C 妮妮与自来水厂

时间限制：1000ms 内存限制：65536kb

通过率：136/163 (83.44%) 正确率：136/786 (17.30%)

网络流模板题

题目描述

勤劳的妮妮入职了家边的自来水厂，入职后妮妮主要负责记录出水口的水流大小。

在水管分布图中，有 n 个阀门和 m 个单向管道，每个单向管道连接两个阀门，并且有各自的容量限制（即经过的水流不能超过该管道的最大容量）。妮妮想知道，在阀门 s 放水时，阀门 t 最大能接收到多大的水流。

形式化地说，给定 n 个点 m 条边的有向图，每条边有最大容量，求从点 s 到点 t 的最大流。

输入格式

第一行一个正整数 T（$1 \leq T \leq 10$），表示数据组数。

对于每组数据，第一行四个正整数 n, m, s, t（$1 \leq n \leq 100$，$1 \leq m \leq 5 \times 10^3$，$1 \leq s, t \leq n$），含义同题目描述。

接下来 m 行，每行三个正整数 u_i, v_i, w_i（$1 \leq u_i, v_i \leq n$，$0 \leq w_i < 2^{31}$），表示第 i 条有向边 $u_i \to v_i$ 的最大容量为 w_i。

图中有可能存在重边和自环。

输出格式

对于每组数据，输出一行一个非负整数，表示点 s 到点 t 的最大流。

Hint

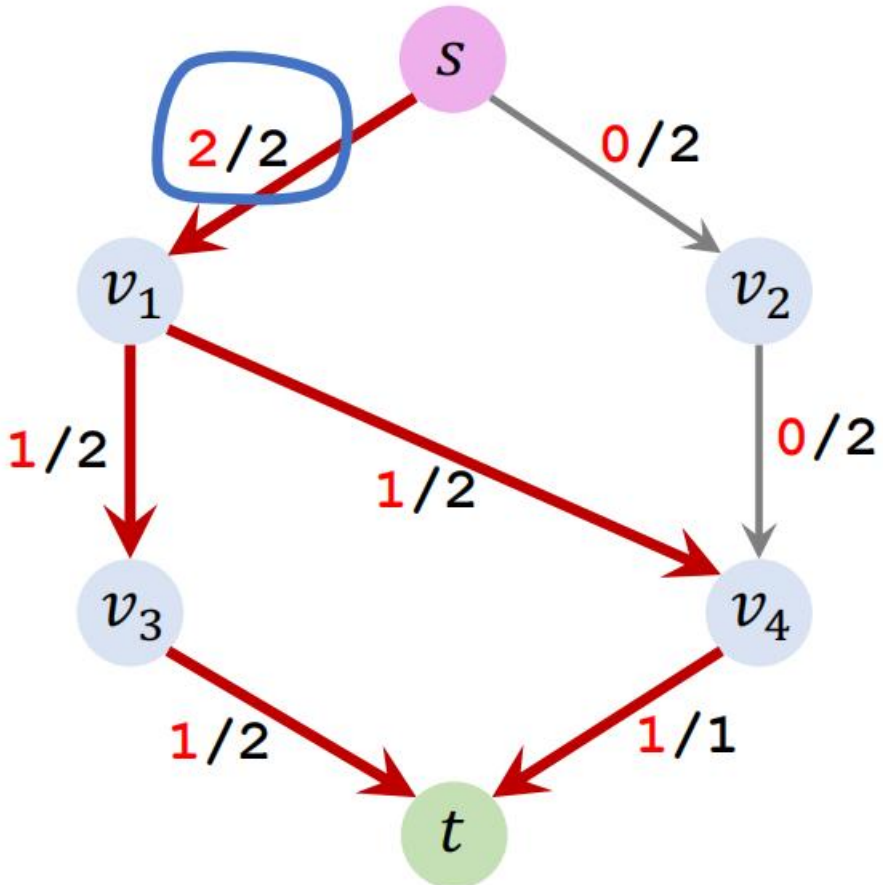
网络流基本模型。

Ford-Fulkerson 算法的复杂度较高，且写法较长。我们推荐学习更高效，更好写的 Dinic 算法。这里有一些参考教程：

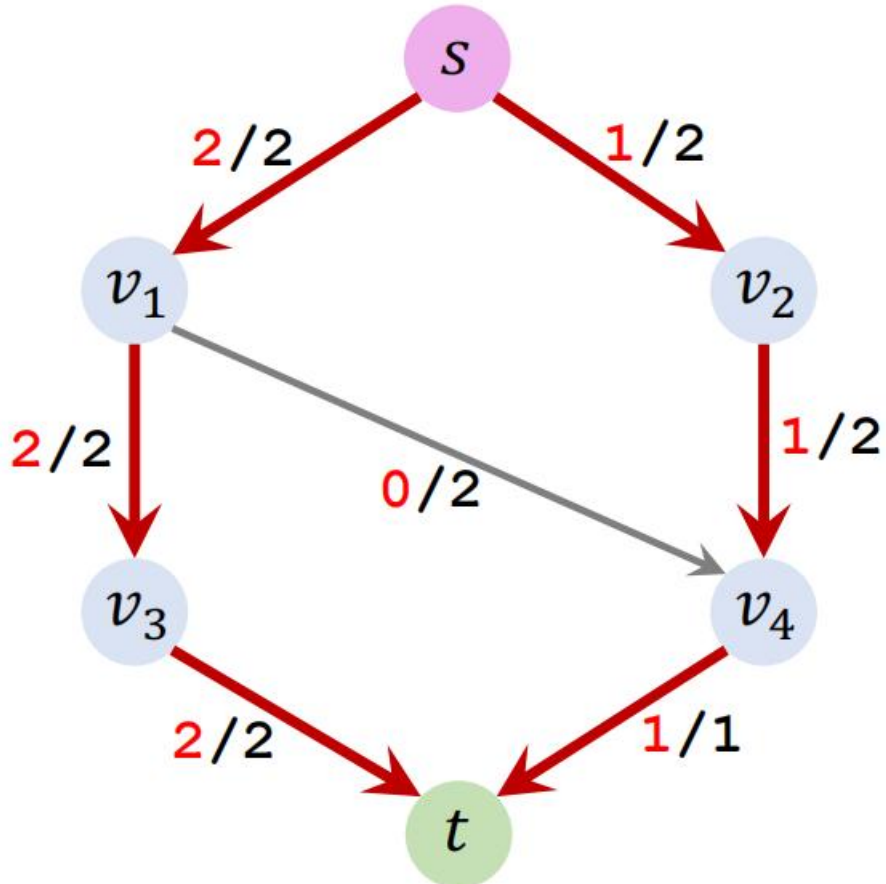
- 网络流详解: 从入门到放弃
- 最大流 - OI Wiki

最终的期末考试中，我们不保证 Ford-Fulkerson 算法可以通过网络流相关题目。

如果我们在层次图 G_L 上找到一个最大的增广流 f_b，使得仅在 G_L 上是不可能找出更大的增广流的，则我们称 f_b 是 G_L 的阻塞流（Blocking Flow）。

