**求强连通分量的算法调研总结**

**摘要**: 图的连通性问题是图论中重要的研究问题。在有向图中，若对于每一对顶点n和m，都存在一条从n到m和从m到n的路径，则称此图为强连通图，非强连通图的极大强连通子图称为强连通分量。给定一个有向图，求解其强连通分量显得格外重要，本文探讨了求解有向图强连通分量的算法，给出了多种求解图的强连通分量的方式。

**关键词:** 强连通分量，Tarjan算法，Garbow算法

**Research summary of algorithms for finding strongly connected components**

**Abstract**: The connectivity of graphs is an important problem in graph theory. In a directed graph, if there is a path from n to m and from m to n for each pair of vertices n and m, the graph is called a strongly connected graph, and the maximum strongly connected subgraph of a non strongly connected graph is called a strongly connected component. Given a directed graph, it is particularly important to solve its strongly connected component. This paper discusses the algorithm for solving the strongly connected component of a directed graph, and gives a variety of ways to solve the strongly connected component of a graph.

**Key Words**: SCC; Tarjan algorithm; Garbow algorithm

—————————————

**1 引 言**

图的连通性问题是图论中重要的研究问题，求解强连通分量问题是图的连通性问题的重要组成部分。强连通分量的求解，在工程管理、交通道路规划、活动组织等实际应用领域有着重大意义。求强连通分量的常用方法有Kosaraju算法，Tarjan算法及Gabow算法，其中Kosaraju算法效率较低，实际应用中较不常用，本文重点调研了求解强连通分量的Tarjan算法，Garbow算法，并简要分析了两个个算法的时间复杂度及执行效率，给出了求解强连通分量的选择。

**2 问题描述及相关理论定义**

**2.1 问题描述**

给定一个有向图，求解其强连通分量。

**2.2 相关理论定义**

图定义：图是由顶点集合及顶点间的关系集合组成的一种数据结构。计算机存储中，图包括了一顶点的有穷非空集合及顶点间关系的有穷集合。顶点间关系称为边，顶点间关系的有穷集合亦称为边集合。

有向图定义：有向图是指顶点间关系为有序边的图。

无向图定义：无向图是指顶点间关系为无序边的图。

连通图与连通分量：在无向图中，若从顶点n到顶点m间有路径，则称顶点n和m是连通的。若图中任意一对顶点都是连通的，则称此图是连通图。非连通图的极大连通子图称为连通分量。

强连通图与强连通分量定义：如果两个顶点可以相互通达，则称两个顶点**强连通。**在有向图中，对于每一对顶点n和m，若从顶点n到顶点m和从顶点m到顶点n都存在一条路径，则称此图为强连通图。非强连通图的强连通子图称为强连通分量。

**3 Tarjan算法**

**3.1 算法简介**

Robert Tarjan是一位美国计算机科学家，以[LCA](https://baike.baidu.com/item/LCA/20414490?fromModule=lemma_inlink)、[强连通分量](https://baike.baidu.com/item/%E5%BC%BA%E8%BF%9E%E9%80%9A%E5%88%86%E9%87%8F/7448759?fromModule=lemma_inlink" \t "_blank)等[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95/209025?fromModule=lemma_inlink)闻名，设计了求解的应用领域的许多问题的广泛有效的算法。Tarjan一种由Robert Tarjan提出的求解有向图[强连通分量](https://baike.baidu.com/item/%E5%BC%BA%E8%BF%9E%E9%80%9A%E5%88%86%E9%87%8F/7448759?fromModule=lemma_inlink)的线性时间的算法。Tarjan算法是基于对图深度优先搜索的算法，每个强连通分量为搜索树中的一棵子树。搜索时，把当前搜索树中未处理的节点加入一个堆栈，回溯时可以判断栈顶到栈中的节点是否为一个强连通分量。

**3.2 算法基础 – 图的深度优先遍历DFS**

图的遍历是指从已给的连通图中某一个顶点出发，沿着一些边访遍图中的所有顶点，且使每一个顶点仅被访问一次。深度优先遍历属于图遍历中的一种，简称DFS，简要过程是对图中每一个可能的分支路径深入到不能深入为止，且在该过程中每个顶点只能访问一次。

DFS算法的基本思路为，在访问图中某一个起始顶点v后，由v出发，访问它的任意一个邻接顶点W1；再从W1出发，访问与W1相邻但还没有被访问过的顶点W2；然后再从W2出发，进行类似的访问。如此进行下去，直到到达的顶点的所有邻接顶点都被访问过为止。接着，回退一步，退到前一次被访问过的顶点，看其还有没有其他未被访问过的顶点，若有，则访问此顶点，再从此顶点出发，进行与前述类似的访问；如果没有，则再退回一步进行搜索，不断重复上述过程，直到连通图中所有的顶点都被访问过为止。完成上述过程，就是完成了图的一次深度优先遍历。深度优先遍历的流程图如图1所示，搜索示例如图2所示。

