图论模板

最短路

1.dijstra（单源无负权）

朴素dij：稠密图 O(n^2)（m>>n)

模板

```
#include <bits/stdc++.h>
//#define LOCAL
using namespace std;
//朴素的dijkstra   稠密图
const int inf = 0x3f3f3f3f;
int m, n, Map[505][505], dis[505], vis[505], S, T;
void init() {
    memset(Map, 0x3f, sizeof Map); //初始化正无穷
    memset(vis, 0, sizeof vis);
    memset(dis, 0x3f, sizeof dis);
}
void dij(int S, int T) {
    dis[S] = 0; //
    vis[S] = 1;
    while (S != T) { //从小到大找最短路，直到找到T就无需继续
        int Min = inf;
        int next;
        for (int i = 1; i <= n; i++) { //**每次从当前起点找其的最短路
            if (Map[S][i] != inf&&!vis[i]) //存在道路
                dis[i] = min(dis[i], Map[S][i] + dis[S]); //**对每个相连点松弛操作
            if ( dis[i] < Min) { //更新当前最短路
                next = i;            //找下一个最近的 做起点
                Min = dis[i];
            }
        }
        if (Min == inf)  break; //之后都没有道路，跳出
        S = next; //更新起点以做下一个松弛操作
        vis[S] = 1; //标记已生成过
    }
}
int main() {
    while (cin >> n >> m, m + n) {
        init();
        for (int i = 1; i <= m; i++) {
            int a, b, dist;
            cin >> a >> b >> dist;
            Map[a][b] = min(dist, Map[a][b]);
            //  Map[b][a] = Map[a][b]; //无向图
        }
```

```
        S = 1;
        T = n;
        dis[S] = 0; //
        vis[S] = 1;
        dij(S, T);
        if (dis[T] == inf)cout << "-1" << endl;
        else    cout << dis[T] << endl;
    }

    return 0;
}
```

堆优化的dij： O(mlogn) //一般最优

```
#include <bits/stdc++.h>
#define LOCAL
#define int long long
using namespace std;
const int inf =0x3f3f3f3f;
const int N=1e5+5;
const int M=2e5+5;
int m,n,s,tot;
int head[N],vis[N],dis[N];

struct Edge{
  int from,to,w,next;
}edge[M];
void add(int u,int v,int w){
  edge[++tot].to =v;
  edge[tot].from =u;
  edge[tot].w=w;
  edge[tot].next =head[u];
  head[u]=tot;
}
struct node{// 点  用于 优先队列
int w,id;
bool operator < (const node &b)const{  //小顶堆
  return w>b.w;
}
};
void dij(int s){
    priority_queue<node>que;
    dis[s]=0;
    que.push({0,s});
    while(!que.empty()){
      node u=que.top();que.pop();
      int cur=u.id;//当前点序号
      if(vis[cur])continue;// 只入队一次
```

```
      vis[cur]=1;
      for(int i=head[cur];~i;i=edge[i].next){
          int v=edge[i].to;
          int dist=edge[i].w;
          if(dis[v]>dis[cur]+dist){
            dis[v]=dis[cur]+dist;
            que.push({dis[v],v});
          }
      }
    }
}
signed main(){
#ifdef LOCAL
    freopen("data.in","r",stdin);
    freopen("data.out","w",stdout);
#endif
cin>>n>>m>>s;
tot=0;
memset(dis,0x3f,sizeof dis);
memset(vis,0,sizeof vis);
memset(head,-1,sizeof head);
for(int i=1;i<=m;i++){
  int u,v,c;
  cin>>u>>v>>c;
  add(u,v,c);
}
dij(s);
for(int i=1;i<n;i++)cout<<dis[i]<<" ";cout<<dis[n]<<endl;
    return 0;
}
```

