Assignment 1

Computer Science 531

Due: 16:00, Friday October 9, 2015

Instructor: Majid Ghaderi

This assignment has a problem solving part (Questions 1-4) and a programming part (Questions 5-6):

- For the problem solving part, either type or neatly handwrite your answers, then submit it using the designated assignment *drop box* in the Math Sciences building.
- For the programming part, submit your program via D2L following the submission instructions described there. The submission deadline for the programming part is 23:55, Friday October 9, 2015 (same day, but midnight).

QUESTION 1 (10 Marks)

Show that the geometric distribution is memoryless.

QUESTION 2 (10 Marks)

Consider a random variable X with the following PMF,

$$p(x) = \frac{1}{n+1}, \qquad \text{for } x = 0, 1, 2, \ldots, n\,.$$

Compute the mean and variance of X.

QUESTION 3 (10 Marks)

Lead time for an item is approximated by a normal distribution with a mean of 20 days and a variance of 4 days^2. Find values of lead time that will be exceeded only 1% and 10% of the time.

QUESTION 4 (20 Marks)

In an industrial facility, three shafts are made and assembled into a linkage. The length of each shaft, in centimeters, is distributed as follows:

- Shaft 1: $N(60, 0.09)$
- Shaft 2: $N(40, 0.05)$
- Shaft 3: $N(50, 0.11)$

where $N(x, y)$ denotes the normal distribution with mean x and variance y.

(a) What is the distribution of the length of the linkage?

(b) What is the probability that the linkage will be longer than 150.2 centimeters?

(c) The tolerance limits for the assembly are $(149.83, 150.21)$. What proportions of assemblies are within the tolerance limits?

The Java class `MCRandom` is provided to you for generating uniformly distributed random numbers. You have to exclusively use this class for random number generation in the following questions. Read the Javadoc documentation provided in the source code on how to use this class in your code.

QUESTION 5 (25 Marks)

A factory makes dice whose six sides have the following PMF:

x	1	2	3	4	5	6
$p(x)$	$\frac{1}{10}$	$\frac{0}{10}$	$\frac{1}{10}$	$\frac{2}{10}$	$\frac{4}{10}$	$\frac{2}{10}$

Consider the random experiment of rolling two such dice. Let X_1 and X_2 denote, respectively, the first and second number rolled. We are interested in computing the probability $\mathbb{P}\{X_1 \leq X_2\}$ using the Monte Carlo simulation technique. You should write a Java class named `MCDice` to simulate this experiment n times, and then use the outcome of the experiments to compute the required probability. Your class should implement the following method (with the exact signature):

```
public double solve(int n)
```

where, the parameter `n` is the number of times you need to repeat the experiment. The return value of this method is the desired probability $\mathbb{P}\{X_1 \leq X_2\}$.

To simulate the random experiment of rolling a die, you may use the following algorithm.

Algorithm 1 Generating a random roll of a die.

1: Let F denote the CDF of the die based on PMF $p(x)$
2: Generate a uniformly distributed random number u between 0 and 1
3: If $F(j-1) < u \leq F(j)$, then the roll number is j

QUESTION 6 (25 Marks)

In this question, you are asked to write a program to solve the Knapsack problem using the Monte Carlo technique. The Knapsack problem is defined as follows. Consider a knapsack of fixed weight capacity w. Given a set of n items numbered from 1 to n, each with a weight w_i

and value v_i, determine which items to put in the knapsack so that the total weight is less than or equal to W and the total value is as large as possible (see Wikipedia for more information on the Knapsack problem: `https://en.wikipedia.org/wiki/Knapsack_problem`).

The pseudo-code presented in Algorithm 2 describes an algorithm that can be used to solve the Knapsack problem. In this algorithm, $S = (s_1, s_2, \ldots, s_n)$ denotes the state of the knapsack (*i.e.*, the system state), where, $s_i = 1$ if item i is in the knapsack, and $S_i = 0$, otherwise.

Initially, there is no item in the knapsack. In a loop that repeats for T times, one item is selected randomly and its state is changed. This results in a new state X. If state X is not feasible, then the state of the knapsack does not change (*i.e.*, S does not change). If X is feasible, then the state of the knapsack changes with probability α (thus, in this case, the state S does not change with probability $1 - \alpha$). The value of α is given by:

$$\alpha = \min\{1,\, e^{value(X) - value(S)}\}\,.$$

A state $X = (x_1, x_2, \ldots, x_n)$ is feasible if $\sum_{i=1}^{n} x_i w_i \leq W$. The value of a state $X = (x_1, x_2, \ldots, x_n)$ is given by $value(X) = \sum_{i=1}^{n} x_i v_i$.

Algorithm 2 Monte Carlo Knapsack.

1: Let the initial state $S = (0, \ldots, 0)$
2: **for** $t = 1$ **to** T **do**
3: Choose j uniformly at random from $\{1, 2, \ldots, n\}$
4: Let $X = (s_1, \ldots, 1 - s_j, \ldots, s_n)$
5: If X is feasible, then set $S = X$ with probability α
6: **end for**

Write a Java class named `MCKnapsack` to implement this algorithm. Your class should implement the following method (with the exact signature):

```
public int[] solve(int T, int W, int[] weights, int[] values)
```

where,

- `T`: number of iterations of the algorithm
- `W`: the capacity of the knapsack
- `weights`: the array of item weights
- `values`: the array of item values

The return value of this method is an array, where the i-th element of the array indicates whether item i is in the knapsack or not (using 1 and 0, respectively).