数据结构

树

树的直径

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int inf=0x3f3f3f3f;
const int N=2e5+10;
const int M=1e3+10;
vector<int>e[N];
int mx;
int dp[N];
void dfs(int root,int p)
{
    for(auto it:e[root])
    {
        if(it==p)continue;
        dp[it]=dp[root]+1;
        if(dp[it]>dp[mx])
        {
            mx=it;
        }
        dfs(it,root);
    }
    return ;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    int n;
    cin>>n;
    int u,v;
    for(int i=1;i<n;i++)
    {
        cin>>u>>v;
        e[u].push_back(v);
        e[v].push_back(u);
    }

    dfs(1,-1);
    dp[mx]=0;
    dfs(mx,-1);
    cout<<dp[mx];
    return 0;
}
```

```
const int N = 10000 + 10;

int n, c, d[N];
vector<int> E[N];
```

```
void dfs(int u, int fa) {
  for (int v : E[u]) {
    if (v == fa) continue;
    d[v] = d[u] + 1;
    if (d[v] > d[c]) c = v;
    dfs(v, u);
  }
}

int main() {
  scanf("%d", &n);
  for (int i = 1; i < n; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    E[u].push_back(v), E[v].push_back(u);
  }
  dfs(1, 0);
  d[c] = 0, dfs(c, 0);
  printf("%d\n", d[c]);
  return 0;
}
```

树的重心

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
typedef pair<int,int> pii;
const int inf=0x3f3f3f3f;
const int N=2e6+10;
const int M=1e3+10;
vector<int>e[N];
int dp[N];
int weight[N];
int n;
// int ans=0;
vector<int>ans;
void dfs(int root,int p)
{
    dp[root]=1;
    weight[root]=0;
    for(auto it:e[root])
    {
        if(it==p)continue;
        dfs(it,root);
        dp[root]+=dp[it];
        weight[root]=max(weight[root],dp[it]);
    }
    weight[root]=max(weight[root],n-dp[root]);
```

```
    if(weight[root]<=n/2)
    {
        ans.push_back(root);
    }
    return ;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    // int n;
    cin>>n;
    int u,v;
    for(int i=1;i<n;i++)
    {
        cin>>u>>v;
        e[u].push_back(v);
        e[v].push_back(u);
    }
    dfs(1,-1);
    for(auto it:ans)
    {
        cout<<it<<" ";
    }
    // cout<<ans<<"\n";
    return 0;
}
```

换根dp

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const ll inf=0x3f3f3f3f;
const ll N=1e6+1;
ll a[N];
ll dp[N];
ll dep[N];
ll dist[N];
ll sz[N];
ll n,m,t;
vector<ll>e[N];
void dfs(ll root,ll p)
{
    sz[root]=1;
    for(auto it:e[root])
    {
```

```
        if(it==p)continue;
        dep[it]=dep[root]+1;
        dfs(it,root);
        dp[root]+=dp[it]+sz[it];
        sz[root]+=sz[it];
    }
    return ;
}
void dfs2(ll root,ll p)
{
    for(auto it:e[root])
    {
        if(it==p)continue;
        dist[it]=dist[root]+n-2*sz[it];
        dfs2(it,root);
    }
    return ;
}
void solve()
{
    cin>>n;
    for(ll i=1;i<n;i++)
    {
        ll u,v;
        cin>>u>>v;
        e[u].push_back(v);
        e[v].push_back(u);
    }
    dfs(1,-1);
    for(ll i=1;i<=n;i++){
        dist[1]+=dep[i];
    }
    dfs2(1,-1);
    ll ans=0;ll pos=0;
    for(ll i=1;i<=n;i++)
    {
        if(ans<dist[i])
        {
            ans=dist[i];
            pos=i;
        }
    }
    cout<<pos;
    return ;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    solve();
    return 0;
}
```

kmp

```
/*
int f1()//暴力
{
    string a;
    string b;
    getline(cin,a);
    getline(cin,b);
    int i=0;int j=0;
    int len1=a.size();
    int len2=b.size();
    while(i<len1&&j<len2)
    {
        if(a[i]==b[j])
        {
            i++;
            j++;
        }
        else
        {
            i=i-(j-1);
            j=0;
        }
    }
    if(j==len2)
        return (i-j);
    else
        return -1;
}
*/
#include<iostream>
#include<string>
using namespace std;
string s1;
string s2;
int nextt[1000000];
void kmp()
{
    int i=0,j=0;
    int len1=s1.size(),len2=s2.size();
    while((i<len1)&&(j<len2))
    {
        if(j==-1||s1[i]==s2[j])
        {
            i++;
            j++;
        }
        else
```

```
        {
            j=nextt[j];
        }
        if(j==len2) cout<<i-j<<endl;
    }
    //匹配成功
    return ;
}

void creat_next()
{
    nextt[0]=-1;
    int k=-1;
    int j=0;
    while(j<s2.size())
    {
        if(k==-1||s2[k]==s2[j])
        {
            k++;
            j++;
            nextt[j]=k;
        }
        else
        {
            k=nextt[k];
        }
    }
}
int main()
{
    cin>>s1>>s2;
    creat_next();
    kmp();
    return 0;
}
```

二叉树

前中--->后

```
string middle,front;
void dfs(int ms,int me,int fs,int fe)
{
    if(ms>me||fs>fe)return ;
    for(int i=ms;i<=me;i++)
        if(middle[i]==front[fs])
        {
            dfs(ms,i-1,fs+1,fs+i-ms);
            dfs(i+1,me,fs+i-ms+1,fe);
```

```
            cout<<middle[i];
        }
}
int main()
{
    cin>>middle>>front;
    dfs(0,middle.size()-1,0,front.size()-1);
    return 0;
}
```

后中--->前

