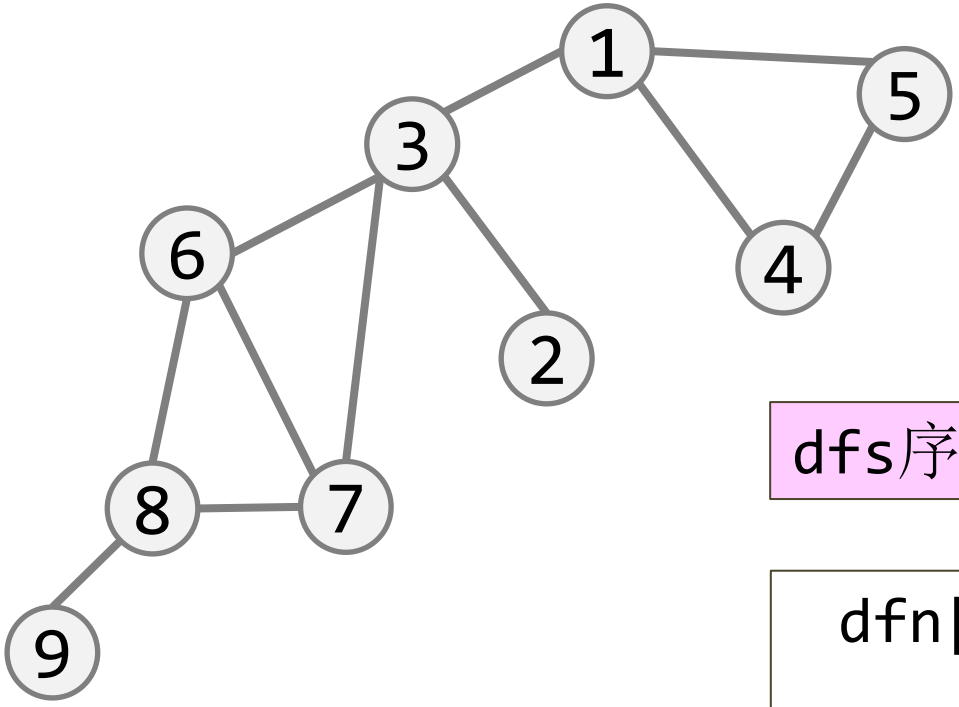
C++算法

搜索树



dfs遍历过程形成一颗搜索树

dfs遍历一张图

从1号开始遍历
请写出dfs访问序列
多个邻居里
选编号小的先访问

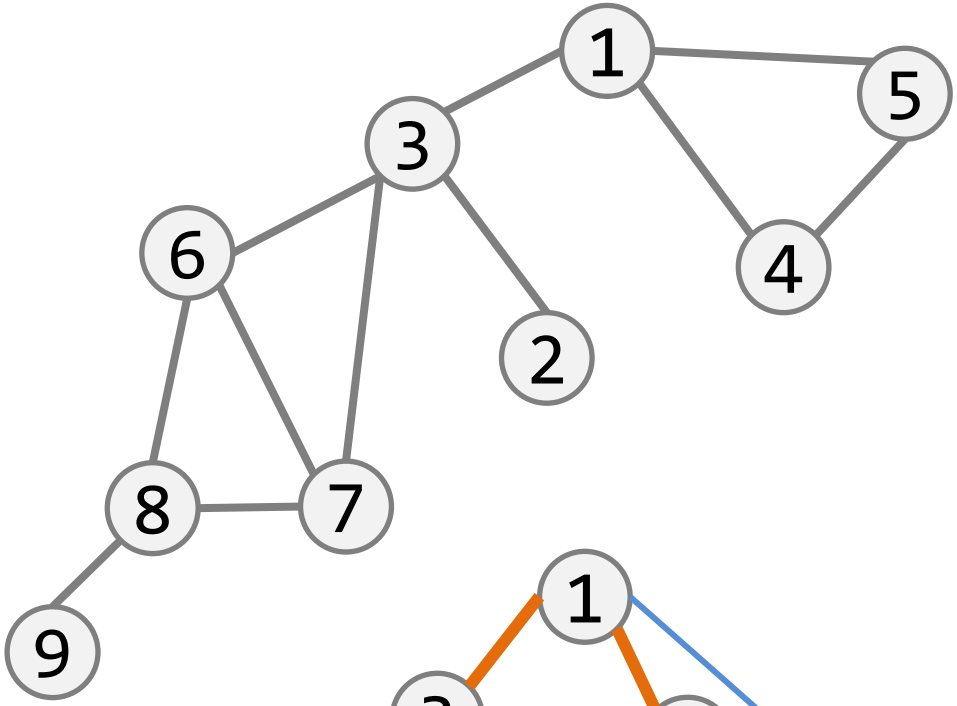
123456789

