Contents

1 Basic

1.1 default code

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;
#define IO_FAST \
    cin.tie(0); \
    ios::sync_with_stdio(0)
#define ll long long
#define LL long long
#define ld long double
#define EPS 1e-8
#define cl(x) ((x << 1))
#define cr(x) ((x << 1) + 1)
#define FZ(x) memset(x, 0, sizeof(x))
#define lowbit(x) (x & -x)
#define INF 0x3f3f3f3f
#define INFLL 0x3f3f3f3f3f3f3f3f
#define endl '\n'
#define pii pair<int,int>
#define pll pair<ll,ll>
#define vi vector<int>
#define vl vector<ll>
#define MP make_pair
#define pi acos(-1)
#define ALL(a) a.begin(), a.end()
#define SZ(x) ((int)x.size())
#define asort(a) sort(a.begin(), a.end())
                    //升冪排序
#define dsort(a) sort(a.begin(), a.end(), greater<int
    >()) //降冪排序           //升冪排序
#define dsortll(a) sort(a.begin(), a.end(), greater<ll
    >()) //降冪排序
#define PB push_back
const int mod=1e8;
int gcd(int a,int b){//輾轉相除法
    while(b){
        int tmp=b;
        b=a%b;
        a=tmp;
    }
    return a;
}
ll qpow(ll base,ll exp){//快速冪
    ll rec=1;
    while(exp){
        if(exp&1)rec*=base;
        base*=base;
        exp>>=1;
    }
    return rec;
}
```

2 flow

2.1 Dinic

```
#define PB push_back
#define SZ(x) (int)x.size()
struct Dinic
{
  struct Edge
  {
    int v, f, re;
  };
  int n, s, t, level[MXN];
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t)
  {
    n = _n;
    s = _s;
    t = _t;
    for (int i = 0; i < n; i++)
      E[i].clear();
  }
  void add_edge(int u, int v, int f)
  {
    E[u].PB({v, f, SZ(E[v])});
    E[v].PB({u, 0, SZ(E[u]) - 1});
  }
  bool BFS()
  {
    for (int i = 0; i < n; i++)
      level[i] = -1;
    queue<int> que;
    que.push(s);
    level[s] = 0;
    while (!que.empty())
    {
      int u = que.front();
      que.pop();
      for (int it=0;it<E[u].size();it++)
      {
        if (E[u][it].f > 0 && level[E[u][it].v] == -1)
        {
          level[E[u][it].v] = level[u] + 1;
          que.push(E[u][it].v);
        }
      }
    }
    return level[t] != -1;
  }
  int DFS(int u, int nf)
  {
    if (u == t)
      return nf;
    int res = 0;
    for (int it=0;it<E[u].size();it++)
    {
      if (E[u][it].f > 0 && level[E[u][it].v] == level[
          u] + 1)
      {
        int tf = DFS(E[u][it].v, min(nf, E[u][it].f));
        res += tf;
        nf -= tf;
        E[u][it].f -= tf;
        E[E[u][it].v][E[u][it].re].f += tf;
        if (nf == 0)
          return res;
      }
    }
    if (!res)
```

```
      level[u] = -1;
    return res;
  }
  int flow(int res = 0)
  {
    while (BFS())
      res += DFS(s, 2147483647);
    return res;
  }
}flow;
```

2.2 ISAP

```
#define PB push_back
#define SZ(x) (int)x.size()
struct Maxflow {
  static const int MAXV = 20010;
  static const int INF  = 1000000;
  struct Edge {
    int v, c, r;
    Edge(int _v, int _c, int _r):
      v(_v), c(_c), r(_r) {}
  };
  int s, t;
  vector<Edge> G[MAXV*2];
  int iter[MAXV*2], d[MAXV*2], gap[MAXV*2], tot;
  void init(int x) {
    tot = x+2;
    s = x+1, t = x+2;
    for(int i = 0; i <= tot; i++) {
      G[i].clear();
      iter[i] = d[i] = gap[i] = 0;
  } }
  void addEdge(int u, int v, int c) {
    G[u].push_back(Edge(v, c, SZ(G[v]) ));
    G[v].push_back(Edge(u, 0, SZ(G[u]) - 1));
  }
  int dfs(int p, int flow) {
    if(p == t) return flow;
    for(int &i = iter[p]; i < SZ(G[p]); i++) {
      Edge &e = G[p][i];
      if(e.c > 0 && d[p] == d[e.v]+1) {
        int f = dfs(e.v, min(flow, e.c));
        if(f) {
          e.c -= f;
          G[e.v][e.r].c += f;
          return f;
    } } }
    if( (--gap[d[p]]) == 0) d[s] = tot;
    else {
      d[p]++;
      iter[p] = 0;
      ++gap[d[p]];
    }
    return 0;
  }
  int solve() {
    int res = 0;
    gap[0] = tot;
    for(res = 0; d[s] < tot; res += dfs(s, INF));
    return res;
  }
  void reset() {
    for(int i=0;i<=tot;i++) {
      iter[i]=d[i]=gap[i]=0;
} } }flow;
```

2.3 KM

```
//最大完美匹配
#define PB push_back
#define SZ(x) (int)x.size()
struct KM{ // max weight, for min negate the weights
  int n, mx[MXN], my[MXN], pa[MXN];
  ll g[MXN][MXN], lx[MXN], ly[MXN], sy[MXN];
  bool vx[MXN], vy[MXN];
  void init(int _n) { // 1-based, N個節點
    n = _n;
    for(int i=1; i<=n; i++) fill(g[i], g[i]+n+1, 0);
  }
  void addEdge(int x, int y, ll w) {g[x][y] = w;} //左
      邊的集合節點x連邊右邊集合節點y權重為w
  void augment(int y) {
    for(int x, z; y; y = z)
      x=pa[y], z=mx[x], my[y]=x, mx[x]=y;
  }
  void bfs(int st) {
    for(int i=1; i<=n; ++i) sy[i]=INF, vx[i]=vy[i]=0;
    queue<int> q; q.push(st);
    for(;;) {
      while(q.size()) {
        int x=q.front(); q.pop(); vx[x]=1;
        for(int y=1; y<=n; ++y) if(!vy[y]){
          ll t = lx[x]+ly[y]-g[x][y];
          if(t==0){
            pa[y]=x;
            if(!my[y]){augment(y);return;}
            vy[y]=1, q.push(my[y]);
          }else if(sy[y]>t) pa[y]=x,sy[y]=t;
      } }
      ll cut = INF;
      for(int y=1; y<=n; ++y)
        if(!vy[y]&&cut>sy[y]) cut=sy[y];
      for(int j=1; j<=n; ++j){
        if(vx[j]) lx[j] -= cut;
        if(vy[j]) ly[j] += cut;
        else sy[j] -= cut;
      }
      for(int y=1; y<=n; ++y) if(!vy[y]&&sy[y]==0){
        if(!my[y]){augment(y);return;}
        vy[y]=1, q.push(my[y]);
  } } }
  ll solve(){ // 回傳值為完美匹配下的最大總權重
    fill(mx, mx+n+1, 0); fill(my, my+n+1, 0);
    fill(ly, ly+n+1, 0); fill(lx, lx+n+1, -INF);
    for(int x=1; x<=n; ++x) for(int y=1; y<=n; ++y)
      lx[x] = max(lx[x], g[x][y]);
    for(int x=1; x<=n; ++x) bfs(x);
    ll ans = 0;
    for(int y=1; y<=n; ++y) ans += g[my[y]][y];
    return ans;
} }graph;
```

2.4 MinCostMaxFlow