```
string middle,behind;
void dfs(int ms,int me,int bs,int be)
{
    if(ms>me||bs>be)return ;
    cout<<behind[be];
    for(int i=ms;i<=me;i++)
    {
        if(middle[i]==behind[be])
        {
            dfs(ms,i-1,bs,bs+i-ms-1);
            dfs(i+1,me,bs+i-ms,be-1);
        }
    }
}
int main()
{
    cin>>middle>>behind;
    dfs(0,middle.size()-1,0,behind.size()-1);
    return 0;
}
```

建树 遍历

```
typedef struct Node
{
    char data;
    struct Node *lchild, *rchild;
}Node;
//通过先序的方式创建树，#表示空节点
/*
                    A
                B       C
              D    E    F  #
            #  # # # #  #
创建上面的树应输入应为  ABD##E##CF###
前序遍历：ABDECF
```

```
中序遍历：DBEAFC
后序遍历：DEBFCA
层次遍历：ABCDEF
*/
void creatTree(Node* &root)
{
    char data;
    cin >> data;
    if (data == '#')
        root = NULL;
    else
    {
        root = new Node;
        root->data = data;
        creatTree(root->lchild);
        creatTree(root->rchild);
    }

}
//打印一个节点的数据
void visit(Node* node)
{
    if(node!=NULL)
        cout << node->data;
}
//递归-前序遍历，先访问根节点，然后访问左节点，最后访问右节点，每一个节点都要准守这样的规则
void preTraversal(Node* root)
{
    //访问根节点
    if (root != NULL)
    {
        visit(root);
        preTraversal(root->lchild);
        preTraversal(root->rchild);
    }
}
//递归-中序遍历，先访问跟左节点，然后访问中节点，最后访问右节点，每一个节点都要准守这样的规则
void midTraversal(Node* root)
{
    if (root != NULL)
    {
        midTraversal(root->lchild);
        visit(root);
        midTraversal(root->rchild);
    }
}
//递归-后序遍历，先访问左节点，然后访问右节点，最后访问根节点，每一个节点都要准守这样的规则
void postTraversal(Node* root)
{
    if (root != NULL)
    {
        postTraversal(root->lchild);
```

```
        postTraversal(root->rchild);
        visit(root);
    }
}

//非递归-前序遍历
/*
思想：用栈来实现。首先访问根节点，然后将根节点入栈，接着访问当前节点的左节点，然后入栈，当左节点访问完后，
出栈，并依次访问右节点
*/
void un_preTraversal(Node* root)
{
    stack<Node*> stack;
    //当前节点
    Node* p = root;
    while (p != NULL || stack.size() != 0)
    {
        if (p != NULL)
        {
            visit(p);//访问p之前一定要保证p不为空
            stack.push(p);
            p = p->lchild;
        }
        else
        {
            p = stack.top();
            stack.pop();
            p = p->rchild;
        }
    }
}

//非递归-中序遍历
/*
思想：用栈来实现。从根节点开始依次遍历当前节点的左节点，并依次入栈，当左节点遍历完成后，获取
栈顶元素并出栈，然后访问该节点，并依次遍历其右节点
*/
void un_midTraversal(Node* root)
{
    stack<Node*> stack;
    Node* p = root;
    while (p != NULL || stack.size() != 0)
    {
        if (p != NULL)
        {
            stack.push(p);
            p = p->lchild;
        }
        else
        {
            p = stack.top();
```

```
            stack.pop();
            visit(p);
            p = p->rchild;
        }
    }
}
//非递归-后序遍历
/*
思想：用栈来实现。先根节点开始依次遍历左节点，已经遍历过了的标记为'l'，然后依次遍历右节点，遍历过的标记为'r'，
只有当标记为'r'时才能访问该节点。
*/
//定义一个有标记的结构体
typedef struct TNode
{
    Node* node;//树的节点的指针
    char tag;//标记
}TNode;


void un_postTraversal(Node* root)
{
    //当前节点
    Node *p = root;
    TNode *n;
    stack<TNode*> stack;
    while (p != NULL || stack.empty() == false)
    {
        //遍历左节点并标记
        while (p != NULL)
        {
            n = new TNode;
            n->node = p;
            n->tag = 'l';
            stack.push(n);
            p = p->lchild;
        }

        //出栈
        n = stack.top();
        stack.pop();

        //遍历当前节点的右子树
        if (n->tag == 'l')
        {
            n->tag = 'r';
            //再次入栈
            stack.push(n);
            //此时p==NULL,一定要给p当前的节点
            p = n->node;
            p = p->rchild;
        }
```

```
        //左右子树遍历完成后访问该节点
        else
        {
            visit(n->node);
            //并把p置空防止
            p = NULL;
        }
    }
}

//树的层次遍历
//思想：使用队列queue。先将根节点入队列，循环判断当前队列不为空时，将头元素出队列并访问头元素，然后在将它的左节点和右节点入队列
void levelTraversal(Node* root)
{
    queue<Node*> q;
    Node* p = root;
    q.push(p);
    while (q.empty() == false)
    {
        p = q.front();
        q.pop();
        visit(p);
        if (p->lchild != NULL)
            q.push(p->lchild);
        if (p->rchild != NULL)
            q.push(p->rchild);
    }
}

int main()
{
    //创建上面的树应输入应为  ABD##E##CF###

    Node* root;
    creatTree(root);
    cout << "递归-前序遍历:";
    preTraversal(root);
    cout << endl;

    cout << "递归-中序遍历:";
    midTraversal(root);
    cout << endl;

    cout << "递归-后序遍历:";
    postTraversal(root);
    cout << endl;

    cout << "非递归-前序遍历:";
    un_preTraversal(root);
    cout << endl;
```

```
    cout << "非递归-中序遍历:";
    un_midTraversal(root);
    cout << endl;

    cout << "非递归-后序遍历:";
    un_postTraversal(root);
    cout << endl;
    cout << "层次遍历:";
    levelTraversal(root);
    cout << endl;
}
```

树状数组

