1215：迷宫

题目描述

一天Extense在森林里探险的时候不小心走入了一个迷宫，迷宫可以看成是由n×n的格点组成，每个格点只有2种状态，.和#，前者表示可以通行后者表示不能通行。同时当Extense处在某个格点时，他只能移动到东南西北(或者说上下左右)四个方向之一的相邻格点上，Extense想要从点A走到点B，问在不走出迷宫的情况下能不能办到。如果起点或者终点有一个不能通行(为#)，则看成无法办到。

输入格式

第1行是测试数据的组数k，后面跟着k组输入。每组测试数据的第1行是一个正整数n(1≤n≤100)，表示迷宫的规模是n×n的。接下来是一个n×n的矩阵，矩阵中的元素为.或者#。再接下来一行是4个整数ha,la,hb,lb，描述A处在第ha行，第la列，B处在第hb行，第lb列。注意到ha,la,hb,lb全部是从0开始计数的。

输出格式

k行，每行输出对应一个输入。能办到则输出“YES”，否则输出“NO”。

输入样例

```
2
3
.##
..#
#..
0 0 2 2
5
.....
###.#
..#..
###..
...#.
0 0 4 0
```

输出样例

```
YES
NO
```

解析

实际上这道题由于是只求能否达到终点，因此使用广搜是更加合适的。但是题目分类给将它放到了深搜中，是刻意为之。因此，我们在使用深搜中一定要注意剪枝，避免超限。

对于该题的减枝办法就是不要回溯！

怎么理解呢？观察下面的常规深搜代码。

```
//b为标记数组
b[x] = 1; //标记已经走过
dfs(); //深搜
b[x] = 0; //回溯并还原标记数组
```

我们要知道这种写法对应的题目要求一般是这样的：**“求总方案数”**、**“列出所有方案”**等。这种就要求我们穷尽所有情况。比如我们一条路走不通了，那么我们回去，而且在我们回去之后，我们以后还是可以经过这条路的，因为我们已经在回溯的过程中还原了标记数组，形象来说就是我们抹除了我们走过这里的记忆，所以以后有可能还会来这儿。

而对于本题，我们则应该这样编写：

```
//b为标记数组
b[x] = 1; //标记已经走过
dfs(); //深搜
```

这种对应的题目一般是“是否能走到”、“有无一种方案能够到达”等。因为我们的目的是找到终点，所以不需要像上面那样穷尽所有的方案，我们只要找到了，那么就停止搜索了。比如我们一条路走不通了，那么我们回去，并且在回去的路上，我们用土把这些走过的地方填上，以后就不走这儿了（这条路找不到终点那我以后还走这儿干嘛！？）。直到我们找到终点，那么搜索就停止了。这样写之所以不会超时，是因为我们把那些走过的路都填上了 ，以后就不会再走了，降低了复杂度。

编码

```
#include <bits/stdc++.h>

using namespace std;

//四个移动方向
int const FORWARD_NUM = 4;


int n;
int ex, ey; //终点

char Map[101][101]; //地图
//记录当前节点是否行走过
```

```
bool Vis[101][101];

//四个个移动方向
int Forward[4][2] = { //八方向，二维（行，列）
       {-1, 0},//上
       {1,  0},//下
       {0,  -1},//左
       {0,  1},//右
};
//判断指定的目标点是否可以行走
bool Check(int x, int y) {
    //1、是否越界
    if (x >= 0 && y >= 0 && x < n && y < n) {
        //2、未访问过
       if (!Vis[x][y]) {
           //3、路径可以通行
           if (Map[x][y] == '.') {
               return true;
           }
       }
    }
    return false;
}
//标记我是否到达终点
bool flag = false;
//深搜代码
void Dfs(int sx, int sy) {
    if (flag)
        return;
    //到达终点
    if ((sx == ex) && (sy == ey)) {
        flag = true;
        return;
    }
    for (int i = 0; i < FORWARD_NUM; ++i) {
        //新的xy坐标
       int newX = sx + Forward[i][0];
       int newY = sy + Forward[i][1];
       if (Check(newX, newY)) {
           //标记为已经行走
         Vis[newX][newY] = true;
           //执行深搜
         Dfs(newX, newY);

           //不需要回溯，提高效率
       }
    }
}
//重置
```

```
void Reset() {
    flag = false;
    memset(Vis, false, sizeof(Vis));
}

void Read() {
    int T;  //测试组数
   cin >> T;
    int sx, sy;
    for (int i = 0; i < T; ++i) {
        //读入地图规模
       cin >> n;
        //读入地图
       for (int j = 0; j < n; ++j) {
            cin >> Map[j];
        }
        cin >> sx >> sy >> ex >> ey;
        //重置代码
       Reset();
        //标记起点
       Vis[sx][sy] = true;
        Dfs(sx, sy);
        if (flag) {
            cout << "YES" << endl;
        } else {
            cout << "NO" << endl;
        }
    }
}
//1、组间重置 方案数，访问列表
//2、终止条件的边界判断
//3、走过的点标记为true，回溯的点标记为false;
int main() {
    Read();
    return 0;
}
```