dfs序列	132678945

dfn[u]表示节点u在dfs序列中第几个访问

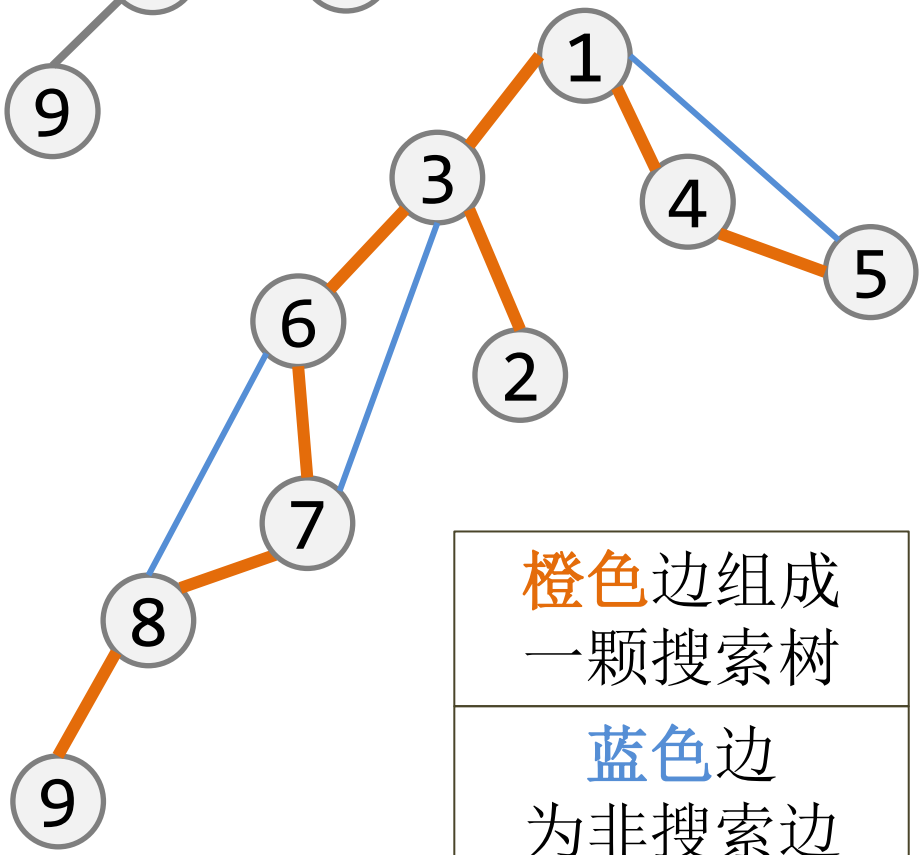
dfn[]	132894567

dfn：dfs序号
对原图节点重新编号



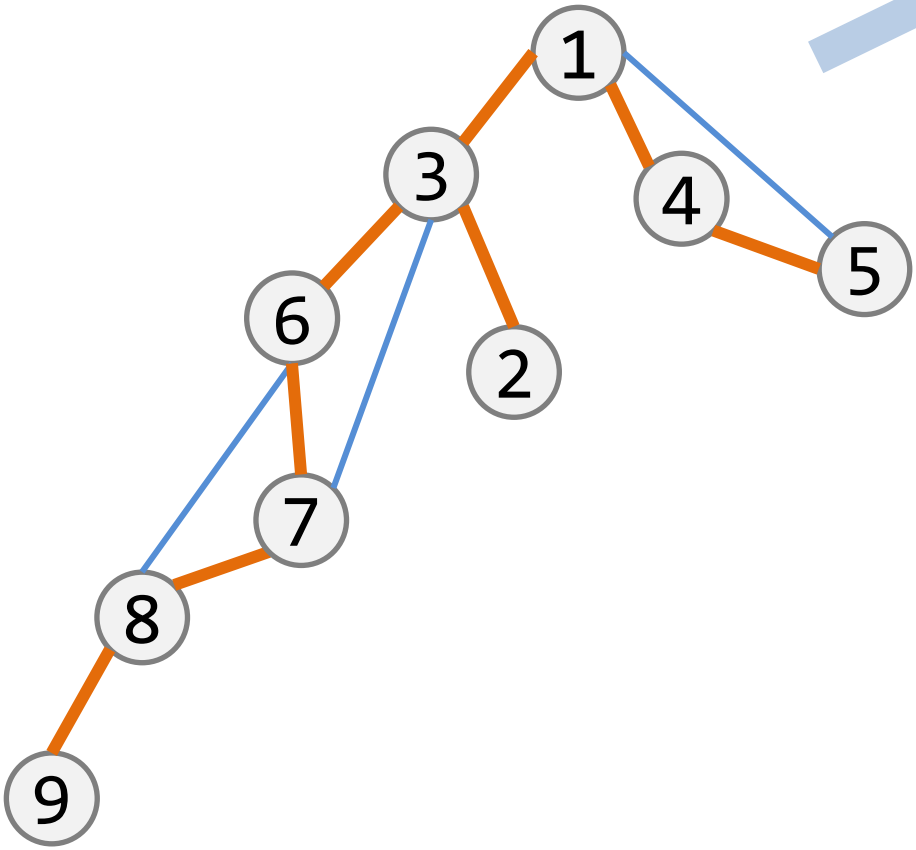
非搜索边有2种被访问情况	