```
#include <bits/stdc++.h>
const int N=500010;
int tree[N];
int n,m;
int lowbit(int x)
{
    return x&(-x);
}
void add(int x,int k)
{
    while(x<=n)
    {
        tree[x]=tree[x]+k;
        x=lowbit(x)+x;
    }
}
int sum(int x)
{
    int ans=0;
    while(x!=0)
    {
        ans=ans+tree[x];
        x=x-lowbit(x);
    }
    return ans;
}
int main()
{
    std::cin>>n>>m;
    int x,y,k,choice;
    for(int i=1;i<=n;i++)
    {
        std::cin>>k;
        add(i,k);
    }
    for(int i=1;i<=m;i++)
```

```
    {
        std::cin>>choice;
        if(choice==1)
        {
            std::cin>>x>>k;
            add(x,k);
        }
        else
        {
            std::cin>>x>>y;
            std::cout<<sum(y)-sum(x-1)<<"\n";
        }
    }
    return 0;
}
```

树状数组区间

```
#include <bits/stdc++.h>
const int N=500010;
int tree[N];
int s1[N];
int s2[N];
int a[N];
int n,m;
int lowbit(int x)
{
    return x&(-x);
}
void add(int x,int k)
{
    int temp=x;
    while(x<=n)
    {
        s1[x]=s1[x]+k;
        s2[x]=s2[x]+temp*k;
        x=lowbit(x)+x;
    }
}
int sum(int x)
{
    int ans=0;
    int temp=x;
    while(x!=0)
    {
        ans=ans+(temp+1)*s1[x]-s2[x];
        x=x-lowbit(x);
    }
    return ans;
```

```
}
int main()
{
    std::cin>>n>>m;
    int x,y,k,choice;
    for(int i=1;i<=n;i++)
    {
        std::cin>>a[i];
    }
    for(int i=1;i<=n;i++)
    {
        add(i,a[i]-a[i-1]);
    }
    for(int i=1;i<=m;i++)
    {
        std::cin>>choice;
        if(choice==1)
        {
            std::cin>>x>>y>>k;
            add(x,k);add(y+1,-k);
        }
        else
        {
            std::cin>>x;
            std::cout<<sum(x)-sum(x-1)<<"\n";
        }
    }
    return 0;
}
```

权值树状数组--第k大

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int inf=0x3f3f3f3f;
const int N=1e6+10;
const int M=1e3+10;
int a[N];
int tree[N];
void add(int x,int k)
{
    while(x<=1e6)
    {
        tree[x]+=k;
        x+=x&(-x);
    }
    return ;
}
int query(int x)
{
```

```
    int res=0;
    while(x)
    {
        res+=tree[x];
        x-=x&(-x);
    }
    return res;
}
int kth(int k)
{
    int l=0;int r=1e6;
    while(l<r)
    {
        int mid=l+r>>1;
        if(query(mid)>=k)
        {
            r=mid;
        }
        else
        {
            l=mid+1;
        }
    }
    return l;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
        add(a[i],1);
    }
    int x,y;
    for(int i=1;i<=m;i++)
    {
        cin>>x>>y;
        add(a[x],-1);
        a[x]=y;
        add(a[x], 1);

        cout<<kth(n/2+1)<<"\n";
    }
    return 0;
}
```

线段树(1)

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int N=1e6;
ll a[N];
ll tag[N];
ll ans[N];
ll ls(ll x)
{
    return x<<1;
}
ll rs(ll x)
{
    return x<<1|1;
}
void push_up(ll p)
{
    ans[p]=ans[ls(p)]+ans[rs(p)];
}
void build(ll p,ll l,ll r)
{
    tag[p]=0;
    if(l==r)
    {
        ans[p]=a[l];
        return ;
    }
    ll mid=(l+r)>>1;
    build(ls(p),l,mid);
    build(rs(p),mid+1,r);
    push_up(p);
}
void f(ll p,ll l,ll r,ll k)
{
    tag[p]=tag[p]+k;            //tag[ls(p)]=tag[ls(p)]+tag[p]
    ans[p]=ans[p]+(r-l+1)*k;    //ans[ls(p)]=ans[ls(p)]+tag[p]
}
void push_down(ll p,ll l,ll r)
{
    ll mid=(l+r)>>1;
    f(ls(p),l,mid,tag[p]);
    f(rs(p),mid+1,r,tag[p]);
    tag[p]=0;
}
void update(ll nl,ll nr,ll l,ll r,ll p,ll k)
{
    if(nl<=l&&nr>=r)
    {
        ans[p]=ans[p]+(r-l+1)*k;
```