```
struct MinCostMaxFlow{
  static const int MAXV = 20010;
  static const int INFf = 1000000;
  //int INFc  = 1e9;
  struct Edge{
    int v, cap;
    int w;
    int rev;
    Edge(){}
    Edge(int t2, int t3, int t4, int t5)
    : v(t2), cap(t3), w(t4), rev(t5) {}
  };
  int V, s, t;
  vector<Edge> g[MAXV];
  void init(int n, int _s, int _t){
    V = n; s = _s; t = _t;
    for(int i = 0; i <= V; i++) g[i].clear();
  }
  void addEdge(int a, int b, int cap, int w){
    g[a].push_back(Edge(b, cap, w, (int)g[b].size()));
    g[b].push_back(Edge(a, 0, -w, (int)g[a].size()-1));
  }
  int d[MAXV];
  int id[MAXV], mom[MAXV];
  bool inqu[MAXV];
  queue<int> q;
  pair<int,int> solve(){
    int mxf = 0; int mnc = 0;
    while(1){
      fill(d, d+1+V, 1e9);
      fill(inqu, inqu+1+V, 0);
      fill(mom, mom+1+V, -1);
      mom[s] = s;
      d[s] = 0;
      q.push(s); inqu[s] = 1;
      while(q.size()){
        int u = q.front(); q.pop();
        inqu[u] = 0;
        for(int i = 0; i < (int) g[u].size(); i++){
          Edge &e = g[u][i];
          int v = e.v;
          if(e.cap > 0 && d[v] > d[u]+e.w){
            d[v] = d[u]+e.w;
            mom[v] = u;
            id[v] = i;
            if(!inqu[v]) q.push(v), inqu[v] = 1;
      } } }
      if(mom[t] == -1) break ;
      int df = INFf;
      for(int u = t; u != s; u = mom[u])
        df = min(df, g[mom[u]][id[u]].cap);
      for(int u = t; u != s; u = mom[u]){
        Edge &e = g[mom[u]][id[u]];
        e.cap             -= df;
        g[e.v][e.rev].cap += df;
      }
      mxf += df;
      mnc += df*d[t];
    }
    return make_pair(mxf,mnc);
} }flow;
```

3 Graph

3.1 Bellman-Ford

```
for(int j = 0; j < n-1; j++){
    for(int i = 0; i < m; i++){ // 對於所有邊都嘗試鬆弛
        if(dis[ edge[i].to ] > dis[ edge[i].from ] +
            edge[i].weight){
            dis[ edge[i].to ] = dis[ edge[i].from ] +
                edge[i].weight;
        }
    }
}
//優化版
int len[N]; // 紀錄每個點是第幾輪被鬆弛到 共n個點，最多
    只會被鬆弛n-1次，超過n-1次代表有負環。
bool inque[N];
queue<int> que;
que.push(start);
while(!que.empty()){
    int u = que.front(); que.pop();
    if(len[u] > n-1)    return -1; // 超過 n-1 輪，找到
        負環
    inque[u] = 0;
    for(int i = 0; i < edge[u].size(); i++){
        int v = edge[u][i].to, w = edge[u][i].weight;
        if(!inque[v] && dis[v] > dis[u] + w){
            dis[v] = dis[u] + w;
            que.push(v);
            inque[v] = 1;
            len[v] = len[u] + 1; // 從來的點 +1輪被鬆弛
                到
        }
    }
}
```

3.2 Dijkstra

```
void dijkstra()
{

  for (int i = 0; i <= vertice; i++)//initialize
  {
    dis[i] = INF;
    vis[i] = 0;
  }
  dis[A] = 0;

  while (1)
  {
    int idx = -1;
    int mx = INF;
    for (int i = 0; i <= vertice; i++)
    {
      if (!vis[i] && dis[i] < mx)
      {
        idx = i, mx = dis[i];
      }
    }
    if (idx == -1)
      break;
    vis[idx] = 1;
    for (int i = 0; i < node[idx].size(); i++)
    {
      int u = node[idx][i];
      int weight = w[idx][u];
      if (dis[u] > dis[idx] + weight)
      {
        dis[u] = dis[idx] + weight;
      }
    }
  }
}
//優化版
init();
priority_queue<pair<ll,int>,vector<pair<ll,int>>,
    greater<pair<ll,int>>> pq;

pq.push(make_pair(dis[start], start)); // 為了方便實
    作，用pair包起來會先比較距離大小

while(!pq.empty()){
    auto [d, u] = pq.top(); pq.pop();
    if(vis[u])    continue;  // 確保每個點最多只被走過
        一遍
    vis[u] = 1;
    for(int i = 0; i < edge[u].size(); i++){  // 窮舉此
        點所有連到的點
        int v = edge[u][i].to, w = edge[u][i].weight;
        if(dis[v] > dis[u] + w){
            dis[v] = dis[u] + w;  // 鬆弛
            pq.push(make_pair(dis[v], v)); // 如果有更
                新距離，則丟進 priority_queue
        }
    }
}
```

3.3 Floyd

```
int dis[N][N];
void init(){
  for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
        if(i != j)
            dis[i][j] = INF;
        else
            dis[i][j] = 0;
    }
}
}
for(int k = 0; k < n; k++){ // 窮舉中繼點 k
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){ // 窮舉點對 (i, j)
            dis[i][j] = min(dis[i][j], dis[i][k]+dis[k
                ][j]);
        }
    }
}
```

3.4 BCC

