ICPC Templates For ZJUT12

EIPsilly

October 25, 2019

Contents

1 树

1.1 树上倍增求 LCA

```
#include<vector>
#include<cstring>
#include<stdio.h>
using namespace std;
#define mem(a) memset(a,0,sizeof(a))

const int Vertex_MAXN = 1e4 + 10;
const int DEEP = 15;

vector<int> g[Vertex_MAXN];

int up[Vertex_MAXN][DEEP] = {},d[Vertex_MAXN] = {};

void dfs(int x)
{
   d[x] = d[up[x][0]] + 1;
   for (int i = 1;i<=14;i++) up[x][i] = up[up[x][i-1]][i-1];
   for (int i = 0;i<g[x].size();i++)
   if (g[x][i] != up[x][0])
   {
      up[g[x][i]][0] = x;
      dfs(g[x][i]);
   }
}

int LCA(int x,int y)
{
   if (d[x] < d[y]) swap(x,y);
   int dis = d[x] - d[y];
   for (int i = 14;i>=0;i--)
      if (dis & (1<<i)) x = up[x][i];
   if (x == y) return x;
   for (int i = 14;i>=0;i--)
   {
      if (up[x][i] != up[y][i])
      {
         x = up[x][i],y = up[y][i];
      }
   }
   return up[x][0];
}

int main()
{
   int n,in[11000] = {};
   scanf("%d",&n);
   for (int i = 1,u,v;i<=n-1;i++)
   {
      scanf("%d%d",&u,&v);//规定前者为后者的父节点
```

```
        g[u].push_back(v);
        in[v]++;
    }
    int u,v;
    for (int i = 1;i<=n-1;i++)
    {
        if (in[i] == 0)
        {
            dfs(i);
            break;
        }
    }
    scanf("%d%d",&u,&v);//询问u和v的LCA
    printf("%d",LCA(u,v));
    mem(up); mem(d);
    for (int i = 1;i<=n-1;i++) g[i].clear();
    return 0;
}
```

