6.2 **Basketball:** You have a basketball hoop and someone says that you can play one of two games.

Game 1: You get one shot to make the hoop.

Game 2: You get three shots and you have to make two of three shots.

If p is the probability of making a particular shot, for which values of p should you pick one game or the other?

pg 123

SOLUTION

To solve this problem, we can apply straightforward probability laws by comparing the probabilities of winning each game.

Probability of winning Game 1:

The probability of winning Game 1 is p, by definition.

Probability of winning Game 2:

Let `s(k,n)` be the probability of making exactly k shots out of n. The probability of winning Game 2 is the probability of making exactly two shots out of three OR making all three shots. In other words:

```
P(winning) = s(2,3) + s(3,3)
```

The probability of making all three shots is:

```
s(3,3) = p^3
```

The probability of making exactly two shots is:

```
P(making 1 and 2, and missing 3)
        + P(making 1 and 3, and missing 2)
        + P(missing 1, and making 2 and 3)
    = p * p * (1 - p) + p * (1 - p) * p + (1 - p) * p * p
    = 3 (1 - p) p^2
```

Adding these together, we get:

```
    = p^3 + 3 (1 - p) p^2
    = p^3 + 3p^2 - 3p^3
    = 3p^2 - 2p^3
```

Which game should you play?

You should play Game 1 if `P(Game 1) > P(Game 2)`:

```
p > 3p^2 - 2p^3.
1 > 3p - 2p^2
2p^2 - 3p + 1 > 0
(2p - 1)(p - 1) > 0
```

Both terms must be positive, or both must be negative. But we know `p < 1`, so `p - 1 < 0`. This means both terms must be negative.

```
2p - 1 < 0
2p < 1
p < .5
```

So, we should play Game 1 if `0 < p < .5` and Game 2 if `.5 < p < 1`.

If `p = 0, 0.5,` or `1`, then `P(Game 1) = P(Game 2)`, so it doesn't matter which game we play.

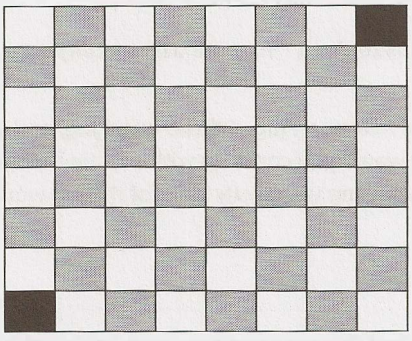
6.3 **Dominos:** There is an 8x8 chessboard in which two diagonally opposite corners have been cut off. You are given 31 dominos, and a single domino can cover exactly two squares. Can you use the 31 dominos to cover the entire board? Prove your answer (by providing an example or showing why it's impossible).

pg 123

SOLUTION

At first, it seems like this should be possible. It's an 8 x 8 board, which has 64 squares, but two have been cut off, so we're down to 62 squares. A set of 31 dominoes should be able to fit there, right?

When we try to lay down dominoes on row 1, which only has 7 squares, we may notice that one domino must stretch into the row 2. Then, when we try to lay down dominoes onto row 2, again we need to stretch a domino into row 3.



For each row we place, we'll always have one domino that needs to poke into the next row. No matter how many times and ways we try to solve this issue, we won't be able to successfully lay down all the dominoes.

There's a cleaner, more solid proof for why it won't work. The chessboard initially has 32 black and 32 white squares. By removing opposite corners (which must be the same color), we're left with 30 of one color and 32 of the other color. Let's say, for the sake of argument, that we have 30 black and 32 white squares.

Each domino we set on the board will always take up one white and one black square. Therefore, 31 dominos will take up 31 white squares and 31 black squares exactly. On this board, however, we must have 30 black squares and 32 white squares. Hence, it is impossible.

6.4 **Ants on a Triangle:** There are three ants on different vertices of a triangle. What is the probability of collision (between any two or all of them) if they start walking on the sides of the triangle? Assume that each ant randomly picks a direction, with either direction being equally likely to be chosen, and that they walk at the same speed.

Similarly, find the probability of collision with n ants on an n-vertex polygon.

pg 123

SOLUTION

The ants will collide if any of them are moving towards each other. So, the only way that they won't collide is if they are all moving in the same direction (clockwise or counterclockwise). We can compute this probability and work backwards from there.

