# 图论

## 树、二叉树

和自然界中的树不同，计算机中的树是上下颠倒的，即根在上，叶子在下。

**树的性质**：

1. 任意两点路径唯一
2. 任意非根节点，爹唯一

有关树的题目需着重关注其性质

### 二叉树的编号

对于编号从1开始的节点k，其左右儿子分别为2k、2k+1。

这样我们就可以用数组保存二叉树，但有时树的深度很深，但是很多节点没有满，这样存比较浪费空间。

### 存二叉树

struct node{  
 int n, l, r; //编号、左右儿子  
}a[233];

有时也需要保存父亲是谁。

### 二叉树的深度优先遍历

* 先序遍历 - 根、左儿子、右儿子
* 中序遍历 - 左儿子、根、右儿子
* 后序遍历 - 左儿子、右儿子、根

这三种都是深度优先遍历（Depth-First Search），总是往深处访问。

已知后序遍历、中序遍历，如何求这棵树？

——可以通过后序遍历找到这棵树的根，然后在中序遍历中找到左子树、右子树分别有哪些节点，然后就可在后序遍历中知道左右儿子，然后递归求解即可。

### 树的直径

两种求解方法都适用边权不一定为1的情况

* 两遍dfs

随便选取一起点，dfs到距离最远的点；再以这个距离最远的点为起点，dfs到距离最远的点，于是可以求出直径。

* dp（记忆化搜索）

其中为过的最长路径，和分别为以为端点的最长路和次长路，这样的话可以通过记忆化搜索来实现。

### 树上最近公共祖先（LCA）

去洛谷看板子题的时候竟然发现之前打过（好像还不是抄的题解），于是直接把洛谷的提交粘过来了（甚至第一遍AC是在2019-11-15 15:59，第一次提交是在2019-11-15 15:48）

基本思路是一遍dfs统计深度和，后者表示节点的往上的第个爹是谁，可以这样求

用人话解释一遍就是，我的第个爹，当然是第个爹的第个爹，很合理。不过一般实现程序的时候，i都会被用作循环变量，还请务必注意（下面的程序也是）

调用这个dfs时，可以dfs(root, 0)，相当于设置一个虚空节点，可以让所有超范围的fa数组的值都指向这个虚空节点

void dfs(int node, int fat) { // fat表示node的爹是谁  
 fa[node][0] = fat; dep[node] = dep[fat] + 1;  
 for (int i = 1; i <= 19; ++i)  
 fa[node][i] = fa[fa[node][i - 1]][i - 1]; // 我的爷爷是我的爹的爹  
 for (int i = 0; i < size[node]; ++i)  
 if (edge[node][i] != fat) dfs(edge[node][i], node);  
}

在预处理完毕之后，照着下面打就对了……

int lca(int a, int b) {  
 if (dep[a] < dep[b]) swap(a, b);  
 for (int i = 19; i >= 0; --i)  
 if (dep[fa[a][i]] >= dep[b]) a = fa[a][i]; // 先找到同层的点  
 if (a == b) return b; // 发现b是a的某个爹就return  
 for (int i = 19; i >= 0; --i)  
 if (fa[a][i] != fa[b][i])  
 a = fa[a][i], b = fa[b][i];  
 return fa[a][0]; // 到最后一定是找到了共同爹的俩儿子  
}

### 树的重心（质心）

定义：删除树的重心后，树的各个连通分量中，节点数最多的连通分量其节点数达到最小值

**性质**：

1. 树上所有的点到树的重心的距离之和是最短的，如果有多个重心，那么总距离相等。
2. 把两棵树通过一条边相连，新的树的重心在原来两棵树重心的连线上。
3. 一棵树添加或者删除一个节点，树的重心最多只移动一条边的位置。
4. 一棵树最多有两个重心，且相邻。

