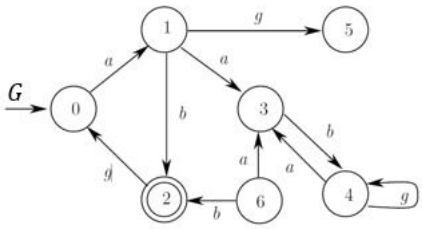
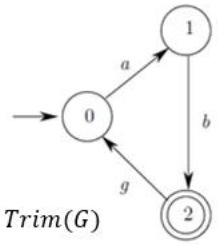
题目：

1. 编写一个算法实现 TRIM 操作。

2，运行上图实例，得到正确结果。

Trim 操作： $Trim(G) \coloneqq CoAc\big(Ac(G)\big) = Ac\big(CoAc(G)\big)$





1. 程序算法

条件：1.用有向连通图的形式来表示自动机，输入起点状态，结果状态，转移事件；

2.邻接方阵来表示有向连通图，1 表示连通，0 表示没有连通，如矩阵第一行第三列为 1，那么表示状态 0 可以到达状态 2。

1. 算法

（1）由输入的有向边、初始状态节点和标记状态节点创建一个自动机，确定邻接矩阵；同时，找出所有由初始状态节点到达标记状态节点的简单（不包含有回路）路径（深度优先遍历），如 0->1->2；

（2）寻找回路，因为最后需要保存的回路必定与初始状态节点到标记状态节点的某条简单路径相接或间接相接，因此，对在简单路径上的所有状态节点依次寻找回路（深度优先遍历），首先将任一在简单路径上的状态节点作为起点，再由起点可达的下一节点开始，寻找一条可以到达起点的简单路径，由此形成回路；同时对寻找到的回路上所有新增的状态节点入栈，继续依次进行回路查找，直到栈内的所有状态节点均查找完毕。

（3）查找完毕，保存新的邻接矩阵，根据矩阵元素是否为 1 输出状态和事件，程序结束。

2. 深度优先遍历算法

以本题为例：

（1）从初始状态节点 0 开始出发，状态 0 被标记访问过，并入栈，到状态 1，标记并入栈；

（2）状态 1 到状态 2，此时状态 2 是终点，到达函数开始的判断条件，输出堆栈经过的路径。一条路找到 0->1->2。此时虽然状态 2 可以到状态 0，但也没有必要探寻下去：算法的角度来说如果继续探寻下去状态 2 就会被标记，就算有路可以到，下次就不会背进入递归了；

（3）状态 2 回溯到状态 1，状态 1 可以走状态 5，入栈并标记，没有可走的路了，堆栈中的状态 5 出栈，并取消标记，回溯到状态 1，状态 1 可以走状态 3……同理；

