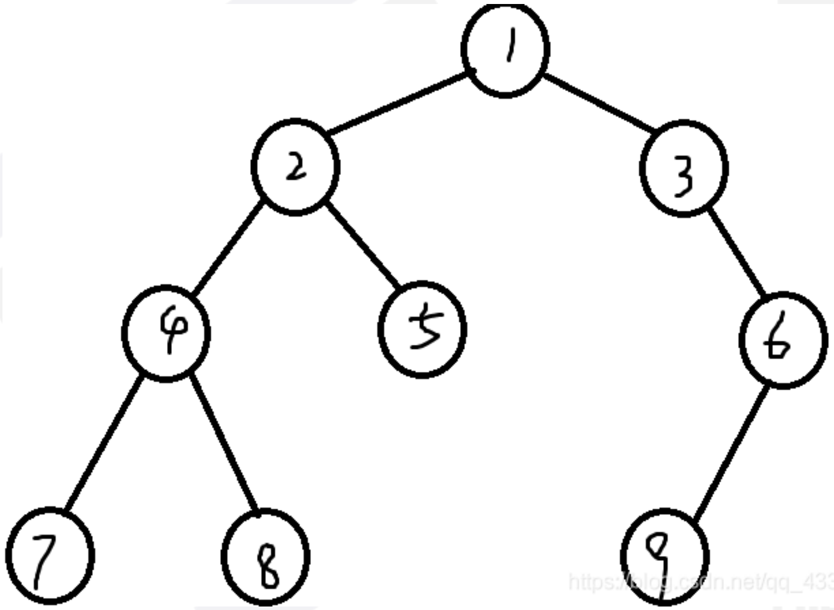
问题

给定n个点的联通图，由n-1条边组成，q次询问，求任意两点的距离

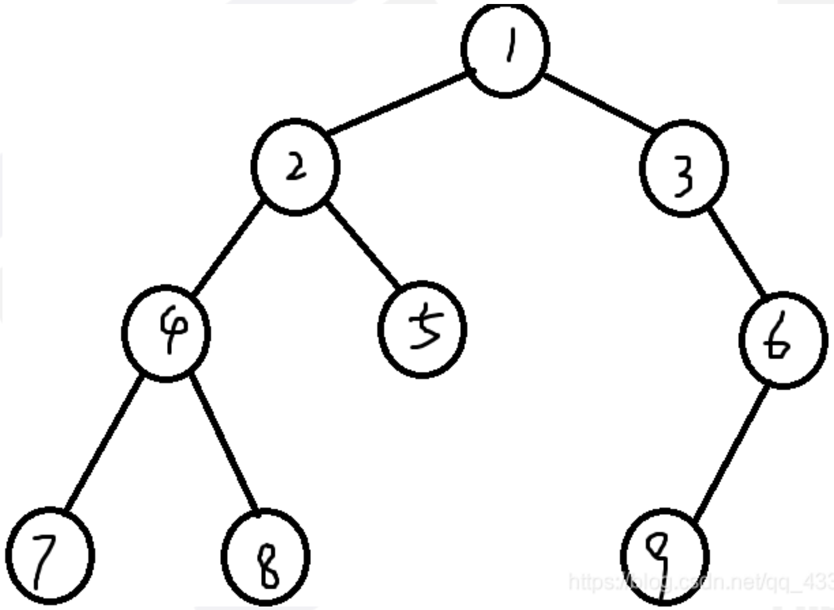


方法一：

Floyed

$O(n^3)$

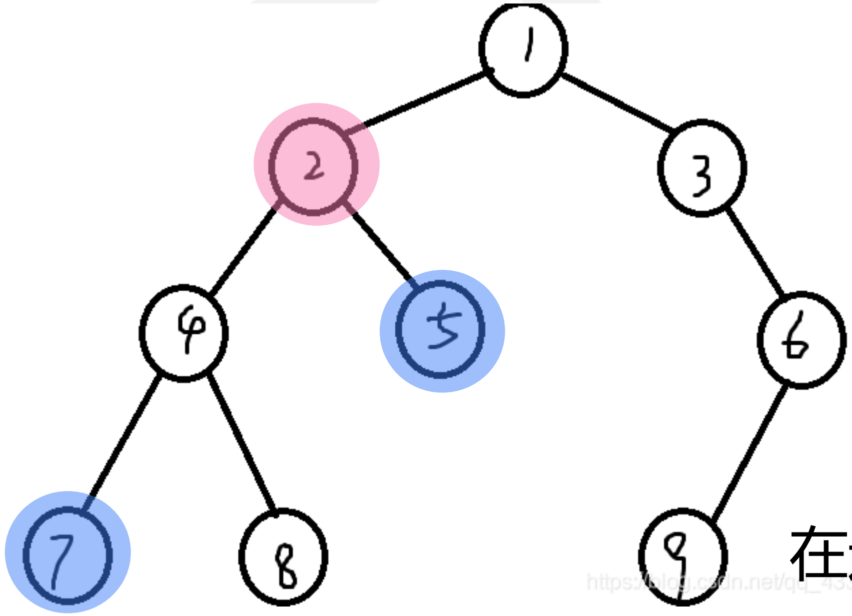
给定n个点的联通图，由n-1条边组成，q次询问，求任意两点的距离



方法二：
dfs或bfs
O(nq)

问题

给定n个点的联通图，由n-1条边组成，q次询问，求任意两点的距离



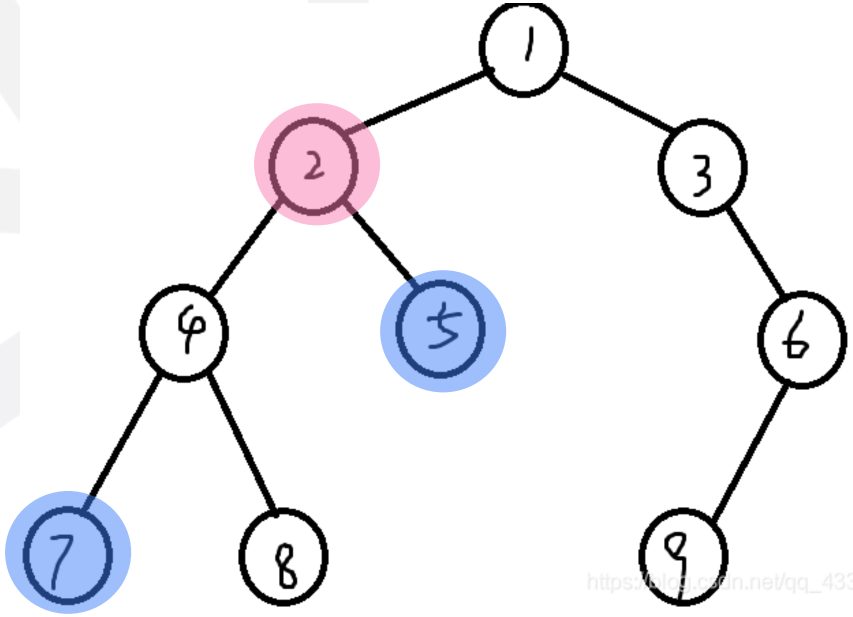
设dep[i]表示i到根节点的距离（即节点深度）

dis(5,7)=
dep[5]+dep[7]-2*dep[2]

在这里2号节点称为5号节点和7号节点的最近公共祖先(LCA)

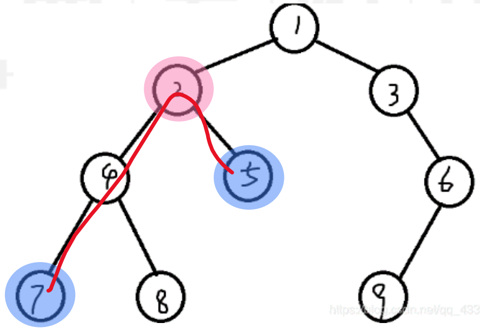
倍增求LCA

最近公共祖先简称 LCA（Lowest Common ancestor）。两个节点的最近公共祖先，就是这两个点的公共祖先里面，离根最远的那个。



需要知道的一些性质

- $LCA(u)=u$
- $LCA(u,v)=LCA(v,u)$
- 如果u是v的祖先，那么$LCA(u,v)=u$
- 如果u不是v的祖先，且v不是u的祖先，那么u和v处在$LCA(u,v)$的两棵不同子树中
- $LCA(u,v,c)=LCA(\mathbf{LCA(u,v)},c)=LCA(u,\mathbf{LCA(v,c)})$
  - 推广:$LCA(a_1,a_2,...,a_n)=LCA(\mathbf{LCA(a_1,a_2,...,a_k)},\mathbf{LCA(a_{k+1},u_{k+2},...,a_n)})$
- 树上任意两点u,v之间**有且仅有**一条简单路径，且$LCA(u,v)$位于这条路径上距离根最近的点