Since each ant can move in two directions, and there are three ants, the probability is:

$$P(\text{clockwise}) = (1/2)^3$$
$$P(\text{counter clockwise}) = (1/2)^3$$
$$P(\text{same direction}) = (1/2)^3 + (1/2)^3 = 1/4$$

The probability of collision is therefore the probability of the ants *not* moving in the same direction:

$$P(\text{collision}) = 1 - P(\text{same direction}) = 1 - 1/4 = 3/4$$

To generalize this to an n-vertex polygon: there are still only two ways in which the ants can move to avoid a collision, but there are 2^n ways they can move in total. Therefore, in general, probability of collision is:

$$P(\text{clockwise}) = (1/2)^n$$
$$P(\text{counter}) = (1/2)^n$$
$$P(\text{same direction}) = 2(1/2)^n = (1/2)^{n-1}$$
$$P(\text{collision}) = 1 - P(\text{same direction}) = 1 - (1/2)^{n-1}$$

6.5 **Jugs of Water:** You have a five-quart jug, a three-quart jug, and an unlimited supply of water (but no measuring cups). How would you come up with exactly four quarts of water? Note that the jugs are oddly shaped, such that filling up exactly "half" of the jug would be impossible.

pg 123

SOLUTION

If we just play with the jugs, we'll find that we can pour water back and forth between them as follows:

5 Quart	3 Quart	Action
5	0	Filled 5-quart jug.
2	3	Filled 3-quart with 5-quart's contents.
2	0	Dumped 3-quart.
0	2	Fill 3-quart with 5-quart's contents.
5	2	Filled 5-quart.
4	3	Fill remainder of 3-quart with 5-quart.
4		Done! We have 4 quarts.

This question, like many puzzle questions, has a math/computer science root. If the two jug sizes are relatively prime, you can measure any value between one and the sum of the jug sizes.

6.6 **Blue-Eyed Island:** A bunch of people are living on an island, when a visitor comes with a strange order: all blue-eyed people must leave the island as soon as possible. There will be a flight out at 8:00pm every evening. Each person can see everyone else's eye color, but they do not know their own (nor is anyone allowed to tell them). Additionally, they do not know how many people have blue eyes, although they do know that at least one person does. How many days will it take the blue-eyed people to leave?

pg 123

SOLUTION

Let's apply the Base Case and Build approach. Assume that there are n people on the island and c of them have blue eyes. We are explicitly told that $c > 0$.

Case c = 1: Exactly one person has blue eyes.

Assuming all the people are intelligent, the blue-eyed person should look around and realize that no one else has blue eyes. Since he knows that at least one person has blue eyes, he must conclude that it is he who has blue eyes. Therefore, he would take the flight that evening.

Case c = 2: Exactly two people have blue eyes.

The two blue-eyed people see each other, but are unsure whether c is 1 or 2. They know, from the previous case, that if $c = 1$, the blue-eyed person would leave on the first night. Therefore, if the other blue-eyed person is still there, he must deduce that $c = 2$, which means that he himself has blue eyes. Both men would then leave on the second night.

Case c > 2: The General Case.

As we increase c, we can see that this logic continues to apply. If $c = 3$, then those three people will immediately know that there are either 2 or 3 people with blue eyes. If there were two people, then those two people would have left on the second night. So, when the others are still around after that night, each person would conclude that $c = 3$ and that they, therefore, have blue eyes too. They would leave that night.

This same pattern extends up through any value of c. Therefore, if c men have blue eyes, it will take c nights for the blue-eyed men to leave. All will leave on the same night.

6.7 **The Apocalypse**: In the new post-apocalyptic world, the world queen is desperately concerned about the birth rate. Therefore, she decrees that all families should ensure that they have one girl or else they face massive fines. If all families abide by this policy—that is, they have continue to have children until they have one girl, at which point they immediately stop—what will the gender ratio of the new generation be? (Assume that the odds of someone having a boy or a girl on any given pregnancy is equal.) Solve this out logically and then write a computer simulation of it.

pg 123

SOLUTION

If each family abides by this policy, then each family will have a sequence of zero or more boys followed by a single girl. That is, if "G" indicates a girl and "B" indicates a boy, the sequence of children will look like one of: G; BG; BBG; BBBG; BBBBG; and so on.