```
        tag[p]=tag[p]+k;
        return ;
    }
    push_down(p,l,r);//把tag往下传
    ll mid=(l+r)/2;
    if(nl<=mid)update(nl,nr,l,mid,ls(p),k);
    if(nr>mid) update(nl,nr,mid+1,r,rs(p),k);
    push_up(p); //回溯
}
ll query(ll q_x,ll q_y,ll l,ll r,ll p)
{
    ll res=0;
    if(q_x<=l&&q_y>=r)return ans[p];
    ll mid=(l+r)/2;
    push_down(p,l,r);
    if(q_x<=mid)res=res+query(q_x,q_y,l,mid,ls(p));
    if(q_y> mid)res=res+query(q_x,q_y,mid+1,r,rs(p));
    return res;
}
int main()
{
    ll n,m;
    cin>>n>>m;
    for(ll i=1;i<=n;i++)
    {
        cin>>a[i];
    }
    build(1,1,n);
    for(ll i=1;i<=m;i++)
    {
        ll chose;
        cin>>chose;
        if(chose==1)
        {
            ll x,y,k;
            cin>>x>>y>>k;
            update(x,y,1,n,1,k);
        }
        else
        {
            ll x,y;
            cin>>x>>y;
            cout<<query(x,y,1,n,1)<<endl;
        }
    }
    return 0;
}
```

线段树(2) 区间加法+乘法 区间查询

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const ll inf=0x3f3f3f3f;
const ll N=1e4+1;
ll a[N];
void pushup(ll p);
void pushdown(ll p);
struct node
{
    ll l;
    ll r;
    ll mul;
    ll add;
    ll add_sum;
    ll pow_sum;
};
node tree[4*N];
void solve()
{
    return ;
}
ll ls(ll p)
{
    return p<<1;
}
ll rs(ll p)
{
    return p<<1|1;
}
void build(ll l,ll r,ll p)
{
    tree[p].l=l;
    tree[p].r=r;
    tree[p].add_sum=0;
    tree[p].pow_sum=0;
    tree[p].add=0;
    tree[p].mul=1;
    if(l==r)
    {
        tree[p].add_sum=a[l];
        tree[p].pow_sum=a[l]*a[l];
        return ;
    }
    ll mid=l+r>>1;
    build(l,mid,ls(p));
    build(mid+1,r,rs(p));
    pushup(p);
    return ;
```

```
}
void pushup(ll p)
{
    tree[p].add_sum=tree[ls(p)].add_sum+tree[rs(p)].add_sum;
    tree[p].pow_sum=tree[ls(p)].pow_sum+tree[rs(p)].pow_sum;
    return ;
}
void pushdown(ll p)
{
    ll mul=tree[p].mul;
    ll add=tree[p].add;


    tree[ls(p)].pow_sum=
        +tree[ls(p)].pow_sum*mul*mul
        +2*add*tree[ls(p)].add_sum
        +(tree[ls(p)].r-tree[ls(p)].l+1)*add*add;

    tree[rs(p)].pow_sum=
        +tree[rs(p)].pow_sum*mul*mul
        +2*add*tree[rs(p)].add_sum
        +(tree[rs(p)].r-tree[rs(p)].l+1)*add*add;

    tree[ls(p)].add_sum=
        +mul*tree[ls(p)].add_sum
        +add*(tree[ls(p)].r-tree[ls(p)].l+1);
    tree[rs(p)].add_sum=
        +mul*tree[rs(p)].add_sum
        +add*(tree[rs(p)].r-tree[rs(p)].l+1);

    tree[ls(p)].mul*=mul;
    tree[rs(p)].mul*=mul;

    tree[ls(p)].add=tree[ls(p)].add*mul+add;
    tree[rs(p)].add=tree[rs(p)].add*mul+add;

    tree[p].mul=1;
    tree[p].add=0;
    return;
}
void modify_add(ll l,ll r,ll k,ll p)
{
    if(l<=tree[p].l&&tree[p].r<=r)
    {
        tree[p].add+=k;
        tree[p].pow_sum+=2*tree[p].add_sum*k+k*k*(tree[p].r-tree[p].l+1);
        tree[p].add_sum+=(tree[p].r-tree[p].l+1)*k;
        return;
    }

    pushdown(p);
    ll mid=tree[p].l+tree[p].r>>1;
```

```
    if(l<=mid)modify_add(l,r,k,ls(p));
    if(r> mid)modify_add(l,r,k,rs(p));
    pushup(p);
    return ;
}
void modify_mul(ll l,ll r,ll k,ll p)
{
    if(l<=tree[p].l&&tree[p].r<=r)
    {
        tree[p].mul*=k;
        tree[p].add*=k;
        tree[p].add_sum*=k;
        tree[p].pow_sum*=k*k;
        return ;
    }
    pushdown(p);
    ll mid=tree[p].l+tree[p].r>>1;
    if(l<=mid)modify_mul(l,r,k,ls(p));
    if(r>mid) modify_mul(l,r,k,rs(p));

    pushup(p);
    return ;
}
ll query_pow_add(ll l,ll r,ll p)
{
    if(l<=tree[p].l&&tree[p].r<=r)
    {
        return tree[p].pow_sum;
    }
    pushdown(p);
    ll mid=tree[p].l+tree[p].r>>1;
    ll ans=0;
    if(l<=mid)ans+=query_pow_add(l,r,ls(p));
    if(r >mid)ans+=query_pow_add(l,r,rs(p));
    return ans;
}
ll query_add(ll l,ll r,ll p)
{
    if(l<=tree[p].l&&tree[p].r<=r)
    {
        return tree[p].add_sum;
    }
    ll ans=0;
    pushdown(p);
    ll mid=tree[p].l+tree[p].r>>1;
    if(l<=mid)ans+=query_add(l,r,ls(p));
    if(r>mid) ans+=query_add(l,r,rs(p));
    return ans;
}

int main()
{
```

```
    ios::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);

    ll n,m;
    cin>>n>>m;

    for(ll i=1;i<=n;i++)
    {
        cin>>a[i];
    }
    build(1,n,1);
    ll op,l,r;
    while(m--)
    {
        cin>>op>>l>>r;
        if(op==1)
        {
            cout<<query_add(l,r,1)<<"\n";
        }
        else if(op==2)
        {
            cout<<query_pow_add(l,r,1)<<"\n";
        }
        else if(op==3)
        {
            ll x;
            cin>>x;
            modify_mul(l,r,x,1);
        }
        else if(op==4)
        {
            ll x;
            cin>>x;
            modify_add(l,r,x,1);
        }
    }
    return 0;
}
```

线段树（3）区间最值max min

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const ll N=1e6+10;
ll maxx[N];
ll minn[N];
ll ans[N];
ll tag[N];
ll a[N];
ll ls(ll x)
```

