Contents

1 Binomial Random Variables **4**
Source 10

2 Birthday Paradox **11**
Source 15

3 Estimating the value of Pi using Monte Carlo **16**
Source 19

4 Expectation or expected value of an array **20**
Source 25

5 Expected Number of Trials until Success **26**
Source 28

6 Find an index of maximum occurring element with equal probability **29**
Source 31

7 Freivald's Algorithm to check if a matrix is product of two **32**
Source 42

8 Generate 0 and 1 with 25% and 75% probability **43**
Source 48

9 Generate integer from 1 to 7 with equal probability **49**
Source 52

10 Implement rand12() using rand6() in one line **53**
Source 55

11 Implement rand3() using rand2() **56**
Source 59

12 Implement random-0-6-Generator using the given random-0-1-Generator **60**
Source 62

13 Karger's algorithm for Minimum Cut | Set 1 (Introduction and Implementation) **63**
Source 70

14 Karger's algorithm for Minimum Cut | Set 2 (Analysis and Applications) 71
Source 74

15 K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time) 75
Source 79

16 Linearity of Expectation 80
Source 82

17 Load Balancing on Servers (Randomized Algorithm) 83
Source 84

18 Make a fair coin from a biased coin 85
Source 86

19 Mid-Square hashing 87
Source 89

20 Operations on Sparse Matrices 90
Source 100

21 Primality Test | Set 2 (Fermat Method) 101
Source 107

22 Program to generate CAPTCHA and verify user 108
Source 110

23 QuickSort using Random Pivoting 111
Source 117

24 Random Walk (Implementation in Python) 118
Source 122

25 Random number generator in arbitrary probability distribution fashion 123
Source 126

26 Randomized Algorithms | Set 0 (Mathematical Background) 127
Source 130

27 Randomized Algorithms | Set 1 (Introduction and Analysis) 131
Source 133

28 Randomized Algorithms | Set 2 (Classification and Applications) 134
Source 135

29 Randomized Algorithms | Set 3 (1/2 Approximate Median) 136
Source 136

30 Randomized Binary Search Algorithm 137
Source 141

31 Reservoir Sampling **142**
Source 148

32 Select a Random Node from a Singly Linked List **149**
Source 155

33 Select a Random Node from a tree with equal probability **156**
Source 157

34 Select a random number from stream, with O(1) space **160**
Source 165

35 Shuffle a deck of cards **166**
Source 170

36 Shuffle a given array **171**
Source 176

37 Shuffle or Randomize a list in Java **177**
Source 180

38 Strong Password Suggester Program **181**
Source 193

39 Write a function that generates one of 3 numbers according to given probabilities **194**
Source 195

Chapter 1

Binomial Random Variables

Binomial Random Variables - GeeksforGeeks

In this post, we'll discuss Binomial Random Variables.

Prerequisite : Random Variables

A specific type of **discrete** random variable that counts how often a particular event occurs in a fixed number of tries or trials.

For a variable to be a binomial random variable, ALL of the following conditions must be met:

1. There are a fixed number of trials (a fixed sample size).
2. On each trial, the event of interest either occurs or does not.
3. The probability of occurrence (or not) is the same on each trial.
4. Trials are independent of one another.

Mathematical Notations

```
n = number of trials
p = probability of success in each trial
k = number of success in n trials
```

Now we try to find out probability of k success in n trials.

Here probability of success in each trial is p independent of other trials.
So we first choose k trials in which there will be success and in rest n-k trials there will be failure. Number of ways to do so is

[illegible]

Since all n events are independent, hence probability of k success in n trials is equivalent to multiplication of probability for each trial.

Here its k success and n-k failures, So probability for each way to achieve k success and n-k failure is
$p^k(1-p)^{n-k}$

Hence final probability is

```
(number of ways to achieve k success
 and n-k failures)
 *
(probability for each way to achieve k
 success and n-k failure)
```

Then Binomial Random Variable Probability is given by:

[illegible]

Let X be binomial random variable with number of trials n and probability of success in each trial be p.
Expected number of success is given by

```
E[X] = np
```

Variance of number of success is given by

```
Var[X] = np(1-p)
```

Example 1 : Consider a random experiment in which a biased coin (probability of head = 1/3) is thrown for 10 times. Find probabilit that number of heads appearing will be 5.

Solution :

```
Let X be binomial random variable
with n = 10 and p = 1/3
P(X=5) = ?
```

Here is a C++ code for the same

C++

```
 // C++ program to compute Binomial Probability
#include <iostream>
#include <cmath>
using namespace std;

// function to calculate nCr i.e., number of
// ways to choose r out of n objects
int nCr(int n, int r)
{
    // Since nCr is same as nC(n-r)
    // To decrease number of iterations
    if (r > n / 2)
        r = n - r;

    int answer = 1;
    for (int i = 1; i <= r; i++) {
        answer *= (n - r + i);
        answer /= i;
    }

    return answer;
}

// function to calculate binomial r.v. probability
float binomialProbability(int n, int k, float p)
{
    return nCr(n, k) * pow(p, k) *
                pow(1 - p, n - k);
}

// Driver code
int main()
{

    int n = 10;
    int k = 5;
    float p = 1.0 / 3;

    float probability = binomialProbability(n, k, p);

    cout << "Probability of " << k;
    cout << " heads when a coin is tossed " << n;
    cout << " times where probability of each head is " << p << endl;
    cout << " is = " << probability << endl;
}
```

Java

```
 // Java program to compute Binomial Probability

import java.util.*;

class GFG
{
    // function to calculate nCr i.e., number of
    // ways to choose r out of n objects
    static int nCr(int n, int r)
    {
        // Since nCr is same as nC(n-r)
        // To decrease number of iterations
        if (r > n / 2)
            r = n - r;

        int answer = 1;
        for (int i = 1; i <= r; i++) {
            answer *= (n - r + i);
            answer /= i;
        }

        return answer;
    }

    // function to calculate binomial r.v. probability
    static float binomialProbability(int n, int k, float p)
    {
        return nCr(n, k) * (float)Math.pow(p, k) *
                        (float)Math.pow(1 - p, n - k);
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 10;
        int k = 5;
        float p = (float)1.0 / 3;

        float probability = binomialProbability(n, k, p);

        System.out.print("Probability of " +k);
        System.out.print(" heads when a coin is tossed " +n);
        System.out.println(" times where probability of each head is " +p);
        System.out.println( " is = " + probability );
    }
}

/* This code is contributed by Mr. Somesh Awasthi */
```

C#

```
 // C# program to compute Binomial
// Probability.
using System;

class GFG {

    // function to calculate nCr
    // i.e., number of ways to
    // choose r out of n objects
    static int nCr(int n, int r)
    {

        // Since nCr is same as
        // nC(n-r) To decrease
        // number of iterations
        if (r > n / 2)
            r = n - r;

        int answer = 1;
        for (int i = 1; i <= r; i++)
        {
            answer *= (n - r + i);
            answer /= i;
        }

        return answer;
    }

    // function to calculate binomial
    // r.v. probability
    static float binomialProbability(
                  int n, int k, float p)
    {
        return nCr(n, k) *
                     (float)Math.Pow(p, k)
                 * (float)Math.Pow(1 - p,
                                 n - k);
    }

    // Driver code
    public static void Main()
    {
        int n = 10;
        int k = 5;
        float p = (float)1.0 / 3;
        float probability =
              binomialProbability(n, k, p);

        Console.Write("Probability of "
                                      + k);
        Console.Write(" heads when a coin "
                       + "is tossed " + n);
        Console.Write(" times where "
           + "probability of each head is "
                                      + p);
        Console.Write( " is = "
                           + probability );
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// php program to compute Binomial
// Probability

// function to calculate nCr i.e.,
// number of ways to choose r out
// of n objects
function nCr($n, $r)
{

    // Since nCr is same as nC(n-r)
    // To decrease number of iterations
    if ($r > $n / 2)
        $r = $n - $r;

    $answer = 1;
    for ($i = 1; $i <= $r; $i++) {
        $answer *= ($n - $r + $i);
        $answer /= $i;
    }

    return $answer;
}

// function to calculate binomial r.v.
// probability
function binomialProbability($n, $k, $p)
{
    return nCr($n, $k) * pow($p, $k) *
```

```
                   pow(1 - $p, $n - $k);
}

// Driver code
    $n = 10;
    $k = 5;
    $p = 1.0 / 3;

    $probability =
         binomialProbability($n, $k, $p);

    echo "Probability of " . $k;
    echo " heads when a coin is tossed "
                                    . $n;
    echo " times where probability of "
                 . "each head is " . $p ;
    echo " is = " . $probability ;

// This code is contributed by nitin mittal.
?>
```

Output:

```
Probability of 5 heads when a coin is tossed 10 times where probability of each head is 0.333333
 is = 0.136565
```

Reference :
stat200

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/binomial-random-variables/

Chapter 2

Birthday Paradox

Birthday Paradox - GeeksforGeeks

How many people must be there in a room to make the probability 100% that at-least two people in the room have same birthday?
Answer: 367 (since there are 366 possible birthdays, including February 29).
The above question was simple. Try the below question yourself.

