Cat-Cat-Carousel 题解

Squareroot2-Bit

一、建图

这是一个图，拿到题目第一时间肯定是建图。那么需要考察节点包含的基本信息。

首先，所有节点出度均为1，节点需要包括**下一个节点的编号**。本题只需要建单向链接即可。

其次，在所有节点出度均为1的有向图中，所有节点要么在简单环上，要么通过数个节点能够到达一个简单环。所以，通过给每个节点打**标记**，用以判断一个节点在环上，还是在环外。然后还要通过一个成员去记录这个节点**对应的环的编号**，可以证明，任何一个节点都不可能同时对应多个环。。

再次，既然要记录环的信息，需要构造环的数据结构，应当包括：编号最小节点（入口）、编号最大节点、猫猫的喜爱程度。

最后，构建图的相关代码如下：

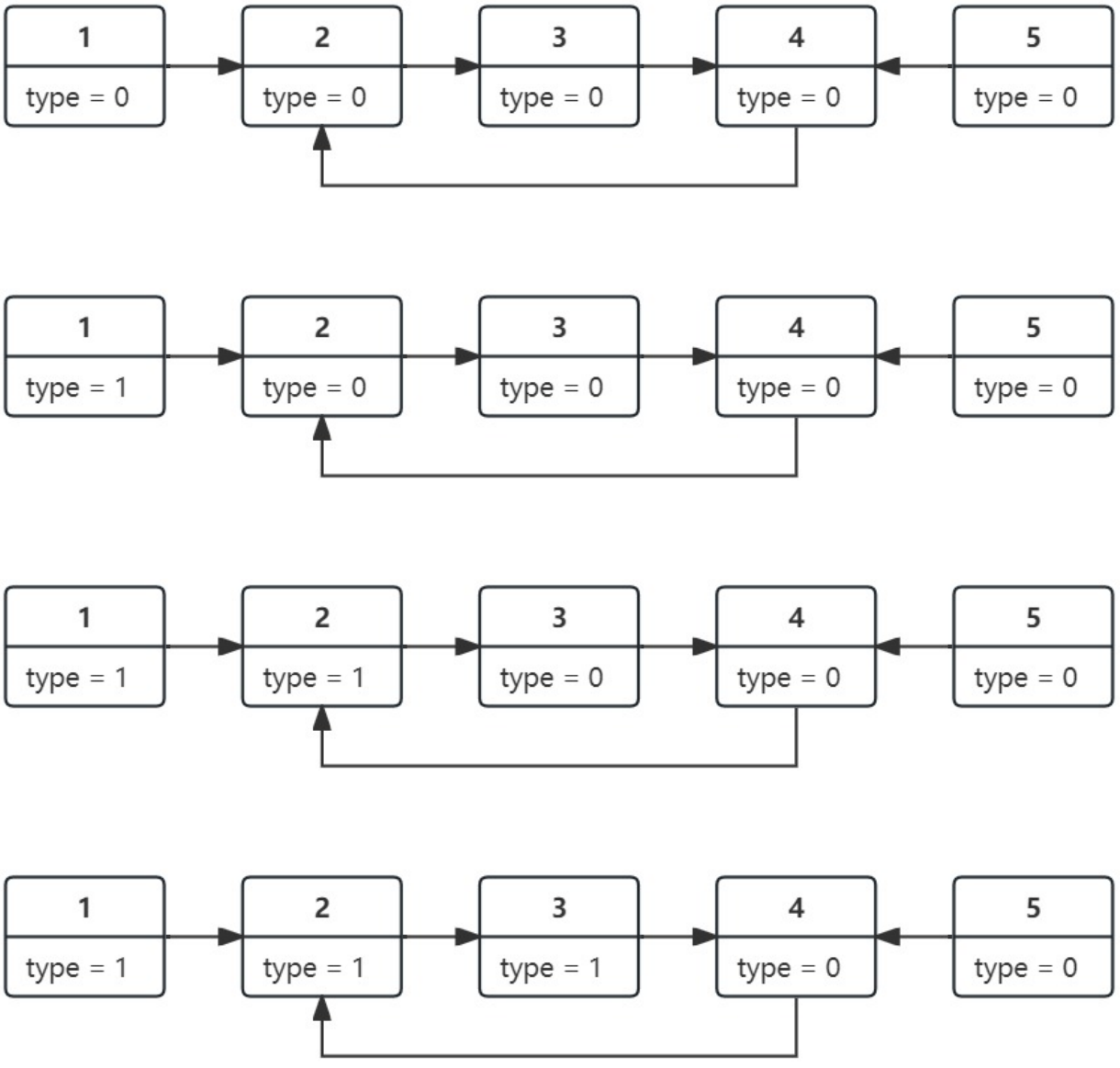
```
struct Node {
    int to;
    int type;    // 0 未标记,1 暂时标记,2 环外,3 环上
    int ring_index;
};
struct Ring {
    int min, max;
    int love;
};
Node nodes[N];
...
int main() {
    int n;
    cin >> n;
    int index, to;
    for (int i = 0; i < n; ++i) {
        cin >> index >> to;
        nodes[index].to = to;
        nodes[index].type = 0;
    }
    ...
}
```