```
{
    return x<<1;
}
ll rs(ll x)
{
    return x<<1|1;
}
void push_up(ll p)
{
    maxx[p]=max(maxx[ls(p)],maxx[rs(p)]);
    minn[p]=min(minn[ls(p)],minn[rs(p)]);
    return ;
}
void build(ll l,ll r,ll p)
{
    if(l==r)
    {
        maxx[p]=a[l];
        minn[p]=a[l];
        return ;
    }
    ll mid=l+r>>1;
    build(l,mid,ls(p));
    build(mid+1,r,rs(p));
    push_up(p);
    return ;
}
ll query_minn(ll q_l,ll q_r,ll l,ll r,ll p)
{
    if(q_l==l&&q_r==r)
    {
        return minn[p];
    }
    ll res=0x3f3f3f3f;
    ll mid=(l+r)>>1;
    if(q_r<=mid)res=min(res,query_minn(q_l,q_r,l,mid,ls(p)));
    else if(q_l>mid)res=min(res,query_minn(q_l,q_r,mid+1,r,rs(p)));
    else
    {
        res=min(res,query_minn(q_l,mid,l,mid,ls(p)));
        res=min(res,query_minn(mid+1,q_r,mid+1,r,rs(p)));
    }
    return res;
}
ll query_maxx(ll q_l,ll q_r,ll l,ll r,ll p)
{
    if(q_l==l&&q_r==r)
    {
        return maxx[p];
    }
    ll res=0;
    ll mid=(l+r)>>1;
```

```
    if(q_r<=mid)res=max(res,query_maxx(q_l,q_r,l,mid,ls(p)));
    else if(q_l>mid)res=max(res,query_maxx(q_l,q_r,mid+1,r,rs(p)));
    else
    {
        res=max(res,query_maxx(q_l,mid,l,mid,ls(p)));
        res=max(res,query_maxx(mid+1,q_r,mid+1,r,rs(p)));
    }
    return res;
}
int main()
{
    ll n,t;
    cin>>n>>t;
    for(ll i=1;i<=n;i++)
    {
        cin>>a[i];
    }
    build(1,n,1);
    while(t--)
    {
        ll l,r;
        cin>>l>>r;
        cout<<query_maxx(l,r,1,n,1)-query_minn(l,r,1,n,1)<<"\n";
    }
    return 0;
}
```

数列分块

区间修改+单点查询

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int bl[N];
int v[N];
int tag[N];
int n,m,t;
void add(int l,int r,int k)
{
    for(int i=l;i<=min(r,bl[l]*m);i++)
    {
        v[i]+=k;
    }
    if(bl[l]!=bl[r])
    {
        for(int i=(bl[r]-1)*m+1;i<=r;i++)
        {
```

```
            v[i]+=k;
        }
    }
    for(int i=bl[l]+1;i<=bl[r]-1;i++)
    {
        tag[i]+=k;
    }
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    cin>>n;
    cin>>t;
    m=sqrt(n);
    for(int i=1;i<=n;i++)
    {
        cin>>v[i];
    }
    for(int i=1;i<=n;i++)
    {
        bl[i]=(i-1)/m+1;//ceil i/m;
    }
    for(int i=1;i<=t;i++)
    {
        int op,l,r,k;
        cin>>op;
        if(op==0)
        {
            cin>>l>>r>>k;
            add(l,r,k);
        }
        else
        {
            cin>>r;
            cout<<v[r]+tag[bl[r]]<<"\n";
        }
    }
    return 0;
}
```

图

单源最短路

Dijkstra 优先队列优化（边权>0)

```
#include <bits/stdc++.h>
const int N=200010;
const int inf=0x3f3f3f3f;
struct edge{
    int u;
    int v;
    int dis;
    int next;
};
edge e[N];
bool vis[N];
int dis[N];
int pre[N];
int n,m,s;
void CreatAdjacencyList()
{
    std::cin>>n>>m>>s;
    int u,v,dis;
    int index=1;
    memset(pre,-1,sizeof(pre));
    for(int i=1;i<=m;i++)
    {
        std::cin>>u>>v>>dis;
        e[index].v=v;
        e[index].dis=dis;
        e[index].next=pre[u];
        pre[u]=index;index++;
    }
}
struct node
{
    int u;
    int dis;
    bool operator<(const node &a)const{
        return a.dis<dis;
    }
};
std::priority_queue<node>heap;
void dijkstra()
{
    for(int i=1;i<=n;i++)
    {
        dis[i]=2147483647;
    }
    dis[1]=0;
    heap.push({1,0});
    while(!heap.empty())
    {
        node temp=heap.top();
```

```
        heap.pop();
        int tmp_u=temp.u;int tmp_dis=temp.dis;//1->u的长度
        if(vis[tmp_u])continue;
        vis[tmp_u]=true;
        for(int j=pre[tmp_u];j!=-1;j=e[j].next)
        {
            int v=e[j].v;
            if(dis[v]>e[j].dis+tmp_dis)//1->v  >  u->v + 1->u
            {
                dis[v]=e[j].dis+tmp_dis;
                heap.push({v,dis[v]});
            }
        }
    }
}
int main()
{
    CreatAdjacencyList();
    dijkstra();
    for(int i=1;i<=n;i++)
    {
        std::cout<<dis[i]<<" ";
    }
    return 0;
}

//nlog(n)
```

SPFA判断负环

```
#include <bits/stdc++.h>
using namespace std;
const int inf=0x3f3f3f3f;
const int N=1e5+10;
int n,m;
int dis[N];
int vis[N];
int cnt[N];
#define pii pair<int,int>
vector<pii>e[N];
int u,v,val;
bool SPFA()
{
    queue<int>q;
    for(int i=1;i<=n;i++)
    {
        q.push(i);
    }

    while(q.size())
```

```
    {
        int t=q.front();
        q.pop();
        vis[t]=0;
        for(auto it:e[t])
        {
            if(dis[it.first]>dis[t]+it.second)
            {
                cnt[it.first]=cnt[t]+1;
                dis[it.first]=dis[t]+it.second;
                if(cnt[it.first]>=n)
                {
                    return true;
                }
                if(!vis[it.first])
                {
                    q.push(it.first);
                }
            }
        }

    }
    return false;
}
int main()
{
    cin>>n>>m;
    memset(dis,0x3f,sizeof(dis));
    for(int i=1;i<=m;i++)
    {
        cin>>u>>v>>val;
        e[u].emplace_back(pair{v,val});
    }
    if(SPFA())cout<<"Yes";
    else      cout<<"No";
}
```

