10 reasons why YOU should care about programming competitions

Daniel Epstein, CSE Grad Student

What are Programming Competitions?

Rock, Paper, Scissors is a two player game, where each player simultaneously chooses one of the three items after counting to three. The game typically lasts a pre-determined number of rounds. The player who wins the most rounds wins the game. Given the number of rounds the players will compete, it is your job to determine which player wins after those rounds have been played.

The rules for what item wins are as follows:

- Rock always beats Scissors (Rock crushes Scissors)
- Scissors always beat Paper (Scissors cut Paper)
- Paper always beats Rock (Paper covers Rock)

Input

The first value in the input file will be an integer t ($0 < t < 1000$) representing the number of test cases in the input file. Following this, on a case by case basis, will be an integer n ($0 < n < 100$) specifying the number of rounds of Rock, Paper, Scissors played. Next will be n lines, each with either a capital `R`, `P`, or `S`, followed by a space, followed by a capital `R`, `P`, or `S`, followed by a newline. The first letter is Player 1's choice; the second letter is Player 2's choice.

Output

For each test case, report the name of the player (`` `Player 1' `` or `` `Player 2' ``) that wins the game, followed by a newline. If the game ends up in a tie, print `` `TIE' ``.

Sample Input

```
3
2
R P
S R
3
P P
R S
S R
1
P R
```

Sample Output

```
Player 2
TIE
Player 1
```

Rock, Paper, Scissors is a two player game, where each player simultaneously chooses one of the three items after counting to three. The game typically lasts a pre-determined number of rounds. The player who wins the most rounds wins the game. Given the number of rounds the players will compete, it is your job to determine which player wins after those rounds have been played.

The rules for what item wins are as follows:

- Rock always beats Scissors (Rock crushes Scissors)
- Scissors always beat Paper (Scissors cut Paper)
- Paper always beats Rock (Paper covers Rock)

Input

The first value in the input file will be an integer t ($0 < t < 1000$) representing the number of test cases in the input file. Following this, on a case by case basis, will be an integer n ($0 < n < 100$) specifying the number of rounds of Rock, Paper, Scissors played. Next will be n lines, each with either a capital `R`, `P`, or `S`, followed by a space, followed by a capital `R`, `P`, or `S`, followed by a newline. The first letter is Player 1's choice; the second letter is Player 2's choice.

Output

For each test case, report the name of the player (`` `Player 1' `` or `` `Player 2' ``) that wins the game, followed by a newline. If the game ends up in a tie, print `` `TIE' ``.

Sample Input

```
3
2
R P
S R
3
P P
R S
S R
1
P R
```

Sample Output

```
Player 2
TIE
Player 1
```

Rock, Paper, Scissors is a two player game, where each player simultaneously chooses one of the three items after counting to three. The game typically lasts a pre-determined number of rounds. The player who wins the most rounds wins the game. Given the number of rounds the players will compete, it is your job to determine which player wins after those rounds have been played.

The rules for what item wins are as follows:

- Rock always beats Scissors (Rock crushes Scissors)
- Scissors always beat Paper (Scissors cut Paper)
- Paper always beats Rock (Paper covers Rock)

Input

The first value in the input file will be an integer t ($0 < t < 1000$) representing the number of test cases in the input file. Following this, on a case by case basis, will be an integer n ($0 < n < 100$) specifying the number of rounds of Rock, Paper, Scissors played. Next will be n lines, each with either a capital `R`, `P`, or `S`, followed by a space, followed by a capital `R`, `P`, or `S`, followed by a newline. The first letter is Player 1's choice; the second letter is Player 2's choice.

Output

For each test case, report the name of the player (`` `Player 1' `` or `` `Player 2' ``) that wins the game, followed by a newline. If the game ends up in a tie, print `` `TIE'. ``

Sample Input

```
3
2
R P
S R
3
P P
R S
S R
1
P R
```

Sample Output

```
Player 2
TIE
Player 1
```

Rock, Paper, Scissors is a two player game, where each player simultaneously chooses one of the three items after counting to three. The game typically lasts a pre-determined number of rounds. The player who wins the most rounds wins the game. Given the number of rounds the players will compete, it is your job to determine which player wins after those rounds have been played.

The rules for what item wins are as follows:

- Rock always beats Scissors (Rock crushes Scissors)
- Scissors always beat Paper (Scissors cut Paper)
- Paper always beats Rock (Paper covers Rock)

Input

The first value in the input file will be an integer t ($0 < t < 1000$) representing the number of test cases in the input file. Following this, on a case by case basis, will be an integer n ($0 < n < 100$) specifying the number of rounds of Rock, Paper, Scissors played. Next will be n lines, each with either a capital `R`, `P`, or `S`, followed by a space, followed by a capital `R`, `P`, or `S`, followed by a newline. The first letter is Player 1's choice; the second letter is Player 2's choice.

Output

For each test case, report the name of the player (`` `Player 1' `` or `` `Player 2' ``) that wins the game, followed by a newline. If the game ends up in a tie, print `` `TIE'. ``

Sample Input

```
3
2
R P
S R
3
P P
R S
S R
1
P R
```

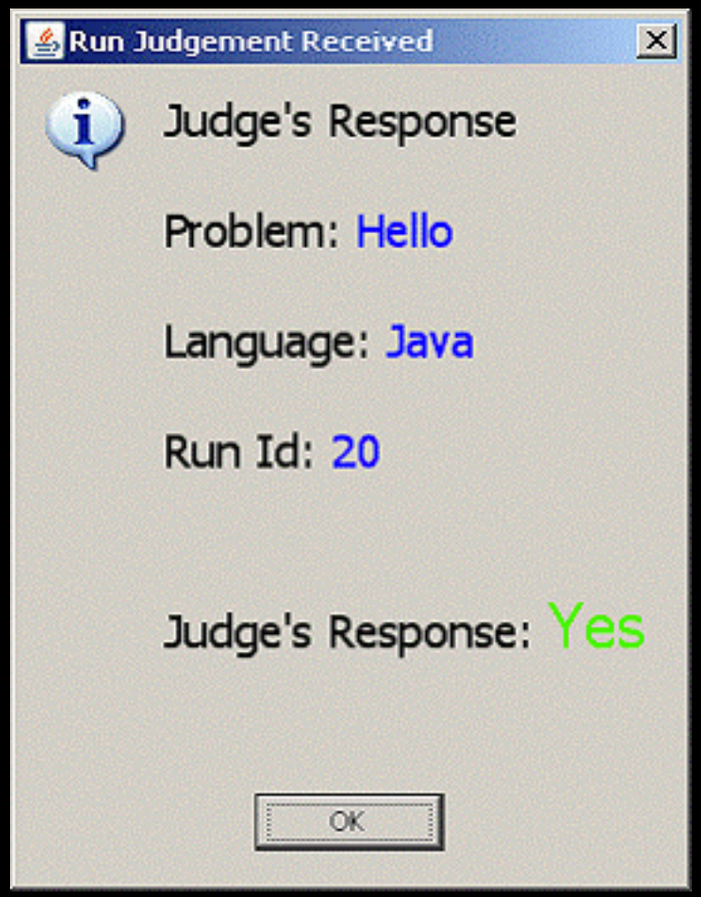
Sample Output

```
Player 2
TIE
Player 1
```

```java
import java.io.*;
import java.util.*;

public class Main {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int t = in.nextInt();
    for(int a = 0; a < t; a++) {
      int n = in.nextInt();
      int p1score = 0, p2score = 0;
      for(int i = 0; i < n; i++) {
        char p1 = in.next().charAt(0);
        char p2 = in.next().charAt(0);
        if(p1 == 'R') {
          if(p2 == 'P') {
            p2score++;
          } else if(p2 == 'S') {
            p1score++;
          }
        } else if(p1 == 'P') {
          if(p2 == 'R') {
            p1score++;
          } else if(p2 == 'S') {
            p2score++;
          }
        } else { //'S'
          if(p2 == 'R') {
            p2score++;
          } else if(p2 == 'P') {
            p1score++;
          }
        }
      }
      System.out.println(p1score==p2score?"TIE":(p1score>p2score?"Player 1":"Player 2"));
    }
  }
}
```

1 try, <5 minutes


Run Judgement Received
Judge's Response
Problem: Hello
Language: Java
Run Id: 20
Judge's Response: Yes
OK

No - Compilation Error

No - Runtime Exception

No - Time Limit Exceeded

No - Wrong Answer

No - See Contest Staff

No - Compilation Error

No - Runtime Exception

No - Time Limit Exceeded

~10,000,000 operations

No - Wrong Answer

No - See Contest Staff

No - Compilation Error

No - Runtime Exception

No - Time Limit Exceeded

~10,000,000 operations

No - Wrong Answer

No - See Contest Staff

I have gotten this message exactly once: I put a while(true){} in my code and was told if I did it again, I'd be disqualified.

ACM-ICPC World Finals

June 22 - June 26

2014

Ekaterinburg

host Ural Federal University

world finals
- Schedule
- Activities
- Local Information
- Teams
- World Finals Rules
- On-Site Registration
- Video/Photo Coverage
- World Finals Results
- Past Problems
- Fact Sheet
- Prog. Environment

regionals
- Regional Finder
- Upcoming Regionals
- Regional Results
- Regional Rules
- Getting Involved
- Starting a Regional
- Free ACM Membership

compete
- Preparation
- Policies & Procedures
- FAQs
- The Problems

community
- IBM
- Upsilon Pi Epsilon
- ACM
- Fact Sheet
- History
- Contacts

ACM-ICPC World Finals

June 22 - June 26

2014

Ekaterinburg

host Ural Federal University

world finals
- Schedule
- Activities
- Local Information
- Teams
- World Finals Rules
- On-Site Registration
- Video/Photo Coverage
- World Finals Results
- Past Problems
- Fact Sheet
- Prog. Environment

regionals
- Regional Finder
- Upcoming Regionals
- Regional Results
- Regional Rules
- Getting Involved
- Starting a Regional
- Free ACM Membership

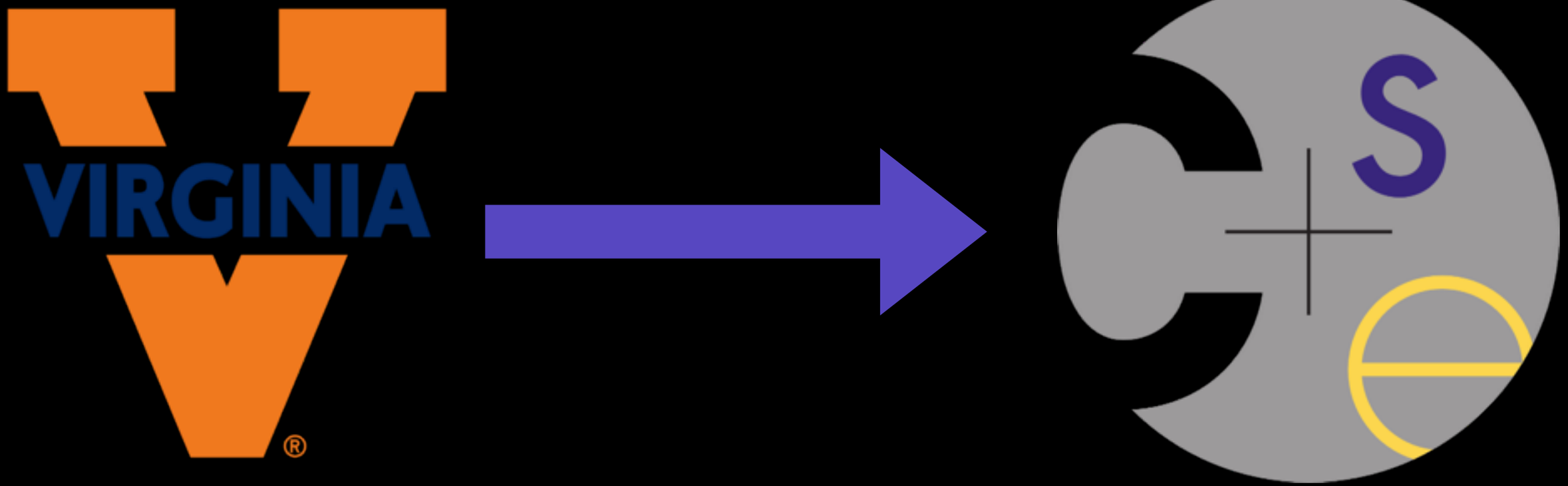
compete
- Preparation
- Policies & Procedures
- FAQs
- The Problems

community
- IBM
- Upsilon Pi Epsilon
- ACM
- Fact Sheet
- History
- Contacts

Google™ Code Jam

TCO13

Who the hell am I?


VIRGINIA
S


November 2010
FIRST PLACE

2011 World Finals
acm International Collegiate Programming Contest

IBM | event sponsor

A - Light Purple

B – Dark Purple

C - Blue

D - RED

#1 → E - White

F - Green

G - Yellow

H - Silver

I – Black

J - Pink

#2 → K- Orange

Team	A	B	C	D	E	F	G	H	I	J	K	S
Zhejiang University	4	1	1		2		4	1		1	1	
niversity of Michigan at Ann Arbor	2	1	2		3		1	3		2	1	
singhua University	1		2		1	2		2		2	1	
. Petersburg State University	1	3	1		1			1		1	1	
zhny Novgorod State University	2	2	1		1		2	2		1	1	
ratov State University	2	2	1		4	10		1		1	1	
drich-Alexander-University Erlangen-Nuremberg	1		1		1			3	1	6	1	
netsk National University	1		1	1	1		2	2		4	1	
ellonian University in Krakow	5	3	1		2	5		2		1	1	
cow State University	3		2		1	3	3	2		2	3	
State University	2		1	2	3		6	4		1	1	
ersity of Waterloo	2		1		2			5	5	4	3	
la V.I. Vernadsky National University	1		1		1		9	1		1		
nal Taiwan University		6	1		3		2	1		1		
rsity of Warsaw	2	4	2		8	2		3		3		
burg State University of IT, Mechanics and Optics	2		2		4		5	2		1		
ng Technological University	1		1		5		7	3		1		
sidad de Buenos Aires - FCEN	1		2		3			1		1		
anced Institute of Science and Technology	1		1		5	4		1		2		
University	3		1		1	2		3		5		

Team	A	B	C	D	E	F	G	H	I	J	K	S
Zhejiang University	4	1	1		2		4	1		1	1	
niversity of Michigan at Ann Arbor			2		3		1	3		2	1	
singhua University	1		2		1	2		2		2	1	
. Petersburg State University	1	3	1		1			1		1	1	
zhny Novgorod State University	2	2	1		1		2	2		1	1	
ratov State University	2	2	1		4	10		1		1	1	
drich-Alexander-University Erlangen-Nuremberg	1		1		1			3	1	6	1	
netsk National University	1		1	1	1		2	2		4	1	
ellonian University in Krakow	5	3	1		2	5		2		1	1	
cow State University	3		2		1	3	3	2		2	3	
State University	2		1	2	3		6	4		1	1	
ersity of Waterloo	2		1		2			5	5	4	3	
la V.I. Vernadsky National University	1		1		1		9	1		1		
nal Taiwan University		6	1		3		2	1		1		
rsity of Warsaw	2	4	2		8	2		3		3		
burg State University of IT, Mechanics and Optics	2		2		4		5	2		1		
ng Technological University	1		1		5		7	3		1		
sidad de Buenos Aires - FCEN	1		2		3			1		1		
anced Institute of Science and Technology	1		1		5	4		1		2		
University	3		1		1	2		3		5		

