树

Tree

河南省实验中学信息技术组

2026 年 02 月 22 日

树的定义

- 树是由 $n(n \geq 0)$ 个结点组成的有限集。
  - 当 $n = 0$ 时，树被称为空树。
  - 否则，树中有且只有一个特定的结点称为根 (Root)。
  - 当 $n > 1$ 时，其余结点可分为 $m(m > 0)$ 个互不相交的有限集 $T_1, T_2, \cdots, T_m$，其中每个集合本身又是一棵树，称为根的的子树 (SubTree)。
- 树的元素之间是一对多的关系。每个结点最多只有一个前驱，每个结点可以有很多后继。
- 树的定义是递归的。

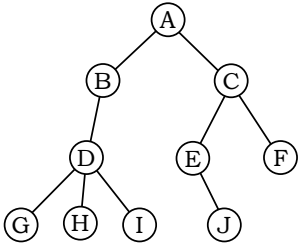


图: 树

树

- 树如果有根，则有且只有一个根。
- 树如果有子树，子树的个数没有限制，但它们一定是互不相交的。

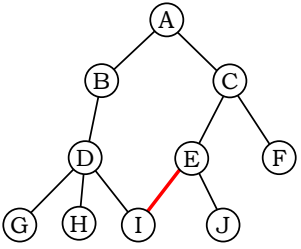


图: 两个子树相交

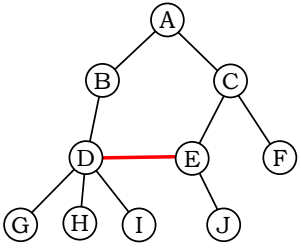


图: 两个子树相交

树的相关概念

- 结点 (node)：包括数据元素及指向其他结点的分支。
- 结点的度 (degree)：结点拥有子树的个数。
- 结点分类：
  - 叶结点 (leaf)：度为 0 的结点，又称为终端结点。
  - 根结点 (root)：没有双亲的结点。
  - 内部结点：度不为 0 且不是根结点的结点。
  - 孩子结点 (child)：结点子树的根结点。
  - 双亲结点 (parent)：结点有孩子结点，则该结点称为孩子结点的双亲结点，有时也称为父结点。
  - 兄弟姐妹结点 (sibling)：同一个双亲的孩子之间互为兄弟姐妹。
  - 祖先结点 (ancestor)：从根结点到该结点所经分支上的所有结点。
  - 后代结点 (descendant)：以某一个结点为根的子树中任一结点都称为该结点的后代。

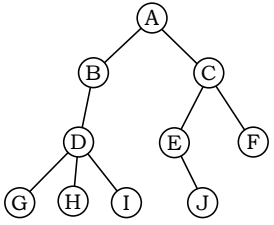


图: 树

树的相关概念

- 结点层次 (level)：从根开始，根为第 1 层，根的孩子为第 2 层，依此类推。
- 树的深度 (depth) 和高度：树中距离根结点最远的结点所处的层次即为树的深度。
- 树的度：树中结点的度的最大值。
- 有序树和无序树：如果将树中结点的各子树看成从左至右是有次序的、不能互换的，则称该树为有序树，否则称为无序树。
- 森林：是 $m(m \geq 0)$ 棵互不相交的树的集合。

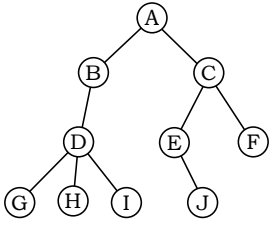


图: 树

树与线性结构比较

线性结构	树
• 首元素：无前驱 • 尾元素：无后继 • 中间元素：一个前驱一个后继	• 根结点：无双亲，唯一 • 叶结点：无孩子，可以多个 • 内部结点：一个双亲，多个孩子

树的存储

- 在考虑树的存储时，需要考虑如何存储结点中的数据和结点之间的关系。
- 树上的结点信息需要记录：结点数据、双亲、孩子。

$data$	p	c_1	c_2	$\cdots$	c_k

```
1 const int N = 100;
2 string data[N + 10];       // 结点数据 (不一定有)
3 int p[N + 10];             // 双亲结点的下标
4 vector<int> c[N + 10];     // c[x] 存储 x 的孩子结点的下标
5 int n = 0, rt = 0;         // 树大小和树的根结点编号
```

