

【DFS/BFS】-树状结构查询

题目描述与示例

题目描述

通常使用多行的节点、父节点表示一棵树，比如

西安 陕西

陕西 中国

江西 中国

中国 亚洲

泰国 亚洲

输入一个节点之后，请打印出来树中他的所有下层节点

输入描述

第一行输入行数

接着是多行数据，每行以空格区分节点和父节点

最后是查询节点

树中的节点是唯一的，不会出现两个节点，是同一个名字

输出描述

输出查询节点的所有下层节点。以字典序排序

示例

输入

```
1 5
2 b a
3 c a
4 d c
5 e c
```

```
6 f d
7 c
```

输出

```
1 d
2 e
3 f
```

解题思路

本题属于非常常规的搜索题目，用DFS或BFS都可以完成。

首先使用哈希表构建邻接表 `children_dict`，对输入进行建树处理，即

```
1 n = int(input())
2 children_dict = defaultdict(list)
3
4 for _ in range(n):
5     child, parent = input().split()
6     children_dict[parent].append(child)
```

如果使用DFS，则递归函数如下

```
1 def dfs(node, ans, children_dic):
2     ans.append(node)
3     for child in children_dic[node]:
4         dfs(child, ans, children_dic)
```

如果使用BFS，则函数如下

```
1 def bfs(target, ans, children_dic):
2     q = deque()
3     q.append(target)
```

```

4     while q:
5         node = q.popleft()
6         for child in children_dic[node]:
7             q.append(child)
8             ans.append(child)

```

在输出结果的时候，需要注意不能输出 `target`。

代码

解法一：DFS

Python

```

1  # 题目：2024E-树状结构查询
2  # 分值：200
3  # 作者：许老师-闭着眼睛学数理化
4  # 算法：DFS
5  # 代码看不懂的地方，请直接在群上提问
6
7
8  from collections import defaultdict
9
10
11 # dfs函数
12 def dfs(node, ans, children_dic):
13     # 将node加入ans中
14     ans.append(node)
15     # 遍历node的所有子节点，递归调用子节点
16     for child in children_dic[node]:
17         dfs(child, ans, children_dic)
18
19 # 输入行数
20 n = int(input())
21 # 使用哈希表，储存某一个节点所有子节点构成的列表
22 # k为某个父节点，v为其所有子节点构成的列表
23 children_dict = defaultdict(list)
24
25 # 遍历N行，输入所有的父子节点连接情况
26 for _ in range(n):
27     # 获得子节点和父节点，注意父节点在后

```

```

28     child, parent = input().split()
29     children_dict[parent].append(child)
30
31 # 输入查询节点
32 target = input()
33
34 # 初始化答案列表
35 ans = list()
36 # dfs入口, 传入的节点为查询节点target
37 dfs(target, ans, children_dict)
38
39 # 根据字典序进行排序
40 ans.sort()
41 # 逐行输出ans中的所有值, 要注意ans中包含了查询target
42 # 根据题意target是不应该输出的, 需要多做一步判断
43 for node in ans:
44     if node != target:
45         print(node)

```

Java

```

1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6          int n = scanner.nextInt();
7          scanner.nextLine(); // Consume the newline
8
9          Map<String, List<String>> childrenMap = new HashMap<>();
10
11         for (int i = 0; i < n; i++) {
12             String[] input = scanner.nextLine().split(" ");
13             String child = input[0];
14             String parent = input[1];
15             childrenMap.computeIfAbsent(parent, k -> new ArrayList<>
16             ()).add(child);
17         }
18
19         String target = scanner.nextLine();
20         List<String> ans = new ArrayList<>();
21
22         dfs(target, ans, childrenMap);
23
24         Collections.sort(ans);

```

```

24         for (String node : ans) {
25             if (!node.equals(target)) {
26                 System.out.println(node);
27             }
28         }
29     }
30
31     private static void dfs(String node, List<String> ans, Map<String,
List<String>> childrenMap) {
32         ans.add(node);
33         List<String> children = childrenMap.getOrDefault(node, new ArrayList<>
());
34         for (String child : children) {
35             dfs(child, ans, childrenMap);
36         }
37     }
38 }
39

```

C++

```

1  #include <iostream>
2  #include <vector>
3  #include <map>
4  #include <algorithm>
5
6  using namespace std;
7
8  void dfs(string node, vector<string>& ans, map<string, vector<string>>&
childrenMap) {
9      ans.push_back(node);
10     vector<string>& children = childrenMap[node];
11     for (const string& child : children) {
12         dfs(child, ans, childrenMap);
13     }
14 }
15
16 int main() {
17     int n;
18     cin >> n;
19     cin.ignore(); // Consume the newline
20
21     map<string, vector<string>> childrenMap;
22
23     for (int i = 0; i < n; i++) {

