1. Modify SelfAvoidingWalk to calculate and print the average length of the paths as well as the dead-end probability. Keep separate the average lengths of escape paths and dead-end paths.
2. You are given a class `ProbDistribution` which can implement an arbitrary distribution given these intervals and cumulative distributions. Below is an example of its use.

```java
/* Each pair in constructor is a value and the cumulative percentage up to that point */
ProbDistribution interArrival = new ProbDistribution({
    {0.1,0.0},{1.0,8},{2.0,25},{3.0,50},{4.0,70},{5.0,85},
    {6,89},{7,93},{8,95},{9,97},{10,100}});

double x = interArrival.sample(); /* returns a double according to the specified probability distribution */
```

We’re all familiar with the normal Gaussian distribution, otherwise known as the “Bell curve.” Use the `ProbDistribution` to construct an approximation to the Gaussian distribution. The normal Gaussian distribution will have a mean, $\mu$, and standard deviation, $\sigma$. The probability density function of a normal random variable at some point $x$, is given by:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

and shown in Figure 1.

![Figure 1: Normal density function](image)

You can compute the cumulative percentage at $x$ by integrating $f(x)$. Integrating $f(x)$ can be approximated by summing the values at fixed intervals up to $x$

```java
for (double approxPoint = min_value, sum = 0; approxPoint <= x;
    approxPoint += interval)
    sum += f(approxPoint)*interval;
```

Your class `GaussianDist` will take a mean and a standard deviation in the constructor and can also implements the corresponding `sample()` method. Should this class be independent of, inherit from, or be a sibling of `ProbDistribution`?
3. Write a method `ranPairs` that takes as parameters a page count \( N \) and a link count \( M \) and returns an `ArrayList` of \( M \) random `Pairs` of integers from 0 to \( N-1 \).