The Puzzles of MrHeer

MrHeer

August 22, 2019

Contents

1 Throwing eggs from a building **1**

2 Throwing two eggs from a building **2**

3 3-collinearity **3**

4 Queue with three stacks [Unsolved] **4**

5 Queue with two stacks **5**

6 Deque with a stack and a steque [Unsolved] **7**

7 Deque with three stacks [Unsolved] **8**

8 Hot or cold **9**

9 3-sum for random values [Unsolved] **12**

10 Birthday problem [Unsolved] **13**

11 Coupon collector problem [Unsolved] **15**

1 Throwing eggs from a building

Question

Suppose that you have an N-story building and plenty of eggs. Suppose also that an egg is broken if it is thrown off floor F or higher, and unhurt otherwise. First, devise a strategy to determine the value of F such that the number of broken eggs is $\sim\lg N$ when using $\sim\lg N$ throws, then find a way to reduce the cost to $\sim 2\lg F$.

Answer

$\sim\lg N$: start at the middle, always cut search space in half $\rightarrow$ binary search.
$\sim 2\lg F$: start at 1, next 2, 4, 8 (i.e., 2^i), once the egg breaks after ($\sim\lg F$ steps) do binary search in the smaller search space (range $< F$ and hence number of searches $< \sim\lg F$) $\rightarrow$ exponential search.

$$2^{\lceil \lg F \rceil - 1} < F \leqslant 2^{\lceil \lg F \rceil}$$

$$range = 2^{\lceil \lg F \rceil} - 2^{\lceil \lg F \rceil - 1} = 2^{\lceil \lg F \rceil - 1} < 2^{\lg F} = F$$

$$range < F$$

2 Throwing two eggs from a building

Question

Consider the previous question, but now suppose you only have two eggs, and your cost model is the number of throws. Devise a strategy to determine F such that the number of throws is at most $2\sqrt{N}$, then find a way to reduce the cost to $\sim c\sqrt{F}$. This is analogous to a situation where search hits (egg intact) are much cheaper than misses (egg broken).

Answer

Let us make our first attempt on k'th floor. If it breaks, we try remaining $(k-1)$ floors one by one. So in worst case, we make k trials. If it doesn't break, we jump $(k-1)$ floors (Because we have already made one attempt and we don't want to go beyond k attempts. Therefore $(k-1)$ attempts are available), Next floor we try is floor $k+(k-1)$ Similarly, if this drop does not break, next need to jump up to floor $k+(k-1)+(k-2)$, then $k+(k-1)+(k-2)+(k-3)$ and so on. Since the last floor to be tried is F'th floor, sum of series should be F for optimal value of k.

$$k+(k-1)+(k-2)+(k-3)+\cdots+1 \geqslant F$$

$$\frac{k(k+1)}{2} \geqslant F$$

$$k \geqslant \frac{\sqrt{8F+1}-1}{2}$$

$$k_{min} = \lceil \frac{\sqrt{8F+1}-1}{2} \rceil \sim \sqrt{2F}$$

Official solution

Solution to Part 1: To achieve $2\sqrt{N}$, drop eggs at floors $\sqrt{N}$, $2*\sqrt{N}$, $3*\sqrt{N}$, ..., $\sqrt{N}\sqrt{N}$. (For simplicity, we assume here that $\sqrt{N}$ is an integer.) Let assume that the egg broke at level $k\sqrt{N}$. With the second egg you should then perform a linear search in the interval $(k-1)\sqrt{N}$ to $k\sqrt{N}$. In total you will be able to find the floor F in at most $2\sqrt{N}$ trials.
Hint for Part 2: $1+2+3+\cdots+k \sim \frac{1}{2}k^2 \geqslant F$.

3 3-collinearity

Question

Suppose that you have an algorithm that takes as input N distinct points in the plane and can return the number of triples that fall on the same line. Show that you can use this algorithm to solve the 3-sum problem. Strong hint: Use algebra to show that (a, a^3), (b, b^3), and (c, c^3) are collinear if and only if $a + b + c = 0$.

Answer

Proof. (a, a^3), (b, b^3), and (c, c^3) are collinear if and only if $a + b + c = 0$: We use a formulation of collinearity which equates gradients (assuming our points are distinct)

$$\frac{y_2 - y_1}{x_2 - x_1} = \frac{y_3 - y_1}{x_3 - x_1}$$

This becomes

$$\frac{b^3 - a^3}{b - a} = \frac{c^3 - a^3}{c - a}$$

which leaves us with

$$b^2 + ab + a^2 = c^2 + ac + a^2$$

so that

$$b^2 - c^2 = a(c - b)$$

$c \neq b$ so we have $a = -(b + c)$

$$a + b + c = 0$$

□

4 Queue with three stacks [Unsolved]

Question

Implement a queue with three stacks so that each queue operation takes a constant (worst-case) number of stack operations.

