**1. I NEED A OFFER!**

**Problem Description**

Speakless很早就想出国，现在他已经考完了所有需要的考试，准备了所有要准备的材料，于是，便需要去申请学校了。要申请国外的任何大学，你都要交纳一定的申请费用，这可是很惊人的。Speakless没有多少钱，总共只攒了n万美元。他将在m个学校中选择若干的（当然要在他的经济承受范围内）。每个学校都有不同的申请费用a（万美元），并且Speakless估计了他得到这个学校offer的可能性b。不同学校之间是否得到offer不会互相影响。“I NEED A OFFER”，他大叫一声。帮帮这个可怜的人吧，帮助他计算一下，他可以收到至少一份offer的最大概率。（如果Speakless选择了多个学校，得到任意一个学校的offer都可以）。

**Input**

输入有若干组数据，每组数据的第一行有两个正整数n,m(0<=n<=10000,0<=m<=10000)  
后面的m行，每行都有两个数据ai(整型),bi(实型)分别表示第i个学校的申请费用和可能拿到offer的概率。  
输入的最后有两个0。

**Output**

每组数据都对应一个输出，表示Speakless可能得到至少一份offer的最大概率。用百分数表示，精确到小数点后一位。

**Sample Input**

10 3

4 0.1

4 0.2

5 0.3

0 0

**Sample Output**

44.0%

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        int n = input.nextInt();

        int m = input.nextInt();

        int[] w = new int[m];

        double[] p = new double[m];

        double[] dp = new double[n+1];

        dp[0]=0.0;

        for(int i=0;i<m;i++){

            w[i] = input.nextInt();

            p[i] = input.nextDouble();

        }

        for(int i=0;i<m;i++){

            for(int j=n;j>=w[i];j--){

                dp[j] = Math.max(dp[j],dp[j-w[i]]+p[i]-p[i]\*dp[j-w[i]]);

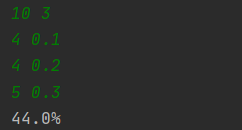
            }

        }

        System.out.println(String.format("%.1f%%",dp[n]\*100));

    }

}



# 2.Rescue

**Problem Description**

Angel was caught by the MOLIGPY! He was put in prison by Moligpy. The prison is described as a N \* M (N, M <= 200) matrix. There are WALLs, ROADs, and GUARDs in the prison.  
  
Angel's friends want to save Angel. Their task is: approach Angel. We assume that "approach Angel" is to get to the position where Angel stays. When there's a guard in the grid, we must kill him (or her?) to move into the grid. We assume that we moving up, down, right, left takes us 1 unit time, and killing a guard takes 1 unit time, too. And we are strong enough to kill all the guards.  
  
You have to calculate the minimal time to approach Angel. (We can move only UP, DOWN, LEFT and RIGHT, to the neighbor grid within bound, of course.)

**Input**

First line contains two integers stand for N and M.  
  
Then N lines follows, every line has M characters. "." stands for road, "a" stands for Angel, and "r" stands for each of Angel's friend.  
  
Process to the end of the file.

**Output**

For each test case, your program should output a single integer, standing for the minimal time needed. If such a number does no exist, you should output a line containing "Poor ANGEL has to stay in the prison all his life."

**Sample Input**

7 8

#.#####.

#.a#..r.

#..#x...

..#..#.#

#...##..

.#......

........

**Sample Output**

13

import java.util.ArrayDeque;

import java.util.Deque;

import java.util.Scanner;

public class Main {

    private static int n;

    private static int m;

    private static int[][] map;

    private static int[] xc = {1,-1,0,0};

    private static int[] yc = {0,0,1,-1};

    private static int best = Integer.MAX\_VALUE;

    private static int[][] vis;

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        n = input.nextInt();

        m = input.nextInt();

        map = new int[n][m];

        vis = new int[n][m];

        int x = 0;

        int y = 0;

        int targetX = 0;

        int targetY = 0;

        for(int i=0;i<n;i++){

            input = new Scanner(System.in);

            String str = input.nextLine();

            for(int j=0;j<m;j++){

                char s = str.charAt(j);

                if(s=='#'){

                    map[i][j]=-1;

                }

                else if(s=='x'){

                    map[i][j]=-2;

                }

                else if(s=='.'){

                    map[i][j]=0;

                }

                else if(s=='r'){

                    x = i;

                    y = j;

                }

                else if(s=='a'){

                    targetX = i;

                    targetY = j;

                }

            }

        }

        bfs(x,y,targetX,targetY);

        System.out.println(best);