We can solve this problem multiple ways.

Mathematically

We can work out the probability for each gender sequence.

- P(G) = $1/2$. That is, 50% of families will have a girl first. The others will go on to have more children.
- P(BG) = $1/4$. Of those who have a second child (which is 50%), 50% of them will have a girl the next time.
- P(BBG) = $1/8$. Of those who have a third child (which is 25%), 50% of them will have a girl the next time.

And so on.

We know that every family has exactly one girl. How many boys does each family have, on average? To compute this, we can look at the expected value of the number of boys. The expected value of the number of boys is the probability of each sequence multiplied by the number of boys in that sequence.

Sequence	Number of Boys	Probability	Number of Boys * Probability
G	0	$1/2$	0
BG	1	$1/4$	$1/4$
BBG	2	$1/8$	$2/8$
BBBG	3	$1/16$	$3/16$
BBBBG	4	$1/32$	$4/32$
BBBBBG	5	$1/64$	$5/64$
BBBBBBG	6	$1/128$	$6/128$

Or in other words, this is the sum of `i` to infinity of i divided by 2^i.

$$\sum_{i=0}^{\infty} \frac{i}{2^i}$$

You probably won't know this off the top of your head, but we can try to estimate it. Let's try converting the above values to a common denominator of 128 (2^6).

$$1/4 = 32/128 \qquad 4/32 = 16/128$$

$$2/8 = 32/128 \qquad 5/64 = 10/128$$

$$3/16 = 24/128 \qquad 6/128 = 6/128$$

$$\frac{32 + 32 + 24 + 16 + 10 + 6}{128} = \frac{120}{128}$$

This looks like it's going to inch closer to $128/128$ (which is of course 1). This "looks like" intuition is valuable, but it's not exactly a mathematical concept. It's a clue though and we can turn to logic here. Should it be 1?

Logically

If the earlier sum is 1, this would mean that the gender ratio is even. Families contribute exactly one girl and on average one boy. The birth policy is therefore ineffective. Does this make sense?

At first glance, this seems wrong. The policy is designed to favor girls as it ensures that all families have a girl.

On the other hand, the families that keep having children contribute (potentially) multiple boys to the population. This could offset the impact of the "one girl" policy.

One way to think about this is to imagine that we put all the gender sequence of each family into one giant string. So if family 1 has BG, family 2 has BBG, and family 3 has G, we would write BGBBGG.

In fact, we don't really care about the groupings of families because we're concerned about the population as a whole. As soon as a child is born, we can just append its gender (B or G) to the string.

What are the odds of the next character being a G? Well, if the odds of having a boy and girl is the same, then the odds of the next character being a G is 50%. Therefore, roughly half of the string should be Gs and half should be Bs, giving an even gender ratio.

This actually makes a lot of sense. Biology hasn't been changed. Half of newborn babies are girls and half are boys. Abiding by some rule about when to stop having children doesn't change this fact.

Therefore, the gender ratio is 50% girls and 50% boys.

Simulation

We'll write this in a simple way that directly corresponds to the problem.

```
1  double runNFamilies(int n) {
2    int boys = 0;
3    int girls = 0;
4    for (int i = 0; i < n; i++) {
5      int[] genders = runOneFamily();
6      girls += genders[0];
7      boys += genders[1];
8    }
9    return girls / (double) (boys + girls);
10 }
11
12 int[] runOneFamily() {
13   Random random = new Random();
14   int boys = 0;
15   int girls = 0;
16   while (girls == 0) { // until we have a girl
17     if (random.nextBoolean()) { // girl
18       girls += 1;
19     } else { // boy
20       boys += 1;
21     }
22   }
23   int[] genders = {girls, boys};
24   return genders;
25 }
```

Sure enough, if you run this on large values of n, you should get something very close to 0.5.

