {
4       LL d,x,y;
5       d=exgcd(a,n,x,y);
6       return d==1?(x+n)%n:-1;
7 }
```

8. 费马小定理（求逆元）

假如 p 是质数,且 gcd(a,p)=1,那么 a^(p-1)≡1(mod p)，即 a^(p-2)≡1/a (mod p)

9. 母函数

```
1   * c1 是保存各项质量砝码可以组合的数目
2   * c2 是中间量，保存每一次的情况
3   const int MAXN = 1e4 + 10;
4   int n;
5   int c1[MAXN];
6   int c2[MAXN];
7   int main() {
8           while (cin >> n) {
9                   for (int i = 0; i <= n; ++i) {
10                          c1[i] = 1;
11                          c2[i] = 0;
12                  }
13                  for (int i = 2; i <= n; ++i) {
14                          for (int j = 0; j <= n; ++j) {
15                                  for (int k = 0; k + j <= n; k += i) {
16                                          c2[j + k] += c1[j];
17                                  }
18                          }
19                          for (int j = 0; j <= n; ++j) {
20                                  c1[j] = c2[j];
```

```
21                          c2[j] = 0;
22                      }
23              }
24              cout << c1[n] << endl;
25      }
26      return 0;
27 }
```

10. 求线性递推式

Berlekamp-Massey

时间复杂度: $(O(n^2\log(k)))$(n 为前置项,k 为求解的那一项)

```
1   #define rep(i,a,n) for (int i=a;i<n;i++)
2   #define per(i,a,n) for (int i=n-1;i>=a;i--)
3   #define pb push_back
4   #define mp make_pair
5   #define all(x) (x).begin(),(x).end()
6   #define fi first
7   #define se second
8   #define SZ(x) ((int)(x).size())
9   typedef vector<int> VI;
10  typedef long long ll;
11  typedef pair<int,int> PII;
12  const ll mod=1000000007;
13  const int maxn=10010;
14
15  ll powmod(ll a,ll b) {
16          ll res=1;
17          a%=mod;
18          assert(b>=0);
19          for(; b; b>>=1) {
20                  if(b&1)
21                          res=res*a%mod;
22                  a=a*a%mod;
23          }
24          return res;
25  }
26  // head
27
28  ll n;
29  namespace linear_seq {
```

```
ll res[maxn],base[maxn],_c[maxn],_md[maxn];

vector<int> Md;
void mul(ll *a,ll *b,int k) {
    rep(i,0,k+k) _c[i]=0;
    rep(i,0,k) if (a[i])
        rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
    for (int i=k+k-1; i>=k; i--)
        if (_c[i])
            rep(j,0,SZ(Md))                    _c[i-k+Md[j]]=(_c[i-k+Md[j]]-
_c[i]*_md[Md[j]])%mod;
    rep(i,0,k) a[i]=_c[i];
}
int solve(ll n,VI a,VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
    ll ans=0,pnt=0;
    int k=SZ(a);
    assert(SZ(a)==SZ(b));
    rep(i,0,k) _md[k-1-i]=-a[i];
    _md[k]=1;
    Md.clear();
    rep(i,0,k) if (_md[i]!=0)
        Md.push_back(i);
    rep(i,0,k) res[i]=base[i]=0;
    res[0]=1;
    while ((1ll<<pnt)<=n)
        pnt++;
    for (int p=pnt; p>=0; p--) {
        mul(res,res,k);
        if ((n>>p)&1) {
            for (int i=k-1; i>=0; i--)
                res[i+1]=res[i];
            res[0]=0;
            rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
        }
    }
    rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
    if (ans<0)
        ans+=mod;
    return ans;
}
VI BM(VI s) {
    VI C(1,1),B(1,1);
    int L=0,m=1,b=1;
    rep(n,0,SZ(s)) {
```

```
74              ll d=0;
75              rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
76              if (d==0)
77                  ++m;
78              else if (2*L<=n) {
79                  VI T=C;
80                  ll c=mod-d*powmod(b,mod-2)%mod;
81                  while (SZ(C)<SZ(B)+m)
82                      C.pb(0);
83                  rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
84                  L=n+1-L;
85                  B=T;
86                  b=d;
87                  m=1;
88              } else {
89                  ll c=mod-d*powmod(b,mod-2)%mod;
90                  while (SZ(C)<SZ(B)+m)
91                      C.pb(0);
92                  rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
93                  ++m;
94              }
95          }
96          return C;
97      }
98      int gao(VI a,ll n) {
99          VI c=BM(a);
100         c.erase(c.begin());
101         rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
102         return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
103     }
104     };
105
106     int main() {
107         /*push_back 进去前 8~10 项左右、最后调用 gao 得第 n 项*/
108         //这里是 fib 数列，用验证正确性 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,
109         //233，377，610，987，1597，2584，4181，6765，10946，17711，28657，
110     46368.....
111         vector<int>v;
112         for(int i=1;i<=10;i++)
113            v.push_back(fib[i]);
114         int nCase;
115         scanf("%d", &nCase);
116         while(nCase--) {
117             scanf("%lld", &n);
```

```
118             printf("%lld\n",1LL * linear_seq::gao(v,n-1) % mod);
119         }
200 }
```

二、图

1. 链式前向星

```
1  //存储结构
2  struct Edge
3  {
4          int to;   //边的终点
5          int w;     //边的权值
6          int next; //起点相同的下一条边
7  }edge[M];     //M 为边数，N 为顶点数
8  int head[N]; //head[i]是以 i 为起点的第一条边的编号
9  int cnt;   //记录边数
10 //初始化
11 cnt=0;
12 memset(head,-1,sizeof(head));
13 //建图
14 void addEdge(int u,int v,int w)
15 {
16         edge[cnt].to=v;
17         edge[cnt].w=w;
18         edge[cnt].next=head[u];
19         head[u]=cnt++;
20 }
21 //遍历以 u 为起点的邻接边
22 for(int i=head[u];i!=-1;i=edge[i].next){
23         int to=edge[i].to; //终点
24         int w=edge[i].w;     //权值
25 }
```

2. 最短路

dijkstra(非负权单源最短路(O(V^2)))

```
int dis[maxn];                ///到各点的距离
bool used[maxn];              ///标记该点是否已经使用
int pre[maxn];                ///最短路经过的路径
int map[maxn][maxn];
void dijkstra(int a){
        mem(used,0);
        for(int i=1;i<=n;i++){
                dis[i] = map[a][i];
                pre[i] = (dis[i]==INF)?-1:a;
        }
        dis[a] = 0;
        used[a] = true;
        for(int i=1;i<=n;i++){
                int Min = INF,Np = a;
                for(int j=1;j<=n;j++)
                if(!used[j] && dis[j] <Min){
                        Np = j;
                        Min = dis[j];
                }
                if(Np == a)
                        return;
                used[Np] = true;
                for(int j=1;j<=n;j++)
                        if(dis[j] > (dis[Np]+map[Np][j])){
                                dis[j] = dis[Np]+map[Np][j];
                                pre[j] = Np;
                        }
        }
}
```

bellman-ford(可判负权回路,O(VE))

```
int dis[maxn];
struct edge{
        int s,e;      ///起点，终点
        int w;          ///权值
}e[maxn];
int n,m;              //n 为点,m 为边的总数
bool bellman(int a)        ///可判负环
{
        mem(dis,inf);
        dis[a]=0;
        fori(n-1)
```

```
12              forj(m)
13                  dis[e[j].e]=min(dis[e[j].e],dis[e[j].s]+e[j].w);//判断是否更新
14      for(int i=1;i<=n;i++)                                   ///松弛完后还能再松弛即代表有负环
15              if(dis[e[i].e]>dis[e[i].s]+e[i].w)
16                  return true;
17      return false;
    }
```

SPFA(可判负权回路，最坏 O(VE))