找到祖先	找到后代

橙色全部是搜索树里的父子边
蓝色全部是搜索树里的纵向边: 祖先和后代的连边
没有横向边! 请思考为什么

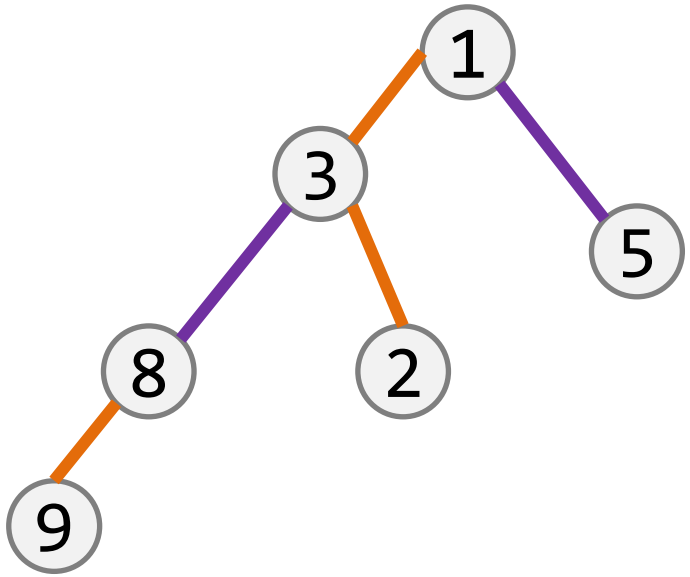


橙色边组成 一颗搜索树
蓝色边 为非搜索边

路径
压缩

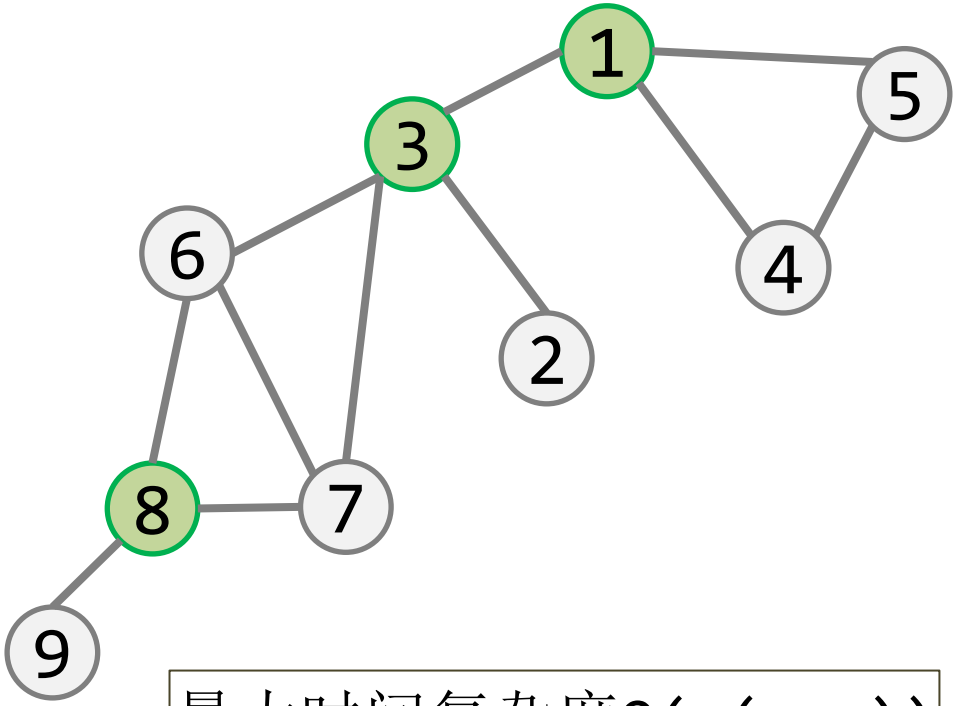