```
//tarjan's algorithm
struct BccVertex {
  int n,nScc,step,dfn[MXN],low[MXN];
  vector<int> E[MXN],sccv[MXN];
  int top,stk[MXN];
  void init(int _n) {  //初始化點的數量
    n = _n; nScc = step = 0;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void addEdge(int u, int v)  // 加無向邊
  { E[u].PB(v); E[v].PB(u); }
  void DFS(int u, int f) {
    dfn[u] = low[u] = step++;
    stk[top++] = u;
    for (auto v:E[u]) {
      if (v == f) continue;
      if (dfn[v] == -1) {
        DFS(v,u);
        low[u] = min(low[u], low[v]);
        if (low[v] >= dfn[u]) {
          int z;
          sccv[nScc].clear();
          do {
            z = stk[--top];
            sccv[nScc].PB(z);
          } while (z != v);
          sccv[nScc++].PB(u);
        }
      }else
        low[u] = min(low[u],dfn[v]);
  } }
  vector<vector<int>> solve() { // 跑 Tarjan
    vector<vector<int>> res;
    for (int i=0; i<n; i++)
      dfn[i] = low[i] = -1;
    for (int i=0; i<n; i++)
      if (dfn[i] == -1) {
        top = 0;
        DFS(i,i);
      }
    REP(i,nScc) res.PB(sccv[i]);
    return res;
  }
}graph;
```

3.5 SCC

```
//kosaraju’ s algorithm
#define FZ(x) memset(x, 0, sizeof(x))
#define PB push_back
struct Scc{
  int n, nScc, vst[MXN], bln[MXN]; // 最後每個點所屬的
      連通分量存在bln陣列
  vector<int> E[MXN], rE[MXN], vec;
  void init(int _n){ //先初始化點的數量
    n = _n;
    for (int i=0; i<MXN; i++)
      E[i].clear(), rE[i].clear();
  }
  void addEdge(int u, int v){  // 加有向邊
    E[u].PB(v); rE[v].PB(u);
  }
  void DFS(int u){
    vst[u]=1;
    for (auto v : E[u]) if (!vst[v]) DFS(v);
    vec.PB(u);
  }
  void rDFS(int u){
    vst[u] = 1; bln[u] = nScc;
    for (auto v : rE[u]) if (!vst[v]) rDFS(v);
  }
  void solve(){  // 跑 kosaraju
    nScc = 0;
    vec.clear();
    FZ(vst);
    for (int i=0; i<n; i++)
      if (!vst[i]) DFS(i);
    reverse(vec.begin(),vec.end());
    FZ(vst);
    for (auto v : vec)
      if (!vst[v]){
        rDFS(v); nScc++;
      }
  }
};
```

3.6 EulerPathCircuit

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> path;
void dfs(int x){
    while(!edge[x].empty()){
        int u = edge[x].back();
        edge[x].pop_back();
        dfs(u);
    }
    path.push_back(x);
}
int main(){
    build_graph();
    dfs(st);
    reverse(path.begin(),path.end());
    for(int i:path)    cout<<i<<' ';
    cout<<endl;
}
```

3.7 HeavyLightDecomposition

```
#include <iostream>
#include <vector>
using namespace std;
int sz[MXN],dep[MXN],fa[MXN],heavy[MXN];
vector<int> treeID;
vector<vector<int>> tree;
int treeSz=1;
int root[MXN], len[MXN];
void dfs1(int x,int f,int d){
    dep[x]=d;  //紀錄深度
    fa[x]=f; //設父節點
    sz[x]=1;     //將自己本身加進子樹大小
    heavy[x]=0;    //一開始先指向空
    for(int i=0;i<edge[x].size();i++){
        if(edge[x][i]==f)    continue;
        dfs1(edge[x][i],x,d+1);        //dfs下去紀錄每
            個子樹的大小
        sz[x]+=sz[edge[x][i]];         //將子樹大小加至
            自己
        if(sz[edge[x][i]]>sz[heavy[x]])  heavy[x]=edge[
            x][i];  //找重兒子
    }
}
void dfs2(int x,int f,bool isLight){
    for(int i=0;i<edge[x].size();i++){
        if(edge[x][i]==f)    continue;
        if(edge[x][i]==heavy[x])            //判斷是否
            為重兒子
            root[edge[x][i]]=root[x];       //若為重兒
                子則跟自己同一條鏈
        else    root[edge[x][i]]=edge[x][i],treeSz++;
            //否則剖分成新鏈
        dfs2(edge[x][i],x);
    }
    len[root[x]]++;                         //紀錄每條
        鏈的長度儲存在第一節點的位置
}

/*void buildTree(){
    tree.resize(treeSz);
    for(int i=1,j=0;i<=n;i++){
        if(root[i]==i){    //判斷鏈的開頭
            treeID[i]=j++;    //設為第j條鏈
            tree[treeID[i]].resize(len[i]*4,0);  //以線
                段樹為例
        }                                        //設第
            j條鏈的長度4倍大小
    }
}線段樹*/

/*void update(int ver,int x,int v,bool ini){
  int dif=val[ver][x-1];
  while(x<BIT[ver].size()){
    if(ini) BIT[ver][x]^=(1<<dif);
    BIT[ver][x]^=v;
    x+=lowbit(x);
  }
}

void build(){
  for(int i=1;i<=n;i++){
    BIT[root[i]].resize(len[root[i]]+5);
  }

  for(int i=1;i<=n;i++){
    if(!visit[root[i]]){
      for(int j=0;j<val[root[i]].size();j++){

        update(root[i],j+1,1<<val[root[i]][j],0);
      }

      visit[root[i]]=1;
    }
  }
}BIT*/

int main(){

}
```