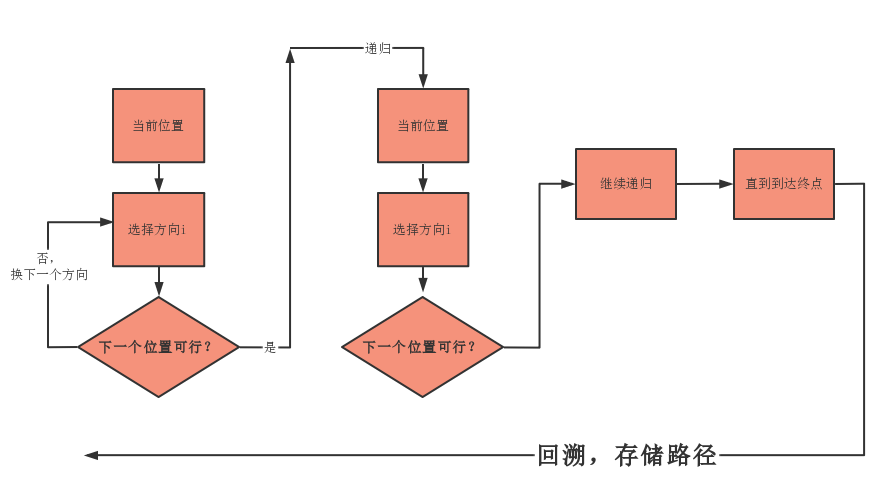


图1

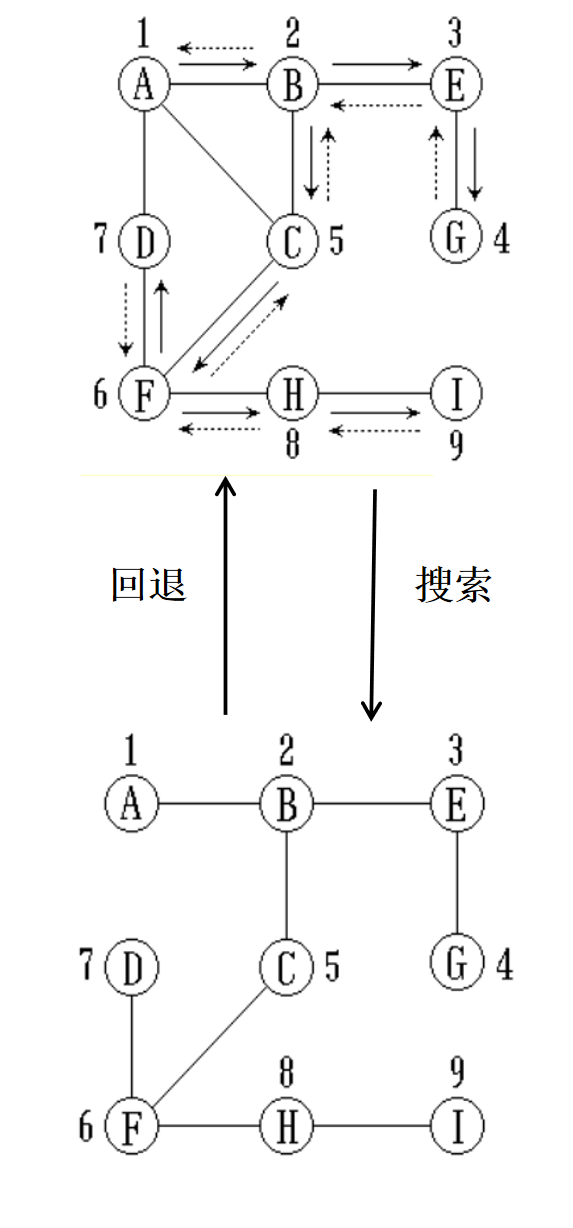


图2

**3.3 Tarjan算法原理**

在利用Tarjan算法完成查找强连通分量前，根据深度优先遍历对图中的所有边进行如下的分类。

1. 树边：指深度优先搜索树上的边。
2. 后向边：指将顶点u连接到其深度优先搜索树中的祖先顶点v的边u->v。即v被访问过，且v比u先被访问。
3. 前向边：指将顶点u连接到其在深度优先搜索树中的后代顶点v的边u->v。即v被访问过，v比u后被访问。
4. 横向边：除了上述边以外的其他边的统称，即边上的两个顶点没有祖先及后代的关系。

以图3为例，对图进行深度优先遍历的遍历次序为1->2->3->4->5->6->7，则图中的边可表示如下。

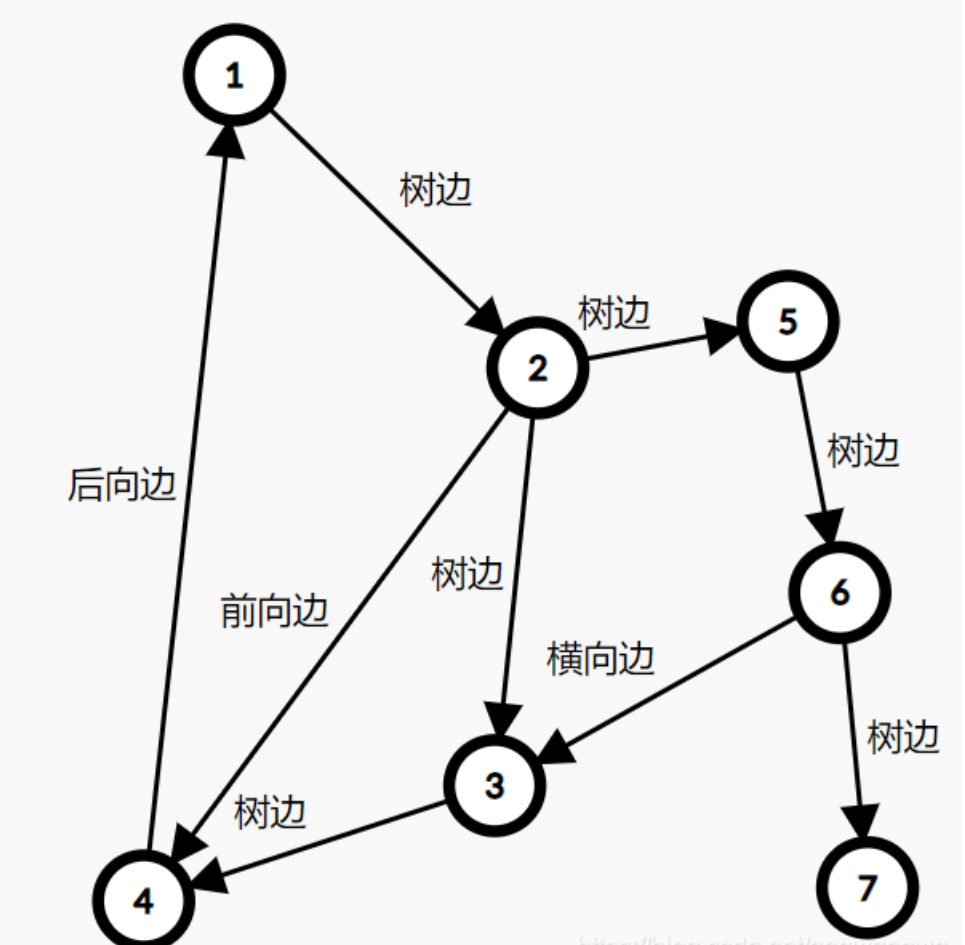


图3

根据图进行分析，前向边的出现不会多生成环，及不会对强连通分量产生影响；后向边的出现必然新产生一个环，必然对强连通分量造成影响；横向边的出现可能生成环，也可能不生成环。