bellman-ford 与spfa： 单源 存在负权边

spfa*** 最坏O(nm) 一般快很多

```
#include <bits/stdc++.h>
#define LOCAL
#define int long long
using namespace std;
const int N = 1e4 + 5;
const int M = 5e5+5;
const int inf = 2147483647;
int m, n, s;                                    // vis标记入队
int head[N], vis[N], dis[N], tot, neg[N];  //判断负环  每个点松弛次数不超过n-1
struct node {
  int from, to, next, w;
} edge[M];
void add(int u, int v, int w) {
  edge[++tot].from = u;
  edge[tot].to = v;
```

```
  edge[tot].w = w;
  edge[tot].next = head[u];
  head[u] = tot;
}
void init() {
  tot = 0;
  memset(head, -1, sizeof head);
  memset(vis, 0, sizeof vis);
  memset(neg, 0, sizeof neg);
}
int spfa(int s) {
  for (int i = 1; i <= n; i++) dis[i] = inf;
  dis[s] = 0;
  queue<int> que;
  que.push(s);
  vis[s] = 1;
  while (!que.empty()) {
    int u = que.front();
    que.pop();
    vis[u] = 0;
    for (int i = head[u]; ~i; i = edge[i].next) {
      int v = edge[i].to;
      int dist = edge[i].w;
      if (dis[v] > dis[u] + dist) {  //松弛操作
        dis[v] = dis[u] + dist;
        if (!vis[v]) {  //只要不在队中就入队      可能多次
          vis[v] = 1;
          que.push(v);
  //        neg[v]++;
 //         if (neg[v] > n) return -1;  //判断负环;
        }
      }
    }
  }
  return 0;
}
signed main() {

  init();
  cin >> n >> m >> s;
  for (int i = 1; i <= m; i++) {
    int a, b, c;
    cin >> a >> b >> c;
    add(a, b, c);
    // add(b,a,c);    //无向图
  }
  int ans = spfa(s);
  if (ans == 0) {
    for (int i = 1; i <= n; i++) {
      cout << dis[i] << " ";
    }
  } else
```

```
    cout << "-1";
  return 0;
}
```

判负环：

floyed： 多源最短路 O(n^3) 可存在负环

```
#include <bits/stdc++.h>
//#define LOCAL
//#define int long long
using namespace std;
//floyed 求多源最短路
// 原理  用每个点 松弛每条边
const int N=205;
const int inf=0x3f3f3f3f;
int n,m,K;
int dis[N][N];
void init(){
  for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j++){
      if(i==j)dis[i][j]=0;
      else dis[i][j]=inf;
    }
  }
}
signed main(){
cin>>n>>m>>K;//m条边 k个询问
init();
for(int i=1;i<=m;i++){
  int u,v,w;
  cin>>u>>v>>w;
  dis[u][v]=min(dis[u][v],w);//   u==v 的特殊情况
}
//floyed
for(int k=1;k<=n;k++){
  for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j++)
    dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
  }
}
//
for(int i=1;i<=K;i++){
  int a,b;
  cin>>a>>b;
 if(dis[a][b]>inf/2) puts("impossible");// inf/2  ---负环
  else cout<<dis[a][b]<<endl;
}
    return 0;
}
```

最小生成树：

kruskal：O(mlogm) 稀疏图

并查集： 每次加入最短边

```
#include <bits/stdc++.h>
//#define LOCAL
//#define int long long
//Kruskal    并查集  + 边排序
const int N=1e3+5;
const int M=2e3+5;
int n,m;
int pre[N];
int find(int x){
  if(x!=pre[x])pre[x]=find(pre[x]);
  return pre[x];
}
struct Edge{
  int u,v,w;
  bool operator <(const Edge &b)const{
    return w<b.w;
  }
}edge[M];
using namespace std;
signed main(){

cin>>n>>m;
for(int i=1;i<=n;i++)pre[i]=i;//init
for(int i=1;i<=m;i++){
  int a,b,w;
  cin>>a>>b>>w;
  edge[i]={a,b,w};
}
sort(edge+1,edge+m+1);
int ans=0;
for(int i=1;i<=m;i++){
  int u=find(edge[i].u);int v=find(edge[i].v);
  if(u!=v){
    pre[v]=u;//合并
    ans+=edge[i].w;
  }
}
int cnt=0;//计集合数
for(int i=1;i<=n;i++){
  if(find(i)==i)cnt++;
}
if(cnt==1){
  cout<<ans<<endl;
```

```
}
else cout<<"orz"<<endl;  //不存在

    return 0;
}
```

prim:稠密图 O(n^2) / (堆优化版本) O(mlogn) (没必要)