1.2 树链剖分（重链剖分）

```
#include<bits/stdc++.h>
using namespace std;
#define mem(a) memset(a,0,sizeof(a))

const int MAXN = 40100;
vector<int> g[MAXN];
int n,size1[MAXN],dfn[MAXN],top[MAXN],son[MAXN],fa[MAXN],d[MAXN],num = 0,aa[MAXN];
map<pair<int,int>,int> w;

//树链剖分
void dfs1(int x)
{
    d[x] = d[fa[x]] + 1;
    size1[x] = 1;
    for (int i = 0;i<g[x].size();i++)
    {
        int v = g[x][i];
        if (v == fa[x]) continue;
        fa[v] = x;
        dfs1(v);
        size1[x] += size1[v];
        if (size1[v] > size1[son[x]]) son[x] = v;
    }
}

void dfs2(int x,int tp)
{
    dfn[x] = ++num;
    top[x] = tp;
    aa[num] = w[make_pair(x,fa[x])];
    if (son[x]) dfs2(son[x],tp);
```

```
   for (int i = 0;i<g[x].size();i++)
      if (g[x][i] != fa[x] && g[x][i] != son[x]) dfs2(g[x][i],g[x][i]);
}

struct node{
   int l,r,lazy,num,nl,nr;
   int mid(){ return (r - l) / 2 + l;}
   void merge(node a,node b)//合并两个区间
   {
      num = a.num + b.num;
      if (a.nr == b.nl && a.nr != -1 && b.nl != -1) num--;
      nl = a.nl; nr = b.nr;
   }
   void overturn() {swap(nr,nl);}//将区间顺序倒置
   // void write() //检验时用
   // {
   //  printf("%d %d %d %d %d %d\n",l,r,lazy,num,nl,nr);
   // }
};

struct seg_tree{
   node tree[MAXN * 4];
   void build(int st,int ed,int x)
   //建树，lazy表示当前结点下的子节点需要更改的值，nl表示左端点的数字，nr表示右端点的数字，num表示有多少段颜色相同的
   {
      tree[x].l = st;
      tree[x].r = ed;
      tree[x].lazy = -1;
      if (st == ed)
      {
         tree[x].nl = tree[x].nr = aa[st];
         tree[x].num = 1;
         if (st == 1)//建树的时候，一条边上两点，深度较大的那个点的dfn作为这条线段的dfn，所以根节点1没有对应的边
         {
            tree[x].nl = tree[x].nr = -1;
            tree[x].num = 0;
         }
         return;
      }
      int mid = tree[x].mid();
      build(st,mid,x * 2);
      build(mid + 1,ed,x * 2 + 1);
      tree[x].merge(tree[x * 2],tree[x * 2 + 1]);
   }
   void modify(int st,int ed,int x,int c)//区间修改值
   {
      if (tree[x].l >= st && tree[x].r <= ed)
      {
         tree[x].nl = tree[x].nr = c;
         tree[x].num = 1;
         tree[x].lazy = c;
```

```
            return;
        }
        if (tree[x].lazy != -1)
        {
            tree[x*2].lazy=tree[x*2+1].lazy=tree[x*2].nl=tree[x*2].nr=tree[x*2+1].nl=tree[x*2+1].nr=tree[x].lazy;
            tree[x * 2].num = tree[x * 2 + 1].num = 1;
            tree[x].lazy = -1;
        }
        int mid = tree[x].mid();
        if (mid >= st) modify(st,ed,x * 2,c);
        if (ed > mid) modify(st,ed,x * 2 + 1,c);
        tree[x].merge(tree[x * 2],tree[x * 2 + 1]);
    }
    node query(int st,int ed,int x,char c)//查询区间
    {
        if (tree[x].l >= st && tree[x].r <= ed)
        {
            node temp = tree[x];
            if (c == 'L') temp.overturn();//如果是从起始点开始合并
            return temp;
        }
        if (tree[x].lazy != -1)
        {
            tree[x*2].lazy=tree[x*2+1].lazy=tree[x*2].nl=tree[x*2].nr=tree[x*2+1].nl=tree[x*2+1].nr=tree[x].lazy;
            tree[x * 2].num = tree[x * 2 + 1].num = 1;
            tree[x].lazy = -1;
        }
        int mid = tree[x].mid();
        node ans;
        ans.nl = -1; ans.nr = -1; ans.num = 0;
        if (mid >= st) ans = query(st,ed,x * 2,c);
        if (ed > mid)
        {
            if (ans.num == 0) ans = query(st,ed,x * 2 + 1,c);
            else if (c == 'L') ans.merge(query(st,ed,x * 2 + 1,c),ans);
            else if (c == 'R') ans.merge(ans,query(st,ed,x * 2 + 1,c));
        }
        tree[x].merge(tree[x * 2],tree[x * 2 + 1]);
        return ans;
    }
}seg;


int mapping(int x,int y,int c)//c表示当前是修改还是查询
{
    int fx = top[x],fy = top[y];
    node ans1,ans2;
    ans1.nl = ans1.nr = ans2.nl = ans2.nr = -1;//ans1表示从起始点开始合并，ans2表示从终点开始合并
    ans1.num = ans2.num = 0;
    while (fx != fy)
    {
        if (d[fx] > d[fy])
        {
            if (c != -1) seg.modify(dfn[fx],dfn[x],1,c);
            else
            {
                if (ans1.num == 0) ans1 = seg.query(dfn[fx],dfn[x],1,'L'); //如果是从起点开始合并
                else ans1.merge(ans1,seg.query(dfn[fx],dfn[x],1,'L'));
            }
            x = fa[fx];
            fx = top[x];
        }
        else
        {
            if (c != -1) seg.modify(dfn[fy],dfn[y],1,c);
            else
            {
                if (ans2.num == 0) ans2 = seg.query(dfn[fy],dfn[y],1,'R');//如果是从终点开始合并
                else ans2.merge(seg.query(dfn[fy],dfn[y],1,'R'),ans2);
            }
            y = fa[fy];
            fy = top[y];
        }
    }
    if (x == y)//如果两条路径相交在同一个点
    {
        if (c != -1) return 0;
        if (ans1.num == 0) return ans2.num;
        else if (ans2.num == 0) return ans1.num;
        ans1.merge(ans1,ans2);
        return ans1.num;
    }
    if (d[x] < d[y])//y在下面
    {
        if (c != -1)
        {
            seg.modify(dfn[x] + 1,dfn[y],1,c);
            return 0;
        }
        if (ans1.num == 0) ans1 = seg.query(dfn[x] + 1,dfn[y],1,'R');
        else ans1.merge(ans1,seg.query(dfn[x] + 1,dfn[y],1,'R'));
        if (ans2.num == 0) return ans1.num;
        else ans1.merge(ans1,ans2);
    }
    else//x在下面
    {
        if (c != -1)
        {
            seg.modify(dfn[y] + 1,dfn[x],1,c);
            return 0;
        }
        if (ans1.num == 0) ans1 = seg.query(dfn[y] + 1,dfn[x],1,'L');
        else ans1.merge(ans1,seg.query(dfn[y] + 1,dfn[x],1,'L'));
        if (ans2.num != 0) ans1.merge(ans1,ans2);
    }
    return ans1.num;
}

int main()
{
    int n,p;
    while (scanf("%d%d",&n,&p) != EOF)
    {
        for (int i = 1,u,v,t;i<=n-1;i++)
        {
            scanf("%d%d%d",&u,&v,&t);
            w[make_pair(u,v)] = t;
            w[make_pair(v,u)] = t;
            g[u].push_back(v);
            g[v].push_back(u);
        }
        dfs1(1);
        dfs2(1,1);
        char c[10];
        seg.build(1,n,1);
        while (p--)
        {
            scanf("%s",&c);
            if (c[0] == 'Q')
            {
                int u,v;
                scanf("%d%d",&u,&v);
                if (u == v) printf("0\n");
                else printf("%d\n",mapping(u,v,-1));
            }
            else if (c[0] == 'C')
            {
                int u,v,t;
                scanf("%d%d%d",&u,&v,&t);
                int temptemp = mapping(u,v,t);
            }
        }
        for (int i = 1;i<=n;i++) g[i].clear();
        mem(son); mem(size1); mem(d); mem(fa); mem(dfn); mem(top); mem(aa);
        num = 0;
        w.clear();
    }
}
```