Answer

5 Queue with two stacks

Question

Implement a queue with two stacks so that each queue operation takes a constant amortized number of stack operations. Hint: If you push elements onto a stack and then pop them all, they appear in reverse order. If you repeat this process, they're now back in order.

Answer

Transfer

```
Result: stack1 to stack2 and reverse order
while stack1 is not empty do
    Item tmp = stack1.pop();
    stack2.push(tmp);
end
```

Enqueue

```
Input: element item
Result: enqueue
stack1.push(item);
```

Dequeue

```
Result: dequeue
if queue is empty then
    throw NoSuchElementException;
end
if stack2 is empty then
    transfer();
end
return stack2.pop();
```

Code

```
public class QueueWithTwoStacks<Item> {
    private Stack<Item> stack1;
    private Stack<Item> stack2;

    public QueueWithTwoStacks() {
        stack1 = new Stack<>();
        stack2 = new Stack<>();
    }

    private void transfer() {
        while (!stack1.isEmpty()) {
            Item tmp = stack1.pop();
            stack2.push(tmp);
        }
    }

    public void enqueue(Item item) {
        stack1.push(item);
    }

    public Item dequeue() {
        if (isEmpty()) throw new NoSuchElementException("Queue underflow");
        if (stack2.isEmpty()) transfer();
        return stack2.pop();
    }
}
```

Analysis

For N items enqueue and dequeue.
enqueue: N times push operations.
dequeue:
    transfer: N times pop and N times push operations.
    dequeue: N times pop operations.
total: $4N$ times stack operations.
So, the amortized number of stack operations is $\frac{4N}{N} = 4$

6 Deque with a stack and a steque [Unsolved]

Question

Implement a deque with a stack and a steque so that each deque operation takes a constant amortized number of stack and steque operations.

Answer

7 Deque with three stacks [Unsolved]

Question

Implement a deque with three stacks so that each deque operation takes a constant amortized number of stack operations.

Answer

8 Hot or cold

Question

Your goal is to guess a secret integer between 1 and N. You repeatedly guess integers between 1 and N. After each guess you learn if your guess equals the secret integer (and the game stops). Otherwise, you learn if the guess is hotter (closer to) or colder (farther from) the secret number than your previous guess. Design an algorithm that finds the secret number in at most $\sim 2 \lg N$ guesses. Then design an algorithm that finds the secret number in at most $\sim 1 \lg N$ guesses.
Hint: use binary search for the first part. For the second part, first design an algorithm that solves the problem in $\sim 1 \lg N$ guesses assuming you are permitted to guess integers in the range $-N$ to $2N$.

Answer

$\sim 2 \lg N$: Suppose the secret between $[a, b]$, $mid = \frac{a+b}{2}$. First geuess a, then geuss b, if b hotter a, geuess $[mid, b]$, else guess $[a, mid]$...
$\sim 1 \lg N$: Suppose you know that your secret integer is in $[a, b]$, and that your last guess is c. You want to divide your interval by two, and to know whether your secret integer lies in between $[a, m]$ or $[m, b]$, with $m = \frac{a+b}{2}$. The trick is to guess d, such that $\frac{c+d}{2} = \frac{a+b}{2}$.

Proof. guess integers in the range $-N$ to $2N$:
Suppose you know that your secret integer is in $[a_i, b_i]$, $b_i - a_i = d_i$, your last guess is g_{i-1}, this guess is g_i, and $\frac{g_i+g_{i-1}}{2} = \frac{a_i+b_i}{2} = m_i$.

$$g_i + g_{i-1} = 2m_i \tag{1}$$

$$g_{i-1} + g_{i-2} = 2m_{i-1} \tag{2}$$

(1) − (2)

$$g_i - g_{i-2} = 2(m_i - m_{i-1})$$

We know that $\mid m_i - m_{i-1} \mid = \frac{d_i}{2}$

$$\mid g_i - g_{i-2} \mid = d_i$$

Because divide interval by two everytime

$$d_i = \frac{1}{2} d_{i-1} = \frac{1}{2^{i-1}} d_1$$

We want to know the g_{max}, so $g_i > g_{i-2}$. When i is even

$$g_2 - g_0 = \frac{1}{2}d_1 \tag{1}$$

$$g_4 - g_2 = \frac{1}{8}d_1 \tag{2}$$

$$\cdots$$

$$g_i - g_{i-2} = \frac{1}{2^{i-1}}d_1 \tag{$\frac{i}{2}$}$$

$(1) + (2) + \cdots + (\frac{i}{2})$

$$\begin{aligned} g_i - g_0 &= (\frac{1}{2} + \frac{1}{8} + \cdots + \frac{1}{2^{i-1}})d_1 \\ &= \frac{2}{3}(1 - \frac{1}{2^i})(N-1) \end{aligned}$$