类似dij

```
#include <bits/stdc++.h>
//#define LOCAL
//  最小生成树 朴素prim算法   非常类似dij算法
//每次找集合外离集合最近的点并标记 更新其他点到集合距离
using namespace std;
const int N=510,inf=0x3f3f3f3f;
int n,m;
int G[N][N];  //存原图所有边
int dist[N];   //点到集合的最短距离
bool vis[N];  //标记是否已经在集合
int prim(){
  memset(dist ,0x3f,sizeof dist);
  int res=0;//  最小生成树答案 边长总和
  int S=1;dist[S]=0;vis[S]=1;//同dij 任选起始点
  for(int i=0;i<n-1;i++){// 将n-1个点做n-1次更新加入集合
    int Min=inf;int ne;// ne 同dij 存最短点
    for(int j=1;j<=n;j++){
      if(!vis[j]){// 未加入集合  **可能与S无边  (与dij不同)
        dist[j]=min(dist[j],G[S][j]);// 松弛  用上一个最近点更新距离
          if(dist[j]<Min){// 更新当前最近的点
              ne=j;Min=dist[j];
          }
      }
    }
    if(Min==inf)return inf;//   不存在
    S=ne;//  赋值给起点
    vis[S]=1;//标记
   res+=Min;
  }
return res;
}
signed main(){

cin>>n>>m;
memset(G,0x3f,sizeof G);
memset(vis,0,sizeof vis);
for(int i=1;i<=m;i++){
    int a,b,c;
```

```
    cin>>a>>b>>c;
    G[a][b]=G[b][a]=min(G[a][b],c);   //无向图  防重边 自环
}
int ans=prim();
if(ans==inf)puts("impossible");
else cout<<ans<<endl;
    return 0;
}
```

二分图（匹配）

定理：

设结点数n

- 二分图最大匹配数==最小顶点覆盖 （m）
- DAG图(有向无环图)最小路径覆盖数最大独立集数n - m

最大独立集 （拆点）
选出最多的点 使得选出的点之间没有边

最小路径覆盖
DAG(有向无环图) （拆点）
用最少的互不相交的路径 将所有点覆盖

匈牙利算法 O(mn)

模板:

```
#include <cstdio>
#include <cstring>
#include <ctime>
#include <iostream>
#define LOCAL
using namespace std;
int k, m, n, G[505][505], link[505];  // G[u][v]   link存右点对象
int vis[505];     //对每次匹配右点访问过与否，防止重复
int dfs(int u) {  //*****从左侧搜
  for (int v = 1; v <= n; v++) {  //对每个右侧点遍历
    if (G[u][v] && !vis[v]) {     //已访问过则无需继续
      vis[v] = 1;
      if (link[v] == -1 ||
          dfs(link[v])) {  //****
                           //若v未匹配则匹配上,返回true,或继续向后搜直到匹配上
        link[v] = u;  // u ,v相匹配,为之后服务
        return 1;     //匹配上返回true
      }
    }
  }
  return 0;//匹配不上返回false
```

```
}
int hungary() {
  int res = 0;                    //最大匹配数
  memset(link, -1, sizeof link);  //初始化link为-1,都未匹配,无对象
  for (int u = 1; u <= m; u++) {  //遍历左侧点向右匹配
    memset(vis, 0, sizeof vis);   //每次 匹配初始化vis 0
    if (dfs(u)) res++;            // dfs搜索,如果搜到匹配数+1;
  }
  return res;
}

int main() {
  int u, v;  //左 右

  while (cin >> k, k) {
    cin >> m >> n;
    memset(G,0,sizeof G);
    for (int i = 1; i <= k; i++) {
      cin >> u >> v;
      G[u][v] = 1;  //此处只考虑女生到男生的边，所以无需 G[v][u]=1;
    }
    int ans = hungary();
    cout<<ans<<endl;
  }
  return 0;
}
```