树的存储

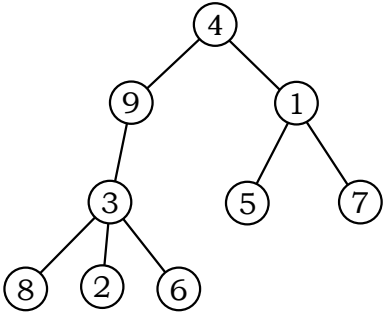


图: 树

下标 x	双亲 $p[x]$	孩子 $c[x]$
1	4	5,7
2	3	
3	9	8,2,6
4	0	9,1
5	1	
6	3	
7	1	
8	3	
9	4	3

树的存储

- 孩子结点：对于结点 x，遍历 $c[x]$ 很容易访问孩子结点，时间复杂度 $\mathrm{O}(\text{树的度})$。
- 双亲结点：对于结点 x，双亲为 $p[x]$，时间复杂度为 $\mathrm{O}(1)$。
- 祖先结点：x 的双亲为 $p[x]$，双亲的 $p[]$ 很容易找到双亲的双亲，以此类推直到 $p[]$ 为 0 结束，时间复杂度为 $\mathrm{O}(\text{树的深度})$。

【例】祖先路径

【题目描述】
给定一棵树，结点从 1 到 n 编号，输出结点到根结点的路径。
【输入格式】
第一行：一个整数 $n(1 \leq n \leq 100)$，表示结点的数目，结点编号从 $1 \sim n$。
接下来一行共 n 个整数，其中第 i 个整数表示结点 i 的双亲结点编号。
接下来一个整数 $q(1 \leq q \leq 100)$，表示 q 个询问。
接下来 q 行，每行一个整数 x，表示询问的结点编号。
【输出格式】
对于每个询问，输出一行数据，表示询问的结点到根结点的的路径上的所有结点编号。

【样例输入】

```
9
4 3 9 0 1 3 1 3 4
3
2
1
4
```

【样例输出】

```
2 3 9 4
1 4
4
```

【例】祖先路径

```cpp
// 非递归方法
void path(int x)  // 求结点到根结点的路径
{
    while(x != 0)
    {
        cout << x << " ";
        x = p[x];   // 向上移动
    }
}
```

```cpp
// 递归方法
void path(int x)
{
    if(x == 0) return;
    cout << x << " ";
    path(p[x]);
}
```

```cpp
// 主函数
int n; cin >> n;
for(int i = 1; i <= n; ++i) cin >> p[i];
int q; cin >> q;
while(q--)
{
    int x; cin >> x;
    path(x); cout << "\n";
}
```

【例】孩子结点

【题目描述】
给定一棵树，结点从 1 到 n 编号，输出结点的孩子结点。
【输入格式】
第一行一个整数 $n(n \leq 100)$，表示树的结点个数。
接下来 $n-1$ 行，每行两个整数 x, y，表示 x 是 y 的孩子 $(1 \leq x, y \leq n)$。
注意：对于每一个结点 f，他的孩子结点 c 在输入文件中越靠前出现，该孩子结点在树中就越靠左。
接下来一个整数 $q(1 \leq q \leq 100)$，表示 q 个询问。
接下来 q 行，每行一个整数 x，表示询问的结点编号。
【输出格式】
对于每个询问，输出孩子结点编号 (按照从左向右的顺序)，如果没有孩子结点，则在相应行输出 0。

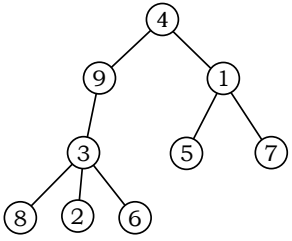
【例】孩子结点

【样例输入】

```
9
9 4
3 9
8 3
2 3
6 3
1 4
5 1
7 1
3
9
3
5
```

【样例输出】

```
3
8 2 6
0
```



【例】孩子结点

```
const int N = 100;
vector<int> c[N + 10]; // 孩子结点数组

void child(int x) // 求结点的孩子结点
{
    for(int i = 0; i < c[x].size(); ++i) cout << c[x][i] << " ";
}

int n; cin >> n;
for(int i = 1; i <= n - 1; ++i)
{
    int x, y; cin >> x >> y;
    c[y].push_back(x);     // 结点 x 是结点 y 的孩子
}
int q; cin >> q;
while(q--)
{
    int x; cin >> x;
    if(c[x].size()) child(x), cout << "\n"; // 有孩子
    else cout << "0\n";   // 无孩子
}
```