可见，后向边和横向边是构成强连通分量的关键因素，对于前向边对判断强连通分量起到的作用不大，可以忽略。Tarjan算法的核心就是在深度优先搜索的基础上，对树边及后向边和横向边进行考察，分析并通过后向边找到所有强连通分量。

**3.4 Tarjan算法实现**

Tarjan算法是基于对图[深度优先搜索](https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2?fromModule=lemma_inlink)的算法，由强连通分量的定义可以知道，图中的每一个强连通分量可以作为深度优先搜索树中的一棵子树。那么，只要确定了每一个强连通分量的根，然后根据这些根从树的最底层开始，一个一个取出强连通分量即可。具体实现过程中，我们在进行搜索时，把当前搜索树中的未处理顶点放入一个栈，回溯时就可以判断栈顶到栈中的顶点是否为一个强连通分量。

为实现Tarjan算法，在代码编写过程中，需要定义并维护两个数组，一个是dfn[MAXN]，一个是low[MAXN]，MAXN为图中的顶点数。

其中dfn数组为时间戳，dfn[v]就表示顶点v被访问的时间，在dfs遍历时，遍历到的第一个顶点就打上1这个时间戳，遍历到的第2个顶点就打上2这个时间戳，第n个顶点就打上第n个时间戳，dfs数组在算法实现过程中起到了记录深度优先遍历的次序及记录了图中的顶点是否遍历过的作用。对dfs数组的处理操作，本质上是对有向图中的树边进行分析。

另一数组low存放顶点通过深度优先搜索树的回边能到达的最低深度优先树即最小的时间戳，数组low的记录方式与dfn相同，每一个顶点的low值与dfn值相同，当进行深度优先遍历时，如果遍历的有向图中存在环路，那么在遍历过程中，必然会出现遍历一条当前顶点指向已经遍历过的顶点的有向边，且该有向边指向的结点不为有向边的起始顶点在深度优先搜索树中的父结点，即该边为**后向边**，是求解强连通分量的关键边，该边指向顶点的时间戳必然必自身要小，那么就将当前顶点的low值更新为指向顶点的low值，深度优先遍历后进行回退时，若结点的low值比自己的子结点low值要小，则当前结点的low值需更新为子结点的low值。对low数组的操作本质上是对有向图中的后向边和横向边进行分析。

在深度优先遍历图的过程中，我们同时按照上述的思路定义并维护low和dfn两个数组，在深度优先遍历结束后，会发现在图中，若图中存在环路，则这个环路中所有顶点的low值均相等，所有low值相等的顶点及这些顶点间所连的便所构成的图就是有向图的强连通分量。其中对于low值与dfn值相等的顶点，以该顶点为根的搜索子树上的所有顶点便是一个强连通分量。在深度优先搜索时，可以把当前搜索树中未处理的顶点加入一个栈中，在深度优先搜索进行回退时，就可以判断栈顶到栈中的顶点是否为强连通分量，具体做法为，在深度优先搜索回退时，若查找到的顶点的low值与dfn值相同时，依次退出顶点栈当中的顶点，直到退出的顶点单元的low值与dfn值相同为止，此次退出的所有元素所构成的生成子图即为有向图的一个强连通分量。按照上述方式，就可以将有向图的所有强连通分量找出。

算法流程的简易示意图如图4所示。

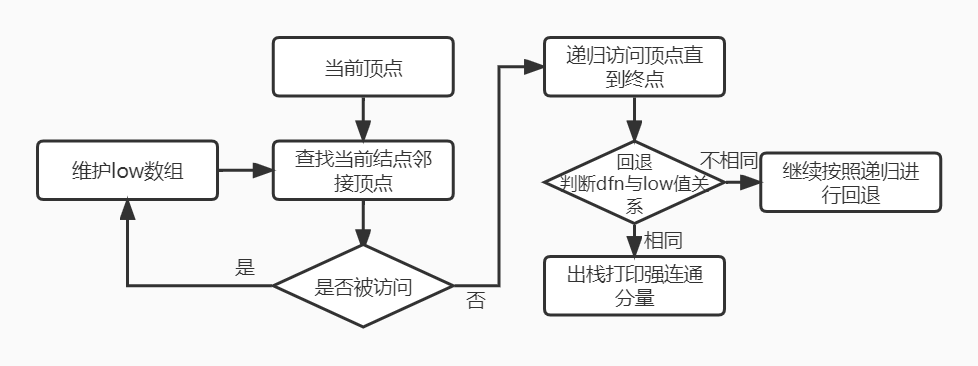


图4

接下来举例对算法流程进行演示，以图5中的有向图为例进行演示。

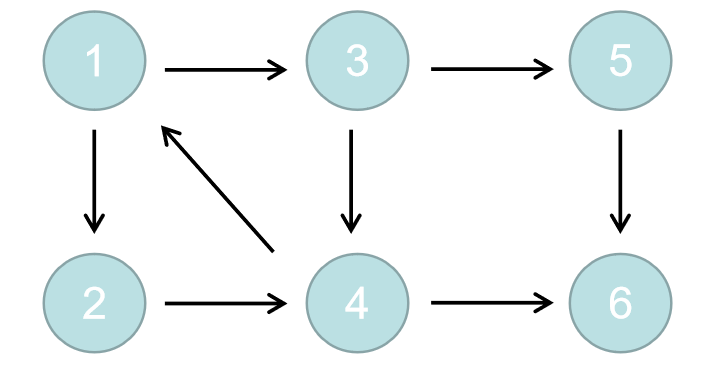


图5

第一步，如图6所示，从顶点1开始进行深度优先遍历，并将所遍历到的顶点压入栈中。搜索过程为1->3->5->6，搜索到6时，无法继续向下搜索，开始回退，更新顶点6的low值后发现dfn[6]=low[6]，表明发现了一个强连通分量，退栈退至顶点6出栈为止。故{6}为一个强连通分量。