3.8 LCAEulerTourtechnique

```
#include <iostream>
#include <vector>
#include <cstring>
#include <algorithm>
#define on 1
#define off 0
#define ll long long
using namespace std;
const int MAX=2e5+10;
vector<int> node[MAX];
vector<pair<int,int>> rec(MAX);//紀錄i節點之子樹左右區
    間
bool status[MAX]={0};//輸入當前各節點狀態
bool for_tree[MAX]={0};
int tree[4*MAX]={0};//線段樹
int lazy_tag[4*MAX]={0};//懶標記
int cnt=0;
//線段數、樹壓平、區間修改、區間訪問

void dfs_time(int x){//得到節點x的子樹區間左右界
    rec[x].first=++cnt;
    for_tree[cnt]=status[x];//Important! 由於節點dfs不
        一定會是按照節點編號遞迴，所以當前紀錄的左界不
        一定等於當前節點編號
    for(auto i:node[x]){
        dfs_time(i);
    }
    rec[x].second=cnt;
}
void build(int l,int r,int idx){//建線段樹
    if(l==r){
        tree[idx]=for_tree[l];
        return ;
    }
    int mid=(l+r)>>1;
    build(l,mid,idx<<1);
    build(mid+1,r,idx<<1|1);
    tree[idx]=tree[idx<<1]+tree[idx<<1|1];
    return ;
}
void push(int l,int r,int cur){//懶標記下移
    if(lazy_tag[cur]){
        int mid=(l+r)>>1;
        //將懶標記往下打
        lazy_tag[cur<<1]^=1;
        lazy_tag[cur<<1|1]^=1;
        //
        lazy_tag[cur]=0;
        tree[cur<<1]=(mid-l+1)-tree[cur<<1];//值為該子
            樹的映射 0->1 / 1->0
        tree[cur<<1|1]=(r-(mid+1)+1)-tree[cur<<1|1];
    }
    return ;
}
void update(int ql,int qr,int l,int r,int cur){//更新區
    間值

    if(ql<=l&&r<=qr){//代表整個區間都需要更新
        tree[cur]=(r-l+1)-tree[cur];
        lazy_tag[cur]^=1;
        return ;
    }
    int mid=(l+r)>>1;
    push(l,r,cur);//檢查是否有之前更新所留下來的懶標記(
        有的話順便更新值)

    if(ql<=mid){
        update(ql,qr,l,mid,cur<<1);
    }
    if(qr>mid){
        update(ql,qr,mid+1,r,cur<<1|1);//cur<<1|1=cur
            *2+1
    }

    tree[cur]=tree[cur<<1]+tree[cur<<1|1];//子節點值更
        新之後需要其祖先的值也進行更新
    return;
}
int query(int ql,int qr,int l,int r,int cur){//詢問區間
    值
    if(l>=ql&&r<=qr){
        return tree[cur];
    }
    int ans=0;
    int mid=(l+r)>>1;
    push(l,r,cur);//順便檢查是否有懶標記
    if(ql<=mid){
        ans+=query(ql,qr,l,mid,cur<<1);
    }
    if(qr>mid){
        ans+=query(ql,qr,mid+1,r,cur<<1|1);
    }
    tree[cur]=tree[cur<<1]+tree[cur<<1|1];
    return ans;
}
int main(){
    int n;
    cin>>n;
    for(int i=2;i<=n;i++){
        int u;
        cin>>u;
        node[u].push_back(i);
    }

    for(int i=1;i<=n;i++){
        cin>>status[i];
    }

    dfs_time(1);
    build(1,n,1);

}
```

3.9 LCA

```
#include <iostream>
#include <cmath>
#include <vector>
using namespace std;
const int mx=1e5+10;
vector<int>edge[mx];
int tin[mx],tou[mx];
int  t=1;
int anc[mx][log2(mx)]={0};
int n;
void dfs(int x,int f){
  anc[x][0]=f;
  tin[x]=t++;
  for(int i:edge[x]){
    if(i==f)continue;
    dfs(i,x);
  }
  tou[x]=t++;
  return ;
}
bool isAncestor(int anc,int child){
  return tin[anc]<=tin[child]&&tou[anc]>=tou[child];
}
int getLca(int u, int v){
    if(isAncestor(u, v))    return u; // 如果 u 為 v 的
        祖先則 lca 為 u
    if(isAncestor(v, u))    return v; // 如果 v 為 u 的
        祖先則 lca 為 u
    for(int i=lgN;i>=0;i--){    // 判斷 2^lgN, 2^(lgN
        -1),...2^1, 2^0 倍祖先
        if(!isAncestor(anc[u][i], v)) // 如果 2^i 倍祖
            先不是 v 的祖先
            u = anc[u][i];            // 則往上移動
    }
    return anc[u][0]; // 回傳此點的父節點即為答案
}
int main(){
  cin>>n;
  int u,v;
  for(int i=1;i<=n;i++){
    cin>>u>>v;
    edge[u].push_back(v);
    edge[v].push_back(u);
  }
  dfs(1,1);
  for(int i=1;i<=log2(n);i++){
    for(int j=1;j<=n;j++){
      anc[j][i]=anc[anc[j][i-1]][i-1];
    }
  }
}
```

3.10 TopologicalSort

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
const int mx=1e5+10;
int tin[mx];
int tou[mx];
vector<int>edge[mx];
int main(){
  int n;
  cin>>n;
  int u,v;
  for(int i=0;i<n;i++){
    cin>>u>>v;
    edge[u].push_back(v);
    edge[v].push_back(u);
  }
  queue<int>q;
  for(int i=0;i<n;i++){
    if(!tin[i]){
      q.push(i);
    }
  }
  vector<int>ans;
  while(!q.empty()){
    int t=q.front();q.pop();
    for(int i:edge[t]){
      tin[i]--;
      if(!tin[i]){
        q.push(i);
      }
    }
    ans.push_back(t);
  }

}
```

3.11 SPFA

```
bool spfa(){
  queue<int>q;
  dis[1]=0;
  cnt[1]=1;
  q.push(1);
  while(!q.empty()){
    int u=q.front();q.pop();
    vis[u]=0;
    for(int i=0;i<edge[u].size();i++){
      int xx=edge[u][i].first;
      int yy=edge[u][i].second;
      if(dis[xx]>dis[u]+yy){
        dis[xx]=dis[u]+yy;
        cnt[xx]++;
        if(cnt[xx]>n)//return true;
        if(!vis[xx])q.push(xx);
      }
    }
  }
  //return false;
}
```

4 DataStructure

4.1 Treap

```
#include <iostream>
#define NO_TAG 0
using namespace std;
struct Treap{
    int val,pri,sz;     //key,priority,size
    int tag;//用於翻轉區間、單點詢問
    Treap *l, *r;       //左右子樹
    Treap(){}
    Treap(int _key){
        val = _key;
        pri = rand();   //隨機的數維持樹的平衡
        sz  = 1;
        l = r = nullptr;
    }
};
Treap *root=new Treap;
int Size(Treap *x){return x? x->sz:0;}
void pull(Treap *x){x->sz= Size(x->l) +Size(x->r)+1; }
    //用於區間加值 內部程式碼依題目要求更改

