

-
1. (a) *Use the subroutine `probefluxcapacitor()` to implement the routine `randbit()` that outputs a uniformly random bit.*

Pseudocode

```
def randbit():  
    while true:  
        (bitA, p) = probefluxcapacitor()  
        (bitB, p) = probefluxcapacitor()  
        if bitA does not equal bitB:  
            return bitA
```

Correctness

If the bits were uniformly random then a 1 or a 0 would be equally probable. This would mean that the probability that either one would occur would need to be $\frac{1}{2}$ in order to be uniformly random.

For the problem we are given that $P(1) = p$ and $P(0) = (1 - p)$. The first thing we have to figure out is how do we make an algorithm such that the probability of returning a 1 is equal to the probability of returning a 0. The second thing we have to figure out is how do we make sure the final probability for any run of `randbit()` is equal $\frac{1}{2}$.

Notice the probability of two independent events, A and B , occurring together is $P(A \cap B) = P(A) \cdot P(B)$ [1]. If we let those two events be the probability of `randbit()` returning a 1, $P(1)$, and the probability of `randbit()` returning a 0, $P(0)$, then we can say that the probability of them both occurring, $P(1 \cap 0)$, is $p \cdot (1 - p)$. This helps because we know that one call to `probefluxcapacitor()` will either return p , for bit value 1, or $(1 - p)$, for bit value 0. But if we combine two

calls to `probefluxcapacitor()` sequentially and then compare the results we can create a probability like $P(1 \cap 0)$.

Two calls to `probefluxcapacitor()` could result in 4 four combinations:

- $P(1 \cap 1) = p \cdot p$
- $P(0 \cap 0) = (1 - p) \cdot (1 - p)$
- $P(1 \cap 0) = p \cdot (1 - p)$
- $P(0 \cap 1) = (1 - p) \cdot p$

The pseudocode above only returns a bit if the probability combination was $p \cdot (1 - p)$ or $(1 - p) \cdot p$ which are in fact equal to each other. This tells us that returning a 1 — bitA is 1 and bitB is 0 — and returning a 0 — bitA is 0 and bitB is 1 — have equal probability now. And since the if statement only looks at two possible combinations from the two calls to `probefluxcapacitor()` and out of only one of those possible combinations will a 1 be returned, the probability of returning a 1, $P(1)$, is $\frac{1}{2}$.

Runtime

In order to find the running time of a randomize algorithm, we need to look at the expectation of the running time. The two calls to `probefluxcapacitor()` each take $O(1)$ because they are just looking at at most two numbers. The checking of the two bits in the if statement is also $O(1)$. This means that one time through the loop takes $O(1)$. The probability of getting out of the while loop and returning is $p \cdot (1 - p)$ for a 1 or $(1 - p) \cdot p$ for a 0.

Let X be the indicator random variable associated with the returning of `rand-bit()`. X_1 means a 1 is returned and X_0 means a 0 is returned. This also means $E[X_1] = P(1)$, the probability of 1 being returned, and $E[X_0] = P(0)$, the prob-

ability of 0 being returned. This gives us

$$\begin{aligned} E[X] &= \sum_{i=0}^1 E[X_i] \\ &= E[X_0] + E[X_1] \\ &= p \cdot (1 - p) + (1 - p) \cdot p \\ &= 2p(1 - p) \end{aligned}$$

This means the running time of `randbit()` is $O(2p(1 - p) \cdot 1) = O(2p(1 - p))$.

- (b) *Implement the algorithm $\text{randbit}(p)$ that outputs an independent random bit with $P(\text{randbit}(p)=1)=p$.*

Pseudocode

Correctness

Runtime

References

- [1] <https://www.mathsisfun.com/data/probability-events-independent.html>