图论进阶 + 网络流

李国鸿

天津大学

2023 年 8 月 2 日

目录

目录

连通性问题概述

分为有向图连通性和无向图连通性。

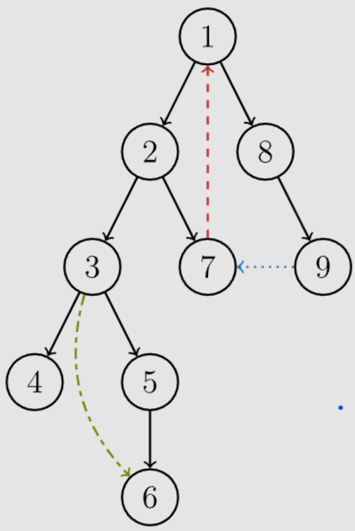
无向图

两点相连则同属一个连通块，并查集或搜索即可处理。
如果删去一个点以后，联通块数量增加，则称该点为割点。
如果删去一条边以后，联通块数量增加，则称该边为割边。

有向图

有向图如果没有环，则被称为有向无环图 (DAG)，可以进行拓扑排序，图论 dp 等。
对于一般的有向图，如果对于一个点集，点集中任意两点相互可达，则被称作是一个强连通分量 (SCC)。

DFS 生成树和 DFS 序



DFS 生成树和 DFS 序

有向图的 DFS 生成树主要有 4 种边（不一定全部出现）:
树边（tree edge）：示意图中以黑色边表示，每次搜索找到一个还没有访问过的结点的时候就形成了一条树边。
前向边（forward edge）：示意图中以绿色边表示，它是在搜索的时候遇到子树中的结点的时候形成的。
反祖边（back edge）：示意图中以红色边表示，也被叫做回边，即指向祖先结点的边。
横叉边（cross edge）：示意图中以蓝色边表示，它主要是在搜索的时候遇到了一个已经访问过的结点，但是这个结点并不是当前结点的祖先。
DFS 的同时，维护一个栈。对于结点 u，第一次访问到的时候，将它加入栈中；当它所属的强连通分量被确定时，将它从栈中退出。

Tarjan

变量

dfn[u]：深度优先搜索遍历时结点 u 被搜索的次序。
low[u]：在 u 的子树中能够回溯到的最早的已经在栈中的结点。
一个结点的子树内结点的 dfn 都大于该结点的 dfn。

过程

v 未被访问：继续对 v 进行深度搜索。在回溯过程中，用 low[v] 更新 low[u]。因为存在从 u 到 v 的直接路径，所以 v 能够回溯到的已经在栈中的结点，u 也一定能够回溯到。
v 被访问过，已经在栈中：根据 low 值的定义，用 dfn[v] 更新 low[u]。
v 被访问过，已不在栈中：说明 v 已搜索完毕，其所在连通分量已被处理，所以不用对其做操作。

Tarjan

```
void tarjan(int u)
{
    dfn[u]=low[u]=++tim;
    sta[++top]=u;ins[u]=1;
    for(int i=0;i<(int)e[u].size();++i)
    {
        int v=e[u][i];
        if(!dfn[v])
        {
            tarjan(v);
            low[u]=min(low[u],low[v]);
        }
        else if(ins[v])low[u]=min(low[u],dfn[v]);
    }

    if(dfn[u]==low[u])
    {
        ++cnt;
        do
        {
            bel[sta[top]]=cnt;
            ins[sta[top]]=0;
        }while(sta[top--]!=u);
    }
}
```

P3387 【模板】缩点

Problem

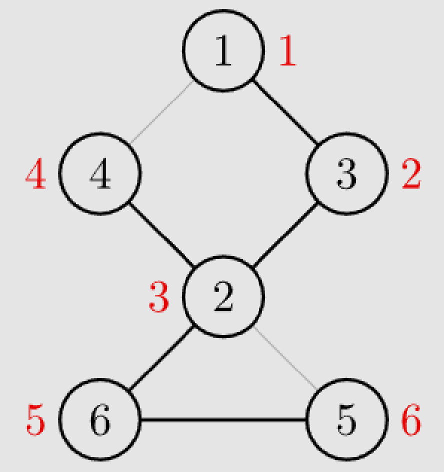
给定一个 n 个点 m 条边的有向图，每个点有一个非负权值，求一条路径，使路径经过的点权值之和最大。你只需要求出这个权值和。
允许多次经过一条边或者一个点，但是，重复经过的点，权值只计算一次。

P3387 【模板】缩点

Solution

注意到对于一个强连通分量，由于权值非负，一定是要获得它们的全部权值才最优，因此可以将一个强连通分量缩成一个点。
于是问题变成了 DAG 上求最大权值和的路径，这是图论 dp 的经典问题，记录 f[i] 表示以 i 结尾的路径中权值最大的，用拓扑排序来确定 dp 顺序。
值得注意的是，tarjan 算法树形遍历的特征，保证了缩点后结点的编号的逆序一定符合拓扑序，因此只要逆序枚举结点 dp 即可，不必确实地做一遍拓扑排序。

割点



割点

我们判断某个点是否是割点的根据是：对于某个顶点 u，如果存在至少一个顶点 v（u 的儿子），使得 low[v] >= dfn[u]，即不能回到祖先，那么 u 点为割点。
此根据惟独不适用于搜索的起始点，需要特殊考虑：若该点不是割点，则从起始点只「向下搜了一次」，即在搜索树内仅有一个子结点。如果在搜索树内有两个及以上的儿子，那么他一定是割点了。

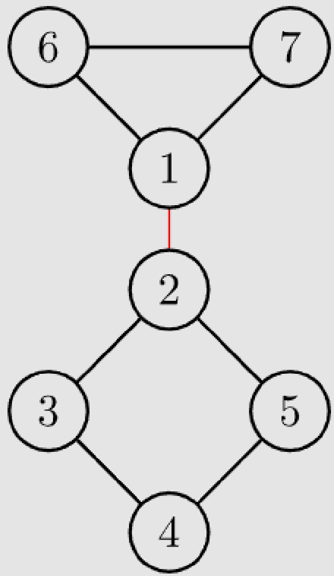
割点

```
void dfs(int u,int frm)
{
    dfn[u]=low[u]=++tim;

    int cnt=0;
    for(int i=0;i<(int)e[u].size();++i)
    {
        int v=e[u][i];
        if(!dfn[v])
        {
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            cnt+=(low[v]>=dfn[u]);
        }
        else if(v!=frm)low[u]=min(low[u],dfn[v]);
    }

    if(frm==0)
    {
        if(cnt>1)ans.push_back(u);
    }
    else
    {
        if(cnt>0)ans.push_back(u);
    }
}
```

