CS360 Assignment 5

5.2-4 **Hat check**. Each of n customers leaves a hat at a hat-check and is then returned a random one when they leave. What is the expected number of customers that get the correct hat back?

Define an indicator random variable X_i to be the event that a customer receives their own hat back.
Then $X = X_1 + X_2 + ... + X_n = \sum_{i=1}^{n} X_i$ is the total number of people that get their own hat back and hence the expected value is given by
$E[X] = E[X_1 + X_2 + ... + X_n] = \sum_{i=1}^{n} E[X_i]$

Since each person is given a random hat
$E[X_i] = \Pr\{X_i\} = 1/n$ (by a similar argument as the probability that the $i^t h$ number of a sequence of n values is a particular value)
$\Rightarrow E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{1}{n} = 1$
Therefore we can expect 1 customer to get their own hat (regardless of the number of people).
Note: No where do we assume independent events (which certainly is *not* the case) but only use the fact that expected value is a linear operator.

5.4-2 Suppose that we toss balls into b bins until some bin contains *two* balls. Each toss is independent, and each ball is equally likely to end up in any bin. What is the expected number of ball tosses?

This problem is identical to the *Birthday Paradox* if we observe that tossing the balls into bins is equivalent to having people with birthdays (where b corresponds to the number of days in a year). Then the problem of having two balls in the same bin is equivalent to asking how many people do we need to have two with the same birthday. Hence the analysis follows the same rational.

Let

$$X_{ij} = I\{\text{ball } i \text{ and ball } j \text{ land in same bin}\}$$

Then for k tosses

$$X = \sum_{i=1}^{k} \sum_{j=i+1}^{k} X_{ij}$$

Taking expected values and recognizing that $E[X_{ij}] = 1/b$ (since for a given slot for ball i, ball j has a $1/b$ chance of landing in the same slot)

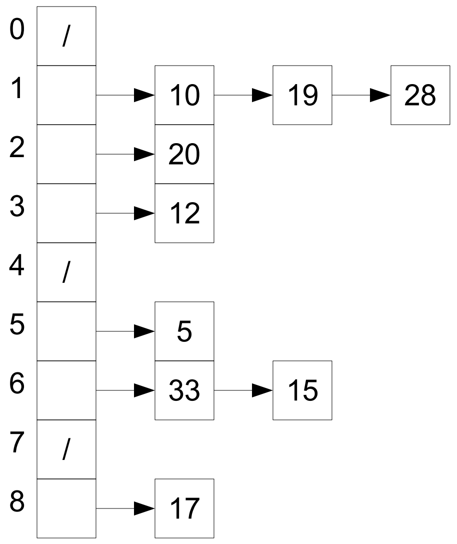
$$\begin{aligned} E[X] &= E\left[\sum_{i=1}^{k}\sum_{j=i+1}^{k} X_{ij}\right] \\ &= \sum_{i=1}^{k}\sum_{j=i+1}^{k} E[X_{ij}] \\ &= \sum_{i=1}^{k}\sum_{j=i+1}^{k} (1/b) \\ &= \frac{k(k-1)}{2b} \end{aligned}$$

Thus setting $E[X] = 1$ gives $k = \frac{\sqrt{8b+1}+1}{2}$.
So for example, if we have 10 bins (i.e. $b = 10$), then for $k = \frac{\sqrt{81}+1}{2} = 5$ ball tosses we would expect that one bin would end up with two balls.

11.2-2 Demonstrate the insertion of keys $< 5, 28, 19, 15, 20, 33, 12, 17, 10 >$ into a hash table using chaining. Let $m = 9$ and $h(k) = k \bmod 9$ (i.e. division method).

$$\begin{aligned} h(5) = 5 \bmod 9 &= 5 \\ h(28) = 28 \bmod 9 &= 1 \\ h(19) = 19 \bmod 9 &= 1 \\ h(15) = 15 \bmod 9 &= 6 \\ h(20) = 20 \bmod 9 &= 2 \\ h(33) = 33 \bmod 9 &= 6 \\ h(12) = 12 \bmod 9 &= 3 \\ h(17) = 17 \bmod 9 &= 8 \\ h(10) = 10 \bmod 9 &= 1 \end{aligned}$$

Hence the hash table will look like



11.3-4 Consider a hash table with $m = 1000$ and $h(k) = \lfloor m(kA \bmod 1) \rfloor$ for $A = \frac{\sqrt{5}-1}{2} \approx 0.618$. Compute the hash values for $< 61, 62, 63, 64, 65 >$

$$\begin{aligned}
h(61) = \lfloor 1000((61)(0.618) \bmod 1) \rfloor = \lfloor 1000(.700) \rfloor &= 700 \\
h(62) = \lfloor 1000((62)(0.618) \bmod 1) \rfloor = \lfloor 1000(.316) \rfloor &= 316 \\
h(63) = \lfloor 1000((63)(0.618) \bmod 1) \rfloor = \lfloor 1000(.934) \rfloor &= 934 \\
h(64) = \lfloor 1000((64)(0.618) \bmod 1) \rfloor = \lfloor 1000(.552) \rfloor &= 552 \\
h(65) = \lfloor 1000((65)(0.618) \bmod 1) \rfloor = \lfloor 1000(.170) \rfloor &= 170
\end{aligned}$$

Your numbers may be different depending on your exact value of A. Note that even though the initial keys are sequential, the corresponding hash slots are well distributed.

15.1-3 Rod Cutting with per cut cost.

We first note that the only place where a cut enters the calculations is in line 6, thus we simply subtract the per cut cost to the profit in the second term of the max() function. However, we also need to stop the loop early since we do not want to incur the cut cost in the case where the entire rod is taken, i.e. no cut is made. Since the solution is generated bottom-up, future subproblems will automatically incorporate the prior cut cost. Hence the new routine would be

```
NEW_CUT_ROD_WITH_CUT_COST(p,n)
1  let r[0..n] be a new array
2  r[0] = 0
3  for j = 1 to n
4     q = -INF
5     for i = 1 to j-1
6        q = max(q, p[i] + r[j-i] - c)
7     r[j] = max(q,p[j])                 // in case there are no cuts
8  return r[n]
```