How many people must be there in a room to make the probability 50% that at-least two people in the room have same birthday?
Answer: 23
The number is surprisingly very low. In fact, we need only 70 people to make the probability 99.9 %.

Let us discuss the generalized formula.

What is the probability that two persons among n have same birthday?
Let the probability that two people in a room with n have same birthday be P(same). P(Same) can be easily evaluated in terms of P(different) where P(different) is the probability that all of them have different birthday.

P(same) = 1 – P(different)

P(different) can be written as 1 x (364/365) x (363/365) x (362/365) x …. x (1 – (n-1)/365)

How did we get the above expression?
Persons from first to last can get birthdays in following order for all birthdays to be distinct:
The first person can have any birthday among 365
The second person should have a birthday which is not same as first person
The third person should have a birthday which is not same as first two persons.
…………….
……………
The n'th person should have a birthday which is not same as any of the earlier considered (n-1) persons.

Approximation of above expression
The above expression can be approximated using Taylor's Series.

$$e^x = 1 + x + \frac{x^2}{2!} + \dots$$

provides a first-order approximation for ex for x << 1:

$$e^x \approx 1 + x$$

To apply this approximation to the first expression derived for p(different), set x = -a / 365. Thus,

$$e^{\frac{-a}{365}} \approx 1 - \frac{a}{365}$$

The above expression derived for p(different) can be written as
1 x (1 – 1/365) x (1 – 2/365) x (1 – 3/365) x …. x (1 – (n-1)/365)

By putting the value of 1 – a/365 as $e^{-a/365}$, we get following.

$$\approx 1 \times e^{\frac{-1}{365}} \times e^{\frac{-2}{365}} \dots \times e^{\frac{-(n-1)}{365}}$$
$$\approx 1 \times e^{\frac{-(1+2+\dots+(n-1))}{365}}$$
$$\approx 1 \times e^{\frac{-(n(n-1))/2}{365}}$$

Therefore,

p(same) = 1- p(different)

$$\approx 1 - e^{-n(n-1)/(2 \times 365)}$$

An even coarser approximation is given by

p(same)

$$\approx 1 - e^{-n^2/(2 \times 365)}$$

By taking Log on both sides, we get the reverse formula.

$$n \approx \sqrt{2 \times 365 \ln\left(\frac{1}{1 - p(same)}\right)}$$

Using the above approximate formula, we can approximate number of people for a given probability. For example the following C++ function find() returns the smallest n for which the probability is greater than the given p.

Implementation of approximate formula.
The following is program to approximate number of people for a given probability.

C++

```
 // C++ program to approximate number of people in Birthday Paradox
// problem
#include <cmath>
#include <iostream>
using namespace std;

// Returns approximate number of people for a given probability
int find(double p)
{
    return ceil(sqrt(2*365*log(1/(1-p))));
}

int main()
{
   cout << find(0.70);
}
```

Java

```
 // Java program to approximate number
// of people in Birthday Paradox problem
class GFG {

    // Returns approximate number of people
    // for a given probability
    static double find(double p) {

        return Math.ceil(Math.sqrt(2 *
            365 * Math.log(1 / (1 - p))));
    }

    // Driver code
    public static void main(String[] args)
    {

        System.out.println(find(0.70));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 code to approximate number
# of people in Birthday Paradox problem
import math
```

```
# Returns approximate number of
# people for a given probability
def find( p ):
    return math.ceil(math.sqrt(2 * 365 *
                     math.log(1/(1-p))));

# Driver Code
print(find(0.70))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# program to approximate number
// of people in Birthday Paradox problem.
using System;

class GFG {

    // Returns approximate number of people
    // for a given probability
    static double find(double p) {

        return Math.Ceiling(Math.Sqrt(2 *
            365 * Math.Log(1 / (1 - p))));
    }

    // Driver code
    public static void Main()
    {
    Console.Write(find(0.70));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to approximate
// number of people in Birthday
// Paradox problem

// Returns approximate number
// of people for a given probability
function find( $p)
{
    return ceil(sqrt(2 * 365 *
                     log(1 / (1 - $p))));
}

// Driver Code
echo find(0.70);

// This code is contributed by aj_36
?>
```

Output :

```
30
```

Source:
http://en.wikipedia.org/wiki/Birthday_problem

Applications:
1) Birthday Paradox is generally discussed with hashing to show importance of collision handling even for a small set of keys.
2) Birthday Attack

This article is contributed by **Shubham**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : nitin mittal, jit_t

Source

https://www.geeksforgeeks.org/birthday-paradox/

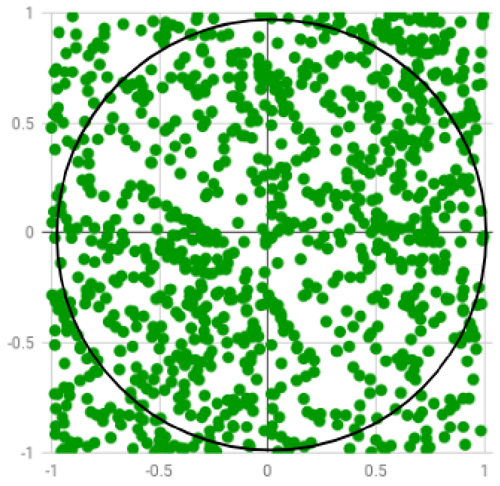
Chapter 3

Estimating the value of Pi using Monte Carlo

Estimating the value of Pi using Monte Carlo - GeeksforGeeks

Monte Carlo estimation
Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. One of the basic examples of getting started with the Monte Carlo algorithm is the estimation of Pi.

Estimation of Pi
The idea is to simulate random (x, y) points in a 2-D plane with domain as a square of side 1 unit. Imagine a circle inside the same domain with same diameter and inscribed into the square. We then calculate the ratio of number points that lied inside the circle and total number of generated points. Refer to the image below:



Random points are generated only few of which lie outside the imaginary circle

We know that area of the square is 1 unit sq while that of circle is [illegible].
Now for a very large number of generated points,

```
that is,
```

The beauty of this algorithm is that we don't need any graphics or simulation to display the generated points. We simply generate random (x, y) pairs and then check if [illegible]. If yes, we increment the number of points that appears inside the circle. In randomized and simulation algorithms like Monte Carlo, the more the number of iterations, the more accurate the result is. Thus, the title is "**Estimating** the value of Pi" and not "Calculating the value of Pi". Below is the algorithm for the method:

The Algorithm
1. Initialize circle_points, square_points and interval to 0.
2. Generate random point x.
3. Generate random point y.
4. Calculate d = x*x + y*y.
5. If d <= 1, increment circle_points.
6. Increment square_points.
7. Increment interval.
8. If increment < NO_OF_ITERATIONS, repeat from 2.
9. Calculate pi = 4*(circle_points/square_points).
10. Terminate.

The code doesn't wait for any input via stdin as the macro INTERVAL could be changed as per the required number of iterations. Number of iterations are the square of INTERVAL. Also, I've paused the screen for first 10 iterations with getch() and outputs are displayed for every iteration with format given below. You can change or delete them as per requirement.

```
x y circle_points square_points - pi
```

Examples:

```
INTERVAL = 5
Output : Final Estimation of Pi = 2.56
```

```
INTERVAL = 10
Output : Final Estimation of Pi = 3.24

INTERVAL = 100
Output : Final Estimation of Pi = 3.0916
```

```
 /* C++ program for estimation of Pi using Monte
   Carlo Simulation */
#include <bits/stdc++.h>

// Defines precision for x and y values. More the
// interval, more the number of significant digits
#define INTERVAL 10000
using namespace std;

int main()
{
    int interval, i;
    double rand_x, rand_y, origin_dist, pi;
    int circle_points = 0, square_points = 0;

    // Initializing rand()
    srand(time(NULL));

    // Total Random numbers generated = possible x
    // values * possible y values
    for (i = 0; i < (INTERVAL * INTERVAL); i++) {

        // Randomly generated x and y values
        rand_x = double(rand() % (INTERVAL + 1)) / INTERVAL;
        rand_y = double(rand() % (INTERVAL + 1)) / INTERVAL;

        // Distance between (x, y) from the origin
        origin_dist = rand_x * rand_x + rand_y * rand_y;

        // Checking if (x, y) lies inside the define
        // circle with R=1
        if (origin_dist <= 1)
            circle_points++;

        // Total number of points generated
        square_points++;

        // estimated pi after this iteration
        pi = double(4 * circle_points) / square_points;

        // For visual understanding (Optional)
        cout << rand_x << " " << rand_y << " " << circle_points
```

```
             << " " << square_points << " - " << pi << endl << endl;

        // Pausing estimation for first 10 values (Optional)
        if (i < 20)
            getchar();
    }

    // Final Estimated Value
    cout << "\nFinal Estimation of Pi = " << pi;

    return 0;
}
```

Output:

```

Final Estimation of Pi = 3.16116
```

Source

https://www.geeksforgeeks.org/estimating-value-pi-using-monte-carlo/

Chapter 4

Expectation or expected value of an array

Expectation or expected value of an array - GeeksforGeeks

Expectation or expected valueof any group of numbers in probability is the long-run average value of repetitions of the experiment it represents. For example, the expected value in rolling a six-sided die is 3.5, because the average of all the numbers that come up in an extremely large number of rolls is close to 3.5. Less roughly, the law of large numbers states that the arithmetic mean of the values almost surely converges to the expected value as the number of repetitions approaches infinity. The expected value is also known as the **expectation**, **mathematical expectation**, EV, or first moment.
Given an array, the task is to calculate the **expected value** of the array.