割边/桥



割边/桥

和割点差不多，只要改一处：low[v]>dfn[u] 就可以了，而且不需要考虑根节点的问题。

```
void dfs(int u,int frm)
{
    dfn[u]=low[u]=++tim;
    for(int i=eh[u];i!=-1;i=e[i].nxt)if(i!=(frm^1))
    {
        int v=e[i].v,w=e[i].w;
        if(!dfn[v])
        {
            dfs(v,i);
            low[u]=min(low[u],low[v]);
            if(low[v]>dfn[u])ans=min(ans,w);
        }
        else low[u]=min(low[u],dfn[v]);
    }
}
```

2-sat 问题

有 n 个 bool 变量 x[i]，和 m 个需要满足的条件。
每个限制条件形如，如果 x[i]=a，那么 x[j]=b。
问是否存在一种方案，给每个 x[i] 赋值，使得所有条件都被满足。

2-sat 模型

对于每个变量，拆出两个点，分别对应 x[i]=0 和 x[i]=1，则这两个点应当恰好选其中一个。
从 x[i]=a 向 x[j]=b 建一条边，表示限制条件；此外还应该建一条边，从 x[j]=!b 向 x[i]=!a，表示原命题的逆否命题。你会发现这样建出来的图具有对称性。
那么，一种方案是可行的，当且仅当选点后，若一个点被选，则它能达到的所有点都被选。
跑 tarjan 算法求强连通分量，对于同属于一个强连通分量中的点，它们应当都被选，或者都不被选。
构造方案时，看 x[i]=0 和 x[i]=1 所属的强连通分量，谁所属的强连通分量编号小，也就是拓扑序大，我们就选谁。
而无解的情况，对应着 x[i]=0 和 x[i]=1 在同一个强连通分量内。

P4782 【模板】2-SAT 问题

Problem

n 个 bool 变量，限制条件形如 x[i]=a 或 x[j]=b，即应当这两个式子至少有一个为真。
判定有无解，并构造可行解。

P4782 【模板】2-SAT 问题

Solution

考虑将限制条件写为命题。如果 x[i]=!a，那么 x[j]=b。如果 x[j]=!b，那么 x[i]=a。这俩命题正好互为逆否。

目录

欧拉路径和欧拉回路

欧拉路径

欧拉路径：通过图中每条边恰好一次的路径。
(起点与终点不同时的判据)
有向图：起点出度比入度多 1，终点入度比出度多 1，其余点出入度相等。
无向图：起点和终点是奇度数，其余点偶度数。

欧拉回路

欧拉回路：起点和终点相同的欧拉路径。
有向图：所有点出入度相同。
无向图：所有点偶度数。

P7771 【模板】欧拉路径

Problem

有向图判定有无欧拉路径，并构造输出字典序最小的解。

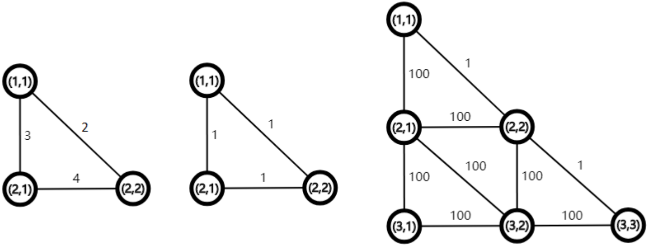
Solution

由于出入度相同，一个点只要能进就一定能出，暴力搜索是可行的。
但可能会存在一些环，不被包含在我们暴力搜索出的路径中，考虑如何将它们插入到路径中。
记录后序遍历的结果，再反过来，会发现这些环正好被插入到了路径中。

第 10 届山东省赛 J. Triangle City

Problem

给一张以固定方式生成的无向图：点编号 (x,y)，且一定有 $x \geq y$。
从 (x,y) 连边向 (x+1,y),(x,y+1),(x+1,y+1)（如果这些点存在）。
给定边权，求解从 (1,1) 到 (n,n) 的最长路，点可以重复经过，边只能走一次。



第 10 届山东省赛 J. Triangle City

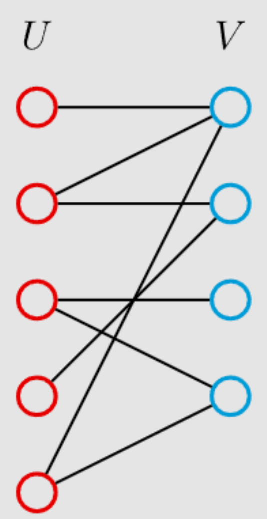
Solution

注意到这张图所有点都是偶度数的，一定存在欧拉回路遍历所有边。只要求解出从 (n,n) 到 (1,1) 的最短路，用总的边权和减去这个最短路，得到的就是从 (1,1) 到 (n,n) 的最长路。
题目要求构造方案，将最短路的边删掉后，对剩下的图构造从 (1,1) 到 (n,n) 的欧拉路径即可。

目录

二分图

二分图是什么？节点由两个集合组成，且两个集合内部没有边的图。换言之，存在一种方案，将节点划分成满足以上性质的两个集合。



二分图

二分图性质

二分图不存在长度为奇数的环。
因为每一条边都是从一个集合走到另一个集合，只有走偶数次才可能回到同一个集合。

二分图判定

如何判定一个图是不是二分图呢？
换言之，我们需要知道是否可以将图中的顶点分成两个满足条件的集合。
显然，直接枚举答案集合的话实在是太慢了，我们需要更高效的方法。
考虑上文提到的性质，我们可以使用 DFS 或者 BFS 来遍历这张图。
如果发现了奇环，那么就不是二分图，否则是。
简而言之就是进行二分图染色/黑白染色。

