2023-9-5 狙击

dfs - 树形dp Python

```python
def dfs(u, fa):
    dp[u][0] = dp[u][1] = dp[u][2] = 0
    for idx in graph[u]:
        edge = e[idx]
        v = edge.u ^ edge.v ^ u
        if v == fa:
            continue
        w = edge.b - edge.w
        dfs(v, u)

        if dp[v][1] + w > dp[u][1]:
            dp[u][2] = dp[u][1]
            dp[u][1] = dp[v][1] + w
        elif dp[v][1] + w > dp[u][2]:
            dp[u][2] = dp[v][1] + w

        dp[u][0] = max(dp[u][0], dp[v][0], dp[u][1] + dp[u][2])
```

1. 参数解释

- `u`：当前遍历的节点。
- `fa`：`u` 的父节点，用于避免回到父节点，造成死循环。

2. 初始化 `dp[u]`

`dp[u]` 是动态规划数组，定义如下：

- `dp[u][0]`：表示以 `u` 为根的子树中可以形成的最大路径值。
- `dp[u][1]` 和 `dp[u][2]`：分别表示以 `u` 为根的子树中可以向下延伸的最大和次大路径值。

初始时，`dp[u]` 被重置为 `[0, 0, 0]`，表示从 `u` 出发还未计算任何路径。

3. 遍历 `u` 的所有邻边

`graph[u]` 存储了与 `u` 相连的边的索引，遍历这些边时：

- **`edge.u ^ edge.v ^ u` 通过异或运算获取当前边的另一个端点 `v`。**
- **如果 `v` 是父节点 `fa`，说明 `v` 是从 `u` 回去的节点，跳过，避免重复遍历。**
- **计算 `w = edge.b - edge.w`，表示当前边的权重差值（或路径值）。**

4. 递归调用 `dfs(v, u)`

对节点 `v` 进行递归调用，继续深度优先搜索其子节点。递归返回后，`dp[v]` 存储了 `v` 相关的路径信息。

5. 更新 `dp[u][1]` 和 `dp[u][2]`

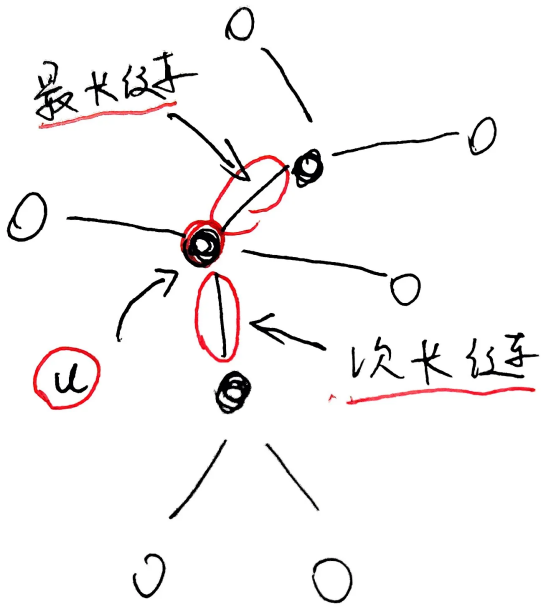
- **判断 `dp[v][1] + w` 是否大于 `dp[u][1]`（当前最大路径值），如果是，则更新 `dp[u][1]` 和 `dp[u][2]`（最大和次大路径值）。**
- **否则，判断是否大于 `dp[u][2]`，如果是，则更新 `dp[u][2]`（次大路径值）。**

6. 更新 `dp[u][0]`

- **`dp[u][0]` 的值是当前节点 `u` 的最大路径值、子节点 `v` 的最大路径值，以及通过 `dp[u][1] + dp[u][2]` 形成的路径中的最大值。**

为什么要用 dp[u][2] -- 次长边

如下图，对于当前节点 u，可能图中的最长链包含了 u，但实际上 u 不是端点。


最长链子
u
次长链子

完整代码 Python

```python
# 超时 8分
# 超时 8分
# 超时 8分
# 超时 8分
# 超时 8分
# 超时 8分
# 超时 8分
# 超时 8分
# 超时 8分
# 超时 8分
# 超时 8分
# 超时 8分
# 超时 8分
# 超时 8分

import sys
from collections import defaultdict

sys.setrecursionlimit(200000)

N = 200005
mod = 998244353

class Edge:
    def __init__(self, u, v, w, b):
        self.u = u
        self.v = v
        self.w = w
        self.b = b

graph = defaultdict(list)
e = [None] * N
dp = [[0] * 3 for _ in range(N)]

def dfs(u, fa):
    dp[u][0] = dp[u][1] = dp[u][2] = 0
    for idx in graph[u]:
        edge = e[idx]
        v = edge.u ^ edge.v ^ u
        if v == fa:
            continue
        w = edge.b - edge.w
        dfs(v, u)

        if dp[v][1] + w > dp[u][1]:
            dp[u][2] = dp[u][1]
            dp[u][1] = dp[v][1] + w
        elif dp[v][1] + w > dp[u][2]:
            dp[u][2] = dp[v][1] + w

        dp[u][0] = max(dp[u][0], dp[v][0], dp[u][1] + dp[u][2])

def work():
    n, m = map(int, input().split())
```

```
    tot = 0

    for _ in range(n - 1):
        tot += 1
        u, v, w, b = map(int, input().split())
        e[tot] = Edge(u, v, w, b)
        graph[u].append(tot)
        graph[v].append(tot)

    dfs(1, -1)
    print(dp[1][0])

    for _ in range(m):
        x, y = map(int, input().split())
        e[x].w = y
        dfs(1, -1)
        print(dp[1][0])

if __name__ == "__main__":
    # Reading input directly from the standard input
    work()

'''
5 3
1 2 6 4
2 3 2 1
3 4 5 3
3 5 8 5
3 2
4 3
1 1

'''
```