```
#include <iostream>
#include <queue>

using namespace std;

struct Node{
    int ID;
    int X;
    int Y;
    char mark;
    struct Node *parent;
    struct Node *first;
    struct Node *second;
};

struct Node* head = NULL;
char check[3][3] = {0};
int move = 0;
int win = 0;

//pre-order
void printnode(struct Node* ptr){
    if(!ptr)return;
    printnode(ptr->first);
    printnode(ptr->second);
//  cout << "ID: " << ptr->ID ;
    cout << ptr->X << " ";
    cout << ptr->Y << " " ;
    cout << ptr->mark << endl;
}

//post-order
void printnode2(struct Node* ptr){
    if(!ptr)return;
    cout << "ID: " << ptr->ID << " ";
    cout << "X:" << ptr->X << " ";
    cout << "Y:" << ptr->Y << " " ;
    cout << "Mark: " << ptr->mark << endl;
    printnode2(ptr->first);
    printnode2(ptr->second);
}
```

```
// level-order traversal
void traversal(Node* root)
{
    queue<Node*> q;
    q.push(root);
    while (!q.empty())
    {
        Node* p = q.front(); q.pop();
        cout << "ID: " << p->ID << " ";
        cout << "X:" << p->X << " ";
        cout << "Y:" << p->Y << " " ;
        cout << "Mark: " << p->mark << endl;
        if (p->first)  q.push(p->first);
        if (p->second) q.push(p->second);
    }
}

void checkwin(struct Node* ptr){
    int i;
    if(win == 0){
    if(!ptr){
        if((check[0][0]==check[0]
[1])&&(check[0][1]==check[0][2])&&(check[0]
[0]!=0)||
        (check[1][0]==check[1][1])&&(check[1]
[1]==check[1][2])&&(check[1][0]!=0)||
        (check[2][0]==check[2][1])&&(check[2]
[1]==check[2][2])&&(check[2][0]!=0)||
        (check[0][0]==check[1][0])&&(check[1]
[0]==check[2][0])&&(check[0][0]!=0)||
        (check[1][0]==check[1][1])&&(check[1]
[1]==check[2][1])&&(check[1][0]!=0)||
        (check[2][0]==check[2][1])&&(check[2]
[1]==check[2][2])&&(check[2][0]!=0)||
        (check[0][0]==check[1][1])&&(check[1]
[1]==check[2][2])&&(check[0][0]!=0)||
        (check[2][0]==check[1][1])&&(check[1]
[1]==check[0][2])&&(check[2][0]!=0)){
            cout << "Win" << endl;
            for(int j=0; j<3; j++){
                for(i=0; i<2; i++){
                    if(check[i][j]==0){
```

```
                    cout << "_";
                }else{
                    cout << check[i][j];
                }
                cout << " ";
            }
            if(check[i][j]==0){
                cout << "_";
            }else{
                cout << check[i][j];
            }
            cout << endl;
        }
    win = 1;
    }
    return;
  }
  check[ptr->X][ptr->Y] = ptr->mark;
  checkwin(ptr->first);
  checkwin(ptr->second);
  check[ptr->X][ptr->Y] = 0;
  }
}

struct Node* foundptr;
int finaldepth = 0;

void findID(struct Node* ptr, int id){
    if(ptr == NULL){
        return;
    }
    if(ptr->ID == id){
        foundptr = ptr;
        return;
    }
    findID(ptr->first, id);
    findID(ptr->second, id);
}

bool findID2(struct Node* ptr, int id){
    if(ptr == NULL) return false;
    if(ptr->ID == id){
        return true;
    }
    int rs1 = findID2(ptr->first, id);
    int rs2 = findID2(ptr->second, id);
    return rs1||rs2;
}

int dis[100] = {0};

int findDepth(struct Node* ptr){
    int depth = 0;
    while(ptr != head){
        dis[depth] = ptr->ID;
        depth++;
        ptr = ptr->parent;
    }
    depth++;
    return depth;
}

void resetdis(int s[]){
    for(int i=0; i<100; i++){
        s[i] = 0;
    }
}

int findDis(int a, int b){