缩点缩边后
形成简化版的
树结构



割点

cut point



若删除节点u
就会使无向连通图
变成不连通
则u为原图的割点

为什么6不是割点
为什么2不是割点

暴力时间复杂度O(n(n+m))

如何暴力求解割点?

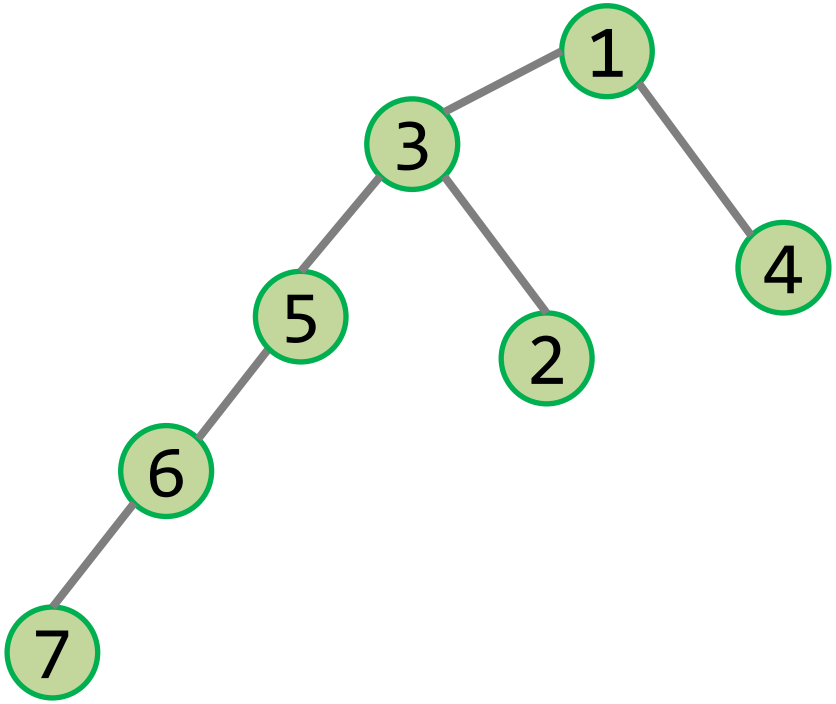
如何加速?

暴力为什么慢?