Examples :

```
Input : [1.0, 2.0, 3.0, 4.0, 5.0, 6.0]
Output : 3.5

Input :[1.0, 9.0, 6.0, 7.0, 8.0, 12.0]
Output :7.16
```

Below is the implementation :

C++

```
 // CPP code to calculate expected
// value of an array
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to calculate expectation
float calc_Expectation(float a[], float n)
{
    /*variable prb is for probability
    of each element which is same for
    each element  */
    float prb = (1 / n);

    // calculating expectation overall
    float sum = 0;
    for (int i = 0; i < n; i++)
        sum += a[i] * prb;

    // returning expectation as sum
    return sum;
}

// Driver program
int main()
{
    float expect, n = 6.0;
    float a[6] = { 1.0, 2.0, 3.0,
                   4.0, 5.0, 6.0 };

    // Function for calculating expectation
    expect = calc_Expectation(a, n);

    // Display expectation of given array
    cout << "Expectation of array E(X) is : "
         << expect << "\n";
    return 0;
}
```

Java

```
 // Java code to calculate expected
// value of an array
import java.io.*;

class GFG
{
    // Function to calculate expectation
    static float calc_Expectation(float a[], float n)
    {
        // Variable prb is for probability of each
        // element which is same for each element
        float prb = (1 / n);
        // calculating expectation overall
        float sum = 0;
        for (int i = 0; i < n; i++)
            sum += a[i] * prb;

        // returning expectation as sum
        return sum;
    }

    // Driver program
    public static void main(String args[])
    {
        float expect, n = 6f;
        float a[] = { 1f, 2f, 3f,
                      4f, 5f, 6f };

        // Function for calculating expectation
        expect = calc_Expectation(a, n);

        // Display expectation of given array
        System.out.println("Expectation of array E(X) is : "
                           + expect);
    }
}

// This code is contributed by Anshika Goyal.
```

Python3

```
 # python code to calculate expected
# value of an array

# Function to calculate expectation
def calc_Expectation(a, n):

    # variable prb is for probability
    # of each element which is same for
    # each element
    prb = 1 / n

    # calculating expectation overall
    sum = 0
    for i in range(0, n):
        sum += (a[i] * prb)

    # returning expectation as sum
    return float(sum)
```

```
# Driver program
n = 6;
a = [ 1.0, 2.0, 3.0,4.0, 5.0, 6.0 ]

# Function for calculating expectation
expect = calc_Expectation(a, n)

# Display expectation of given array
print( "Expectation of array E(X) is : ",
                                 expect )

# This code is contributed by Sam007
```

C#

```
 // C# code to calculate expected
// value of an array
using System;

class GFG {

    // Function to calculate expectation
    static float calc_Expectation(float []a,
                                    float n)
    {

        // Variable prb is for probability
        // of each element which is same
        // for each element
        float prb = (1 / n);

        // calculating expectation overall
        float sum = 0;

        for (int i = 0; i < n; i++)
            sum += a[i] * prb;

        // returning expectation as sum
        return sum;
    }

    // Driver program
    public static void Main()
    {
        float expect, n = 6f;
        float []a = { 1f, 2f, 3f,
                    4f, 5f, 6f };
        // Function for calculating
        // expectation
        expect = calc_Expectation(a, n);

        // Display expectation of given
        // array
        Console.WriteLine("Expectation"
               + " of array E(X) is : "
                             + expect);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP code to calculate expected
// value of an array

// Function to calculate expectation
function calc_Expectation($a, $n)
{
    /*variable prb is for probability
    of each element which is same for
    each element */
    $prb = (1 / $n);

    // calculating expectation overall
    $sum = 0;
    for ($i = 0; $i < $n; $i++)
        $sum += $a[$i] * $prb;

    // returning expectation as sum
    return $sum;
}

// Driver Code
$n = 6.0;
$a = array(1.0, 2.0, 3.0,
           4.0, 5.0, 6.0);

// Function for calculating expectation
$expect = calc_Expectation($a, $n);

// Display expectation of given array
echo "Expectation of array E(X) is : ".
```

```
                        $expect . "\n";

// This code is contributed by Sam007
?>
```

Output :

```
Expectation of array E(X) is : 3.5
```

Time complexity of the program is **O(n)**.

As we can see that the expected value is actually average of numbers, we can also simply compute average of array.

Improved By : vt_m, Sam007

Source

https://www.geeksforgeeks.org/expectation-expected-value-array/

Chapter 5

Expected Number of Trials until Success

Expected Number of Trials until Success - GeeksforGeeks

Consider the following famous puzzle.

In a country, all families want a boy. They keep having babies till a boy is born. What is the expected ratio of boys and girls in the country?

This puzzle can be easily solved if we know following interesting result in probability and expectation.

If probability of success is p in every trial, then expected number of trials until success is 1/p

Proof: Let R be a random variable that indicates number of trials until success.

```
The expected value of R is sum of following infinite series
E[R] = 1*p + 2*(1-p)*p + 3*(1-p)2*p + 4*(1-p)3*p + ........

Taking 'p' out
E[R] = p[1 + 2*(1-p) + 3*(1-p)2 + 4*(1-p)3 + .......] ---->(1)

Multiplying both sides with '(1-p)' and subtracting
(1-p)*E[R] = p[1*(1-p) + 2*(1-p)2 + 3*(1-p)3 + .......] --->(2)

Subtracting (2) from (1), we get

p*E[R] = p[1 + (1-p) + (1-p)2 + (1-p)3 + ........]

Canceling p from both sides
E[R] = [1 + (1-p) + (1-p)2 + (1-p)3 + ........]
```

```
Above is an  infinite geometric  progression with ratio (1-p).
Since (1-p) is less than, we can apply sum formula.
  E[R] = 1/[1 - (1-p)]
       = 1/p
```

Solution of Boys/Girls ratio puzzle:
Let us use the above result to solve the puzzle. In the given puzzle, probability of success in every trial is 1/2 (assuming that girls and boys are equally likely).

```
Let p be probability of having a baby boy.
Number of kids until a baby boy is born = 1/p
                                        = 1/(1/2)
                                        = 2
Since expected number of kids in a family is 2,
ratio of boys and girls is 50:50.
```

Let us discuss another problem that uses above result.

Coupon Collector Problem:
Suppose there are n types of coupons in a lottery and each lot contains one coupon (with probability 1 = n each). How many lots have to be bought (in expectation) until we have at least one coupon of each type.

The solution of this problem is also based on above result.

Let X_i be the number of lots bought before i'th new coupon is collected.

Note that X_1 is 1 as the first coupon is always a new coupon (not collected before).

Let 'p' be probability that 2nd coupon is collected in next buy. The value of p is (n-1)/n. So the number of trials needed before 2nd new coupon is picked is 1/p which means n/(n-1). [This is where we use above result]

Similarly, the number of trials needed before 3rd new coupon is collected is n/(n-2)

```
Using Linearity of expectation,
we can say that the total number of expected trials =
                 1 + n/(n-1) + n/(n-2) + n/(n-3) + .... + n/2 + n/1
               = n[1/n + 1/(n-1) + 1/(n-2) + 1/(n-3) + ....+ 1/2 + 1/1]
               = n * Hn
Here Hn is n-th Harmonic number

Since Logn <= Hn <= Logn + 1, we need to buy around
nLogn lots to collect all n coupons.
```

Exercise:
1) A 6 faced fair dice is thrown until a '5' is seen as result of dice throw. What is the expected number of throws?

2) What is the ratio of boys and girls in above puzzle if probability of a baby boy is 1/3?

Reference:

http://www.cse.iitd.ac.in/~mohanty/col106/Resources/linearity_expectation.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-video-lectures/lecture-22-expectation-i/

Source

https://www.geeksforgeeks.org/expected-number-of-trials-before-success/

Chapter 6

Find an index of maximum occurring element with equal probability

Find an index of maximum occurring element with equal probability - GeeksforGeeks

Given an array of integers, find the most occurring element of the array and return any one of its indexes randomly with equal probability.