（4）状态 1 回溯到状态 0，没路可走了，程序结束。

2. 算法实现

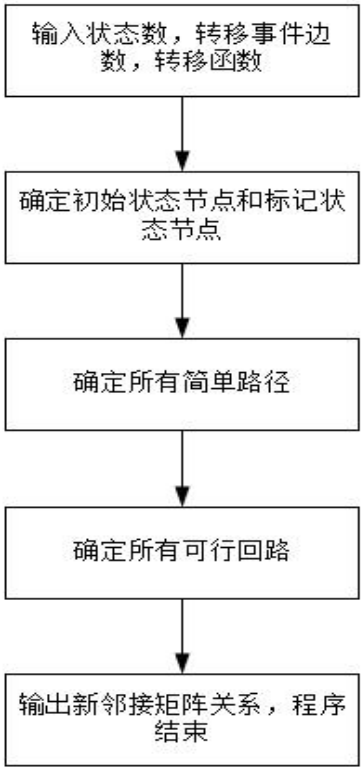
1. 源程序：

```
#include<stdio.h>
#include<math.h>
int map[100][100]={0};
char event[100][100];
int trimap[100][100]={0};
int stack[120],pathstack[120],v[100]={0},top=0,m,n,start,end,l=0;
void dfs(int b,int c)//深度优先遍历
{
    int i,p;
    if(b==end){
        if(c==1)
        {
            trimap[end][stack[0]]=1;
        }
        trimap[end][stack[0]]=1;
        trimap[stack[top-1]][end]=1;
        for(i=0;i<top;i++)
        {
            if(i<top-1)
            {
                trimap[stack[i]][stack[i+1]]=1;
            }
            int q=0;
            for(p=0;p<=l;p++)
            {
```

```
                    if(pathstack[p]==stack[i])
                    {
                        q=1;
                    }
                }
                if(q!=1)
                {
                    pathstack[l]=stack[i];
                    l++;
                }
                printf("%d ",stack[i]);
            }
            printf("%d\n",end);
            return;
        }
        v[b]=1;
        stack[top++]=b;
        for(i=0;i<n;i++){
            if(!v[i]&&map[b][i])
                dfs(i,c);
        }
        v[b]=0;
        top--;
}
void circle(int a,int n)//查找环路
{
        int j;
        for(j=0;j<n;j++)
        {
            if(j==a&&map[a][a]==1)
            {
                trimap[a][a]=1;
                printf("%d %d\n",a,a);
            }
            else if(map[a][j]==1)
            {
                end=a;
                dfs(j,1);
            }
        }
}
int main()
{
        int i,j,x,y,z,b;
```

```
    char k1;
    printf("Please enter the number of nodes and edges:");//输入状态数，边数
    scanf("%d %d",&n,&m);
    printf("Please enter the event and the couples of nodes with directed edges:\n");//输入状态和
事件
    for(i=0; i<m; i++) {
        scanf("%d %d %c", &x,&y,&k1);
        map[x][y] = 1;
        event[x][y]=k1;
    }
    printf("Initial£¨adjacency matrix representation£©:\n");//输出邻接矩阵
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%d ",map[i][j]);
        printf("\n");
    }
    printf("Please enter the initial node and the marked node:");//输入初始状态和标记状态
    scanf("%d %d", &start,&end);
    printf("The paths of trim (without circles):\n");//输出简单路径
    dfs(start,0);
    printf("Circles on the paths: \n");//输出回路
    pathstack[l]=end;
    l++;
    for(b=0;b<=l;b++)
        circle(pathstack[b],n);
    printf("TRIM:\n");//最后结果
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            {
                if(trimap[i][j]==1)
                {
                    printf("%d----%c---->%d \n",i,event[i][j],j);
                }
            }
    }
    fflush(stdin);
    getchar();
    return 0;
}
```