6.8 **The Egg Drop Problem:** There is a building of 100 floors. If an egg drops from the Nth floor or above, it will break. If it's dropped from any floor below, it will not break. You're given two eggs. Find N, while minimizing the number of drops for the worst case.

pg 124

SOLUTION

We may observe that, regardless of how we drop Egg 1, Egg 2 must do a linear search (from lowest to highest) between the "breaking floor" and the next highest non-breaking floor. For example, if Egg 1 is dropped from floors 5 and 10 without breaking, but it breaks when it's dropped from floor 15, then Egg 2 must be dropped, in the worst case, from floors 11, 12, 13, and 14.

The Approach

As a first try, suppose we drop an egg from the 10th floor, then the 20th, ...

- If Egg 1 breaks on the first drop (floor 10), then we have at most 10 drops total.
- If Egg 1 breaks on the last drop (floor 100), then we have at most 19 drops total (floors 10, 20, ...,90, 100, then 91 through 99).

That's pretty good, but all we've considered is the absolute worst case. We should do some "load balancing" to make those two cases more even.

Our goal is to create a system for dropping Egg 1 such that the number of drops is as consistent as possible, whether Egg 1 breaks on the first drop or the last drop.

1. A perfectly load-balanced system would be one in which `Drops(Egg 1) + Drops(Egg 2)` is always the same, regardless of where Egg 1 breaks.
2. For that to be the case, since each drop of Egg 1 takes one more step, Egg 2 is allowed one fewer step.
3. We must, therefore, reduce the number of steps potentially required by Egg 2 by one drop each time. For example, if Egg 1 is dropped on floor 20 and then floor 30, Egg 2 is potentially required to take 9 steps. When we drop Egg 1 again, we must reduce potential Egg 2 steps to only 8. That is, we must drop Egg 1 at floor 39.
4. Therefore, Egg 1 must start at floor X, then go up by X-1 floors, then X-2, ..., until it gets to 100.
5. Solve for X.

$$X+(X-1)+(X-2)+...+1=100$$
$$X(X+1)/2=100$$
$$X\approx 13.65$$

X clearly needs to be an integer. Should we round X up or down?

- If we round X up to 14, then we would go up by 14, then 13, then 12, and so on. The last increment would be 4, and it would happen on floor 99. If Egg 1 broke on any of the prior floors, we know we've balanced the eggs such that the number of drops of Egg 1 and Egg 2 always sum to the same thing: 14. If Egg 1 hasn't broken by floor 99, then we just need one more drop to determine if it will break at floor 100. Either way, the number of drops is no more than 14.
- If we round X down to 13, then we would go up by 13, then 12, then 11, and so on. The last increment will be 1 and it will happen at floor 91. This is after 13 drops. Floors 92 through 100 have not been covered yet. We can't cover those floors in just one drop (which would be necessary to merely tie the

"round up" case).

Therefore, we should round X up to 14. That is, we go to floor 14, then 27, then 39, ... This takes 14 steps in the worse case.

As in many other maximizing / minimizing problems, the key in this problem is "worst case balancing."

The following code simulates this approach.

```
int breakingPoint = ...;
int countDrops = 0;

boolean drop(int floor) {
  countDrops++;
  return floor >= breakingPoint;
}

int findBreakingPoint(int floors) {
  int interval = 14;
  int previousFloor = 0;
  int egg1 = interval;

  /* Drop egg1 at decreasing intervals. */
  while (!drop(egg1) && egg1 <= floors) {
    interval -= 1;
    previousFloor = egg1;
    egg1 += interval;
  }

  /* Drop egg2 at 1 unit increments. */
  int egg2 = previousFloor + 1;
  while (egg2 < egg1 && egg2 <= floors && !drop(egg2)) {
    egg2 += 1;
  }

  /* If it didn't break, return -1. */
  return egg2 > floors ? -1 : egg2;
}
```

If we want to generalize this code for more building sizes, then we can solve for x in:

$$x(x+1)/2 = \text{number of floors}$$

This will involve the quadratic formula.

6.9 100 Lockers: There are 100 closed lockers in a hallway. A man begins by opening all 100 lockers. Next, he closes every second locker. Then, on his third pass, he toggles every third locker (closes it if it is open or opens it if it is closed). This process continues for 100 passes, such that on each pass `i`, the man toggles every `i`th locker. After his 100th pass in the hallway, in which he toggles only locker #100, how many lockers are open?

pg 124

SOLUTION

We can tackle this problem by thinking through what it means for a door to be toggled. This will help us deduce which doors at the very end will be left opened.