1.3 点分治

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

```
#define ll long long
const int MAXN = 2e4 + 10;

struct node{
    int v,w,nx;
}edge[MAXN<<1];

int tot,head[MAXN];

inline void add(int u,int v,int w)
{
    edge[++tot].v = v;
    edge[tot].w = w;
    edge[tot].nx = head[u];
    head[u] = tot;
}

int sz[MAXN],maxsubtree,root,size,vis[MAXN];//vis标记当前这个点有没有被作为分治点

void getroot(int x,int fa)//找重心，把它作为当前树的根，是根据定义来求的
{
    int maxn = 0;
    sz[x] = 1;
    for (int i = head[x];i;i = edge[i].nx)
    {
        int v = edge[i].v;
        if (v == fa || vis[v]) continue;
        getroot(v,x);
        sz[x] += sz[v];
        maxn = max(maxn,sz[v]);//记录以x为根的最大子树大小
    }
    maxn = max(maxn,size - sz[x]);//当以x为根时，其祖先结点也变成了x的子树
    if (maxn < maxsubtree)//寻找最大子树最小
    {
        maxsubtree = maxn;
        root = x;
    }
}

ll ans = 0,sum[4];

void dfs1(int x,int fa,int st)//统计子树中到分治点（重心）对3取模的路径有几条
{
    for (int i = head[x];i;i = edge[i].nx)
    {
        int v = edge[i].v;
        if (v == fa || vis[v]) continue;
        sum[(st + edge[i].w) % 3]++;
        dfs1(v,x,(st + edge[i].w) % 3);
    }
}

ll cal(int x,int st)//计算路径数量
```

```
{
   sum[0] = sum[1] = sum[2] = 0;
   sum[st]++;
   ll res = 0;
   dfs1(x,0,st);//以重心为起始点，跑其子树的dfs，得到 到重心的路径权值%3为0,1,2的 数量
   res = sum[1] * sum[2] * 2 + sum[0] * sum[0];//统计答案
   return res;
}

void dfs(int x)
{
   ans += cal(x,0);//计算经过当前点的路径的数量
   vis[x] = 1;
   for (int i = head[x];i;i = edge[i].nx)
   {
      int v = edge[i].v;
      if (vis[v]) continue;
      ans -= cal(v,edge[i].w);//容斥，去掉统计答案时子树中互相组成路径没有经过重心（分治点）的路径数
      //之所以要加上edge[i].w是因为 以重心为根，计算子树到重心的路径权值的时候，加上了这条边
      //所以以其儿子点为根计算时也要加上这条边
      maxsubtree = 2147483647; size = sz[v];
      getroot(v,0);//寻找子树的重心
      dfs(root);//以子树的重心为根分治子树
   }
}

int main()
{
   int n;
   scanf("%d",&n);
   for (int i = 1,u,v,w;i<n;i++)
   {
      scanf("%d%d%d",&u,&v,&w);
      add(u,v,w % 3);
      add(v,u,w % 3);
   }
   maxsubtree = 2147483647; size = n;
   getroot(1,0);//先以1为根，寻找树的重心
   dfs(root);//依照重心分治
   ll t = (ll)n * (ll)n;
   ll g = __gcd(ans,t);
   printf("%lld/%lld",ans / g,t / g);
   return 0;
}
```