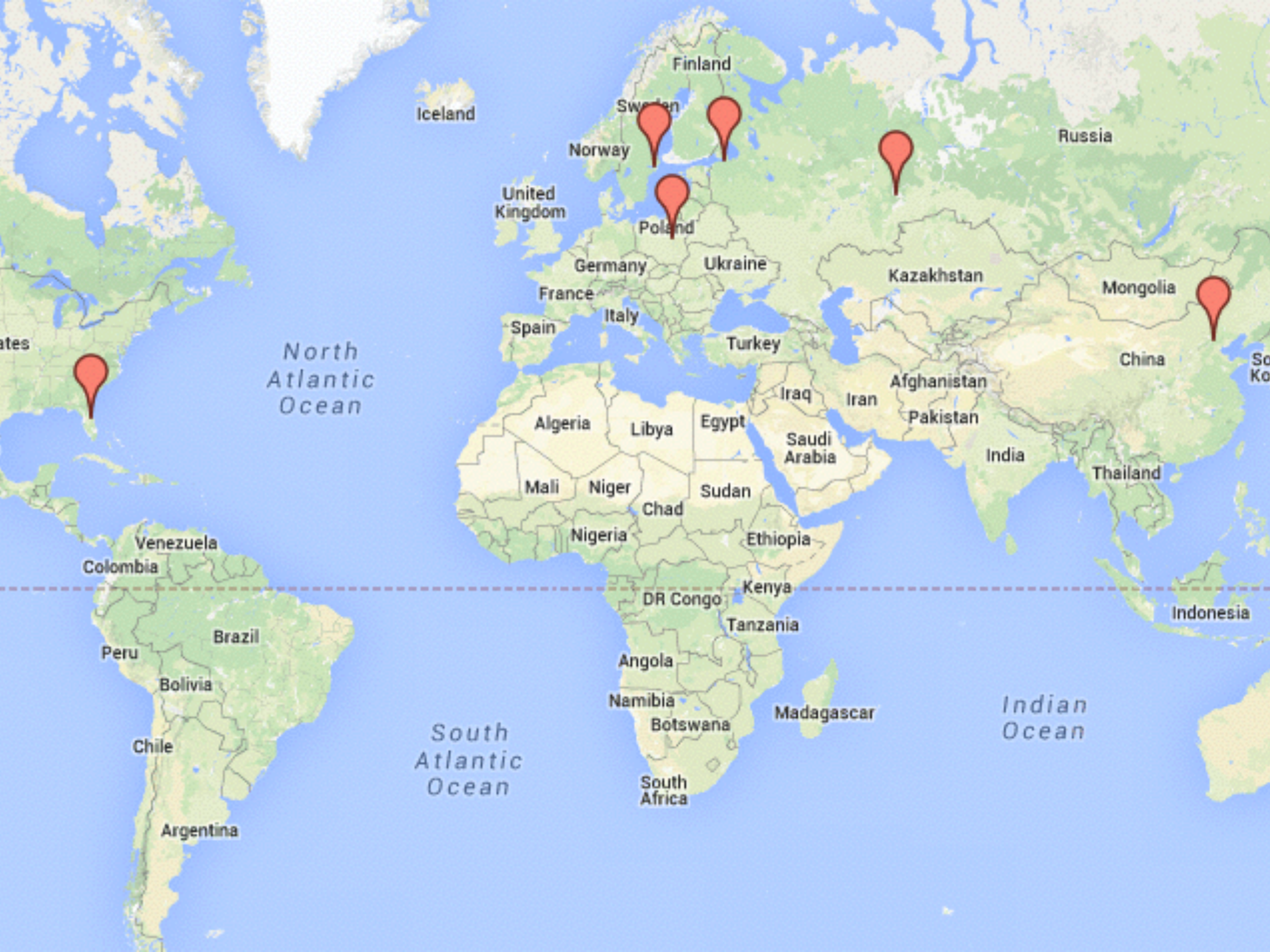

lenovo


COACH

10 reasons why YOU should care about programming competitions

Daniel Epstein, CSE Grad Student

#1: Travel to Exotic Locations


Finland
Iceland
Sweden
Norway
Russia
United Kingdom
Poland
Germany
Ukraine
Kazakhstan
Mongolia
France
Italy
Spain
Turkey
China
North Atlantic Ocean
Afghanistan
Iraq
Iran
Pakistan
Algeria
Libya
Egypt
Saudi Arabia
India
Thailand
Mali
Niger
Sudan
Chad
Nigeria
Ethiopia
Venezuela
Colombia
Kenya
DR Congo
Indonesia
Tanzania
Brazil
Peru
Angola
Bolivia
Namibia
Madagascar
Botswana
Chile
South Atlantic Ocean
Indian Ocean
South Africa
Argentina


Finland
Iceland
Sweden
Norway
Russia
United Kingdom
Poland
Germany
Ukraine
Kazakhstan
Mongolia
France
Italy
Spain
Turkey
China
North Atlantic Ocean
Afghanistan
Iraq
Iran
Pakistan
Algeria
Libya
Egypt
Saudi Arabia
India
Mali
Niger
Sudan
Chad
Thailand
Nigeria
Ethiopia
Venezuela
Colombia
Kenya
DR Congo
Indonesia
Tanzania
Brazil
Peru
Angola
Bolivia
Namibia
Madagascar
Botswana
Chile
South Atlantic Ocean
Indian Ocean
South Africa
Argentina


Finland
Iceland
Sweden
Norway
Russia
United Kingdom
Poland
Germany
Ukraine
Kazakhstan
Mongolia
France
Italy
Spain
Turkey
China
North Atlantic Ocean
Afghanistan
Iraq
Iran
Pakistan
Algeria
Libya
Egypt
Saudi Arabia
India
Thailand
Mali
Niger
Sudan
Chad
Nigeria
Ethiopia
Venezuela
Colombia
Kenya
DR Congo
Indonesia
Tanzania
Brazil
Peru
Angola
Bolivia
Namibia
Madagascar
Botswana
Chile
South Atlantic Ocean
Indian Ocean
South Africa
Argentina

#2: Free Stuff!

U Virginia

2011 World Finals
Orlando

acm International Collegiate Programming Contest

IBM | event sponsor


ACM-ICPC
2009
World Finals
Stockholm,
Sweden
IBM

#3: Job opportunities, looks great on a resume

Google

Microsoft

#4: Learn to play nice with others

#5: Good practice debugging of both your own and other's code

List1

List1

List2

List1 1 2 3

List2 6 5 8 9

Find the bug!

```java
import java.io.*;
import java.util.*;

public class iterate {

  public static void main(String[] args) {
    ArrayList<Integer> list1 = new ArrayList<Integer>(Arrays.asList(new Integer[]{1, 2, 3}));
    ArrayList<Integer> list2 = new ArrayList<Integer>(Arrays.asList(new Integer[]{6, 1, 3, 5, 2, 8, 9}));
    // Remove all occurrences of elements in list1 from list2
    for(int i=0;i<list2.size();i++) {
      for(int j=0;j<list1.size(); j++) {
        if(list2.get(i) == list1.get(j)) {
          list2.remove(i);
          break;
        }
      }
    }
    System.out.println(list2);
  }
}
```

Find the bug!

```java
import java.io.*;
import java.util.*;

public class iterate {

  public static void main(String[] args) {
    ArrayList<Integer> list1 = new ArrayList<Integer>(Arrays.asList(new Integer[]{1, 2, 3}));
    ArrayList<Integer> list2 = new ArrayList<Integer>(Arrays.asList(new Integer[]{6, 1, 3, 5, 2, 8, 9}));
    // Remove all occurrences of elements in list1 from list2
    for(int i=0;i<list2.size();i++) {
      for(int j=0;j<list1.size(); j++) {
        if(list2.get(i) == list1.get(j)) {
          list2.remove(i);
          break;
        }
      }
    }
    System.out.println(list2);
  }
}
```

i--;

#6: You'll ace your technical interviews

Bessie has gone on a trip, and she's riding a roller coaster! Bessie really likes riding the roller coaster, but unfortunately she often gets dizzy.

The roller coaster has a number of distinct sections that Bessie rides in order. At the beginning of the ride, Bessie's dizziness and fun levels are both at 0. For each section of the roller coaster, Bessie can either keep her eyes open or keep them closed (and must keep them that way for the whole section). If she keeps her eyes open for a section, her total fun increases by a Fun factor for that section, and her dizziness increases by a Dizziness factor for that section. However, if she keeps her eyes closed for the section, her total fun will not change, but her dizziness will decrease by a value that's constant for the entire roller coaster. (Note that her dizziness can never go below 0.)

If, at any point, Bessie's dizziness is above a certain limit, Bessie will get sick. Write a program to find the maximum amount of fun Bessie can have without getting sick.

Input

There will be several test cases in the input. Each test case will begin with a line with three integers:

$N\ K\ L$

Where N ($1 \leq N \leq 1,000$) is the number of sections in this particular roller coaster, K ($1 \leq K \leq 500$) is the amount that Bessie's dizziness level will go down if she keeps her eyes closed on any section of the ride, and L ($1 \leq L \leq 300,000$) is the limit of dizziness that Bessie can tolerate - if her dizziness ever becomes larger than L, Bessie will get sick, and that's not fun!

Each of the next N lines will describe a section of the roller coaster, and will have two integers:

$F\ D$

Where F ($1 \leq F \leq 20$) is the increase to Bessie's total fun that she'll get if she keeps her eyes open on that section, and D ($1 \leq D \leq 500$) is the increase to her dizziness level if she keeps her eyes open on that section.

The sections will be listed in order. The input will end with a line with three `0`s.

Output

For each test case, output a single integer, representing the maximum amount of fun Bessie can have on that roller coaster without exceeding her dizziness limit. Print each integer on its own line with no spaces. Do not print any blank lines between answers.

Bessie has gone on a trip, and she's riding a roller coaster! Bessie really likes riding the roller coaster, but unfortunately she often gets dizzy.

The roller coaster has a number of distinct sections that Bessie rides in order. At the beginning of the ride, Bessie's dizziness and fun levels are both at 0. For each section of the roller coaster, Bessie can either keep her eyes open or keep them closed (and must keep them that way for the whole section). If she keeps her eyes open for a section, her total fun increases by a Fun factor for that section, and her dizziness increases by a Dizziness factor for that section. However, if she keeps her eyes closed for the section, her total fun will not change, but her dizziness will decrease by a value that's constant for the entire roller coaster. (Note that her dizziness can never go below 0.)

If, at any point, Bessie's dizziness is above a certain limit, Bessie will get sick. Write a program to find the maximum amount of fun Bessie can have without getting sick.

Input

There will be several test cases in the input. Each test case will begin with a line with three integers:

$N\ K\ L$

Where $N\ (1 \leq N \leq 1,000)$ is the number of sections in this particular roller coaster, $K\ (1 \leq K \leq 500)$ is the amount that Bessie's dizziness level will go down if she keeps her eyes closed on any section of the ride, and $L\ (1 \leq L \leq 300,000)$ is the limit of dizziness that Bessie can tolerate - if her dizziness ever becomes larger than L, Bessie will get sick, and that's not fun!

Each of the next N lines will describe a section of the roller coaster, and will have two integers:

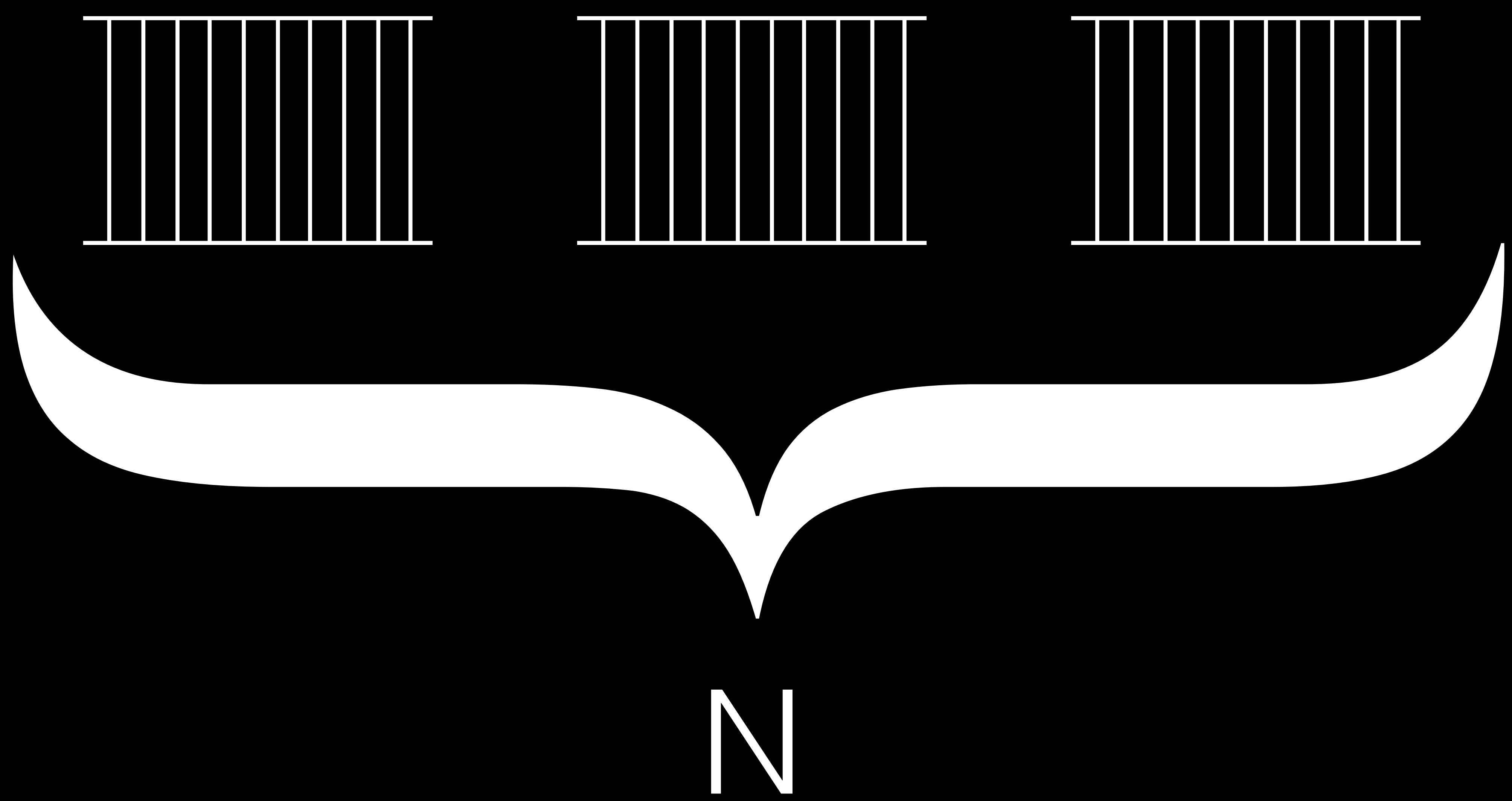
$F\ D$

Where $F\ (1 \leq F \leq 20)$ is the increase to Bessie's total fun that she'll get if she keeps her eyes open on that section, and $D\ (1 \leq D \leq 500)$ is the increase to her dizziness level if she keeps her eyes open on that section.

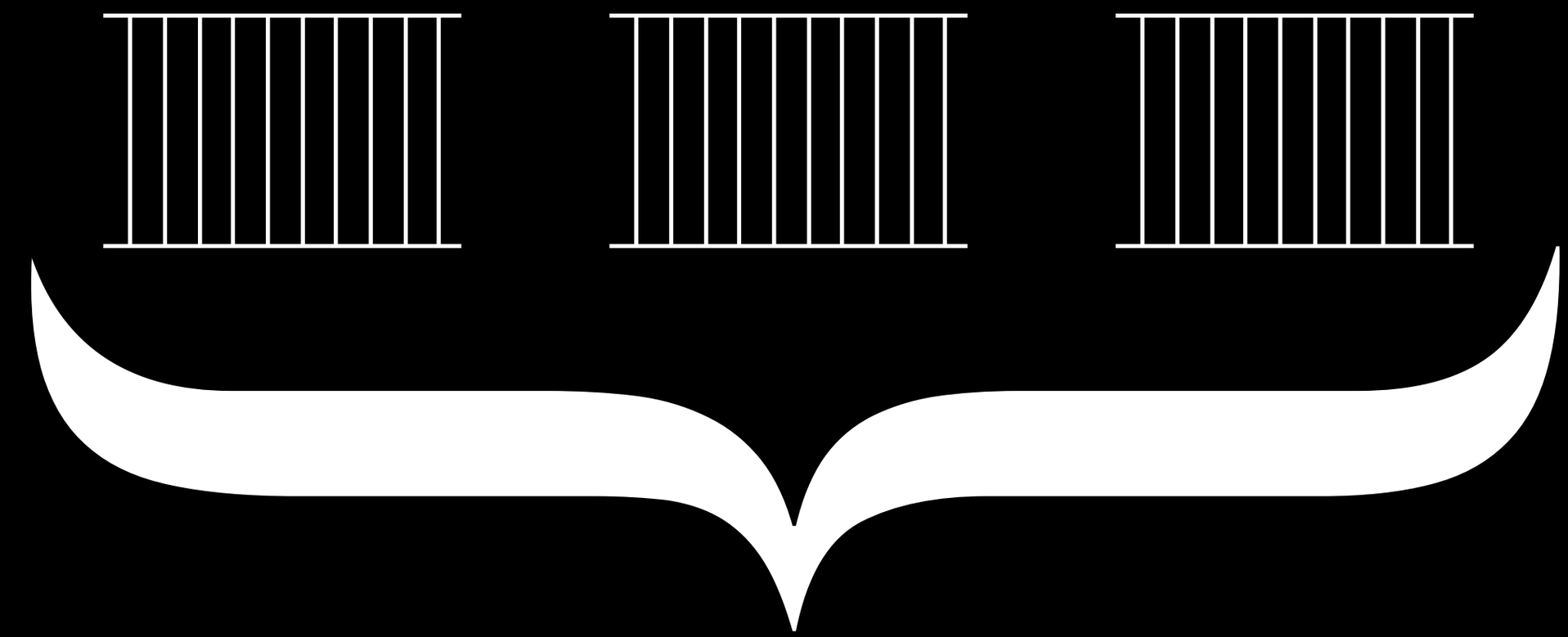
The sections will be listed in order. The input will end with a line with three `0`s.

