**深 圳 大 学 实 验 报 告**

**课程名称：­ 算法设计与分析**

**实验项目名称： 最大流应用**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 杨烜**

**报告人： 陈述 学号： 2020281051**

**实验时间： 2022/6/17**

**实验报告提交时间： 2022/6/18**

**教务部制**

|  |  |
| --- | --- |
| **一、实验目的：**  （1） 掌握最大流算法思想。  （2） 学会用最大流算法求解应用问题。 | |
| **二、问题描述：**  1. 有m篇论文和n个评审，每篇论文需要安排a个评审，每个评审最多评b篇论文。请设计一个论文分配方案。  2. 要求应用最大流解决上述问题，画出m=10，n=3的流网络图并解释说明流网络图与论文评审问题的关系。  3. 编程实现所设计算法，计算a和b取不同值情况下的分配方案，如果没有可行方案则输出无解。 | |
| **三、算法设计原理：**  **（一）理解流网络**  **1.什么是流网络？**  ·流网络其实就是一个连通的有向图，图中每条边都有一个非负的容量值。  ·其拥有一个源点和一个汇点，顾名思义想象有一定的流量从源点一直流到汇点  ·其拥有一个重要的概念就是流量守恒，也就是说除了流的源点和汇点外，其他任何一个结点接收多少输入，就必须输出多少，结点自身是不存储流量。    图表 1 流网络例图  **2.重要概念理解**  **（1）增广路径：**从源点到汇点流量未满的有向路径。  如下图，如果将节点1看作源点，将节点4看作汇点，那路径1->2->3->4就是一条增广路径。    图表 2 增广路径例图  **（2）关键边：**该增广路径上拥有最小权值的边，该边决定该路径最多能通过多少流量。每条增广路径都有至少一条的关键边。  （如下图增广路径S->2->1->T，关键边就为2->1，此最多只能流过一个单位的量。）    图表 3 关键边例图  **（3）最大流：**每条增广路径上的最小权值之和，即该流网络最多能流过多少流量。  （如下图，流过该网络的增广路径总共有两条，将两条增广路上的最小权值相加就为其最大流。）    图表 4 最大流例图  **（4）残留网络：**通过添加反向边，实现流量退回的操作，即给程序一个“后悔”的机会。  （还是拿上图的例子，如果第一次选该路径通过一个单位的流量后，节点1到节点4就没有可行流了，这样就造成我们的最大流计算错误。为了解决这个问题引入残留网络。）    图表 5 解释残留网络存在意义  如下图，在我找到一条增广路径之后，我们将该条路径上每条边都减去最小权值，同时增加反向边，反向边的容量即为上述的最小权值。    图表 6 增广路径增加反向边  如此得到的图即为残留网络。为了更加直观，我们把容量为0的边隐藏，并在此残留网络上继续寻找增广路径，直到最后没有增广路径。    图表 7 基于残留网络寻找增广路径  如此操作，即可计算得到正确的最大流值。  **（二）基于问题构建其网络流**  **1.设计流网络**    图表 8 m=10,n=3的网络层次图  根据问题描述做出上述流网络，其总共分为四层，第一层和第四层分别为源点和汇点，中间两层分别为需要评审的论文数和评审论文的评委数。   1. 源点至论文：源点到论文的有向边的容量为a（每篇论文需要被评审的数目）。该层主要决定从源点流出的流量总和为a\*m。 2. 论文至评委：每篇论文都有指向每个评委的有向边，每条边的容量为1.若该边流量为0，则说明该评委没有评审该论文；相反若为1，则表示该论文被这个评委评审。 3. 评委至汇点：评委到汇点的有向边的容量为b（每个评委最多能评审的论文数）。而该边的流量表示该评委评审的论文数。   **2.** **流网络中最大流与论文评审问题解的关系**  ①基于每个评委不能对同一篇论文重复评审，所以每篇论文需要的评审数量要少于或等于评审的人数，即  a ≤ n  ②论文需要的评审总次数小于等于评审们最多能评审的数量  A × m ≤ b × n  综上又结合网络流流量守恒的性质，该问题就是计算上述流网络的最大流，如果（**a × m = 最大流**）则该问题有可行解。  **（二）算法求解最大流**  **1.** **Ford-Fulkerson方法**  **（1）方法思想：**  ·找到一条增广路径  ·求这条路径的最小权值（该路径最大流）  ·更新权值并起来画出其残留网络  ·在残留网络中继续寻找增广路径  ·重复以上步骤直到没有增广路径到汇点  **（2）伪代码**  Ford-Fulkerson(G,s,t)  for each edge(u,v) in G.E  (u,v).f=0//初始化流  while there exits a path p //找到增广路  cf(p)=minF//路径上最小权值  for each edge(u,v) in p  if(u,v) in E  (u,v).f=(u,v).f+cf(p)//增广  else (v,u).f=(v,u).f-cf(p)  **（3）路径选择造成重复问题**  我们知道，一开始对增广路径的选择非常重要，如下图，如果我们选择的第一条增广路径为S->2->1->T时，我们就需要在两条增广路（S->2->1->T和S->1->2->T）之间反复执行2000000次，才可以结束程序，可想而知这效率很低。    