```

```

24     string child, parent;
25     cin >> child >> parent;
26     childrenMap[parent].push_back(child);
27 }
28
29 string target;
30 cin >> target;
31
32 vector<string> ans;
33
34 dfs(target, ans, childrenMap);
35
36 sort(ans.begin(), ans.end());
37
38 for (const string& node : ans) {
39     if (node != target) {
40         cout << node << endl;
41     }
42 }
43
44 return 0;
45 }
46

```

解法二：BFS

Python

```

1  # 题目：2024E-树状结构查询
2  # 分值：200
3  # 作者：许老师-闭着眼睛学数理化
4  # 算法：BFS
5  # 代码看不懂的地方，请直接在群上提问
6
7
8  from collections import defaultdict, deque
9
10
11 # bfs函数
12 def bfs(target, ans, children_dic):
13     # 初始化队列，包含节点target
14     # 注意target不用加入ans中
15     q = deque()
16     q.append(target)
17     # 进行BFS

```

```

18     while q:
19         # 弹出队头元素node
20         node = q.popleft()
21         # 考虑node的所有子节点
22         for child in children_dic[node]:
23             # 将子节点加入q和ans中
24             q.append(child)
25             ans.append(child)
26
27 # 输入行数
28 n = int(input())
29 # 使用哈希表, 储存某一个节点所有子节点构成的列表
30 # k为某个父节点, v为其所有子节点构成的列表
31 children_dict = defaultdict(list)
32
33 # 遍历N行, 输入所有的父子节点连接情况
34 for _ in range(n):
35     # 获得子节点和父节点, 注意父节点在后
36     child, parent = input().split()
37     children_dict[parent].append(child)
38
39 # 输入查询节点
40 target = input()
41
42 # 初始化答案列表
43 ans = list()
44 # 进行bfs
45 bfs(target, ans, children_dict)
46
47 # 根据字典序进行排序
48 ans.sort()
49 # 逐行输出ans中的所有值
50 for node in ans:
51     print(node)

```

Java

```

1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6         int n = scanner.nextInt();
7         scanner.nextLine(); // Consume the newline
8

```

```

9      Map<String, List<String>> childrenMap = new HashMap<>();
10
11      for (int i = 0; i < n; i++) {
12          String[] input = scanner.nextLine().split(" ");
13          String child = input[0];
14          String parent = input[1];
15          childrenMap.computeIfAbsent(parent, k -> new ArrayList<>
16      ()).add(child);
17
18      }
19
20      String target = scanner.nextLine();
21      List<String> ans = new ArrayList<>();
22
23      bfs(target, ans, childrenMap);
24
25      Collections.sort(ans);
26      for (String node : ans) {
27          System.out.println(node);
28      }
29
30      private static void bfs(String target, List<String> ans, Map<String,
31      List<String>> childrenMap) {
32
33          Queue<String> queue = new LinkedList<>();
34          queue.offer(target);
35
36          while (!queue.isEmpty()) {
37              String node = queue.poll();
38              List<String> children = childrenMap.getOrDefault(node, new
39      ArrayList<>());
40
41              for (String child : children) {
42                  queue.offer(child);
43                  ans.add(child);
44              }
45          }
46      }
47  }
48  }
49  }

```

C++

```

1  #include <iostream>
2  #include <vector>
3  #include <map>
4  #include <queue>

```



```

5 #include <algorithm>
6
7 using namespace std;
8
9 void bfs(string target, vector<string>& ans, map<string, vector<string>>&
  childrenMap) {
10     queue<string> q;
11     q.push(target);
12
13     while (!q.empty()) {
14         string node = q.front();
15         q.pop();
16         vector<string>& children = childrenMap[node];
17         for (const string& child : children) {
18             q.push(child);
19             ans.push_back(child);
20         }
21     }
22 }
23
24 int main() {
25     int n;
26     cin >> n;
27     cin.ignore(); // Consume the newline
28
29     map<string, vector<string>> childrenMap;
30
31     for (int i = 0; i < n; i++) {
32         string child, parent;
33         cin >> child >> parent;
34         childrenMap[parent].push_back(child);
35     }
36
37     string target;
38     cin >> target;
39
40     vector<string> ans;
41
42     bfs(target, ans, childrenMap);
43
44     sort(ans.begin(), ans.end());
45
46     for (const string& node : ans) {
47         cout << node << endl;
48     }
49
50     return 0;

```

```
51 }  
52
```

时空复杂度

时间复杂度： $O(N)$ 。无论是DFS还是BFS，最差的情况都是遍历整棵树。

空间复杂度： $O(N)$ 。`children_dict` 所占用空间。