SPFA(边权可负)

```
#include <bits/stdc++.h>
using namespace std;
#define pii pair<int,int>
const int inf=0x3f3f3f3f;
const int N=1e5+10;
int vis[N];
int dis[N];
vector<pii>e[N];
void SPFA()
{
    queue<int>q;
```

```
    q.push(1);
    while(q.size())
    {
        int t=q.front();
        q.pop();
        vis[t]=0;
        for(auto it:e[t])
        {
            if(dis[it.first]>dis[t]+it.second)
            {
                dis[it.first]=dis[t]+it.second;
                if(!vis[it.first])
                {
                    q.push(it.first);
                    vis[it.first]=1;
                }
            }
        }
    }
}
int main()
{
    int n,m;
    cin>>n>>m;
    int u,v,val;
    memset(dis,0x3f,sizeof(dis));
    dis[1]=0;
    vis[1]=1;
    for(int i=1;i<=m;i++)
    {
        cin>>u>>v>>val;
        e[u].push_back({v,val});
    }
    SPFA();
    if(dis[n]>=inf)cout<<"impossible";
    else cout<<dis[n];

    return 0;
}
```

bellman ford 有边数限制的最短路

```
#define ll long long
const int inf=2e9;
const int N=1e5+1;
struct node{
    int u;
    int v;
    int val;
};
node e[N];
```

```
int dis[N];
int last[N];
int main()
{
    ios::sync_with_stdio(false);
    int n,k,m;
    cin>>n>>m>>k;
    for(int i=1;i<=m;i++)
    {
        cin>>e[i].u>>e[i].v>>e[i].val;
    }
    memset(dis,0x7f,sizeof(dis));
    //cout<<dis[1];
    dis[1]=0;
    for(int i=1;i<=k;i++)
    {
        memcpy(last,dis,sizeof(dis));
        for(int i=1;i<=m;i++)
        {
            dis[e[i].v]=min(last[e[i].u]+e[i].val,dis[e[i].v]);
        }
    }
    if(dis[n]>=inf)cout<<"impossible";
    else cout<<dis[n];
    return 0;
}
```

多源最短路

Floyd

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int inf=2e6;
const int N=300+1;
int a[N];
#define pii pair<int,int>
int G[N][N];
int dis[N];
int main()
{
    ios::sync_with_stdio(false);
    int n,m,k;
    cin>>n>>m>>k;
    memset(G,0x3f,sizeof(G));
    for(int i=1;i<=n;i++)
    {
        G[i][i]=0;
```

```
    }
    for(int i=1;i<=m;i++)
    {
        int u,v,val;
        cin>>u>>v>>val;
        G[u][v]=min(G[u][v],val);
        //G[v][u]=min(G[v][u],val);
    }
    for(int k=1;k<=n;k++)
    {
        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=n;j++)
            {
                G[i][j]=min(G[i][k]+G[k][j],G[i][j]);
            }
        }
    }
    while(k--)
    {
        int x,y;
        cin>>x>>y;
        if(G[x][y]>=inf)cout<<"impossible"<<endl;
        else cout<<G[x][y]<<endl;
    }
    return 0;
}
```

最小生成树

Kruskal

```
const int N=5e5+10;
const int inf=0x3f3f3f3f;
int parent[N];
int find(int x)
{
    return parent[x]==x?x:parent[x]=find(parent[x]);
}
void to_union(int x1,int x2)
{
    int f1=find(x1);
    int f2=find(x2);
    if(f1<f2)swap(f1,f2);
    parent[f1]=f2;
}
struct node{
    int u;
    int v;
```

```
    int val;
    bool operator<(const node&a)const
    {
        return a.val>val;
    }
};
node e[N];
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        parent[i]=i;
    }
    int u,v,val;
    for(int i=1;i<=m;i++)
    {
        cin>>u>>v>>val;
        e[i].u=u;
        e[i].v=v;
        e[i].val=val;
    }
    sort(e+1,e+m+1);
    // for(int i=1;i<=m;i++)
    // {
    //     cout<<e[i].val<<endl;
    // }
    int ans=0;
    for(int i=1;i<=m;i++)
    {
        if(find(e[i].u)==find(e[i].v))continue;
        ans=ans+e[i].val;
        to_union(e[i].u,e[i].v);
    }
    int same=find(1);
    int flag=0;
    for(int i=2;i<=n;i++)
    {
        if(same!=find(i))
        {
            flag=1;
            break;
        }
    }
    if(flag==1)
    {
        cout<<"impossible";
    }
    else cout<<ans;
    return 0;
}
```

二分图

