图论

dijkstra堆优化

```
#include<iostream>
#include<cstdio>
#include<queue>
#include<cstring>
#include<algorithm>
using namespace std;
int n,m,s,g[10020],d[10020],vis[10020];
struct Edge{
        int u,v,w,next;
}e[500020];
struct ds{
        int d,u;
        bool operator<(const ds &x)const{return d>x.d;}
};
void dij(int s){
        memset(vis,0,sizeof(vis));
        priority_queue<ds>q;
        for(int i=1;i<=n;i++) d[i]=500000000;
        d[s]=0;
        q.push((ds){0,s});
        while(!q.empty()){
                ds x=q.top();q.pop();
                int u=x.u;
                if(vis[u]) continue;
                vis[u]=1;
                for(int i=g[u];i>0;i=e[i].next){
                        int v=e[i].v,w=e[i].w;
                        if(d[u]+w<d[v]){
                                d[v]=d[u]+w;
                                q.push((ds){d[v],v});
                        }
                }
        }


}
int main(){
        scanf("%d%d%d",&n,&m,&s);
        int x,y,z;
        for(int i=1;i<=m;i++){
                scanf("%d%d%d",&x,&y,&z);
                e[i]=(Edge){x,y,z,g[x]};g[x]=i;
        }
        dij(s);
        for(int i=1;i<=n;i++){
                if(d[i]!=500000000) printf("%d ",d[i]);
                else{
```

```
                        printf("2147483647 ");
                }
        }
        printf("\n");
        return 0;
}
```

最小生成树

```
#include<iostream>
#include<cstdio>
#include<algorithm>
using namespace std;
int n,m;
int f[5020];
struct Edge{
        int u,v,w;
}e[200020];
int cmp(Edge a,Edge b){
        return a.w<b.w;
}
int find(int x){
        return f[x]==0?x:f[x]=find(f[x]);
}
int main(){
        scanf("%d%d",&n,&m);
        int x,y,z;
        for(int i=1;i<=m;i++){
                scanf("%d%d%d",&x,&y,&z);
                e[i]=(Edge){x,y,z};
        }
        sort(e+1,e+1+m,cmp);
        int cnt=0,sum=0;
        for(int i=1;i<=m;i++){
                int x=find(e[i].u),y=find(e[i].v);
                if(x!=y){
                        f[x]=y;
                        cnt++;
                        sum+=e[i].w;
                }
                if(cnt==n-1){
                        printf("%d\n",sum);
                        return 0;
                }
        }
        printf("orz\n");
        return 0;
}
```

SPFA

```

```

```cpp
#include<iostream>
#include<cstdio>
using namespace std;
#include<cstring>
#include<algorithm>
#include<cmath>
#include<queue>
int n,m,s,t;
const int MAXN=10020;
const int MAXM=500020;
struct Edge{
    int u,v,w,next;
}e[MAXM];
int g[MAXN];
int inq[MAXN],d[MAXN];
void spfa(int s){
    queue<int>q;
    q.push(s);
    inq[s]=1;
    memset(d,0x3f,sizeof(d));
    d[s]=0;
    while(!q.empty()){
        int uu=q.front();inq[uu]=0;
        q.pop();
        for(int i=g[uu];i>0;i=e[i].next){
            int vv=e[i].v,ww=e[i].w;
            if(d[uu]+ww<d[vv]){
                d[vv]=d[uu]+ww;
                if(!inq[vv]){
                    q.push(vv);
                    inq[vv]=1;
                }
            }
        }
    }
    return;
}
int main(){
    scanf("%d%d%d",&n,&m,&s);
    for(int i=1;i<=m;i++){
        int xx,yy,ww;
        scanf("%d%d%d",&xx,&yy,&ww);
        e[i]=(Edge){xx,yy,ww,g[xx]};g[xx]=i;
    }
    spfa(s);
    for(int i=1;i<=n;i++){
        if(d[i]>1000000000) printf("2147483647 ");
        else{
            printf("%d ",d[i]);
        }
    }
    return 0;
}
```