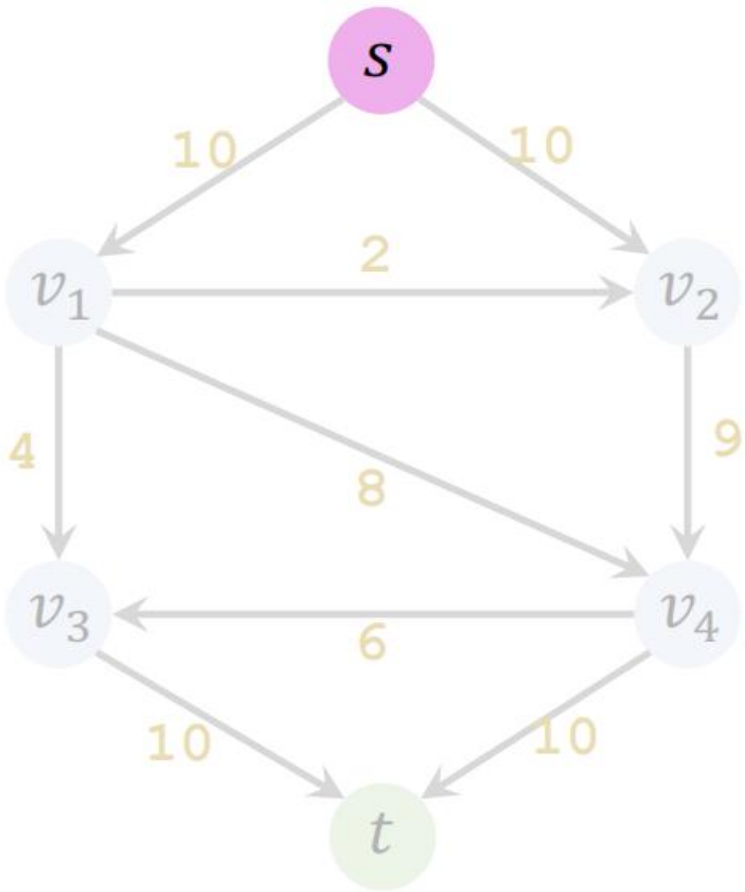


Another Blocking Flow

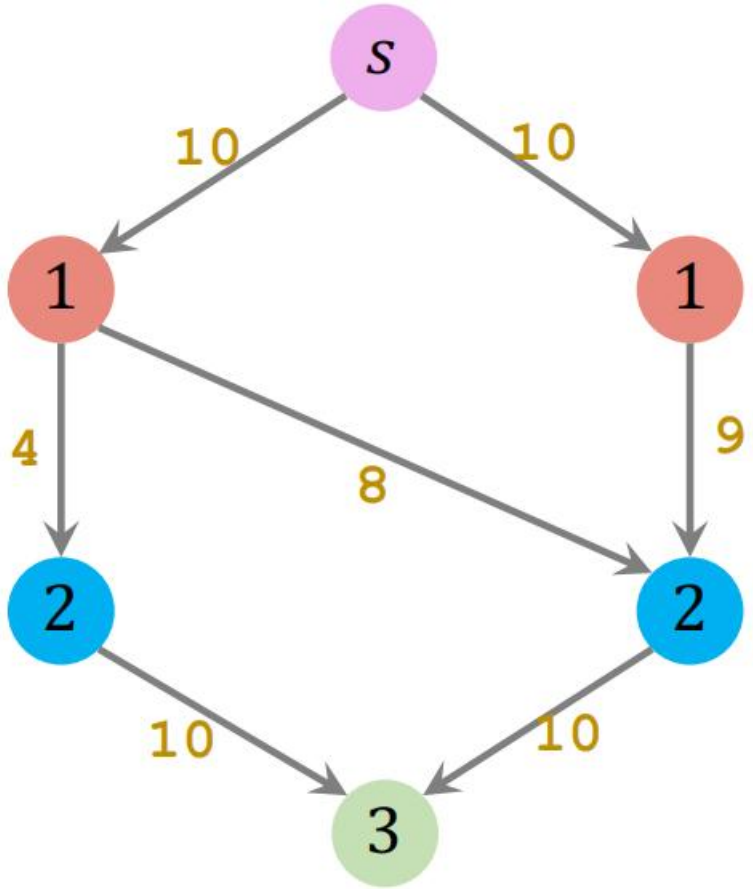


A Blocking Flow

分层图



Original Graph



Level Graph

定义层次图和阻塞流后，Dinic 算法的流程如下。

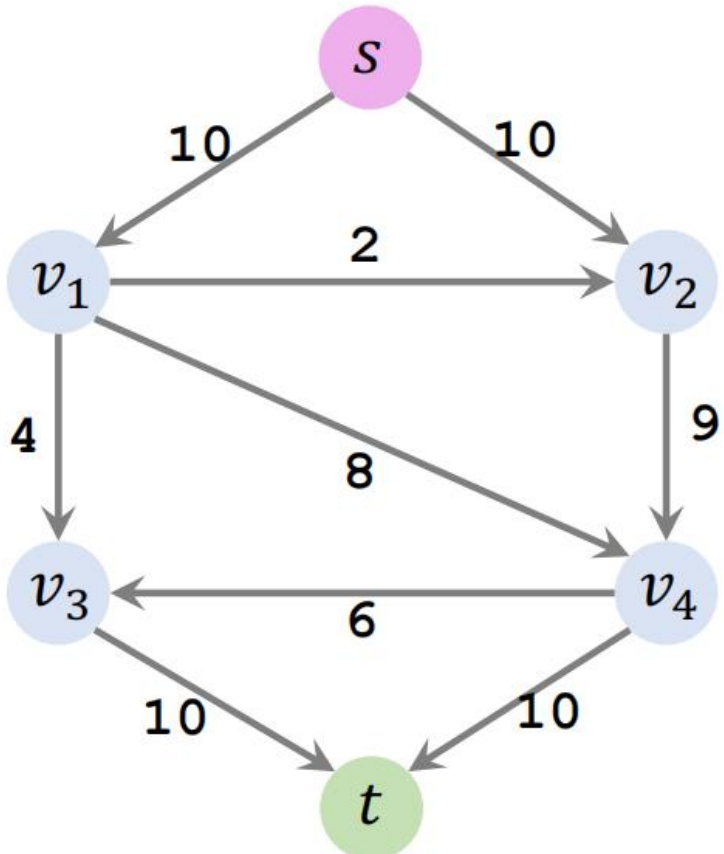
1. 在 G_f 上 BFS 出层次图 G_L。
2. 在 G_L 上 DFS 出阻塞流 f_b。