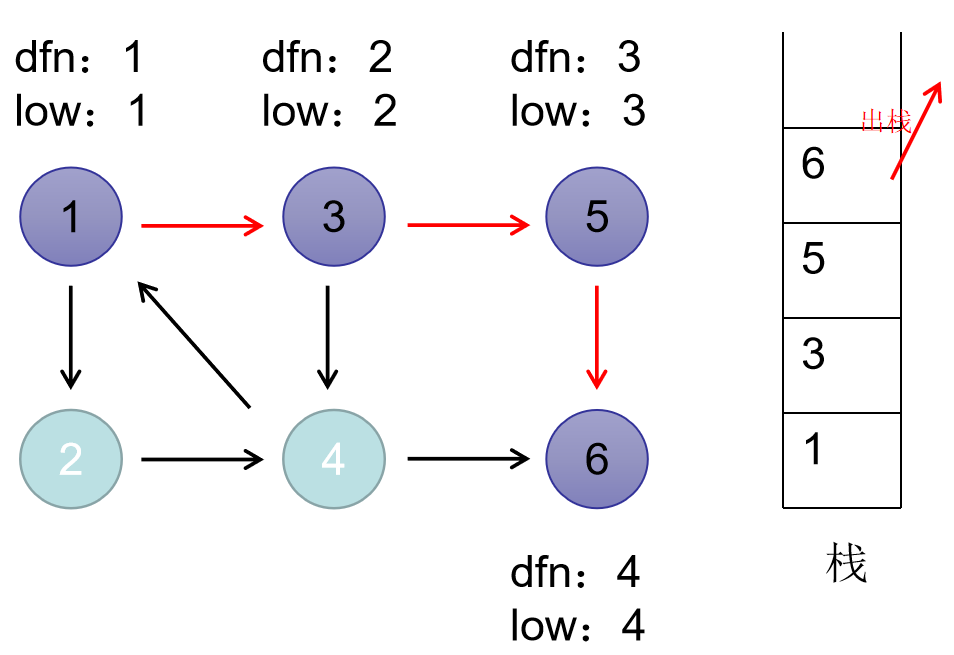


图6

第二步，退回到顶点5，顶点5的邻接顶点均被搜索过，无法继续向下搜索，更新顶点5的low值，发现dfn[5]=low[5]，表明发现了一个强连通分量，退栈退至顶点5出栈为止。故{5}为一个强连通分量。如图7所示。

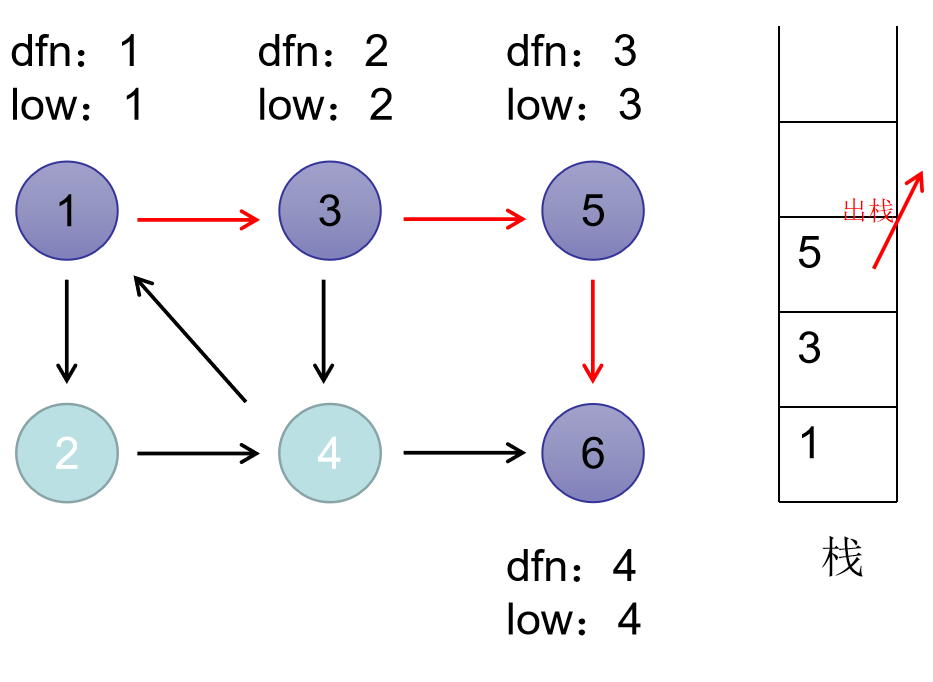


图7

第三步，返回顶点3，更新顶点3的low值并继续向下搜索至顶点4，把顶点4加入栈中。继续向下搜索，搜索到顶点1，顶点1已经被搜索过，且顶点1还存在在栈中，表明边4->1为后向边，更新顶点4的low值为顶点1的low值1。搜索到顶点6，顶点6已经被搜索过，但顶点6不存在在栈中，表明边4->6为横向边，对强连通分量无影响，忽略。具体如图8所示。