```
1   //链式前向星实现，也可用邻接矩阵
2   bool SPFA(int pos)    //源点
3   {
4       memset(vis,0,sizeof(vis)); //标记是否在队列
5       memset(num,0,sizeof(num)); //记录各顶点松弛次数(即进队列的次数)
6       for(int i=1;i<=n;i++)
7           dis[i]=INF;
8       dis[pos]=0;
9       q.push(pos);
10      vis[pos]=true;
11      num[pos]++;
12      while(!q.empty()){
13          int u=q.front();
14          q.pop();
15          vis[u]=false;
16          for(int i=head[u];i!=-1;i=edge[i].next){
17              int to=edge[i].to;
18              if(dis[u]+edge[i].w<dis[to]){
19                  dis[to]=dis[u]+edge[i].w;
20                  if(!vis[to]){
21                      q.push(to);
22                      vis[to]=true;
23                      num[to]++;
24                      if(num[to]>=n) //当某顶点松弛次数大于或等于 n 时，存在负权回路
25                          return true;
26                  }
27              }
28          }
29      }
30      return false;
31  }
```

floyd(多源最短路,O(V^3))

```
void Floyd(){
    for(int k=0;k<len;k++)
        for(int i=0;i<len;i++)
            for(int j=0;j<len;j++)
                map[i][j]=min(map[i][j],map[i][k]+map[k][j]);
}
```

Dijkstra+堆优化((V+E)logV)

```
int n, m, s;
int fir[maxn], next[maxm], to[maxm], val[maxm], cnt;
void add_edge(int u, int v, int w) //前向星加边
{
    next[++cnt] = fir[u];
    fir[u] = cnt;
    to[cnt] = v;
    val[cnt] = w;
}
struct Node
{
    int d, id;
    Node(){}
    Node(int d, int id) : d(d), id(id){}
    bool operator < (const Node& rhs) const
      {
        return d > rhs.d;//重载 < 方便堆
      }
};
int dis[maxn], vis[maxn];
void Dijkstra(int s)
{
    for(int i = 1; i <= n; i++)
        dis[i] = INF;
    dis[s]=0;
    priority_queue<Node> Q;
    Q.push(Node(0,s));
    while(!Q.empty())
      {
        Node u = Q.top(); Q.pop();
        if(vis[u.id]) continue;   //若某个点已经被更新到最优,就不用再次更新其他点
```

```
33              vis[u.id] = 1;
34              for(int e = fir[u.id]; e; e = next[e])
35                 {
36                    int v = to[e], w = val[e];
37                    if(u.d + w < dis[v])
38                       {
39                          dis[v] = u.d + w;
40                          Q.push(Node(dis[v],v));
41                       }
42                 }
43           }
44 }
```

3. k 短路

1. 朴素 A*

```
1  int    s,t,k;
2  bool vis[maxn];
3  int dis[maxn];
4  struct node{
5         int v,c;
6         node(int _v=0,int _c=0) : v(_v),c(_c) {};       //构造
7         node(){};
8         bool operator < (const node & buf) const{
9                return c+ dis[v]    > buf.c + dis[buf.v];
10        }
11 };
12
13 struct edge{
14        int v,cost;
15        edge(int _v=0,int _c=0) : v(_v),cost(_c){};
16 };
17
18 vector <edge> e[maxn],reve[maxn];          //反向存图(无向图则不需要)
19 priority_queue<node> q;
20 void dijkstra(int n,int s){          //dijkstra+队列优化最短路
21        mem(vis,false);
22        mem(dis,0x3f);
23        while(!q.empty()) q.pop();
```

```
24      dis[s] = 0;
25      q.push(node(s,0));
26      while(!q.empty()){
27          node tmp = q.top();
28          q.pop();
29          int u = tmp.v;
30          if(vis[u])
31              continue;
32          vis[u] = true;
33          fori(e[u].size()){
34              int v = e[u][i].v;
35              int cost = e[u][i].cost;
36              if(!vis[v] && dis[v] > dis[u] + cost){
37                  dis[v] = dis[u] + cost;
38                  q.push(node(v,dis[v]));
39              }
40          }
41      }
42 }
43
44
45 int aStar(int s){
46      while(!q.empty()) q.pop();
47      q.push(node(s,0));
48      k--;
49      while(!q.empty()){
50          node pre = q.top();
51          q.pop();
52          int u = pre.v;
53          if(u == t){                 //终点第 K 次入队代表这为第 K 短路
54              if(k)   k--;
55              else    return pre.c;
56          }
57          fori(reve[u].size()){   //将点 u 连接的所有边入队
58              int v = reve[u][i].v;
59              int c = reve[u][i].cost;
60              q.push(node(v,pre.c+c));
61          }
62      }
63      return -1;
64 }
65
66 void addedge(int u,int v,int w){
67      reve[u].pb(edge(v,w));
```

```
68      e[v].pb(edge(u,w));
69 }
70 //加边->跑一遍 dijkstra->判断是否起点等于终点(==则 K+1)->调用 aStar
```

4.二分图的最大匹配(O(VE))

```
1   int Find(int x)
2   {
3       for(int i=1;i<=m;i++){
4           if(map[x][i] && !used[i]){
5               used[i]=1;
6               if(next[i]==-1 || Find(next[i])){
7                   next[i]=x;
8                   return 1;
9               }
10          }
11      }
12      return 0;
13  }
14  int hungarian()
15  {
16      int sum=0;
17      memset(next,-1);
18      for(int i=1;i<=n;i++){
19          mem(used,0);
20          sum+=Find(i);
21      }
```

最小顶点覆盖要求用最少的点（X 或 Y 中都行），让每条边都至少和其中一个点关联
knoig 定理:二分图的最小顶点覆盖数 = 二分图的最大匹配数（m）
最小路径覆盖:用尽量少的不相交简单路径覆盖有向无环图(DAG)G 的所有顶点
结论：DAG 图的最小路径覆盖数 = 节点数（n）- 最大匹配数（m）
二分图的最大独立集
结论：二分图的最大独立集数 = 节点数（n）- 最大匹配数（m）

5.拓扑排序(O(V+E))

```
1   int tpSort(int n)
2   {
```

```
3       memset(vis,0,sizeof(vis));
4       for(int i=1;i<=n;i++){
5           int pos=-1;
6           for(int j=1;j<=n;j++){
7               if(!vis[j]&&rd[j]==0){ //找到入度为 0 的点
8                   s[i]=j;
9                   vis[j]=true;
10                  pos=j;
11                  break;
12              }
13          }
14          if(pos==-1) //存在回路
15              return 0;
16          for(int j=1;j<=n;j++){ //与 pos 相连的边入度都减一
17              if(edge[pos][j])
18                  rd[j]--;
19          }
20      }
21      return 1;
22  }
```

三.树

1.最小生成树

prim 算法(O(V^2))

```
1   int map[maxn][maxn];
2   bool Is_Used[maxn];
3   int Low_Cost[maxn];
4   int n,m;
5   int prim() {
6       int ans = 0;
7       mem(Is_Used,0);
8       Is_Used[0] = true;
9       for(int i=1; i<n; i++)
10          Low_Cost[i] = map[0][i];
11      for(int i=1; i<n; i++) {
12          int Min = INF;
13          int New_Point = -1;
```

```
14              for(int j=0; j<n; j++) {
15                  if(!Is_Used[j] && Min > Low_Cost[j]) {
16                      Min = Low_Cost[j];
17                      New_Point = j;
18                  }
19              }
20              if(Min = INF)           //不存在最小生成树
21                  return -1;
22              ans += Min;
23              Is_Used[New_Point] = true;
24              for(int j=0; j<n; j++)
25                  if(!Is_Used[j])
26                      Low_Cost[j] = min(Low_Cost[j],map[New_Point][j]);
27      }
28      return ans;
29  }
```

kruskal (O(ElogE))

```
1   struct edge{
2       int a,b;
3       int v;
4   }s[maxn];
5   int pre[maxn];
6   bool cmp(edge a,edge b){return a.v < b.v;}
7   int Find(int a){
8       int root = a;
9       int tmp;
10      while(root != pre[root])root = pre[root];
11      while(a != root){
12          tmp = a;
13          a = pre[a];
14          pre[tmp] = root;
15      }
16      return root;
17  }
18  void combine(int a,int b){
19      int x = Find(a);
20      int y = Find(b);
21      if(x > y)
22          swap(x,y);
23      pre[y] = x;
24  }
```