    }

    public static void bfs(int x,int y,int targetX,int targetY){

        Deque<Node> deque = new ArrayDeque<>();

        Node first = new Node();

        first.x = x;

        first.y = y;

        first.length = 0;

        deque.push(first);

        while (!deque.isEmpty()){

            Node node = deque.poll();

            if(node.x==targetX&&node.y==targetY){

                if(node.length<best){

                    best = node.length;

                }

            }

            else {

                if(node.length>best){

                    continue;

                }

                for(int i=0;i<4;i++){

                    int xx = node.x+xc[i];

                    int yy = node.y+yc[i];

                    if(xx>=0&&xx<n&&yy>=0&&yy<m){

                        if(map[xx][yy]==-1){

                            continue;

                        }

                        else if(map[xx][yy]==0&&vis[xx][yy]==0){

                            vis[xx][yy]=1;

                            Node newNode = new Node();

                            newNode.length = node.length+1;

                            newNode.x=xx;

                            newNode.y=yy;

                            deque.push(newNode);

                        }

                        else if(map[xx][yy]==-2&&vis[xx][yy]==0){

                            vis[xx][yy]=1;

                            Node newNode = new Node();

                            newNode.length = node.length+2;

                            newNode.x=xx;

                            newNode.y=yy;

                            deque.push(newNode);

                        }

                    }

                }

            }

        }

    }

}

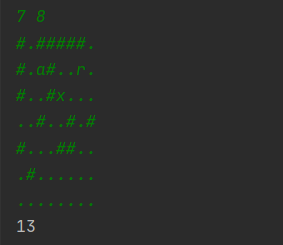
class Node{

    public int x;

    public int y;

    public int length;

}



# 3.哈密顿绕行世界问题

**Problem Description**

一个规则的实心十二面体，它的 20个顶点标出世界著名的20个城市，你从一个城市出发经过每个城市刚好一次后回到出发的城市。

**Input**

前20行的第i行有3个数,表示与第i个城市相邻的3个城市.第20行以后每行有1个数m,m<=20,m>=1.m=0退出.

**Output**

输出从第m个城市出发经过每个城市1次又回到m的所有路线,如有多条路线,按字典序输出,每行1条路线.每行首先输出是第几条路线.然后个一个: 后列出经过的城市.参看Sample output

import java.util.\*;

public class Main {

    private static int[][] city = new int[21][3];

    private static TreeSet<Integer[]> treeSet = new TreeSet<>(new Comparator<Integer[]>() {

        @Override

        public int compare(Integer[] o1, Integer[] o2) {

            for(int i=1;i<= o1.length-1;i++){

                if(o1[i]>o2[i]){

                    return 1;

                }

                else {

                    return -1;

                }

            }

            return 0;

        }

    });

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        for(int i=1;i<=city.length-1;i++){

            city[i][0] = input.nextInt();

            city[i][1] = input.nextInt();

            city[i][2] = input.nextInt();

        }

        int m = input.nextInt();

        bfs(m);

        int x = 1;

        for(Integer[] arr:treeSet){

            System.out.print(x+":"+" ");

            for(int i=1;i<= arr.length-1;i++){

                System.out.print(arr[i]+" ");

            }

            System.out.println(m);

        }

    }

    public static void bfs(int m){

        Deque<Node> deque = new ArrayDeque<>();

        Node first = new Node();

        first.cityId = m;

        first.level = 1;

        deque.push(first);

        while (!deque.isEmpty()){

            Node node = deque.poll();

            if(node.level==20){

                int flag = 0;

                for(int i=0;i<3;i++){

                    if(city[node.cityId][i]==5){

                        flag=1;

                    }

                }

                if(flag==0){

                    continue;

                }

                Integer[] result = new Integer[21];

                Node temp = node;

                int j = 20;

                while (true){

                    if(temp.level==1){

                        result[1] = temp.cityId;

                        break;

                    }

                    result[j] = temp.cityId;

                    temp=temp.father;

                    j--;

                }

                treeSet.add(result);

            }

            else {

                for(int i=0;i<3;i++){

                    if(check(node,city[node.cityId][i])){

                        Node newNode = new Node();

                        newNode.cityId = city[node.cityId][i];

                        newNode.level = node.level+1;

                        newNode.father = node;

                        deque.push(newNode);

                    }

                }

            }

        }

    }

    public static boolean check(Node node,int x){

        Node temp = node;

        while(true){

            if(temp.level==1){

                if(x== temp.cityId){

                    return false;

                }

                break;

