Randomized Algorithms and Probabilistic Methods

Solution to Exercise 1

Surprisingly it's not possible to win with a higher probability than \( \frac{1}{2} \). Let's assume that \( i \) cards have been revealed, if we choose to end the game on the \( i+1 \)st card we could just as well reveal the bottom card since the rest of the deck is in random order. That is for every strategy you cannot do better than revealing the bottom card.

Solution to Exercise 2

Let \( A \) denote the Las Vegas algorithm given, and consider the following algorithm:

\[
\begin{align*}
1 & \text{ do} \\
2 & \text{ solution } = A(I) \\
3 & \text{ while ( solution == '?' )}
\end{align*}
\]

That is, the algorithm \( A \) is repeatedly called until it gives an answer different from '?''. This answer is then guaranteed to be correct, cf. the definition of a Las Vegas algorithm.

In each iteration the success probability of the call to \( A \) is \( p \), and the time needed for each call is \( t(|I|) \).

Let \( X \) denote the number of iterations. Obviously \( X \) has a geometric distribution:

\[
\Pr[X = k] = p \cdot (1 - p)^{k-1}
\]

and therefore \( \mathbb{E}[X] = \frac{1}{p-1} \).

Thus the expected running time is \( \frac{1}{p-1} \cdot t(|I|) \).

Solution to Exercise 3

For every \( n \geq 0 \) let \( X_n \) denote the random variable that counts the number of blue balls in the bag after \( n \) steps. Calculating with a couple of small values of \( n \) leads to the following claim.

**Claim 1** For every \( n \geq 0 \) and \( 1 \leq k \leq n+1 \) we have

\[
\Pr[X_n = k] = \frac{1}{n+1}.
\]

**Proof:** We apply induction on \( n \). For \( n = 0 \) the claim is true since \( \Pr[X_0 = 1] = 1 \).

Now, let \( n \geq 1 \) and \( 1 \leq k \leq n+1 \). By induction hypothesis (short: I.H.) we know that for every \( 1 \leq k \leq n \) we have \( \Pr[X_{n-1} = k] = \frac{1}{n} \). Clearly, we can only have \( k \) blue balls in the bag after \( n \) steps if there were either \( k \) blue balls in the bag after \( n-1 \) steps and we drew a red ball, or \( k-1 \) blue balls in the bag after \( n-1 \) steps and we drew a blue ball. Hence, we have

\[
\begin{align*}
\Pr[X_n = k] &= \Pr[X_{n-1} = k] \Pr[\text{Draw red in step } n | X_{n-1} = k] \\
&\quad + \Pr[X_{n-1} = k-1] \Pr[\text{Draw blue in step } n | X_{n-1} = k-1] \\
&\overset{\text{I.H.}}{=} \frac{1}{n} \cdot \frac{n+1 - k}{n+1} + \frac{1}{n} \cdot \frac{k-1}{n+1} \\
&= \frac{1}{n+1} \left( \frac{n+1 - k}{n} + \frac{k-1}{n} \right) = \frac{1}{n+1}.
\end{align*}
\]

\[\blacksquare\]
Solution to Exercise 4

Let $Y$ denote the random variable that counts the number of dice we roll until we see a six. Further, let $X$ denote the random variable that counts the number of dice we roll until we see two consecutive sixes. Furthermore, let $X_2$ denote the random variable that, starting after the first six and one additional throw (which are all ignored), counts the number of dice we roll until we see two consecutive sixes. E.g., for the outcome $\omega = (1, 6, 3, 5, 6, 2, 4, \ldots)$ we have $X(\omega) = 6$ and $X_2(\omega) = 3$ (we start to count at the 5).

Observe that both random variables $X$ and $X_2$ have the same distribution (i.e. $\Pr[X = i] = \Pr[X_2 = i]$ for all $i \geq 0$, and therefore also $\mathbb{E}[X] = \mathbb{E}[X_2]$), although they will usually have different values for a particular experiment. In this exercise we are interested in $\mathbb{E}[X]$.

Let $E$ denote the event that the die we roll after we see the first six is also a six. Clearly, we have $\Pr[E] = \frac{1}{6}$.