LCA最近公共祖先（倍增）

```
//C++代码实现：
/*输入
12 5
1
1
1
2
2
5
5
6
6
10
10
3 11
7 12
4 8
9 12
8 10
输出
1
2
1
6
2
*/
#include<iostream>
#include<stdio.h>
#include<memory.h>
using namespace std;
#define max_size 1010
int d[max_size],p[max_size][10];
int head[max_size];
int cnt;
struct Edge{
    int v;
    int pre;
}eg[max_size];
//建树的函数
void add(int x,int y){
    eg[cnt].v=y;//儿子
    eg[cnt].pre=head[x];//head记录x连接的边在临界表中的编号……
    head[x]=cnt++;
}
//dfs()初始整颗树，算出d[1-n],p[1-n][j];
void dfs(int k){
    if (head[k]==0) return;
    int m,x,i,j;
    for(i=head[k];i!=0;i=eg[i].pre){
        x=eg[i].v;//找儿砸
        p[x][0]=k;//记录x的爸爸(祖先)
        m=k;
```

```
        d[x]=d[k]+1;//记录深度
        for(j=0;p[m][j]!=0;j++){
            p[x][j+1]=p[m][j];//利用公式p[x][j]=p[p[x][j-1]][j-1],这里的m就是p[x][j-
            m=p[m][j];//理解 2^3=2^2*2;
        }
    dfs(x);
    }
}
int find_lca(int x,int y){
    int m,k;
    if(x==y) return x;
    if(d[x]<d[y]){m=x;x=y;y=m;}//使x的深度>y
    m=d[x]-d[y];
    k=0;
    while(m){/*将x的深度调到和y的深度一样*///倍增法 k记录2^k次方
        if(m&1) x=p[x][k];
        m>>=1;//m=m>>1;
        k++;
    }
    if(x==y)return x;
    k=0;/*向上调节，找最近公共祖先，算法的核心，相当于一个二分查找。*/
    while(x!=y){
        if (p[x][k]!=p[y][k]||p[x][k]==p[y][k]&&k==0){
                /*如果p[x][k]还不相等，说明节点p[x][k]还在所求点的下面，
                所以继续向上调节;如果相等了，并且就是他们父节点，则那个节点一定就是所求点
                */
            x=p[x][k];
            y=p[y][k];
            k++;
        }
        else k--;
                /*如果p[x][k]=p[y][k],可以说明p[x][k]一定是x和y的共祖先，但不一定是最近
                所以向下找看还有没有更近的公共祖先.*/
    }
    return x;
}
int main(){
    int i,n,m,x,y;
    while(cin>>n>>m){
        memset(head,0,sizeof(head));
        memset(p,0,sizeof(p));
        memset(d,0,sizeof(d));
        cnt=1;
        for(i=2;i<=n;i++){
            scanf("%d",&x);
            add(x,i);
        }
        dfs(1);
        for(i=0;i<m;i++){
            scanf("%d%d",&x,&y);
            printf("%d\n",find_lca(x,y));
        }
    }
    return 0;
}
```

最大流

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
const int N=20,M=1010,inf=2147483647;
struct edge
{
        int v,w,next;
};
edge e[M*2];
int h[N],n,m,x,y,z,top,T;
bool vis[N];
void add_edge(int xx,int yy,int zz)
{
        top++;
        e[top]=(edge){yy,zz,h[xx]};
        h[xx]=top;
        top++;
        e[top]=(edge){xx,0,h[yy]};
        h[yy]=top;
        return ;
}
int dfs(int now,int f)
{
        if(now==n) return f;
        vis[now]=true;
        for(int i=h[now];i>0;i=e[i].next)
                if(!vis[e[i].v] && e[i].w>0)
                {
                        int d=dfs(e[i].v,min(f,e[i].w));
                        if(d>0)
                        {
                                e[i].w-=d;
                                e[((i-1)^1)+1].w+=d;
                                return d;
                        }
                }
        return 0;
}
int max_flow()
{
        int flow,f;
        flow=0;
        for(;;)
        {
                memset(vis,false,sizeof(vis));
                f=dfs(1,inf);
                if(f==0) return flow;
                flow+=f;
```