1.4 长链剖分

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

```

```
const int MAXN = 1e6 + 10;

struct node{
    int u,v,nx;
}edge[MAXN<<1];

int tot,head[MAXN];

inline void add(int u,int v)
{
    edge[++tot].v = v;
    edge[tot].nx = head[u];
    head[u] = tot;
}

int len[MAXN],son[MAXN],fa[MAXN],num;
//找长链
void dfs1(int x)
{
    len[x] = 1;
    for (int i = head[x];i;i = edge[i].nx)
    {
        int v = edge[i].v;
        if (v == fa[x]) continue;
        fa[v] = x;
        dfs1(v);
        len[x] = max(len[x],len[v] + 1);
        if (len[v] > len[son[x]]) son[x] = v;
    }
}

int *dp[MAXN],tmp[MAXN],*id = tmp;
int ans[MAXN];

void dfs2(int x)
{
    dp[x][0] = 1;
    if (son[x])//沿长链dp
    {
        dp[son[x]] = dp[x] + 1;
        dfs2(son[x]);
        ans[x] = ans[son[x]] + 1;
    }
    for (int i = head[x];i;i = edge[i].nx)
    {
        int v = edge[i].v;
        if (v == fa[x] || v == son[x]) continue;
        dp[v] = id;
        id += len[v];
        dfs2(v);
        //合并两条链
        for (int j = 0;j<len[v];j++)
        {
            dp[x][j + 1] += dp[v][j];
            if (dp[x][j+1] > dp[x][ans[x]] || (dp[x][j+1] == dp[x][ans[x]] && j + 1 < ans[x]))
            ans[x] = j + 1;
        }
    }
    if (dp[x][ans[x]] == 1) ans[x] = 0;
}

int main()
{
    int n;
    cin>>n;
    for (int i = 1,u,v;i<n;i++)
    {
        scanf("%d%d",&u,&v);
        add(u,v);
        add(v,u);
    }
    dfs1(1);
    dp[1] = id; id += len[1];
    dfs2(1);
    for (int i = 1;i<=n;i++) printf("%d\n",ans[i]);
}
```

2 线段树

2.1 zkw 线段树

```
//zkw单点修改，区间查询
#include<iostream>
#include<stdio.h>
#include<cstring>
#include<math.h>
using namespace std;
const int NUM = 50000;

int m,tree[NUM * 4];

void modify(int n,int v)
{
    for (tree[n += m] += v,n>>=1;n;n>>=1) tree[n] = tree[n * 2] + tree[n * 2 + 1];
}

int query(int st,int ed)
{
    int ans = 0;
    for (int l = m + st - 1,r = m + ed + 1;l ^ r ^ 1;l>>=1,r>>=1)
    {
        if (~l & 1) ans += tree[l ^ 1];
        if (r & 1) ans += tree[r ^ 1];
    }
    return ans;
}

int main()
{
    memset(tree,0,sizeof(tree));
    int n;
    cin>>n;
    m = log(n) / log(2);
    if (pow(2,m) < n) m++;
    m = pow(2,m);
    for (int i = m + 1;i<=m + n;i++) scanf("%d",&tree[i]);
    for (int i = m - 1;i;i--) tree[i] = tree[i * 2] + tree[i * 2 + 1];
    char c[10];
    while (1)
    {
        //E结束 Query询问x~y的区间和 Add在x处+y Sub在x处-y
        int x,y;
        scanf("%s",c);
        if (c[0] == 'E') break;
        scanf("%d%d",&x,&y);
        if (c[0] == 'Q') printf("%d\n",query(x,y));
        else if (c[0] == 'A') modify(x,y);
        else modify(x,-y);
    }
    return 0;
}
```

2.2 树状数组维护区间和

```
//树状数组维护区间和
#include<iostream>
#define ll long long
using namespace std;

int lowbit(int x)
{
    return x&-x;
}

int n;
ll a[50001] = {},b[50001] = {};

void change(ll x,ll w)
{
    while (x <= n)
    {
        b[x] += w;
        x +=lowbit(x);
    }
}

```

```
ll sum(int x)
{
    ll num = 0;
    while (x > 0)
    {
        num+=b[x];
        x -=lowbit(x);
    }
    return num;
}

int main()
{
    scanf("%d",&n);
    for (int i = 1;i<=n;i++)
    {
        scanf("%d",&a[i]);
        a[i] += a[i-1];
        b[i] = a[i] - a[i - lowbit(i)];
    }
    char c[10] = {};
    scanf("%s",c);
    while (c[0] != 'E')
    {
        //E结束 Query询问x~y的区间和 Add在x处+y Sub在x处-y
        int i,j;
        scanf("%d%d",&i,&j);
        if (c[0] == 'Q')
        {
            ll t = sum(j) - sum(i-1) ;
            printf("%lld\n",t);
        }
        else if (c[0] == 'A') change(i,j);
        else change(i,-j);
        scanf("%s",c);
    }
}
```

2.3 树状数组维护区间最大值