二分图最大匹配

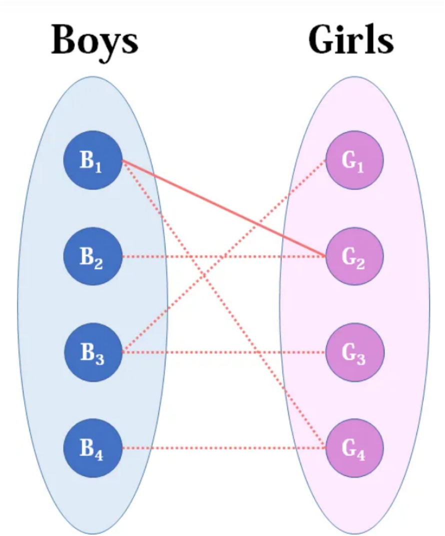
给定一个二分图 G，即分左右两部分，各部分之间的点没有边连接，要求选出一些边，使得这些边没有公共顶点，且边的数量最大。
考虑现在已有一个匹配，如何使它的匹配数增多？

匈牙利算法

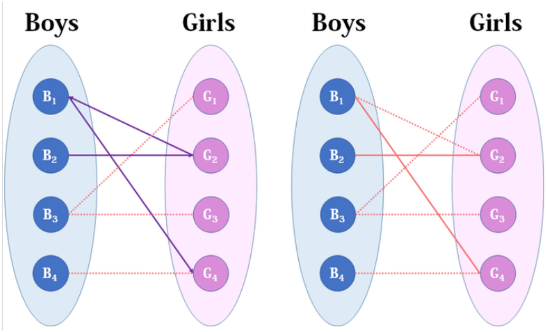
增广路

现在从一个非匹配点出发，寻找一条含奇数条边的路径，满足非匹配边-匹配边-非匹配边交替的性质，我们称这条路径为一条增广路
那么，将这条增广路上的所有匹配边变成非匹配边，非匹配边变成匹配边，我们就得到了一个新的合法匹配，且匹配数增加 1。
可以证明，对于一个二分图，枚举非匹配点，反复地寻找增广路，就可以得到它的最大匹配。
时间复杂度 $O(nm)$。

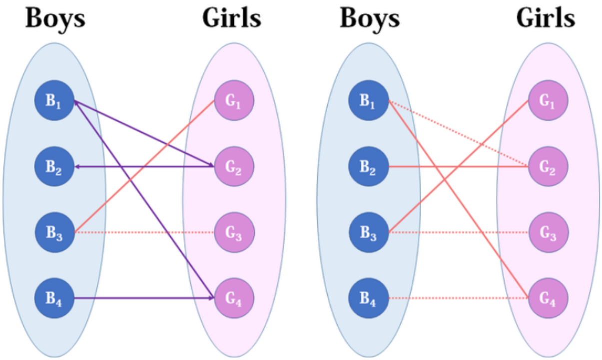
匈牙利算法



匈牙利算法


Boys
Girls
B_1
B_2
B_3
B_4
G_1
G_2
G_3
G_4
Boys
Girls
B_1
B_2
B_3
B_4
G_1
G_2
G_3
G_4

匈牙利算法


Boys
Girls
Boys
Girls
B1
B2
B3
B4
G1
G2
G3
G4
B1
B2
B3
B4
G1
G2
G3
G4

匈牙利算法

```
void aug(int v)
{
    while(v)
    {
        int t=px[pre[v]];
        px[pre[v]]=v;
        py[v]=pre[v];
        v=t;
    }
}
bool bfs(int s)
{
    memset(pre,0,sizeof(pre));//初始化路径
    memset(vx,0,sizeof(vx));//初始化vis
    memset(vy,0,sizeof(vy));

    while(!q.empty())q.pop();//清空队列
    q.push(s);

    int u;
    while(!q.empty())
    {
        u=q.front();q.pop();
        vx[u]=1;
        for(int v=1;v<=n[1];++v)if(e[u][v] && !vy[v])
        {
            vy[v]=1;
            pre[v]=u;//记录路径
            if(!py[v]){aug(v);return 1;}//找到增广路终点
            else q.push(py[v]);//v已有匹配，令u=match[v]
        }
    }
    return 0;//增广失败
}
```

二分图匹配的常见转化

二分图最小点覆盖（König 定理）

最小点覆盖：选最少的点，满足每条边至少有一个端点被选。
二分图中，最小点覆盖 = 最大匹配。

二分图最大独立集

最大独立集：选最多的点，满足两两之间没有边相连。
因为在最小点覆盖中，任意一条边都被至少选了一个顶点，所以对于其点集的补集，任意一条边都被至多选了一个顶点，所以不存在边连接两个点集中的点，且该点集最大。因此二分图中，最大独立集 = n - 最小点覆盖。

二分图匹配的常见转化

二分图博弈

二分图博弈是一类博弈模型，它可以被抽象为：给出一张二分图和起始点 H，A 和 B 轮流操作，每次只能选与上个被选择的点（第一回合则是点 H）相邻的点，且不能选择已选择过的点，无法选点的人输掉。一个经典的二分图博弈模型是在国际象棋棋盘上，双方轮流移动一个士兵，不能走已经走过的格子，问谁先无路可走。
考虑二分图的最大匹配，如果最大匹配一定包含 H，那么先手必胜，否则先手必败。
也就是说，最大匹配一定包含 H，等价于，先手必胜。

二分图匹配的常见转化

证明

最大匹配一定包含 H -> 先手必胜
如果后手走到一个有匹配的点，先手走到对应匹配点。
如果后手走到一个没有匹配的点，那么对路径上的所有边进行反转（匹配变非匹配，非匹配变匹配），将得到一个同样规模的最大匹配，而这个匹配不包含 H，矛盾。
先手必胜 -> 最大匹配一定包含 H
考虑证明逆否命题，即存在一个不包含 H 的最大匹配 -> 先手必败。
如果先手走到一个有匹配的点，后手走到对应匹配点。
如果先手走到一个没有匹配的点，那么对路径上的所有边进行反转（匹配变非匹配，非匹配变匹配），将得到一个更大的匹配，而这与原匹配是最大匹配相矛盾。