```
        }
        return 0;
}
int main()
{
        scanf("%d",&T);
        for(int k=1;k<=T;k++)
        {
                memset(h,0,sizeof(h));
                top=0;
                scanf("%d%d",&n,&m);
                for(int i=1;i<=m;i++)
                {
                        scanf("%d%d%d",&x,&y,&z);
                        add_edge(x,y,z);
                }
                printf("Case %d: %d\n",k,max_flow());
        }
        return 0;
}
```

dinic当前弧优化

```
 #include<iostream>
#include<algorithm>
#include<cstring>
#include<cstdio>
#include<cstdlib>
#include<cmath>
#include<map>
#include<vector>
#include<queue>
#include<set>
#define inf 2147483647
using namespace std;
typedef pair<int,int> pii;
const int N=210,M=210;
struct edge
{
        int v,w,next;
};
edge e[M*2];
int n,m,x,y,z,h[N],cur[N],dep[N],top,st,ed,num;//num为节点数
void add_edge(int xx,int yy,int zz)
{
        top++;
        e[top]=(edge){yy,zz,h[xx]};
        h[xx]=top;
        top++;
        e[top]=(edge){xx,0,h[yy]};
        h[yy]=top;
        return ;
}
```

```
bool bfs()
{
	int head;
	queue<int> q;
	memset(dep,0,sizeof(dep));
	q.push(st);
	dep[st]=1;
	while(!q.empty())
	{
		head=q.front();
		q.pop();
		if(head==ed)
			return true;
		for(int i=h[head];i>0;i=e[i].next)
			if(dep[e[i].v]==0 && e[i].w>0)
			{
				q.push(e[i].v);
				dep[e[i].v]=dep[head]+1;
			}
	}
	return false;
}
int dfs(int now,int f)
{
	if(now==ed)
		return f;
	int ff=0,temp;
	for(int &i=cur[now];i>0;i=e[i].next)//?
	{
		if(e[i].w>0 && dep[e[i].v]==dep[now]+1)
		{
			temp=dfs(e[i].v,min(e[i].w,f-ff));
			e[i].w-=temp;
			e[((i-1)^1)+1].w+=temp;
			ff+=temp;
			if(f==ff) return f;
		}
	}
	return ff;
}
int max_flow()
{
	int flow=0,ff;
	while(bfs()){
		for(int i=1;i<=num;i++)
			cur[i]=h[i];
		ff=dfs(st,inf);
		if(ff==0)
			break;
		flow+=ff;
	}
	return flow;
}
int main()
{
```

```
        memset(h,0,sizeof(h));
        top=0;
        scanf("%d%d",&m,&n);
        num=n;
        for(int i=1;i<=m;i++)
        {
                scanf("%d%d%d",&x,&y,&z);
                add_edge(x,y,z);
        }
        printf("%d\n",max_flow());
        return 0;
}
```

匈牙利算法

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int edge[1010*2][1010*2],pei[1010*2];
bool vis[1010*2];
int n,m,ans,x,y,z;
bool findz(int x)
{
        int i;
        for (i=n+1;i<=m+n;i++)
                if(edge[x][i] && !vis[i])
                {
                        vis[i]=true;
                        if(!pei[i] || findz(pei[i]))
                        {
                                pei[i]=x;
                                return true;
                        }
                }
        return false;
}
void Hungary()
{
        ans=0;
        memset(pei,0,sizeof(pei));
        for (int i=1;i<=n;i++)
        {
                memset(vis,0,sizeof(vis));
                if (findz(i))
                        ans++;
        }
        return ;
}
int main()
{
//      freopen("testdata.in","r",stdin);
        cin>>n>>m>>z;
```

```
        memset(edge,0,sizeof(edge));
        for (int i=1;i<=z;i++)
        {
                cin>>x>>y;
                if(x>n||y>m)
                        continue;
                edge[x][y+n]=1;

                edge[y+n][x]=1;
        }
        Hungary();
        cout<<ans<<endl;
        return 0;
}
```