孩子兄弟表示法[1]

- 对于树中任意一个结点，它的第一个孩子如果存在就是唯一的，它的右兄弟如果存在也是唯一的。
- 孩子兄弟表示法：每个结点除了记录数据外，还记录第一个孩子的的位置和该结点第一个右兄弟的位置。

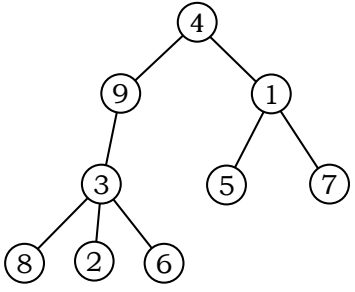
[1]感兴趣自行了解。

树的遍历

- 在应用树结构解决问题时，往往要求按照某种次序获得树中全部结点的信息，这种操作称为树的遍历。
- 遍历方法有很多种，常用的有：
  - 先序 (根) 遍历
  - 后序 (根) 遍历
  - 层次遍历

先序遍历和后序遍历

- 先序遍历：先访问根结点，然后从左到右按照先序遍历的方法遍历各棵子树。
- 后序遍历：先从左到右按照后序遍历的方法遍历各棵子树，再访问根结点。
- 如下图的树，先序遍历的序列为：4 9 3 8 2 6 1 5 7；后序遍历的序列为8 2 6 3 9 5 7 1 4。



图：树

先序遍历和后序遍历实现

- 从定义来看，遍历方法是递归的，所以在程序实现时使用递归。

```
1  void preorder(int x)   // 先序遍历以 x 为根的子树
2  {
3      // 先判断访问的树是否为空树 (递归出口)
4      if(x == 0) return;
5      cout << x << " ";              // 访问根结点
6      for(int i = 0; i < c[x].size(); ++i)
7          preorder(c[x][i]);          // 先序遍历各子树
8  }
9
10 void postorder(int x)  // 后序遍历以 x 为根的子树
11 {
12     // 先判断访问的树是否为空树 (递归出口)
13     if(x == 0) return;
14     for(int i = 0; i < c[x].size(); ++i)
15         postorder(c[x][i]);            // 后序遍历各子树
16     cout << x << " ";                // 访问根结点
17 }
18 // 主函数调用
19 preorder(rt);
20 postorder(rt);
```

层次遍历

- 层次遍历：对于一棵树，按照层次从小到大向下访问，同一层次按照从左到右的次序访问。
- 按照层次遍历的方法，下图中遍历顺序为：4 9 1 3 5 7 8 2 6

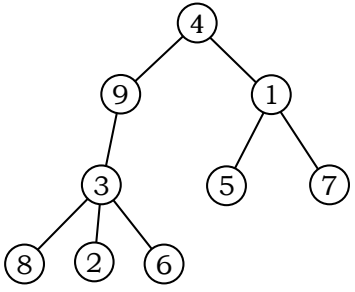


图: 树

层次遍历

通过观察可以发现，同一层次的结点，如果双亲结点相同，那么按照从左向右的顺序访问；如果双亲结点不同，那么如果双亲结点先访问，那么孩子结点也会先访问。所以，在访问结点时需要保存其孩子结点的访问顺序，可以设计一个队列来存储访问的顺序，方法如下：

- 首先，将根结点放入队列。
- 只要队列不为空，取出队头 (访问)，将其孩子结点放入队列。

层次遍历

```
1  void levelorder(int rt)
2  {
3      if(rt == 0) return; // 空树 直接结束
4      queue<int> q; q.push(rt); // 将根入队
5      while(!q.empty())
6      {
7          int x = q.front(); q.pop(); // 队头元素访问完 出队
8          cout << x << " ";   // 访问结点 打印结点数据
9          // 子结点入队
10         for(int i = 0; i < c[x].size(); ++i)
11         {
12             int y = c[x][i]; // y 是 x 的孩子
13             q.push(y);
14         }
15     }
16 }
17 // 主函数中调用
18 levelorder(rt);
```