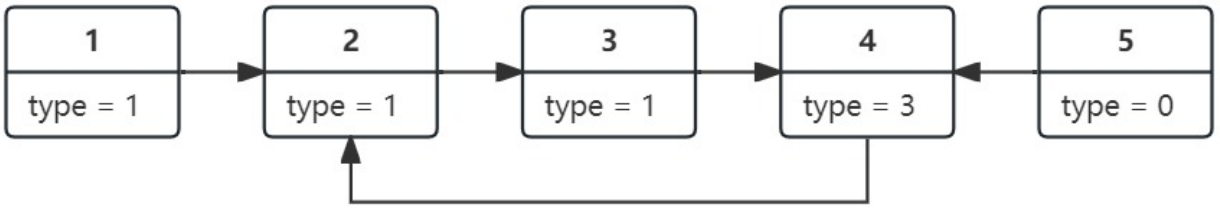
二、找环

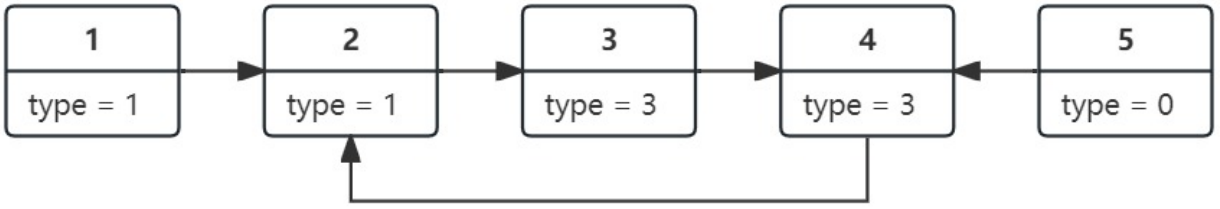
因为每个节点出度为1，bfs和dfs没有区别，这里用dfs函数表示搜索。
依次对所有的节点调用dfs函数，若：

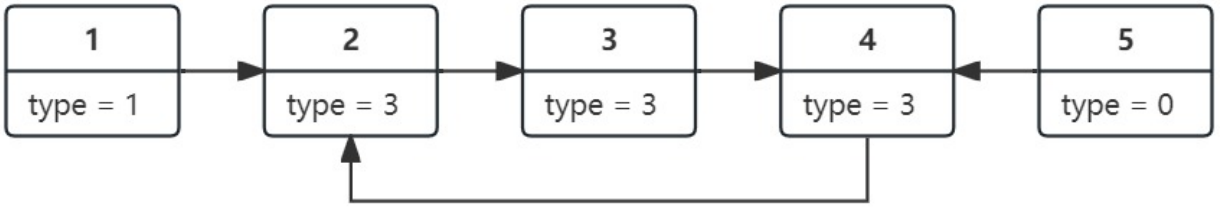
- 该节点尚未标记（0），先为其做暂时标记（1），然后对下一个节点调用dfs函数。若返回值为：
  -- 0，则证明该节点通向一个已经成型的环。将该节点标记为在环外（2），返回0。
  -- 本节点编号，则证明该节点在环上，且是本次搜索中进入环的第一个节点，在该节点之前的节点都在环外。将该节点标记为在环上（3），返回0。
  -- 其他，则证明该节点在环上。将该节点标记为在环上（3），返回dfs函数的返回值。
- 该节点已做暂时标记（1），证明新找到了一个环，返回该节点的编号。
- 该节点已被标记为在环外（2）或环上（3），证明找到了一个旧的环，返回0。

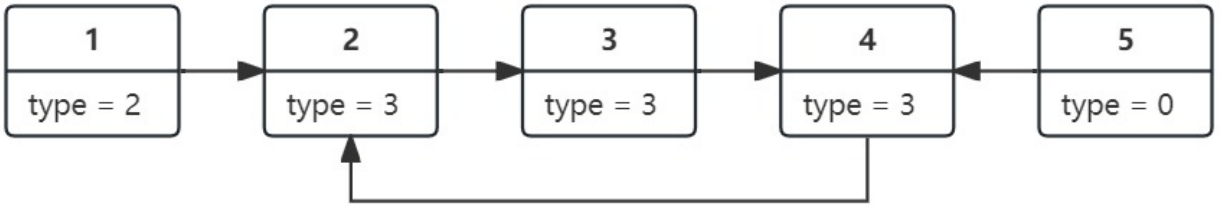
如图所示：











相关代码如下：

```
int dfs(int i) {
    if (nodes[i].type == 0) {
        nodes[i].type = 1;
        int d = dfs(nodes[i].to);
        if (d == 0) {   //该节点不在环上
            nodes[i].type = 2;
            nodes[i].ring_index = nodes[nodes[i].to].ring_index;
            return 0;
        } else {        //该节点在环上
            nodes[i].type = 3;
            nodes[i].ring_index = nodes[nodes[i].to].ring_index;
            if (i == d)     //该节点为入环节点
                return 0;
            else            //该节点不是入环节点
                return d;
        }
```

```
    } else if (nodes[i].type == 1) {
        rings[ring_index].min = rings[ring_index].max = i;
        rings[ring_index].love = 0;
        nodes[i].ring_index = ring_index;
        ring_index++;
        return i;
    } else return 0;         // if (nodes[i].type == 2 || nodes[i].type ==
3)
}
int main() {
    ...
    for (int i = 1; i <= n; ++i) {
            dfs(i);
        }
        ...
}
```