二分图最大权匹配

Problem

二分图的最大权匹配是指二分图中边权和最大的匹配。

Solution

考虑原图的一个子图，点集不变，但我们只考虑其中权值最大的一些边。
跑匈牙利算法，看对于这个子图是否能得到最大匹配。
如果不能，就再加入一条权值略小一些的边。
可以证明这种贪心策略的正确性。
实际算法中，我们不是真的这样暴力拉边，而是通过一种顶标的设计进行的。

KM 算法

定义顶标 lx[i],ly[i]。
对于任意一条边 (u,v)，都有性质 lx[u]+ly[v]>=e[u][v]。
当且仅当 lx[u]+ly[v]==e[u][v]，我们称该边为相等边。
所有点和相等边组成的子图，称为相等子图。
核心算法：贪心地将增广所需的边中，边权最大的那些边变成相等边，即逐渐扩大相等子图。
核心性质：扩大相等子图至其刚好有完美匹配时，该匹配即为原图的最大权完美匹配。
由此，我们便能将 km 算法简单地理解为：匈牙利算法 + 扩大相等子图。

KM 算法

顶标变化与松弛量

对于每轮增广，记录 slack[i] 表示松弛量。
slack[i]=min(lx[u]+ly[i]-e[u][i])
每当希望走一条不在相等子图上的边 (u,i)，就用上式去更新 slack[i]。
最后的实际顶标变化值取 d=min(slack[i])。

扩大相等子图

那么相等子图是如何被扩大的呢？
我们考虑如何让已在相等子图里的边仍在的情况下，拉进新边。

顶标变化的影响

Code

```
if(vx[i])lx[i]-=d;
if(vy[i])ly[i]+=d;else slack[i]-=d;
```

我们分类讨论这样修改顶标对边 (u,v) 的影响：
1. vy[v]=1，已经能搜到 v 这个右部点了，用不到 (u,v) 这条边，无所谓影响。
2. vx[u]=0，vy[v]=0，未遍历到的 => 不修改
3. vx[u]=1，vy[v]=0，该边非相等边 => 修改后可能为相等边 => 可能提供新增广路。
这里的序号 3 就是修改顶标拉入相等边，提供新的增广可能的秘密所在。

顶标变化的影响

需要注意的是我们虽然希望修改过后，原来在相等子图中的边不会被拉出，但这其实是不可能的。

因为我们要用 $2n$ 个顶标，去满足 n^2 条边对应的方程，这显然有可能无解。

但也不用去纠结这个问题，扩大相等子图这种说法，只是方便我们记忆算法的一种理解。

两个常见的 KM 注意事项

说白了就是两个坑点。
第一个是复杂度的，用 dfs 写的 km 复杂度是 $O(n^4)$ 的，bfs 才是 $O(n^3)$ 的，这个与 bfs 的状态延续策略相关。
第二个是关于无解的，这个要根据题意来决定，你要保证无论如何算法能正常终止。
首先，无论如何，要让左部点个数小于等于右部点个数，通过添加虚拟点。
比如题目说，如果无法找到一个匹配使得所有左部点都在其中，就输出无解。
那么我们就要给没有边的点对连接一条权值 -inf 的虚拟边，如果这条边被匹配上了就无解。
再比如题目说，允许某些左部点不匹配，只要已有匹配的权值和最大就好。
那么对于没有边的点对，要连的就是权值为 0 的虚拟边。

一般图最大匹配

比较冷门，没怎么见过这类题目。
一般图最大的问题在于一般图可能含有奇数环，无法进行二分图染色，所以匹配起来会很乱，你无法确定一个点究竟是左部点还是右部点。
常规做法是用带花树算法，将奇数环定义成一种叫做花的结构，然后考察花是如何增广的，花上的每个点都同时具有两种颜色，因此要让它们以这两种身份各自向花外求一遍增广路。更进一步可能会有花套花的情况，所以要用并查集维护这种关系。时间复杂度 $O(n^3)$。
还有一种很厉害的随机做法是这样的：我们考虑不做二分图染色，就用普通的匈牙利算法去做，然后也不考虑什么奇数环偶数环，干脆避开所有环，只找链，也就是绝不访问当次增广已经访问过的点。
这种做法很容易弄几个奇数环以后再精心构造一下边的顺序来卡掉，但如果我们把点和边的顺序随机打乱，再反复多做几次取最大值，就会有非常优秀的实际表现。
(现在 uoj79 又加入了新的 hack 数据，把随机做法卡掉了……)

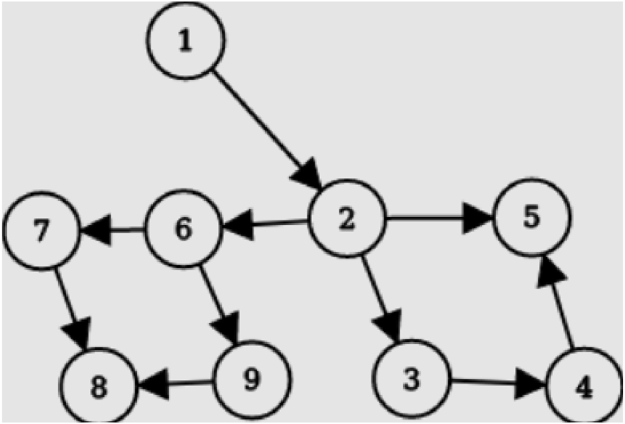
目录

什么是支配

我们在任意的一个有向图上钦定一个入口结点 s，对于结点 u，如果任意从 s 到 u 的路径都经过结点 v，则我们称 v 支配 u，记作 v dom u。对于从 s 无法到达的结点，讨论其支配关系是没有意义的，所以默认 s 能够到达图上的任何一个结点。
根据支配关系的定义，我们容易得到一种求解支配关系的暴力算法：考察删去 v 结点后，是否还有从 s 到达 u 的路径，我们便可得知是否有 v dom u。