            }

            if(x == temp.cityId){

                return false;

            }

            temp = temp.father;

        }

        return true;

    }

}

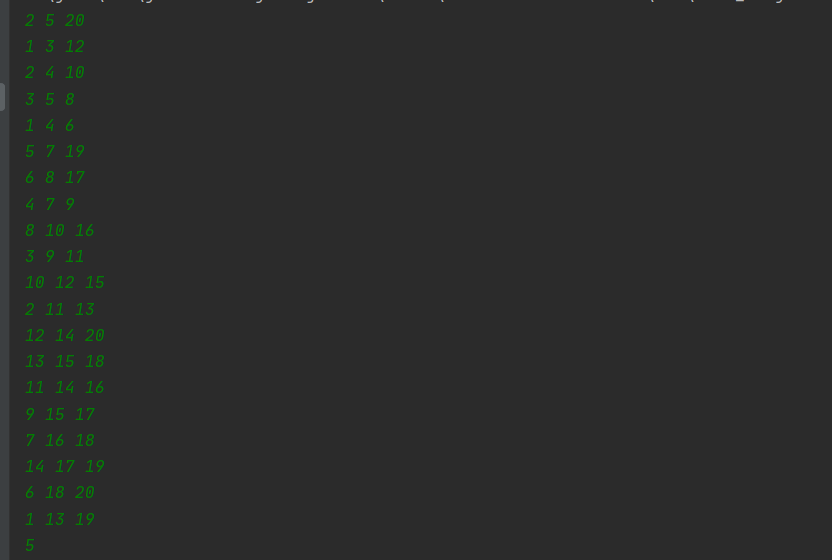
class Node{

    public int cityId;

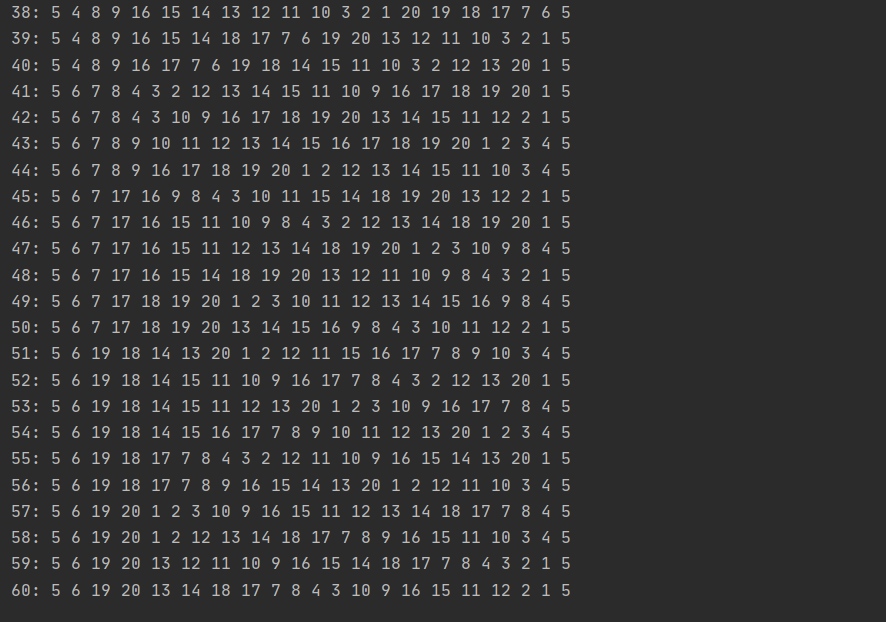
    public int level;

    public Node father;

}



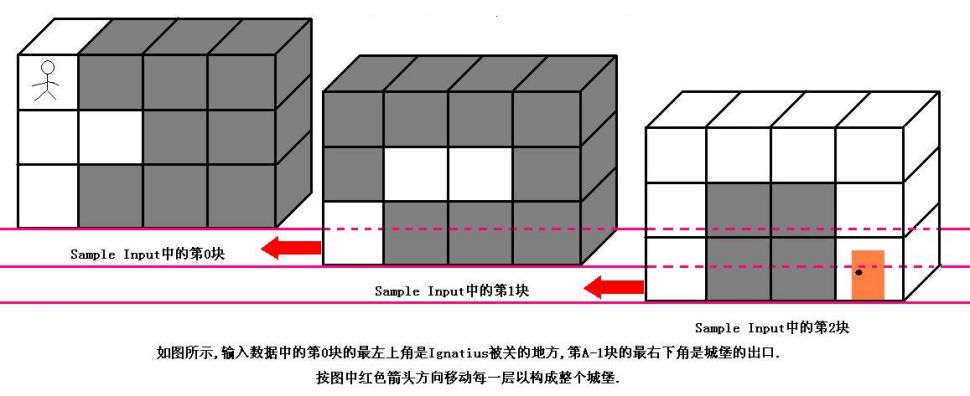




**4. 胜利大逃亡**

**Problem Description**

Ignatius被魔王抓走了,有一天魔王出差去了,这可是Ignatius逃亡的好机会.  
  