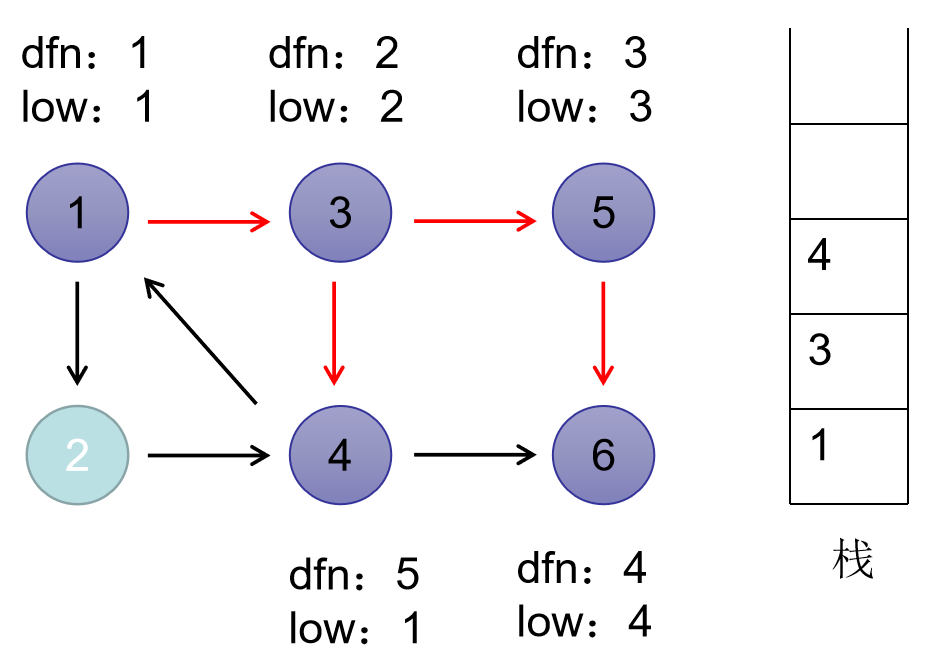


图8

第四步，顶点4无法继续向下搜索，开始回退到顶点3，再回退到顶点1，过程中更新3的low值为1.回退到1后，继续向下搜索到顶点2，将顶点2压入栈中。顶点2继续搜索发现后向边2->4，更新顶点2的low值为顶点4的low值1。如图9所示。

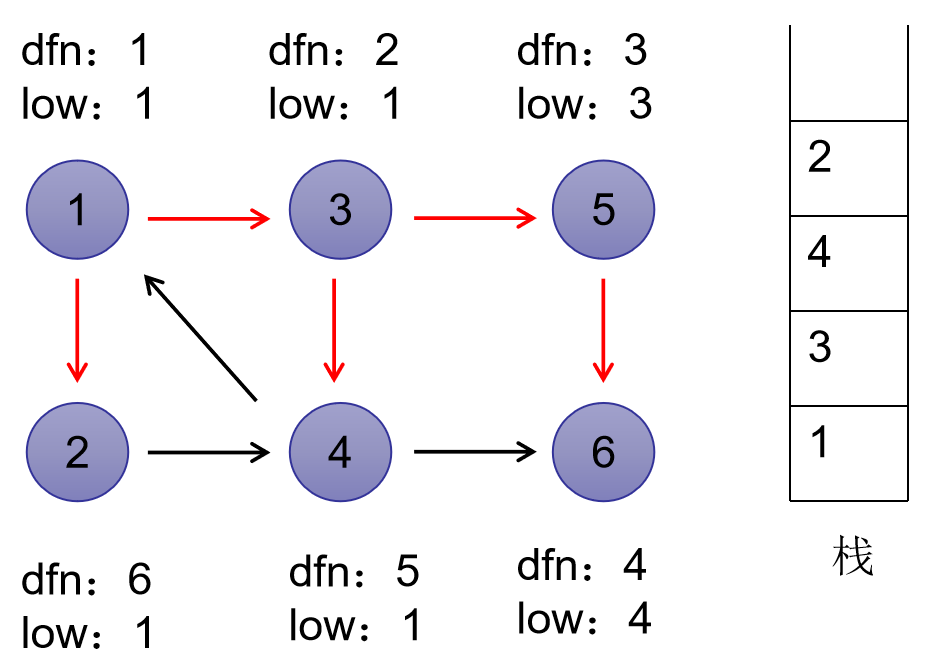


图9

最后，回退到顶点1，发现顶点1的low值与dfn值相同，不断退出栈中的顶点，直到退出的顶点为1为止。本次退出的元素组成一个强连通分量{2，4，3，1}。算法计算结束，综上运行结果，图4中图的强连通分量有3个，分别为{6}，{5}，{2，4，3，1}.

以上便是Tarjan算法的实现流程。

**3.5 伪代码实现Tarjan算法**

针对Tarjan算法的实现流程，Tarjan算法的伪代码实现如下：

|  |
| --- |
| tarjan(u)  {      DFN[u]=Low[u]=++Index；//为节点u设定次序编号和Low初值      Stack. push(u)；//将节点u压入栈中  **for** （each(u, v) in E）//枚举每一条边  **if** (v is not visited)//如果节点v未被访问过              tarjan (v)；//继续向下找              Low[u]=min(Low[u],Low[v])；  **else** **if** (v in S)//如果节点v还在栈内                  Low[u]=min(Low[u],DFN[v])；  **if** (DFN[u]==Low[u])//如果节点u是强连通分量的根      do{          v=Stack. Pop；//将v退栈，为该强连通分量中一个顶点          print v；      }  while(u！=v)  } |

**3.6 Tarjan算法分析**

分析Tarjan算法的实现过程，算法对每条边进行了一次访问，对每个顶点进行了一次访问，故算法的时间复杂度为O(v + e)，其中v表示有向图中的顶点个数，e表示有向图中边的条数，该算法求解图的强连通分量的方法中较为高效的实现方法。但Tarjan算法存在了一个不足之处，深度优先搜索的过程中，每次向下搜索或者回退均要对辅助数组low进行维护，修改对应顶点的low值，对low值的修改相对频繁，具有很大效率上的浪费，在这一点上存在值得优化的地方。