拓扑序

模板

```
#include <bits/stdc++.h>
#define LOCAL
//#define int long long
// 判断是否有环--找不到入度为0的点
using namespace std;
const int N=3e5+5;
const int M=1e6+5;
struct Edge{
    int from,to,next;
}edge[M];
int head[N];
vector<int>ans;
int T,n,m,tot,in[N];
void add(int u,int v){
    edge[++tot].from=u;
    edge[tot].to=v;
    edge[tot].next=head[u];
    head[u]=tot;
}
```

```
signed main(){
 std::ios::sync_with_stdio(false);
cin.tie(0);
cout.tie(0);
while(cin>>n>>m,m+n){
    tot=0;
    memset(head,-1,sizeof head);
    memset(in,0,sizeof in);
    for(int i=1;i<=m;i++){
        int u,v;
        cin>>u>>v;
        add(u,v);
        in[v]++;
    }
    queue<int>que;
    int cnt=0;
    for(int i=0;i<n;i++){//0kaishi
        if(in[i]==0){
            que.push(i);//
            cnt++;
        }
    }
    while(!que.empty()){
        int u=que.front();que.pop();
        ans.push_back(u);
        for(int i=head[u];~i;i=edge[i].next){
            int v=edge[i].to;
            in[v]--;
            if(in[v]==0){que.push(v);cnt++;}
        }
    }
    // 判环
    if(n==cnt)cout<<"YES"<<endl;
    else {cout<<"NO"<<endl; continue}   //成环
    //输出
    for(auto i:ans)cout<<i<<' ';cout<<endl;
}
    return 0;
}
```

有向图强连通分量（缩点）

kosaraju:

两次dfs 正图+反图 缩点（编号不连续）

```
//原理:欲通过dfs求强连通分量,必须按拓扑序遍历,所以两次dfs,第一次先求得拓扑序,再求得scc
#include <bits/stdc++.h>
#define LOCAL
using namespace std;
```

```
//强连通分量 模板
//1.vector
const int MAX=100005;
int t,n,m;
vector<int> G[MAX],G2[MAX],G3[MAX];//原图，反图,新图
vector<int>S;// 部分拓扑序
int vis[MAX],index[MAX];//每个点所属scc编号(缩点编号)
int scc_cnt;//scc数
//若要求每个点强连通分量数,通过index数组遍历每个点计数即可
void dfs1(int u ){
    if(vis[u])return;
    vis[u]=1;
    for(int i=0;i<G[u].size();i++)dfs1(G[u][i]);//继续dfs
    S.push_back(u);//得部分拓扑序
}
void dfs2(int u){
  if(index[u])return ;//相当于标记，
  index[u]=scc_cnt;//计数
  for(int i=0;i<G2[u].size();i++)dfs2(G2[u][i]);
}
void build(){//缩点后新图
  for(int i=1;i<=n;i++){//遍历
    for(int j=0;j<G[i].size();j++){
      int v=G[i][j];
      if(index[i]!=index[v]){//不为同一个scc
        G3[index[i]].push_back(index[v]);
      }
    }
  }
}
void find_scc(int n){
  scc_cnt=0;
  S.clear();
  memset(index,0,sizeof index);
  memset(vis,0,sizeof vis);
  for(int i=0;i<n;i++)dfs1(i);
  for(int i=n-1;i>=0;i--){//逆拓扑序遍历反图
    if(!index[S[i]]){//相当于标记
      scc_cnt++;//scc数
      dfs2(S[i]);
    }
  }
}
int main(){

cin>>t;
while(t--){
cin>>n>>m;
  for(int i=0;i<=n;i++){
    G[i].clear();
    G2[i].clear(); G3[i].clear();
  }
```

```
for(int i=1;i<=m;i++){
  int u,v;
  cin>>u>>v;
  G[u].push_back(v);
 G2[v].push_back(u);
}
find_scc(n);
cout<<scc_cnt;
    return 0;
}
```

tarjan O(m+n)