```
//树状数组维护区间最大值(不可有修改操作)
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const int MAXN = 3e6 + 10;
#define ll long long

ll treemax[MAXN],a[MAXN];
int n;

inline int lowbit(int x)
{
    return x & -x;
}

void insert(ll v,int x)
{
    while (x <= n)
    {
        treemax[x] = max(v,treemax[x]);
        x += lowbit(x);
    }
}

ll query(int l,int r)
{
    ll res = -9e18;
    while (r >= l)
    {
        if (r - lowbit(r) < l)
        {
            res = max(res,a[r]);
            r--;
        }
        else
        {
            res = max(res,treemax[r]);
            r -=lowbit(r);
        }
    }
    return res;
}

int main()
{
    int m;
    scanf("%d%d",&n,&m);
    fill(treemax,treemax+MAXN,-9e18);
    for (int i = 1;i<=n;i++)
    {
        scanf("%lld",&a[i]);
        insert(a[i],i);
    }
    int l,r;
    while (m--)
    {
        scanf("%d%d",&l,&r);
        printf("%lld\n",query(l,r));
    }
    return 0;
}
```

2.4 主席树

```
//询问区间第k大
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 2e5 + 10;//数组大小

int mapping[MAXN],root[MAXN];

struct node{
    int val,index,num;
}a[MAXN];

bool cmp1(node i,node j){return i.val < j.val;}
bool cmp2(node i,node j){return i.index < j.index;}

struct tree{
    int l[MAXN<<5],r[MAXN<<5],num[MAXN<<5],numNode;
    int update(int pre,int L,int R,int x)//新增一条从根节点到叶子的链
    {
        int cur = ++numNode;
        l[cur] = l[pre],r[cur] = r[pre],num[cur] = num[pre] + 1;
        //延用上一个结点的左右子节点
        //num存从序列开始到现在插入的这个值的位置，该节点下面由多少个值
        if (L < R)
        {
            if (x <= (L + R) / 2) l[cur] = update(l[pre],L,(L+R)/2,x);
            //如果x的值在左边，则新建一个左结点连接在该节点上
            else r[cur] = update(r[pre],(L + R) / 2 + 1,R,x);
            //如果x的值在右边，则新建一个右结点连接在该节点上
        }
        return cur;
    }
    int query(int st,int ed,int L,int R,int k)
    {
        int sum = num[l[ed]] - num[l[st]];//区间内的数值在左边的个数
        if (L == R) return L;
        if (sum >= k) return query(l[st],l[ed],L,(R + L) / 2,k);
        //左边的值的个数比k大，则第k大的值在左边
        else return query(r[st],r[ed],(R + L) / 2 + 1,R,k - sum);
        //否则第k大的值在右边
    }
}zxs;

int main()
{
    int n,m;
    scanf("%d%d",&n,&m);
    for (int i = 1;i<=n;i++) scanf("%d",&a[i].val),a[i].index = i;
    sort(a+1,a+n+1,cmp1);
    int count = 0;
    for (int i = 1;i<=n;i++)
    {
        if (a[i].val != a[i-1].val || i == 1) mapping[a[i].num = ++count] = a[i].val;
```

```
      else a[i].num = count;
   }
   sort(a+1,a+n+1,cmp2);
   //离散化，mapping表示离散化后的值对应原来的值是哪个
   for (int i = 1;i<=n;i++)
      root[i] = zxs.update(root[i-1],1,count,a[i].num);//建树
   while (m--)
   {
      int l,r,k;
      scanf("%d%d%d",&l,&r,&k);
      printf("%d\n",mapping[zxs.query(root[l-1],root[r],1,count,k)]);
      //返回[l,r]第k大的值
   }
   return 0;
}
```

3 图论

3.1 最短路

3.1.1 dijkstra 求最短路

```
//有向图
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define PII pair<ll,ll>

const int Vertex_MAXN = 1e4 + 10;
const int Edge_MAXN = 5e5 + 10;
const ll inf = 9e18;

struct node{
   int v,nx;
   ll w;
}edge[Edge_MAXN];

int head[Vertex_MAXN],tot,n,m,s;
ll dis[Vertex_MAXN];

inline void add(int u,int v,ll w)
{
   edge[++tot].v = v;
   edge[tot].w = w;
   edge[tot].nx = head[u];
   head[u] = tot;
}

void dij(int s) {
   priority_queue<PII,vector<PII>,greater<PII>> q;
   fill(dis,dis+n+1,inf);
   dis[s] = 0;
```