Output

For each test case, output a single integer, representing the maximum amount of fun Bessie can have on that roller coaster without exceeding her dizziness limit. Print each integer on its own line with no spaces. Do not print any blank lines between answers.


N

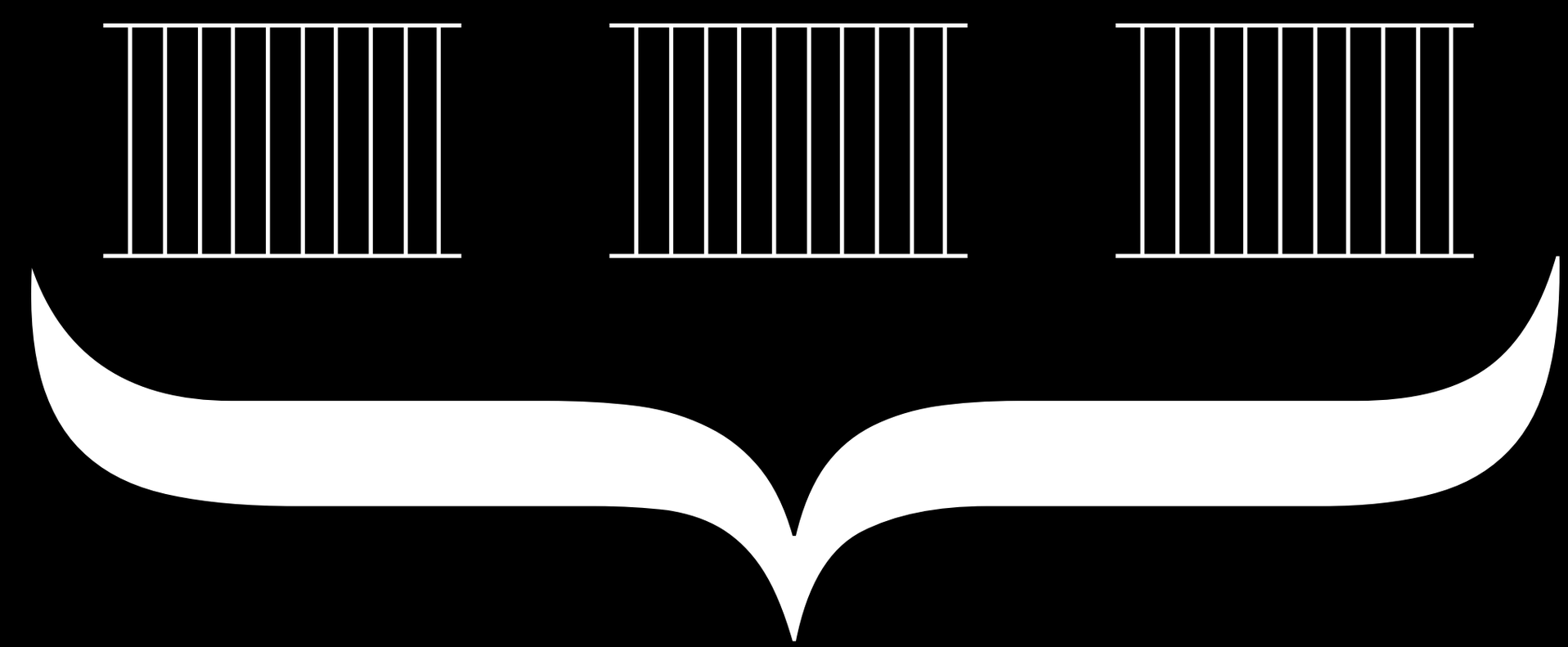
Fun=X, Dizzy=Y



N

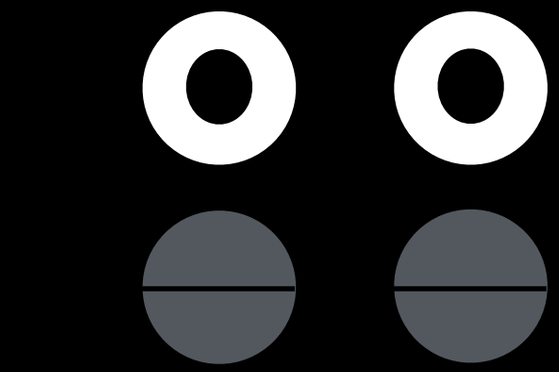
Fun=X, Dizzy=Y

Fun+=F[i], Dizzy+=D[i]



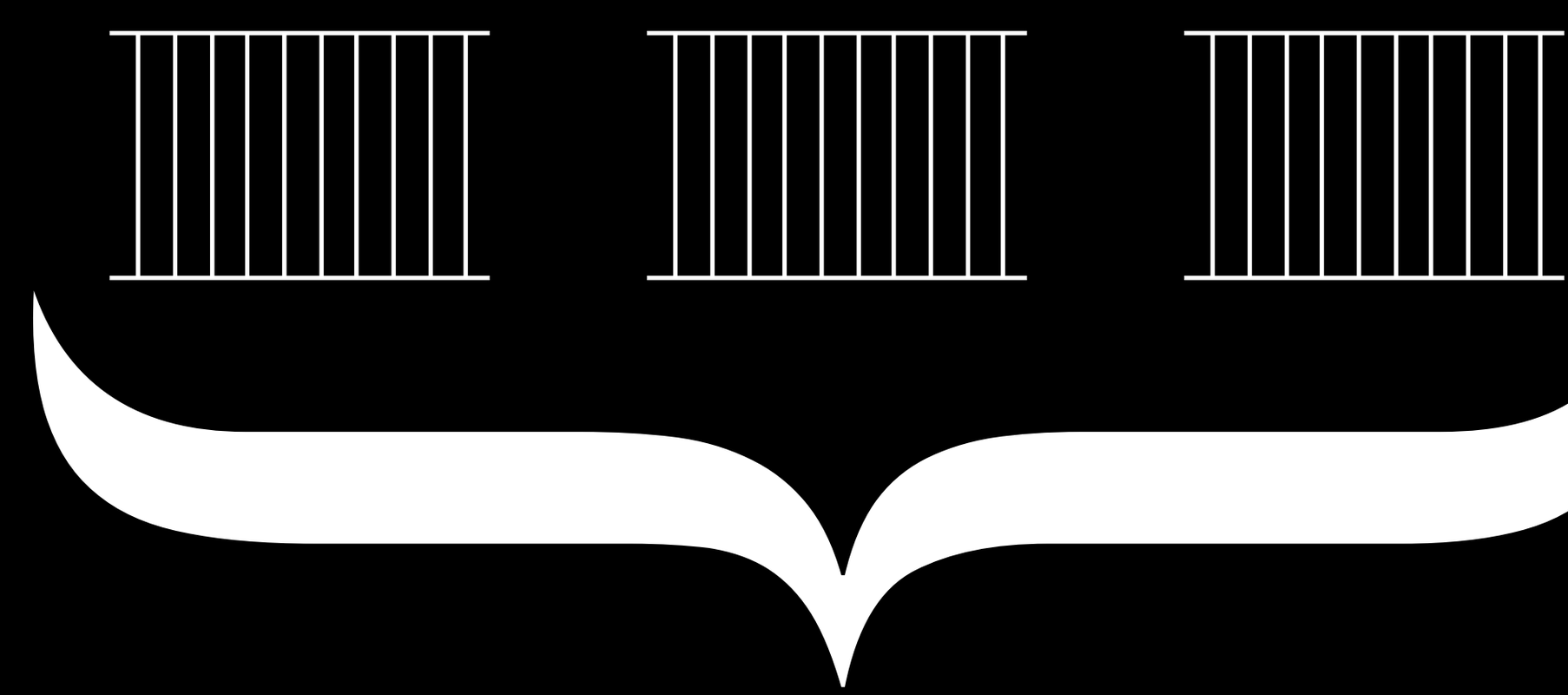
N

Fun=X, Dizzy=Y



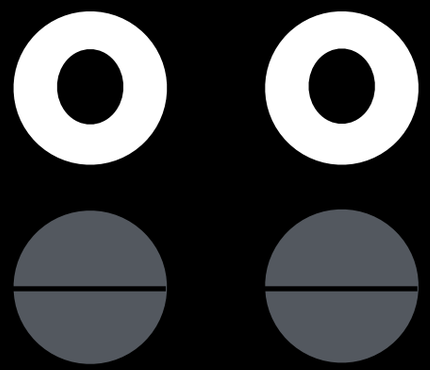
Fun+=F[i], Dizzy+=D[i]

Dizzy-=K



N

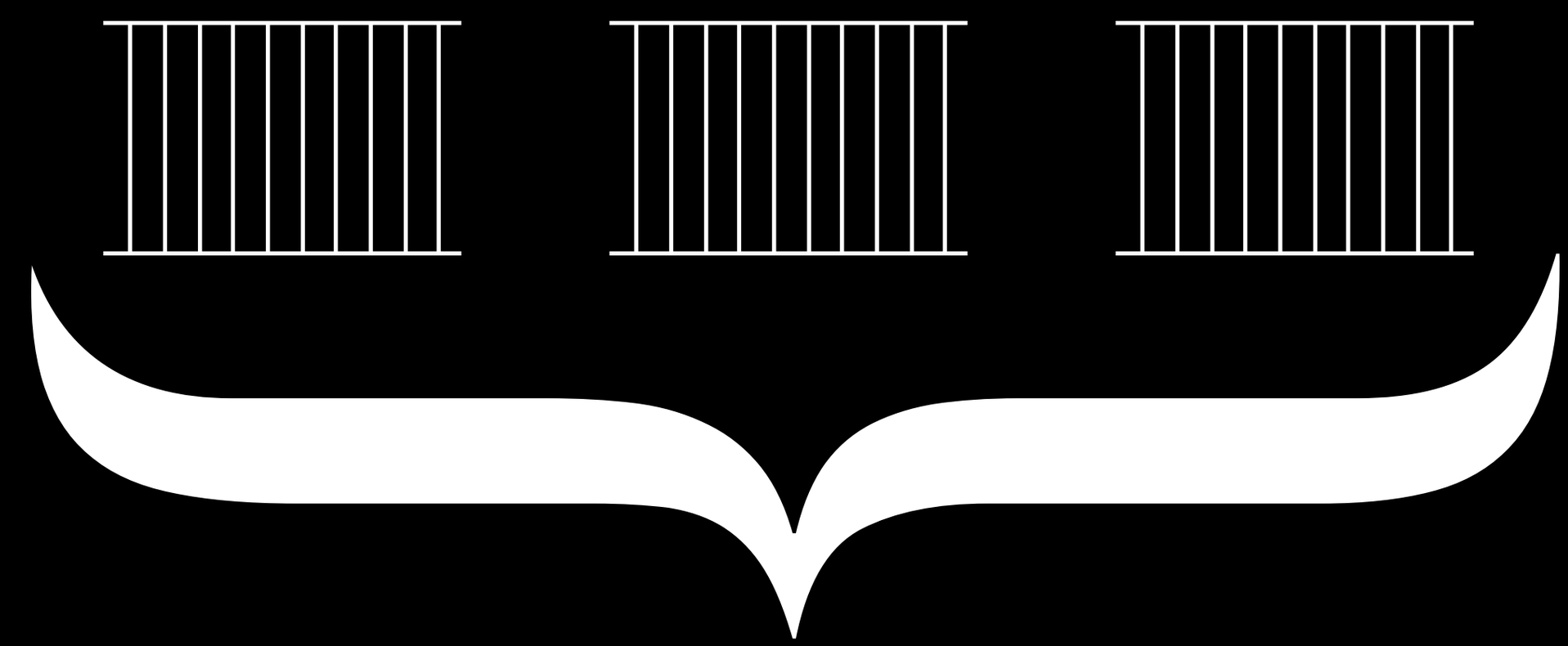
Fun=X, Dizzy=Y



Fun+=F[i], Dizzy+=D[i]

Dizzy-=K

Dizzy <= L
at all times



N

```java
import java.io.*;
import java.util.*;

public class Main {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    while(true) {
        int N = in.nextInt(), K = in.nextInt(), L = in.nextInt();
        if(N==0 && K==0 && L==0) {
          break;
        }
        int[] fun = new int[N], dizzy = new int[N];
        for(int i=0; i<N; i++) {
          fun[i] = in.nextInt();
          dizzy[i] = in.nextInt();
        }
        // Now what?
    }
  }
}
```

```java
import java.io.*;
import java.util.*;

public class Main {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    while(true) {
        int N = in.nextInt(), K = in.nextInt(), L = in.nextInt();
        if(N==0 && K==0 && L==0) {
          break;
        }
        int[] fun = new int[N], dizzy = new int[N];
        for(int i=0; i<N; i++) {
          fun[i] = in.nextInt();
          dizzy[i] = in.nextInt();
        }
        // Now what?
    }
  }
}
```

dp[n][l] = maximum possible fun as of section n with dizziness less than l

dp[n][l] = **max**(dp[n-1][l - dizzy[n]] + fun[n], dp[n-1][l+k])

O(N*L)

```java
import java.io.*;
import java.util.*;

public class Main {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    while(true) {
        int N = in.nextInt(), K = in.nextInt(), L = in.nextInt();
        if(N==0 && K==0 && L==0) {
          break;
        }
        int[] fun = new int[N], dizzy = new int[N];
        for(int i=0; i<N; i++) {
          fun[i] = in.nextInt();
          dizzy[i] = in.nextInt();
        }
        // Now what?
    }
  }
}
```

`dp[n][l] = maximum possible fun as of section n with dizziness less than l`

`dp[n][l] = max(dp[n-1][l - dizzy[n]] + fun[n], dp[n-1][l+k])`

O(N*L)

> Where N ($1 \leq N \leq 1,000$) is the number of sections in this particular roller coaster, K ($1 \leq K \leq 500$) is the amount that Bessie's dizziness level will go down if she keeps her eyes closed on any section of the ride, and L ($1 \leq L \leq 300,000$) is the limit of dizziness that Bessie can tolerate - if her dizziness ever becomes larger than L, Bessie will get sick, and that's not fun!

1,000 * 300,000 = 300,000,000 -> WAY TOO HIGH!!

Input

There will be several test cases in the input. Each test case will begin with a line with three integers:

N K L

Where *N* ($1 \leq N \leq 1,000$) is the number of sections in this particular roller coaster, *K* ($1 \leq K \leq 500$) is the amount that Bessie's dizziness level will go down if she keeps her eyes closed on any section of the ride, and *L* ($1 \leq L \leq 300,000$) is the limit of dizziness that Bessie can tolerate - if her dizziness ever becomes larger than *L*, Bessie will get sick, and that's not fun!

Each of the next *N* lines will describe a section of the roller coaster, and will have two integers:

F D

Where *F* ($1 \leq F \leq 20$) is the increase to Bessie's total fun that she'll get if she keeps her eyes open on that section, and *D* ($1 \leq D \leq 500$) is the increase to her dizziness level if she keeps her eyes open on that section.