**4 Garbow算法**

**4.1 算法简介**

Garbow算法的基本思想与Tarjan算法的基本思想相同，算法的原理及算法基础也与Tarjan算法相同，通常是被认为是Tarjan算法的另一种实现思路。但实现方法上Garbow算法对Tarjan算法进行了优化和改进，思路上也更为精妙，被广泛的应用于强连通分量的实际求解过程中。

**4.2 算法实现**

算法实现上，于Tarjan算法里面用维护dfn数组和low数组的方式来求解出强连通分量在深度优先搜索树的根不同，Garbow算法是在编程时利用一个辅助栈来求出强连通分量的根，强连通分量的根求解出来以后，强连通分量也随之确定。

在Tarjan算法中，每次low值的修改都是由于在深度优先遍历中发现了反向边，即在有向图中构成了环，否则low值不可能变小。遍历图的过程中每次出现反向边，那么在所对应环当中只剩下一个顶点的low值没有被改变。Garbow算法中，需要对两个栈进行处理，其中第一个栈与Tarjan算法中的栈作用相同，按照深度优先的图遍历顺序，依次将图中的顶点压入栈中，用该栈来按照深度优先搜索顺序存储图中顶点。另一个栈的变化就是用来删除构成环的顶点，只剩下深度最低的顶点。删除的过程利用出栈来实现。因为深度最低的顶点一定比前面的先访问，那么只要出栈一直到栈顶那个顶点的访问时间不大于深度最低的顶点。其中每个被弹出的结点属于同一个强连通分量。Tarjan算法中的判断条件用第二个栈弹出的顶点是否为当前顶点来替代。

具体实现流程为：

1. 首先定义一个两个堆Stack1和Stack2。Stack1用以按照深度优先搜索顺序存储图中顶点，Stack2起到删除构成环的顶点的作用。
2. 深度遍历开始搜索，找到一个没有被访问过的顶点v，若找不到，算法结束。
3. 将顶点v压入栈Stack1及栈Stack2中，对于v的所有临界顶点u。若u没有被访问过，则转步骤②；如果u被访问过，则维护栈Stack2。最后回退到顶点v时，如果Stack2的栈顶顶点等于v，则输出相应的强连通分量。

算法简单示意图如图10所示。

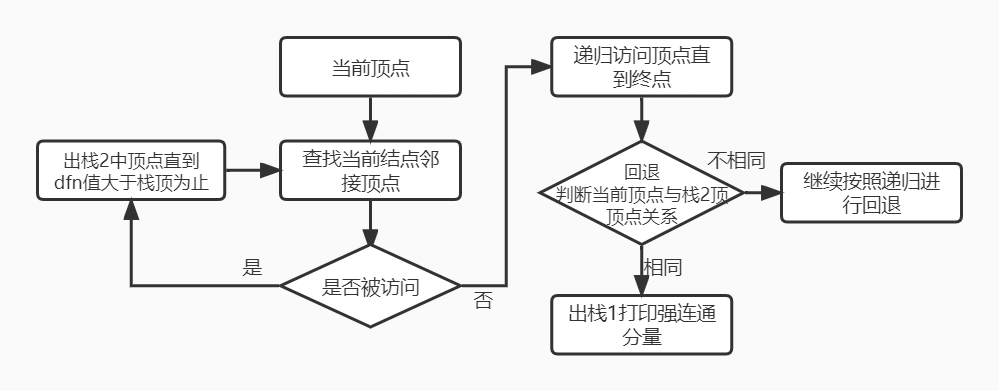


图10

接下来举例对算法流程进行演示，仍以图5中的有向图为例进行演示。

第一步，如图11所示，从顶点1开始进行深度优先遍历，并将所遍历到的顶点压入栈Stack1和Stack2中。搜索过程为1->3->5->6，搜索到6时，无法继续向下搜索，开始回退。回退到顶点6时发现当前顶点6与栈Stack2中栈顶元素相同，退出栈Stack1的栈顶元素，直到退出的栈顶元素同Stack2中栈顶元素相同，退出的顶点为{6}，顶点6为图中的一个强连通分量，退出栈Stack2中的栈顶元素。

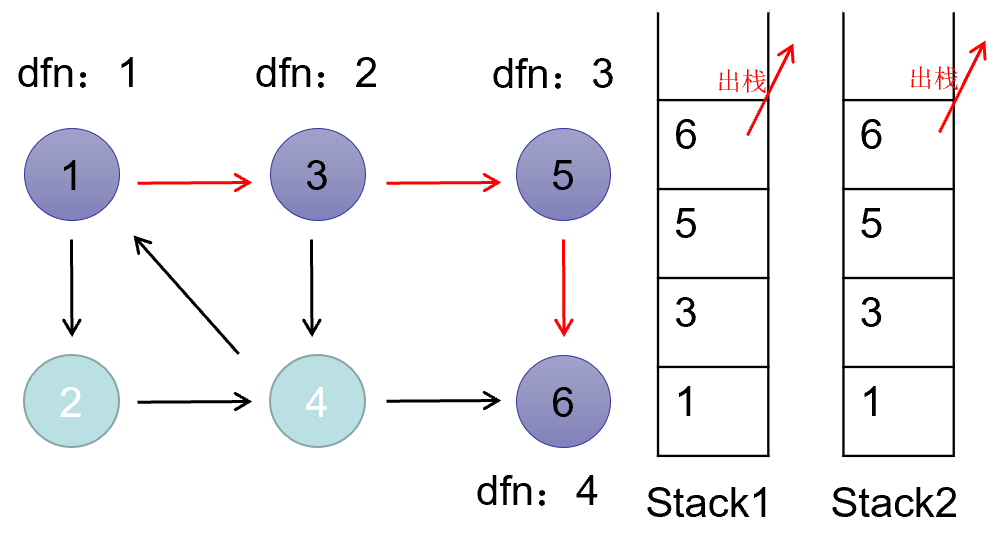


图11

第二步，退回到顶点5，顶点5的邻接顶点均被搜索过，无法继续向下搜索，当前顶点5与栈Stack2中栈顶元素相同，退出栈Stack1的栈顶元素，直到退出的栈顶元素同Stack2中栈顶元素相同，退出的顶点为{5}表明发现了一个强连通分量，退出栈Stack2中的栈顶元素。如图12所示。