3. 将 f_b 并到原先的流 f 中，即 $f \leftarrow f + f_b$。
4. 重复以上过程直到不存在从 s 到 t 的路径。

此时的 f 即为最大流。

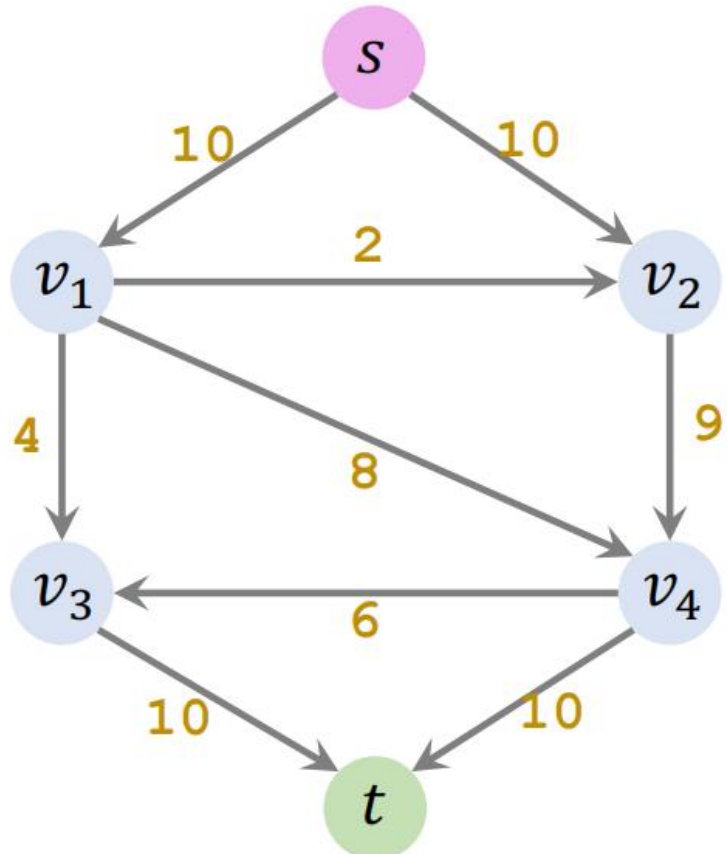
定义层次图和阻塞流后，Dinic 算法的流程如下。

1. 在 G_f 上 BFS 出层次图 G_L。
2. 在 G_L 上 DFS 出阻塞流 f_b。
3. 将 f_b 并到原先的流 f 中，即 $f \leftarrow f + f_b$。
4. 重复以上过程直到不存在从 s 到 t 的路径。