遍历应用

- 树的遍历方法将树的复杂结构转化为线性序列。
- 树的遍历方法提供了对树中结点的不同访问顺序，可以在遍历过程中对结点进行不同的处理。

遍历应用

- 计算以 x 为根的树的结点个数：递归遍历根结点 x 各子树，计算各子树的结点个数之和，然后加上 1(根结点 x)。

```
1  int size(int x)
2  {
3      if(x == 0) return 0;  // 空树 大小为 0
4      int tot = 0;          // 树大小
5      for(int i = 0; i < c[x].size(); ++i)
6      {
7          int y = c[x][i]; // y 是 x 的孩子
8          tot += size(y);
9      }
10     return tot + 1;          // 树大小 = 子树大小之和 + 1
11 }
```

遍历应用

- 计算以 x 为根的树的深度 (高度)：递归遍历根结点的 x 各子树，计算各子树高度并取最大值，然后加 1(根结点 x 自身的深度)。
- 用层次遍历能实现吗？请读者自行实现。

```
1  int depth(int x)
2  {
3      if(x == 0) return 0;  // 空树 深度为 0
4      int d = 0;
5      for(int i = 0; i < c[x].size(); ++i)
6      {
7          int y = c[x][i]; // y 是 x 的孩子
8          d = max(d, depth(y));
9      }
10     return d + 1;           // 树深度 = 各子树深度最大值 + 1
11 }
```

遍历应用

- 计算结点层次 (深度)：层次遍历，孩子结点的层次等于双亲结点层次加 1。

```
1  int d[N + 10]; // 结点的层次 (深度)
2  void level(int rt)
3  {
4      if(rt == 0) return;   // 空树
5      queue<int> q; q.push(rt); d[rt] = 1;  // 根结点层次为 1
6      while(!q.empty())
7      {
8          int x = q.front(); q.pop();
9          for(int i = 0; i < c[x].size(); ++i)
10         {
11             int y = c[x][i];  // y 是 x 的孩子
12             d[y] = d[x] + 1;  // 子结点层次等于双亲结点层次加 1
13             q.push(y);
14         }
15     }
16 }
17 //主函数
18 level(rt);
19 for(int i = 1; i <= n; ++i) cout << d[i] << " ";
```

遍历应用

- 计算结点层次 (深度)：先序遍历，根结点的层次为 1，在先序遍历过程中，孩子结点的层次等于双亲结点层次加 1。

```
1  int d[N + 10]; // 结点的层次 (深度)
2  void level(int x)
3  {
4      if(rt == 0) return;   // 空树
5      for(int i = 0; i < c[x].size(); ++i)
6      {
7          int y = c[x][i];
8          d[y] = d[x] + 1;  // 子结点层次等于双亲结点层次加 1
9          level(y);
10     }
11 }
12 //主函数
13 d[rt] = 1;
14 level(rt);
15 for(int i = 1; i <= n; ++i) cout << d[i] << " ";
```

【例】树的权

【题目描述】
给定一棵有 n 个结点的树，树的结点编号从 1 到 n，树的每个结点都有权值，开始时权值都为 0。
我们定义了三种对树的权值进行处理的操作：权值加 1、权值减 1、求权值之和。
【输入格式】
第一行一个整数 $n(2 \leq n \leq 100)$，表示树的结点个数；
第 $2 \sim n$ 行为每个结点的信息，每行有两个整数 x 和 y，表示 x 是 y 的孩子 $(1 \leq x, y \leq n)$。
接下来一个整数 $q(q \leq 500)$，表示询问的次数。
接下来 q 行，每行两个整数 op, x，表示操作类型和操作的子树的根结点。
若 $op = 1$，则将以 x 为根的子树所有结点的权值加 1；若 $op = 2$，则将以 x 为根的子树所有结点的权值减 1；若 $op = 3$，则求以 x 为根的子树上所有结点的权值之和。
【输出格式】
对于权值加 1、权值减 1 操作，不需要输出。
对于求权值操作，输出一行一个整数，表示询问的子树的权值之和。