//記得在Treap結構裡加入變數tag
void push(Treap* x){//翻轉後中序輸出
    if(x->tag){                      //如果區間要翻轉
        swap(x->l,x->r);             //交換左右子樹
        if(x->l)    x->l->tag ^= 1;  //加tag到左子樹
        if(x->r)    x->r->tag ^= 1;  //加tag到右子樹
        x->tag = NO_TAG;
    }
}

//左邊Treap的key(value)需<=右邊的Treap
Treap *merge(Treap *a, Treap *b){
  if(!a||!b) return a?a:b;
  if(a->pri>b->pri){
    a->r=merge(a->r,b);
    pull(a);
    return a;
  }
  else{
    b->l=merge(a,b->l);
    pull(b);
    return b;
  }
}
//x 欲分割Treap
void splitByKey(Treap *x,int k,Treap* &a,Treap* &b){
    if(!x){ a=b=nullptr; }
    else if(x->val<=k){//自己本身以及左子樹均小於等於k
      a=x;
      splitByKey(x->r,k,a->r,b);
      pull(a);
    }
    else{
      b=x;
      splitByKey(x->l,k,a,b->l);
      pull(b);
    }
}
//希望左邊Treap大小為k個 右邊為n-k個
void splitByKth(Treap *x,int k,Treap* &a,Treap* &b){
  if(!x){ a=b=nullptr;}
  else if(Size(x->l)+1<=k){ //若自己+左子樹<=k個
    a=x;
    splitByKth(x->r,k-Size(x->l)-1,a->r,b);
    pull(a);
  }
  else{
    b=x;
    splitByKth(x->l,k,a,b->l);
    pull(b);
  }
}
void insert(int val){             //新增一個值為val的元
    素
    Treap *x = new Treap(val);    //設一個treap節點
    Treap *l,*r;
    splitByKey(root, val, l, r);  //找到新節點要放的位置
    root = merge(merge(l,x),r);   //合併到原本的treap裡
}
void erase(int key){
    Treap *l,*mid,*r;
    split(root,key,l,r);//<=key給l,>key給r
    split(l,key-1,l,mid);//<key給l ==key給mid(l只儲存<=key的值)
    root=merge(l,r);
}
void cut(int ql,int qr){//欲翻轉區間 ，split函式根據依照kth分割或者key分割而有所變動
    push(root);
    Treap *l,*mid,*r,*tmp;
    split(root,ql-1,l,tmp);//l=[1~ql-1] tmp=[ql~n]
    split(tmp,qr-ql+1,mid,r);// mid=[ql~qr] r=[qr+1~n]
    mid->tag^=1;
    root=merge(merge(l,mid),r);
    return ;
}
```

4.2 Bit+Treap

```
#include <bits/stdc++.h>
using namespace std;
const int mx = 6e4 + 5;

struct Treap
{
    Treap *l, *r;
    int val, sz, pri;
    Treap()
    {
        pri = rand(); //維持Treap平衡
        l = r = nullptr;
    };
    Treap(int x)
    {
        val = x;
        sz = 1;
        pri = rand();
        l = r = nullptr;
    }
};
Treap addr[800000];
int cal = 0;
Treap *newnode()
{
    addr[cal].l = addr[cal].r = nullptr;
    return &addr[cal++];
}
int lowbit(int x) { return x & (-x); }
int Size(Treap *x)
{
    return x ? x->sz : 0;
}

void pull(Treap *x)
{
    x->sz = Size(x->l) + Size(x->r) + 1;
    return;
}

Treap *merge(Treap *a, Treap *b)
{ // Treap a所有key需均<=Treap b
    if (!a || !b)
        return a ? a : b;
    if (a->pri > b->pri)
    {
        a->r = merge(a->r, b);
        pull(a);
        return a;
    }
    else
    {
        b->l = merge(a, b->l);
        pull(b);
        return b;
    }
}
void split(Treap *x, int k, Treap *&a, Treap *&b)
{ //將Treap x依照k值大小 <=k的子樹分割給a >k則給b
    if (!x)
    {
        a = b = nullptr;
    }
    else if (x->val <= k)
    {
        a = x;
        split(x->r, k, a->r, b);
        pull(a);
    }
    else
    {
        b = x;
        split(x->l, k, a, b->l);
        pull(b);
    }
}
Treap *insert(Treap *root, int key)
{
    Treap *x = newnode();
    x->val = key;
    x->sz=1;
    x->pri=rand();
    Treap *l, *r;
    split(root, key, l, r);
    return merge(merge(l, x), r);
}
Treap *erase(Treap *root, int key)
{
    Treap *l, *mid, *r;
    split(root, key, l, r);    //<=key給l,>key給r
    split(l, key - 1, l, mid); //<key給l ==key給mid(l只儲存<=key的值)
    return merge(merge(merge(l,mid->l),mid->r), r);//只需移除一個元素 故移除端點元素 其餘子傑點去做merge後回傳
}
struct query
{
    int l, r, k;
};
vector<query> Q;
Treap *bit[mx];
vector<int> num;
int rec[mx] = {0};
void update(int pos, int val, int mode)
{
    if (mode)
    {
        while (pos < mx)
        {
            bit[pos] = insert(bit[pos], val);
            pos += lowbit(pos);
        }
    }
    else
    {
        while (pos < mx)
        {
            bit[pos] = erase(bit[pos], val);
            pos += lowbit(pos);
        }
    }
    return;
}
int cut(Treap *root, int ql, int qr)
{
    Treap *l, *mid, *r;
    split(root, qr, l, r);    // l=[1,qr] r=[qr+1,n]
    split(l, ql - 1, mid, l); // l=[ql,qr] mid=q[1~ql-1]
    int ans = Size(l);
    root = merge(merge(mid, l), r);
    return ans;
}
int check(int pos, int l, int r)
{
    int rec = 0;
    while (pos)
    {
        rec += cut(bit[pos], l, r);
        pos -= lowbit(pos);
    }
    return rec;
}
int id(int val)
{
    return lower_bound(num.begin(), num.end(), val) -
        num.begin() + 1;
}
int query(int ql, int qr, int k)
{
    int l = 1, r = mx;
    while (l < r)
    {
        int mid = (l + r) >> 1;
        if (check(mid, ql, qr) < k)
        {
            l = mid + 1;
        }
        else
            r = mid;
    }
    return r;
}
int main()
{
    cin.tie(0);
    ios::sync_with_stdio(0);
    int t, n, q, x;
    char cmd;
    cin >> t;
    while (t--)
    {
        memset(rec,0,sizeof(rec));
        for (int i = 0; i < mx; i++)
        {
            bit[i] = nullptr;
        }
        cal = 0;
        num.clear();
        Q.clear();
        cin >> n >> q;
        num.resize(n);
        for (int i = 0; i < n; i++)
        {
            cin >> num[i];
            rec[i] = num[i];
        }
        while (q--)
        {
            cin >> cmd;
            int l, r, k, pos;
            if (cmd == 'Q')
            {
                cin >> l >> r >> k;
                Q.push_back({l, r, k});
            }
            else
            {
                cin >> pos >> k;
                num.push_back(k);
                Q.push_back({0, pos, k});
            }
        }
        sort(num.begin(), num.end());
        num.erase(unique(num.begin(), num.end()), num.
            end());
        for (int i = 0; i < n; i++)
        {
            rec[i] = id(rec[i]);
            update(rec[i], i + 1, 1);
        }
        for (int i = 0; i < Q.size(); i++)
        {
            if (Q[i].l == 0)
            { //更換操作
                int pos = Q[i].r;
                update(rec[pos - 1], pos, 0);
                update(id(Q[i].k), pos, 1);
                rec[pos - 1] = id(Q[i].k);
            }
            else
            {
                cout << num[query(Q[i].l, Q[i].r, Q[i].
                    k) - 1] << '\n';
            }
        }
    }
    return 0;
}
```

4.3 2D-SegTree