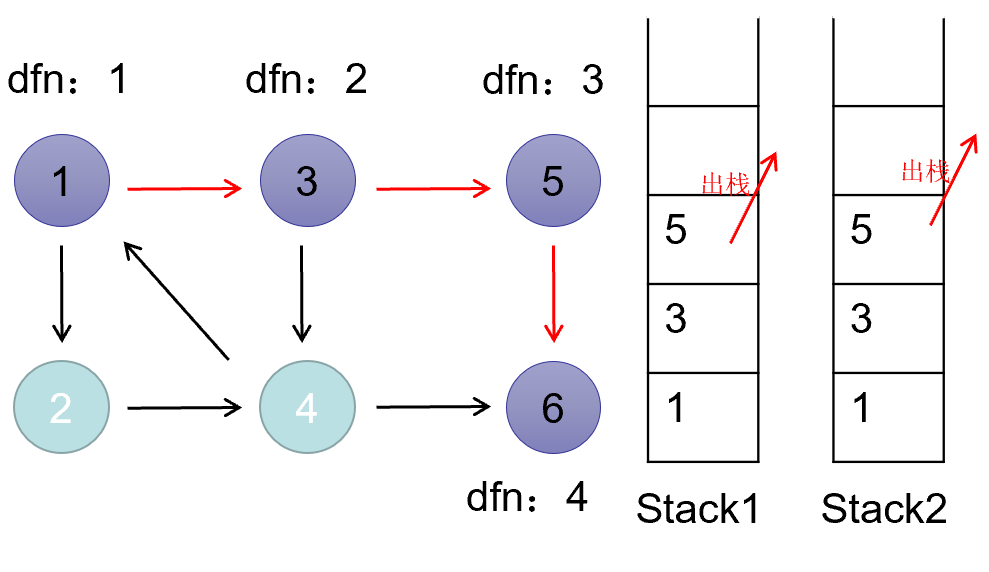


图12

第三步，返回顶点3，继续向下搜索至顶点4，把顶点4加入栈中。如图13所示。

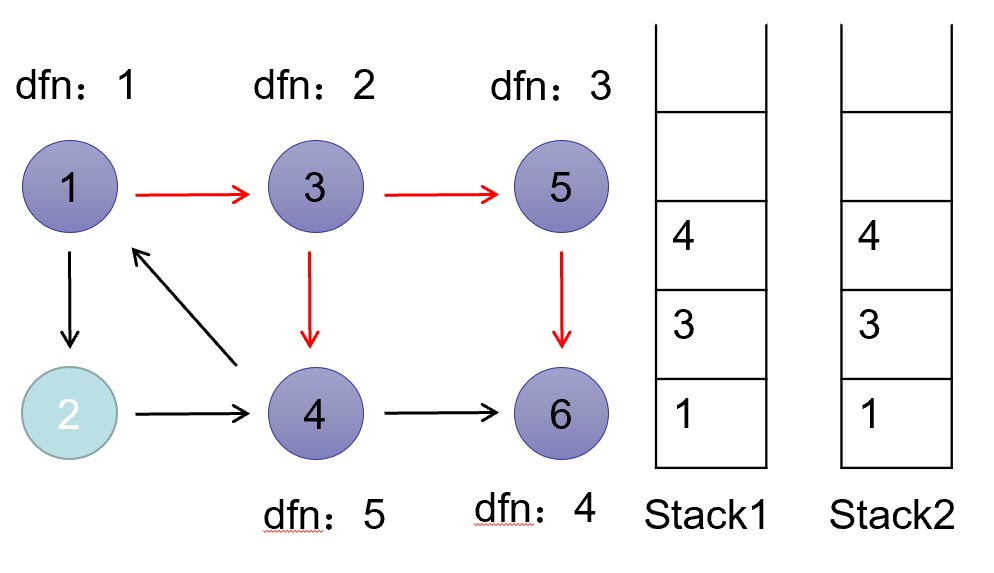


图13

第四步，继续向下搜索，搜索到顶点1，顶点1已经被搜索过，且顶点1的dfn值比顶点4小，表明边4->1为后向边，Stack2退栈直到栈顶的dfn值小于或等于结点1的dfn值为止。及将后向边所生成环的顶点删除到仅剩一个顶点。接着由顶点4回退到顶点3，再回退到顶点1，回退过程中的当前顶点与Stack2栈顶顶点均不相等，不操作。如图14所示。