三、计算猫猫的喜爱程度

猫猫对于一个“旋转木马”的喜爱程度定义为：“旋转木马”简单环上的节点数量减去不在这个“旋转木马”简单环上，但是存在路径能到达这个“旋转木马”简单环的节点数量。

因此，每个节点对它对应的环的猫猫的喜爱程度贡献度为：

- 在环上：1
- 在环外：-1

遍历每个节点，计算每个环的猫猫的喜爱程度、编号最小节点和编号最大节点。

遍历每个环，找出猫猫最喜欢的“旋转木马”简单环，并输出。

相关代码如下：

```
...
Ring rings[N];
int ring_index = 0;
int main() {
    ...
    for (int i = 1; i <= n; ++i) {
        if(nodes[i].type == 2)
                rings[nodes[i].ring_index].love--;
            if(nodes[i].type == 3) {
                rings[nodes[i].ring_index].love++;
                if (rings[nodes[i].ring_index].min == 0)
                            rings[nodes[i].ring_index].min = i;
                        rings[nodes[i].ring_index].max = i;
            }
```

```
    }
    int r = 0;
    for (int i = 1; i < ring_index; ++i) {
        if (rings[i].love > rings[r].love ||
            (rings[i].love == rings[r].love && rings[i].max >
rings[r].max))
            r = i;
    }
    int p = rings[r].min;
    do {
        cout << p;
        p = nodes[p].to;
        if (p != rings[r].min)
            cout << " ";
    } while (p != rings[r].min);
    return 0;
}
```

四、代码优化

发现在找环和计算猫猫喜爱程度时各自遍历了一遍节点，将其结合得到：

```
#include <iostream>
#define N 100005

using namespace std;
struct Node {
    int to = 0;
    int type = 0;   // 0 未标记,1 暂时标记,2 环外,3 环上
    int ring_index = 0;
};
struct Ring {
    int min, max;
    int love;
};
Node nodes[N];
Ring rings[N];
int ring_index = 0;

// 返回入环节点序号
int dfs(int i) {
    if (nodes[i].type == 0) {   // 该节点尚未标记（0）
        nodes[i].type = 1;              //先为其做暂时标记（1）
        int d = dfs(nodes[i].to);   //然后对下一个节点调用dfs函数。
        if (d == 0) {   //该节点在环外，使得该环的love--
            nodes[i].type = 2;  //将该节点标记为在环外（2）
            nodes[i].ring_index = nodes[nodes[i].to].ring_index;
            rings[nodes[i].ring_index].love--;
```

```
            return 0;
        } else {        //该节点在环上，使得该环的love++
            nodes[i].type = 3;  //将该节点标记为在环上（3）
            nodes[i].ring_index = nodes[nodes[i].to].ring_index;
            if (i < rings[nodes[i].ring_index].min) //更新该环上的编号最小节点
                rings[nodes[i].ring_index].min = i;
            if (i > rings[nodes[i].ring_index].max) //更新该环上的编号最大节点
                rings[nodes[i].ring_index].max = i;
            rings[nodes[i].ring_index].love++;
            if (i == d)     //该节点为入环节点
                return 0;
            else            //该节点不是入环节点
                return d;
        }
    } else if (nodes[i].type == 1) {    // 该节点已做暂时标记（1），证明新找到了一个环，返回该节点的编号。
        rings[ring_index].min = rings[ring_index].max = i;  // 初始化该环的编号最小节点、编号最大节点
        rings[ring_index].love = 0;     // 初始化该环的猫猫喜爱程度
        nodes[i].ring_index = ring_index;
        ring_index++;
        return i;
    } else  // 该节点已被标记为在环外（2）或环上（3），证明找到了一个旧的环，返回0。
        return 0;
}

int main() {
    int n;
    cin >> n;
    int index, to;
    for (int i = 0; i < n; ++i) {   //建表
        cin >> index >> to;
        nodes[index].to = to;
        nodes[index].type = 0;
    }
    for (int i = 1; i <= n; ++i) {  //找环，并计算猫猫的喜爱程度
        dfs(i);
    }
    int r = 0;
    for (int i = 1; i < ring_index; ++i) {  // 寻找猫猫最喜欢的“旋转木马”
        if (rings[i].love > rings[r].love ||
            (rings[i].love == rings[r].love && rings[i].max > rings[r].max))
            r = i;
    }
    int p = rings[r].min;
    do {            // 输出猫猫最喜欢的“旋转木马”
        cout << p;
        p = nodes[p].to;
        if (p != rings[r].min)
```

```
            cout << " ";
    } while (p != rings[r].min);
    return 0;       // return 0
}
```