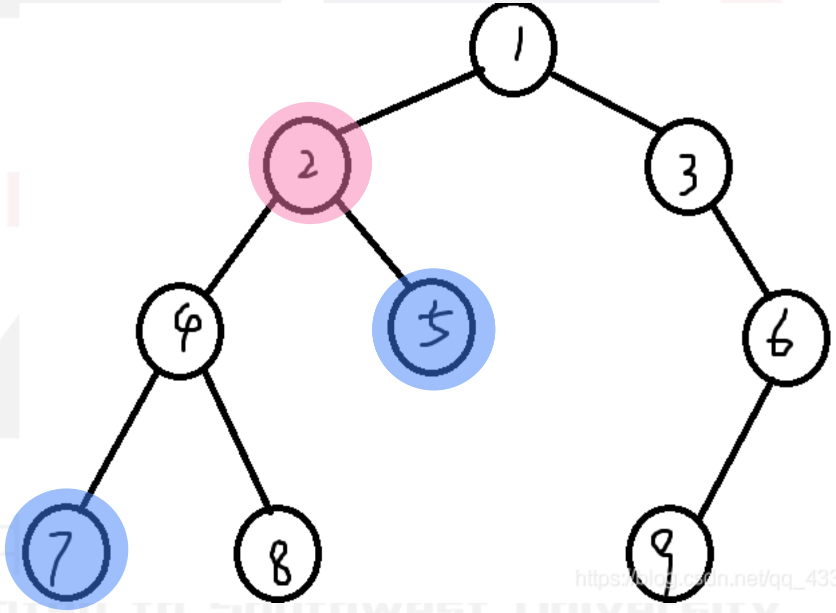


计算LCA

给定任意两个点u,v计算他们的LCA

如何计算?

方法一：向上标记法
1.从u向上走到根节点，并标记所经过的节点。
2.从v向上走到根节点，当第一次遇到标记的点，就是LCA(u,v)。
时间复杂度O(n)
缺陷：每次求解LCA都需要遍历一遍



计算LCA

给定任意两个点u,v计算他们的LCA

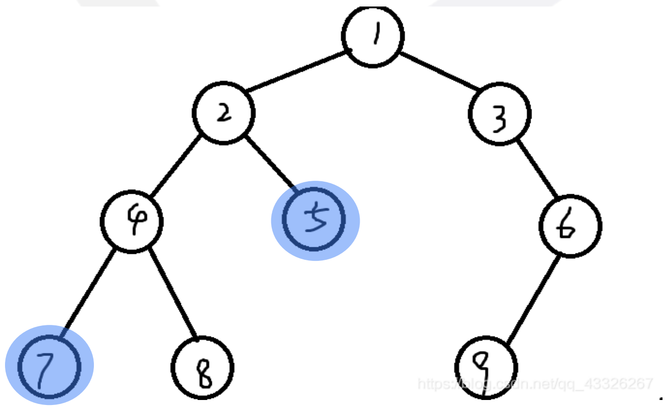
如何计算?

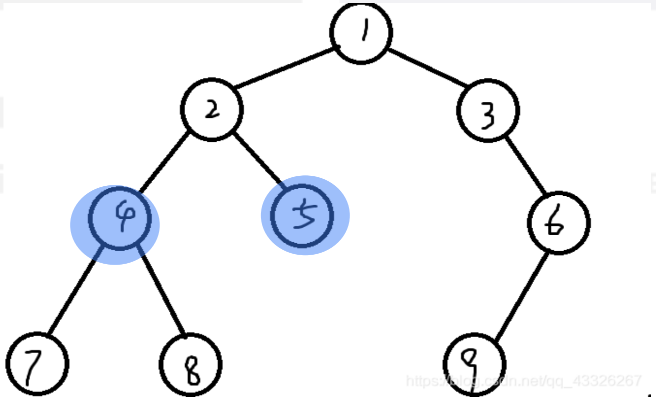
方法二：跳跃法

先找到求出每个点的深度

让两个点中较深的点，让他向上跳直到和另一个点在同一深度

然后两个点同时向上跳，直到两个点相一步一步的遇，相遇的地方就是他们的LCA





复杂度分析

- 需要预处理每一个点的深度
- 时间复杂度为O(n)
- 单次向上跳最坏情况下需要跳n次
- 时间复杂度为O(n)

如何求解每个节点的深度?

dfs/bfs

朴素算法-伪代码实现

f[i]:i的父亲节点

dfs预处理dep,f数组

```
void dfs(int u,int v){
    f[u]=v;dep[u]=dep[v]+1;
    for( int i=head[u];i;i=e[i].nex){
        int t=e[i].to;
        if(t!=f)
         dfs(t,u);
    }
}
```

计算LCA

```
int LCA(int u,int v){
    if(dep[u]<dep[v])
        swap(u,v)//让u表示较深的点
    while(dep[u]>dep[v])
        u=f[u];//向上跳一个
    while(u!=v){
        u=f[u]; //同时向上跳
        v=f[v];
    }
    return u;
}
```

有没有更快的做法?