具体求解可以通过一遍dfs（记忆化搜索）来求，很好求的。

### 树上前缀和

表示节点到根的权值和

* 若是点权，路径上权值和为
* 若是边权，路径上权值和为

### 树上差分

树上差分之前需要求LCA，具体而言，有对于点的差分，也有对于边的差分

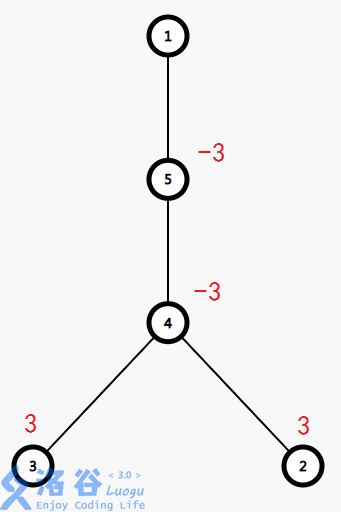
* 点的差分

eg：给定表示从走到，统计树上的点被经过的次数

若想将从到路径上的点权+x，可以这样维护差分数组：

diff[u] += x; diff[LCA(u, v)] -= x; // 可以看做维护从u到LCA的儿子  
diff[v] += x; diff[fa[LCA(u, v)][0]] -= x; // 可以看做维护从v到LCA

结合下面图片更好理解：



树上点的差分

这样的话，从这张图可以看出，实际求前缀和维护点权的时候是**从叶子向根节点更新的**

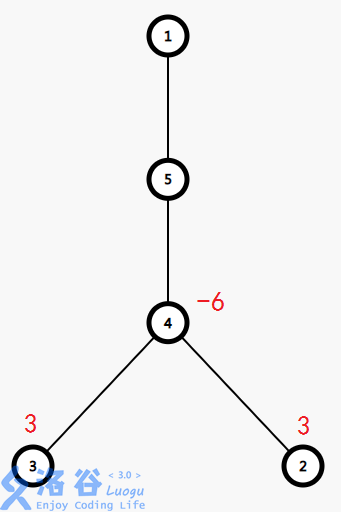
代码实现可以从根开始dfs，从儿子节点返回后累加diff数组。

void dfs\_solve(int now, int fat) {  
 tot[now] = diff[now];  
 for (int i = 0; i < to[now].size(); ++i) {  
 if (to[now][i] == fat) {  
 continue;  
 }  
 dfs\_solve(to[now][i], now); // 一定是递归调用自己  
 tot[now] += tot[to[now][i]];  
 }  
 ans = max(ans, tot[now]);  
}

注意这里的函数名，因为我的板子里求LCA也有一个叫dfs的函数，这个一定要区分好

* 边的差分

diff[u] += x; diff[v] += x;   
diff[LCA(u, v)] += x + x;



树上边的差分

只要dfs一遍，遍历时统计以每个节点为根的树的节点的权值和，就是当前节点到父亲节点的边的最终权值了

* eg：NOIP2015 运输计划

给你一个n个点的树，对于n-1条边各有边权。给出一些点对(x,y),同时定义d(x,y)表示x,y两点间的树上距离，现允许你将一条边的权值变为0，请你最小化最大的d值

思路：

两点路径一遍dfs预处理之后，求解，表示到根节点的路径和，那不难得出

再二分答案，即二分最大的dis。对于每次check，标记边权为这条边有多少条的路径经过（边的差分），并统计当前有多少条路径和长度大于。

check为true的条件是，存在一条边，全部d值超过mid的路径都经过这条边，并且把这条边的长度变成0之后，可以使得

## 图

不难证明（虽然我不会），连通的无环图就是树。

需要注意的是，图论问题并不都是直接以图的形式展示出来的，有时候需要选手根据题目自行建图。二元关系（两个元素之间的关系）都可以用图来描述，节点表示元素，边表示关系。

下面给出对于图的一些最基本的操作。

* 存图

int out[MAXN]; //存储每个点的出度  
struct node{int node; llint dis; }; //表示节点编号、单元最短路距离  
struct edge{int f, t, w; };  
vector<edge> v[MAXN];

* 加边

//加一条从f到t的边，其权值为w。这样加的边是单向的，若想加双向边加两次即可。  
void add(int f, int t, int w) {v[f].push\_back((edge){f, t, w}); ++out[f]; }