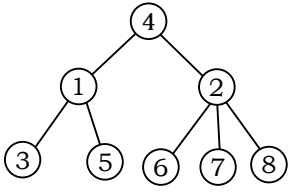
【例】树的权

【样例输入】

```
8
1 4
2 4
3 1
5 1
6 2
7 2
8 2
4
1 4
2 1
3 6
3 4
```

【样例输出】

```
1
5
```



【例】树的权

- 对于权值加 1 和减 1 的操作，在树的遍历 (前序、后序、层次) 过程中，修改结点的权值。
- 对于权值求和的操作，树的权值 = 根结点权值 + 子树的权值，在后序遍历过程中计算。

```
1  int d[N + 10];  // 树结点的权值
2  void update(int x, int val)
3  {
4      if(x == 0) return;
5      d[x] += val;
6      for(int i = 0; i < c[x].size(); ++i)
7          update(c[x][i]);
8  }
9
10 int value(int x)
11 {
12     int res = d[x];
13     for(int i = 0; i < c[x].size(); ++i)
14         res += d[c[x][i]];
15     return res;
16 }
```

【例】最近公共祖先

【题目描述】
给定一棵有根树，若结点 z 既是结点 x 的祖先，也是结点 y 的祖先，则称 z 是 x, y 的公共祖先。
在 x, y 的公共祖先中，深度最大的一个结点称为 x, y 的最近公共祖先，记为 $\mathrm{LCA}(x, y)$。
现给定一棵有 n 个结点构成的有根树，请编程找出某两点的最近公共祖先。
【输入格式】
第一行一个正整数 $n(n \leq 10000)$，表示树中结点个数。
接下来 $n-1$ 行，每行两个正整数 x, y，表示结点 x 是 y 的父亲，结点的孩子数不超过 100。
接下来一个正整数 $q(q \leq n)$，表示下面会有 q 次询问。
最后 q 行，每行两个正整数 x, x，表示询问 x, y 的最近公共祖先。。
【输出格式】
输出 q 行，每行一个正整数，表示 x 和 y 的最近公共祖先结点的编号。

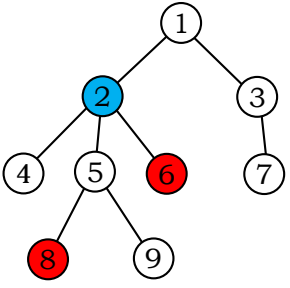
【例】最近公共祖先

【样例输入】

```
9
1 2
1 3
2 4
2 5
2 6
3 7
5 8
5 9
2
6 8
7 9
```

【样例输出】

```
2
1
```



【例】最近公共祖先

- 利用层次遍历或前序遍历求树中每个结点深度。
- 对于 x 和 y，首先将两个结点调整到深一致，即深度较大的向上移动。
- 然后同时向上移动 x 和 y 直到 $x = y$，此时 x 和 y 都是最近公共祖先。

```
int d[N + 10];  // 树结点的深度

int lca(int x, int y)
{
    if(d[x] < d[y]) swap(x, y);    // 让 x 深度大一些
    while(d[x] > d[y]) x = p[x];   // 将 x 调整到和 y 一样的深度
    while(x != y) x = p[x], y = p[y];// 二者一起上移直到相遇即为 LCA
    return x;
}
//主函数
level(rt);  // 求结点的深度
int q; cin >> q;
while(q--)
{
    int x, y; cin >> x >> y;
    cout << lca(x, y) << "\n";
}
```

【例】树结点距离

- 给定两个结点 x 和 y，求两个结点的距离 (x 到 y 路径上的边数)。
- 方法 1：两个结点的距离就是求两个顶点 LCA 时结点上移的次数。

```
1 int dist(int x, int y)
2 {
3     int ans = 0;
4     if(d[x] < d[y]) swap(x, y);
5     while(d[x] > d[y]) x = p[x], ans += 1; // 一个点上移
6     while(x != y) x = p[x], y = p[y], ans += 2; // 两个点同时上移
7     return ans;
8 }
```

- 方法 2：两个结点的距离显然等于 x 到 $lca(x,y)$ 与 y 到 $lca(x,y)$ 的距离之和，那么答案显然为 $d[x]+d[y]-2\times d[lca(x,y)]$。

练习

- 祖先路径 (A0212)
- 双亲和孩子 (A0213)
- 找树根和孩子 (A0214)
- 树的先序和后序遍历 (A0215)
- 树的层次遍历 (A0216)
- 树的权 (A0217)
- 子树大小 (A0218)
- 重儿子 (A0219)
- 树结点深度 (A0220)
- LCA 基础版 (A0221)
- 树结点距离 (A0221)
- 单词查找树 [NOI 2000](COGS 293)