```
int kruscal(int n,int m){
    int ans = 0;
    For(i,m){
        if(Find(s[i].a)!=Find(s[i].b)){
            combine(s[i].a,s[i].b);
            ans += s[i].v;
            n--;
        }
        if(n == 1)
            break;
    }
    return ans;
}
```

2. 最小树形图（朱刘算法）

```
struct edg {
    int u, v, w;
} edge[M];
int n, m;
int in[N],id[N],vis[N],pre[N];
int Directed_MST(int root) {
    int ret = 0;
    while(true) {
        for(int i=0;i<n;i++)
            in[i]=inf;
        for(int i=0;i<m;i++){      //找最小入边
            int u = edge[i].u;
            int v = edge[i].v;
            if(edge[i].w < in[v] && u != v) {
                in[v] = edge[i].w;
                pre[v] = u;
            }
        }
        for(int i=0;i<n;i++) { //如果存在除 root 以外的孤立点，则不存在最小树形图
            if(i == root)    continue;
            if(in[i] == inf)    return -1;
        }
        int cnt = 0;    //顶点从 0 开始编号
        memset(vis,-1,sizeof(vis));
        memset(id,-1,sizeof(id));
        in[root] = 0;
        for(int i=0;i<n;i++) {      //找环
```

```
28                  ret += in[i];
29                  int v = i;
30                  while(vis[v] != i && id[v] == -1 && v != root) {
31                      vis[v] = i;
32                      v = pre[v];
33                  }
34                  if(v != root && id[v] == -1) {    //重新标号
35                      for(int u = pre[v]; u != v; u = pre[u])
36                          id[u] = cnt;
37                      id[v] = cnt++;
38                  }
39              }
40              if(cnt == 0) break;
41              for(int i=0;i<n;i++)
42                  if(id[i] == -1)
43                      id[i] = cnt++;       //重新标号
44              for(int i=0;i<m;i++){       //更新其他点到环的距离
45                  int v =edge[i].v;
46                  edge[i].u = id[edge[i].u];
47                  edge[i].v = id[edge[i].v];
48                  if(edge[i].u != edge[i].v)
49                      edge[i].w -= in[v];
50              }
51              n = cnt;
52              root = id[root];
53          }
54          return ret;
55  }
```

3.LCA(最近公共祖先)

Tarjan 离线(O(V+E))

```
vector <int >a[maxn];     //邻接表存图
vector <int >q[maxn];     //邻接表存查询
int p[maxn];                  //并查集中的 father 数组
bool v[maxn];                 //是否被访问
bool root[maxn];              //根节点
int res[maxn];                //答案数组
int n;
int Find(int x){
    if(p[x]!=x)
        p[x] = Find(p[x]);
    return p[x];
}
void join(int root1, int root2)
{
    int x, y;
    x = Find(root1);
    y = Find(root2);
    if(x != y)
        p[y] = x;
}
void Tarjan(int u){
    fori(a[u].size())
        if(!v[a[u][i]]){
            Tarjan(a[u][i]);
            join(u,a[u][i]);
            v[a[u][i]] = true;
        }
    fori(q[u].size()){              //对于所有查询，如果 i 点已被访问过，则能查询其公共祖先
        if(v[q[u][i]] == true)
            res[Find(q[u][i])]++;
    }
}
void Init(){                        //初始化
    fori(n+1){
        a[i].clear();
        q[i].clear();
        p[i] = i;
        res[i] = 0;
        v[i] = 0;
        root[i] = 1;
    }
}
```

在线 ST+RMQ(VlogV)

```
1  void ST(int n) {
2          for(int i=1; i<=n; i++)
3                  dp[i][0] = i;
4          for(int j=1; (1<<j)<=n; j++) {
5                  for(int i=1; i+(1<<j)-1<=n; i++) {
6                          int a = dp[i][j-1];
7                          int b = dp[i+(1<<(j-1))][j-1];
8                          dp[i][j] = R[a]<R[b]?a:b;
9                  }
10         }
11 }
12 //中间部分是交叉的。
13 int RMQ(int l,int r) {
14         int k=0;
15         while((1<<(k+1))<=r-l+1)
16                 k++;
17         int a = dp[l][k]
18         int b = dp[r-(1<<k)+1][k]; //保存的是编号
19         return R[a]<R[b]?a:b;
20 }
21
22 int LCA(int u,int v) {
23         int x = first[u], y = first[v];
24         if(x > y)
25                 swap(x,y);
26         int res = RMQ(x,y);
27         return ver[res];
28 }
```

四、计算几何

1. 两圆相交面积

```
1  //求两点间的距离
2  double Distance(point a,point b)
3  {
```

```
4         return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
5   }
6   double InterArea(point a,double R,point b,double r)
7   {
8         if(R<r){
9               double temp=R;
10              R=r;
11              r=temp;
12        }
13        double dis=Distance(a,b);
14        if(dis>=R+r)      //两圆相离，相交面积为 0
15              return 0;
16        if(dis<=R-r)      //两圆内含，相交面积为小圆的面积
17              return PI*r*r;
18        //两圆相交时
19        double angle1=acos((R*R+dis*dis-r*r)/(2.0*R*dis));
20        double angle2=acos((r*r+dis*dis-R*R)/(2.0*r*dis));
21        double s=R*angle1*R+r*angle2*r;
22        s-=R*dis*sin(angle1);
23        return s;
24  }
```

2. 叉积

|c|=|a×b|=|a||b|sin 可用来求平行四变形面积
利用叉积判断两向量的顺逆时针关系：
若 P × Q > 0 ， 则 P 在 Q 的顺时针方向。
若 P × Q < 0 ， 则 P 在 Q 的逆时针方向。
若 P × Q = 0 ， 则 P 与 Q 共线，但可能同向也可能反向。

```
1 double chaji(point a,point b)
2 {
3       return a.x*b.y-a.y*b.x;
4 }
```

3. 点积

cos=a·b/(|a|*|b|),可用来判断两向量是否垂直(a·b=0)

```
1 double dianji(point a,point b)
```

```
2 {
3       return a.x*b.x+a.y*b.y;
4 }
```

4. pick 定理

Pick 定理：设平面上以格子点为顶点的多边形的内部点个数为 a，边上点个数为 b，面积为 S,则 S = a + b/2 -1.
以格子点为顶点的线段，覆盖的点的个数为 gcd(|dx|,|dy|)，其中，|dx|,|dy| 分别为线段横向增量和纵向增量。

五、动态规划

1. 背包

初始化：若没有要求必须将背包装满，而是只希望价值尽量大，初始化时应该将 f[0…V]全设为 0
若要求背包恰好装满，则要将 f[0]=0，其他赋为负无穷，因为背包为 0 可由 0 件物品恰好装满，得到的价值为 0

```
1   int dp[maxn];
2   int volume[maxn], value[maxn], c[maxn];
3
4   int n, v; //  总物品数，背包容量量
5   //val 最大容量,val 总价值,amount 数量
6   // 01 背包
7   void ZeroOnepark(int val, int vol) {
8         for (int j = v ; j >= vol; j--)
9               dp[j] = max(dp[j], dp[j - vol] + val);
10  }
11  //完全背包
12  void Completepark(int val, int vol) {
13        for (int j = vol; j <= v; j++)
14              dp[j] = max(dp[j], dp[j - vol] + val);
15  }
16
```

```
// 多重背包
void Multiplepark(int val, int vol, int amount) {
	if (vol * amount >= v)
		Completepark(val, vol);
	else {
		int k = 1;
		while (k < amount) {
			ZeroOnepark(k * val, k * vol);
			amount -= k;
			k <<= 1;
		}
		if (amount > 0)
			ZeroOnepark(amount * val, amount * vol);
	}
}

for (int i = 1; i <= n; i++)
	Multiplepark(value[i], volume[i], c[i]);
```

2. LCS(最长公共子序列)