魔王住在一个城堡里,城堡是一个A\*B\*C的立方体,可以被表示成A个B\*C的矩阵,刚开始Ignatius被关在(0,0,0)的位置,离开城堡的门在(A-1,B-1,C-1)的位置,现在知道魔王将在T分钟后回到城堡,Ignatius每分钟能从一个坐标走到相邻的六个坐标中的其中一个.现在给你城堡的地图,请你计算出Ignatius能否在魔王回来前离开城堡(只要走到出口就算离开城堡,如果走到出口的时候魔王刚好回来也算逃亡成功),如果可以请输出需要多少分钟才能离开,如果不能则输出-1.



**Input**

输入数据的第一行是一个正整数K,表明测试数据的数量.每组测试数据的第一行是四个正整数A,B,C和T(1<=A,B,C<=50,1<=T<=1000),它们分别代表城堡的大小和魔王回来的时间.然后是A块输入数据(先是第0块,然后是第1块,第2块......),每块输入数据有B行,每行有C个正整数,代表迷宫的布局,其中0代表路,1代表墙.(如果对输入描述不清楚,可以参考Sample Input中的迷宫描述,它表示的就是上图中的迷宫)  
  
特别注意:本题的测试数据非常大,请使用scanf输入,我不能保证使用cin能不超时.在本OJ上请使用Visual C++提交.

import java.util.ArrayDeque;

import java.util.Deque;

import java.util.Scanner;

public class Main {

    private static int a;

    private static int b;

    private static int c;

    private static int[][][] map;

    private static int[][][] vis;

    private static int maxTime;

    private static int[] xc = {1,-1,0,0,0,0};

    private static int[] yc = {0,0,1,-1,0,0};

    private static int[] zc = {0,0,0,0,1,-1};

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        int v = input.nextInt();

        a = input.nextInt();

        b = input.nextInt();

        c = input.nextInt();

        maxTime = input.nextInt();

        map = new int[a+1][b+1][c+1];

        vis = new int[a+1][b+1][c+1];

        for(int i=1;i<=a;i++){

            for (int j = 1; j <= b; j++) {

                for (int k = 1; k <= c ; k++) {

                    map[i][j][k] = input.nextInt();

                }

            }

        }

        bfs();

    }

    public static void bfs(){

        Deque<Node> deque = new ArrayDeque<>();

        Node first = new Node();

        first.x = 1;

        first.y = 1;

        first.z = 1;

        first.times = 0;

        deque.push(first);

        while (!deque.isEmpty()){

            Node node = deque.poll();

            if(node.x==a&&node.y==b&&node.z==c){

                if(node.times<=maxTime){

                    System.out.println(node.times);

                }

                else {

                    System.out.println(-1);

                }

                break;

            }

            else {

                if(node.times>=maxTime){

                    continue;

                }

                for(int i=0;i<6;i++){

                    for (int j = 0; j < 6; j++) {

                        for (int k = 0; k < 6; k++) {

                            int xx = node.x+xc[i];

                            int yy = node.y+yc[i];

                            int zz = node.z+zc[i];

                            if(xx>=1&&xx<=a&&yy>=1&&yy<=b&&zz>=1&&zz<=c){

                                if(vis[xx][yy][zz]==0&&map[xx][yy][zz]==0){

                                    vis[xx][yy][zz]=1;

                                    Node newNode = new Node();

                                    newNode.times = node.times+1;

                                    newNode.x = xx;

                                    newNode.y = yy;

                                    newNode.z = zz;

                                    deque.push(newNode);

                                }

                            }

                        }

                    }

                }

            }

        }

    }

}

class Node{

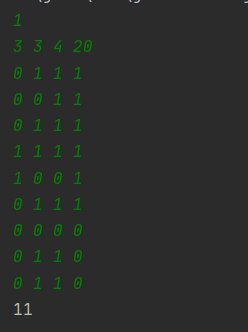
    public int x;

    public int y;

    public int z;

    public int times;