原理：*//tarjan 模板（缩点）* –》dag

//原理:dfs树包含若干个scc，通过找到起始节点(不唯一)将其分开

//一次dfs并入栈 当找到某个scc起始节点即找到整个scc出栈即可;

//起始节点u求法： u的所有子节点v 的最早祖先都不能先于u low(u)<=dfn(n)(=成立)->low(u)==dfn(u)

//缩点：直接通过原点的scc 编号index建新图

链式前向星再写一遍

```
#include <bits/stdc++.h>
//define int long long
using namespace std;
const int N=1e5+5;
const int M=1e6+5;
int dfn[N],low[N],head[N],id[N];//low 孩子最早祖先  dfn 时间戳
int tot,scc_cnt,dfs_clock;
stack<int>S;

struct Edge{
    int from,to,next,w;
}edge[M];
void add(int u,int v,int w){
    edge[++tot].to=v;
    edge[tot].from=u;//可省
    edge[tot].w=w;
    edge[tot].next=head[u];
    head[u]=tot;
}

void tarjan(int u){
    dfn[u]=low[u]=++dfs_clock;//时间戳
    S.push(u);
    for(int i=head[u];~i;i=edge[i].next){//递归遍历子节点
        int v=edge[i].to;
        if(!dfn[v]){//未访问
          tarjan(v);
```

```
            low[u]=min(low[u],low[v]);//更新low[u]
        }else if(!id[v]){//已访问 未生成scc
            low[u]=min(low[u],dfn[v]);//反向边更新
        }
    }

    if(low[u]==dfn[u]){//存在scc
        scc_cnt++;
        int x;
        do{
            x=S.top();S.pop();
            id[x]=scc_cnt;// 编号
        }while(x!=u);
    }
}
void init(){
scc_cnt=dfs_clock=tot=0;//
memset(head,-1,sizeof head);
memset(id,0,sizeof id);
memset(dfn,0,sizeof dfn);
}
int m,n;
signed main(){
    while(cin>>m>>n){
        init();
        for(int i=1;i<=m;i++){
            int a,b;
            cin>>a>>b;
            add(a,b,0);
        }

        for(int i=1;i<=n;i++){
            if(!dfn[i])tarjan(i);
        }
    }

    return 0;
}
```

2-sat

一元限制

xi = true: xi’ ->xi

xi =false xi->xi’

二元限制

1. u->v ：u则v

   命题 和 逆否命题都要连**　u->v　v'->u'

2. u or v =true　u' ->v　v'->u　(一个0另一个必为1)

3. u or v =false　u->u'　v->v' (两个都为0)

4. u and v =false　u->v'　v->u'　(不能同时为1)

5. u and v = true　u' ->u　　v'->v　　　(全1)