We also know that if $E$ occurs then $X$ equals $Y + 1$, and that otherwise $X$ equals $Y + 1 + X_2$. Hence we have by the law of total expectation that

$$
\mathbb{E}[X] = \Pr[E] \cdot \mathbb{E}[X \mid E] + \Pr[\overline{E}] \cdot \mathbb{E}[X \mid \overline{E}]
$$

$$
\leq \frac{1}{6} \cdot \mathbb{E}[Y + 1] + \frac{5}{6} \cdot \mathbb{E}[Y + 1 + X_2]
$$

$$
\overset{L.o.E.}{=} \mathbb{E}[Y] + 1 + \frac{5}{6} \mathbb{E}[X_2],
$$

where we used that $\mathbb{E}[X] = \mathbb{E}[X_2]$. Since $Y \sim Geom\left(\frac{1}{6}\right)$ we have $\mathbb{E}[Y] = 6$ and obtain

$$
\frac{1}{6} \mathbb{E}[X] = 7,
$$

which gives us $\mathbb{E}[X] = 42$.

Solution to Exercise 5

The idea is to assign the colors randomly and to show that for $s < r^{t-1}$ the probability that no $F_i$ is monochromatic is greater than 0.

Assign each color independently and uniformly to each element in $M$:

$$
\Pr[c(x) = j] = \frac{1}{r} \quad \text{for all } x \in M \text{ and } j \in \{1, \ldots, r\}.
$$

What is the probability that $F_i$ is monochromatic? There are $r$ possibilities to choose a color. For each choice all $|F_i| = t$ elements of $F_i$ have to receive this color. Hence

$$
\Pr[F_i \text{ monochromatic}] = r \cdot r^{-|F_i|} = r^{-t+1}.
$$

Let $E$ denote the event that no $F_i$ is monochromatic. That is,

$$
E = \bigcap_{i=1}^{s} \{F_i \text{ not monochromatic}\} \quad \text{and thus } \overline{E} = \bigcup_{i=1}^{s} \{F_i \text{ monochromatic}\}.
$$

Observe that due to $\Pr[E] = 1 - \Pr[\overline{E}]$, in order to prove $\Pr[E] > 0$ it suffices to show $\Pr[\overline{E}] < 1$. By using the union bound we obtain

$$
\Pr[\overline{E}] = \Pr[\bigcup_{i=1}^{s} F_i \text{ monochromatic}] \leq \sum_{i=1}^{s} \Pr[F_i \text{ monochromatic}] = s \cdot r^{-t+1}.
$$

Due to our assumption $s < r^{t-1}$ we obtain

$$
\Pr[\overline{E}] \leq s \cdot r^{-t+1} < 1 \quad \text{and thus } \Pr[E] = 1 - \Pr[\overline{E}] > 0.
$$
Solution to Exercise 6

The idea in this exercise is, for some input \( x \in \{0, \ldots, n-1\} \), to not lookup \( f(x) \) directly, but instead choose a random value \( y \in \{0, \ldots, n-1\} \), set \( z := x - y \mod n \) and then lookup \( f(y) \) and \( f(z) \). If both these lookups are not corrupted, we can calculate \( f(x) \) from these values since \( f(x) = f((y + z) \mod n) = (f(y) + f(z)) \mod m \). Moreover, we will see that we can boost the success probability to compute \( f(x) \) correctly by repeating this procedure.

The above idea is formalized in Algorithm 1.1, where ‘Majority’ returns the value that appears most often in the list of results of all loop iterations (if this value is not unique we pick a random one).

\[ \text{Algorithm 1.1 Evaluate from corrupted table} \]

<table>
<thead>
<tr>
<th>Input:</th>
<th>( x \in {0, \ldots, n-1} ), number ( k \in \mathbb{N} ) of repetitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>candidate for ( f(x) ).</td>
</tr>
</tbody>
</table>

\[ \text{forall } 1 \leq i \leq k \text{ do} \]

\[ \text{Choose } y \in \{0, \ldots, n-1\} \text{ uniformly at random.} \]

\[ \text{Set } z := x - y \mod n. \]

\[ r[i] := (f(y) + f(z)) \mod m \quad // \text{2 lookups} \]

\[ \text{return Majority}(r[1], \ldots, r[k]). \]