例:删除3678这块外的点
不会影响块内连通性

图简化为树

启发灵感



树上所有点都是割点

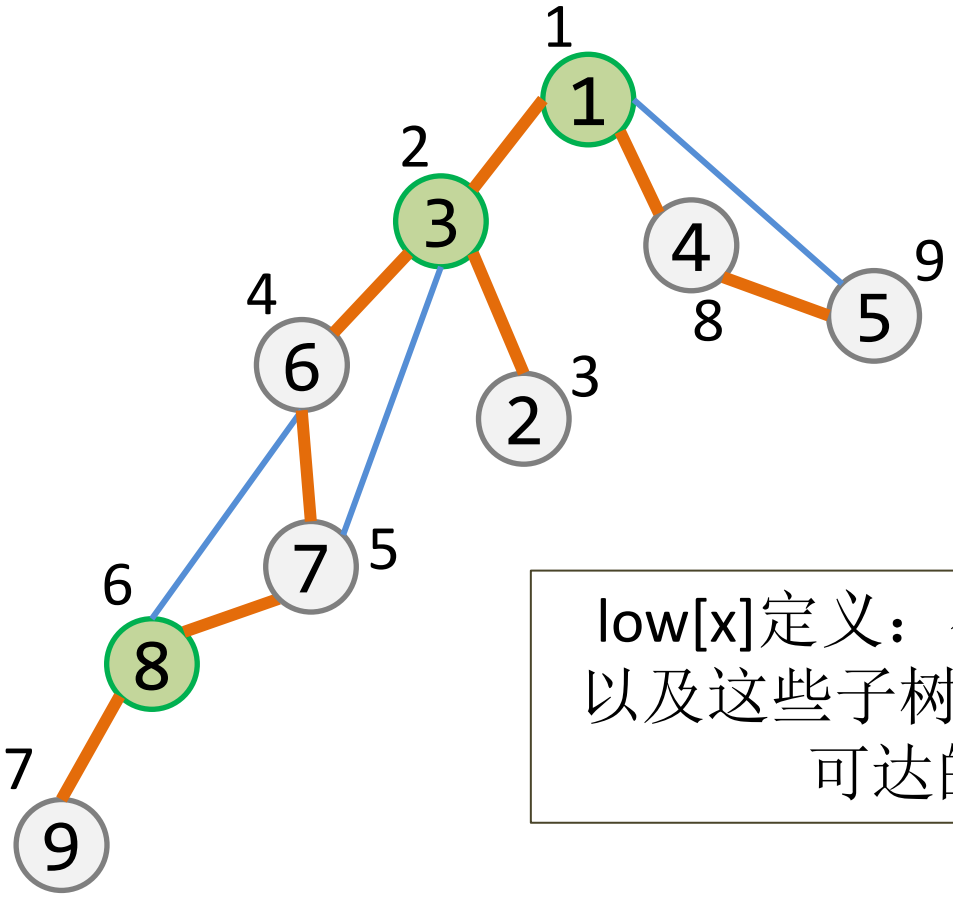
割点条件

u是割点的条件

u存在某个儿子v
满足low[v]>=dfn[u]

搜索树里u所有子孙在
原图里参与的非搜索边
都无法直接到达u的祖先

特判根和叶



low[x]定义：在搜索树中x为根的子树节点，
以及这些子树节点开始最多通过1条非搜索边
可达的节点里的最小dfn编号

low[x]的计算
是从上往下
还是从上往下?

123456789

dfn[]	132894567
low[]	132112247

快快编程812

割点条件 | Tarjan算法

对于给定的无向无权连通图

求出原图对应的搜索树
每个节点在搜索树中的儿子列表

不同于原图的邻接表

求出每个节点u的dfn[u]
求出每个节点u的low[u]

对每个节点u判断是否为割点
注意特判:根和叶

```
38      scanf("%lld %lld",&n,&m);
39      for(ll i=1;i<=m;++i){
40          ll u,v;
41          scanf("%lld %lld",&u,&v);
42          to[u].push_back(v);
43          to[v].push_back(u);
44      }
45      tarjan(rt=1,0);
46      ll nCut=0;
47      for(ll u=1;u<=n;++u)if(isCut(u)){
48          ++nCut;
49          printf("%lld\n",u);
50      }
51      if(!nCut) printf("perfect\n");
```