此时的 f 即为最大流。



Original Graph

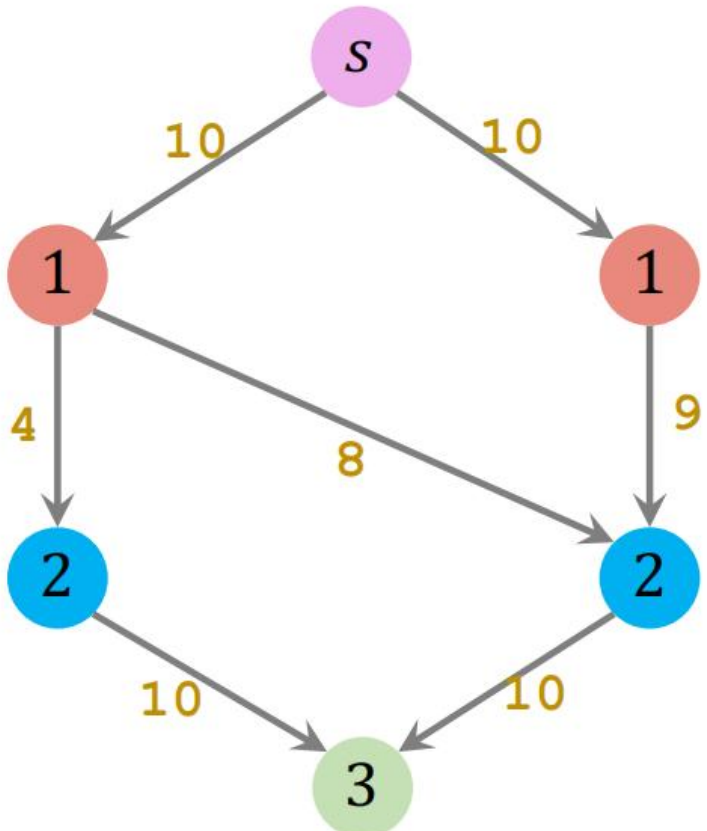


Residual Graph

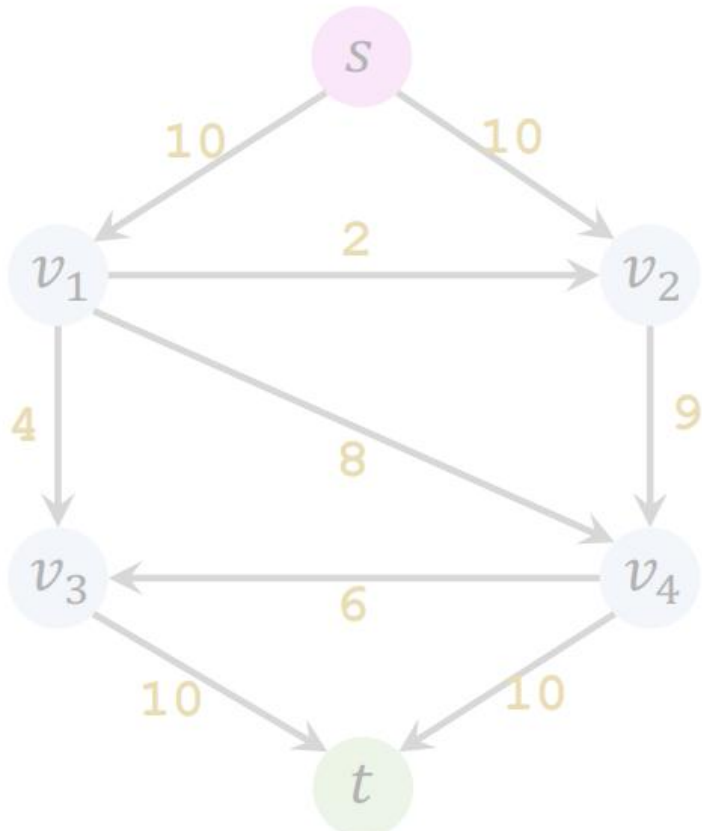
定义层次图和阻塞流后，Dinic 算法的流程如下。

1. 在 G_f 上 BFS 出层次图 G_L。
2. 在 G_L 上 DFS 出阻塞流 f_b。
3. 将 f_b 并到原先的流 f 中，即 $f \leftarrow f + f_b$。
4. 重复以上过程直到不存在从 s 到 t 的路径。