```
void LCS()    //s[1-m],t[1-n]
{
	memset(dp,0,sizeof(dp));
	for(int i=1;i<=m;i++)
		for(int j=1;j<=n;j++){
			if(s[i]==t[j])
				dp[i][j]=dp[i-1][j-1]+1;
			else
				dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
		}
}
```

3. LIS(最长上升子序列)

O(n^2)

```
void back_path(int pos)
{
	if(pos!=path[pos])
		back_path(path[pos]);
	printf("%d\n",pos);
}
int LIS(int n)
```

```
{
    for(int i=1;i<=n;i++){
        dp[i]=1;
        path[i]=i; //记录路径
        for(int j=1;j<i;j++){
            if(a[j]<a[i]&&dp[j]+1>dp[i]){
                dp[i]=dp[j]+1;
                path[i]=j;    //记录路径
            }
        }
    }
    int k=1;
    for(int i=2;i<=n;i++){ //求最大值
        if(dp[i]>dp[k]){
            k=i;
        }
    }
    back_path(k);    //回溯路径
    return dp[k];
}
```

O(n*log n)

```
int bin_find(int x)
{
    int l=0,r=n;
    while(l<=r){
        int mid=(l+r)/2;
        if(x>c[mid])
            l=mid+1;
        else if(x<c[mid])
            r=mid-1;
        else
            return mid;
    }
    return l;
}
int LIS()
{
    for(int i=0;i<=n;i++)
        c[i]=INF;
    c[0]=-1;
    c[1]=a[0];
    int len=1;
    for(int i=1;i<n;i++){
        int j=bin_find(a[i]);
```

```
24              c[j]=a[i];
25              if(j>len)
26                  len=j;
27      }
28      return len;
29  }
```

4. 数位 DP

```
1   typedef long long ll;
2   int a[20];
3   ll dp[20][state];//不同题目状态不同
4   ll dfs(int pos,/*state 变量*/,bool lead/*前导零*/,bool limit/*数位上界变量*/)//不是每个题都要判断前导零
5   {
6     //递归边界，既然是按位枚举，最低位是 0，那么 pos==-1 说明这个数我枚举完了
7     if(pos==-1) return 1;/*这里一般返回 1，表示你枚举的这个数是合法的，那么这里就需要你在枚举时必须每一
8   位都要满足题目条件，也就是说当前枚举到 pos 位，一定要保证前面已经枚举的数位是合法的。不过具体题目
9   不同或者写法不同的话不一定要返回 1 */
10    //第二个就是记忆化(在此前可能不同题目还能有一些剪枝)
11    if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];
12    /*常规写法都是在没有限制的条件记忆化，这里与下面记录状态是对应，具体为什么是有条件的记忆化后面
13  会讲*/
14    int up=limit?a[pos]:9;//根据 limit 判断枚举的上界 up;这个的例子前面用 213 讲过了
15    ll ans=0;
16    //开始计数
17    for(int i=0;i<=up;i++)//枚举，然后把不同情况的个数加到 ans 就可以了
18    {
19      if() ...
20      else if()...
21      ans+=dfs(pos-1,/*状态转移*/,lead && i==0,limit && i==a[pos]) //最后两个变量传参都是这样写的
22      /*这里还算比较灵活，不过做几个题就觉得这里也是套路了
23      大概就是说，我当前数位枚举的数是 i，然后根据题目的约束条件分类讨论
24      去计算不同情况下的个数，还有要根据 state 变量来保证 i 的合法性，比如题目
25      要求数位上不能有 62 连续出现,那么就是 state 就是要保存前一位 pre,然后分类，
26      前一位如果是 6 那么这意味就不能是 2，这里一定要保存枚举的这个数是合法*/
27    }
28    //计算完，记录状态
29    if(!limit && !lead) dp[pos][state]=ans;
30    /*这里对应上面的记忆化，在一定条件下时记录，保证一致性，当然如果约束条件不需要考虑 lead，这里就
31  是 lead 就完全不用考虑了*/
32    return ans;
33  }
```

```
ll solve(ll x)
{
    int pos=0;
    while(x)//把数位都分解出来
    {
        a[pos++]=x%10;//个人老是喜欢编号为[0,pos),看不惯的就按自己习惯来，反正注意数位边界就行
        x/=10;
    }
    return dfs(pos-1/*从最高位开始枚举*/,/*一系列状态 */,true,true);//刚开始最高位都是有限制并且有前导零的，显然比最高位还要高的一位视为 0 嘛
}
int main()
{
    ll le,ri;
    while(~scanf("%lld%lld",&le,&ri))
    {
        //初始化 dp 数组为-1,这里还有更加优美的优化,后面讲
        printf("%lld\n",solve(ri)-solve(le-1));
    }
}
```

六、串

1. 最长回文串(Manacher 算法 O(n))

```
void Manacher(char s[],int len) {//原字符串和串长
        int l = 0;
        String[l++] = '$'; // 0 下标存储为其他字符,防止越界
        String[l++] = '#';
        for (int i = 0; i < len; i++) {
            String[l++] = s[i];
            String[l++] = '#';
        }
        String[l] = 0; //  空字符
        int MaxR = 0;
        int flag = 0;
        for (int i = 0; i < l; i++) {
            cnt[i] = MaxR > i ? min(cnt[2 * flag - i], MaxR - i) : 1;//2*flag-i 是 i 点关于 flag 的对称点
            while (String[i + cnt[i]] == String[i - cnt[i]])
                cnt[i]++;
            if (i + cnt[i] > MaxR) {
```

```
17                      MaxR = i + cnt[i];
18                      flag = i;
19              }
20      }
21 }
22 /*
23 * String: $ # a # b # a # a # b # a # 0
24 * cnt:       1 1 2 1 4 1 2 7 2 1 4 1 2 1
25 */
```

2.模式匹配(KMP 算法(O(n+m)))

```
1   void Find_Next(char s2[]){
2           Next[0] = -1;
3           Next[1] = 0;
4           int cnt = 0;
5           int leng = strlen(s2);
6           int i = 2;
7           while(i<=leng){
8                   if(s2[i-1]==s2[cnt])
9                           Next[i++] = ++ cnt;
10                  else if(cnt>0)
11                          cnt = Next[cnt];
12                  else
13                          Next[i++] = 0;
14          }
15  }
16
17  int kmp(char s1[],char s2[],int x,int y){
18          int i,j;//i,x->s1      y,j->s2
19          i = j = 0;
20          if(x < y)
21                  return -1;
22          Find_Next(s2);
23          while(i<x&&j<y){
24                  if(j==-1||s1[i]==s2[j])
25                          i++,j++;
26                  else
27                          j = Next[j];
28          }
29          if(j==y)
30                  return i-j+1;
31          else
```

```
32              return -1;
33 }
```

3. 字典树(O(logn))

```
1  Int Tire[maxn][26];
2  char str[2000005];
3  bool v[maxn];
4  string s;
5  int cnt = 1;
6  //建树,每输入一个单词到 s 里面就调用_insert()就好
7  void _insert(){
8          int root = 0;
9          fori(s.size()){
10             int next = s[i] - 'A';
11             if(!Tire[root][next])
12                 Tire[root][next] = ++cnt;
13             root = Tire[root][next];
14         }
15         v[root] = true;//这里用了一个标记数组表示该点存在一个完整的单词,比如说`app`和`apple`
16 }
17 //查找最长公共前缀
18 int _find(char bufs[],int leng){
19         int root = 0;
20         int cns = 0;
21         int next;
22         int res = 0;
23         fori(leng){
24             next = bufs[i] - 'A';
25             if(Tire[root][next] == 0)
26                 break;
27             root = Tire[root][next];
28             cns++;
29             if(v[root])
30                 res = cns;
31         }
32         return res;
33 }
```

3.AC 自动机(O(strlen))