```
    q.push(PII(0,s));
    while (!q.empty())
    {
        ll d = q.top().first;
        int u = q.top().second;
        q.pop();
        if (d != dis[u]) continue;
        for (int i = head[u];i;i = edge[i].nx)
        {
            PII y(d+edge[i].w,edge[i].v);
            if (dis[y.second]>y.first)
            {
                dis[y.second] = y.first;
                q.push(y);
            }
        }
    }
}

int main()
{
    scanf("%d%d%d",&n,&m,&s);
    for (int i = 1,u,v,w;i<=m;i++)
    {
        scanf("%d%d%d",&u,&v,&w);
        add(u,v,w);
    }
    dij(s);
    for (int i = 1;i<=n;i++) printf("%lld ",dis[i]);
    return 0;
}
```

3.1.2 spfa 求最短路

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

const int MAXN = 1e4 + 10;

struct node{
    int v,nx,w;
}edge[500010];

int head[MAXN],tot;

inline void add(int u,int v,int w)
{
    edge[++tot].v = v;
    edge[tot].nx = head[u];
    edge[tot].w = w;
    head[u] = tot;
}
```

```

int dis[MAXN],vis[MAXN],s;

void spfa()
{
    queue<int> q;
    fill(dis,dis+MAXN,2147483647);
    q.push(s);
    vis[s] = 1;
    dis[s] = 0;
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for (int i = head[u];i;i = edge[i].nx)
        {
            int v = edge[i].v;
            if (dis[v] > dis[u] + edge[i].w)
            {
                dis[v] = dis[u] + edge[i].w;
                if (!vis[v]) vis[v] = 1,q.push(v);
            }
        }
    }
}

int main()
{
    int n,m;
    scanf("%d%d%d",&n,&m,&s);
    for (int i = 1,u,v,w;i<=m;i++)
    {
        scanf("%d%d%d",&u,&v,&w);
        add(u,v,w);
    }
    spfa();
    for (int i = 1;i<=n;i++) printf("%d ",dis[i]);
    return 0;
}
```

3.2 网络流

3.2.1 dinic

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

const int MAXN = 2e3;
const int inf = 2147483647;

int s,t;
```

```

struct node{
    int v,w,nx;
}edge[MAXN];

int tot,head[MAXN];

inline void add(int u,int v,int w)
{
    edge[tot].v = v;
    edge[tot].nx = head[u];
    edge[tot].w = w;
    head[u] = tot++;
}

int dis[MAXN];
queue<int> q;

bool bfs()
{
    memset(dis,-1,sizeof(dis));
    dis[s] = 0;
    q.push(s);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (int i = head[u];i != -1;i = edge[i].nx)
        {
            int v = edge[i].v;
            if (dis[v] == -1 && edge[i].w > 0)
            {
                dis[v] = dis[u] + 1;
                q.push(v);
            }
        }
    }
    return dis[t] != -1;
}

int dfs(int u,int exp)
{
    if (u == t) return exp;
    int flow = 0,tmp = 0;
    for (int i = head[u];i != -1;i = edge[i].nx)
    {
        int v = edge[i].v;
        if (dis[v] == dis[u] + 1 && edge[i].w > 0)
        {
            tmp = dfs(v,min(exp,edge[i].w));
            if (!tmp) continue;
            exp -= tmp;
            flow += tmp;
            edge[i].w -= tmp;
            edge[i^1].w += tmp;
            if (!exp) break;
        }
    }
    return flow;
}

void dinic()
{
    int ans = 0;
    while (bfs()) ans += dfs(s,inf);
    printf("%d",ans);
}

