链滴

一天攻克最大流算法

作者：Licoded

原文链接：https://ld246.com/article/1564484997196

来源网站：链滴

墙裂推荐这篇博客主要写的算法都正确，而且都有一些可取之处。

01：Edmonds-Karp算法

EK算法据说是最简单的。

但大家都是这么评价EK算法的：由于EK算法容易超时，所以这个在比赛中不怎么用。

可是我对最大流太懵逼了，还是从最简单、最辣鸡的开始吧!

稀里糊涂地模了一个板子，感觉就差不多了。

参考博客链接，理解不动就只能先放弃呗!

简单讲一下现在的理解：

1. bfs查找有无从源点s到汇点t的路，bfs的过程中记录路径在pre数组中。

bfs采用队列方式，需要另开一个vis数组避免重复遍历，然后只要到大汇点t就结束bfs返回true，若bf跑完仍然没有找到汇点t就返回false，无路可走说明已经得到最大流。

2.bfs失败说明已经找到最大流，结束循环。

若bfs返回true，就遍历两次得到的路径。

第一次找到路径上的最小流，为该路径上的最大流，加入到结果中。

第二次将用过的边(u,v)减去本次的最大流，并建立反向边(v,u)大小为本次最大流。

while循环结束时return 结果flow就OK。

思路简单，但理解的不是很彻底，勉强吧!

EK算法板子/poj1273 Drainage Ditches（排水沟AC代码）

这道题目非常的水，这应该是我第一次在poj上看到自己的代码运行时间为0ms

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <queue>
#define MAXN 205
#define INF 0x3f3f3f3f
using namespace std;

int  net[MAXN][MAXN];//remain network, initialized as original graph
int  pre[MAXN];

int N, M;

bool bfs(int s, int t)//find an augmented path from s to t
{
    int now;
    queue<int> que;
    memset(pre, -1, sizeof(pre));
    pre[s] = s;
```

```
    que.push(s);
    while(!que.empty())
    {
        now = que.front();  que.pop();
        for(int i = 1; i < N+1; i++)
        {
            if(net[now][i] && pre[i] == -1)
            {
                pre[i] = now;
                if(i == t)
                    return true;
                que.push(i);
            }//end if
        }//end for
    }//end while
    return false;
}

int EdmondsKarp(int s, int t)
{
#define v pre[u]
    int flow = 0;
    while(bfs(s, t))
    {
        int d = INF;
        for(int u = t; u != s; u = v)
        {
            d = min(d, net[v][u]);
        }
        for(int u = t; u != s; u = v)
        {
            net[v][u] -= d;
            net[u][v] += d;
        }
        flow += d;
    }
    return flow;
#undef v
}

int main()
{
    while(~scanf("%d%d", &M, &N))
    {
        int a, b, d;
        memset(net, 0, sizeof(net));
        for(int i = 1; i < M+1; i++)
        {
            scanf("%d%d%d", &a, &b, &d);
            net[a][b] += d;//pay attention to repetitive edges
        }
        printf("%d\n", EdmondsKarp(1, N));
    }
    return 0;
```

```
}
```

02：Dinic算法

Dinic算法在EK算法的基础上增加了分层图的概念。

就是计算一下每个点到源点s的dis（就是边权值为1时的距离呗）

在普通情况下， DINIC算法时间复杂度为O(V^2E)

在二分图中， DINIC算法时间复杂度为O(\sqrt VE)

这个现在还不懂。

HDU 3549 Flow ProblemDinicAC代码

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <queue>
#define MAXN 100100
#define INF 0x3f3f3f3f
using namespace std;

struct edge{//chain forward star
    int to;
    int next;
    int val;
};

edge Edge[MAXN];
int  head[MAXN];

int cnt;
void add_edge(int u, int v, int w)
{
    Edge[cnt].to = v;
    Edge[cnt].next = head[u];
    Edge[cnt].val = w;
    head[u] = cnt++;
}

int dis[MAXN];
bool bfs(int s, int t)//mark the no-weight distance to s (run once, move just 1)
{
    queue<int> que;
    memset(dis, -1, sizeof(dis));
    que.push(s);    dis[s] = 0;
    while(!que.empty())
    {
        int now = que.front(); que.pop();
        for(int i = head[now]; i != -1; i = Edge[i].next)//Traverse all edges starting from now
        {
            int v = Edge[i].to;
```

```
            if(Edge[i].val && dis[v] == -1)
            {
                dis[v] = dis[now] + 1;
                if(v == t)
                    return true;
                que.push(v);
            }
        }
    }
    return false;
}

int N, M;

int dfs(int now, int left)//left: eesidual/剩余 flow
{
    if(now == N)
        return left;
    int used = 0;
    for(int i = head[now]; i != -1; i = Edge[i].next)//Traverse all edges starting from s
    {
        int v = Edge[i].to;
        if(Edge[i].val && dis[v] == dis[now] + 1)
        {
            int d = dfs(v, min(left-used, Edge[i].val));
            Edge[i].val -= d;
            Edge[i^1].val += d;
            used += d;
            if(used == left)
                return used;
        }
    }
    return used;
}

int Dinic(int s, int t)
{
    int max_flow = 0;
    while(bfs(s, t))
        max_flow += dfs(s, INF);
    return max_flow;
}

int main()
{
    int Case;
    scanf("%d", &Case);
    int ct = 1;
    while (Case--)
    {
        memset(head, -1, sizeof(head));
        cnt = 0;
        scanf("%d%d", &N, &M);
        int a, b, d;
```

```
        for (int i = 1; i < M+1; i++)
        {
            scanf("%d%d%d", &a, &b, &d);
            add_edge(a, b, d);
            add_edge(b, a, 0);//construct the reverse edge
        }
        printf("Case %d: %d\n", ct, Dinic(1, N));
        ct++;
    }
    return 0;
}
```

用EK算法也写了一份也AC了.