```
int trie[maxn][26]; //字典树
int cntword[maxn];   //记录该单词出现次数
int fail[maxn];        //失败时的回溯指针
int cnt = 0;

void insertWords(string s){
        int root = 0;
        for(int i=0;i<s.size();i++){
                int next = s[i] - 'a';
                if(!trie[root][next])
                        trie[root][next] = ++cnt;
                root = trie[root][next];
        }
        cntword[root]++;          //当前节点单词数+1
}
void getFail(){
        queue <int>q;
        for(int i=0;i<26;i++){          //将第二层所有出现了的字母扔进队列
                if(trie[0][i]){
                        fail[trie[0][i]] = 0;
                        q.push(trie[0][i]);
                }
        }
        while(!q.empty()){
                int now = q.front();
                q.pop();

                for(int i=0;i<26;i++){          //查询 26 个字母
                        if(trie[now][i]){
                                fail[trie[now][i]] = trie[fail[now]][i];
                                q.push(trie[now][i]);
                        }
                        else//否则就让当前节点的这个子节点
                                //指向当前节点 fail 指针的这个子节点
                                trie[now][i] = trie[fail[now]][i];
                }
        }
}
int query(string s){
        int now = 0,ans = 0;
        for(int i=0;i<s.size();i++){       //遍历文本串
```

```
42              now = trie[now][s[i]-'a'];
43              for(int j=now;j && cntword[j]!=-1;j=fail[j]){
44                  //一直向下寻找,直到匹配失败(失败指针指向根或者当前节点已找过).
45                  ans += cntword[j];
46                  cntword[j] = -1;       //将遍历过后的节点标记,防止重复计算
47              }
48      }
49      return ans;
50 }
51 int main() {
52      int n;
53      string s;
54      cin >> n;
55      for(int i=0;i<n;i++){
56              cin >> s ;
57              insertWords(s);
58      }
59      fail[0] = 0;
60      getFail();
61      cin >> s ;
62      cout << query(s) << endl;
63      return 0;
64 }
```

七、博弈

1. 常用博弈

巴什博弈

问题模型：只有一堆 n 个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取 m 个，最后取光者得胜。
结论：n%(m+1)!=0 时，先手赢，否则后手赢

威佐夫博奕

问题模型：有两堆各若干个物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

```
1 if(num1 > num2)
2       swap(num1, num2);
3 tmp = floor((num2 - num1) * (1 + sqrt(5.0)) / 2.0); //黄金分割
4 if(tmp == num1)
5       printf("Lose\n"); //奇异局势必败
6 else
7       printf("Win\n");
```

尼姆博弈

问题模型：有任意堆石子，每堆石子个数也是任意的，双方轮流从中取出石子，规则如下：
1)每一步应取走至少一枚石子，每一步只能从某一堆中取走部分或全部石子；
2)如果谁取到最后一枚石子就胜

```
1  for( i = 0; i < n; i++ )
2       cin >> temp[ i ]; //第 i 堆物品的数量
3  min = temp[ 0 ];
4  for( i = 1; i < n ; i++ )
5       min = min^temp[ i ]; //按位异或
6  if( min == 0 )
7       cout << "Lose" << endl; //输
8  else
9       cout << "Win" << endl; //赢
```

斐波那契博弈
每次至少取 1,至多取上次的 2 倍,若 n 为斐波那契数列,则后手胜;

八.组合数

卡特兰数($h(n)=h(n-1)*(4n-2)/(n+1)$)

前 5 项:1,1,2,5,14.

(1).矩阵连乘： P=a1×a2×a3×……×an，依据乘法结合律，不改变其顺序，只用括号表示成对的乘积，试问有几种括号化的方案?

(2).一个栈(无穷大)的进栈序列为 1，2，3，…，n，有多少个不同的出栈序列?

(3).在一个凸多边形中，通过若干条互不相交的对角线，把这个多边形划分成了若干个三角形。任务是键盘上输入凸多边形的边数 n，求不同划分的方案数

(4).给定 N 个节点，能构成多少种不同的二叉搜索树？

(5).给定 n 对括号，求括号正确配对的字符串数

第一类 Stirling 数($s[n][k] = (n-1) * s[n-1][k] + s[n-1][k-1]$)

s[n][0] = 0,s[n][n] = 1

(1).将 n 个物体排成 k 个非空循环排列(非空的)的方法数

第二类 Stirling 数（$s[n][k] = s[n-1][k-1] + s[n-1][k] * k$）

(1). n 个元素划分成 k 个无序集合的方案数

Bell 数(B[n]= $\sum_{k=1}^{n} S[n][k]$**）(其中 S 为第二类 Stirling 数)**

包含 n 个元素的集合的划分方法的数目。

那罗延数（ $N(n)(k) = \frac{1}{n} c_n^k c_n^{k-1} C_n^k$ **）**

(1).在由 n 对"("和")"组成的字符串中，共有 k 对"("与")"相邻，这样的字符串一共有 N(n,k)个

莫慈金数（M（n+1）=(2n+3)M(n)+3nM(n-1)）/(n+3))

前五项：1，2，4，9，21

(1).在一个圆上的 n 个点间，画出彼此不相交的弦的全部方法的总数

卢卡斯定理（大组合数求模）

$$C_n^m \%p = C_{\frac{n}{p}}^{\frac{m}{p}} * C_{n\%p}^{m\%p}$$

康托展开

已知一个排列,求这个排列在全排列中是第几个 x = a[n]*(n-1)！+...+a[1]*0!中 a[i]为当前未出现的元素中是排在第几个（从 0 开始）

```
LL Work(char str[])
{
    int len = strlen(str);
    LL ans = 0;
    for(int i=0; i<len; i++)
    {
        int tmp = 0;
        for(int j=i+1; j<len; j++)
            if(str[j] < str[i]) tmp++;
        ans += tmp * f[len-i-1];   //f[]为阶乘
    }
    return ans;   //返回该字符串是全排列中第几大，从 1 开始
}
```

平面划分问题

平面分割空间	$f(n) = (n^3 + 5n)/6 + 1$
封闭曲线分平面	$f(n) = n^2 - n + 2$
三角形划分区域	$f(n) = 3n(n-1) + 2$
折线分平面	$f(n) = 2n^2 - n + 1$
n 条直线分平面	$f(n) = n(n+1)/2 + 1$

错排（$s[n] = (n-1)(s[n-1] + s[n-2])$）

$(s[1]=0, s[2]=1)$

基姆拉尔森公式(计算日期)

```
void CalculateWeekDay(int y, int m,int d) {
    if(m==1||m==2)
        m+=12,y--;
    int iWeek = (d+2*m+3*(m+1)/5+y+y/4-y/100+y/400)%7;
//iWeek0-6 分别代表星期一,星期二....星期天
}
```

九.数据结构

1. 并查集

```
void init() { //初始化
	for(int i=1; i<=n; i++) {
		pre[i]=i;
		rank[i]=0;	///权值
	}
}
//递归法压缩路径
int find(int x) {
	if(x!=pre[x])
		pre[x]=find(pre[x]);
	return pre[x];
}
//普通合并
void combine(int x,int y) {
	int fx=find(x);
	int fy=find(y);
	if(x > y)
		swap(x,y);//默认合并到小的中
	if(fx!=fy)
		pre[fx]=fy;
}
//考虑 rank 合并
void combine(int a,int b) {
	int fa=find(a);
	int fb=find(b);
	if(rank[fa]>rank[fb])
		pre[fb]=fa;
	else
		pre[fa]=fb;
	if(rank[fa]==rank[fb])
		rank[fb]++;
}
```

2. 线段树