We first analyze the algorithm for \( k = 1 \). In this case the algorithm uses 2 lookups, one for \( f(y) \) and one for \( f(z) \), and returns \((f(y) + f(z)) \mod m \). We know that each of these lookups is corrupted with probability at most \( 1/5 \), i.e.,

\[ \Pr[f(y) \text{ is corrupted}] \leq \frac{1}{5}, \quad \text{and} \quad \Pr[f(z) \text{ is corrupted}] \leq \frac{1}{5}. \quad (1) \]

Altogether, we obtain

\[ \Pr[\text{Algorithm 1.1 returns } f(x)] = \Pr[f(y) \text{ is not corrupted} \land f(z) \text{ is not corrupted}] \]

\[ = 1 - \Pr[f(y) \text{ is corrupted} \lor f(z) \text{ is corrupted}] \]

\[ \geq 1 - \Pr[f(y) \text{ is corrupted}] \land \Pr[f(z) \text{ is corrupted}] \]

\[ \geq 1 - 2 \cdot \frac{1}{5} = \frac{3}{5}. \]

We thus found a randomized algorithm which uses only 2 lookups and, regardless of what table entries the evil adversary changed, returns for every input \( x \in \{0, \ldots, n-1\} \) the value \( f(x) \) with probability at least \( 1/2 \).

Now what happens for larger values of \( k \)? (The remainder essentially follows the proof of Lemma 1.4 in the lecture notes.) We know that a single iteration of the loop calculates \( f(x) \) correctly with probability at least \( p := 3/5 \). Now since the runs are independent, and since the algorithm can only return an incorrect value if at most \( \lfloor (k - 1)/2 \rfloor \) runs yield the correct value for \( f(x) \), we obtain

\[ \Pr[\text{Algorithm 1.1 does not return } f(x)] \leq \Pr[\text{At most } \lfloor \frac{k-1}{2} \rfloor \text{ loop iterations yield the correct answer}] \]

\[ \leq \sum_{i=0}^{\lfloor \frac{k-1}{2} \rfloor} \binom{k}{i} p^i(1-p)^{k-i} \]

\[ = (1-p)^k \cdot \sum_{i=0}^{\lfloor \frac{k-1}{2} \rfloor} \binom{k}{i} \left( \frac{p}{1-p} \right)^i \]

\[ p > \frac{1}{2} \]

\[ \leq (1-p)^k \cdot \left( \frac{p}{1-p} \right)^{\frac{k}{2}} \cdot \sum_{i=0}^{\lfloor \frac{k-1}{2} \rfloor} \binom{k}{i} \]

\[ \leq (1-p)^{\frac{k}{2}} \cdot p^{\frac{k}{2}} \cdot 2^k. \]
By writing $p = \frac{1}{2} + \frac{1}{10}$ and $1 - p = \frac{1}{2} - \frac{1}{10}$ we get

$$(1 - p)^\frac{k}{2} \cdot p^\frac{k}{2} \cdot 2^k = \left(\frac{1}{4} - \frac{1}{100}\right)^{\frac{k}{2}} \cdot 2^k = \left(1 - \frac{1}{25}\right)^{\frac{k}{2}},$$

and thus

$$\Pr[\text{Algorithm 1.1 returns } f(x)] \geq 1 - \left(1 - \frac{1}{25}\right)^{\frac{k}{2}},$$

which tends to 1 if $k$ goes to infinity.

**Remark (cf. Lemma 1.4 in the lecture notes):** Note that this calculation works for any algorithm that returns the correct answer with probability at least $p = 1/2 + \varepsilon$ for some $\varepsilon > 0$. It then results in an error probability of $(1 - 4\varepsilon^2)^{\frac{k}{2}}$, which means that for any $p^* < 1$, a constant number of repetitions suffices to achieve success probability $p^*$. This is also the reason why it suffices to require a probability of $1/2 + \varepsilon$ in the definition of Monte Carlo algorithms.