此时的 f 即为最大流。

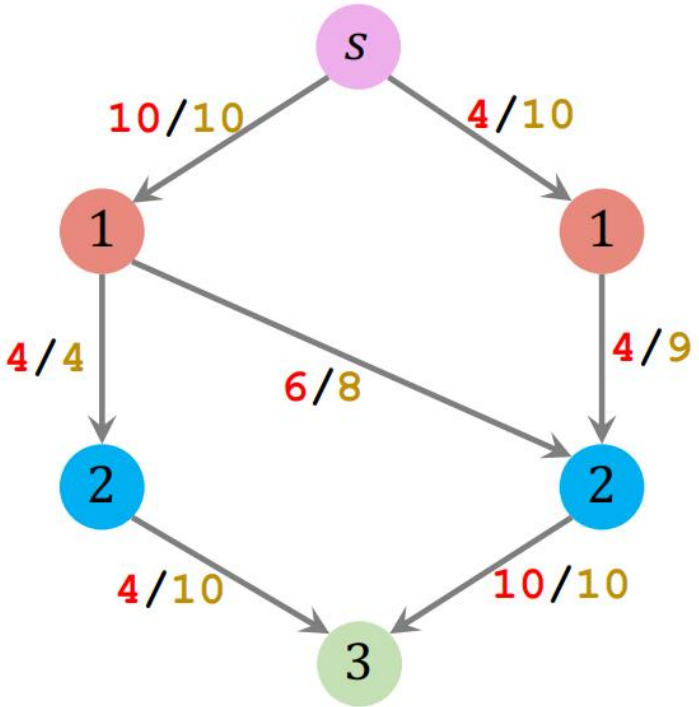


Level Graph

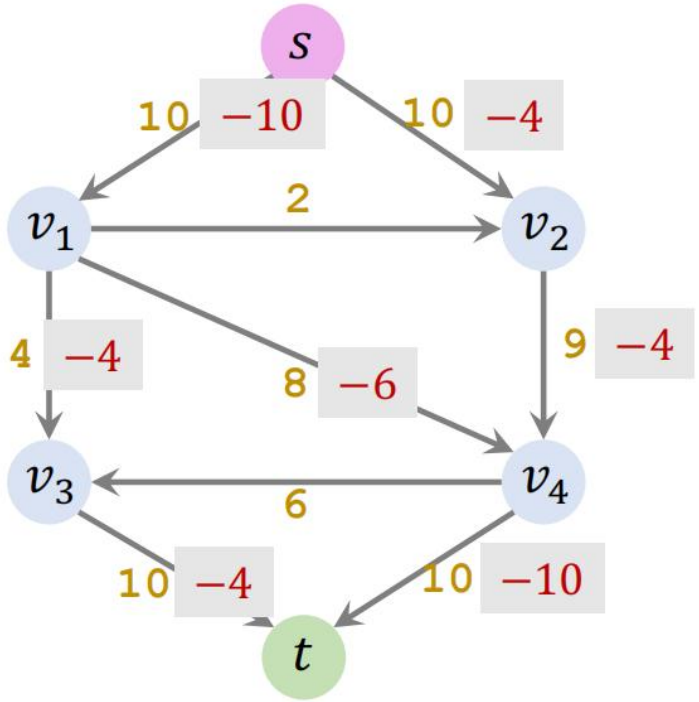


Residual Graph

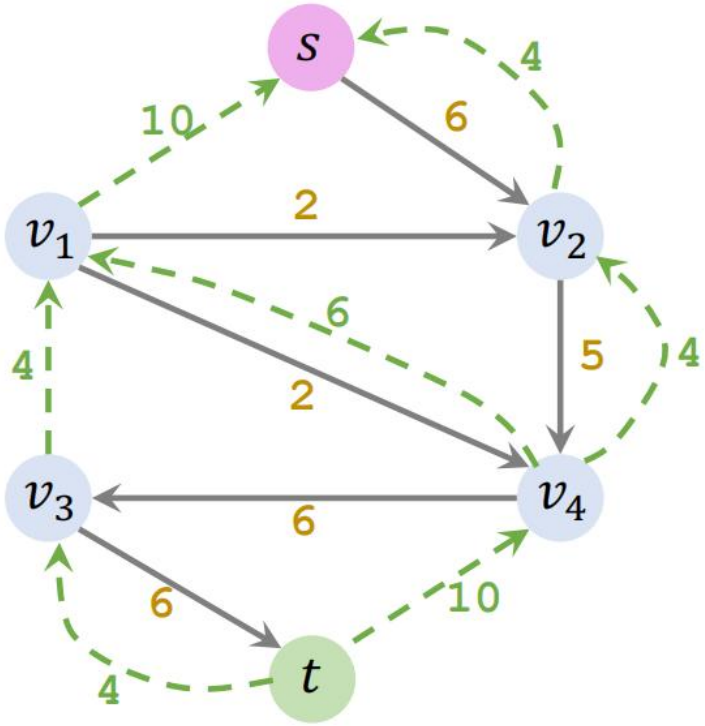
ans=14



Level Graph



Old Residual Graph