int main()
{
    int n;
    scanf("%d",&n);
    memset(head,-1,sizeof(head));
    for (int i = 1,u,v,w;i<=n;i++)
    {
        scanf("%d%d%d",&u,&v,&w);
        add(u,v,w);
        add(v,u,0);
    }
    s = 65,t = 90;
    dinic();
    return 0;
}
```

3.3 二分图

3.3.1 二分图 KM 板子

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

const int MAXN = 3e2 + 10;

int a[MAXN][MAXN],d,n;
int c_girl[MAXN],c_boy[MAXN];//记录匹配对象
int ex_girl[MAXN],ex_boy[MAXN];//记录男生和女生的期望
bool vis_girl[MAXN],vis_boy[MAXN];//记录每一轮匹配匹配过的女生和男生
int slack[MAXN]; // 记录每个汉子如果能被妹子倾心最少还需要多少期望值

bool dfs(int u)//匈牙利算法找增广路径
{
    vis_girl[u] = 1;
    for (int i = 1;i<=n;i++)
    {
        if (vis_boy[i] == 0)//每一轮匹配 每个男生只尝试一次
        {
            int num = ex_girl[u] + ex_boy[i] - a[u][i];
            if (num == 0)//如果符合要求
            {
                vis_boy[i] = 1;
                if (c_boy[i] == 0 || dfs(c_boy[i])) //找到一个没有匹配的男生 或者该男生的妹子可以找到其他人
                {
                    c_girl[u] = i;
                    c_boy[i] = u;
                    return 1;
                }
            }
            else slack[i] = min(slack[i],num);//slack可以理解为该男生要得到女生的倾心 还需多少期望值 取最小值
        }
    }
    return 0;
}

int KM()
{
    memset(ex_girl,0,sizeof(ex_girl));// 每个女生的初始期望值是与她相连的男生最大的好感度
    memset(ex_boy,0,sizeof(ex_boy));// 初始每个男生的期望值为0
    memset(c_girl,0,sizeof(c_girl));// 初始每个男生都没有匹配的女生
    memset(c_boy,0,sizeof(c_boy));// 初始每个女生都没有匹配的男生
    for (int i = 1;i<=n;i++)
    for (int j = 1;j<=n;j++)
    ex_girl[i] = max(ex_girl[i],a[i][j]);// 每个女生的初始期望值是与她相连的男生最大的好感度

    for (int i = 1;i<=n;i++) // 尝试为每一个女生解决归宿问题
    {
        fill(slack,slack + MAXN,2147483647);// 因为要取最小值 初始化为无穷大
        while (true) // 为每个女生解决归宿问题的方法是 ： 如果找不到就降低期望值，直到找到为止
        {
            memset(vis_girl,0,sizeof(vis_girl));
            memset(vis_boy,0,sizeof(vis_boy));// 记录每轮匹配中男生女生是否被尝试匹配过
            if (dfs(i)) break;// 找到归宿 退出

             // 如果不能找到 就降低期望值
            // 最小可降低的期望值
            d = 2147483647;
            for (int j = 1;j<=n;j++)
            {
                if (!vis_boy[j]) d = min(d,slack[j]);
            }
            for (int j = 1;j<=n;j++)
            {
                if (vis_girl[j] == 1) ex_girl[j] -= d;//所有访问过的女生降低期望值
                if (vis_boy[j] == 1) ex_boy[j] += d;//所有访问过的男生增加期望值
                else slack[j] -= d; //没有访问过的boy 因为girl们的期望值降低，距离得到女生
```

```
                      倾心又进了一步!
            }
        }
    }
    // 匹配完成 求出所有配对的好感度的和
    int res = 0;
    for (int i = 1;i<=n;i++) res += a[i][c_girl[i]];
    return res;
}

int main()
{
    cin>>n;
    for (int i = 1;i<=n;i++)
    for (int j = 1;j<=n;j++)
    scanf("%d",&a[i][j]);
    cout<<KM()<<'\n';
}
```

3.3.2 二分图最大匹配

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

const int MAXN = 1e3;

int vis[MAXN][MAXN],cx[MAXN],cy[MAXN],n,m;
bool check[MAXN];

bool dfs(int u)
{
    for (int v = 1;v<=n;v++)
    {
        if (vis[u][v] && !check[v])//邻接矩阵存储,如果u-v之间有一条路 而且 v没有被访问过
        {
            check[v] = 1;//标记v已经访问
            if (cy[v] == -1 || dfs(cy[v]))
            {
                cx[u] = v;
                cy[v] = u;
                return 1;
            }
        }
    }
    return 0;
}

int maxmatch()
{
    int ans = 0;
    memset(cx,-1,sizeof(cx));
    memset(cy,-1,sizeof(cy));
    for (int i = 1;i<=n;i++)//字典序从大到小
    {
        if (cx[i] == -1)
        {
            memset(check,0,sizeof(check));
            ans += dfs(i);
        }
    }
    return ans;
}

int main()
{
    cin>>n;
    for (int i = 1;i<=n;i++)
    for (int j = 1;j<=n;j++)
    cin>>vis[i][j];
    int ans = maxmatch();
    cout<<ans;
}
```

3.4 tarjan

3.4.1 tarjan 求环的个数和点的染色

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 2e2 + 10;

vector<int> g[MAXN];
int dfn[MAXN],low[MAXN],s[MAXN],vis[MAXN],num,slen,scnt,col[MAXN];

void tarjan(int u)
{
    low[u] = dfn[u] = ++num;
    //low存u的子树里所能到达的dfn最小的点
    s[++slen] = u;//s为栈
    vis[u] = 1;//标记u已经放到了栈里
    for (int i = 0;i<g[u].size();i++)
    {
        int v = g[u][i];
        if (!dfn[v])//如果v没有访问过
        {
            tarjan(v);
            low[u] = min(low[u],low[v]);
        }
        else if (vis[v]) low[u] = min(low[u],dfn[v]);
        //一旦遇到已入栈的点，就将该点作为连通量的根
        //这里用dfn[e[i].v]更新的原因是：这个点可能
        //已经在另一个强连通分量中了但暂时尚未出栈
        //所以now不一定能到达low[e[i].v]但一定能到达
        //dfn[e[i].v].
    }
    if (dfn[u] == low[u])
    {
        scnt++;//环的数量
        do
        {
            vis[s[slen]] = 0;//出栈
            col[s[slen]] = scnt;//染色
        }while (s[slen--] != u);
    }
}

int main()
{
    int p;
    cin>>p;
    while (p--)
    {
        int n,m;
        scanf("%d%d",&n,&m);
        for (int i = 1,u,v;i<=m;i++)
        {
            scanf("%d%d",&u,&v);
            g[u].push_back(v);
        }
        for (int i = 0;i<n;i++)
        if (dfn[i] == 0) tarjan(i);
        for (int i = 0;i<n;i++) g[i].clear();
        printf("%d\n",scnt);
        memset(dfn,0,sizeof(dfn));
        memset(low,0,sizeof(low));
        memset(s,0,sizeof(s));
        memset(vis,0,sizeof(vis));
        scnt = num = 0;
    }
}
```