```
struct Node
{
```

```
3         int l,r,w,flag;
4         Node() {};
5         Node(int _l,int _r,int _v,int _f){flag=_f,l=_l,r=_r,w=_v;}
6         int mid(){return (l+r)/2;}
7   };
8   Node a[maxn<<2];
9   void build(int k,int l,int r) {                    //建树
10        a[k] = Node(l,r,0,0);
11        if(a[k].l == a[k].r){
12              cin >> a[k].w;
13              return;
14        }
15        build(k<<1,l,(l+r)/2);
16        build(k<<1|1,(l+r)/2+1,r);
17        a[k].w = a[k<<1].w+a[k<<1|1].w;
18  }
19
20  void down(int k) {                                 //延迟标记下传
21        a[k<<1].flag += a[k].flag;
22        a[k<<1|1].flag += a[k].flag;
23        a[k<<1].w += a[k].flag*(a[k<<1].r-a[k<<1].l+1);
24        a[k<<1|1].w += a[k].flag *(a[k<<1|1].r-a[k<<1|1].l+1);
25        a[k].flag = 0;
26  }
27  int res;
28
29  void update(int k,int x,int y,int z) {             //区间更新
30        if(a[k].l>=x &&a[k].r<=y) {
31              a[k].w += z*(a[k].r-a[k].l+1);
32              a[k].flag += z;
33              return;
34        }
35        if(a[k].flag)
36              down(k);
37        if(x <= a[k].mid())
38              update(k<<1,x,y,z);
39        if(y > a[k].mid())
40              update(k<<1|1,x,y,z);
41        a[k].w = a[k<<1].w+a[k<<1|1].w;
42  }
43  int ans;
44  void query(int k,int x,int y) {                    //区间查询
45        if(a[k].l>=x && a[k].r<=y) {
46              res += a[k].w;
```

```
47              return;
48      }
49      if(a[k].flag)
50              down(k);
51      if(x <= a[k].mid())
52              query(k<<1,x,y);
53      if(y > a[k].mid())
54              query(k<<1|1,x,y);
55      a[k].w = a[k<<1].w+a[k<<1|1].w;
56  }
```

3. 一维树状数组

单点更新+区间查询

```
1   void update(int x,int y,int n){
2           for(int i=x;i<=n;i+=lowbit(i))
3                   c[i] += y;
4   }
5   int getsum(int x){
6           int ans = 0;
7           for(int i=x;i;i-=lowbit(i))
8                   ans += c[i];
9           return ans;
10  }
```

区间更新+单点查询

```
1   void add(int p, int x){ //这个函数用来在树状数组中直接修改
2           while(p <= n) sum[p] += x, p += p & -p;
3   }
4   void range_add(int l, int r, int x){ //给区间[l, r]加上 x
5           add(l, x), add(r + 1, -x);
6   }
7   int ask(int p){ //单点查询
8           int res = 0;
9           while(p) res += sum[p], p -= p & -p;
10          return res;
11  }
```

区间更新+区间查询

```
1   void add(ll p, ll x){
2           for(int i = p; i <= n; i += i & -i)
3                   sum1[i] += x, sum2[i] += x * p;
```

```
}
void range_add(ll l, ll r, ll x){
        add(l, x), add(r + 1, -x);
}
ll ask(ll p){
        ll res = 0;
        for(int i = p; i; i -= i & -i)
                res += (p + 1) * sum1[i] - sum2[i];
        return res;
}
ll range_ask(ll l, ll r){
        return ask(r) - ask(l - 1);
}
```

4. 二维树状数组

单点更新+区间查询

```
void add(int x, int y, int z){ //将点(x, y)加上 z
        int memo_y = y;
        while(x <= n){
                y = memo_y;
                while(y <= n)
                        tree[x][y] += z, y += y & -y;
                x += x & -x;
        }
}
void ask(int x, int y){//求左上角为(1,1)右下角为(x,y)  的矩阵和
        int res = 0, memo_y = y;
        while(x){
                y = memo_y;
                while(y)
                        res += tree[x][y], y -= y & -y;
                x -= x & -x;
        }
}
```

区间更新+单点查询

```
void add(int x, int y, int z){
```

```
2         int memo_y = y;
3         while(x <= n){
4                 y = memo_y;
5                 while(y <= n)
6                         tree[x][y] += z, y += y & -y;
7                 x += x & -x;
8         }
9   }
10  void range_add(int xa, int ya, int xb, int yb, int z){
11        add(xa, ya, z);
12        add(xa, yb + 1, -z);
13        add(xb + 1, ya, -z);
14        add(xb + 1, yb + 1, z);
15  }
16  void ask(int x, int y){
17        int res = 0, memo_y = y;
18        while(x){
19                y = memo_y;
20                while(y)
21                        res += tree[x][y], y -= y & -y;
22                x -= x & -x;
23        }
24  }
```

区间更新+区间查询

```
1   #include <cstdio>
2   #include <cmath>
3   #include <cstring>
4   #include <algorithm>
5   #include <iostream>
6   using namespace std;
7   typedef long long ll;
8   ll read(){
9         char c; bool op = 0;
10        while((c = getchar()) < '0' || c > '9')
11                if(c == '-') op = 1;
12        ll res = c - '0';
13        while((c = getchar()) >= '0' && c <= '9')
14                res = res * 10 + c - '0';
15        return op ? -res : res;
16  }
17  const int N = 205;
18  ll n, m, Q;
19  ll t1[N][N], t2[N][N], t3[N][N], t4[N][N];
```

```
20 void add(ll x, ll y, ll z){
21         for(int X = x; X <= n; X += X & -X)
22                 for(int Y = y; Y <= m; Y += Y & -Y){
23                         t1[X][Y] += z;
24                         t2[X][Y] += z * x;
25                         t3[X][Y] += z * y;
26                         t4[X][Y] += z * x * y;
27                 }
28 }
29 void range_add(ll xa, ll ya, ll xb, ll yb, ll z){ //(xa, ya) 到 (xb, yb) 的矩形
30         add(xa, ya, z);
31         add(xa, yb + 1, -z);
32         add(xb + 1, ya, -z);
33         add(xb + 1, yb + 1, z);
34 }
35 ll ask(ll x, ll y){
36         ll res = 0;
37         for(int i = x; i; i -= i & -i)
38                 for(int j = y; j; j -= j & -j)
39                         res += (x + 1) * (y + 1) * t1[i][j]
40                                 - (y + 1) * t2[i][j]
41                                 - (x + 1) * t3[i][j]
42                                 + t4[i][j];
43         return res;
44 }
45 ll range_ask(ll xa, ll ya, ll xb, ll yb){
46         return ask(xb, yb) - ask(xb, ya - 1) - ask(xa - 1, yb) + ask(xa - 1, ya - 1);
47 }
48 int main(){
49         n = read(), m = read(), Q = read();
50         for(int i = 1; i <= n; i++){
51                 for(int j = 1; j <= m; j++){
52                         ll z = read();
53                         range_add(i, j, i, j, z);
54                 }
55         }
56         while(Q--){
57                 ll ya = read(), xa = read(), yb = read(), xb = read(), z = read(), a = read();
58                 if(range_ask(xa, ya, xb, yb) < z * (xb - xa + 1) * (yb - ya + 1))
59                         range_add(xa, ya, xb, yb, a);
60         }
61         for(int i = 1; i <= n; i++){
62                 for(int j = 1; j <= m; j++)
63                         printf("%lld ", range_ask(i, j, i, j));
```

```
64              putchar('\n');
65      }
66      return 0;
67 }
68
```

5.划分树

```
1   #define _mid(a,b) ((a+b)/2)
2   using namespace std;
3   typedef long long ll;
4   const int maxn =    1e5+10;
5   const int INF = 0x3f3f3f3f;
6   int sorted[maxn];
7   int cnt[20][maxn];
8   int tree[20][maxn];
9   void build(int l,int r,int k){
10      if(r==l)                                                    //如果区间内只有一个数，返回
11              return;
12      int mid = _mid(l,r),flag = mid-l+1;                         //求出 flag
13      for(int i=l;i<=r;i++)
14              if(tree[k][i] < sorted[mid])    //sorted 代表排序好了的数组
15                      flag -- ;
16      int bufl = l,bufr = mid+1;
17      for(int i=l;i<=r;i++){
18              cnt[k][i] = (i==l)?0:cnt[k][i-1];                   //初始化
19              if(tree[k][i]<sorted[mid] || tree[k][i]==sorted[mid]&&flag>0){//如果有多个中值
20                      tree[k+1][bufl++] = tree[k][i];
21                      cnt[k][i]++;                                //进入左子树
22                      if(tree[k][i] == sorted[mid])
23                              flag--;
24              }
25              else                                                //进入右子树
26                      tree[k+1][bufr++] = tree[k][i];
27      }
28      build(l,mid,k+1);
29      build(mid+1,r,k+1);
30  }
31
32  int ask(int k,int sl,int sr,int l,int r,int x){
33      if(sl==sr)
34              return tree[k][sl];
```