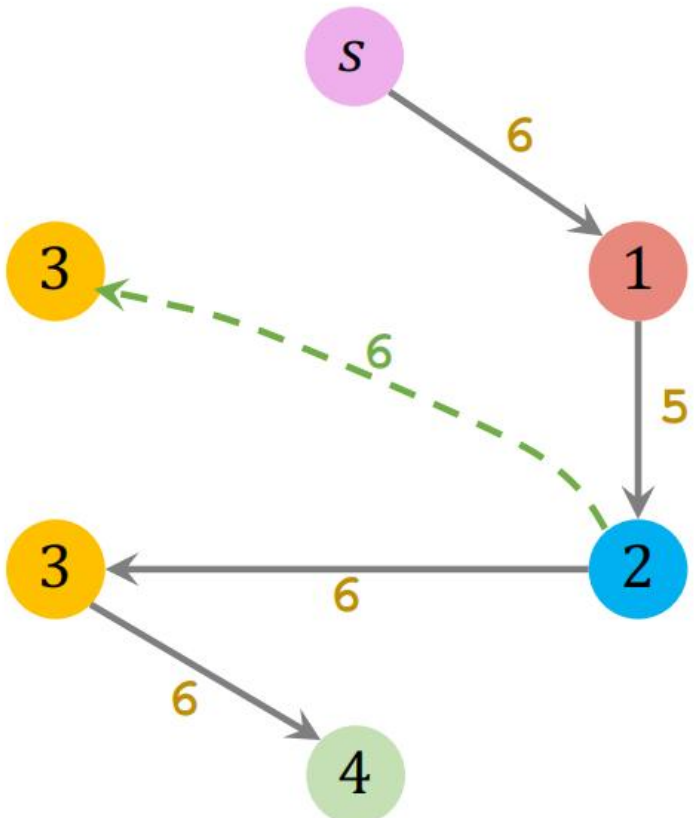


Residual Graph

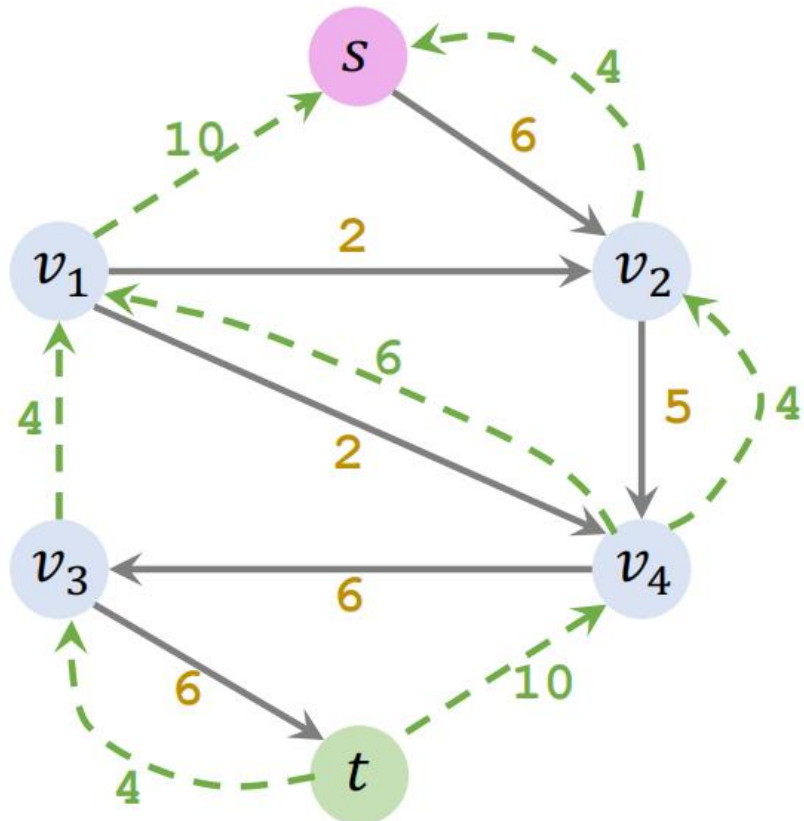
定义层次图和阻塞流后，Dinic 算法的流程如下。

1. 在 G_f 上 BFS 出层次图 G_L。
2. 在 G_L 上 DFS 出阻塞流 f_b。
3. 将 f_b 并到原先的流 f 中，即 $f \leftarrow f + f_b$。
4. 重复以上过程直到不存在从 s 到 t 的路径。