什么是支配

例如这张有向图中，2 被 1 支配，3 被 1, 2 支配，4 被 1, 2, 3 支配，5 被 1, 2 支配， etc



支配树

我们证明，支配关系一定是树形的。
言外之意，有如下引理成立。
1. 如果 u dom v，v dom w，那么 u dom w。（传递性）
2. 如果 u dom v，v dom u，那么 u=v。（反身性）
3. 如果 u dom w，v dom w，那么有 u dom v 或 v dom u 成立。
u 在支配树上的祖先都能够支配 u，称 u 在支配树上的父亲是 u 的直接支配点。

DAG 上的支配树

一般的有向图求解支配树比较复杂，需要引入半支配点的概念辅助求解。
这里我们考虑求解 DAG 上的支配树，s 可到达所有点，因此认为是拓扑序最小的点。
我们按拓扑序从小到大依次加入结点，来维护这颗支配树，假设现在加入了新的结点 u，考察原图上能直接到达它的所有点 v，它们已经在支配树上确定了位置，那么 u 的直接支配点应当是所有 v 在支配树上的 lca。
一边维护支配树，一边维护 lca 的倍增数组即可，时间复杂度 $O(m \log n)$。
模板：P2597 [ZJOI2012] 灾难

牛客多校 3 Koraidon, Miraidon and DFS Shortest Path

Problem

给一个有向图，保证 1 号点可以到达所有点，从 1 号点开始 BFS 求解最短路。
现在从 1 号点 DFS 求解最短路，问求得的最短路数组是否会出错。

牛客多校 3 Koraidon, Miraidon and DFS Shortest Path

Solution

先建立 BFS 生成树，考虑不在生成树上的每条边。
同一层内连的边会导致出错。
当前层向下一层连的边不会导致出错。
当前层向上层连的边是否导致出错，比如 u->v，要看是否 v dom u。
如果 v dom u 成立，那么搜到 u 的时候 v 一定已经被搜过，dis[v] 不会被错误更新。
如果 v dom u 不成立，那么存在一条路径，先到 u，再到 v，这样 dis[v] 就会被错误更新。

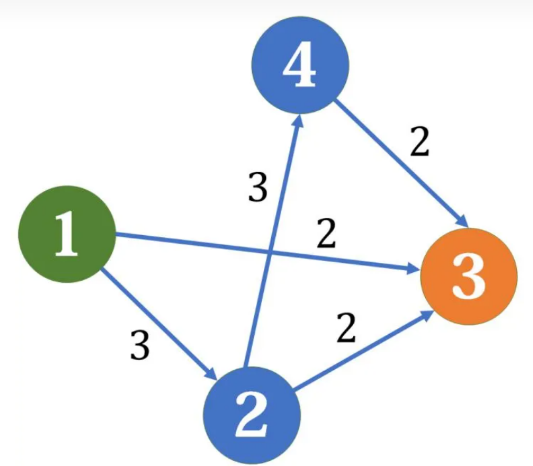
目录

基础概念

形式化的网络流定义用到流函数的概念，每条边的流受各种约束条件的限制，包括容量限制、斜对称性和流守恒性。且依此可以用线性规划的思路来理解，但这种讲述方式对初学者并不友好。
这里我们来用更具象地方式来讲述，在一张有向图中，我们认为每条边都是一根有向且有容量限制的水管，水可以从这条边的起点流向终点，且其中的流量不能超过它的容量。
在图中，我们定义一个源点和一个汇点，所有的水流从源点出发，汇集到汇点。那么最大流问题，就是在问在符合容量限制的前提下，源点到汇点的最大流量。
一个网络的最大流量是唯一的，但符合这个最大流量的流函数构造并不唯一。

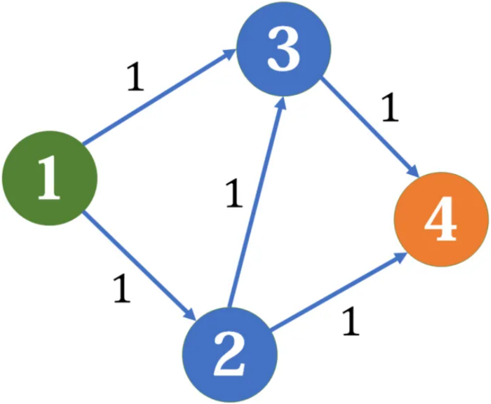
最大流

如图源点是 1，汇点是 3，最大流是 5。



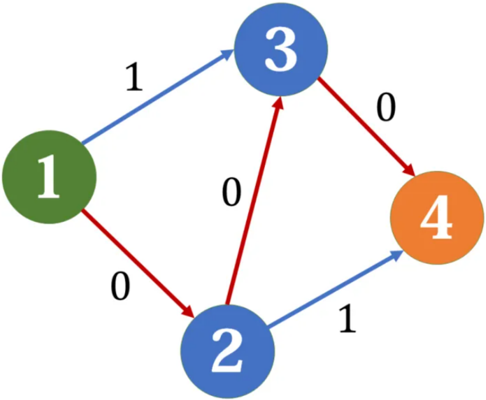
FF 算法

在不考虑时间复杂度的情况下，我们考虑如何暴力求解最大流。不难想到 DFS 搜索一条从源点到汇点的路径，而流量就是路径所有边剩余容量的最小值。获取这个流量之后，将路径上的所有边的容量减去这个容量。