感觉现在看的例题都太入门了。

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <queue>
#define MAXN 1010
#define INF 0x3f3f3f3f
using namespace std;

int  net[MAXN][MAXN];//remain network, initialized as original graph
int  pre[MAXN];

int N, M;

bool bfs(int s, int t)//find an augmented path from s to t
{
    int now;
    queue<int> que;
    memset(pre, -1, sizeof(pre));
    pre[s] = s;
    que.push(s);
    while(!que.empty())
    {
        now = que.front();  que.pop();
        for(int i = 1; i < N+1; i++)
        {
            if(net[now][i] && pre[i] == -1)
            {
                pre[i] = now;
                if(i == t)
                    return true;
                que.push(i);
            }//end if
        }//end for
    }//end while
    return false;
}

int EdmondsKarp(int s, int t)
```

```
{
#define v pre[u]
    int flow = 0;
    while(bfs(s, t))
    {
        int d = INF;
        for(int u = t; u != s; u = v)
        {
            d = min(d, net[v][u]);
        }
        for(int u = t; u != s; u = v)
        {
            net[v][u] -= d;
            net[u][v] += d;
        }
        flow += d;
    }
    return flow;
#undef v
}

int main()
{
    int Case;
    scanf("%d", &Case);
    int ct = 1;
    while (Case--)
    {
        memset(net, 0, sizeof(net));
        scanf("%d%d", &N, &M);
        int a, b, d;
        for (int i = 1; i < M+1; i++)
        {
            scanf("%d%d%d", &a, &b, &d);
            net[a][b] += d;//construct the reverse edge
        }
        printf("Case %d: %d\n", ct, EdmondsKarp(1, N));
        ct++;
    }
    return 0;
}
```

优化

多路增广

每次不是寻找一条增广路，而是在DFS中，只要可以就递归增广下去，实际上形成了一张增广网。

当前弧优化

对于每一个点，都记录上一次检查到哪一条边。因为我们每次增广一定是彻底增广（即这条已经被增过的边已经发挥出了它全部的潜力，不可能再被增广了），下一次就不必再检查它，而直接看第一个

被检查的边。

优化之后渐进时间复杂度没有改变，但是实际上能快不少。

实际写代码的时候要注意，head数组初始值为-1，存储时从0开始存储，这样在后面写反向弧的时候较方便，直接异或即可。

关于复制head的数组cur；目的是为了当前弧优化。已经增广的边就不需要再走了.

但是有自环的时候或者因为一些其他原因，cur数组可能会导致TLE，不信你试一下HDU 3549 Flow P~~oblem~~

上面纯属屁话，优化不会导致TLE，但我还不知道那段代码为什么TLE(可能时是数组不够大，但数组够大怎么就TLE了呢，~~理解不懂就别理解了~~)。**~~改了一下dfs的思想，成功就返回，下次再继续跑就行了~~**
另外auto用在取que.front()时可以加速几十秒，很好用，感觉很帅！

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <queue>
#define MAXN 2010
#define INF 1e9+7
using namespace std;

struct edge{//chain forward star
    int to;
    int next;
    int val;
};

edge Edge[MAXN];
int  head[MAXN];
int   cur[MAXN];

int cnt;
void add_edge(int u, int v, int w)
{
    Edge[++cnt].to = v;
    Edge[cnt].next = head[u];
    Edge[cnt].val = w;
    head[u] = cnt;
}

int N, M;
int dis[MAXN];
bool bfs(int s, int t)//mark the no-weight distance to s (run once, move just 1)
{
    int i;
    queue<int> que;
    memset(dis, 0, sizeof(dis));
    que.push(s);    dis[s] = 1;
    while(!que.empty())
    {
        auto now = que.front(); que.pop();//The auto here can speed up your code!!!!
        for(i = head[now]; i; i = Edge[i].next)//Traverse all edges starting from now
        {
```

```
            int v = Edge[i].to, c = Edge[i].val;
            if(c && !dis[v])
            {
                dis[v] = dis[now] + 1;
                que.push(v);
            }
        }
    }

    return dis[t];
}
int dfs(int now, int left)//left: eesidual/剩余 flow
{
    if(now == N || !left)
        return left;
    for(int &i = cur[now]; i; i = Edge[i].next)//Traverse all edges starting from s
    {
        int v = Edge[i].to, c = Edge[i].val;
        if(c && dis[v] == dis[now] + 1)
        {
            int d = dfs(v, min(left, c));
            if(d)
            {
                Edge[i].val -= d;
                Edge[i^1].val += d;
                return d;
            }
        }
    }
    return 0;
}
int Dinic(int s, int t)
{
    int max_flow = 0;
    while(bfs(s, t))
    {
        int val;
        for(int i = 1; i < N+1; i++)
            cur[i] = head[i];
        while((val = dfs(s, INF)))
            max_flow += val;
    }
    return max_flow;
}
int main()
{
    int Case;
    scanf("%d", &Case);
    int ct = 1;
    while (Case--)
    {
        memset(head, -1, sizeof(head));
        cnt = 1;
        scanf("%d%d", &N, &M);
```

```
        int a, b, d;
        for (int i = 1; i < M+1; i++)
        {
            scanf("%d%d%d", &a, &b, &d);
            add_edge(a, b, d);
            add_edge(b, a, 0);//construct the reverse edge
        }
        printf("Case %d: %d\n", ct, Dinic(1, N));
        ct++;
    }
    return 0;
}
```

推送-重贴标签算法

在基于增广路径或线性规划的最大流问题的解中，非常适合

Hopcroft-Karp二分匹配算法

目前最好的解决最大二分匹配问题的算法