Examples:

```
Input:
arr[] = [-1, 4, 9, 7, 7, 2, 7, 3, 0, 9, 6, 5, 7, 8, 9]

Output:
Element with maximum frequency present at index 6
OR
Element with maximum frequency present at Index 3
OR
Element with maximum frequency present at index 4
OR
Element with maximum frequency present at index 12

All outputs above have equal probability.
```

The idea is to iterate through the array once and find out the maximum occurring element and its frequency n. Then we generate a random number r between 1 and n and finally return the r'th occurrence of maximum occurring element in the array.

Below is C++ implementation of above idea –

```
 // C++ program to return index of most occurring element
// of the array randomly with equal probability
#include <iostream>
#include <unordered_map>
#include <climits>
using namespace std;

// Function to return index of most occurring element
// of the array randomly with equal probability
void findRandomIndexOfMax(int arr[], int n)
{
    // freq store frequency of each element in the array
    unordered_map<int, int> freq;
    for (int i = 0; i < n; i++)
        freq[arr[i]] += 1;

    int max_element; // stores max occurring element

    // stores count of max occurring element
    int max_so_far = INT_MIN;

    // traverse each pair in map and find maximum
    // occurring element and its frequency
    for (pair<int, int> p : freq)
    {
        if (p.second > max_so_far)
        {
            max_so_far = p.second;
            max_element = p.first;
        }
    }

    // generate a random number between [1, max_so_far]
    int r = (rand() % max_so_far) + 1;

    // traverse array again and return index of rth
    // occurrence of max element
    for (int i = 0, count = 0; i < n; i++)
    {
        if (arr[i] == max_element)
            count++;

        // print index of rth occurence of max element
        if (count == r)
        {
            cout << "Element with maximum frequency present "
                 "at index " << i << endl;
            break;
        }
    }
}

// Driver code
int main()
{
    // input array
    int arr[] = { -1, 4, 9, 7, 7, 2, 7, 3, 0, 9, 6, 5,
                  7, 8, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // randomize seed
    srand(time(NULL));

    findRandomIndexOfMax(arr, n);

    return 0;
}
```

Output:

```
Element with maximum frequency present at index 12
```

Time complexity of above solution is O(n).
Auxiliary space used by the program is O(n).

Source

https://www.geeksforgeeks.org/find-index-maximum-occurring-element-equal-probability/

Chapter 7

Freivald's Algorithm to check if a matrix is product of two

Freivald's Algorithm to check if a matrix is product of two - GeeksforGeeks

Given three matrices A, B and C, find if C is a product of A and B.

Examples:

```
Input : A = 1 1
            1 1
        B = 1 1
            1 1
        C = 2  2
             2 2
Output : Yes
C = A x B

Input : A = 1 1 1
            1 1 1
            1 1 1
        B = 1 1 1
            1 1 1
            1 1 1
        C = 3 3 3
            3 1 2
            3 3 3
Output : No
```

A **simple solution** is to find product of A and B and then check if product is equal to C or not. A possible time complexity of this method is $O(n^{2.8874})$ using Stression's matrix multiplication.

Freivalds' algorithm is a probabilistic randomized algorithm that works in time $\mathbf{O(n^2)}$ with high probability. In $\mathbf{O(kn^2)}$ time the algorithm can verify a matrix product with probability of failure less than $\mathbf{2^{-k}}$. Since the output is not always correct, it is a Monte Carlo randomized algorithm.

Steps :

1. Generate an $\mathbf{n \times 1}$ random 0/1 vector $\mathbf{r}$.
2. Compute $\mathbf{P = A \times (Br) - Cr}$.
3. Return true if $\mathbf{P = (0, 0, ..., 0)^T}$, return false otherwise.

The idea is based on the fact that if C is actually a product, then value of A × (Br) − Cr will always be 0. If the value is non-zero, then C can not be a product. The error condition is that the value may be 0 even when C is not a product.

C++

```
 // CPP code to implement Freivald's Algorithm
#include <bits/stdc++.h>
using namespace std;

#define N 2

// Function to check if ABx = Cx
int freivald(int a[][N], int b[][N], int c[][N])
{
    // Generate a random vector
    bool r[N];
    for (int i = 0; i < N; i++)
        r[i] = random() % 2;

    // Now comput B*r for evaluating
    // expression A * (B*r) - (C*r)
    int br[N] = { 0 };
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            br[i] = br[i] + b[i][j] * r[j];

    // Now comput C*r for evaluating
    // expression A * (B*r) - (C*r)
    int cr[N] = { 0 };
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            cr[i] = cr[i] + c[i][j] * r[j];

    // Now comput A* (B*r) for evaluating
    // expression A * (B*r) - (C*r)
    int axbr[N] = { 0 };
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            axbr[i] = axbr[i] + a[i][j] * br[j];

    // Finally check if value of expression
    // A * (B*r) - (C*r) is 0 or not
    for (int i = 0; i < N; i++)
        if (axbr[i] - cr[i] != 0)
            false;

    return true;
}

// Runs k iterations Freivald. The value
// of k determines accuracy. Higher value
// means higher accuracy.
bool isProduct(int a[][N], int b[][N],
               int c[][N], int k)
{
    for (int i=0; i<k; i++)
        if (freivald(a, b, c) == false)
            return false;
    return true;
}

// Driver code
int main()
{
    int a[N][N] = { { 1, 1 }, { 1, 1 } };
    int b[N][N] = { { 1, 1 }, { 1, 1 } };
    int c[N][N] = { { 2, 2 }, { 2, 2 } };
    int k = 2;
    if (isProduct(a, b, c, k))
        printf("Yes");
    else
        printf("No");
    return 0;
}
```

Java

```
 // Java code to implement
// Freivald's Algorithm
import java.io.*;
import java.util.*;
import java.math.*;

class GFG {
```

```
static int N = 2;

// Function to check if ABx = Cx
static boolean freivald(int a[][], int b[][],
                                   int c[][])
{
    // Generate a random vector
    int r[] = new int[N];
    for (int i = 0; i < N; i++)
    r[i] = (int)(Math.random()) % 2;

    // Now comput B*r for evaluating
    // expression A * (B*r) - (C*r)
    int br[] = new int[N];
    Arrays.fill(br, 0);
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            br[i] = br[i] + b[i][j] * r[j];

    // Now comput C*r for evaluating
    // expression A * (B*r) - (C*r)
    int cr[] = new int[N];
    Arrays.fill(cr, 0);
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            cr[i] = cr[i] + c[i][j] * r[j];

    // Now comput A* (B*r) for evaluating
    // expression A * (B*r) - (C*r)
    int axbr[] = new int[N];
    Arrays.fill(axbr, 0);
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            axbr[i] = axbr[i] + a[i][j] * br[j];

    // Finally check if value of expression
    // A * (B*r) - (C*r) is 0 or not
    for (int i = 0; i < N; i++)
        if (axbr[i] - cr[i] != 0)
            return false;

    return true;
}

// Runs k iterations Freivald. The value
// of k determines accuracy. Higher value
// means higher accuracy.
static boolean isProduct(int a[][], int b[][],
                                 int c[][], int k)
    {
        for (int i = 0; i < k; i++)
            if (freivald(a, b, c) == false)
                return false;
        return true;
    }

    // Driver code
    public static void main(String args[])
    {
        int a[][] = { { 1, 1 }, { 1, 1 } };
        int b[][] = { { 1, 1 }, { 1, 1 } };
        int c[][] = { { 2, 2 }, { 2, 2 } };
        int k = 2;
        if (isProduct(a, b, c, k))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

/*This code is contributed by Nikita Tiwari.*/
```

Python3

```
 # Python3 code to implement Freivald's Algorithm
from random import *
N = 2

# Function to check if ABx = Cx
def freivald(a, b, c) :

    # Generate a random vector
    r = [0] * N

    for i in range(0, N) :
        r[i] = (int)(random()) % 2

    # Now comput B*r for evaluating
    # expression A * (B*r) - (C*r)
    br = [0] * N

    for i in range(0, N) :
        for j in range(0, N) :
            br[i] = br[i] + b[i][j] * r[j]

    # Now comput C*r for evaluating
    # expression A * (B*r) - (C*r)
    cr = [0] * N
    for i in range(0, N) :
        for j in range(0, N) :
            cr[i] = cr[i] + c[i][j] * r[j]

    # Now comput A* (B*r) for evaluating
    # expression A * (B*r) - (C*r)
    axbr = [0] * N
    for i in range(0, N) :
        for j in range(0, N) :
            axbr[i] = axbr[i] + a[i][j] * br[j]

    # Finally check if value of expression
    # A * (B*r) - (C*r) is 0 or not
    for i in range(0, N) :
        if (axbr[i] - cr[i] != 0) :
            return False

    return True

# Runs k iterations Freivald. The value
# of k determines accuracy. Higher value
# means higher accuracy.
def isProduct(a, b, c, k) :

    for i in range(0, k) :
        if (freivald(a, b, c) == False) :
            return False
    return True

# Driver code
a = [ [ 1, 1 ], [ 1, 1 ] ]
b = [ [ 1, 1 ], [ 1, 1 ] ]
c = [ [ 2, 2 ], [ 2, 2 ] ]
k = 2

if (isProduct(a, b, c, k)) :
    print("Yes")
else :
    print("No")

# This code is contributed by Nikita Tiwari
```