遍历后再判断割点

遍历后再判断割点

```
25 bool isCut(ll u){
26     if(u==rt)return nSon[rt]>1;
27     if(nSon[u]==0)return 0;
28     for(ll i=0;i<son[u].size();++i){
29         ll v=son[u][i];
30         if(low[v]>=dfn[u])return 1;
31     }
32     return 0;
33 }
```

割点条件 | Tarjan算法

原图邻接表
搜索树儿子列表

```
 9 void tarjan(ll u,ll fa){
10     dfn[u]=low[u]=++timer;
11     for(ll i=0;i<to[u].size();++i){
12         ll v=to[u][i];
13         if(v==fa)continue;
14         if(dfn[v]){
15             low[u]=min(low[u],          );
16             continue;
17         }
18         tarjan(v,u);
19         low[u]=min(low[u],          );
20         son[u].push_back(v);
21         ++nSon[u];
22     }
23 }
```

13,14行过滤语句
能否交换顺序

遍历时直接判断割点

割点条件 | Tarjan算法

邻接表

遍历时直接判断割点

```
 9 void tarjan(ll u,ll fa){
10     dfn[u]=low[u]=++timer;
11     for(ll i=0;i<to[u].size();++i){
12         ll v=to[u][i];
13         if(v==fa)continue;
14         if(dfn[v]){low[u]=min(low[u],dfn[v]);continue;}
15         tarjan(v,u);
16         low[u]=min(low[u],low[v]);
17         ++nSon[u];
18         if(low[v]<dfn[u])continue;
19         if(u!=rt)cut[u]=1;
20         else cut[rt]=(nSon[rt]>1);
21     }
22 }
```

搜索树在遍历时隐式出现
没有显式储存

另类代码

链式前向星
储存所有边

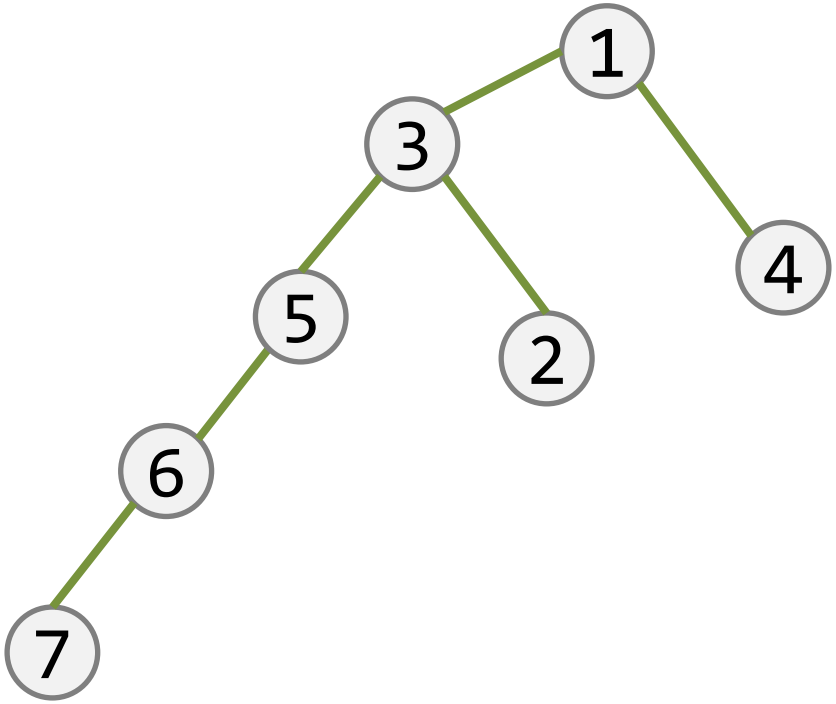
```
 6 struct edge{ll to,nxt;} e[M*2];
 7 ll n,m,rt,nE,hd[N];
 8 ll dfn[N],low[N],nSon[N],timer;
 9 bool cut[N];
10 void add(ll u,ll v){
11     ++nE;
12     e[nE]=(edge){v,hd[u]};
13     hd[u]=nE;
14 }
```

```
15 void tarjan(ll u,ll fa){
16     dfn[u]=low[u]=++timer;
17     for(ll i=hd[u];i;i=e[i].nxt){
18         ll v=e[i].to;
19         if(v==fa)continue;
20         if(dfn[v]){low[u]=min(low[u],dfn[v]);continue;}
21         tarjan(v,u);
22         low[u]=min(low[u],low[v]);
23         ++nSon[u];
24         if(low[v]<dfn[u])continue;
25         cut[u]=1;
26         cut[rt]=(nSon[rt]>1);
27     }
28 }
```