图表 9 路径选择造成重复问题示例图  实际上，如果我们一开始选择增广路径为S->1->T或S->2->T，则我们执行2次就可以结束程序得到最大流。    图表 10 路径选择无重复问题示例图  所以对于实现Ford-Fulkerson方法，选择一条最短的增广路径很重要。下面就介绍Ford-Fulkerson方法的一种实现算法——Edmonds-Karp 算法。   1. **Edmonds-Karp 算法实现**   Ford-Fulkerson方法只能称之为一种方法而不是算法，原因在于它并没有说明找增广路径的方法。而Edmonds-Karp 算法就是Ford-Fulkerson方法的一种实现，其利用**BFS**寻找增广路径，使得每次找到的增广路径都为**最短路径**。   1. **伪代码**   EK\_bfs()  memset(st, false, sizeof st)  q[0] = S, st[S] = true, d[S] = INF  while(！queue.empty())//队列不空  t = q[hh ++ ] //入队  for( i = h[t]; ~i; i = ne[i])  ver = e[i]  if(!st[ver] && f[i]) //未被访问且有流量  st[ver] = true  d[ver] = min(d[t], f[i]); //记录增广路上容量最小值  pre[ver] = i //记录前向边  if(ver == T) return true  q[ ++ tt] = ver;  return false;  EK()  int r = 0;  while(EK\_bfs())  r += d[T]  for(int i = T; i != S; i = e[pre[i] ^ 1]) //增广  f[pre[i]] -= d[T], f[pre[i] ^ 1] += d[T]  res = r   1. **时间复杂度分析**   **·一次BFS寻找增广路径**  首先我们知道一次BFS寻找增广路径的时间复杂度为 **O(V+E)**。  **·增广操作**  增广操作主要的时间消耗在增加反向边，因为增广路径没有环，所以时间复杂度为**O（V）**  **·最多有几条增广路径（需要做几次增广）？**  我们知道每次找到的一条增广路径都至少有一条关键边，而每条边都有成为关键边的机会，那  **增广路径总数=边数\*每条边成为关键边的次数**  要解决这个问题，我们需要先明确两个定理：增广路径非递减，成为关键边的次数有限。  以下分别为两个定理的证明：  ①**增广路径非递减：**  不妨假设下一次 BFS 搜索出现了长度更短的路径 p ′，则一定出现反向边( v , u ) ， p ′ = ( s → v → u → t ) 。而 BFS 性质保证第一次搜索中 d i s ( s , u )与 d i s ( v , t )都是最短的，此时有    如果有更短的边，则：    但是我们知道    所以结果与我们的假设矛盾，所以可以证明得到不存在长度更短的路径p ′，即BFS 找到的增广路径长度是递增的。  **②成为关键边的次数有限。**  设 u 和 v 为集合 V 中的两个节点，且这两个节点由 E 中的一条有向边连通。由于增广路径都是最短路径，因此当边( u , v )第一次成为关键边时，此时有：    在对流进行增加后，边( u , v )将被从残留网络中删除。并且，以后也不会重新出现，在另一条增广路径上，直到从 u到 v的网络流减小后为止，并且只有当( u , v )出现在增广路径上时，这种情况才会发生。如果当这一事件发生时 f ′ 是 G的流，则有    根据上述证明的增广路径非递减定理：    最后可以联立得到这个下面的式子：    因此，从边( u , v )成为关键边到下一次再成为关键边，从源点 s到节点 u的距离至少增加两个单位，而从源点 s 到节点 u的最初距离至少为0，从 s到 u的最短路径上的中间节点不可能包括节点 s、u 或者 t（因为边( u , v )处于一条增广路径上意味着 u与 t不同）。因此，一直到节点 u成为不可达的节点前，其距离最多为∣ V ∣ − 2。因此，在边( u , v )第一次成为关键边时，它还可以至多再成为次关键边，即边( u , v )成为关键边的总次数为。所以最多总共有条增广路径。  综上，**时间复杂度为**  由于上述的网络流关系，对于**稠密图**来说，其实E要远远大于V，所以实际一次的时间复杂度可近似于**O(E)**。如基于本次评审问题，顶点数V=2+m+n;边数E=m+n\*m+n;当数量级较大时E>>V。所以以上复杂度可以约为**。**  **2.Dinic算法**   1. **与EK算法的不同**   Dinic算法是EK算法的优化，他们之间的**区别**在于，dinic算法在求增长通路的时候，先BFS进行图的分层操作，在DFS寻找增长通路的时候，榨干一条路径上的所有流量。简单的说就是，在Dinic算法中，我们用**一个dfs过程代替多次bfs**来寻找阻塞流。   