C#

```
 // C# code to implement
// Freivald's Algorithm
```

```
using System;

class GFG
{
    static int N = 2;

    // Function to check
    // if ABx = Cx
    static bool freivald(int [,]a,
                         int [,]b,
                         int [,]c)
    {
        // Generate a
        // random vector
        Random rand = new Random();
        int []r = new int[N];

        for (int i = 0; i < N; i++)
        r[i] = (int)(rand.Next()) % 2;

        // Now compute B*r for
        // evaluating expression
        // A * (B*r) - (C*r)
        int []br = new int[N];

        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                br[i] = br[i] +
                        b[i, j] * r[j];

        // Now compute C*r for
        // evaluating expression
        // A * (B*r) - (C*r)
        int []cr = new int[N];

        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                cr[i] = cr[i] +
                        c[i, j] * r[j];

        // Now compute A* (B*r) for
        // evaluating expression
        // A * (B*r) - (C*r)
        int []axbr = new int[N];

        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                axbr[i] = axbr[i] +
                      a[i, j] * br[j];

        // Finally check if value
        // of expression A * (B*r) -
        // (C*r) is 0 or not
        for (int i = 0; i < N; i++)
            if (axbr[i] - cr[i] != 0)
                return false;

        return true;
    }

    // Runs k iterations Freivald.
    // The value of k determines
    // accuracy. Higher value
    // means higher accuracy.
    static bool isProduct(int [,]a, int [,]b,
                      int [,]c, int k)
    {
        for (int i = 0; i < k; i++)
            if (freivald(a, b, c) == false)
                return false;
        return true;
    }

    // Driver code
    static void Main()
    {
        int [,]a = new int[,]{ { 1, 1 },
                               { 1, 1 }};
        int [,]b = new int[,]{ { 1, 1 },
                               { 1, 1 }};
        int [,]c = new int[,]{ { 2, 2 },
                               { 2, 2 }};
        int k = 2;
        if (isProduct(a, b, c, k))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}
// This code is contributed
// by Manish Shaw(manishshaw1)
```

PHP

```
 <?php
// PHP code to implement
// Freivald's Algorithm
$N = 2;

// Function to check
// if ABx = Cx
function freivald($a, $b, $c)
{
    global $N;

    // Generate a
    // random vector
    $r = array();
    $br = array();
    $cr = array();
    $axbr = array();

    for ($i = 0; $i < $N; $i++)
    {
        $r[$i] = mt_rand() % 2;
        $br[$i] = 0;
        $cr[$i] = 0;
        $axbr[$i] = 0;
    }

    // Now comput B*r for
    // evaluating expression
    // A * (B*r) - (C*r)
    for ($i = 0; $i < $N; $i++)
    {
        for ($j = 0; $j < $N; $j++)
            $br[$i] = $br[$i] +
                      $b[$i][$j] *
                      $r[$j];
    }

    // Now comput C*r for
    // evaluating expression
    // A * (B*r) - (C*r)
    for ($i = 0; $i < $N; $i++)
    {
        for ($j = 0; $j < $N; $j++)
            $cr[$i] = $cr[$i] +
                      $c[$i][$j] *
                      $r[$j];
    }

    // Now comput A* (B*r) for
    // evaluating expression
    // A * (B*r) - (C*r)
    for ($i = 0; $i < $N; $i++)
    {
        for ($j = 0; $j < $N; $j++)
            $axbr[$i] = $axbr[$i] +
                        $a[$i][$j] *
                        $br[$j];
    }

    // Finally check if value
    // of expression A * (B*r) -
    // (C*r) is 0 or not
    for ($i = 0; $i < $N; $i++)
        if ($axbr[$i] - $cr[$i] != 0)
            return false;

    return true;
}

// Runs k iterations Freivald.
// The value of k determines
// accuracy. Higher value
// means higher accuracy.
function isProduct($a, $b, $c, $k)
{
    for ($i = 0; $i < $k; $i++)
        if (freivald($a,
                     $b, $c) == false)
            return false;
    return true;
}

// Driver code
$a = array(array(1, 1),
           array(1, 1));
$b = array(array(1, 1),
           array(1, 1));
$c = array(array(2, 2),
           array(2, 2));
$k = 2;
if (isProduct($a, $b,
              $c, $k))
    echo ("Yes");
else
    echo ("No");

// This code is contributed
// by Manish Shaw(manishshaw1)
```

```
?>
```

Output:

```
Yes
```

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/freivalds-algorithm/

Chapter 8

Generate 0 and 1 with 25% and 75% probability

Generate 0 and 1 with 25% and 75% probability - GeeksforGeeks

Given a function rand50() that returns 0 or 1 with equal probability, write a function that returns 1 with 75% probability and 0 with 25% probability using rand50() only. Minimize the number of calls to rand50() method. Also, use of any other library function and floating point arithmetic are not allowed.

The idea is to use **Bitwise OR**. A bitwise OR takes two bits and returns 0 if both bits are 0, while otherwise the result is 1. So it has 75% probability that it will return 1.

Below is the implementation of above idea :

C++

```
 // Program to print 1 with 75% probability and 0
// with 25% probability
#include <iostream>
using namespace std;

// Random Function to that returns 0 or 1 with
// equal probability
int rand50()
{
    // rand() function will generate odd or even
    // number with equal probability. If rand()
    // generates odd number, the function will
    // return 1 else it will return 0.
    return rand() & 1;
}

// Random Function to that returns 1 with 75%
// probability and 0 with 25% probability using
// Bitwise OR
bool rand75()
{
    return rand50() | rand50();
}

// Driver code to test above functions
int main()
{
    // Intialize random number generator
    srand(time(NULL));

    for(int i = 0; i < 50; i++)
        cout << rand75();

    return 0;
}
```

PHP

```
 <?php
// Program to print 1 with 75% probability
// and 0 with 25% probability

// Random Function to that returns 0 or
// 1 with equal probability
function rand50()
{

    // rand() function will generate
    // odd or even number with equal
    // probability. If rand() generates
    // odd number, the function will
    // return 1 else it will return 0.
    return rand() & 1;
}

// Random Function to that returns
// 1 with 75% probability and 0
// with 25% probability using
// Bitwise OR
function rand75()
{
    return rand50() | rand50();
}

    // Driver Code
    // Intialize random
    // number generator
    srand(time(NULL));

    for($i = 0; $i < 50; $i++)
        echo rand75();

// This code is contributed m_kit
?>
```

Output:

```
11101111110010010110011111111101111110111100011000
```

On similar lines, we can also use **Bitwise AND**. Since it returns 0 with 75% probability, we have to invert the result.

```
// Random Function to that returns 1 with 75%
// probability and 0 with 25% probability using
// Bitwise AND
bool rand75()
{
    return !(rand50() & rand50());
}
```

We can replace Bitwise OR and Bitwise AND operator by **OR and AND operators** as well –

```
// Random Function to that returns 1 with 75%
// probability and 0 with 25% probability using
// OR or AND operator
int rand75()
{
    return !(rand50() && rand50());
    // return rand50() || rand50()
}
```

We can also achieve the result using **left shift operator and Bitwise XOR** –

C++

```
 // Program to print 1 with 75% probability and 0
// with 25% probability
```

```
#include <iostream>
using namespace std;

// Random Function to that returns 0 or 1 with
// equal probability
int rand50()
{
    // rand() function will generate odd or even
    // number with equal probability. If rand()
    // generates odd number, the function will
    // return 1 else it will return 0.
    return rand() & 1;
}

// Random Function to that returns 1 with 75%
// probability and 0 with 25% probability using
// left shift and Bitwise XOR
int rand75()
{
    // x is one of {0, 1}
    int x = rand50();

    x = x << 1;

    // x is now one of {00, 10}

    x = x ^ rand50();

    // x is now one of {00, 01, 10, 11}

    return (x > 0) ? 1 : 0;
}

// Driver code to test above functions
int main()
{
    // Intialize random number generator
    srand(time(NULL));

    for (int i = 0; i < 50; i++)
        cout << rand75();

    return 0;
}
```

PHP

```
<?php
```

```
// Program to print 1 with
// 75% probability and 0
// with 25% probability

// Random Function to that
// returns 0 or 1 with
// equal probability
function rand50()
{
    // rand() function will
    // generate odd or even
    // number with equal
    // probability. If rand()
    // generates odd number,
    // the function will return
    // 1 else it will return 0.
    return rand() & 1;
}

// Random Function to that
// returns 1 with 75%
// probability and 0 with
// 25% probability using
// left shift and Bitwise XOR
function rand75()
{
    // x is one of {0, 1}
    $x = rand50();

    $x = $x << 1;

    // x is now one
    // of {00, 10}

    $x = $x ^ rand50();

    // x is now one of
    // {00, 01, 10, 11}

    return ($x > 0) ? 1 : 0;
}

// Driver code

// Intialize random
// number generator
srand(time(NULL));
```

```
for ($i = 0; $i < 50; $i++)
    echo rand75();

// This code is contributed
// by ajit
?>
```

Output:

```
01101110111011000111111111111000111101110111011011 0
```

Please note above solutions will produce **different results** every time we run them.

