Dinic 算法

Dinic 算法 可用于求解网络最大流问题。

Dinic 算法 的过程是这样的：每次增广前，我们先用 BFS 来将图分层。设源点的层数为 0，那么一个点的层数便是它离源点的最近距离。

通过分层，我们可以干两件事情：

1. 如果不存在到汇点的增广路（即汇点的层数不存在），我们即可停止增广。
2. 确保我们找到的增广路是最短的。（原因见下文）

接下来是 DFS 找增广路的过程。

我们每次找增广路的时候，都只找比当前层数多 1 的点进行增广（这样就可以确保我们找到的增广路是最短的）。

Dinic 算法有两个优化：

1. **多路增广**：每次找到一条增广路的时候，如果残余流量没有用完怎么办呢？我们可以再找出一条增广路。这样就可以在一次 DFS 中找出多条增广路，大大提高了算法的效率。
2. **当前弧优化**：如果一条边已经被增广过，那么它就没有可能被增广第二次。那么，我们下一次进行增广的时候，就可以不必再走那些已经被增广过的边。

设点数为 n，边数为 m，那么 Dinic 算法的时间复杂度（在应用上面两个优化的前提下）是 $O(n^2m)$，在稀疏图上效率和 EK 算法相当，但在稠密图上效率要比 EK 算法高很多。

特别地，在求解二分图最大匹配问题时，可以证明 Dinic 算法的时间复杂度是 $O(m\sqrt n)$。

模板题：

P3376 【模板】网络最大流

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=205,M=1e4+5;
const ll INF=0x3f3f3f3f;
int tot=1,head[N],cur[N],to[M],nxt[M],level[N];
ll val[M];//剩余容量
bool vis[N];
void add(int u,int v,ll c){
    nxt[++tot]=head[u];
    head[u]=tot;
    to[tot]=v;
    val[tot]=c;
}
void addedge(int u,int v,ll c){
    add(u,v,c);
```

```
    add(v,u,0);
}
bool BFS(int s,int t){
    memset(vis,0,sizeof(vis));
    queue<int>que;
    que.push(s);
    level[s]=0;
    vis[s]=1;
    cur[s]=head[s];
    while(!que.empty()){
        int u=que.front();
        que.pop();
        for(int i=head[u];i;i=nxt[i]){
            int v=to[i];
            if(!vis[v]&&val[i]>0){
                vis[v]=1;
                level[v]=level[u]+1;
                cur[v]=head[v];
                que.push(v);
                if(v==t) return true;
            }
        }
    }
    return vis[t];
}
ll DFS(int u,ll ret,int s,int t){
    if(u==t||ret==0) return ret;//ret 表示 S 到这里最多能流入的流量
    ll Flow=0,f;//flow 表示经过该点的所有流量和（相当于流出的总量□□f 表示当前最小的剩余容量
    for(int i=cur[u];i&&ret;i=nxt[i]){
        cur[u]=i;//当前弧优化
        int v=to[i];
        if(level[u]+1==level[v]&&val[i]>0){
            f=DFS(v,min(ret,val[i]),s,t);
            if(f==0) vis[v]=0;
            val[i]-=f;
            val[i^1]+=f;
            Flow+=f;
            ret-=f;
        }
    }
    return Flow;//返回实际使用的流量
}
ll Dinic(int s,int t){
    ll Flow=0;
    while(BFS(s,t)){
        Flow+=DFS(s,INF,s,t);
    }
    return Flow;
}
```

```
int main(){
    int n,m,s,t;
    scanf("%d %d %d %d",&n,&m,&s,&t);
    for(int i=1;i<=m;i++){
        int u,v;
        ll c;
        scanf("%d %d %lld",&u,&v,&c);
        addedge(u,v,c);
    }
    printf("%lld",Dinic(s,t));
    return 0;
}
```

参考链接

OI Wiki