引入倍增，优化跳的过程

改进

优化的关键在于：

- 如何快速让u找到与v同深度的点
- 如何让u与v快速相遇

```
int LCA(int u,int v){
    if(dep[u]<dep[v])
        swap(u,v)
    while(dep[u]>dep[v])
        u=fa[u];//向上跳一个
    while(u!=v){
        u=fa[u];//同时向上跳
        v=fa[v];
    }
    return u;
}
```

根据二进制拆分的思想：

任何一个非负整数都可以写成2的次幂相加的形式：

$S=2^{a1}+2^{a2}+\cdots+2^{an}$

如果要跳S步，那么可以第一次跳2^{a1}步，第二次跳2^{a2}步，…第n次跳2^{an}步。

设数组f[x][k]：

为与x相距2^k条边的祖先节点的编号(即x跳2^k次才能到的祖先节点)

- 当2^k超过根的时候，f[x][k]就等于根节点;若节点不存在， f[x][k]=0

  例如图中：f[7][0]=4; f[7][1]=2 ;f[7][2]=**1**

- 显然对于任意一棵树，f数组的第二维k<=log(n),f[x][0]为父亲节点

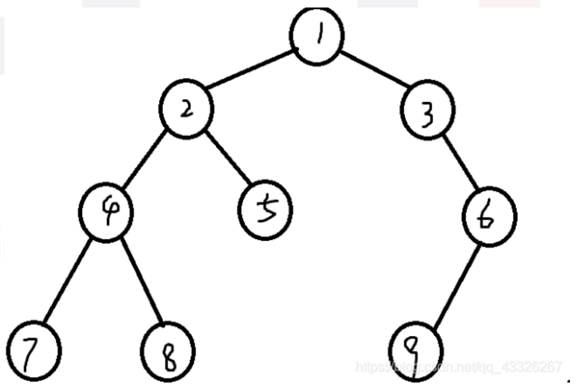
如何计算f数组，如何快速求得节点x的深度为y的祖先呢?

引入倍增:

x跳2^k步可以看作x先跳2^{k-1}步，再跳2^{k-1}步:

f[x][k]=f[f[x][k-1]][k-1]

这类似一个动态规划的过程，“阶段”就是结点的深度



算法过程

西南大学附属中学
High School Affiliated to Southwest University

1、预处理出每个节点的深度dep[i]，以及预处理f[x][k]

2、先将x，y跳到深度一致的位置

3、x,y同时向上跳2^k步，保持深度一致且不会相遇
4、若x,y没有相遇x=f[x][k],y=f[y][k]，继续上面这个过程

为保证更快，k从大到小枚举

5、向上跳结束时，x和y必定只差一步就到LCA了

预处理代码

```
void dfs(int u, int v)
{
	f[u][0] = v; depth[u] = depth[v] + 1;
	for(int i = 1; i <= 15; ++i)  //一般预处理到15即可
		f[u][i] = f[f[u][i-1]][i-1];
	for(int i = head[u]; i; i = e[i].nex){  //前向星建图
		int t= e[i].to;
		if(t!= v)
			dfs(t, u);
	}
}
```

```
struct edge {
	int to, nex;
};
```

初始化调用：dfs(Root,0   )

预处理时间复杂度：O(nlogn)

求解lca代码


```
int lca(int u,int v)
{
// 若u的深度比v浅
    if(depth[u] < depth[v]){
        swap(u,v);
    }
// 把u跳到和v同一层
    for(int i = 15;i >= 0;i --){
        if(depth[f[u][i]] >= depth[v]){
            u = f[u][i];
        }
    }
    if(u == v) return u;
//  u和v同时倍增往上跳,
    for(int i = 15;i >= 0;i --){
        if(f[u][i] != f[v][i]){
            u = f[u][i],v = f[v][i];
        }
    }
    return f[u][0];
}
```

询问结点x到结点y的距离：

dep[x]+dep[y]-2*dep[LCA(x,y)]

单次询问时间复杂度：O(logn)

小结

倍增求LCA只是其中一种方法

还有tarjan算法、转化DFS序+RMQ问题的解决、树链剖分等
https://blog.csdn.net/Cold_Chair/article/details/71249622

目前讲的只是查询距离，还未涉及到修改操作—树上差分

感兴趣的同学可以提前去了解一下，越到后期，想要进步更快更多的是自己的主动学习

FZOI

西|大|附 中|信|息|学|竞 赛

High School Affiliated to Southwest University