Improved By : jit_t

Source

https://www.geeksforgeeks.org/generate-0-1-25-75-probability/

Chapter 9

Generate integer from 1 to 7 with equal probability

Generate integer from 1 to 7 with equal probability - GeeksforGeeks

Given a function foo() that returns integers from 1 to 5 with equal probability, write a function that returns integers from 1 to 7 with equal probability using foo() only. Minimize the number of calls to foo() method. Also, use of any other library function is not allowed and no floating point arithmetic allowed.

Solution :
We know foo() returns integers from 1 to 5. How we can ensure that integers from 1 to 7 occur with equal probability?
If we somehow generate integers from 1 to a-multiple-of-7 (like 7, 14, 21, …) with equal probability, we can use modulo division by 7 followed by adding 1 to get the numbers from 1 to 7 with equal probability.

We can generate from 1 to 21 with equal probability using the following expression.

```
 5*foo() + foo() -5
```

Let us see how above expression can be used.
1. For each value of first foo(), there can be 5 possible combinations for values of second foo(). So, there are total 25 combinations possible.
2. The range of values returned by the above equation is 1 to 25, each integer occurring exactly once.
3. If the value of the equation comes out to be less than 22, return modulo division by 7 followed by adding 1. Else, again call the method recursively. The probability of returning each integer thus becomes 1/7.

The below program shows that the expression returns each integer from 1 to 25 exactly once.

CPP

```
 #include <stdio.h>

int main()
{
    int first, second;
    for ( first=1; first<=5; ++first )
        for ( second=1; second<=5; ++second )
            printf ("%d \n", 5*first + second - 5);
    return 0;
}
```

Java

```
 // Java code to demonstrate
// expression returns each integer
// from 1 to 25 exactly once

class GFG {
    public static void main(String[] args)
    {
        int first, second;
        for ( first=1; first<=5; ++first )
            for ( second=1; second<=5; ++second )
            System.out.printf ("%d \n", 5*first + second - 5);
    }
}

// This code is contributed by
// Smitha Dinesh Semwal
```

Python3

```
 # Python3 code to demonstrate
# expression returns each integer
# from 1 to 25 exactly once

if name == '__main__':

    for first in range(1, 6):
        for second in range(1, 6):
                print(5 * first + second - 5)

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# code to demonstrate expression returns
```

```
// each integer from 1 to 25 exactly once
using System;

class GFG {

    public static void Main()
    {
        int first, second;

        for ( first = 1; first <= 5; ++first )
            for ( second = 1; second <= 5; ++second )
                Console.WriteLine ( 5*first + second - 5);
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to Generate integer
// from 1 to 7 with equal probability

    $first;
    $second;
    for ( $first = 1; $first <= 5; ++$first )
        for ( $second = 1; $second <= 5; ++$second )
            echo 5 * $first + $second - 5, "\n";

// This code is contributed by ajit.
?>
```

Output :

```
1
2
.
.
24
25
```

The below program depicts how we can use foo() to return 1 to 7 with equal probability.

```
 #include <stdio.h>

int foo() // given method that returns 1 to 5 with equal probability
```

```
{
    // some code here
}

int my_rand() // returns 1 to 7 with equal probability
{
    int i;
    i = 5*foo() + foo() - 5;
    if (i < 22)
        return i%7 + 1;
    return my_rand();
}

int main()
{
    printf ("%d ", my_rand());
    return 0;
}
```

This article is compiled by **Aashish Barnwal** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : jit_t

Source

https://www.geeksforgeeks.org/generate-integer-from-1-to-7-with-equal-probability/

Chapter 10

Implement rand12() using rand6() in one line

Implement rand12() using rand6() in one line - GeeksforGeeks

Given a function rand6() that returns random numbers from 1 to 6 with equal probability, implement one-liner function rand12() using rand6() that returns random numbers from 1 to 12 with equal probability. Solution should minimize the number of calls to rand6() method. Use of any other library function and floating point arithmetic are not allowed.

The idea is to use expression **rand6() + (rand6() % 2) * 6**. It returns random numbers from 1 to 12 with equal probability. The expression is equivalent to –

```
// if rand6() is even
if (rand6() % 2)
    return 6 + rand6();
else // if rand6() is odd
    return rand6();
```

We can also use any one of below expressions that works in a similar way –

- **rand6() + !(rand6() % 2) * 6** or
- **rand6() + (rand6() & 1) * 6** or
- **rand6() + !(rand6() & 1) * 6**

Below is C++ implementation of above idea –

```
 // C++ Program to print random numbers from 1 to 12
// with equal probability using a function that returns
// random numbers from 1 to 6 with equal probability
#include <iostream>
```

```
using namespace std;

// Function that returns random numbers from 1 to 6
// with equal probability
int rand6()
{
    // rand() will generate random numbers between 0 and
    // RAND_MAX with equal probability
    // rand() % 6 returns number from 0 to 5 equal probability
    // (rand() % 6) + 1 returns number from 1 to 6 with
    // equal probability
    return (rand() % 6) + 1;
}

// The function uses rand6() to return random numbers
// from 1 to 12 with equal probability
int rand12()
{
    return rand6() + (rand6() % 2) * 6;
}

// Driver code to test above functions
int main()
{
    // Initialize random number generator
    srand(time(NULL));
    int N = 12;

    int freq[N + 1] = { 0 };

    // call rand12() multiple times and store its results
    for (int i = 0; i < N * 100000; i++)
        freq[rand12()]++;

    // print frequency of numbers 1-12
    for (int i = 1; i <= N; i++)
        cout << freq[i] << " ";

    return 0;
}
```

Output:

```
100237 100202 99678 99867 100253 99929 100287 100449 99827 99298 100019 99954
```

Another Solution –

```
int rand12()
{
    return (rand6() * 2) - (rand6() & 1);
}
```

rand6() * 2 will return even numbers 2, 4, 6, 8, 10 and 12 with equal probability and **rand6() & 1** will return 0 or 1 based on rand6() is even or odd respectively. So, the expression **(rand6() * 2)** – **(rand6() & 1)** will return random numbers from 1 to 12 with equal probability.

Please note above solutions will produce different results every time we run them.

Source

https://www.geeksforgeeks.org/implement-rand12-using-rand6-in-one-line/

Chapter 11

Implement rand3() using rand2()

Implement rand3() using rand2() - GeeksforGeeks

Given a function rand2() that returns 0 or 1 with equal probability, implement rand3() using rand2() that returns 0, 1 or 2 with equal probability. Minimize the number of calls to rand2() method. Also, use of any other library function and floating point arithmetic are not allowed.

The idea is to use expression **2 * rand2() + rand2()**. It returns 0, 1, 2, 3 with equal probability. To make it return 0, 1, 2 with equal probability, we eliminate the undesired event 3.

Below is the implementation of above idea –

C++

```
 // C++ Program to print 0, 1 or 2 with equal
// probability
#include <iostream>
using namespace std;

// Random Function to that returns 0 or 1 with
// equal probability
int rand2()
{
    // rand() function will generate odd or even
    // number with equal probability. If rand()
    // generates odd number, the function will
    // return 1 else it will return 0.
    return rand() & 1;
}
```

```
// Random Function to that returns 0, 1 or 2 with
// equal probability 1 with 75%
int rand3()
{
    // returns 0, 1, 2 or 3 with 25% probability
    int r = 2 * rand2() + rand2();

    if (r < 3)
        return r;

    return rand3();
}

// Driver code to test above functions
int main()
{
    // Intialize random number generator
    srand(time(NULL));

    for(int i = 0; i < 100; i++)
        cout << rand3();

    return 0;
}
```

PHP

```
 <?php
// PHP Program to print 0, 1 or
// 2 with equal probability

// Random Function to that
// returns 0 or 1 with
// equal probability
function rand2()
{
    // rand() function will generate
    // odd or even number with equal
    // probability. If rand() generates
    // odd number, the function will
    // return 1 else it will return 0.
    return rand() & 1;
}

// Random Function to that
// returns 0, 1 or 2 with
// equal probability 1 with 75%
```

```
function rand3()
{
    // returns 0, 1, 2 or 3
    // with 25% probability
    $r = 2 * rand2() + rand2();

    if ($r < 3)
        return $r;

    return rand3();
}

// Driver Code

// Intialize random
// number generator
srand(time(NULL));

for($i = 0; $i < 100; $i++)
    echo rand3();

// This code is contributed by aj_36
?>
```

Output :

```
211101110111200211100202021011202202202221110010012120202110210001020012111212101220110221110200
```

Another Solution –

If x = rand2() and y = rand2(), x + y will return 0 and 2 with 25% probability and 1 with 50% probability. To make probability of 1 equal to that of 0 and 2 i.e. 25%, we eliminate one undesired event that's resulting in x + y = 1 i.e. either (x = 1, y = 0) or (x = 0, y = 1).

```
int rand3()
{
    int x, y;

    do {
        x = rand2();
        y = rand2();
    } while (x == 0 && y == 1);

    return x + y;
}
```

Please note above solutions will produce different results every time we run them.