Input

There will be several test cases in the input. Each test case will begin with a line with three integers:

$N\ K\ L$

Where N ($1 \leq N \leq 1,000$) is the number of sections in this particular roller coaster, K ($1 \leq K \leq 500$) is the amount that Bessie's dizziness level will go down if she keeps her eyes closed on any section of the ride, and L ($1 \leq L \leq 300,000$) is the limit of dizziness that Bessie can tolerate - if her dizziness ever becomes larger than L, Bessie will get sick, and that's not fun!

Each of the next N lines will describe a section of the roller coaster, and will have two integers:

$F\ D$

Where F ($1 \leq F \leq 20$) is the increase to Bessie's total fun that she'll get if she keeps her eyes open on that section, and D ($1 \leq D \leq 500$) is the increase to her dizziness level if she keeps her eyes open on that section.

Input

There will be several test cases in the input. Each test case will begin with a line with three integers:

$N\ K\ L$

Where N ($1 \leq N \leq 1,000$) is the number of sections in this particular roller coaster, K ($1 \leq K \leq 500$) is the amount that Bessie's dizziness level will go down if she keeps her eyes closed on any section of the ride, and L ($1 \leq L \leq 300,000$) is the limit of dizziness that Bessie can tolerate - if her dizziness ever becomes larger than L, Bessie will get sick, and that's not fun!

Each of the next N lines will describe a section of the roller coaster, and will have two integers:

$F\ D$

Where F ($1 \leq F \leq 20$) is the increase to Bessie's total fun that she'll get if she keeps her eyes open on that section, and D ($1 \leq D \leq 500$) is the increase to her dizziness level if she keeps her eyes open on that section.

```
dp[n][f] = minimum dizziness with fun level f
dp[n][f] = min(dp[n-1][f - fun[n]] + dizzy[n], dp[n-1][f] - k)
```

O(N*F)

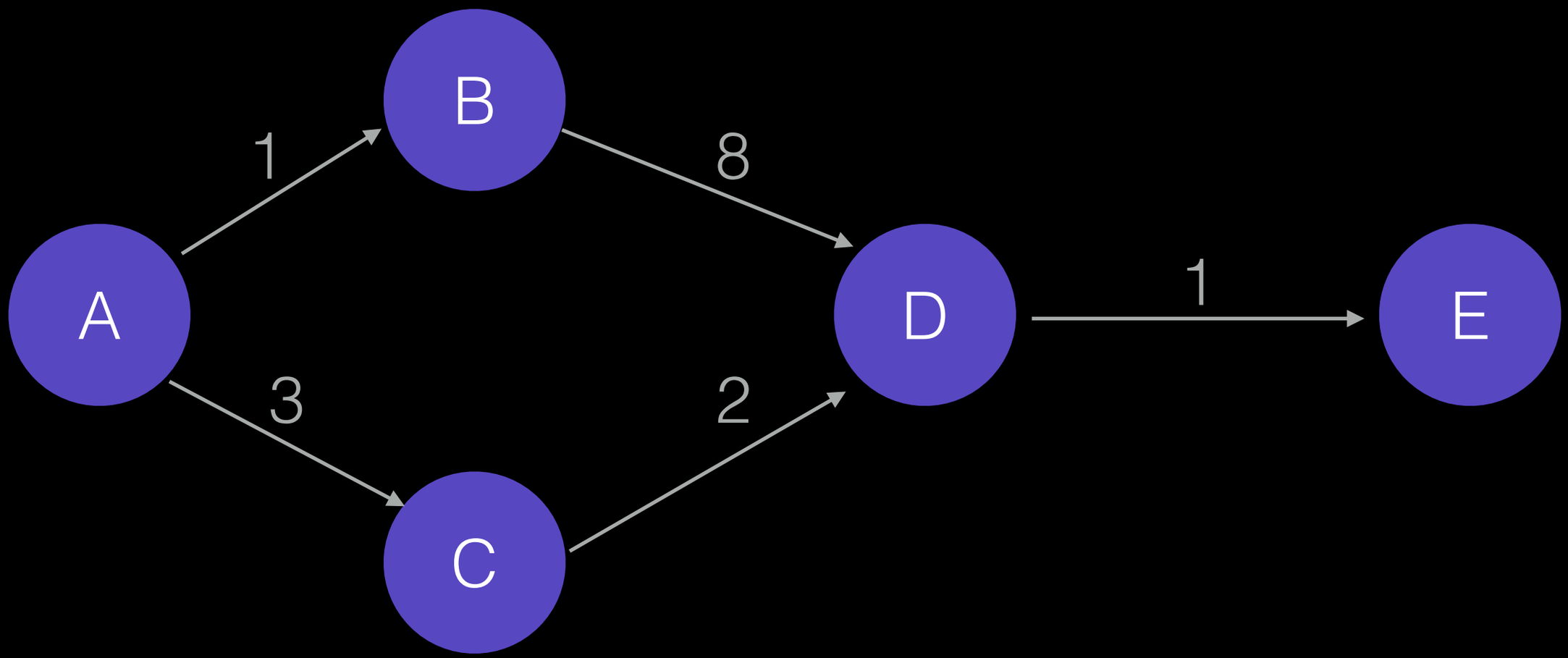
1,000 * 20 = 20,000 -> Totally reasonable!

```java
import java.io.*;
import java.util.*;

public class Main {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    while(true) {
        int N = in.nextInt(), K = in.nextInt(), L = in.nextInt();
        if(N==0 && K==0 && L==0) {
          break;
        }
        int[] fun = new int[N], dizzy = new int[N];
        int maxF = 0;
        for(int i=0; i<N; i++) {
          fun[i] = in.nextInt();
          dizzy[i] = in.nextInt();
          maxF += fun[i];
        }
        // Now what?
        int[][] dp = new int[N][maxF+1];
        for(int i=0;i<N;i++) {
          for(int j=0;j<=maxF;j++) {
            dp[i][j] = Integer.MAX_VALUE;
          }
        }
        dp[0][0] = 0; //Populate with the first values
        if(dizzy[0] <= L) {
          dp[0][fun[0]] = dizzy[0];
        }
        for(int n=1; n<N; n++) {
          for(int f=0; f<=maxF; f++) {
            if(dp[n-1][f] != Integer.MAX_VALUE) {
              dp[n][f] = Math.max(dp[n-1][f] - K, 0); // Dizziness can't go below 0
            }
            if(f - fun[n] >= 0 && dp[n-1][f-fun[n]] != Integer.MAX_VALUE && dp[n-1][f-fun[n]] + dizzy[n] <= L) {
              dp[n][f] = Math.min(dp[n-1][f - fun[n]] + dizzy[n], dp[n][f]);
            }
          }
        }
        for(int i=maxF; i>=0; i--) {
          if(dp[N-1][i] != Integer.MAX_VALUE) {
            System.out.println(i);
            break;
          }
        }
    }
  }
}
```

2 tries, ~40 minutes (off-by-one error)

#7: Deepen understanding of language semantics


B
1
8
A
D
1
E
3
2
C

Find the bug!

```java
import java.io.*;
import java.util.*;

public class dijkstra {

  public static void main(String[] args) {
    Node a = new Node();
    Node b = new Node();
    Node c = new Node();
    Node d = new Node();
    Node e = new Node();
    a.edges.put(b, 1);
    a.edges.put(c, 3);
    b.edges.put(d, 8);
    c.edges.put(d, 2);
    d.edges.put(e, 1);
    ArrayList<Node> allNodes = new ArrayList<Node>(Arrays.asList(new Node[]{e, d, c, b, a}));
    dijkstra(a, allNodes);
    System.out.printf("Distance to Node e is: %d\n", e.distance);
  }

  public static void dijkstra(Node root, ArrayList<Node> allNodes) {
    PriorityQueue<Node> q = new PriorityQueue<Node>();
    root.distance = 0;
    //Add all nodes to the priority queue
    for(Node n : allNodes) {
      q.add(n);
    }

    while(q.size() > 0) {
      Node u = q.poll();
      for(Node n : u.edges.keySet()) {
        n.distance = Math.min(n.distance, u.distance + u.edges.get(n));
      }
    }
  }
}

class Node implements Comparable<Node> {
  public HashMap<Node, Integer> edges = new HashMap<Node, Integer>();
  public int distance = Integer.MAX_VALUE;

  public int compareTo(Node o) {
    return (distance < o.distance) ? -1 : ((distance == o.distance) ? 0 : 1);
  }
}
```

Find the bug!

```java
import java.io.*;
import java.util.*;

public class dijkstra {

  public static void main(String[] args) {
    Node a = new Node();
    Node b = new Node();
    Node c = new Node();
    Node d = new Node();
    Node e = new Node();
    a.edges.put(b, 1);
    a.edges.put(c, 3);
    b.edges.put(d, 8);
    c.edges.put(d, 2);
    d.edges.put(e, 1);
    ArrayList<Node> allNodes = new ArrayList<Node>(Arrays.asList(new Node[]{e, d, c, b, a}));
    dijkstra(a, allNodes);
    System.out.printf("Distance to Node e is: %d\n", e.distance);
  }

  public static void dijkstra(Node root, ArrayList<Node> allNodes) {
    PriorityQueue<Node> q = new PriorityQueue<Node>();
    root.distance = 0;
    //Add all nodes to the priority queue
    for(Node n : allNodes) {
      q.add(n);
    }

    while(q.size() > 0) {
      Node u = q.poll();
      for(Node n : u.edges.keySet()) {
        n.distance = Math.min(n.distance, u.distance + u.edges.get(n));
      }
    }
  }
}

class Node implements Comparable<Node> {
  public HashMap<Node, Integer> edges = new HashMap<Node, Integer>();
  public int distance = Integer.MAX_VALUE;

  public int compareTo(Node o) {
    return (distance < o.distance) ? -1 : ((distance == o.distance) ? 0 : 1);
  }
}
```

Doesn’t re-order the priority queue

#8: Learn clever optimizations

Competition has been fierce among the cellphone providers in the town of Eastern WestField. Several companies have been running ads in which they each claim to provide more coverage to the town than do any of their competitors.

AetherTech Telecommunications suspects that they, in fact, really do cover a larger portion of the town than do their competitors. They would like to compute their actual coverage so thay they can advertise the true amount.

The metropolitan area of EasternWestField will be modeled as a circle around the town center. The area covered by each cell tower can also be modeled as a circle, centered on the location of the tower, indicating the area within which a the tower provides sufficient signal strength for a phone to connect (e.g., one bar on the average phone s signal strength meter).

All cell towers to be considered will have their centers within the metropolitan area, though their area of coverage may extend beyond the border of the metropolitan area. The areas covered by different towers can overlap, though for economic reasons no two towers have been constructed so close to one another that the center of one would lie within the coverage area of the other.

Compute the fraction of the metropolitan area within which a cell phone would be covered by at least one tower.

Input

Input consists of multiple data sets. Each data set begins with a line containing a single integer N, $1 \le N \le 25$. This line is followed by N lines, each containing three floating point numbers x, y, r. These give the (x,y) coordinates of the circle s center and the radius of that circle, respectively. The first circle in the list denotes the metropolitan area of Eastern Westfield. Each of the remaining N-1 circles describes a cell tower.

The final data set is followed by a line containing a zero.

Output

For each data set, compute the fraction of the metropolitan area covered by the cell towers (as a number in the range 0.00...1.00 and print a line containing that fraction as a real number presented to 2 decimal points precision.

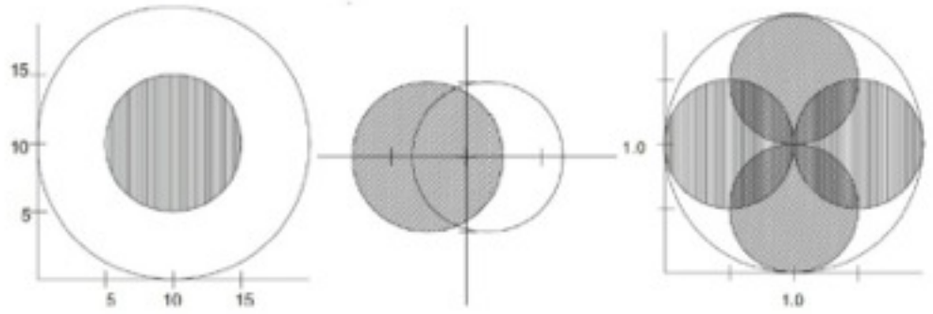
Sample Input

```
2
10 10 10
10 10 5
2
0.404 0.0 1.0
-0.404 0.0 1.0
5
0 0 2
1 0 1
0 1 1
-1 0 1
0 -1 1
0
```

Sample Output

```
0.25
0.50
0.82
```

Note: The images corresponding to the sample input are:



Competition has been fierce among the cellphone providers in the town of Eastern WestField. Several companies have been running ads in which they each claim to provide more coverage to the town than do any of their competitors.

AetherTech Telecommunications suspects that they, in fact, really do cover a larger portion of the town than do their competitors. They would like to compute their actual coverage so thay they can advertise the true amount.

The metropolitan area of EasternWestField will be modeled as a circle around the town center. The area covered by each cell tower can also be modeled as a circle, centered on the location of the tower, indicating the area within which a the tower provides sufficient signal strength for a phone to connect (e.g., one bar on the average phone s signal strength meter).

All cell towers to be considered will have their centers within the metropolitan area, though their area of coverage may extend beyond the border of the metropolitan area. The areas covered by different towers can overlap, though for economic reasons no two towers have been constructed so close to one another that the center of one would lie within the coverage area of the other.

Compute the fraction of the metropolitan area within which a cell phone would be covered by at least one tower.

Input

Input consists of multiple data sets. Each data set begins with a line containing a single integer N, $1 \le N \le 25$. This line is followed by N lines, each containing three floating point numbers x, y, r. These give the (x,y) coordinates of the circle s center and the radius of that circle, respectively. The first circle in the list denotes the metropolitan area of Eastern Westfield. Each of the remaining N-1 circles describes a cell tower.

The final data set is followed by a line containing a zero.

Output

For each data set, compute the fraction of the metropolitan area covered by the cell towers (as a number in the range 0.00...1.00 and print a line containing that fraction as a real number presented to 2 decimal points precision.