```
#define cl(x) (x<<1)+1
#define cr(x) (x<<1)+2
void build_y(int idx, int idy, int lx, int rx, int ly,
     int ry){
    if(ly == ry){
        tree[idx][idy] = arr[idx][idy];
        return;
    }
    int my = (ly+ry)/2;
    build_y(idx, cl(idy), lx, rx, ly, my);
    build_y(idx, cr(idy), lx, rx, my+1, ry);
    tree[idx][idy] = tree[idx][cl(idy)] + tree[idx][cr(
        idy)];
}
void build_x(int idx, int lx, int rx){
    if(lx != rx){
        int mx = (l+r)/2;
        build_x(cl(idx), lx, mx);
        build_x(cr(idx), mx+1, rx);
    }
    build_y(idx, 0, lx, rx, 0, m-1);
}
  int query_y(int idx, int idy, int ly, int ry, int qly
      , int qry){
    if(qly <= ly && ry <= qry){
        return tree[idx][idy];
    }
    int my = (ly+ry)/2, ret = 0;
    if(qy <= my) ret += query_y(idx, cl(idy), ly, my,
        qly, qry);
    if(qy >  my) ret += query_y(idx, cr(idy), my+1, ry,
         qly, qry);
    return ret;
}
//詢問區間 (qxl,qyl)□(qxr,qyr)的區間和
int query_x(int idx, int lx, int rx, int qlx, int qly,
     int qrx, int qry){
    if(qlx <= lx && rx <= qrx){
        return query_y(idx, 0, 0, m-1, qly, qry);
    }
    int mx = (lx + rx)/2, ret = 0;
    if(qlx <= mx)
        ret += query_x(cl(idx), lx, mx, qlx, qly, qrx,
            qry);
    if(qrx >  mx)
        ret += query_x(cr(idx), mx+1, qrx, qlx, qly,
            qrx, qry);
    return ret;
}
  void update_y(int idx, int idy, int lx, int rx, int
      ly, int ry, int qy, int val){
    if(ly == ry){
        if(lx == rx)
            tree[idx][idy] = val;
        else
            tree[idx][idy] = tree[cl(idx)][idy] + tree[
                cr(idx)][idy];
        return;
    }
    int my = (ly+ry)/2;
    if(qy <= my) update_y(idx, cl(idy), lx, rx, ly, my,
         qy, val);
    else         update_y(idx, cr(idy), lx, rx, my+1,
        ry, qy, val);
    tree[idx][idy] = tree[idx][cl(idy)] + tree[idx][cr(
        idy)];
}
void update_x(int idx, int lx, int rx, int qx, int qy,
     int val){
```

```
    if(lx != rx){
        int mx = (lx + rx)/2;
        if(qx <= mx)
            update_x(cl(idx), lx, mx, qx, qy, val);
        else
            update_x(cr(idx), mx+1, rx, qx, qy, val);
    }
    update_y(idx, 0, lx, rx, 0, m-1, qy, val);
}
```

4.4 2D-BIT

```
int lowbit(int x){return x&(-x); }
void update(int r,int c,int val){
  for(int i=r;i<=n;i+=lowbit(i)){
    for(int j=c;j<=n;j+=lowbit(j)){
      BIT[i][j]+=val;
    }
  }
  return ;
}
int query(int r,int c){
  int rec=0;
  for(int i=r;i;i-=lowbit(i)){
    for(int j=c;j;j-=lowbit(j)){
      rec+=BIT[i][j];
    }
  }
  return rec;
}
```

4.5 HJT-Tree

```
#include <iostream>
#include <vector>
using namespace std;

struct node{
    ll val;
    node *l, *r;
};
 /*偽指標 一直new node可能會MLE 故開一個node 陣列 每次要新建節點就給予該陣列的位址
 node addr[MXN];
int id=0;
node *newNode(){
  return &addr[id++];
}
*/
vector<node *> version;    //用一個vector紀錄全部版本的根節點

void build(node *now_version, l, r){
    if(l==r){
        now_version->w=0;
        return ;
    }
    int mid=(l+r)>>1;
    now_version->l=getnode();
    now_version->r=getnode();
    build(now_version->l,l,mid);
    build(now_version->r,mid+1,r);
    now_version->w=0;
    return ;
}

int query(node *cur,int l,int r,int pos){//查詢最後出現版本小於POS的數值
    if(l==r)return l;
    int mid=(l+r)>>1;
    if(cur->l->w<pos)return query(cur->l,l,mid,pos);//若左子節點版本數<pos(ql)
    else return query(cur->r,mid+1,r,pos);
}




void add_version(int x,int v){    //修改位置 x 的值為 v
    version.push_back(update_version(version.back(), 0, n-1, x, v));
}

node *update_version(node *pre_version, l, r, pos, v){
    node *x = new node();    //當前位置建立新節點
    if(l == r){
        x->val = v;
        return x;
    }
    int mid = (l+r)>>1;
    if(pos <= mid){ //更新左邊
        x->l = update(pre->l, l, mid, pos, v); //左邊節點連向新節點
        x->r = pre->version->r;
            //右邊連到原本的右邊
    }
    else{ //更新右邊
        x->l = pre->l;                        //左邊連到原本的左邊
        x->r = update(pre->r, r, mid, pos, v);  //右邊節點連向新節點
    }
    x->val = x->l->val + x->r->val;
    return x;
}
```

5 Geometry

5.1 ConvexHull