Improved By : jit_t

Source

https://www.geeksforgeeks.org/implement-rand3-using-rand2/

Chapter 12

Implement random-0-6-Generator using the given random-0-1-Generator

Implement random-0-6-Generator using the given random-0-1-Generator - GeeksforGeeks

Given a function random01Generator() that gives you randomly either 0 or 1, implement a function that utilizes this function and generate numbers between 0 and 6(both inclusive). All numbers should have same probabilities of occurrence.

Examples:

```
on multiple runs, it gives
3
2
3
6
0
```

Approach : The idea here is to find the range of the smallest number greater than the given range. Here, since range is 6, decide that 2^3 is the smallest number greater than 6. Hence, k here will be 3.

So, try forming a binary number of 3 characters each time and once all 3 characters are obtained, look up for the corresponding decimal representation of that binary number and return that value.

For e.g., if it gives 0 in first random01 call, then 1 in next call and finally again 1 in next call, it can be said that binary number thus formed is 011 which is decimal representation of number 3 and hence, return 3. If in case it gives 111, which is 7 which out of range. So, just discard this number and repeat the whole process again until it gives the number in the required range.

On calling random01Generator() 6 times, the probability of occurrence of numbers won't be same. For e.g., for first time, the probability of occurrence of 0 is 1/2. Now, for second time, again the probability is 1/2 which makes total probability as 1/4. If this process is repeated 6 times by calling the random function 6 times, this will make the probability of occurrence of 0 as 1/(2^6). Similarly, the probability of occurrence of 1 will be greater than that of 0, of 2 will be greater than 1 and so on, following the binomial distribution pattern.

Doing above way will ensure that all probabilities are equal as it is considering only that range in which one can generate those numbers and rejecting any cases giving us result greater than the maximum specified range. Hence, the above approach.

Obviously, this idea can be extended, if there is some other random generator, say random0mGenerator() and there is need to generate number between 0 to n, where n>m. In this case, modify below function to incorporate m and n.

Prerequisites : BigInteger in Java.

Java

```
 // Java code to generate random numbers
import java.math.BigInteger;
import java.util.Random;

public class RandomImp{

    public int random01Generator() {

        Random rand = new Random();
        return rand.nextInt(2);
    }

    // function will use the above
    // method and return numbers
    // between 0 and 6 inclusive.
    public void random06Generator(){

        Random rand = new Random();
        int val = 7;
        while (val >= 7) {

            String res = "";
            for (int i = 0; i < 3; i++)
                res +=
                 String.valueOf(random01Generator());

            BigInteger bg = new BigInteger(res,2);

            val = bg.intValue();
        }
        System.out.println(val);
    }

    // Driver Code
    public static void main(String[] args){
        RandomImp r = new RandomImp();
        r.random06Generator();
    }
}
```

Output :

```
3
2
4
1
```

This question is inspired from this stackoverflow link

Source

https://www.geeksforgeeks.org/implement-random-0-6-generator-using-the-given-random-0-1-generator/

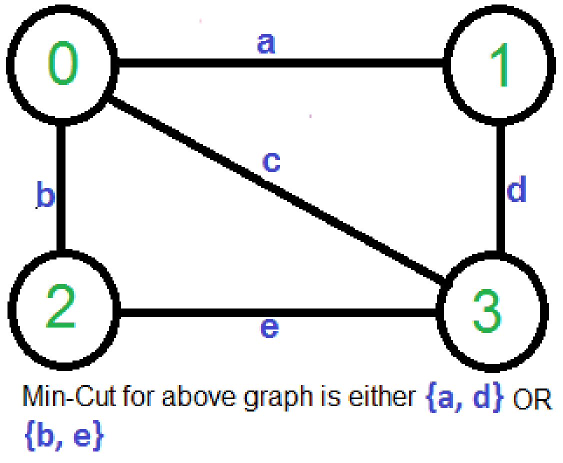
Chapter 13

Karger's algorithm for Minimum Cut | Set 1 (Introduction and Implementation)

Karger's algorithm for Minimum Cut | Set 1 (Introduction and Implementation) - GeeksforGeeks

Given an undirected and unweighted graph, find the smallest cut (smallest number of edges that disconnects the graph into two components).
The input graph may have parallel edges.

For example consider the following example, the smallest cut has 2 edges.



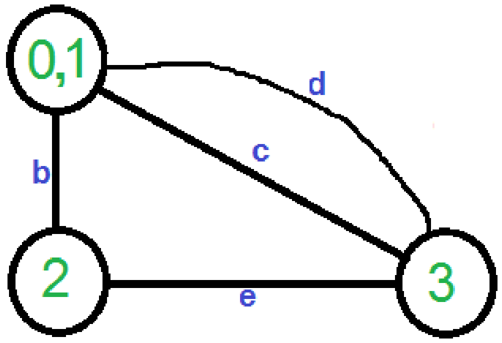
A Simple Solution use Max-Flow based s-t cut algorithm to find minimum cut. Consider every pair of vertices as source 's' and sink 't', and call minimum s-t cut algorithm to find the s-t cut. Return minimum of all s-t cuts. Best possible time complexity of this algorithm is $O(V^5)$ for a graph. [How? there are total possible V^2 pairs and s-t cut algorithm for one pair takes O(V*E) time and $E = O(V^2)$].

Below is simple Karger's Algorithm for this purpose. Below Karger's algorithm can be implemented in $O(E) = O(V^2)$ time.

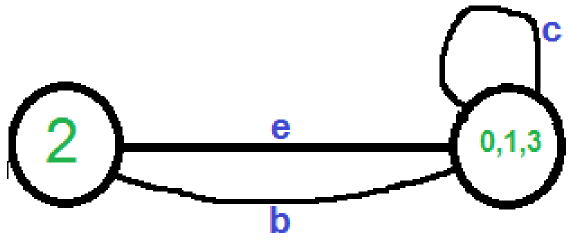
```
1)  Initialize contracted graph CG as copy of original graph
2)  While there are more than 2 vertices.
      a) Pick a random edge (u, v) in the contracted graph.
      b) Merge (or contract) u and v into a single vertex (update
         the contracted graph).
      c) Remove self-loops
3) Return cut represented by two vertices.
```

Let us understand above algorithm through the example given.

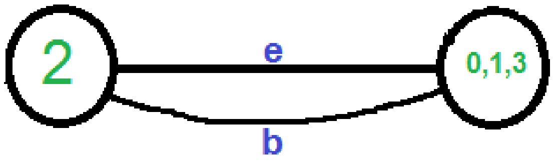
Let the first randomly picked vertex be '**a**' which connects vertices 0 and 1. We remove this edge and contract the graph (combine vertices 0 and 1). We get the following graph.



Let the next randomly picked edge be 'd'. We remove this edge and combine vertices (0,1) and 3.