当然，我们可以用邻接矩阵，开一个二维数组dis[n][n]来表示边，不过在某些情况下这样可能很占空间。

### 求联通块

dfs即可，注意开一个mark数组标记某一点是否访问过

### BFS求最短路

对于边权为1的图，显然先遍历到的就是最近的。入队列时应把距离一并压入。

### 拓扑排序

适用于有向无环图（DAG）。值得注意的一点是，有向无环图并不是树，因为树的边是无向的。下面给出拓扑排序的步骤：

1. 统计所有入度为0的点，将其加入队列中。
2. 对于BFS到的某一个点，使其指向的点入度减一。如果某点入度被更新为0，将其加入队列中。

值得注意的是，如果结果中的节点个数不等于图的节点个数，说明该图存在环，无法进行拓扑排序。

### 欧拉回路

欧拉回路为走过图上所有边且每条边只经过一次的回路，但同一个节点可以走很多次。我们把存在欧拉回路的图成为欧拉图，可以形象地理解为能够“一笔画”的图。

对于无向图而言，若存在欧拉回路，则最多可以有两个奇数度数的节点，分别是起点和重点。

对于有向图，在忽略边的方向后，图必须是联通的；且最多只能有两个点入度和出度不相等，其中一个入度比出度多1，作为终点，另一个出度比入度多1，作为起点。

判断并打印欧拉回路只需要进行DFS即可，建一个vis[][]数组表示某条边是否被遍历过。

### 最小生成树（Kruskal算法）

即给定无向图，求边权和最小的生成树。

Kruskal算法实现步骤如下：

1. 将所有边按照权重升序排序；
2. 创建并查集；
3. 按照升序遍历所有边，如果不在一个集合中，则加入该条边；
4. 加入边数为(n - 1)时结束。

并查集相关请见“数据结构初步”。

### 单源最短路（Dijkstra算法）

~~你说SPFA？它死了。~~

Dijkstra适用于边权为正的情况。基本思路为：

下面的代码是高中的时候写的，建图存图和本篇方法相同。

//之前已经将小于号重定义成大于号  
memset(dis, 0x3f, sizeof dis); dis[s] = 0; q.push((node){s, 0});  
while (!q.empty()) {  
 int now; llint ndis; node tmp; tmp = q.top(); q.pop();  
 now = tmp.node; ndis = tmp.dis;  
 if (vis[now]) continue; vis[now] = true;  
 for (int i = 0; i < out[now]; ++i) {  
 int t = v[now][i].t, w = v[now][i].w  
 llint tdis = dis[t]; // dis要开long long  
 if (vis[t]) continue;  
 if (tdis > ndis + w) {  
 dis[t] = ndis + w; q.push((node){t, ndis + w});   
 }  
 }  
}

算法的基本实现方法如下：

1. 初始化dis数组，并创建vis数组标记某点的dis是否确定；
2. 创建优先队列q，并重定义小于号；
3. 把起始点的dis赋值为0，并将其加入q中；
4. 当q不为空时，重复取出队首执行以下步骤：
   * 如果vis过，那么continue，否则将队首的vis标记为true；
   * 遍历该点的每个出边，尝试更新目标点的dis；
   * 若没有vis且更新了dis，则将目标点加入队列中。

### SPFA

它死了。

实现基本与SPFA相似，不同的区别在于用普通队列维护需要进行松弛操作的点，用vis维护当前有哪些点在队列中（避免重复放入队列），对于取到的点，进行与Dijkstra相同的松弛操作。

### 多源最短路（Floyd-Warshall算法）

记住下面的代码，注意是**先枚举k**

memset(dis, 0x3f, sizeof(dis));  
for (int i = 0; i < n; ++i) dis[i][i] = 0;  
//中间省略赋值环节，即d[i][j]初始值是边的长度  
for (int k = 0; k < n; ++k)   
 for (int i = 0; i < n; ++i)  
 for (int j = 0; j < n; ++j)  
 d[i][j] = min(d[i][k] + d[j][k], d[i][j]);

### 图匹配

先给些定义：

1. 两两没有公共点的边组成的集合称为**匹配**
2. 所有匹配中边数最大的称为**最大匹配**
3. 图中带权时，边权和最大的匹配称为**最大权匹配**

* **二分图最大匹配**

（学不动了）