【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

```
5
b a
c a
d c
e c
```

```
f d
c
```

输出

```
d
e
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

```
n = int(input())
children_dict = defaultdict(list)

for _ in range(n):
    child, parent = input().split()
    children_dict[parent].append(child)
```

如果使用DFS，则递归函数如下

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

如果使用BFS，则函数如下

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

```
    while q:
        node = q.popleft()
        for child in children_dic[node]:
            q.append(child)
            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

```python
# 题目：2024E-树状结构查询
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：DFS
# 代码看不懂的地方，请直接在群上提问


from collections import defaultdict


# dfs函数
def dfs(node, ans, children_dic):
    # 将node加入ans中
    ans.append(node)
    # 遍历node的所有子节点，递归调用子节点
    for child in children_dic[node]:
        dfs(child, ans, children_dic)

# 输入行数
n = int(input())
# 使用哈希表，储存某一个节点所有子节点构成的列表
# k为某个父节点，v为其所有子节点构成的列表
children_dict = defaultdict(list)

# 遍历N行，输入所有的父子节点连接情况
for _ in range(n):
    # 获得子节点和父节点，注意父节点在后
```

```
    child, parent = input().split()
    children_dict[parent].append(child)

# 输入查询节点
target = input()

# 初始化答案列表
ans = list()
# dfs入口，传入的节点为查询节点target
dfs(target, ans, children_dict)

# 根据字典序进行排序
ans.sort()
# 逐行输出ans中的所有值，要注意ans中包含了查询target
# 根据题意target是不应该输出的，需要多做一步判断
for node in ans:
    if node != target:
        print(node)
```

Java

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume the newline

        Map<String, List<String>> childrenMap = new HashMap<>();

        for (int i = 0; i < n; i++) {
            String[] input = scanner.nextLine().split(" ");
            String child = input[0];
            String parent = input[1];
            childrenMap.computeIfAbsent(parent, k -> new ArrayList<>
()).add(child);
        }

        String target = scanner.nextLine();
        List<String> ans = new ArrayList<>();

        dfs(target, ans, childrenMap);

        Collections.sort(ans);
```

```
        for (String node : ans) {
            if (!node.equals(target)) {
                System.out.println(node);
            }
        }
    }

    private static void dfs(String node, List<String> ans, Map<String,
List<String>> childrenMap) {
        ans.add(node);
        List<String> children = childrenMap.getOrDefault(node, new ArrayList<>
());
        for (String child : children) {
            dfs(child, ans, childrenMap);
        }
    }
}

```

C++

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>

using namespace std;

void dfs(string node, vector<string>& ans, map<string, vector<string>>&
childrenMap) {
    ans.push_back(node);
    vector<string>& children = childrenMap[node];
    for (const string& child : children) {
        dfs(child, ans, childrenMap);
    }
}

int main() {
    int n;
    cin >> n;
    cin.ignore(); // Consume the newline

    map<string, vector<string>> childrenMap;

    for (int i = 0; i < n; i++) {
```

```
        string child, parent;
        cin >> child >> parent;
        childrenMap[parent].push_back(child);
    }

    string target;
    cin >> target;

    vector<string> ans;

    dfs(target, ans, childrenMap);

    sort(ans.begin(), ans.end());

    for (const string& node : ans) {
        if (node != target) {
            cout << node << endl;
        }
    }

    return 0;
}

```

解法二：BFS

Python

```
# 题目：2024E-树状结构查询
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：BFS
# 代码看不懂的地方，请直接在群上提问


from collections import defaultdict, deque


# bfs函数
def bfs(target, ans, children_dic):
    # 初始化队列，包含节点target
    # 注意target不用加入ans中
    q = deque()
    q.append(target)
    # 进行BFS
```

```python
    while q:
        # 弹出队头元素node
        node = q.popleft()
        # 考虑node的所有子节点
        for child in children_dic[node]:
            # 将子节点加入q和ans中
            q.append(child)
            ans.append(child)

# 输入行数
n = int(input())
# 使用哈希表，储存某一个节点所有子节点构成的列表
# k为某个父节点，v为其所有子节点构成的列表
children_dict = defaultdict(list)

# 遍历N行，输入所有的父子节点连接情况
for _ in range(n):
    # 获得子节点和父节点，注意父节点在后
    child, parent = input().split()
    children_dict[parent].append(child)

# 输入查询节点
target = input()

# 初始化答案列表
ans = list()
# 进行bfs
bfs(target, ans, children_dict)

# 根据字典序进行排序
ans.sort()
# 逐行输出ans中的所有值
for node in ans:
    print(node)
```

Java

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume the newline

```

```
        Map<String, List<String>> childrenMap = new HashMap<>();

        for (int i = 0; i < n; i++) {
            String[] input = scanner.nextLine().split(" ");
            String child = input[0];
            String parent = input[1];
            childrenMap.computeIfAbsent(parent, k -> new ArrayList<>
()).add(child);
        }

        String target = scanner.nextLine();
        List<String> ans = new ArrayList<>();

        bfs(target, ans, childrenMap);

        Collections.sort(ans);
        for (String node : ans) {
            System.out.println(node);
        }
    }

    private static void bfs(String target, List<String> ans, Map<String,
List<String>> childrenMap) {
        Queue<String> queue = new LinkedList<>();
        queue.offer(target);

        while (!queue.isEmpty()) {
            String node = queue.poll();
            List<String> children = childrenMap.getOrDefault(node, new
ArrayList<>());
            for (String child : children) {
                queue.offer(child);
                ans.add(child);
            }
        }
    }
}

```

C++

```
#include <iostream>
#include <vector>
#include <map>
#include <queue>
```

```cpp
#include <algorithm>

using namespace std;

void bfs(string target, vector<string>& ans, map<string, vector<string>>&
childrenMap) {
    queue<string> q;
    q.push(target);

    while (!q.empty()) {
        string node = q.front();
        q.pop();
        vector<string>& children = childrenMap[node];
        for (const string& child : children) {
            q.push(child);
            ans.push_back(child);
        }
    }
}

int main() {
    int n;
    cin >> n;
    cin.ignore(); // Consume the newline

    map<string, vector<string>> childrenMap;

    for (int i = 0; i < n; i++) {
        string child, parent;
        cin >> child >> parent;
        childrenMap[parent].push_back(child);
    }

    string target;
    cin >> target;

    vector<string> ans;

    bfs(target, ans, childrenMap);

    sort(ans.begin(), ans.end());

    for (const string& node : ans) {
        cout << node << endl;
    }

    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。