Question: For which rounds is a door toggled (open or closed)?

A door n is toggled once for each factor of n, including itself and 1. That is, door 15 is toggled on rounds 1, 3, 5, and 15.

Question: When would a door be left open?

A door is left open if the number of factors (which we will call x) is odd. You can think about this by pairing factors off as an open and a close. If there's one remaining, the door will be open.

Question: When would x be odd?

The value x is odd if n is a perfect square. Here's why: pair n's factors by their complements. For example, if n is 36, the factors are (1, 36), (2, 18), (3, 12), (4, 9), (6, 6). Note that (6, 6) only contributes one factor, thus giving n an odd number of factors.

Question: How many perfect squares are there?

There are 10 perfect squares. You could count them (1, 4, 9, 16, 25, 36, 49, 64, 81, 100), or you could simply realize that you can take the numbers 1 through 10 and square them:

```
1*1, 2*2, 3*3, ..., 10*10
```

Therefore, there are 10 lockers open at the end of this process.

6.10 **Poison**: You have 1000 bottles of soda, and exactly one is poisoned. You have 10 test strips which can be used to detect poison. A single drop of poison will turn the test strip positive permanently. You can put any number of drops on a test strip at once and you can reuse a test strip as many times as you'd like (as long as the results are negative). However, you can only run tests once per day and it takes seven days to return a result. How would you figure out the poisoned bottle in as few days as possible?

Follow up: Write code to simulate your approach.

pg 124

SOLUTION

Observe the wording of the problem. Why seven days? Why not have the results just return immediately?

The fact that there's such a lag between starting a test and reading the results likely means that we'll be doing something else in the meantime (running additional tests). Let's hold on to that thought, but start off with a simple approach just to wrap our heads around the problem.

Naive Approach (28 days)

A simple approach is to divide the bottles across the 10 test strips, first in groups of 100. Then, we wait seven days. When the results come back, we look for a positive result across the test strips. We select the bottles associated with the positive test strip, "toss" (i.e., ignore) all the other bottles, and repeat the process. We perform this operation until there is only one bottle left in the test set.

1. Divide bottles across available test strips, one drop per test strip.
2. After seven days, check the test strips for results.
3. On the positive test strip: select the bottles associated with it into a new set of bottles. If this set size is 1,

we have located the poisoned bottle. If it's greater than one, go to step 1.

To simulate this, we'll build classes for `Bottle` and `TestStrip` that mirror the problem's functionality.

```
class Bottle {
  private boolean poisoned = false;
  private int id;

  public Bottle(int id) { this.id = id; }
  public int getId() { return id; }
  public void setAsPoisoned() { poisoned = true; }
  public boolean isPoisoned() { return poisoned; }
}

class TestStrip {
  public static int DAYS_FOR_RESULT = 7;
  private ArrayList<ArrayList<Bottle>> dropsByDay =
    new ArrayList<ArrayList<Bottle>>();
  private int id;

  public TestStrip(int id) { this.id = id; }
  public int getId() { return id; }

  /* Resize list of days/drops to be large enough. */
  private void sizeDropsForDay(int day) {
    while (dropsByDay.size() <= day) {
      dropsByDay.add(new ArrayList<Bottle>());
    }
  }

  /* Add drop from bottle on specific day. */
  public void addDropOnDay(int day, Bottle bottle) {
    sizeDropsForDay(day);
    ArrayList<Bottle> drops = dropsByDay.get(day);
    drops.add(bottle);
  }

  /* Checks if any of the bottles in the set are poisoned. */
  private boolean hasPoison(ArrayList<Bottle> bottles) {
    for (Bottle b : bottles) {
      if (b.isPoisoned()) {
        return true;
      }
    }
    return false;
  }

  /* Gets bottles used in the test DAYS_FOR_RESULT days ago. */
  public ArrayList<Bottle> getLastWeeksBottles(int day) {
    if (day < DAYS_FOR_RESULT) {
      return null;
    }
    return dropsByDay.get(day - DAYS_FOR_RESULT);
  }

  /* Checks for poisoned bottles since before DAYS_FOR_RESULT */
  public boolean isPositiveOnDay(int day) {
```