    int i=0, end=0, adep, bdep;
    int disa[100]={0};
    int disb[100]={0};
    findID(head, a);
    adep = findDepth(foundptr);
    for(i=0; i<adep-1; i++){
        disa[i] = dis[adep-2-i];
    }
    resetdis(dis);
    findID(head, b);
    bdep = findDepth(foundptr);
    for(i=0; i<bdep-1; i++){
        disb[i] = dis[bdep-2-i];
    }
    resetdis(dis);
    int dist=0;
    for(i=0; i<adep-1; i++){
        if(disa[i] == disb[i]){
            bdep--;
        }else{
            dist++;
        }
    }
    dist = dist + bdep-1;
    return dist;
}

void insert(int new_ID, int new_X, int new_Y,
char new_mark, int new_parent){
    struct Node* new_node = new Node;
    new_node->ID = new_ID;
    new_node->X = new_X;
    new_node->Y = new_Y;
    new_node->mark = new_mark;
    struct Node* ptr = head;

    findID(head, new_parent);

    new_node->parent = foundptr;

    if(foundptr->first==NULL){
        foundptr->first = new_node;
    }else{
        foundptr->second = new_node;
    }
}

void insert2(int new_ID, int new_X, int
new_Y, char new_mark, int new_parent, int
child){
    struct Node* new_node = new Node;
    new_node->ID = new_ID;
    new_node->X = new_X;
    new_node->Y = new_Y;
    new_node->mark = new_mark;

    findID(head, new_parent);
    new_node->parent = foundptr;
    cout << foundptr->first->ID << endl;

    if(foundptr->first->ID == child){
        foundptr->first->parent = new_node;
        new_node->first = foundptr->first;
        foundptr->first = new_node;
    }else{
        foundptr->second->parent == new_node;
        new_node->first = foundptr->second;
        foundptr->second = new_node;
    }

}

void delnode(int id){
    findID(head, id);
    struct Node* ptr = foundptr->parent;
    if(ptr->first == foundptr){
        ptr->first = foundptr->first;
    }else{
        ptr->second = foundptr->first;
    }
    foundptr->first->parent = ptr;
    delete foundptr;
}


int main(){
    int commandnum;
    int inID, inX, inY, inParent;
    char inTurn;
    int i;


    cin >> commandnum;
    cin >> inID >> inParent >> inX >> inY >>
inTurn;

    struct Node* new_node = new Node;
    new_node->ID = inID;
    new_node->X = inX;
    new_node->Y = inY;
    new_node->mark = inTurn;
    new_node->parent = new_node;
    head = new_node;

//  cout << head->ID << head->X << head->Y <<
path[0][0] << head->mark<< path[0][1]<< endl;

    for(;commandnum>1; commandnum--){
        cin >> inID >> inParent >> inX >> inY
>> inTurn;
//      cout <<" "<< inID <<" "<< inParent
<<" "<< inX <<" "<<inY <<" "<< inTurn <<
endl;
        insert(inID, inX, inY, inTurn,
inParent);
    }

//  printnode(head);
    checkwin(head);

    if(win == 0){
        cout << "Tie" << endl;
        printnode(head);
    }

//  traversal(head);
/*
    insert2(12, 1, 0, 'O', 8, 11);

    for(i=0; i<13; i++){
        findID(head, i);
        cout << i << "'s depth is " <<
findDepth(foundptr) << endl;
    }

    cout << "14 is in tree? " <<
findID2(head, 14) << endl;
    cout << "8 is in tree? " << findID2(head,
8) << endl;
    cout << "12 is in tree? " <<
findID2(head, 12) << endl;

    printnode2(head);

    delnode(8);
    cout << "14 is in tree? " <<
findID2(head, 14) << endl;
    cout << "8 is in tree? " << findID2(head,
8) << endl;
    cout << "12 is in tree? " <<
findID2(head, 12) << endl;

    printnode2(head);

    cout << findDis(12, 6) << endl;
    cout << findDis(12, 4) << endl;
*/
}
```