```
#include <iostream>
#include <algorithm>
#include <cmath>
#include <vector>
#include <iomanip>
#define ll long long
#define ld long double
using namespace std;
struct pt{
    int x,y;
    pt(){};
    pt(int _x,int _y){
        x=_x;y=_y;
    }
    pt operator-( pt &a){
        return pt(x-a.x,y-a.y);
    }
};
vector<pt>p;
vector<pt>stk;
int cross( pt &a,  pt &b, pt &c){
    pt lhs=b-a,rhs=c-a;
    return lhs.x*rhs.y-lhs.y*rhs.x;
}
bool cmp(const pt &a,const pt &b){
    if(a.x==b.x)return a.y<b.y;
    return a.x<b.x;
}
int dis(const pt &a,const pt &b){
    return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
}
void hull(){
    sort(p.begin(), p.end(),cmp);//依照x大小排序
    stk.push_back(p[0]);
    stk.push_back(p[1]);

    int top=2;
    for(int i=2;i<p.size();i++){
        while(top>=2 && cross(stk[top-2], stk[top-1],p[i])<=0){//使得stk內維持兩個點以上，並維護連續3個點的外積>0(逆時鐘旋轉)
            top--;
            stk.pop_back();
        }
        top++;
        stk.push_back(p[i]);
    }
    int rec=top+1;
    for(int i=p.size()-2;i>=0;i--){
        while(top>=rec&&cross(stk[top-2], stk[top-1], p[i])<=0){
            top--;
            stk.pop_back();
        }
        top++;
        stk.push_back(p[i]);
    }
    stk.pop_back();//考慮逆時針做完凸包後，最左端點被重
        複計算
    top--;
    stk.resize(top);
    return ;
}
int farthestdis(){
    int rec=0;
    for(int i=0;i<stk.size();i++){
        int j=i+1;
        while( dis(stk[i],stk[j])<dis(stk[i],stk[ (j+1)
            %stk.size() ]) ){
            j= (j+1)%stk.size();
        }
        rec=max(rec,dis(stk[i],stk[j]));
    }
    return rec;
}
int main(){
    int c;
    cin>>c;
    int m,n;
    for(int i=1;i<=c;i++){
        cin>>m>>n;
        p.push_back(pt(m,n));
    }
    hull();//尋找凸包
    int ans=farthestdis();//尋找最遠點對

    cout<<fixed<<setprecision(10)<<sqrt(ans)<<'\n';
}
```

5.2 MaximumAreaOfQuadrilateral

```
ll MXN_quadrilaterals(){
    int size=hull.size();

    ll ans=0;
    int t;
    int tt;
   for(int i=0;i<size;i++){
        t=(i+1)%size;
        tt=(i+3)%size;
       for(int j=(i+2)%size;j!=i;j=(j+1)%size){//分成兩
           部分做最大三角形 t在i和j之間(上半部) , tt在j
           到i之間(下半部)

           while(t!=j&&area(hull[i],hull[j],hull[t])<
               area(hull[i],hull[j],hull[(t+1)%size])){
               //
               t=(t+1)%size;
           }
            while(tt!=i&&area(hull[i],hull[j],hull[tt])
                <area(hull[i],hull[j],hull[(tt+1)%size
                ])){
                tt=(tt+1)%size;
            }
            ans=max(ans,area(hull[i],hull[j],hull[t])+
                area(hull[i],hull[j],hull[tt]));
       }
   }
    return ans ;
}
```

5.3 GeometryTemplate

```
struct pt{
    ll x,y;
    pt (){}
    pt(int a,int b){
       x=a;y=b;
   }
   pt operator-(const pt &a){
       return pt(x-a.x,y-a.y);
   }
   ll operator*(const pt&a){//內積
       return x*a.x+y*a.y;
   }
   bool operator==(const pt &a){
       return x==a.x && y==a.y;
   }
   ll operator^(const pt &a){//外積
       return x*a.y-y*a.x;
   }
};
struct Line{
    pt st,ed;
    Line(){}
    Line(pt a,pt b){
        st=a,ed=b;
    }
};
typedef struct Line Line;

bool collinearity(pt &a ,pt &b, pt &c){//是否三點共線，
    外積面積為0
    return ( (b-a) ^ (c-a) )==0;
}

bool inLine(pt &check, Line &tmp){//判斷點是否在線上
    if(check==tmp.st||check==tmp.ed)return true;//若兩
        條線有相同端點
    return collinearity(tmp.st,tmp.ed,check)&& ((tmp.st
        -check)*(tmp.ed-check)) <0;//三點共線，且交點往
        線的兩端點之向量為反方向=>內積<0
}

bool isColid( Line &a, Line &b){//判斷是否有交點

    //判斷是否3點共線(交點在端點上)
    if(inLine(a.st,b)) return true;
    if(inLine(a.ed,b)) return true;
    if(inLine(b.st,a)) return true;
    if(inLine(b.ed,a)) return true;


    /*判斷是否交點在線段上(某線上一點會在另條線的相異
        側，也就是對於a點，另一條的的c,d點，c對a為逆時
        針，d對a為順時針)
    (ABxAC)(ABxAD)<0 && (CDxCA)(CDxCB)<0*/
    ll w,x,y,z;
    w=(a.ed-a.st)^(b.st-a.st);
    x=(a.ed-a.st)^(b.ed-a.st);
    y=(b.ed-b.st)^(a.st-b.st);
    z=(b.ed-b.st)^(a.ed-b.st);
    if( w*x < 0 && y*z <0 )return true;

    return false;
}
```

6 String

6.1 Bitwise-Trie

```
struct trie{
    trie *nxt[2];  // 差別
    int cnt;    //紀錄有多少個數字以此節點結尾
    int sz;     //有多少數字的前綴包括此節點
    trie():cnt(0),sz(0){
        memset(nxt,0,sizeof(nxt));
    }
};