$$g_i = \frac{2}{3}(1 - \frac{1}{2^i})(N-1) + g_0$$

When $i \to \infty$ and $g_0 = N$

$$\begin{aligned} g_{max} &= \lim_{i\to\infty} \frac{2}{3}(1 - \frac{1}{2^i})(N-1) + N \\ &= \frac{2}{3}(N-1) + N \end{aligned}$$

$$g_{max} < 2N$$

When i is odd

$$g_i = \frac{1}{3}(1 - \frac{1}{2^{i-1}})(N-1) + g_1$$

When $i \to \infty$ and $g_1 = N$

$$\begin{aligned} g_{max} &= \lim_{i\to\infty} \frac{1}{3}(1 - \frac{1}{2^{i-1}})(N-1) + N \\ &= \frac{1}{3}(N-1) + N \end{aligned}$$

$$g_{max} < 2N$$

Above all

$$g_i < 2N$$

The same reason, $g_i < g_{i-2}$

$$g_i = \begin{cases} g_0 - \frac{2}{3}(1 - \frac{1}{2^i})(N-1) & \text{if } i \text{ is even,} \\ g_1 - \frac{1}{3}(1 - \frac{1}{2^{i-1}})(N-1) & \text{if } i \text{ is odd.} \end{cases}$$

When $i \to \infty$ and $g_0 = g_1 = 1$

$$g_{min} = \begin{cases} 1 - \frac{2}{3}(N-1) & \text{if } i \text{ is even,} \\ 1 - \frac{1}{3}(N-1) & \text{if } i \text{ is odd.} \end{cases}$$

$$g_i > -N$$

So, guess integers in the range $-N$ to $2N$.

□

9 3-sum for random values [Unsolved]

Question

Formulate and validate a hypothesis describing the number of triples of N random int values that sum to 0. If you are skilled in mathematical analysis, develop an appropriate mathematical model for this problem, where the values are uniformly distributed between $-M$ and M, where M is not small.

Answer

10 Birthday problem [Unsolved]

Question

Write a program that takes an integer N from the command line and uses `StdRandom.uniform()` to generate a random sequence of integers between 0 and $N-1$. Run experiments to validate the hypothesis that the number of integers generated before the first repeated value is found is $\sim\sqrt{\frac{N}{2}\pi}$.

Answer

We can use `BitSet.get(next)` to determine if *next* appears.

Code

```
public class BirthdayProblem {
    private static int findFirstRepeat(int N) {
        BitSet bitSet = new BitSet(N);
        int next = StdRandom.uniform(N);
        int count = 0;
        while (!bitSet.get(next)) {
            bitSet.set(next);
            next = StdRandom.uniform(N);
            count++;
        }
        return count;
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int count;
        int countTest = 0;
        double avg = 0;
        while (true) {
            count = findFirstRepeat(N);
            countTest++;
            avg = avg + (count - avg) / countTest;
            StdOut.println(avg);
```

```
        }
    }
}
```

Proof. the number of integers generated before the first repeated value is found is $\sim\sqrt{\frac{N}{2}\pi}$. □

11 Coupon collector problem [Unsolved]

Question

Generating random integers as in the previous exercise, run experiments to validate the hypothesis that the number of integers generated before all possible values are generated is $\sim NH_N$.

Answer

Same to previous exercise, we can ues `BitSet` to validate the hypothesis.

Code

```
public class CouponCollectorProblem {
    private static int findAllRepeat(int N) {
        BitSet bitSet = new BitSet(N);
        BitSet fullBitSet = new BitSet(N);
        fullBitSet.flip(0, N);
        int next = StdRandom.uniform(N);
        int count = 0;
        while (!bitSet.equals(fullBitSet)) {
            bitSet.set(next);
            next = StdRandom.uniform(N);
            count++;
        }
        return count;
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int count;
        int countTest = 0;
        double avg = 0;
        while (true) {
            count = findAllRepeat(N);
            countTest++;
            avg = avg + (count - avg) / countTest;
```

```
            StdOut.println(avg);
        }
    }
}
```

Proof. the number of integers generated before all possible values are generated is $\sim NH_N$. □