```
35      int cntl;
36      cntl = (l==sl)?0:cnt[k][l-1];       //是否和查询区间重合
37      int cntl2r = cnt[k][r]-cntl;        //计算 l 到 r 有 cntl2r 个数进入左子树
38      if(cntl2r >= x)         //如果大于当前查询的 k 则进入左子树（因为左子树中最大的数大于第 k 大的数）
39          return ask(k+1,sl,_mid(sl,sr),sl+cntl,sl+cnt[k][r]-1,x);
40      else{                                       //否则进入右子树
41          int lr = _mid(sl,sr) + 1 + (l-sl-cntl);
42          return ask(k+1,_mid(sl,sr)+1,sr,lr,lr+r-l-cntl2r,x-cntl2r);
43      }
44  }
```

6. RMQ

```
1   void ST(int n) {
2       for (int i = 1; i <= n; i++)
3           dp[i][0] = A[i];
4       for (int j = 1; (1 << j) <= n; j++) {
5           for (int i = 1; i + (1 << j) - 1 <= n; i++) {
6               dp[i][j] = max(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
7           }
8       }
9   }
10  int RMQ(int l, int r) {
11      int k = 0;
12      while ((1 << (k + 1)) <= r - l + 1) k++;
13      return max(dp[l][k], dp[r - (1 << k) + 1][k]);
14  }
```

十.数学

1. 解方程

1.牛顿迭代： $X_0=X_0-F(X_0)/F'(X_0)$（其中 $F'(x)$ 为 $F(x)$的导数）

2.二分迭代。

3.高斯消元

```
1   #include<stdio.h>
2   #include<algorithm>
3   #include<iostream>
4   #include<string.h>
```

```
#include<math.h>
using namespace std;
const int MAXN=50;
int a[MAXN][MAXN];//增广矩阵
int x[MAXN];//解集
bool free_x[MAXN];//标记是否是不确定的变元
inline int gcd(int a,int b)
{
    int t;
    while(b!=0)
    {
        t=b;
        b=a%b;
        a=t;
    }
    return a;
}
inline int lcm(int a,int b)
{
    return a/gcd(a,b)*b;//先除后乘防溢出
}

// 高斯消元法解方程组(Gauss-Jordan elimination).(-2 表示有浮点数解，但无整数解，
//-1 表示无解，0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数)
//有 equ 个方程，var 个变元。增广矩阵行数为 equ,分别为 0 到 equ-1,列数为 var+1,分别为 0 到 var.
int Gauss(int equ,int var)
{
    int i,j,k;
    int max_r;// 当前这列绝对值最大的行.
    int col;//当前处理的列
    int ta,tb;
    int LCM;
    int temp;
    int free_x_num;
    int free_index;

    for(int i=0; i<=var; i++)
    {
        x[i]=0;
        free_x[i]=true;
    }

    //转换为阶梯阵.
    col=0; // 当前处理的列
```

```
49      for(k = 0; k < equ && col < var; k++,col++)
50      {
51          // 枚举当前处理的行.
52  // 找到该 col 列元素绝对值最大的那行与第 k 行交换.(为了在除法时减小误差)
53          max_r=k;
54          for(i=k+1; i<equ; i++)
55          {
56              if(abs(a[i][col])>abs(a[max_r][col])) max_r=i;
57          }
58          if(max_r!=k)
59          {
60              // 与第 k 行交换.
61              for(j=k; j<var+1; j++) swap(a[k][j],a[max_r][j]);
62          }
63          if(a[k][col]==0)
64          {
65              // 说明该 col 列第 k 行以下全是 0 了，则处理当前行的下一列.
66              k--;
67              continue;
68          }
69          for(i=k+1; i<equ; i++)
70          {
71              // 枚举要删去的行.
72              if(a[i][col]!=0)
73              {
74                  LCM = lcm(abs(a[i][col]),abs(a[k][col]));
75                  ta = LCM/abs(a[i][col]);
76                  tb = LCM/abs(a[k][col]);
77                  if(a[i][col]*a[k][col]<0)tb=-tb;      //异号的情况是相加
78                  for(j=col; j<var+1; j++)
79                  {
80                      a[i][j] = a[i][j]*ta-a[k][j]*tb;
81                  }
82              }
83          }
84      }
85
86      // 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).
87      for (i = k; i < equ; i++)
88      {
89          // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交
90  换.
91          if (a[i][col] != 0) return -1;
92      }
```

```
93      // 2. 无穷解的情况: 在 var * (var + 1)的增广阵中出现(0, 0, ..., 0)这样的行，即说明没有形成严格的上三角
94  阵.
95      // 且出现的行数即为自由变元的个数.
96      if (k < var)
97      {
98          // 首先，自由变元有 var - k 个，即不确定的变元至少有 var - k 个.
99          for (i = k - 1; i >= 0; i--)
100         {
101             // 第 i 行一定不会是(0, 0, ..., 0)的情况，因为这样的行是在第 k 行到第 equ 行.
102             // 同样，第 i 行一定不会是(0, 0, ..., a), a != 0 的情况，这样的无解的.
103             free_x_num = 0; // 用于判断该行中的不确定的变元的个数，如果超过 1 个，则无法求解，它们
104 仍然为不确定的变元.
105             for (j = 0; j < var; j++)
106             {
107                 if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
108             }
109             if (free_x_num > 1) continue; // 无法求解出确定的变元.
110             // 说明就只有一个不确定的变元 free_index，那么可以求解出该变元，且该变元是确定的.
111             temp = a[i][var];
112             for (j = 0; j < var; j++)
113             {
114                 if (a[i][j] != 0 && j != free_index) temp -= a[i][j] * x[j];
115             }
116             x[free_index] = temp / a[i][free_index]; // 求出该变元.
117             free_x[free_index] = 0; // 该变元是确定的.
118         }
119         return var - k; // 自由变元有 var - k 个.
120     }
121     // 3. 唯一解的情况: 在 var * (var + 1)的增广阵中形成严格的上三角阵.
122     // 计算出 Xn-1, Xn-2 ... X0.
123     for (i = var - 1; i >= 0; i--)
124     {
125         temp = a[i][var];
126         for (j = i + 1; j < var; j++)
127         {
128             if (a[i][j] != 0) temp -= a[i][j] * x[j];       //--因为 x[i]存的是 temp/a[i][i]的值，即是 a[i][i]=1 时 x[i]对
129 应的值
130         }
131         if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.
132         x[i] = temp / a[i][i];
133     }
134     return 0;
135 }
136 int main(void)
```

```
{
    freopen("in.txt", "r", stdin);
    freopen("out.txt","w",stdout);
    int i, j;
    int equ,var;
    while (scanf("%d %d", &equ, &var) != EOF)
    {
        memset(a, 0, sizeof(a));
        for (i = 0; i < equ; i++)
        {
            for (j = 0; j < var + 1; j++)
            {
                scanf("%d", &a[i][j]);
            }
        }
        int free_num = Gauss(equ,var);
        if (free_num == -1) printf("无解!\n");
        else if (free_num == -2) printf("有浮点数解，无整数解!\n");
        else if (free_num > 0)
        {
            printf("无穷多解! 自由变元个数为%d\n", free_num);
            for (i = 0; i < var; i++)
            {
                if (free_x[i]) printf("x%d  是不确定的\n", i + 1);
                else printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        else
        {
            for (i = 0; i < var; i++)
            {
                printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        printf("\n");
    }
    return 0;
}
```

2. 矩阵快速幂