//創建新的字典樹
trie *root = new trie();

void insert(int x){
    trie *now = root;  // 每次從根結點出發
    for(int i=30;i>=0;i--)
        now->sz++;
        if(now->nxt[x>>i&1] == NULL){
            now->nxt[x>>i&1] = new trie();
        }
        now = now->nxt[x>>i&1];  //走到下一個字母
    }
    now->cnt++;
    now->sz++;
}
ll query(int x)//依題目要求更改
{
    ll ans = 0;
    trie *now = root;
    for (int i = 30; i >= 0; i--)
    {
        //bool j = x & (1 << i) ? 0 : 1;
        if (now->next[j] != NULL && now->next[j]->cnt >
            0)
        {
            ans += (1 << i);
            now = now->next[j];
        }
        else
            now = now->next[j ^ 1];
    }
    return ans;
}
```

6.2 Trie

```
#include <iosteam>
using namespace std;
struct trie{
    trie *nxt[26];
    int cnt;    //紀錄有多少個字串以此節點結尾
    int sz;     //有多少字串的前綴包括此節點
    trie():cnt(0),sz(0){
        memset(nxt,0,sizeof(nxt));
    }
};

//創建新的字典樹
trie *root = new trie();

void insert(string& s){
    trie *now = root;  // 每次從根結點出發
    for(auto i:s){
        now->sz++;
        if(now->nxt[i-'a'] == NULL){
            now->nxt[i-'a'] = new trie();
        }
        now = now->nxt[i-'a'];  //走到下一個字母
    }
    now->cnt++;
    now->sz++;
}
  // O(|s|)
int query_prefix(string& s){  //查詢有多少前綴為 s
    trie *now = root;    // 每次從根結點出發
    for(auto i:s){
        if(now->nxt[i-'a'] == NULL){
            return 0;
        }
        now = now->nxt[i-'a'];
    }
    return now->sz;
}
int query_count(string& s){  //查詢字串 s 出現次數
    trie *now = root;    // 每次從根結點出發
    for(auto i:s){
        if(now->nxt[i-'a'] == NULL){
            return 0;
        }
        now = now->nxt[i-'a'];
    }
    return now->cnt;
}
```

6.3 RollingHash

```
const ll P = 75577;
const ll MOD = 998244353;
string s;
ll Hash[MXN];    //Hash[i] 為字串 [0,i] 的 hash值
Hash[0]=s[0];
ll _p[MXN];
void build(const string& s){
    for(int i=i; i<s.size(); i++){
        Hash[i]=(Hash[i-1]*p%mod+s[i])%mod;
    }
}
bool check(int l,int r){
  ll rec=Hash[r]-Hash[l-1]*p[r-l+1]%mod;
  if(rec<0)rec+=mod;
  return rec;
}
int main(){
  _p[0]=1;

  for(int i=1;i<MXN;i++){
    _p[i]=_p[i-1]*p%mod;
  }

}
```

7 SqrtStructure

7.1 SqrtDecomposition

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

struct blk{
    vector<int> local;    //每塊的全部元素
    int global;           //儲存每塊的總和
    int tag;              //儲存整塊一起更新的值
    blk(){                //初始化
        local.clear();    //清空區間元素
        tag = global = 0; //將區間總和先設為0
    }
};

int len=sqrt(n);//每塊長度(取下界)
int num=(len+n-1)/len;//塊數(取上界)

void update(int ql,int qr,int v){//區間更新，加值
    int blk_l=ql/len,blk_r=qr/len,ret=0;
    if(blk_l == blk_r){     //如果都在同一塊直接一個一個
        跑過去就好
        for(int i=ql;i<=qr;i++)
            b[blk_l].local[i%len]+=v;
        b[blk_l].global+=(qr-ql+1)*v;
        return;
    }
    for(int i=ql;i<(blk_l+1)*len;i++){  //最左的那一塊
        b[blk_l].local[i%len]+=v;
        b[blk_l].global+=v;
    }
    for(int i=blk_l+1;i<blk_r;i++){  //中間每塊
        b[i].tag+=v;
        b[i].global+=v*len;
    }
    for(int i=blk_r*len;i<=qr;i++){  //最右的那一塊
        b[blk_r].local[i%len]+=v;
        b[blk_r].global+=v;
    }
}
int query(int ql,int qr){//區間詢問
    int blk_l=ql/len,blk_r=qr/len,ret=0;
    if(blk_l == blk_r){     //如果都在同一塊直接一個一個
        跑過去就好
        for(int i=ql;i<=qr;i++)
            ret+=b[blk_l].local[i%len]+b[blk_l].tag;
        return ret;
    }
    for(int i=ql;i<(blk_l+1)*len;i++)    //最左的那一塊
        ret+=b[blk_l].local[i%len]+b[blk_l].tag;
    for(int i=blk_l+1;i<blk_r;i++)    //中間每塊的總和
        ret+=b[i].global;
    for(int i=blk_r*len;i<=qr;i++)    //最右的那一塊
        ret+=b[blk_r].local[i%len]+b[blk_r].tag;
    return ret;
}
int main(){
    vector<blk> b;
    int len=sqrt(n),num=(n+len-1)/len;
    for(int i=0;i<n;i++){    //第i個元素分在第 i/len 塊
        cin>>x;
        //存入區間中
        b[i/len].local.push_back(x);
        //更新區間總和
        b[i/len].global += x;

  }
}
```

7.2 Mo's

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;
struct qu{
  int l,r,id;
};

bool cmp(const qu &a,const qu &b){
  int len=sqrt(n);
  if(a.l/len==b.l/len){
    return a.r<b.r;
  }
  return a.l<b.l;
}

void add(int x){}
void sub(int x){}
vector<query> q;
void add(int index){ return ; }
void sub(int index){ return ; }
void solve(){
    sort(q.begin(),q.end());
    for(int i=0,l=-1,r=0;i<n;i++){
        while(l>q[i].l)    add(--l);
        while(r<q[i].r)    add(++r);    //記得要先做新增元素的
        while(l<q[i].l)    sub(l++);    //再做移除元素的
        while(r>q[i].r)    sub(r--);
        ans[q[i].id] = num;
  }
}
```