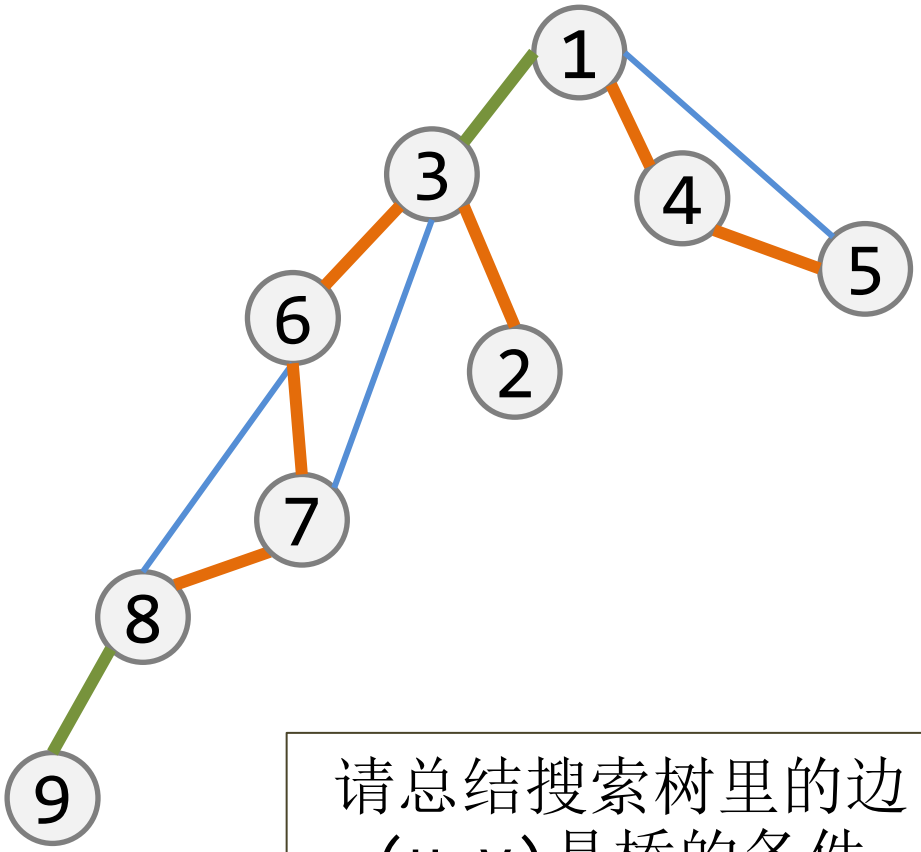
桥

bridge

若删除边e
就会使无向连通图
变成不连通
则e为原图的桥



树上所有边都是桥



对于图的问题
用搜索树为基础去思考

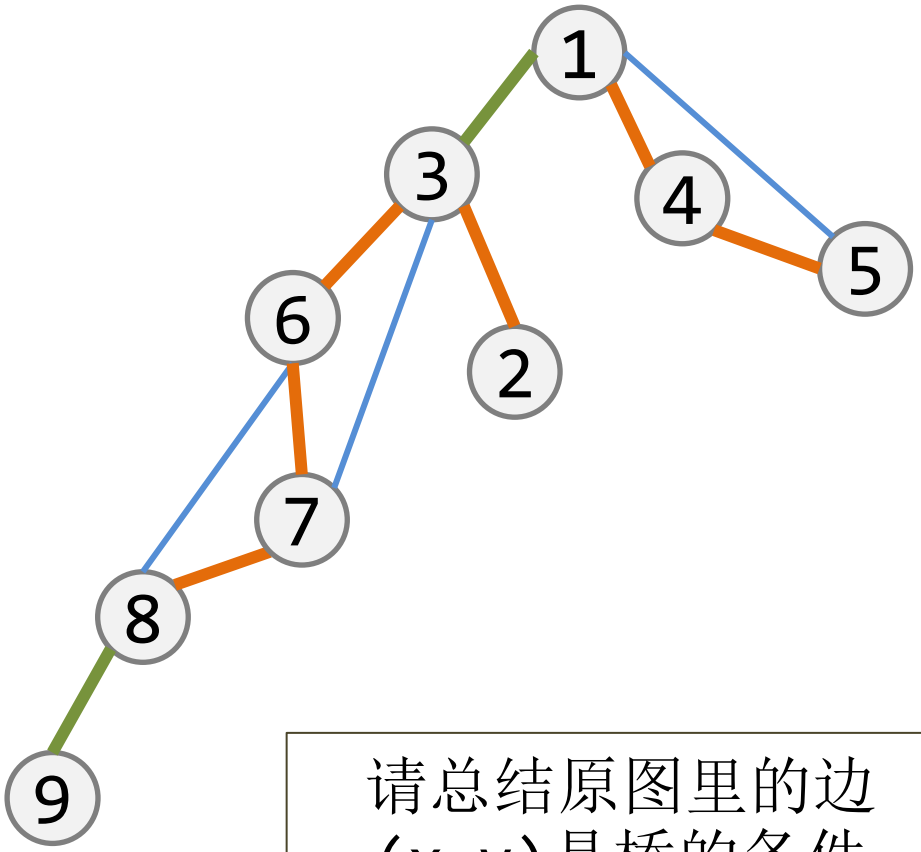
桥一定都在搜索树上
非搜索树边一定不是桥

请总结搜索树里的边
(u,v)是桥的条件

假设搜索树里
u是v父节点

low[v]>dfn[u]

另外dfn[v]>low[u]



对于图的问题
用搜索树为基础去思考

桥一定都在搜索树上
非搜索树边一定不是桥

请总结原图里的边
(x,y)是桥的条件

假设原图里
x和y的关系未知

```
low[x]>dfn[y]
or
low[y]>dfn[x]
```

快快编程828

	基于点的双连通分量	没有割
VDCC	Vertex Double Connected Component	

	基于边的双连通分量	没有桥
EDCC	Edge Double Connected Component	

原图无向边储存为：
2号3号边,4号5号边,……

i号边和i^1号边
描述的是原图同一条无向边

```
40      nE=1;
41      for(ll i=1;i<=m;++i){
42          ll u,v;
43          scanf("%lld %lld",&u,&v);
44          add(u,v);
45          add(v,u);
46      }
47      tarjan(rt=1,0);
48      for(ll u=1;u<=n;++u)if(!vst[u]){
49          ++nEDCC;
50          dfs(u);
51      }
52      printf("%lld\n",nEDCC);
```

```
27 void dfs(ll u){
28     vst[u]=nEDCC;
29     for(ll i=hd[u];i;i=e[i].nxt){
30         ll v=e[i].to;
31         if(vst[v])continue;
32         if(bridge[i])continue;
33         dfs(v);
34     }
35 }
```

vst[u]除了记录u是否被访问过
还记录u属于几号连通块
(基于边的双连通分量)

无向图的桥 | Tarjan算法

前向星

i号边和i^1号边
描述的是原图同一条无向边

```
15 void tarjan(ll u,ll fa){
16     dfn[u]=low[u]=++timer;
17     for(ll i=hd[u];i;i=e[i].nxt){
18         ll v=e[i].to;
19         if(v==fa)continue;
20         if(dfn[v]){low[u]=min(low[u],dfn[v]);continue;}
21         tarjan(v,u);
22         low[u]=min(low[u],low[v]);
23         if(low[v]<=dfn[u])continue;
24         bridge[i]=bridge[i^1]=1;
25     }
26 }
```

找到桥时设定无向边对应的
2条储存的边

边(u,v)是桥的条件	low[v]>dfn[u]

快快编程作业

	812
	813
	828
拓展题	811