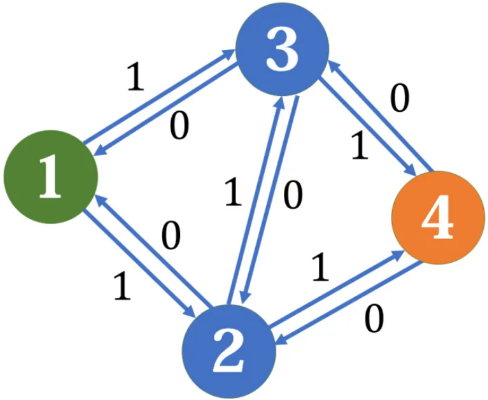
FF 算法

但这种方法是错误的，比如如果找到了这样的一条路径，求得的最大流就是 1。



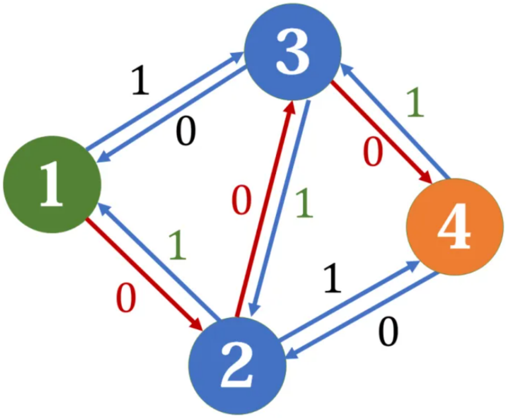
FF 算法

为了解决这个问题，我们引入反向边。在建边的同时，在反方向建一条容量为 0 的边：



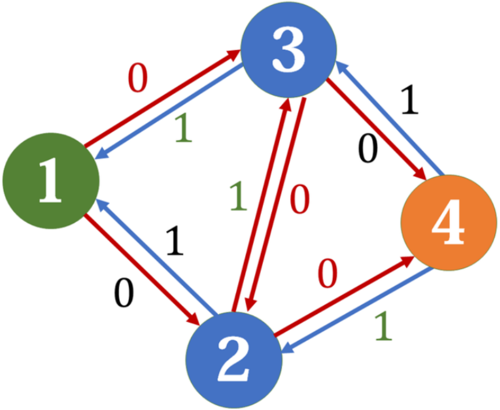
FF 算法

我们仍然选择 1->2->3->4，但在扣除正向边的容量时，反向边要加上等量的容量。



FF 算法

这时我们可以另外找到一条增广路：1->3->2->4。
注意看，现在我们同时选择了 2->3 和 3->2 两条边，我们可以认为，这两条边上的水流抵消了。所以实际上选择的路径就是 1->3->4 和 1->2->4。



FF 算法

其实可以把反向边理解成一种撤销，走反向边就意味着撤回上次流经正向边的若干流量，这也合理解释了为什么扣除正向边容量时要给反向边加上相应的容量：反向边的容量意味着可以撤回的量。

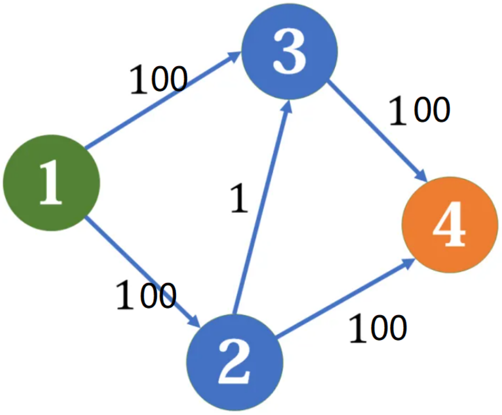
加入了反向边这种反悔机制后，我们就可以保证，当找不到增广路的时候，流到汇点的流量就是最大流。

算法的正确性保证了，那时间复杂度呢？

很可惜，这种做法的复杂度上界是 $O(mf)$ 的，m 是边数，f 是求得的最大流。

FF 算法

这个上界，可以通过像这样构造数据来卡满。



EK 算法

考虑将上述 DFS 寻找增广路的方式，换成 BFS。
BFS 并不像 DFS 那样短见，这种搜索方式的复杂度与最大流量无关，是 $O(nm^2)$ 的。
但我们考虑能不能更快，意识到这种暴力找增广路的方式每次最多只能找到一条增广路。
有没有什么办法可以实现多路增广，即一次找到的是多个增广路的总和？
BFS 的本质，是根据最短路去分层搜索，如果能分层灌注流量，是不是能够达到一个比较优秀的效率？
分层灌注又不方便记录实际流量，修改剩余容量，这种事情 DFS 做起来要方便的多。
如果把这两种思路结合起来……

Dinic 算法

考虑在增广之前，先用 BFS 求出源点到所有结点的距离（只能走有容量的边）。
根据不同的距离，对这个图进行分层。我们限制这一次的增广路，只能按层依次走，即当且仅当 dis[v]==dis[u]+1 时，我们才能从 u 向 v 用 DFS 寻求增广路。
在这样的过程中，我们的 DFS 要记录流入到当前点的流量，然后再把这些流量分给它的出边们，因此并不是只会找到一条增广路，而是多条增广路的一个总和。反复进行上述过程，即一次 BFS，然后一次 DFS，直到无法增广为止。
再加入当前弧优化，可以证明，这样的分层策略的时间复杂度是 $O(n^2m)$ 的。

当前弧优化

需要注意的是，如果对于一个点 u，有很多条入边，和很多条出边，它们都是 $O(m)$ 级别的，那么我们最多会 $O(m)$ 次遍历 u 点，每次都要遍历 u 所能出去的 $O(m)$ 级别的所有边。这个时间效率是无法接受的。为此我们引入当前弧优化，对于 u 点的所有出边，如果一条出边对应的增广方向已经被流满了，我们就将它从出边中去掉。
实际实现中，我们可以认为有一个指针，指向 u 点的所有出边中，尚且没有被流满的出边中最靠前的一个，记作 cur[u]，那么每次枚举出边就从它开始枚举。

Dinic 算法

```
ll ans=0;
while(bfs())
{
    memcpy(cur,eh,sizeof(cur));
    ans+=dfs(s,inf);
}
printf("%lld\n",ans);
```

Dinic 算法

```
bool bfs()
{
    memset(dis,-1,sizeof(dis));
    queue<int> q;

    dis[s]=0;
    q.push(s);

    while(!q.empty())
    {
        int u=q.front();q.pop();
        for(int i=eh[u];i!=-1;i=e[i].nxt)if(e[i].c>0)
        {
            int v=e[i].v;
            if(dis[v]==-1)
            {
                dis[v]=dis[u]+1;
                q.push(v);
            }
        }
    }
    return dis[t]!=-1;
}
```