此时的 f 即为最大流。

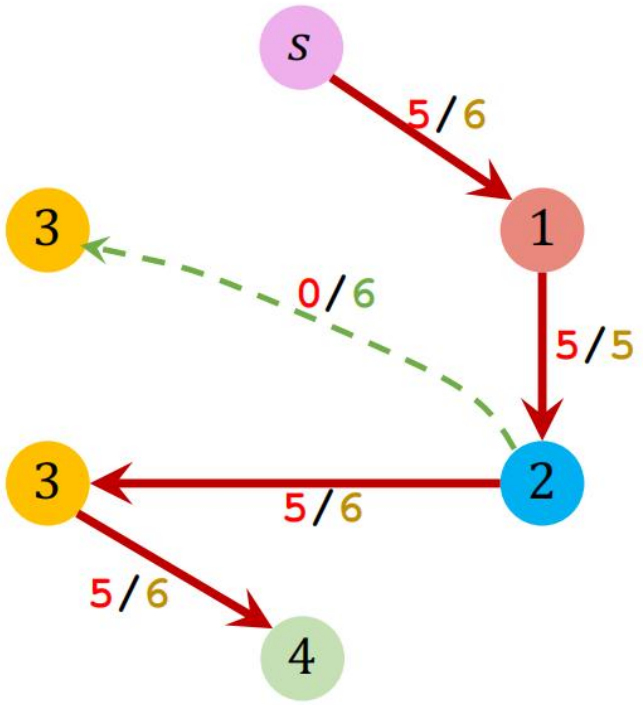


Level Graph

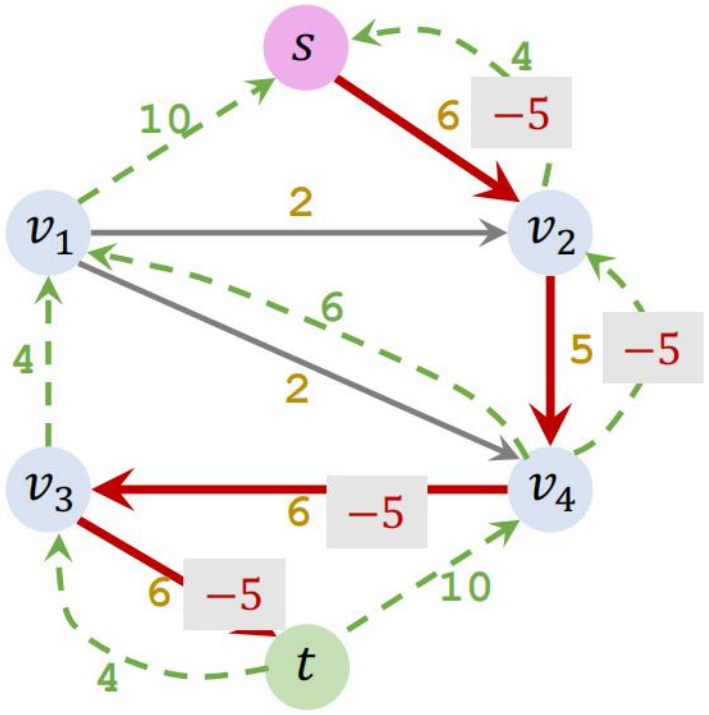


Residual Graph

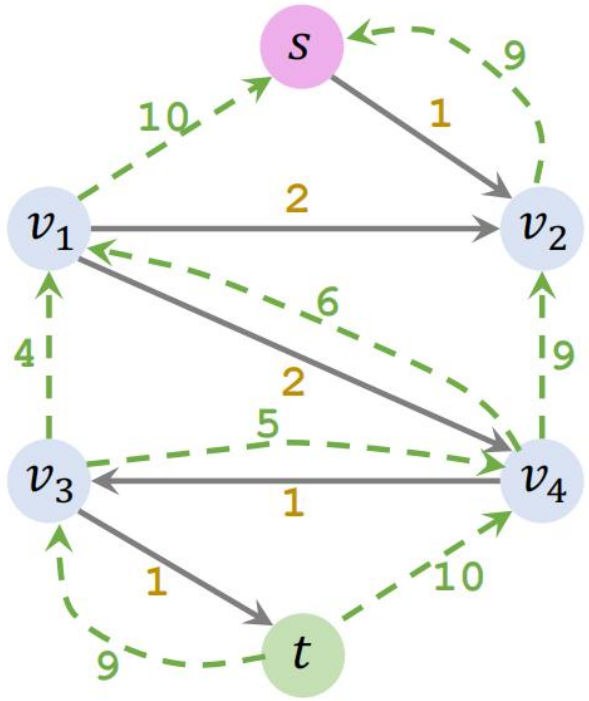
ans=14+5



Blocking Flow



Old Residual Graph



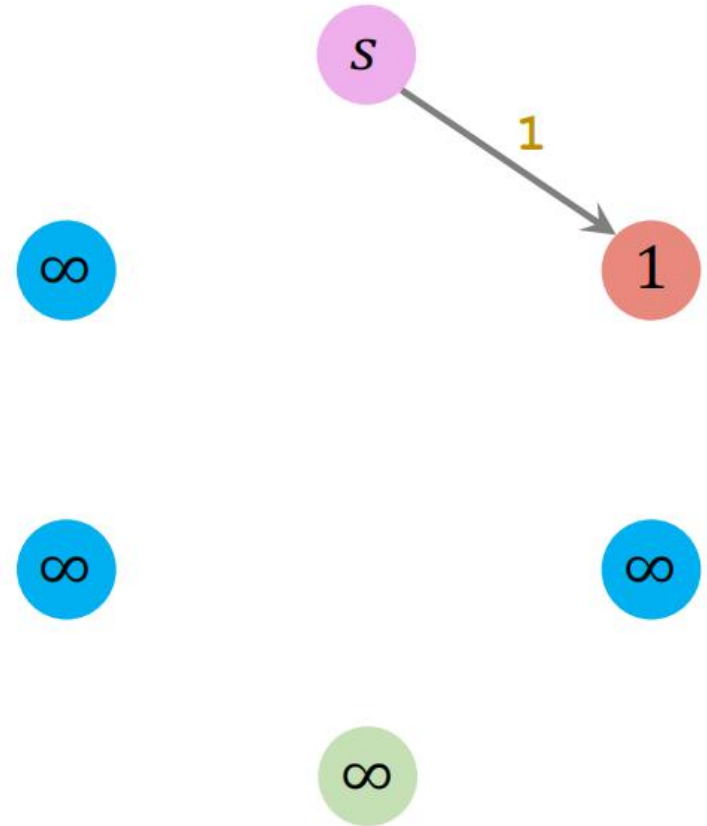
Residual Graph

定义层次图和阻塞流后，Dinic 算法的流程如下。

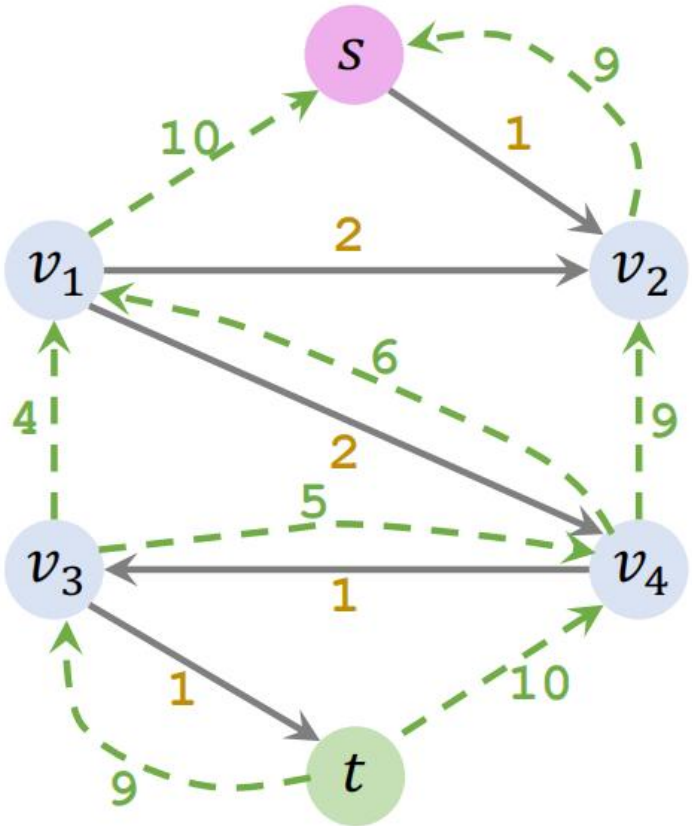
1. 在 G_f 上 BFS 出层次图 G_L。
2. 在 G_L 上 DFS 出阻塞流 f_b。
3. 将 f_b 并到原先的流 f 中，即 $f \leftarrow f + f_b$。
4. 重复以上过程直到不存在从 s 到 t 的路径。