6. u==v　u->v　u'->v'　v->u　v'->u'　都要连

7. 7.u!=v　u->v'　u'->v　v->u'　v'->u

   判断有解：若 存在i 和 i' 在同一强连通分量 则 no,否则yes

求可行解：（对称）（成对出现）

若一个连通分量（a,b,c）则有另一个（a',b',c'）要选都选

tarjan 缩点　按拓扑排序的反序输出　（每对选拓扑序靠后的（被指向为true））u'->u

即对于 i　i'　如果id[i]>id[i']　取 i

例:P4782

```
#include <bits/stdc++.h>
//#define LOCAL
//define int long long
using namespace std;
const int N=2000000;
const int M=10000005;
int dfn[N],low[N],head[N],id[N];//low 孩子最早祖先  dfn 时间戳
int tot,scc_cnt,dfs_clock;
stack<int>S;
struct Edge{
    int to,next;
}edge[M];
void add(int u,int v,int w){
    edge[++tot].to=v;
    edge[tot].next=head[u];
    head[u]=tot;
}

void tarjan(int u){
    dfn[u]=low[u]=++dfs_clock;//时间戳
    S.push(u);
    for(int i=head[u];~i;i=edge[i].next){//递归遍历子节点
        int v=edge[i].to;
        if(!dfn[v]){//未访问
          tarjan(v);
         low[u]=min(low[u],low[v]);//更新low[u]
        }
```

```
        else if(!id[v]){//已访问 未生成scc
        low[u]=min(low[u],dfn[v]);//更新
        }
    }

    if(low[u]==dfn[u]){//存在scc
        scc_cnt++;
        int x;
        do{
            x=S.top();S.pop();
            id[x]=scc_cnt;// 编号
        }while(x!=u);
    }
}
int m,n;
signed main(){
#ifdef LOCAL
    freopen("data.in","r",stdin);
    freopen("data.out","w",stdout);
#endif
tot=scc_cnt=0;
memset(edge,0,sizeof edge);
memset(head,-1,sizeof head);
cin>>n>>m;
for(int k=1;k<=m;k++){
    int i,a,j,b;
    cin>>i>>a>>j>>b;//或
        i --, j -- ;
        add(2 * i + !a, 2 * j + b);
        add(2 * j + !b, 2 * i + a);

}
for(int i=0;i<2*n;i++){
    if(!dfn[i])tarjan(i);
}



    for (int i = 0; i < n * 2; i ++ )
        if (!dfn[i])
            tarjan(i);

    for (int i = 0; i < n; i ++ )
        if (id[i * 2] == id[i * 2 + 1])
        {
            puts("IMPOSSIBLE");
            return 0;
        }

    puts("POSSIBLE");
    for (int i = 0; i < n; i ++ )
        if (id[i * 2] < id[i * 2 + 1]) printf("0 ");
```

```
        else printf("1 ");

    return 0;
}
```

网络流

2.Dinic

```
#include <bits/stdc++.h>
//#define LOCAL
using namespace std;
#define int long long
// dinic
int T;
const int inf = 0x3f3f3f3f;
const int MAXN = 20005;   // DIAN
const int MAXM = 500005;  // BIAN
int n, m, s, t, u, v;
int tot;                 //边序号从2开始  (0~1),  2~3  4~5
int w, ans, dep[MAXN];   //点的层深度
int head[MAXN];          //链式前向星  点i对应起始边
int cur[MAXN];           //当前弧优化
//对于一个节点xx，当它在DFS中走到了第ii条弧时，前i-1条弧到汇点的流一定已经被流满而没有可行的路线了
//那么当下一次再访问xx节点时，前i-1条弧就没有任何意义了
//所以我们可以在每次枚举节点x所连的弧时，改变枚举的起点，这样就可以删除起点以前的所有弧，来达到优化剪枝的效果
//对应到代码中，就是cur数组
struct EDGE {
  int to, next;
  int flow;          //剩余流量
} edge[MAXM << 1];   //反向边双倍

void add(int u, int v, int w) {  //同时加正反两条边
  edge[++tot].to = v;
  edge[tot].flow = w;
  edge[tot].next = head[u];
  head[u] = tot;

  edge[++tot].to = u;
  edge[tot].flow = 0;   //反向弧初始化为0
  edge[tot].next = head[v];
  head[v] = tot;
}

int bfs(int s, int t) {  // bfs  在残量网络中构造分层图
  for (int i = 0; i <= n; i++)
    dep[i] = inf, cur[i] = head[i];  // dep初始化无穷 同时起标记作用
  queue<int> que;  //  cur 初始化为head   用于之后弧优化
```

```
  que.push(s);
  dep[s] = 0;
  while (!que.empty()) {
    int u = que.front();
    que.pop();
    for (int i = head[u]; ~i; i = edge[i].next) {  //遍历边
      if (edge[i].flow == 0) continue;             //无剩余流量跳过
      int v = edge[i].to;
      if (edge[i].flow>0 && dep[v] == inf) {  // 有残余流量且为访问
        que.push(v);
        dep[v] = dep[u] + 1;   //记录深度
        if (v == t) return 1;  //跳出  找到一条增广路
      }
    }
  }
  return 0;
}
// 在层次图基础上不断dfs 求得增广路 (多条)
int dfs(int u, int sum) {  // sum表示当前流入可该点的剩余流量
  if (u == t) return sum;
  int k, res = 0;  // k为当前最小剩余容量
  for (int i = cur[u]; (~i )&& sum; i = edge[i].next) {  // 弧优化  sum
    cur[u] = i; //当前弧优化                                        //弧优化
    int v = edge[i].to;
    if (edge[i].flow > 0 && (dep[v] == dep[u] + 1)) {  //有剩余流量  层数差1
      k = dfs(v, min(sum, edge[i].flow));              //  u流入v的流量
      if (k == 0) dep[v] = inf;  //*剪枝，  去掉增广完的点
      edge[i].flow -= k;
      edge[i ^ 1].flow += k;
      res += k;  // res表示经过该点的所有流量和（相当于流出的总量）
      sum -= k;  // sum表示经过该点的剩余流量 (有多个v)
    }
  }
  return res;
}
signed main() {
  memset(head, -1, sizeof head);
  memset(edge, 0, sizeof edge);
tot=1;
ans=0;
cin>>n>>m>>s>>t;
for(int i=1;i<=m;i++){
  cin>>u>>v>>w;
  add(u,v,w);
}


while(bfs(s,t)){
  ans+=dfs(s,inf);
}
cout<<ans<<endl;
  return 0;
```

```
}
```