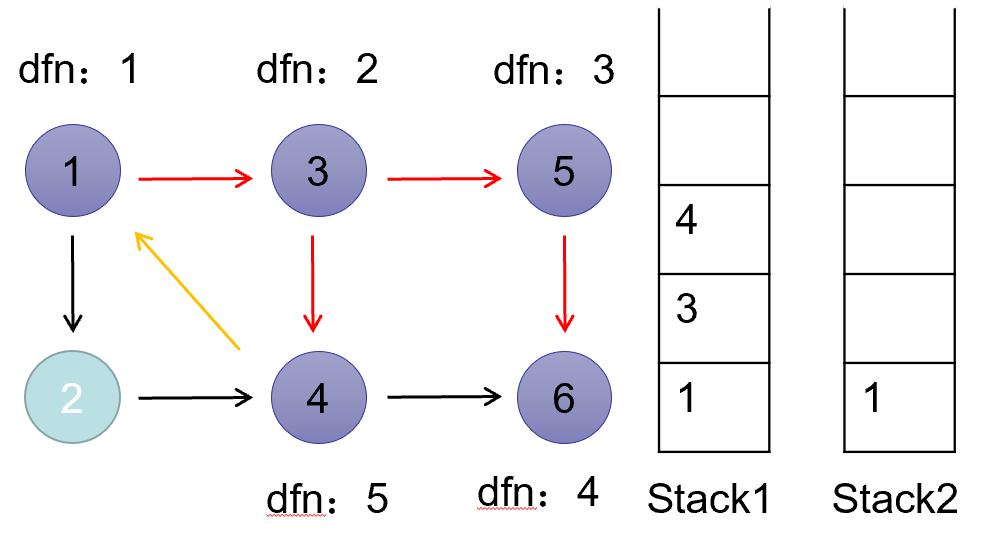
****

图14

第五步，由顶点1向下搜索，搜索至顶点2，将顶点2加入两个栈中。顶点2继续向下搜索，搜索至顶点4，发现顶点4的dfn值比2小，表明边2->4为后向边，Stack2退栈直到栈顶的dfn值小于或等于结点4的dfn值为止。如图15所示。

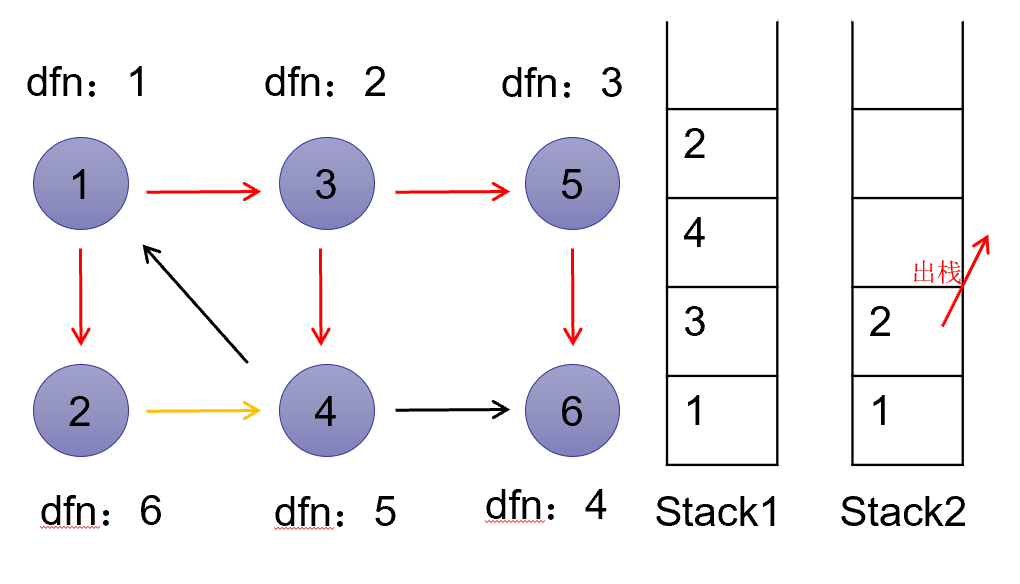


图15

最后，回退至顶点1，发现顶点1与栈Stack2中的栈顶顶点相同，依次退出栈Stack1中元素，直到退出元素与栈Stack2中元素相同为止，退出的顶点为{2，4，3，1}，此为一个强连通分量。综上运行结果，图4中图的强连通分量有3个，分别为{6}，{5}，{2，4，3，1}。

**4.3 Garbow算法伪代码实现**

针对Garbow算法的实现流程，Garbow算法的伪代码实现如下：

|  |
| --- |
| Garbow(u)  {      DFN[u]=++Index；//为节点u设定次序编号      Stack1.push(u)；Stack12.push(u)；//将节点u压入两栈中  **for** （each(u, v) in E）//枚举每一条边  **if** (v is not visited)//如果节点v未被访问过              Garbow (v)；//继续向下找  **else** **if**(DFN[u]>= DFN[v])//节点v比结点u更早被访问                   while(DFN [Stack2.top()] > DFN[v])                        Stack2.pop();//不断出栈直到Stack2 的DFN值小于或等于v的DFN值为止  **if** (u== Stack2.top())//如果节点u与Stack2中栈顶相同      do{          v=Stack2.pop；//将v退栈，为该强连通分量中一个顶点          print v；      }  while(DFN[v]>= DFN[u]) //直到栈顶元素DFN值小于等于u为止  } |

**4.4 Garbow算法分析**

用v表示有向图中的顶点个数，用e表示有向图中边的条数，分析算法Garbow算法的实现过程，算法对每条边进行了一次访问，对每个顶点进行了一次访问，故算法的时间复杂度为O(v + e)。在时间复杂度上，Garbow算法与Tarjan算法相同，但Garbow算法以维护一个新栈的方式取代了Tarjan算法中low数组，避免了对low数组的频繁修改，实现方法上相较于Tarjan算法更为精妙，时间复杂度上常数相对Tarjan算法较小，执行效率上比Tarjan算法高，是Tarjan算法的一种迭代优化，被更广泛地运用于实际问题的解决中。

**5 算法调研总结**

在图论中，强连通分量对外可以收缩为一个点进行分析，求解出强连通分量后可以大大降低图论问题的复杂度，具有很大的实际意义。本文重点调研分析了Tarjan算法与Garbow算法，这两种算法是求解图论中有向图强连通分量中效率较高、最为常用的两种方法，两种算法的实现思路大致相同，其中Garbow算法的效率略优于Tarjan算法，两种算法均被广泛运用在实际问题的解决当中。

**参考文献**

1. 殷人昆.数据结构[M].北京:清华大学出版社,2007
2. 付海奎. 基于Tarjan算法的极大点连通子图研究. 10.14004/j.cnki.ckt.2021.2189
3. 廖小飞. 一种高效的面向动态有向图的增量强连通分量算法. 中国科学:信息科学. 2019,49(08)