```
#include <bits/stdc++.h>
using namespace std;

struct augment_path {
  vector<vector<int> > g;
  vector<int> pa;  // 匹配
  vector<int> pb;
  vector<int> vis;  // 访问
  int n, m;         // 顶点和边的数量
  int dfn;          // 时间戳记
  int res;          // 匹配数

  augment_path(int _n, int _m) : n(_n), m(_m) {
    assert(0 <= n && 0 <= m);
    pa = vector<int>(n, -1);
    pb = vector<int>(m, -1);
    vis = vector<int>(n);
    g.resize(n);
    res = 0;
    dfn = 0;
  }

  void add(int from, int to) {
    assert(0 <= from && from < n && 0 <= to && to < m);
    g[from].push_back(to);
  }

  bool dfs(int v) {
    vis[v] = dfn;
    for (int u : g[v]) {
      if (pb[u] == -1) {
        pb[u] = v;
        pa[v] = u;
        return true;
      }
    }
    for (int u : g[v]) {
      if (vis[pb[u]] != dfn && dfs(pb[u])) {
        pa[v] = u;
        pb[u] = v;
        return true;
      }
    }
    return false;
  }

  int solve() {
    while (true) {
      dfn++;
```

```
      int cnt = 0;
      for (int i = 0; i < n; i++) {
        if (pa[i] == -1 && dfs(i)) {
          cnt++;
        }
      }
      if (cnt == 0) {
        break;
      }
      res += cnt;
    }
    return res;
  }
};

int main() {
  int n, m, e;
  cin >> n >> m >> e;
  augment_path solver(n, m);
  int u, v;
  for (int i = 0; i < e; i++) {
    cin >> u >> v;
    u--, v--;
    solver.add(u, v);
  }
  cout << solver.solve() << "\n";
  for (int i = 0; i < n; i++) {
    cout << solver.pa[i] + 1 << " ";
  }
  cout << "\n";
}
```

线

并查集

维护根深度

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int p[N];int d[N];
int find(int x)
{
    if(x!=p[x])
    {
        int root=find(p[x]);
        d[x]+=d[p[x]];
```

```
        p[x]=root;
    }
    return p[x];
}
int main()
{
    int n,m;
    int cnt=0;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        p[i]=i;
    }
    int c,x,y;
    for(int i=1;i<=m;i++)
    {
        cin>>c>>x>>y;
        if(x>n||y>n)
        {
            cnt++;
            continue;
        }
        int px=find(x);int py=find(y);
        if(c==1)
        {
            if(px==py)
                {if((d[x]-d[y])%3)cnt++;}
            else
                {
                    p[px]=py;
                    d[px]=d[y]-d[x];
                }
        }
        else
        {
            if(px==py)
                {if((d[x]-d[y]-1)%3)cnt++;}
            else
                {
                    p[px]=py;
                    d[px]=d[y]+1-d[x];
                }
        }
    }
    cout<<cnt;
    return 0;
}
```

维护集合元素个数

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const ll N=2e5+5;
ll parent[N];
ll rrank[N];
ll cnt[N];
void to_union(ll x,ll y)
{
    if(x>y)swap(x,y);
    parent[y]=parent[x];
    cnt[x]+=cnt[y];
};
ll find(ll x)
{
    return x==parent[x]?x:find(parent[x]);
}
int main(){
    ll n,m;
    cin>>n>>m;
    for(ll i=1;i<=n;i++)
    {
        parent[i] = i;
        cnt[i] = 1;
    }

    for(ll i=1;i<=m;i++)
    {
        ll x,y;
        cin>>x>>y;
        ll f1=find(x);
        ll f2=find(y);
        if(f1!=f2)to_union(f1,f2);
    }
    map<ll,ll>mp;
    for(ll i=1;i<=n;i++)
    {
        ll x;
        cin>>x;
        if(x)mp[find(i)]++;
    }
    if(mp.size()>=2)cout<<0<<endl;
    else if(!mp.size())
    {
        ll ans=0;
        for(ll i=1;i<=n;i++)
        {
            if(find(i)==i)
                ans+=pow(cnt[i],2);
```

```
        }
        cout<<ans;
    }
    else
    {
        ll temp;
        auto i=mp.begin();
        temp=i->first;
        cout<<cnt[temp]*cnt[temp];
    }

    return 0;
}
```

边点数

```
#include <bits/stdc++.h>
using namespace std;
const int N=2e5+1;
int parent[N];
int vis[N];
int cnt1[N];
int cnt2[N];
int rrank[N];
int find(int x)
{
    return parent[x]==x?x:find(parent[x]);
}
void to_union(int &f1,int &f2)
{

    if(f1>f2)swap(f1,f2);
    parent[f2]=parent[f1];
    cnt1[f1]+=cnt1[f2];
    cnt2[f1]+=cnt2[f2];
    cnt2[f2]=0;

}
int main()
{
    int n,q;
    cin>>n>>q;
    for(int i=1;i<=n;i++)
    {
        cnt1[i]=1;cnt2[i]=0;
        parent[i]=i;
    }
    for(int i=1;i<=q;i++)
    {
        int x1,x2;
```

```
        cin>>x1>>x2;
        int f1=find(x1);
        int f2=find(x2);
        if(f1!=f2)
        to_union(f1,f2);

        cnt2[f1]++;
    }
    int flag=0;
    for(int i=1;i<=n;i++)
    {
        if(vis[find(i)])continue;
        vis[find(i)]=1;
        if(cnt1[find(i)]==cnt2[find(i)])continue;
        else
        {
            flag=1;
            break;
        }
    }
    if(flag==1)puts("No");
    else puts("Yes");
    return 0;
}
```

实例 及格线

```
#include<iostream>
#include<cstring>
#include<algorithm>
using namespace std;
using LL = long long;
const int maxn = 1e5 + 5;
int p[maxn], c[maxn], sz[maxn];

int find(int x){
    return p[x] == x ? x : p[x] = find(p[x]);
}

void merge(int x, int y){
    x = find(x), y = find(y);
    if (x != y){
        p[x] = y;
        sz[y] += sz[x];
    }
}

int main(){
    int n, m;
    cin >> n >> m;
```

```
    for(int i = 1; i <= n; i++) p[i] = i, sz[i] = 1;
    while(m--){
        int a, b;
        cin >> a >> b;
        merge(a, b);
    }
    for(int i = 1; i <= n; i++){
        int x;
        cin >> x;
        c[find(i)] += x;
    }
    int cnt = 0;
    LL sum = 0, ans = 0;
    for(int i = 1; i <= n; i++){
        if (find(i) == i){
            if (c[i] > 0){
                cnt++;
                ans = 1LL * sz[i] * sz[i];
            }
            sum += 1LL * sz[i] * sz[i];
        }
    }
    if (cnt >= 2) cout << 0 << '\n';
    else if (cnt == 1) cout << ans << '\n';
    else cout << sum << '\n';