关于二分图

匈牙利（点权都为1）

二分图最大匹配数=二分图最小顶点覆盖=（n-最大独立集数）=（n-最小路径覆盖数）

最小割一定是简单割（有限，不包含无穷）

网络流（点权任意非负）

二分图最大流=最小割->二分图最小权点覆盖集（点权->流量） =总权值-最大权独立集

树链剖分(轻重链)

应用场景：LCA 以及各种关于树上路径、子树的操作(结合线段树)， 将一条路径上的链分成连续部分 更新 查询操作

核心操作：

每条重链除top 都是重儿子 轻儿子是重链的头

轻边 -连轻儿子 重边—连重儿子

性质 每次不超过O(log)

-----size [ ] dep[ ] fa[ ] ,son[ ] (重儿子) ,top [ ] 的定义

---- 两次dfs

第一次： 算出size[x] dep[x] fa[x] 和重儿子son[x]

第二次： 算出top[x]， x和x的重儿子的top相同（同一条重链）O(n)

lca 相关操作 每次不超过O(log)

```
//lca模板
#include <bits/stdc++.h>
#define LOCAL
//#define int long long
using namespace std;
const int N=1e5+5;
int T,n,tot,head[N];
int dep[N],fa[N],size[N];//子树大小
int son[N];  //重儿子
int top[N] ;  // 重链顶端
struct Edge{int from;int to;int w;int next;}edge[2*N];
void add(int u,int v,int w){
    edge[++tot].from=u;
    edge[tot].to=v;
    edge[tot].w=w;
    edge[tot].next=head[u];
```

```
    head[u]=tot;
}
void dfs1(int u,int father){
    size[u]=1;;//初始大小为1
    dep[u]=dep[father]+1;  // 深度
    son[u]=0; fa[u]=father;
    for(int i=head[u];~i;i=edge[i].next){
        int t=edge[i].to;
        if(t==father)continue; //  向上连的边跳过
        dfs1(t,u);   //  递归搜索子树
        size[u]+=size[t];   //更新size
        if(size[son[u]]<size[t])   son[u]=t; // 寻找更新重儿子
    }
}
void dfs2(int u,int top_u){  // 更新top
    top[u]=top_u;
    if(son[u]!=0)dfs2(son[u],top_u);// 存在子树  向下更新
    for(int i=head[u];~i;i=edge[i].next){
        int t=edge[i].to;
        if(t!=fa[u]&&t!=son[u])   //  轻儿子
        dfs2(t,t);           // 开启新的重链
    }
}
//求x,y 的lca  找x,y的lca的重链
//如果 x,y 在同一条重链 那么LCA 就是深度小的点
//  否则将top深度大的点往上跳到top的父亲  这一步跳过一条轻边 到另一条重链
int lca(int x,int y){
        while(top[x]!=top[y]){//不在同一条重链
            if(dep[top[x]]<dep[top[y]])swap(x,y);  ///让x深度大
         x=fa[top[x]];   //x 跳到top的父亲 跳过一条轻边
        }
        //  跳出后    在一条重链
        return dep[x]<dep[y] ? x:y;   // lca为深度小的点
}
signed main(){
    return 0;
}
```