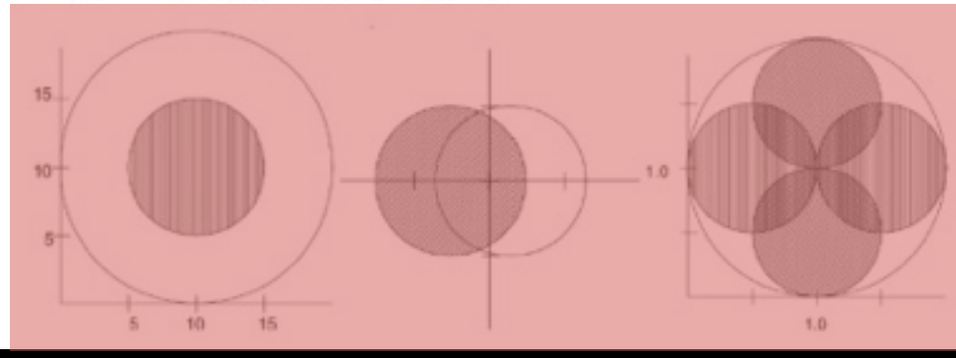
Sample Input

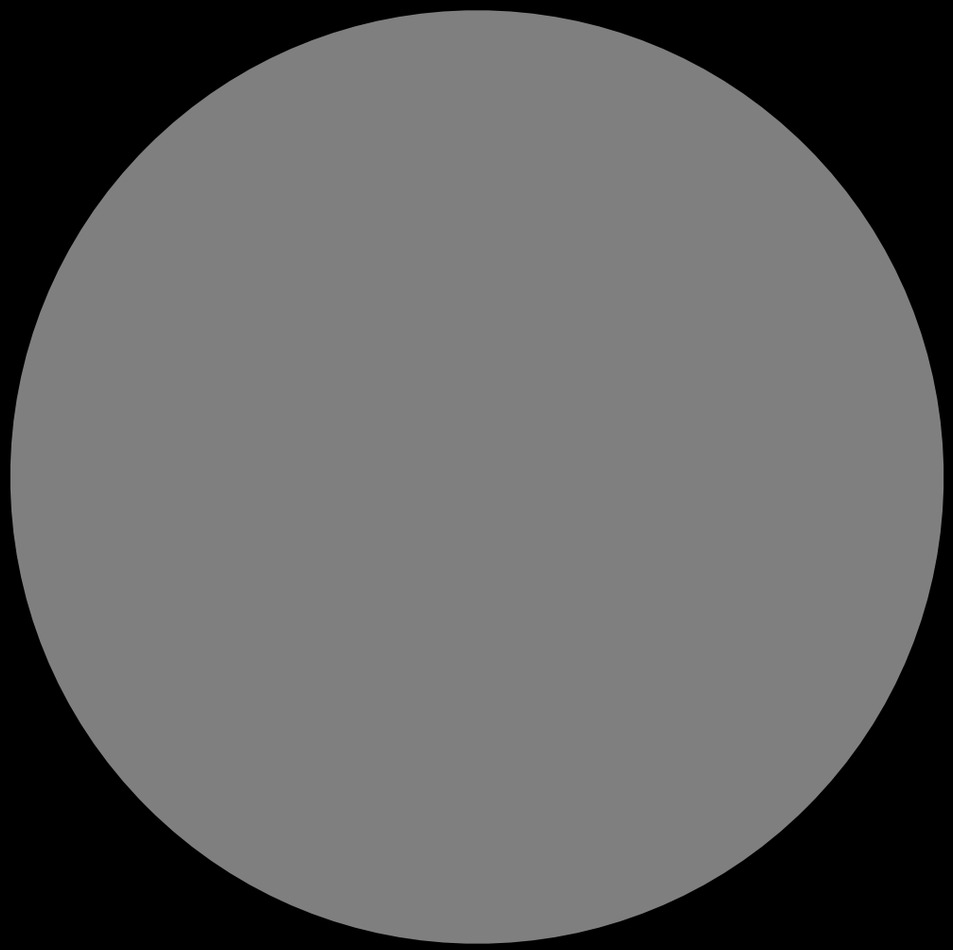
```
2
10 10 10
10 10 5
2
0.404 0.0 1.0
-0.404 0.0 1.0
5
0 0 2
1 0 1
0 1 1
-1 0 1
0 -1 1
0
```

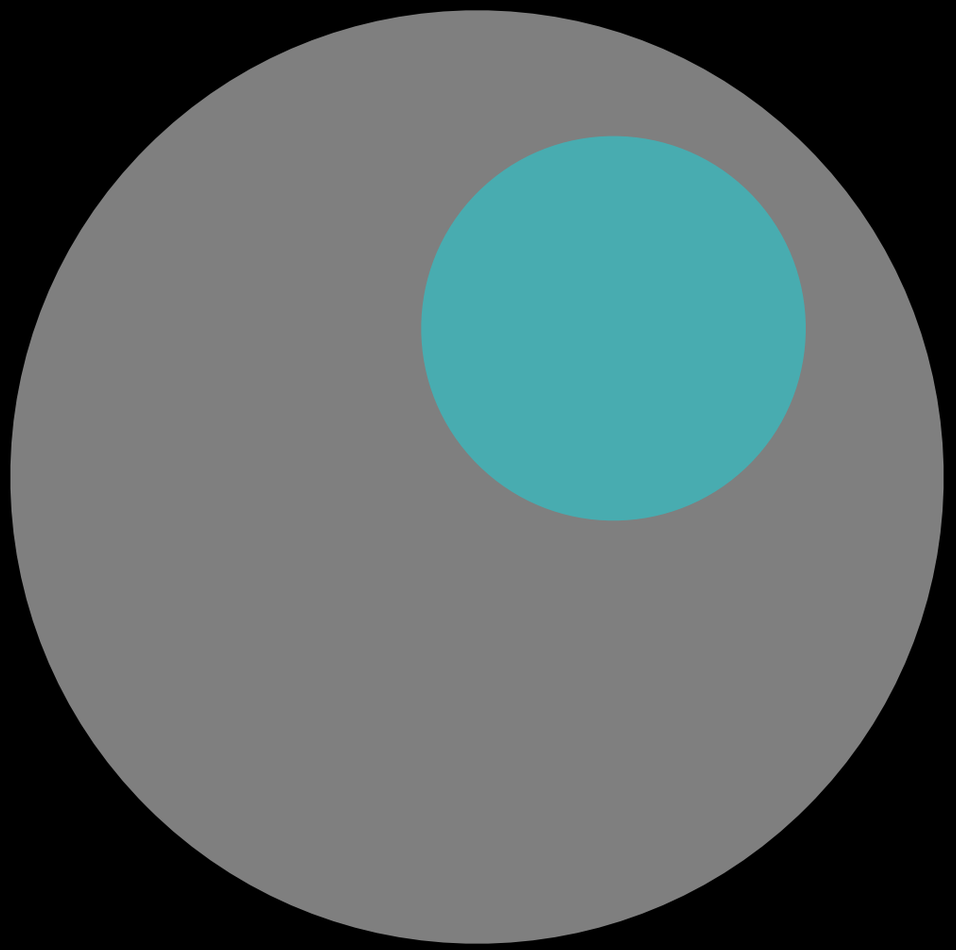
Sample Output

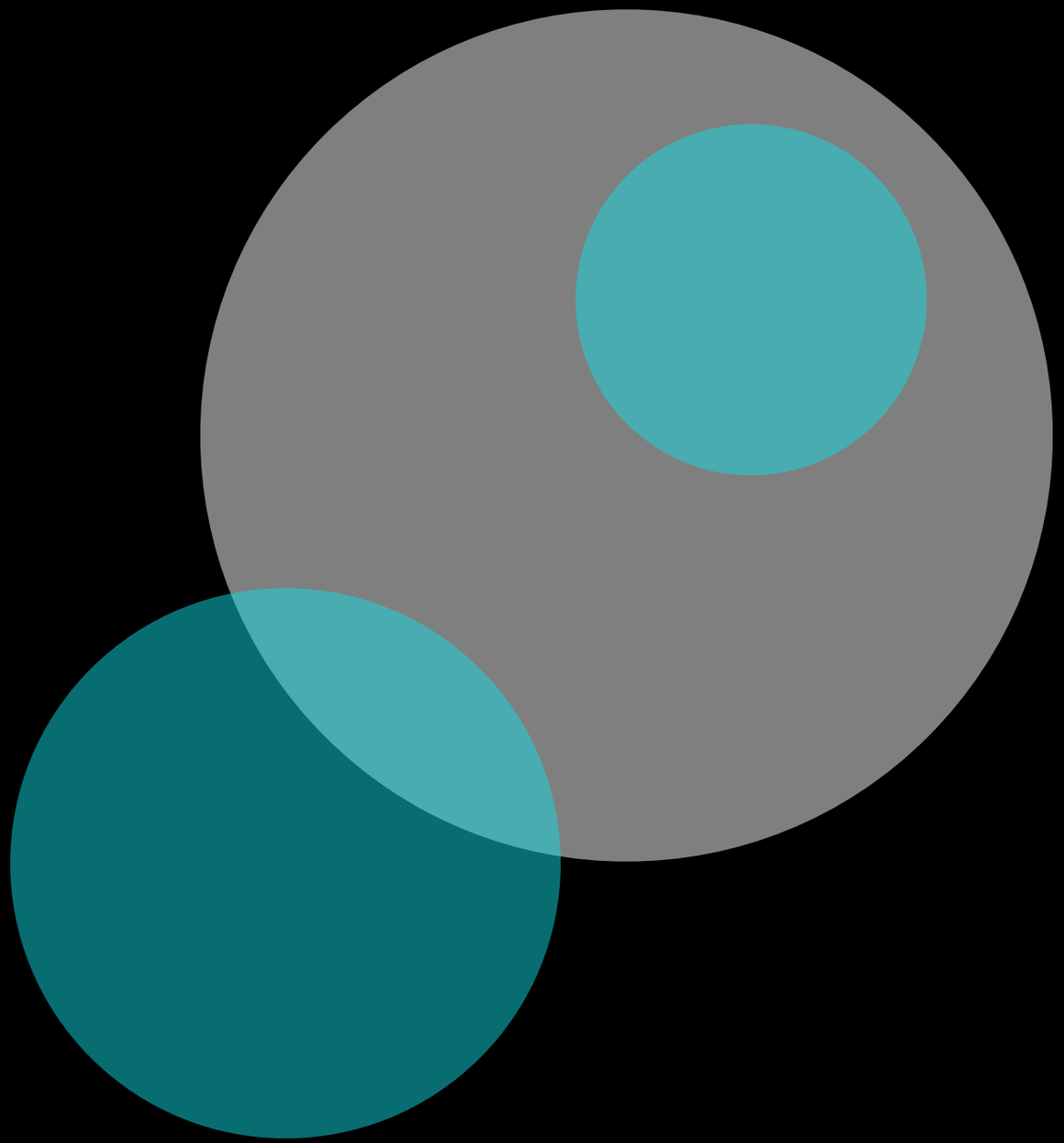
```
0.25
0.50
0.82
```

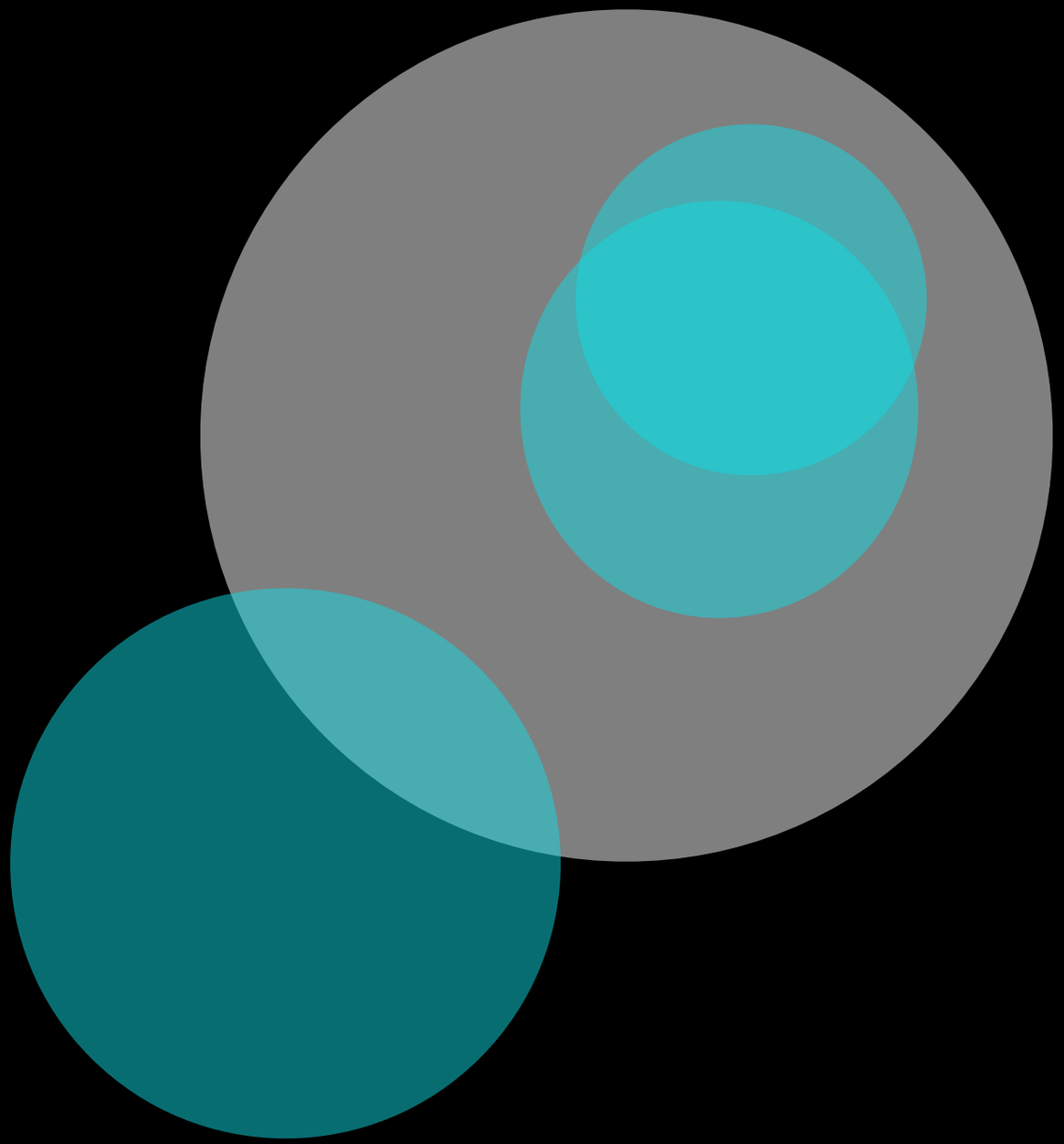
Note: The images corresponding to the sample input are:

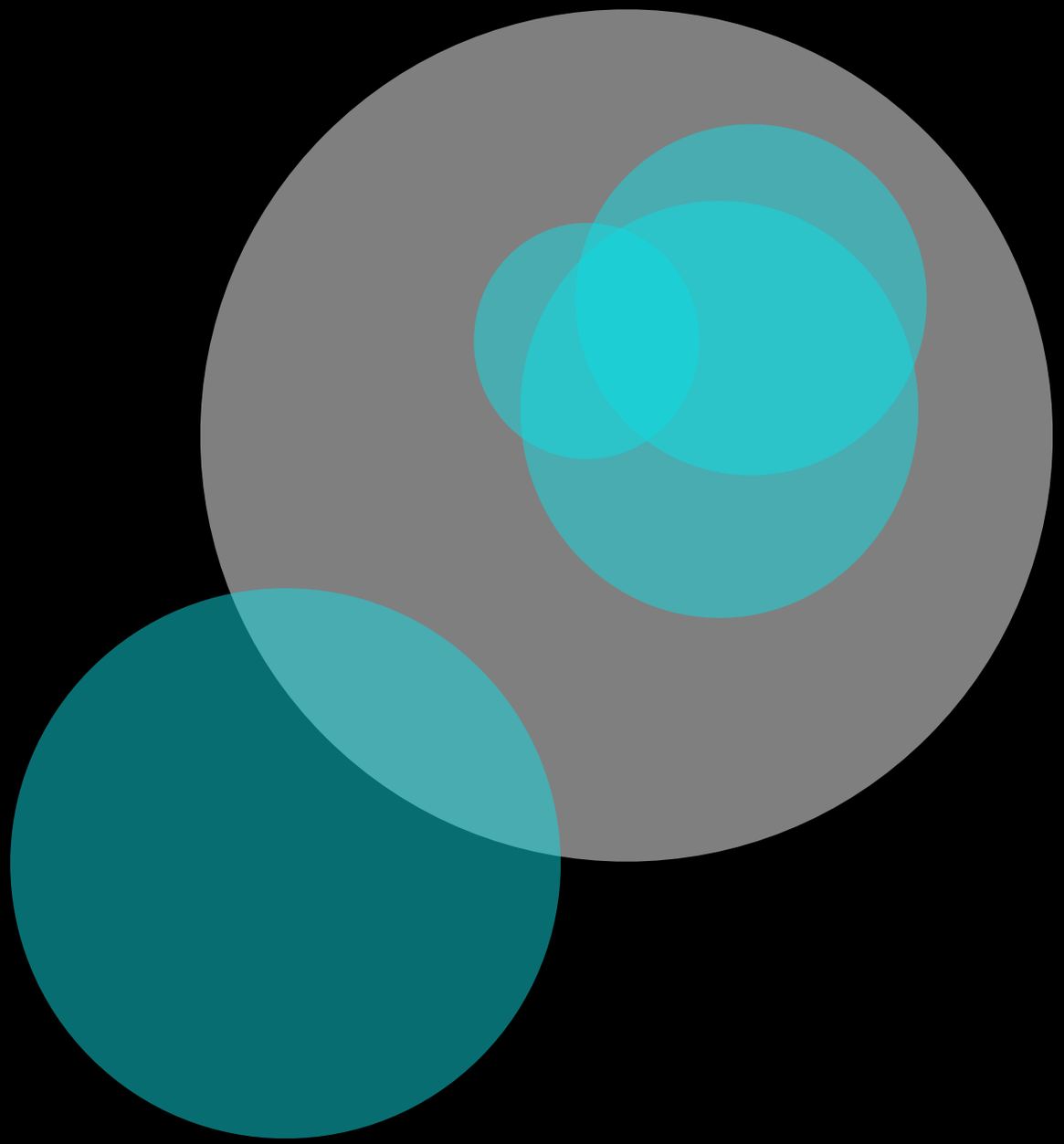


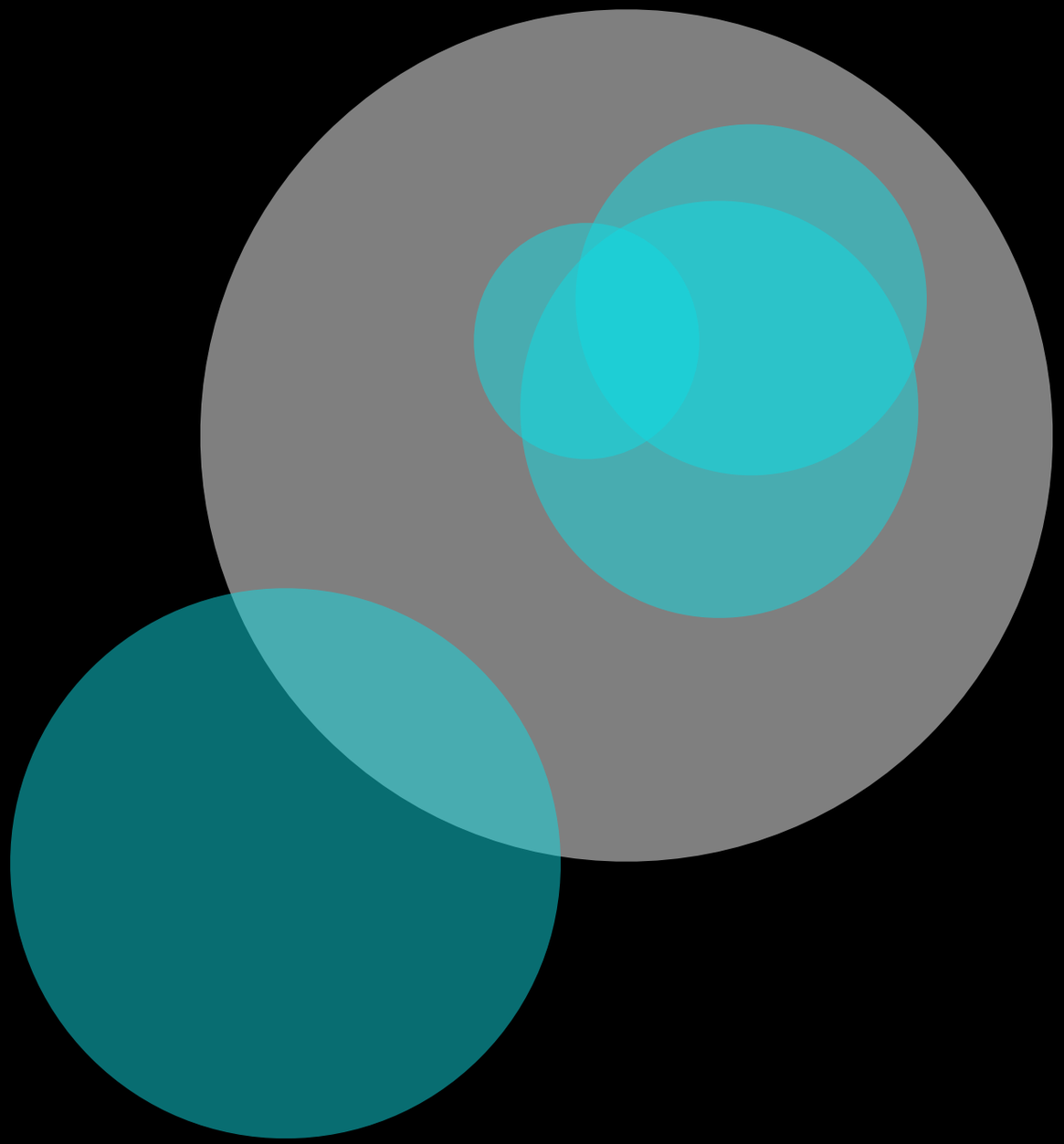












Competition has been fierce among the cellphone providers in the town of Eastern WestField. Several companies have been running ads in which they each claim to provide more coverage to the town than do any of their competitors.

AetherTech Telecommunications suspects that they, in fact, really do cover a larger portion of the town than do their competitors. They would like to compute their actual coverage so thay they can advertise the true amount.

The metropolitan area of EasternWestField will be modeled as a circle around the town center. The area covered by each cell tower can also be modeled as a circle, centered on the location of the tower, indicating the area within which a the tower provides sufficient signal strength for a phone to connect (e.g., one bar on the average phone s signal strength meter).

All cell towers to be considered will have their centers within the metropolitan area, though their area of coverage may extend beyond the border of the metropolitan area. The areas covered by different towers can overlap, though for economic reasons no two towers have been constructed so close to one another that the center of one would lie within the coverage area of the other.

Compute the fraction of the metropolitan area within which a cell phone would be covered by at least one tower.

Input

Input consists of multiple data sets. Each data set begins with a line containing a single integer N, $1 \le N \le 25$. This line is followed by N lines, each containing three floating point numbers x, y, r. These give the (x,y) coordinates of the circle s center and the radius of that circle, respectively. The first circle in the list denotes the metropolitan area of Eastern Westfield. Each of the remaining N-1 circles describes a cell tower.

The final data set is followed by a line containing a zero.

Output

For each data set, compute the fraction of the metropolitan area covered by the cell towers (as a number in the range 0.00...1.00 and print a line containing that fraction as a real number presented to 2 decimal points precision.

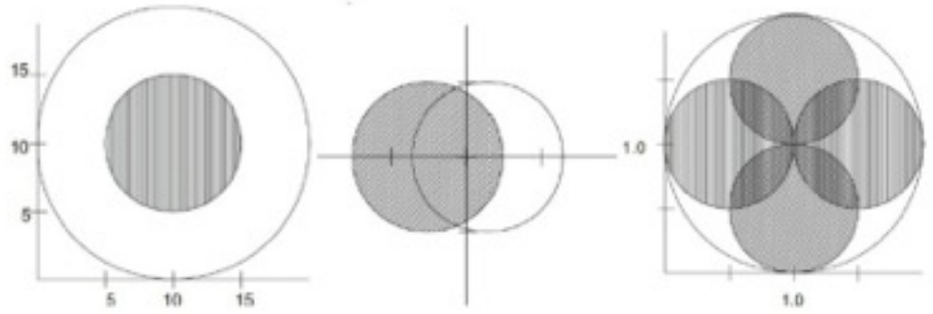
Sample Input

```
2
10 10 10
10 10 5
2
0.404 0.0 1.0
-0.404 0.0 1.0
5
0 0 2
1 0 1
0 1 1
-1 0 1
0 -1 1
0
```

Sample Output

```
0.25
0.50
0.82
```

Note: The images corresponding to the sample input are:



Competition has been fierce among the cellphone providers in the town of Eastern WestField. Several companies have been running ads in which they each claim to provide more coverage to the town than do any of their competitors.

AetherTech Telecommunications suspects that they, in fact, really do cover a larger portion of the town than do their competitors. They would like to compute their actual coverage so thay they can advertise the true amount.

The metropolitan area of EasternWestField will be modeled as a circle around the town center. The area covered by each cell tower can also be modeled as a circle, centered on the location of the tower, indicating the area within which a the tower provides sufficient signal strength for a phone to connect (e.g., one bar on the average phone s signal strength meter).

All cell towers to be considered will have their centers within the metropolitan area, though their area of coverage may extend beyond the border of the metropolitan area. The areas covered by different towers can overlap, though for economic reasons no two towers have been constructed so close to one another that the center of one would lie within the coverage area of the other.

Compute the fraction of the metropolitan area within which a cell phone would be covered by at least one tower.

Input

Input consists of mu
This line is followe
coordinates of the circle's center and the radius of that circle, respectively. The first circle in the list denotes the metropolitan area of Eastern Westfield. Each of the remaining N-1 circles describes a cell tower.

The final data set is followed by a line containing a zero.

Output

For each data set, compute the fraction of the metropolitan area covered by the cell towers (as a number in the range 0.00...1.00 and print a line containing that fraction as a real number presented to 2 decimal points precision.

Sample Input

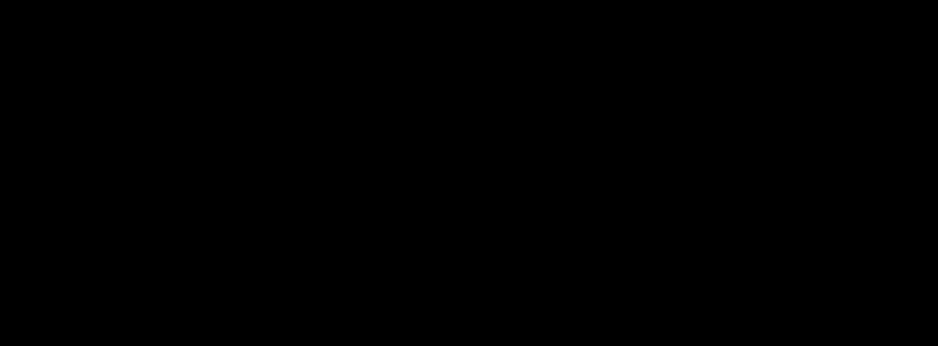
```
2
10 10 10
10 10 5
2
0.404 0.0 1.0
-0.404 0.0 1.0
5
0 0 2
1 0 1
0 1 1
```

```
-1 0 1
0 -1 1
0
```

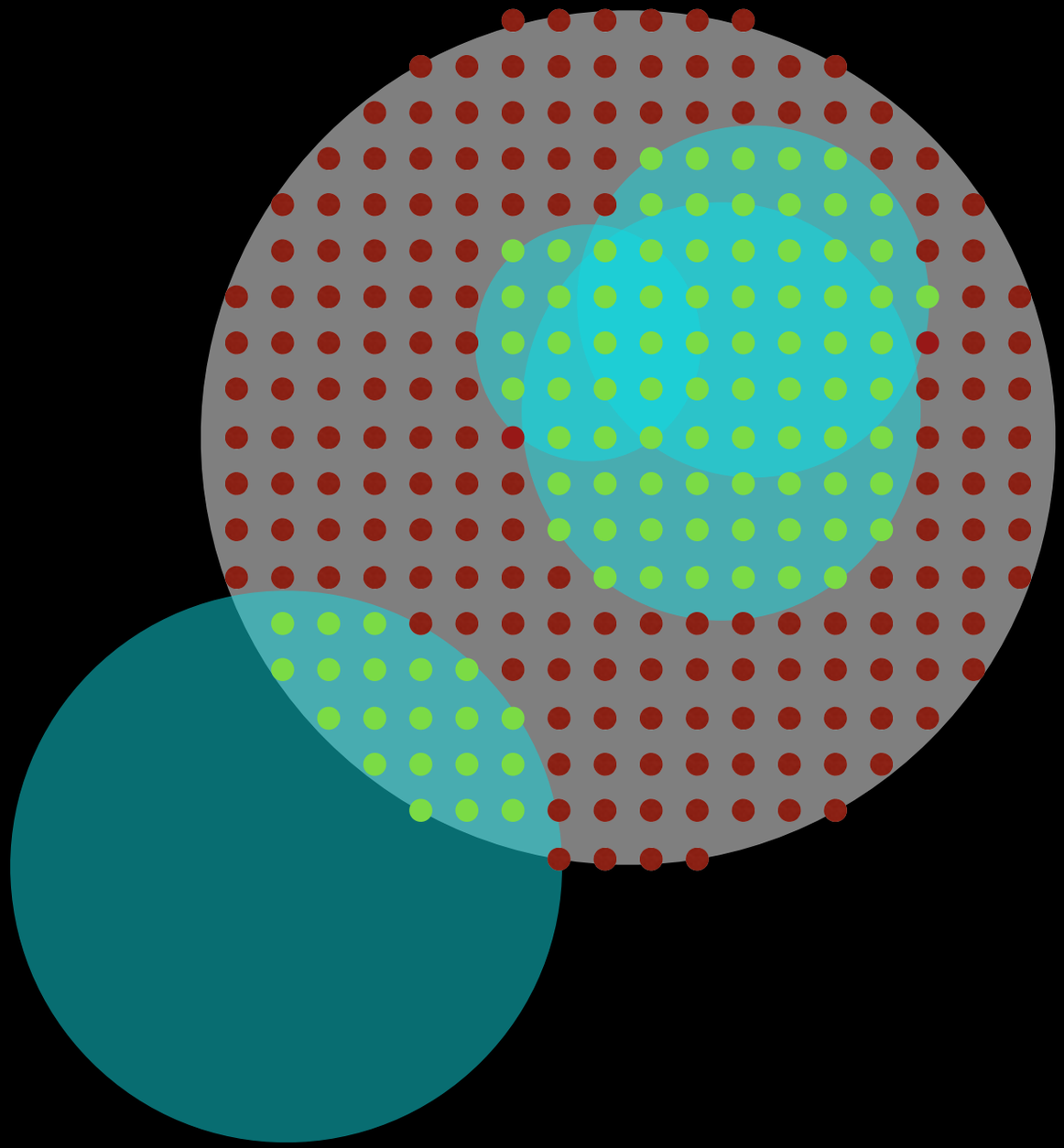
Sample Output

```
0.25
0.50
0.82
```

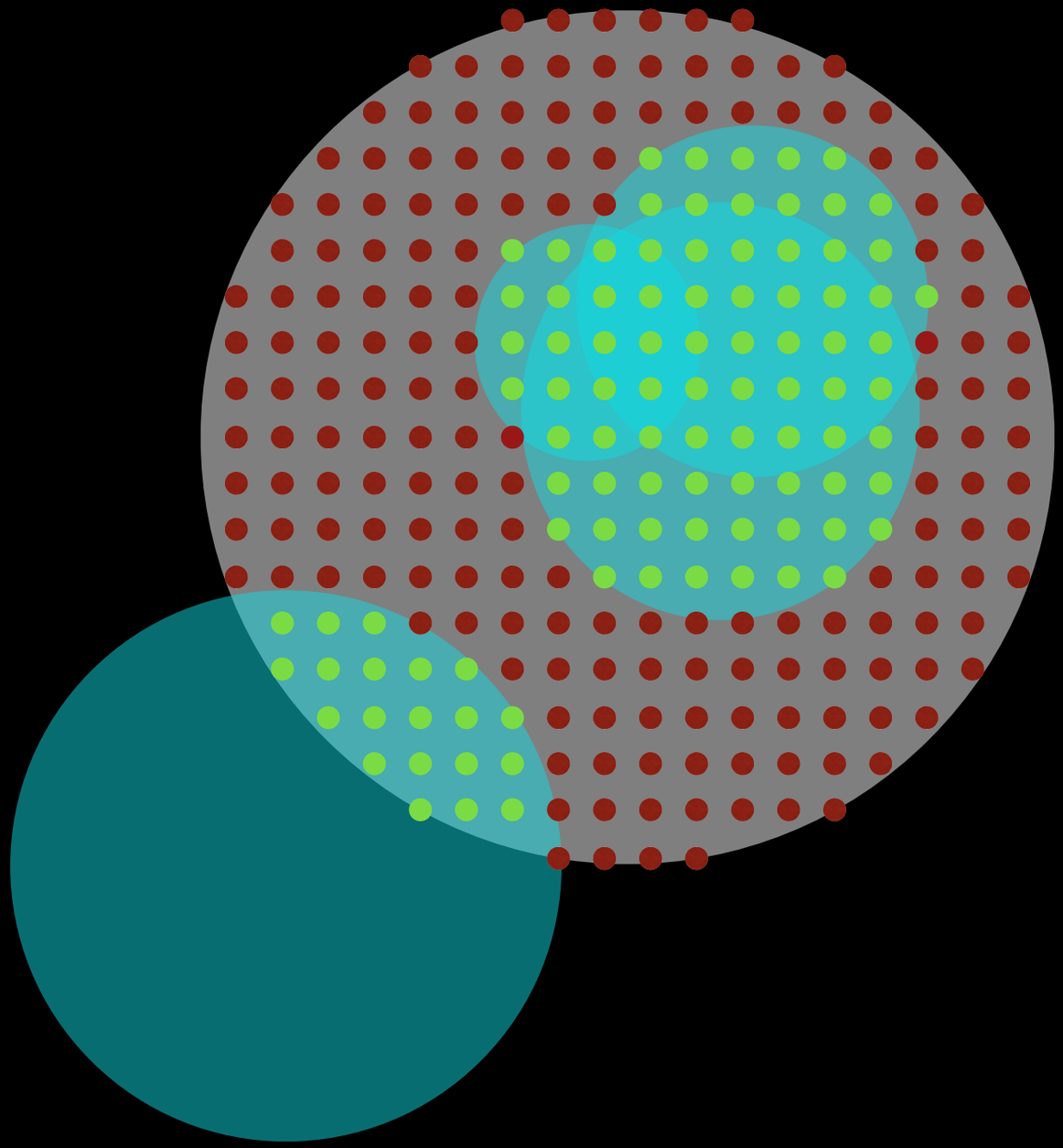
Note: The images corresponding to the sample input are:



For each data set, compute the fraction of the metropolitan area covered by the cell towers (as a number in the range 0.00...1.00 and print a line containing that fraction as a real number presented to 2 decimal points precision.