此时的 f 即为最大流。

算法结束，ans=19



Level Graph



Residual Graph

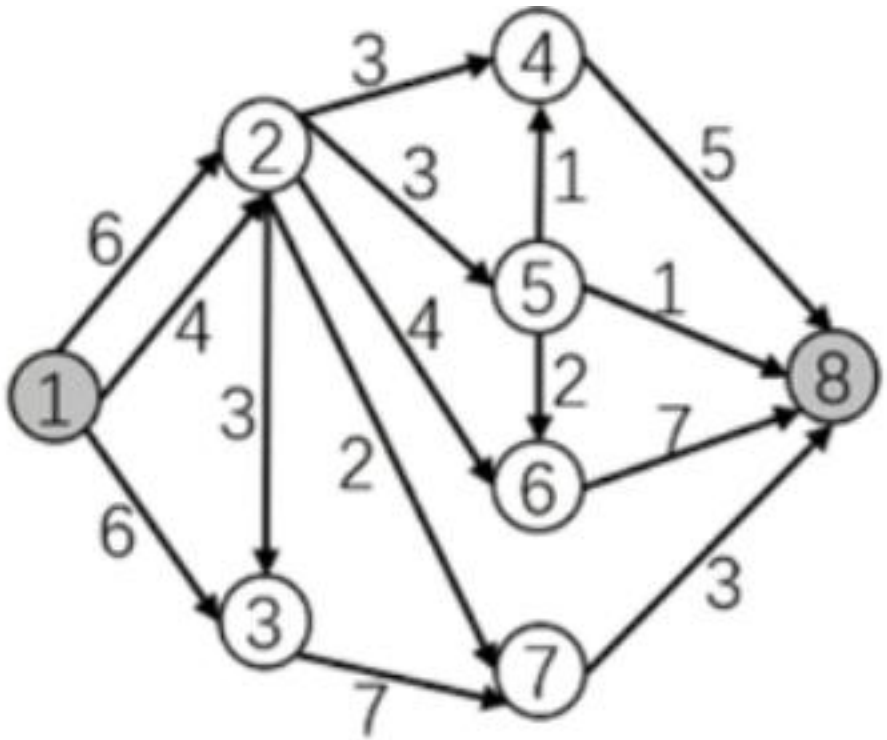
```cpp
bool bfs(){//在残量网络中构造分层图
    for(int i=1;i<=n;i++)
        depth[i]=INT_MAX;
    queue<int> q;
    q.push(s);
    depth[s]=0;
    while(!q.empty()){
        int x=q.front();
        q.pop();
        cur[x]=head[x];
        for(int i=head[x];i;i=edge[i].next){
            int v=edge[i].to;
            if(edge[i].wei>0 && depth[v]==INT_MAX) {
                depth[v]=depth[x]+1;
                q.push(v);
                if(v==t)
                    return true;
            }
        }
    }
    return false;//分层图到不了汇点
}
```

```cpp
long long dfs(int x,long long flow){
    if(x==t)
        return flow;
    long long sum=0;
    for(int i=cur[x];i;i=edge[i].next) {
        cur[x]=i;//当前弧优化
        int v=edge[i].to;
        if(edge[i].wei>0&&(depth[v]==depth[x]+1)) {
            long long tmp=dfs(v,min(flow,edge[i].wei));
            flow-=tmp;
            sum+=tmp;
            edge[i].wei-=tmp;
            edge[i^1].wei+=tmp;
            if(flow==0)
                break;//余量优化
        }
    }
    if(sum==0)//残枝优化
        depth[x]=INT_MAX;
    return sum;
}
```

```cpp
long long dinic(){
    long long ans=0;
    while(bfs())
        ans+=dfs(s,inf);
    return ans;
}
```

```cpp
for(int i=1;i<=m;i++) {
    cin>>u>>v>>w;
    add(u,v,w);
    add(v,u,0);
}
```

```cpp
long long dfs(int x,long long flow){
    if(x==t)
        return flow;
    long long sum=0;
    for(int i=cur[x];i;i=edge[i].next) {
        cur[x]=i;//当前弧优化
        int v=edge[i].to;
        if(edge[i].wei>0&&(depth[v]==depth[x]+1)) {
            long long tmp=dfs(v,min(flow,edge[i].wei));
            flow-=tmp;
            sum+=tmp;
            edge[i].wei-=tmp;
            edge[i^1].wei+=tmp;
            if(flow==0)
                break;//余量优化
        }
    }
    if(sum==0)//残枝优化
        depth[x]=INT_MAX;
    return sum;
}
```



```cpp
struct edge{
    int ed;//指向哪条边
    int wei;//流
    int id;//为了获取反向边而设立的编号
};
vector <edge> e[N];
```

```cpp
for(int i=1;i<=m;i++){
    cin>>u>>v>>w;
    int st = e[u].size();
    int ed = e[v].size();
    e[u].push_back({v, w, ed});
    e[v].push_back({u, 0, st});
}
```

vector做，也不是不行

Thanks~