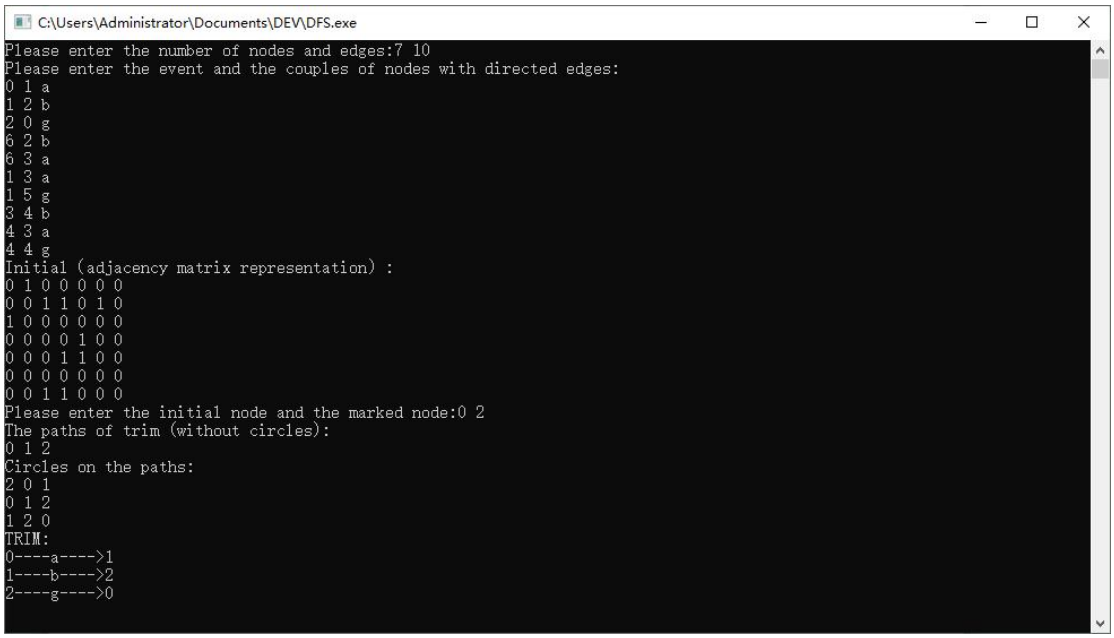
2.流程图：



3. 程序结果

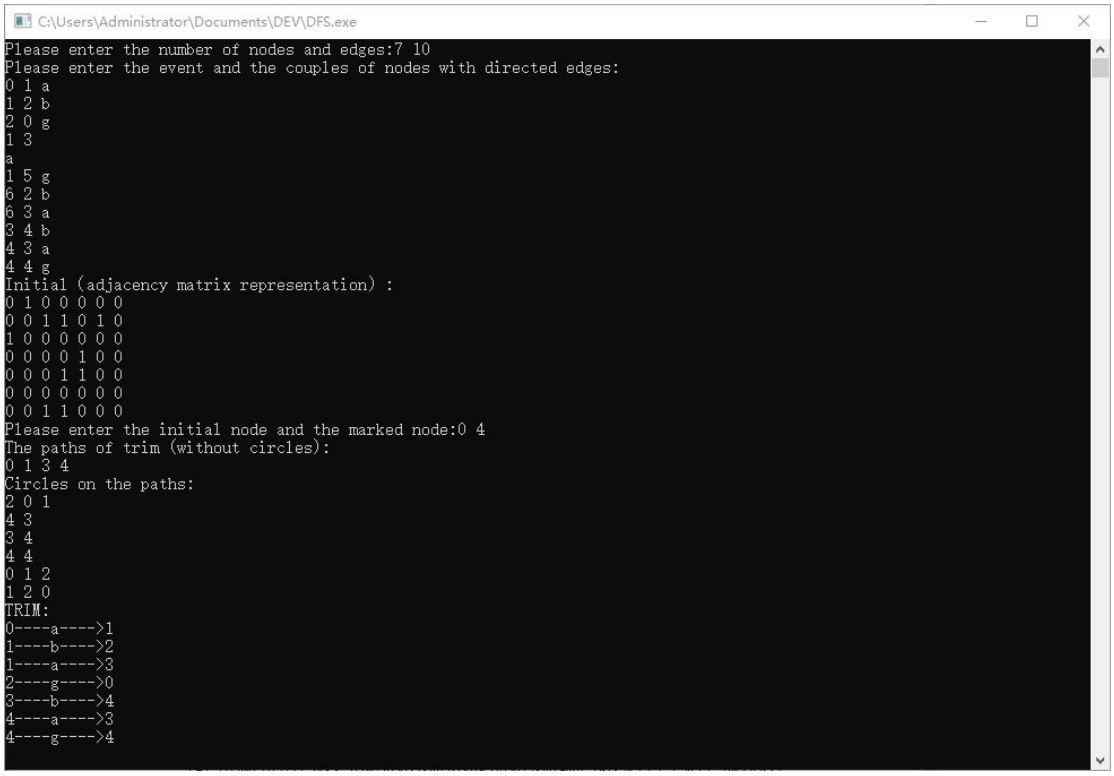
1.

起点	终点
0（初始状态）	1
1	5
1	3
1	2
2（标记状态）	0
3	4
4	3
4	4
6	3
6	2



2. 测试

起点	终点
0（初始状态）	1
1	5
1	3
1	2
2	0
3	4
4（标记状态）	3
4	4
6	3
6	2



4. 分析

优点：1.使用邻接矩阵的形式代替链表堆栈，简化了 C 程序；

2.没有用节点出度入度来搜寻回路，而用依次深度遍历来搜索，降低了程序复杂度。

缺点：1.查找回路时会出现重复回路，造成算法冗余。后续需要改进，如保存已出现过的回路；