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;
#define maxn 205
#define INF 0x3f3f3f3f
typedef long long ll;

struct Edge {
  ll from, to, cap, flow;
  Edge(ll u, ll v, ll c, ll f) : from(u), to(v), cap(c), flow(f) {}
};

struct Dinic {
  ll n, m, s, t;
  vector<Edge> edges;
  vector<ll> G[maxn];
  ll d[maxn], cur[maxn];
```

```
  bool vis[maxn];

  void init(ll n) {
    for (ll i = 1; i <= n; i++) G[i].clear();
    edges.clear();
    memset(d,0,sizeof(d));
    memset(cur,0,sizeof(cur));
    memset(vis,0,sizeof(vis));
  }

  void AddEdge(ll from, ll to, ll cap) {
    edges.push_back(Edge(from, to, cap, 0));
    edges.push_back(Edge(to, from, 0, 0));
    m = edges.size();
    G[from].push_back(m - 2);
    G[to].push_back(m - 1);
  }

  bool BFS() {
    memset(vis, 0, sizeof(vis));
    queue<ll> Q;
    Q.push(s);
    d[s] = 0;
    vis[s] = 1;
    while (!Q.empty()) {
      ll x = Q.front();
      Q.pop();
      for (ll i = 0; i < G[x].size(); i++) {
        Edge& e = edges[G[x][i]];
        if (!vis[e.to] && e.cap > e.flow) {
          vis[e.to] = 1;
          d[e.to] = d[x] + 1;
          Q.push(e.to);
        }
      }
    }
    return vis[t];
  }

  ll DFS(ll x, ll a) {
    if (x == t || a == 0) return a;
    ll flow = 0, f;
    for (ll& i = cur[x]; i < G[x].size(); i++) {
      Edge& e = edges[G[x][i]];
      if (d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a, e.cap - e.flow))) >
0) {
        e.flow += f;
        edges[G[x][i] ^ 1].flow -= f;
        flow += f;
        a -= f;
```

```
        if (a == 0) break;
      }
    }
    return flow;
  }

  ll Maxflow(ll s, ll t) {
    this->s = s;
    this->t = t;
    ll flow = 0;
    while (BFS()) {
      memset(cur, 0, sizeof(cur));
      flow += DFS(s, INF);
    }
    return flow;
  }
}D;
int main(){
    ll n,m,s,t;
    scanf("%lld %lld %lld %lld",&n,&m,&s,&t);
    D.init(n);
    for(ll i=1;i<=m;i++){
        ll u,v,w;
        scanf("%lld %lld %lld",&u,&v,&w);
        D.AddEdge(u,v,w);
    }
    printf("%lld",D.Maxflow(s,t));
    return 0;
}
```

参考链接

OI Wiki

From:
https://wiki.cvbbacm.com/ - **CVBB ACM Team**

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:lgwza:dinic_%E7%AE%97%E6%B3%95&rev=1597307225

Last update: **2020/08/13 16:27**