1. **算法思想（流程）：**   ·初始化流量  ·根据残留网络利用BFS分层  ·在层次图内用DFS进行增广（每次只找下一层的点增广）  ·重复至汇点不在层次图内    图表 11 Dinic算法 示例图   1. **伪代码**   DFS(x, flow)//寻找增广路径  if x==汇点  return flow  for y in 邻接[x]  fxy = 边xy允许的流量最大值  r = DFS(y, min(flow, fxy))  if r≠ 0  更新边xy的权值  return r  return 0  Dinic()  while true  flag = BFS()  if !flag  break  while true  r = DFS(起点, infinity)  if r = 0  break   1. **优化** 2. **多路增广**   我们知道在寻找一条增广路径时，我们往往都会退回到源点再查找下一条增广路径。而多路增广的思想主要是将路径退回到父节点观察能否还有路径可以增广。如此操作可以减少递归造成的时间消耗。    图表 12 多路增广思路   1. **废点优化**   当我们完成一次操作时，可以观察到如果一个点流入它的流量为0，那我们就可以把该点的层数设置为-1，这样下一次我们就不用将该点考虑进增广路径。    图表 13 废点优化   1. **当前弧优化**   该优化思想也很简单，基于多路增广，我们知道当一条边被增广过后，该边的流量就一定达到了饱和，那么下一次增广时我们就不用再对这条边进行遍历。    图表 14 当前弧优化  **优化后伪代码：**  //多路增广  find(int u, int limit)  if (u == T) return limit  flow = 0  for (i = cur[u]; ~i && flow < limit; i = ne[i])  cur[u] = i // 当前弧优化  ver = e[i]  if (d[ver] == d[u] + 1 && f[i])  t = find(ver, min(f[i], limit - flow))  if (!t) d[ver] = -1 //剪枝，去掉增广过的点（废点优化）  f[i] -= t, f[i ^ 1] += t, flow += t  return flow   1. **时间复杂度分析：** 2. BFS 建立层网络，耗时（与 EK 算法中 BFS 时间消耗一致）；   ② 一次 DFS 中搜索出来的增广路径也存在关键边，且关键边的个数为，即一次 DFS 可能搜索出条增广路径，对残留网络进行修改需要，所以一次 DFS 时间消耗为。  ③与 EK 算法相似，Dinic 算法存在层网络限制每次 DFS 找到的增广路径长度递增，且最大长度为**O ( V )**。  综上，Dinic 算法的时间复杂度为  因为该问题基于稠密图所以以上时间复杂度可以近似为 | |
| **四、实验数据及分析：**  1.小规模数据  m=10，n=3，通过改变不同的a（每篇论文需要评审数），b（一个评委最多评审论文数）值，判断算法的正确性，即计算最大流，判断是否有可行解。    图表 15 最大流小规模验证  如上图表格数据显示，当a\*m>最大流时，方案无解。  2.在其他值都不变的条件下，分别改变a、b、n、m各个值，观察EK算法与Dinic算法的时间效率曲线  不变数据分别为（b=25,n=500,m=500）（a=5,n=500,m=500）（a=5,b=25,m=500）（a=5,b=25,n=500）    图表 16 EK算法与Dinic算法效率比较  分析：   1. 看出优化后的Dinic算法明显比EK算法拥有更好的效率。 2. 观察到随着a,n,m的增大，EK消耗的时间都在上升。因为n和m决定着图的规模，两者越大，图的顶点数就越大，稠密图上的边就更多。而在其他数不变的条件下，a决定着总流量的大小，流量越大，需要查找的增广路就越多。 3. 与其他不同的是，增大b的值，时间效率反而递减。因为b决定着末尾流过的流量的限制大小，当b的值大了，在一条增广路径上能通过的流量就多，这样就不用重复多次寻找下一条增广路就能达到最大流，也就是说当b越大，我们就可以用越少的路径流过相同的流量，所以时间消耗会减少。   3.再次扩大数据单独观察Dinic算法的效率    可见Dinic算法总是很稳定。 |
| **五、实验结论：**  通过本次最大流应用问题实验，我学习了如何根据实际问题建立流网络，并可以明确现实问题的解空间中可行解和流网络中可行流之间一一对应的关系，进而我们就可以使用学习到的最大流算法来求解问题的最大可行解。除了该类问题，生活中也有很多问题可以利用流网络算法思想进行解决，如二分图匹配、任务分配问题和运输问题等。 | |
| **指导教师批阅意见：**  **成绩评定：**  **指导教师签字：**  **年 月 日** | |
| **备注：** | |

**注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。**

**2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内**。