差分约束

d[x]-d[y]<=dis(x,y) 两边之差小于第三边

对于x - y <=c

  c-> dis

求a-b最大值   统一为dx-dy<=c   最短路小于最短的

(x-y<c -> x-y<=c-1)

a-b最小值 统一为dx-dy>=c   最长路

理解：最大值小于任意dis(x,y),故小于最短路

最小值大于任意dis(x,y),故大于最长路

无解->负环

bell ford/spfa

x-y=c -> x-y>=c &&x-y<=c

①：对于差分不等式，a - b <= c ，建一条 b 到 a的权值为 c 的边，求的是最短路，得到的是最大值
②：对于不等式 a - b >= c ，建一条 b 到 a 的权值为 c 的边，求的是最长路，得到的是最小值
③：存在负环的话是无解

```
#include <bits/stdc++.h>
#define LOCAL
//#define int long long
using namespace std;
//#define int long long
// SAS u v 表示 v开始后 u 才能开始；f(u)>=f(v);
// SAF u v 表示 v结束后 u 才能开始；f(u)>=f(v)+a[v]
// FAF u v 表示 v结束后 u 才能结束；f(u)+a[u]>=f(v)+a[v];
// FAS u v 表示 v开始后 u 才能结束；f(u)+a[u]>=f(v);
// f[1]=0;                    求 f[i]  即 f[i]-f[1]
const int N = 1e5 + 5;
const int M = 1e6 + 5;
const int inf = 0x3f3f3f3f;
int n;                          // vis标记入队
int head[N], vis[N], dis[N], tot, neg[N];  //判断负环  每个点松弛次数不超过n-1
struct node {
  int from, to, next, w;
} edge[M];
void add(int u, int v, int w) {
  edge[++tot].from = u;
  edge[tot].to = v;
  edge[tot].w = w;
  edge[tot].next = head[u];
  head[u] = tot;
}
void init() {
  tot = 0;
  memset(edge, 0, sizeof edge);
  memset(head, -1, sizeof head);
  memset(vis, 0, sizeof vis);
  memset(neg, 0, sizeof neg);
}
int spfa(int s) {  //最长路   正环
  for (int i = 1; i <= n; i++) dis[i] = -inf;
  dis[s] = 0;
  queue<int> que;
  que.push(s);
  vis[s] = 1;
  while (!que.empty()) {
    int u = que.front();
    que.pop();
```

```
    vis[u] = 0;
    for (int i = head[u]; ~i; i = edge[i].next) {
      int v = edge[i].to;
      int dist = edge[i].w;
      if (dis[v] < dis[u] + dist) {  //松弛操作
        dis[v] = dis[u] + dist;
        if (!vis[v]) {  //只要不在队中就入队      可能多次
          vis[v] = 1;
          que.push(v);
          neg[v]++;
          if (neg[v] > n) return 0;  //判断负环；
        }
      }
    }
  }
  return 1;
}
signed main() {
  int a[N];  //持续时间
  int Case =0;
  while (cin >> n, n) {
     cout<<"Case "<<++Case<<":"<<endl;
         init();
    for (int i = 1; i <= n; i++) {
      cin >> a[i];
     add(0,i,0);
    }
    string ss;

    while (cin >> ss && ss != "#") {
      int u, v;
      cin >> u >> v;
      if (ss == "SAS") add(v, u, 0);// u,v 顺序
      if (ss == "SAF") add(v, u, a[v]);
      if (ss == "FAF") add(v, u, a[v] - a[u]);
      if (ss == "FAS") add(v, u, -a[u]);
    }
    int ans=spfa(0);
    if(ans){
      for(int i=1;i<=n;i++){
        cout<<i<<" "<<dis[i]<<endl;
      }
      cout<<endl;
    }
    else {cout<<"impossible"<<endl;
  cout<<"\n";}
  }
    return 0;
  }
```