500 * 500 = 250,000 random samples
worst case check every circle, N <= 25
250,000 * 25 = 6,250,000 -> Manageable!



```java
import java.util.*;
import java.io.*;

public class cells {
  public static final double SAMPLE = 1000;

  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    while(true) {
      int N = in.nextInt();
      if(N==0)
        break;
      double X = in.nextDouble(), Y = in.nextDouble(), R = in.nextDouble();
      double[] x = new double[N-1], y = new double[N-1], r = new double[N-1];
      for(int i=0;i<N-1;i++) {
        x[i] = in.nextDouble();
        y[i] = in.nextDouble();
        r[i] = in.nextDouble();
      }
      int inRange = 0;
      int covered = 0;
      for(double a = X-R; a<=X+R; a+= R/SAMPLE)
        for(double b = Y-R; b<=Y+R; b+= R/SAMPLE) {
          if(distSQ(X, Y, a, b) <= R*R) {
            inRange++;
            for(int i=0;i<N-1;i++) {
              if(distSQ(x[i],y[i], a, b) <= r[i]*r[i]) {
                covered++;
                break;
              }
            }
          }
        }
      System.out.printf("%.02f\n", ((double)covered)/inRange);
    }
  }

  public static double distSQ(double x1, double y1, double x2, double y2) {
    double dx = x1-x2, dy=y1-y2;
    return dx*dx+dy*dy;
  }
}
```

1 try, <10 minutes

#9: Makes you appreciate algorithms, data structures

Problem J
Oil Skimming

Thanks to a certain "green" resources company, there is a new profitable industry of oil skimming. There are large slicks of crude oil floating in the Gulf of Mexico just waiting to be scooped up by enterprising oil barons. One such oil baron has a special plane that can skim the surface of the water collecting oil on the water's surface. However, each scoop covers a 10m by 20m rectangle (going either east/west or north/south). It also requires that the rectangle be completely covered in oil, otherwise the product is contaminated by pure ocean water and thus unprofitable!

Given a map of an oil slick, the oil baron would like you to compute the maximum number of scoops that may be extracted. The map is an NxN grid where each cell represents a 10m square of water, and each cell is marked as either being covered in oil or pure water.

Input

The input starts with an integer K ($1 \leq K \leq 100$) indicating the number of cases. Each case starts with an integer N ($1 \leq N \leq 600$) indicating the size of the square grid. Each of the following N lines contains N characters that represent the cells of a row in the grid. A character of '#' represents an oily cell, and a character of '.' represents a pure water cell.

Output

For each case, one line should be produced, formatted exactly as follows: "Case X: M" where X is the case number (starting from 1) and M is the maximum number of scoops of oil that may be extracted.

Sample Input	Output for the Sample Input
1 6 ..... .##... .##... ....#. ....##	Case 1: 3

Problem J
Oil Skimming

Thanks to a certain "green" resources company, there is a new profitable industry of oil skimming. There are large slicks of crude oil floating in the Gulf of Mexico just waiting to be scooped up by enterprising oil barons. One such oil baron has a special plane that can skim the surface of the water collecting oil on the water's surface. However, each scoop covers a 10m by 20m rectangle (going either east/west or north/south). It also requires that the rectangle be completely covered in oil, otherwise the product is contaminated by pure ocean water and thus unprofitable!

Given a map of an oil slick, the oil baron would like you to compute the maximum number of scoops that may be extracted. The map is an NxN grid where each cell represents a 10m square of water, and each cell is marked as either being covered in oil or pure water.

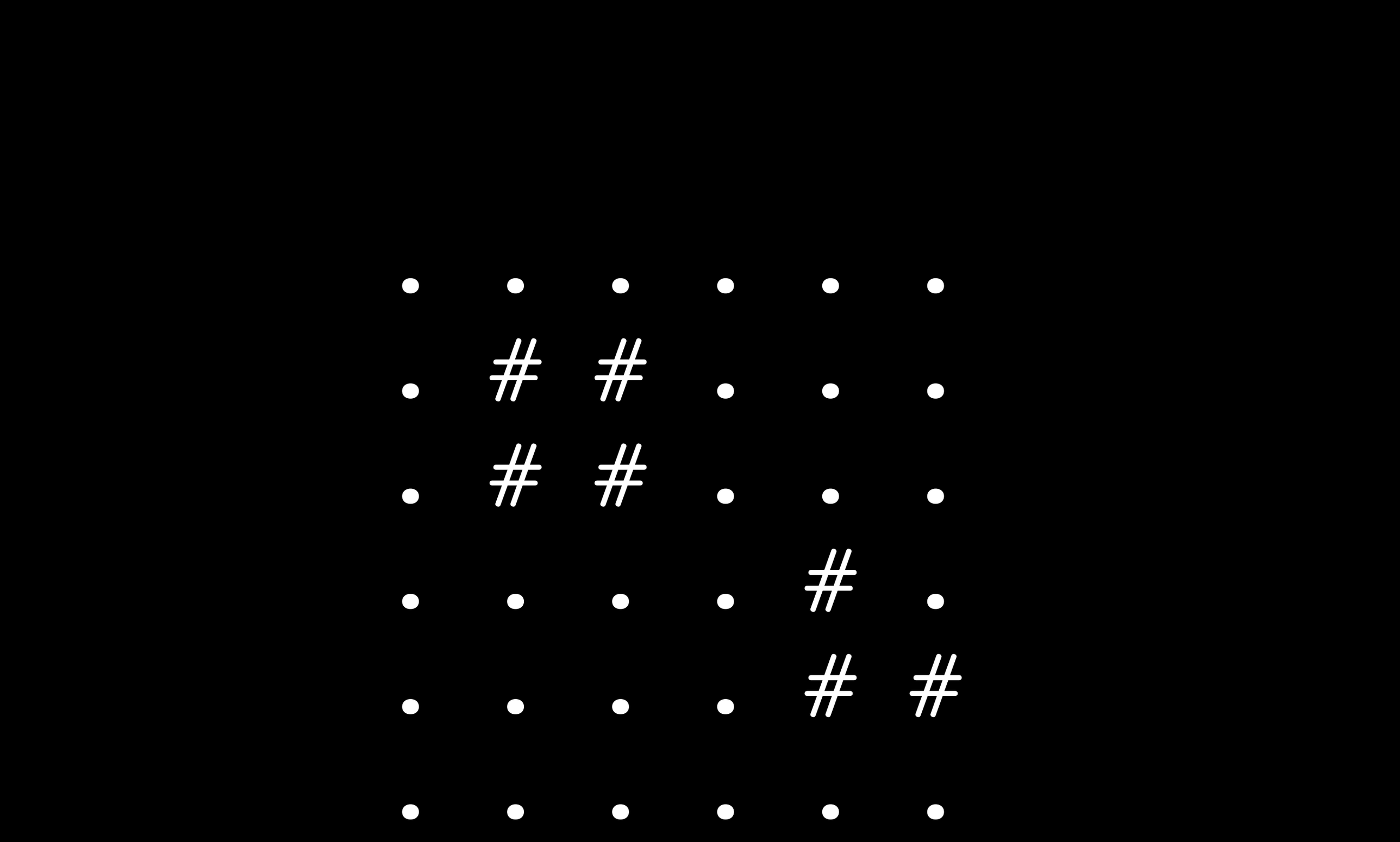
Input

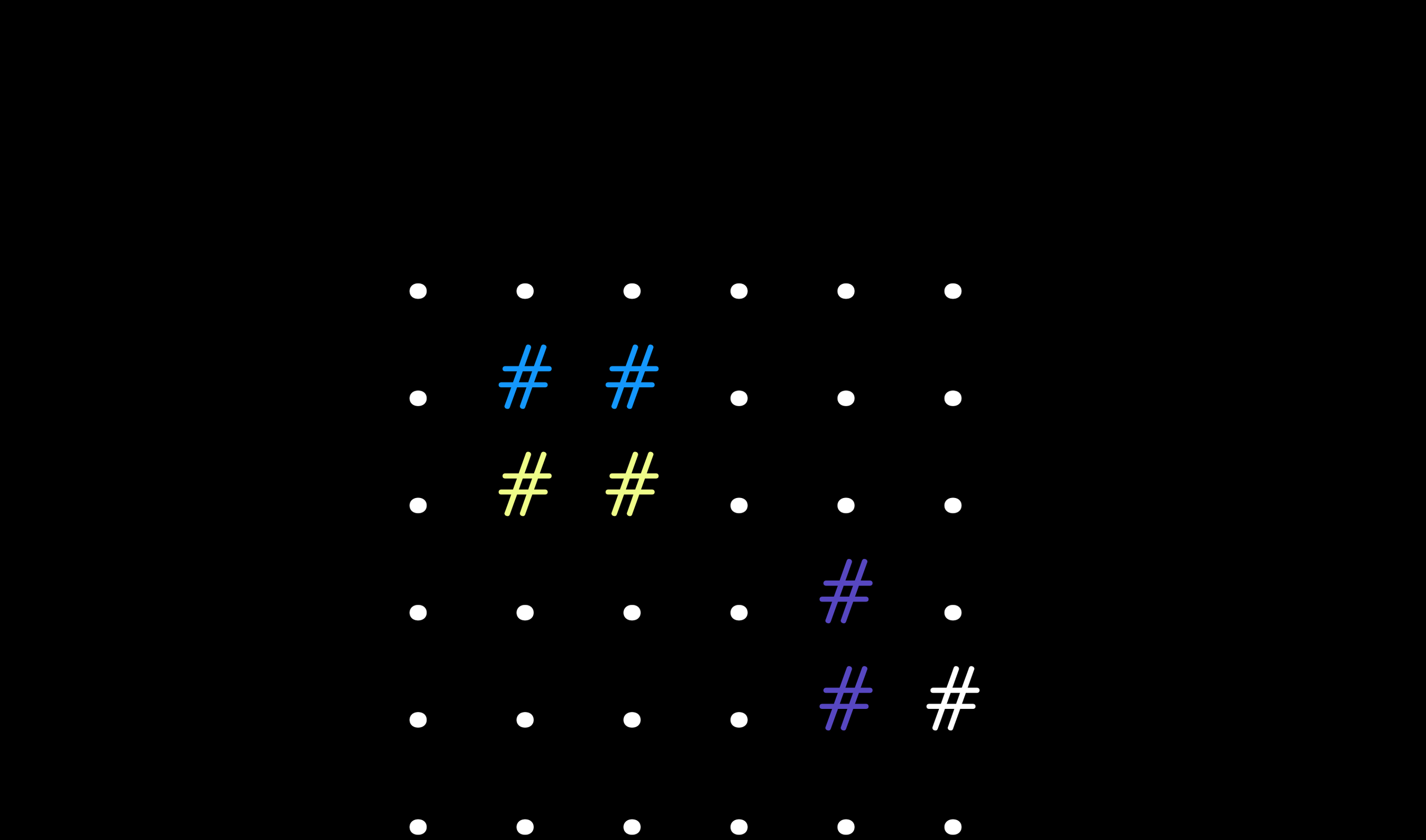
The input starts with an integer K ($1 \leq K \leq 100$) indicating the number of cases. Each case starts with an integer N ($1 \leq N \leq 600$) indicating the size of the square grid. Each of the following N lines contains N characters that represent the cells of a row in the grid. A character of '#' represents an oily cell, and a character of '.' represents a pure water cell.

Output

For each case, one line should be produced, formatted exactly as follows: "Case X: M" where X is the case number (starting from 1) and M is the maximum number of scoops of oil that may be extracted.

Sample Input	Output for the Sample Input
1 6 ..... .##... .##... ....#. ....##	Case 1: 3