```
#include<iostream>
#include<cstdio>
```

```
#include<cstring>
using namespace std;

const int maxn = 2;
const int mod = 10000;
//矩阵结构体
struct Matrix{
    int a[maxn][maxn];
    void init(){    //初始化为单位矩阵
        memset(a, 0, sizeof(a));
        for(int i=0;i<maxn;++i){
            a[i][i] = 1;
        }
    }
};
//矩阵乘法
Matrix mul(Matrix a, Matrix b){
    Matrix ans;
    for(int i=0;i<maxn;++i){
        for(int j=0;j<maxn;++j){
            ans.a[i][j] = 0;
            for(int k=0;k<maxn;++k){
                ans.a[i][j] += a.a[i][k] * b.a[k][j];
                ans.a[i][j] %= mod;
            }
        }
    }
    return ans;
}

//矩阵快速幂
Matrix qpow(Matrix a, int n){
    Matrix ans;
    ans.init();
    while(n){
        if(n&1) ans = mul(ans, a);
        a = mul(a, a);
        n /= 2;
    }
    return ans;
}

int main(){
    Matrix a;
```

```
47        a.a[0][0] = 1;
48        a.a[0][1] = 1;
49        a.a[1][0] = 1;
50        a.a[1][1] = 0;
51
52        Matrix ans = qpow(a, 10);
53        return 0;
54  }
```

十一.其他

1. 常用函数及语句

```
1   string sub2 = s.substr(5, 3); //从下标为 5 开始截取长度为 3 位：sub2 = "567"
2   string sub1 = s.substr(5);
3   //只有一个数字 5 表示从下标为 5 开始一直到结尾：sub1 = "56789"
4   prev_permutation(a,a+a.size())              //求数组 a 的上一个排列
5   next_permutation(a,a+a.size())              //求数组 a 的下一个排列
6   lower_bound(a,a+a.size(),x)                 //二分(查不到时偏小)
7   upper_bound(a,a+a.size(),x)                 //二分(偏大)
8   fill(a,a+n,x)                               //a[0]~a[n-1]填充 x
9   reverse(a+x,a+y)                            //反转区间(x,y)
10 __builtin_popcount(x)                        //x 二进制中'1'的个数
11 freopen("input.txt","r",stdin);
12 freopen("output.txt", "w", stdout);
13 #define IO do{\
14    ios::sync_with_stdio(false);\
15    cin.tie(0);\
16    cout.tie(0);}while(0)
17 struct cmp{
18        bool operator ()(DateType &a,DateType &b) {    //重载
19                ……
20        }
21  }
22
```

2. 数据类型取值范围

	取值范围
Char	-128~127(大约 3 位)

Short	-32768~32767(大约 5 位)
unsigned short	**0 ~ 65536 (2 Bytes**，大大约五位**)**
Int	**2147483648 ~ 2147483647**（**4 Bytes**，大约十位）
unsigned int	**0 ~ 4294967295 (4 Bytes**，大大约十十位**)**
long	相当于 **int**
long long	**-9223372036854775808 ~ 9223372036854775807 (8 Bytes**，大大约十十九位**)**
unsigned long long	**0 ~ 18446744073709551615**（大大约二二十十位）
double	**1.7 * 10^308 (8 Bytes)**

3. c++大数

```
1   #include <iostream>
2   #include <string>
3   #include <cstring>
4   #include <cstdio>
5   using namespace std;
6
7   const int maxn = 1000;
8
9   struct bign {
10      int d[maxn], len;
11  // 去掉大数的前导 0
12      void clean() {
13          while(len > 1 && !d[len-1])
14              len--;
15      }
16  //默认构造为 0
17      bign(){
18          memset(d, 0, sizeof(d));
19          len = 1;
20      }
21  // 初始化：可以用“bign [bign] = [int];”或“bign [bign]([int]);”
22      bign(int num)   {
23          *this = num;
24      }
25  // 初始化：可以用“bign [bign] = [char*];”或“bign [bign](char*);”
26
27      bign(char* num) {
28          *this = num;
29      }
```

```
30  // 赋值：可以用“[bign] = [char*];”
31      bign operator = (const char* num) {
32          memset(d, 0, sizeof(d));
33          len = strlen(num);
34          for(int i = 0; i < len; i++)
35              d[i] = num[len-1-i] - '0';
36          clean();
37          return *this;
38      }
39  // 赋值：可以用“[bign] = [int];”
40      bign operator = (int num) {
41          char s[20];
42          sprintf(s, "%d", num);
43          *this = s;
44          return *this;
45      }
46
47      bign operator + (const bign& b) {
48          bign c = *this;
49          int i;
50          for (i = 0; i < b.len; i++) {
51              c.d[i] += b.d[i];
52              if (c.d[i] > 9)
53                  c.d[i]%=10, c.d[i+1]++;
54          }
55          while (c.d[i] > 9)
56              c.d[i++]%=10, c.d[i]++;
57          c.len = max(len, b.len);
58          if (c.d[i] && c.len <= i)
59              c.len = i+1;
60          return c;
61      }
62      bign operator - (const bign& b) {
63          bign c = *this;
64          int i;
65          for (i = 0; i < b.len; i++) {
66              c.d[i] -= b.d[i];
67              if (c.d[i] < 0)
68                  c.d[i]+=10, c.d[i+1]--;
69          }
70          while (c.d[i] < 0)
71              c.d[i++]+=10, c.d[i]--;
72          c.clean();
73          return c;
```

```
74      }
75      bign operator * (const bign& b)const {
76          int i, j;
77          bign c;
78          c.len = len + b.len;
79          for(j = 0; j < b.len; j++)
80              for(i = 0; i < len; i++)
81                  c.d[i+j] += d[i] * b.d[j];
82          for(i = 0; i < c.len-1; i++)
83              c.d[i+1] += c.d[i]/10, c.d[i] %= 10;
84          c.clean();
85          return c;
86      }
87      bign operator / (const bign& b) {
88          int i, j;
89          bign c = *this, a = 0;
90          for (i = len - 1; i >= 0; i--) {
91              a = a*10 + d[i];
92              for (j = 0; j < 10; j++)
93                  if (a < b*(j+1))
94                      break;
95              c.d[i] = j;
96              a = a - b*j;
97          }
98          c.clean();
99          return c;
100     }
101     bign operator % (const bign& b) {
102         int i, j;
103         bign a = 0;
104         for (i = len - 1; i >= 0; i--) {
105             a = a*10 + d[i];
106             for (j = 0; j < 10; j++)
107                 if (a < b*(j+1))
108                     break;
109             a = a - b*j;
110         }
111         return a;
112     }
113
114     bign operator += (const bign& b) {
115         *this = *this + b;
116         return *this;
117     }
```

```
118
119         bool operator <(const bign& b) const {
120             if(len != b.len)
121                 return len < b.len;
122             for(int i = len-1; i >= 0; i--)
123                 if(d[i] != b.d[i])
124                     return d[i] < b.d[i];
125             return false;
126         }
127         bool operator >(const bign& b) const {
128             return b < *this;
129         }
130         bool operator<=(const bign& b) const {
131             return !(b < *this);
132         }
133         bool operator>=(const bign& b) const {
134             return !(*this < b);
135         }
136         bool operator!=(const bign& b) const {
137             return b < *this || *this < b;
138         }
139         bool operator==(const bign& b) const {
140             return !(b < *this) && !(b > *this);
141         }
142     // 将 int 数组存储的值转换为高精度的字符串形式
143         string str() const {
144             char s[maxn]= {};
145             for(int i = 0; i < len; i++)
146                 s[len-1-i] = d[i]+'0';
147             return s;
148         }
149     };
150
151     istream& operator >> (istream& in, bign& x) {
152         string s;
153         in >> s;
154         x = s.c_str();
155         return in;
156     }
157
158     ostream& operator << (ostream& out, const bign& x) {
159         out << x.str();
160         return out;
161     }
```

```
162
163  int main() {
164          bign a,b;
165          cin >>a    >> b;
166          cout << (a/b) << endl;
167  }
```