Dinic 算法

```
int dfs(int u,int flow)
{
    if(u==t)
    {
        return flow;
    }

    int tmp=flow;
    for(int i=cur[u];i!=-1;i=e[i].nxt)
    {
        cur[u]=i;

        int v=e[i].v;
        if(e[i].c>0 && dis[v]==dis[u]+1)
        {
            int f=dfs(v,min(flow,e[i].c));

            flow-=f;
            e[i].c-=f;
            e[i^1].c+=f;

            if(flow==0)break;
        }
    }
    return tmp-flow;
}
```

二分图最大匹配

Problem

用最大流算法求二分图的最大匹配。

二分图最大匹配

Solution

从源点向所有左部点连边，所有右部点向汇点连边，所有能匹配的左部点向对应右部点连边。
上述所有边的容量都是 1。
对这张图跑 Dinic 求最大流，得到的最大流就是二分图的最大匹配。
可以证明，对于这种特殊的图，Dinic 的时间复杂度是 $O(n\sqrt{m})$ 的，优于匈牙利算法。

P3254 圆桌问题

Problem

有来自 m 个不同单位的代表参加一次国际会议。第 i 个单位派出了 r[i] 个代表。
会议的餐厅共有 n 张餐桌，第 i 张餐桌可容纳 c[i] 个代表就餐。
为了使代表们充分交流，希望从同一个单位来的代表不在同一个餐桌就餐。请给出一个满足要求的代表就餐方案。

P3254 圆桌问题

Solution

类似二分图最大匹配的建图思路。
任意单位可以向任意餐桌安排最多 1 个代表，从第 i 个单位向第 j 个餐桌连一条容量为 1 的边。
从源点向第 i 个单位连容量为 r[i] 的边。
从第 i 个餐桌向汇点连容量为 c[i] 的边。
跑最大流，看这个值是否等于代表人数之和。
构造方案时，就看第 i 个单位向第 j 个餐桌连的这条边是否被流过，如果是就做相应的安排。

ABC274G Security Camera 3

Problem

有 $H \times W$ 的棋盘，每个点是空地或者障碍物。
用最少数量的水平和竖直线段，将棋盘上所有空地覆盖。
不允许线段穿过障碍物，允许空地被重复覆盖。

ABC274G Security Camera 3

Solution

贪心地想，任何一条线段都应当延伸到最长，将所有这样可能的水平和竖直线段列出，并标号。
那么每个空地应恰好属于 1 条水平线段和 1 条竖直线段。
将水平线段视作左部点，竖直线段视作右部点，空地视作将它们对应相连的边。
问题转化为二分图的最小点覆盖。
有结论二分图最小点覆盖 = 最大匹配，用最大流求解。
点数和边数都是 $O(HW)$ 级别。

杭电多校 2 1012 Coin

Problem

有 n 个人，其中 k 个是小 F 的朋友。
初始时候，每个人手里都有 1 个硬币。
依次进行 m 次操作，对于第 i 次操作，给定两个人 A[i] 和 B[i]，执行其中一个操作：
1. A[i] 给 B[i] 一个硬币。
2. B[i] 给 A[i] 一个硬币。
3. 无事发生。
同时，对于第 i 个人，还会有一个手中硬币的数量限制 a[i]。
对于这 m 次操作，决定每次操作具体执行哪一个，以使得所有操作结束后，小 F 的这 k 个朋友手里的硬币之和最大。

杭电多校 2 1012 Coin

Solution

考虑如何处理时间顺序，对于每个人，我们将它拆成 m 个点，从上一层向下一层连容量 a[i] 的边，以保证任意时刻满足硬币数限制。
那么对于第 i 次操作，我们就找到 i 时刻对应的那一层，给 A[i] 和 B[i] 连容量 1 的边。
源点给 1 时刻的每个人连容量 1 表示给初始硬币。
m 时刻的 k 个朋友给汇点连容量 a[i] 表示收集朋友手中的硬币。
跑最大流即可，但这样的总点数是 $O(nm)$ 的。

杭电多校 2 1012 Coin

Solution

能否优化建图，发现第 i 次操作，我们会额外多建 n 个点，但其中实际的操作对象只有 2 个点。
我们只对这 2 个点额外建点，其余点在 i 时刻的点沿用它在第 i-1 时刻的点即可。
维护每个人最后一次建的点 hd[i]，这样下次再建点就从 hd[i] 向新点连容量 a[i] 即可。
总点数降低至 $O(n+m)$。

最小割问题

Problem

对于一个网络，我们定义割掉一条边的代价是这条边的容量。现在我们希望割掉若干条边，使得源点不能到达汇点。求最小代价。

最小割问题

Solution

直观理解，如果这个网络的流满了，那么一定是存在若干瓶颈边被流满了。
我们考虑割掉这些瓶颈边，这些瓶颈边的权值和就是最小割，同时也是最大流。
结论：一个网络的最小割，等于它的最大流。
值得注意的是，虽然我们上述关于最小割的问题是基于网络的。
但实际上泛化地从建模的角度讲，我们可以对于任意带权无向图，选定 S 和 T，求解割掉边权和最小的边集，使得 S 和 T 不连通。
求解这个问题，只需把每条无向边，变成两条容量等于边权的，方向相反的有向边，跑最大流。

P2057 [SHOI2007] 善意的投票

Problem

有 n 个物品，和两个集合 A、B。
每个物品应恰好被分到 A、B 两个集合中的一个。
对于第 i 个物品，没有分到 A 集合的代价是 a[i]，没有分到 B 集合的代价是 b[i]。
同时，有 m 对额外代价，第 i 对代价形如：如果 x[i] 和 y[i] 被分到不同集合，那么将产生 w[i] 的代价。
现在，你需要对每个物品，决定它被分入 A 集合还是 B 集合，并使得总代价最小。

P2057 [SHOI2007] 善意的投票

Solution

这是一个经典的二者选其一的最小割题目。
我们设置源点 S 和汇点 T 表示集合 A 和 B，第 i 个点由 S 连一条容量为 a[i] 的边、向 T 连一条容量为 b[i] 的边。对于限制条件 x,y,w，我们在 x,y 之间连容量为 w 的双向边。
注意到当源点和汇点不相连时，代表这些点都选择了其中一个集合。
如果将连向 S 或 T 的边割开，表示不放在 A 或 B 集合，如果把物品之间的边割开，表示这两个物品不放在同一个集合。
最小割就是最小花费。