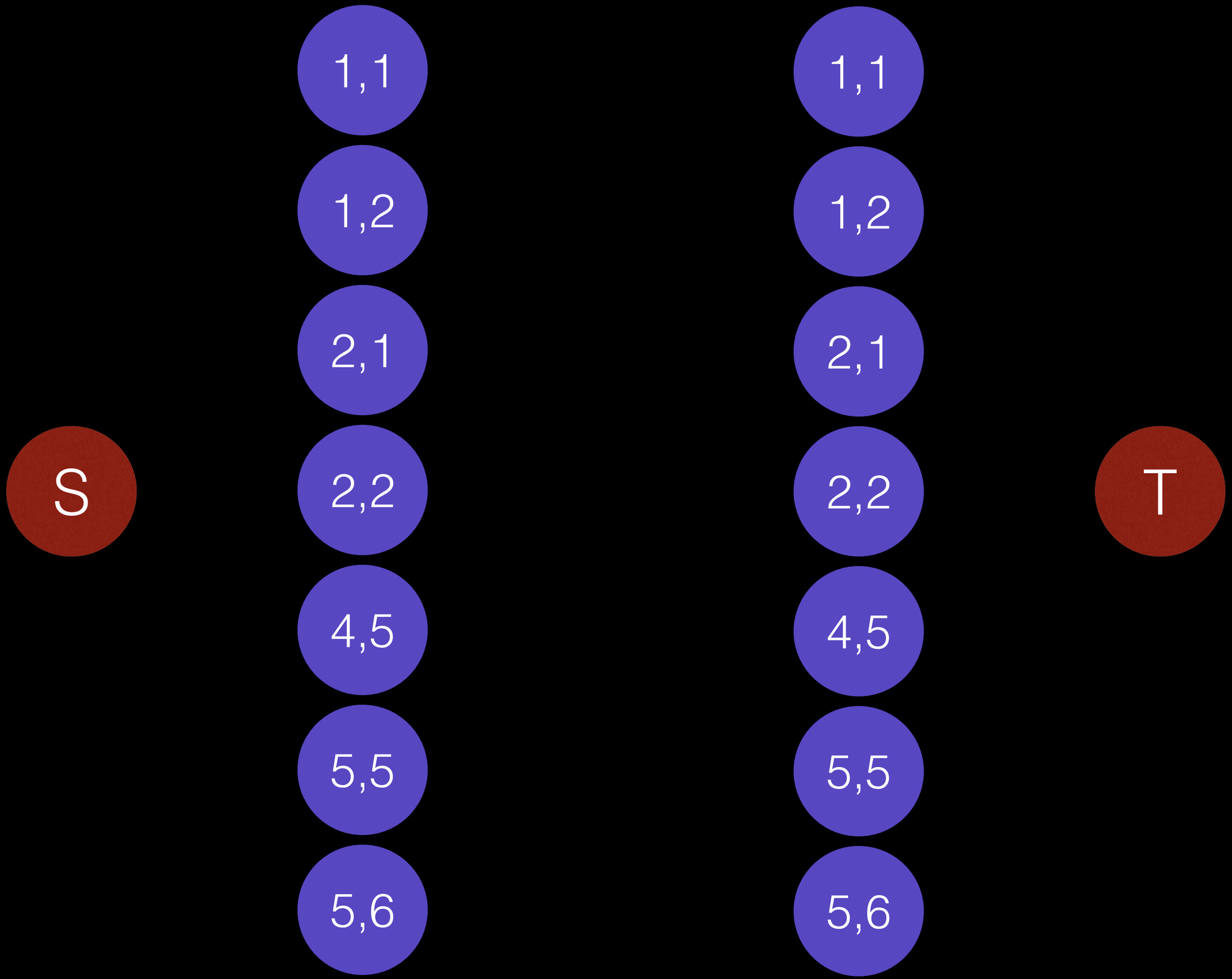


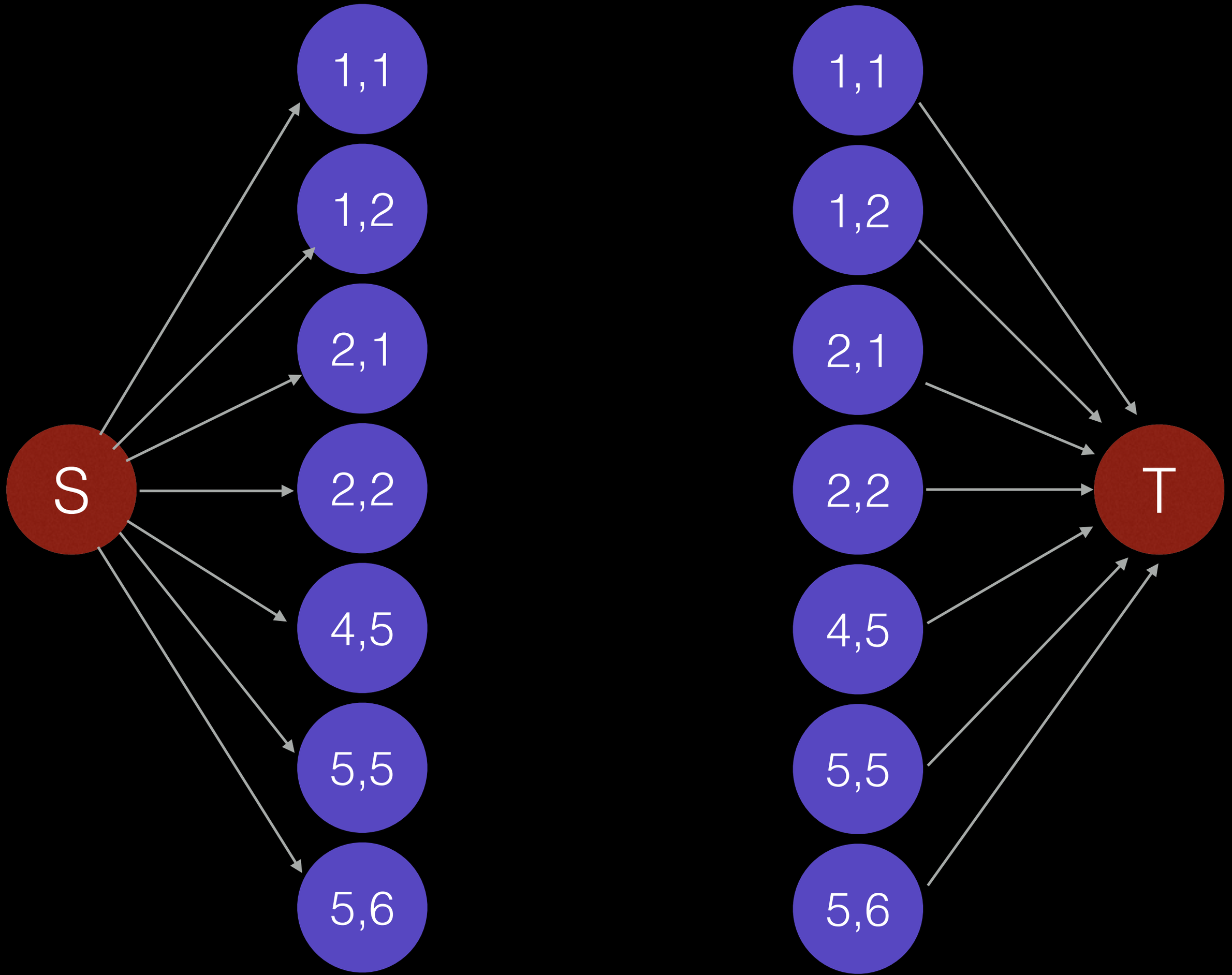


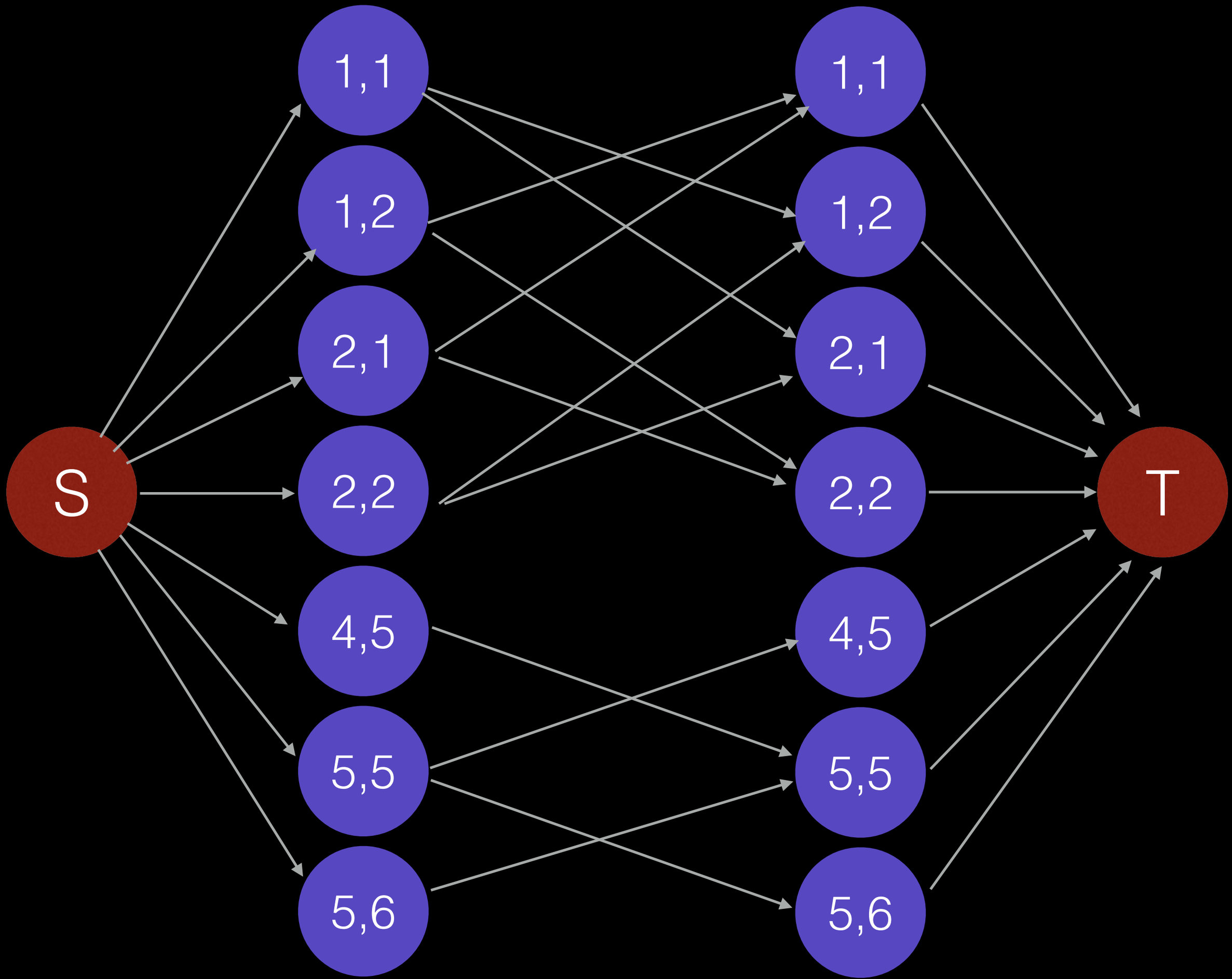
This is actually a max flow problem!

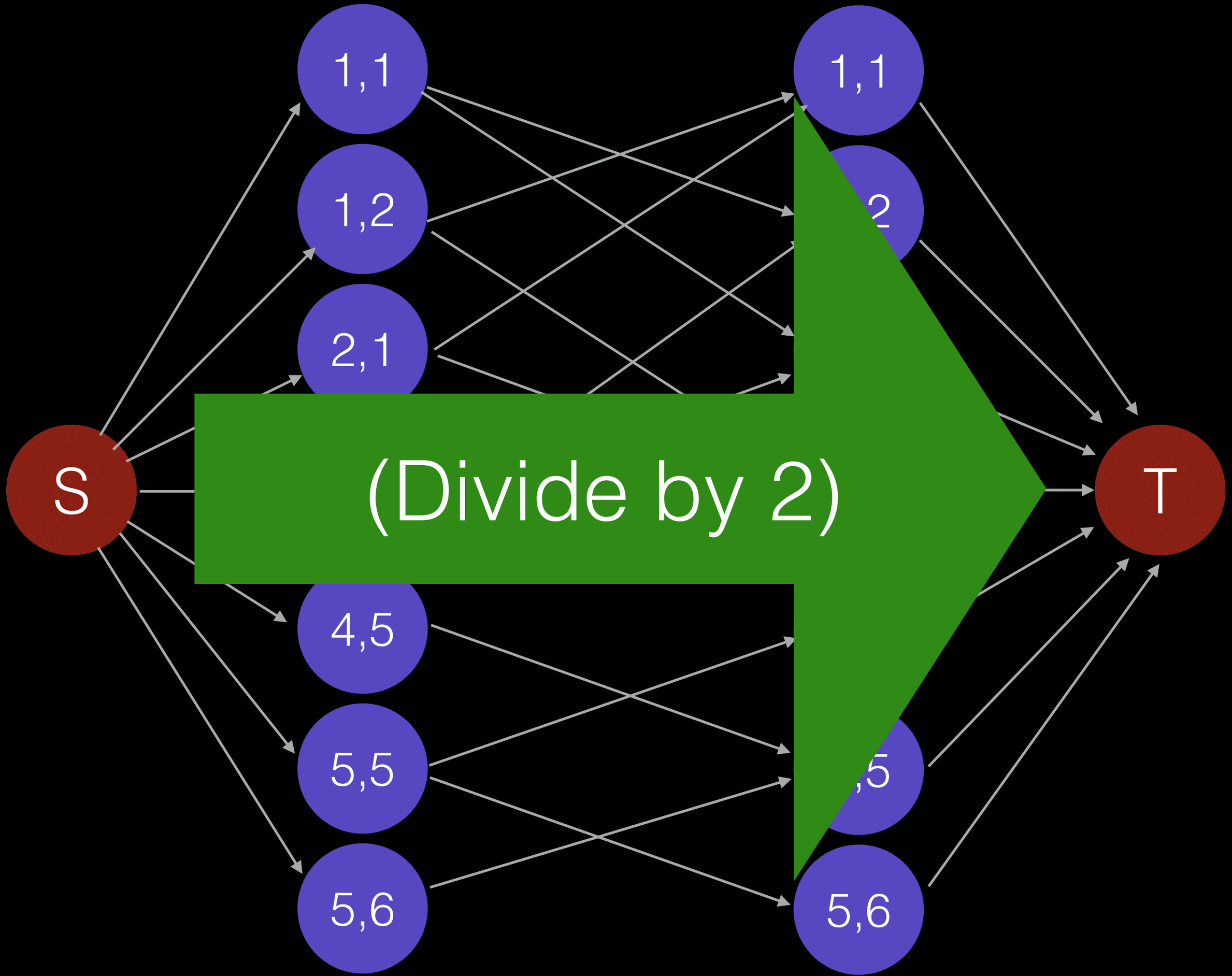
S

T


1,1
1,2
2,1
2,2
4,5
5,5
5,6
S
T
1,1
1,2
2,1
2,2
4,5
5,5
5,6


1,1
1,2
2,1
2,2
S
4,5
5,5
5,6
1,1
1,2
2,1
2,2
T
4,5
5,5
5,6


1,1
1,2
2,1
2,2
4,5
5,5
5,6
1,1
1,2
2,1
2,2
4,5
5,5
5,6
S
T


1,1
1,2
2,1
1,1
2
S
T
(Divide by 2)
4,5
5,5
5,6
,5
5,6

```java
import java.util.*;
import java.io.*;

public class oil {

	static HashMap<String, Node> nodes = new HashMap<String, Node>();

	public static void main(String[] args) {
		Scanner in = new Scanner(System.in);

		int numCases = in.nextInt();
		for (int c = 1; c <= numCases; c++) {
			nodes.clear();

			int N = in.nextInt();
			boolean[][] grid = new boolean[N][N];
			for (int i = 0; i < N; i++) {
				String s = in.next();
				for (int j = 0; j < N; j++)
					grid[i][j] = s.charAt(j)=='#';
			}

			Node source = new Node();
			Node sink = new Node();
			source.isSourceOrSink = true;
			sink.isSourceOrSink = true;
			int numNodes = 2;

			Node[][] aNodes = new Node[N][N];
			Node[][] bNodes = new Node[N][N];
			for (int i = 0; i < N; i++)
				for (int j = 0; j < N; j++) {
					if (grid[i][j]) {
						Node n = new Node();
						new Edge(n, source, 1, false);
						aNodes[i][j] = n;
						n = new Node();
						new Edge(sink, n, 1, false);
						bNodes[i][j] = n;
						numNodes += 2;
					}
				}
			source.height = numNodes;
			source.excess = Integer.MAX_VALUE;
			for (int i = 0; i < N; i++)
				for (int j = 0; j < N; j++) {
					if (grid[i][j]) {
						if (i - 1 >= 0 && grid[i - 1][j])
							new Edge(bNodes[i-1][j], aNodes[i][j], 1, false);
						if (j - 1 >= 0 && grid[i][j - 1])
							new Edge(bNodes[i][j-1], aNodes[i][j], 1, false);
						if (i + 1 < N && grid[i + 1][j])
							new Edge(bNodes[i+1][j], aNodes[i][j], 1, false);
						if (j + 1 < N && grid[i][j + 1])
							new Edge(bNodes[i][j+1], aNodes[i][j], 1, false);
					}
				}
			PreflowPush(source, sink);
			System.out.printf("Case %d: %d\n", c, sink.excess/2);
		}

	}
```

```java
	public static void PreflowPush(Node source, Node sink) {
		// source.height should be the number of nodes, sink.height should be 1
		LinkedList<Node> q = new LinkedList<Node>();
		q.add(source);
		while (!q.isEmpty()) {
			Node node = q.remove();
			int minHeight = Integer.MAX_VALUE;
			for (int i = 0; i < node.neighbors.size(); i++) {
				if (node.neighbors.get(i).dest.height < minHeight
						&& node.neighbors.get(i).remaining() > 0)
					minHeight = node.neighbors.get(i).dest.height;
			}
			if (minHeight != Integer.MAX_VALUE && minHeight >= node.height)
				node.height = minHeight + 1;
			for (int i = 0; i < node.neighbors.size(); i++) {
				if (node.neighbors.get(i).dest.height < node.height) {
					int pushedFlow = node.neighbors.get(i).remaining();
					if (pushedFlow > node.excess)
						pushedFlow = node.excess;
					if (pushedFlow > 0) {
						node.neighbors.get(i).flow += pushedFlow;
						node.neighbors.get(i).back.flow -= pushedFlow;
						node.neighbors.get(i).dest.excess += pushedFlow;
						node.excess -= pushedFlow;
						if (!node.neighbors.get(i).dest.isSourceOrSink)
							q.add(node.neighbors.get(i).dest);
						if (node.excess <= 0)
							break;
					}
				}
			}
			if (node.excess > 0 && !node.isSourceOrSink)
				q.add(node);
		}
	}

	class Node {
		public ArrayList<Edge> neighbors = new ArrayList<Edge>();
		public int height = 1;
		public int excess = 0;
		public boolean isSourceOrSink = false;
	}

	class Edge {
		public Node dest;
		public Edge back;
		public int capacity;
		public int flow = 0;

		public Edge(Node dest, Node source, int capacity, boolean isBack) {
			this.dest = dest;
			this.capacity = capacity;
			source.neighbors.add(this);
			if (!isBack) {
				Edge backEdge = new Edge(source, this.dest, 0, true);
				this.back = backEdge;
				backEdge.back = this;
			}
		}

		public int remaining() {
			return capacity - flow;
		}
	}
```

1 try, ~1 hour (copying/understanding max flow code)

#10: You get to hang out with friends, goof off, and write bad code you'll never look at again.

How can I help you?

https://github.com/depstein/programming-competitions