}



**5. A计划**

**Problem Description**

可怜的公主在一次次被魔王掳走一次次被骑士们救回来之后，而今，不幸的她再一次面临生命的考验。魔王已经发出消息说将在T时刻吃掉公主，因为他听信谣言说吃公主的肉也能长生不老。年迈的国王正是心急如焚，告招天下勇士来拯救公主。不过公主早已习以为常，她深信智勇的骑士LJ肯定能将她救出。  
现据密探所报，公主被关在一个两层的迷宫里，迷宫的入口是S（0，0，0），公主的位置用P表示，时空传输机用#表示，墙用\*表示，平地用.表示。骑士们一进入时空传输机就会被转到另一层的相对位置，但如果被转到的位置是墙的话，那骑士们就会被撞死。骑士们在一层中只能前后左右移动，每移动一格花1时刻。层间的移动只能通过时空传输机，且不需要任何时间。

**Input**

输入的第一行C表示共有C个测试数据，每个测试数据的前一行有三个整数N，M，T。 N，M迷宫的大小N\*M（1 <= N,M <=10)。T如上所意。接下去的前N\*M表示迷宫的第一层的布置情况，后N\*M表示迷宫第二层的布置情况。

**Output**

如果骑士们能够在T时刻能找到公主就输出“YES”，否则输出“NO”。

import java.util.ArrayDeque;

import java.util.Deque;

import java.util.Scanner;

public class Main {

    private static int[][][] map;

    private static int[][][] vis;

    private static int c;

    private static int n;

    private static int m;

    private static int t;

    private static int[] xc = {1,-1,0,0};

    private static int[] yc = {0,0,1,-1};

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        c = input.nextInt();

        n = input.nextInt();

        m = input.nextInt();

        t = input.nextInt();

        map = new int[2][n][m];

        vis = new int[2][n][m];

        int ti = 0;

        int tx = 0;

        int ty = 0;

        for(int i = 0;i < 2;i++){

            for (int j = 0; j < n; j++) {

                input = new Scanner(System.in);

                String s = input.nextLine();

                for (int k = 0; k < m; k++) {

                    char x = s.charAt(k);

                    if(x=='\*'){

                        map[i][j][k] = -1;

                    }

                    else if(x=='#'){

                        map[i][j][k] = -2;

                    }

                    else if(x=='P'){

                        ti = i;

                        tx = j;

                        ty = k;

                    }

                }

            }

        }

        bfs(ti,tx,ty);

    }

    public static void bfs(int ti,int tx,int ty){

        Deque<Node> deque = new ArrayDeque<>();

        Node first = new Node();

        first.i = 0;

        first.x=0;

        first.y=0;

        first.time=0;

        deque.push(first);

        while (!deque.isEmpty()){

            Node node = deque.poll();

            if(node.i==ti&&node.x==tx&&node.y==ty){

                if(node.time<=t){

                    System.out.println("YES");

                }

                else {

                    System.out.println("NO");

                }

                break;

            }

            else {

                if(node.time>=t){

                    continue;

                }

                for(int i=0;i<4;i++){

                    int xx = node.x+xc[i];

                    int yy = node.y+yc[i];

                    if(xx>=0&&xx<n&&yy>=0&&yy<m){

                        if(map[node.i][xx][yy]==-2&&vis[node.i][xx][yy]==0){

                            vis[node.i][xx][yy]=1;

                            int a = 0;

                            if(node.i==0){

                                a = 1;

                            }

                            if(map[a][xx][yy]==-1){

                                continue;

                            }

                            vis[a][xx][yy]=1;

                            Node newNode = new Node();

                            newNode.i=a;

                            newNode.x=xx;

                            newNode.y=yy;

                            newNode.time= node.time;

                            deque.push(newNode);

                            System.out.println(newNode);

                        }

                        else if(map[node.i][xx][yy]==-1){

                            continue;

                        }

                        else if(map[node.i][xx][yy]==0&&vis[node.i][xx][yy]==0){

                            vis[node.i][xx][yy]=1;

                            Node newNode = new Node();

                            newNode.i= node.i;

                            newNode.x=xx;

                            newNode.y=yy;

                            newNode.time= node.time+1;

                            deque.push(newNode);

                            System.out.println(newNode);

                        }

                    }

                }

            }

        }

    }

}

class Node{

    public int time;

    public int i;

    public int x;

    public int y;

    @Override

    public String toString() {

        return "Node{" +

                "time=" + time +

                ", i=" + i +

                ", x=" + x +

                ", y=" + y +

                '}';

    }

}