}
```

链表

单向

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int inf=0x3f3f3f3f;
const int N=1e5+1;
int e[N], l[N], r[N], idx;
void init(){
    //初始化，使得0, 1位置为两个端点
    // 设置节点0为起始节点，节点1为尾节点，下标从2开始，不设head指针
    idx = 2;
    r[0] = 1;//开始
    l[1] = 0;//结束
}

//index为k的右边插入x的节点
void add(int k, int x){
    e[idx] = x;
    l[idx] = k;
    r[idx] = r[k];
```

```
    l[r[k]] = idx;
    r[k] = idx;
    idx++;
}
//删除k节点
void remove(int k){
    r[l[k]] = r[k];
    l[r[k]] = l[k];
}
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        solve();
    }
    return 0;
}
```

双向

```
#include <bits/stdc++.h>
const int N=1e5+1;
struct node
{
    int l;
    int r;
    int data;
};
int idx=2;
node a[N];
void insert(int x,int k)
{
    a[idx].data=x;
    a[idx].l=k;
    a[idx].r=a[k].r;
    a[a[k].r].l=idx;
    a[k].r=idx;
    idx++;
}
void remove(int k)
{
    a[a[k].r].l=a[k].l;
    a[a[k].l].r=a[k].r;
}
int main()
{
    idx=2;
    a[0].r=1;
```

```
    a[1].l=0;
    int n;
    std::cin>>n;
    int x,k;std::string c;
    while(n--)
    {
        std::cin>>c;
        if(c=="L")
        {
            std::cin>>x;
            insert(x,0);
        }
        else if(c=="R")
        {
            std::cin>>x;
            insert(x,a[1].l);
        }
        else if(c=="D")
        {
            std::cin>>k;
            remove(k+1);
        }
        else if(c=="IL")
        {
            std::cin>>k>>x;
            insert(x,a[k+1].l);
        }
        else if(c=="IR")
        {
            std::cin>>k>>x;
            insert(x,k+1);
        }
    }
    for(int i=a[0].r;i!=1;i=a[i].r)
    {
        std::cout<<a[i].data<<" ";
    }
    return 0;
}
```

实例 string

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int inf=0x3f3f3f3f;
const int N=1e5+1;
struct node
{
    int adress;
```

```
    int data;
    int next;
};
node a[N];
node b[N];
node c[N];
int flag[N];
int main()
{
    int head;
    int n;
    cin>>head>>n;
    int t1,t2,t3;
    for(int i=1;i<=n;i++)
    {
        cin>>t1>>t2>>t3;
        a[t1].adress=t1;
        a[t1].data=t2;
        a[t1].next=t3;
    }
    int cnt1=0;
    int cnt2=0;
    int i=head;
    while(a[i].next!=-1)
    {
        if(flag[abs(a[i].data)])//出现过
        {
            c[cnt2].adress=a[i].adress;
            c[cnt2].data=a[i].data;
            c[cnt2++].next=a[i].next;
        }
        else
        {
            flag[abs(a[i].data)]=1;
            b[cnt1].adress=a[i].adress;
            b[cnt1].data=a[i].data;
            b[cnt1++].next=a[i].next;
        }
        i=a[i].next;

    }
    if(flag[abs(a[i].data)])//出现过
        {
            c[cnt2].adress=a[i].adress;
            c[cnt2].data=a[i].data;
            c[cnt2++].next=a[i].next;
        }
        else
        {
            flag[abs(a[i].data)]=1;
            b[cnt1].adress=a[i].adress;
            b[cnt1].data=a[i].data;
```

```
            b[cnt1++].next=a[i].next;
        }
    printf("%05d %d ",b[0].adress,b[0].data);
    for(int i=1;i<cnt1;i++)
    {
        printf("%05d\n",b[i].adress);
        printf("%05d %d ",b[i].adress,b[i].data);
    }
    cout<<-1<<"\n";
    if(cnt2)
    {
        printf("%05d %d ",c[0].adress,c[0].data);
        for(int i=1;i<cnt2;i++)
        {
            printf("%05d\n",c[i].adress);
            printf("%05d %d ",c[i].adress,c[i].data);
        }
        cout<<-1<<"\n";
    }

    return 0;
}
```

栈

计算器

```
#include <bits/stdc++.h>
using namespace std;
stack<int>num;
stack<char>op;
void eval()
{
    auto b=num.top();num.pop();
    auto a=num.top();num.pop();
    auto c=op.top(); op .pop();
    int temp;
    if(c=='+')temp=a+b;
    else if(c=='-')temp=a-b;
    else if(c=='*')temp=a*b;
    else temp=a/b;
    num.push(temp);
}
int main()
{
    map<char, int>mp;
    mp['-']=1;mp['+']=1;
    mp['*']=2;mp['/']=2;
    string s;
    cin>>s;
```

```
    for (int i = 0; i < s.size(); ++i)
    {
        auto c=s[i];
        if(isdigit(c))
        {
            int x=0;int j=i;
            while(j<s.size()&&isdigit(s[j]))
            {
                x=x*10+s[j]-'0';
                j++;
            }
            i=j-1;
            num.push(x);
        }
        else if(c=='(')
        {
            op.push(c);
        }
        else if(c==')')
        {
            while(op.top()!='(')eval();
            op.pop();
        }
        else
        {
            while(op.size()&&mp[op.top()]>=mp[c])eval();
            op.push(c);
        }
    }
    while(op.size())eval();
    cout<<num.top();
    return 0;
}
```