P2762 太空飞行计划问题

Problem

最大权值闭合图，即给定一张有向图，每个点都有一个权值（可以为正或负或 0），你需要选择一个权值和最大的子图，使得子图中每个点的后继都在子图中。

P2762 太空飞行计划问题

Solution

建立源点 S 和汇点 T，若节点 u 权值为正，则 S 向 u 连一条有向边，边权即为该点点权；若节点 u 权值为负，则由 u 向 T 连一条有向边，边权即为该点点权的相反数。原图上所有边权改为 ∞。
答案为正权和减去最小割，而最小割又等于最大流。

牛客多校 2 B Link with Railway Company

Problem

n 座城市由 n-1 条双向道路联通，第 i 条道路有 c[i] 的维护费用。
有 m 条路线，第 i 条路线有起点 s[i]，终点 t[i]，利润 w[i]。
如果希望开通第 i 条路线并获取它的利润，那么从 s[i] 到 t[i] 路径上所有的道路都应当被维护，一个被多条路线需求的道路只需要支付一次维护费用。
求开通路线和维护道路的方案，使得收益最大。$n, m \leq 10^4$

牛客多校 2 B Link with Railway Company

Solution

是一个裸的最大权闭合子图，路线是正权，道路是负权，路线向所求需求的道路连边即可。
但这样的边数会达到 $O(mn)$ 级别，不可接受。
由于每条路线所需要连向的道路，在树上体现为一条连续的路径，考虑将它剖成 $O(\log n)$ 级别的重链和轻边，对应 $O(\log n)$ 个 dfs 序上的连续段，线段树优化建图，即可达到 $O(m\log^2 n)$ 级别的边数。

基础概念

在网络流模型中，我们引入了边的容量这个新概念。
而在传统图论模型中，我们对于每条边，会有一个边权的概念。
假设现在我们同时加入这两个概念，认为一条边的边权是在它上面流过单位流量所耗费的费用。
于是会引出一个自然的问题，由于我们知道满足最大流的流函数构造不唯一，那么在保证获得最大流的前提下，如何使我们耗费的费用最少。
这就是所谓的费用流问题，全称最小费用最大流。

SPFA+EK/Dinic

一种贪心思路是这样的：
我们从源点开始，寻找一条单位流量总费用最小的增广路，然后将这条增广路流满，重复上述过程。
反向边的退流机制肯定还得保留，如果正向流过单位流量的费用是 w，那么反向退回单位流量的费用应当设计为 -w，这样才能保证正流再退流以后对费用不产生影响。
那么我们总费用最小的增广路，是在一个边权可正可负的图上进行的，应当使用能够处理负环的 SPFA. 用 SPFA 确定路径以后，用 EK 进行单源增广。
也可以用 SPFA 对图进行分层，然后用 Dinic 进行多源增广，也就是在 Dinic 的过程中用 SPFA 代替 BFS，u 能访问 v 当且仅当 dis[v]==dis[u]+w。这种做法似乎还有一个名字叫 zkw 费用流。
可以证明，如果在初始网络中，不存在由剩余容量非 0 的边所组成的负权环，那么上述算法的正确性可以保证。

消圈算法和 SSP 的复杂度

如果在初始网络中，存在由剩余容量非 0 的边所组成的负权环，SPFA 算法将显然无法找到最短路径。
为此，我们需要消圈，也就是消去负权环。在不考虑源汇点的情况下，用 SPFA 找到一个负权环，然后用流量灌满它，即可将它消去。
值得注意的是，你会发现这里我们加入的一圈流量，是与源汇点无关的，但这其实也是符合流函数限制的，你会发现圈上每个点流入多少就流出多少，符合流量守恒。(实际上消圈算法很慢，做题时候还有一种科技是直接让负权边满流，然后用上下界网络流相关的思路保证流量守恒。)
关于费用流算法的复杂度，其最坏复杂度是 $O(nmf)$ 的，其中 f 是最大流量。
这是一个不可接受的复杂度，但实际上，在正解是费用流的题目当中，我们可以认为它的复杂度是多项式级别的。

二分图最大权匹配

Problem

用费用流算法求二分图的最大权匹配。

二分图最大权匹配

Solution

和用最大流求最大匹配的图基本长一个样子。
只不过左部点和右部点之间的边要给一个 -w 的费用，然后跑最小费用流。
选的 -w 之和最小，那么 w 之和就最大，由此得到最大权匹配。
比 km 算法慢，km 是 $O(nm)$，所以这个就不属于费用流正解的题目。

P2053 [SCOI2007] 修车

Problem

有 n 辆车和 m 个工人，第 i 个工人修第 j 辆车所需时间是 T[i][j]。
一个工人可以被分配多辆车，但必须按你所给定的顺序修，而不能同时修多辆。
安排这 n 辆车给这 m 个工人的分配和排序方案，使得每辆车车主的平均等待时间最短。

P2053 [SCOI2007] 修车

Solution

正向地思考这个问题，如果一个工人被分配 3 辆车，所需时间分别为 a,b,c。那么对应车主的等待时间分别是 a,a+b,a+b+c。
如果一个工人被分配 k 辆车，其中第 i 辆车所需时间为 w[i]，那么这辆车对总等待时间的贡献是 $(k-i+1)\times w[i]$，其中的 $(k-i+1)$ 表示它是修车队列中倒数第 $(k-i+1)$ 辆车。于是逆向思考，把第 x 个工人拆成 n 个点，第 i 个点表示倒数第 i 次修车的他。
那么如果他修第 y 辆车的时间是 T[x][y]，就连容量 1 费用 $T[x][y]\times i$ 的边。
S 向车连容量 1，工人拆成的每个点向 T 连容量 1。
跑最小费用最大流。

P2053 [SCOI2007] 修车

Solution

这题还有一个加强版本，P2050 [NOI2012] 美食节。
暴力拆点无法被接受，需要动态加点，一遍跑单源增广费用流，一遍加入需要的点。

目录

Thank you! Any Question?