3.4.2 tarjan 求无向图割边和割点

```
//tarjan求无向图割边和割点
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 2e4 + 10;

vector<int> g[MAXN];

int dfn[MAXN],low[MAXN],num,fa[MAXN];

struct EDGE{
    EDGE(int a = 0,int b = 0):u(a),v(b){}
    int u,v;
```

```
};

vector<EDGE> cutedge;
vector<int> cutnode;

bool cmp(EDGE a,EDGE b) {return a.u == b.u?a.v<b.v:a.u<b.u;}

void tarjan(int u)
{
    dfn[u] = low[u] = ++num;
    bool flag = false;
    int son = 0;
    for (int i = 0;i<g[u].size();i++)
    {
        int v = g[u][i];
        if (v == fa[u]) continue;
        if (!dfn[v])
        {
            son++;
            fa[v] = u;
            tarjan(v);
            low[u] = min(low[u],low[v]);
            if (low[v] >= dfn[u]) flag = true;
            //判断是否存在子节点只能通过u访问到u的祖先
            if (low[v] > dfn[u]) cutedge.push_back(EDGE(min(u,v),max(v,u)));
            //判断割边
        }
        else low[u] = min(low[u],dfn[v]);
    }
    if ((fa[u] == 0 && son >= 2) || (fa[u] != 0 && flag)) cutnode.push_back(u);
    //判断割点
    //u是根节点且u有两个连通分量则u是割点
    //u不是根节点，u存在一个子节点只能通过u访问到u的祖先
}

int main()
{
    int n,m;
    scanf("%d%d",&n,&m);
    for (int i = 1,u,v;i<=m;i++)
    {
        scanf("%d%d",&u,&v);
        g[u].push_back(v);
        g[v].push_back(u);
    }
    for (int i = 1;i<=n;i++)
    if (dfn[i] == 0) tarjan(i);
    sort(cutedge.begin(),cutedge.end(),cmp);
    sort(cutnode.begin(),cutnode.end());
    //割点从小到大输出
    //割边(u,v)，u<v，按照u为第一关键字，v为第二关键字排序
    if (cutnode.size() == 0) printf("Null\n");
    else
    {
        printf("%d",cutnode[0]);
        for (int i = 1;i<cutnode.size();i++) printf(" %d",cutnode[i]);
    }
    puts("");
    for (int i = 0;i<cutedge.size();i++)
    {
        printf("%d %d\n",cutedge[i].u,cutedge[i].v);
    }
    return 0;
}
```

4 字符串

4.1 Manacher

```
#include <bits/stdc++.h>
#define maxn 2000005
using namespace std;
int mp[maxn];
string str;
char c[maxn];
void Manacher(string s,int len){
    int l=0,R=0,C=0;;
    c[l++]='$', c[l++]='#';
    for(int i=0;i<len;i++){
        c[l++]=s[i], c[l++]='#';
    }
    for(int i=0;i<l;i++){
        mp[i]=R>i?min(mp[2*C-i],R-i):1;
        while(i+mp[i]<l&&i-mp[i]>0){
            if(c[i+mp[i]]==c[i-mp[i]]) mp[i]++;
            else break;
        }
        if(i+mp[i]>R){
            R=i+mp[i], C=i;
        }
    }
}
int main()
{
    int cnt=0;
    while(cin>>str){
        if(str=="END") break;
        int len=str.length();
        Manacher(str,len);
        int ans=0;
        for(int i=0;i<2*len+4;i++){
            ans=max(ans,mp[i]-1);
        }
        printf("Case %d: %d\n",++cnt,ans);
    }
    return 0;
}
```