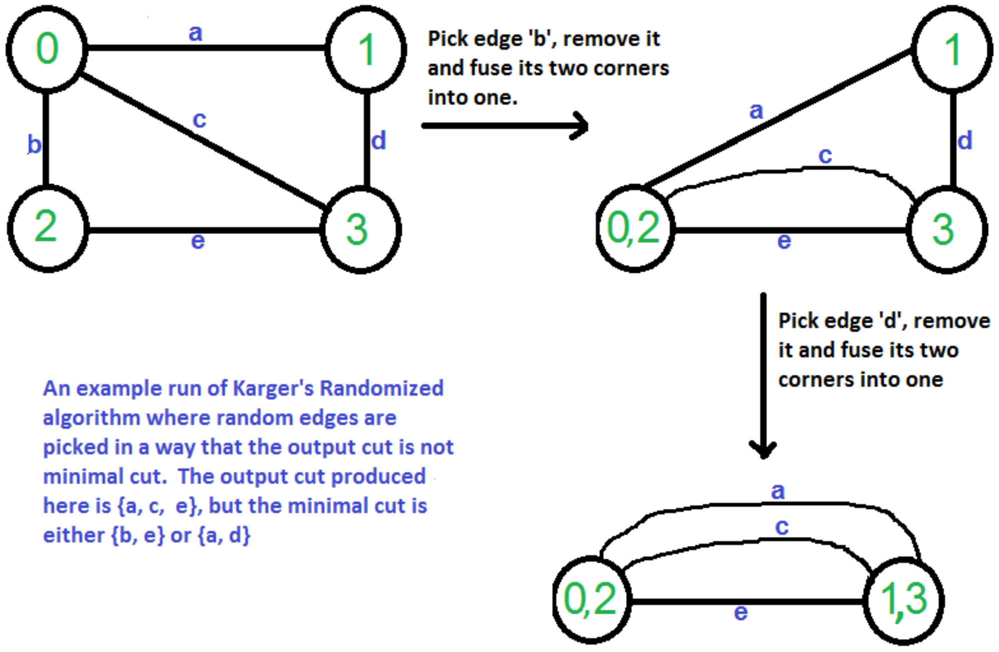


We need to remove self-loops in the graph. So we remove edge 'c'



Now graph has two vertices, so we stop. The number of edges in the resultant graph is the cut produced by Karger's algorithm.

Karger's algorithm is a Monte Carlo algorithm and cut produced by it may not be minimum. For example, the following diagram shows that a different order of picking random edges produces a min-cut of size 3.


0
1
2
3
a
b
c
d
e
Pick edge 'b', remove it and fuse its two corners into one.
0,2
Pick edge 'd', remove it and fuse its two corners into one
1,3
An example run of Karger's Randomized algorithm where random edges are picked in a way that the output cut is not minimal cut. The output cut produced here is {a, c, e}, but the minimal cut is either {b, e} or {a, d}

Below is C++ implementation of above algorithm. The input graph is represented as a collection of edges andunion-find data structure is used to keep track of components.

```
 // Karger's algorithm to find Minimum Cut in an
// undirected, unweighted and connected graph.
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// a structure to represent a unweighted edge in graph
struct Edge
{
    int src, dest;
};

// a structure to represent a connected, undirected
// and unweighted graph as a collection of edges.
struct Graph
{
    // V-> Number of vertices, E-> Number of edges
    int V, E;
    // graph is represented as an array of edges.
    // Since the graph is undirected, the edge
    // from src to dest is also edge from dest
    // to src. Both are counted as 1 edge here.
    Edge* edge;
};

// A structure to represent a subset for union-find
struct subset
{
    int parent;
    int rank;
};

// Function prototypes for union-find (These functions are defined
// after kargerMinCut() )
int find(struct subset subsets[], int i);
void Union(struct subset subsets[], int x, int y);

// A very basic implementation of Karger's randomized
// algorithm for finding the minimum cut. Please note
// that Karger's algorithm is a Monte Carlo Randomized algo
// and the cut returned by the algorithm may not be
// minimum always
int kargerMinCut(struct Graph* graph)
{
    // Get data of given graph
    int V = graph->V, E = graph->E;
    Edge *edge = graph->edge;

    // Allocate memory for creating V subsets.
    struct subset *subsets = new subset[V];

    // Create V subsets with single elements
    for (int v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    // Initially there are V vertices in
    // contracted graph
    int vertices = V;

    // Keep contracting vertices until there are
    // 2 vertices.
    while (vertices > 2)
    {
       // Pick a random edge
       int i = rand() % E;

       // Find vertices (or sets) of two corners
       // of current edge
       int subset1 = find(subsets, edge[i].src);
       int subset2 = find(subsets, edge[i].dest);

       // If two corners belong to same subset,
       // then no point considering this edge
       if (subset1 == subset2)
         continue;

       // Else contract the edge (or combine the
       // corners of edge into one vertex)
       else
       {
          printf("Contracting edge %d-%d\n",
                 edge[i].src, edge[i].dest);
          vertices--;
          Union(subsets, subset1, subset2);
       }
    }

    // Now we have two vertices (or subsets) left in
    // the contracted graph, so count the edges between
    // two components and return the count.
    int cutedges = 0;
    for (int i=0; i<E; i++)
    {
        int subset1 = find(subsets, edge[i].src);
        int subset2 = find(subsets, edge[i].dest);
        if (subset1 != subset2)
          cutedges++;
    }

    return cutedges;
}

// A utility function to find set of an element i
// (uses path compression technique)
int find(struct subset subsets[], int i)
{
    // find root and make root as parent of i
    // (path compression)
    if (subsets[i].parent != i)
      subsets[i].parent =
             find(subsets, subsets[i].parent);

    return subsets[i].parent;
}

// A function that does union of two sets of x and y
// (uses union by rank)
void Union(struct subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    // Attach smaller rank tree under root of high
    // rank tree (Union by Rank)
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;

    // If ranks are same, then make one as root and
    // increment its rank by one
    else
    {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

// Creates a graph with V vertices and E edges
struct Graph* createGraph(int V, int E)
{
    Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;
    graph->edge = new Edge[E];
    return graph;
}

// Driver program to test above functions
int main()
{
    /* Let us create following unweighted graph
        0------1
        | \    |
        |   \  |
        |     \|
        2------3   */
    int V = 4;  // Number of vertices in graph
    int E = 5;  // Number of edges in graph
    struct Graph* graph = createGraph(V, E);

    // add edge 0-1
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;

    // add edge 0-2
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;

    // add edge 0-3
    graph->edge[2].src = 0;
    graph->edge[2].dest = 3;

    // add edge 1-3
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;

    // add edge 2-3
    graph->edge[4].src = 2;
    graph->edge[4].dest = 3;

    // Use a different seed value for every run.
    srand(time(NULL));

    printf("\nCut found by Karger's randomized algo is %d\n",
           kargerMinCut(graph));

    return 0;
}
```

Output:

```
Contracting edge 0-2
Contracting edge 0-3

Cut found by Karger's randomized algo is 2
```

Note that the above program is based on outcome of a random function and may produce different output.

In this post, we have discussed simple Karger's algorithm and have seen that the algorithm doesn't always produce min-cut. The above algorithm produces min-cut with probability greater or equal to that $1/(n^2)$. See next post on Analysis and Applications of Karger's Algortihm, applications, proof of this probability and improvements are discussed.

References:
http://en.wikipedia.org/wiki/Karger%27s_algorithm

https://www.youtube.com/watch?v=P0l8jMDQTEQ

https://www.cs.princeton.edu/courses/archive/fall13/cos521/lecnotes/lec2final.pdf

http://web.stanford.edu/class/archive/cs/cs161/cs161.1138/lectures/11/Small11.pdf

Source

https://www.geeksforgeeks.org/kargers-algorithm-for-minimum-cut-set-1-introduction-and-implementation/

Chapter 14

Karger's algorithm for Minimum Cut | Set 2 (Analysis and Applications)

Karger's algorithm for Minimum Cut | Set 2 (Analysis and Applications) - GeeksforGeeks

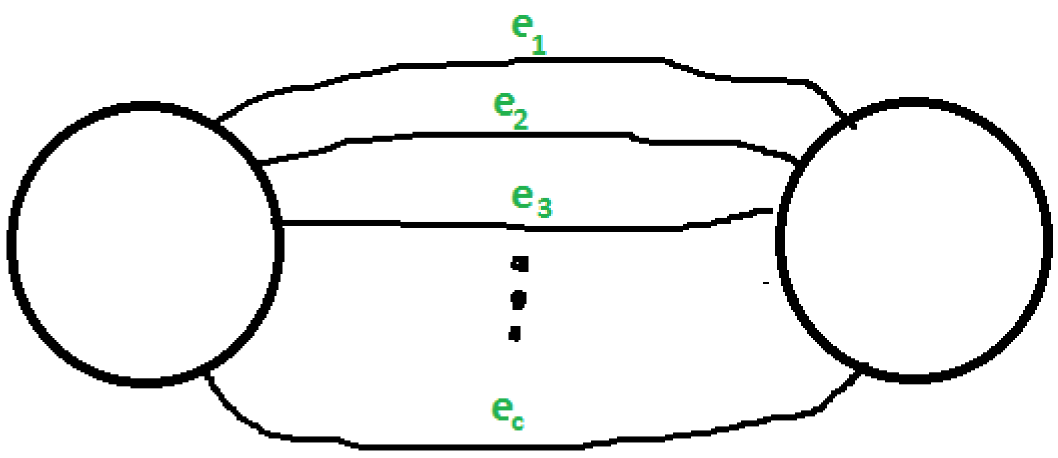
We have introduced and discussed below Karger's algorithm in set 1.

```
1)  Initialize contracted graph CG as copy of original graph
2)  While there are more than 2 vertices.
      a) Pick a random edge (u, v) in the contracted graph.
      b) Merge (or contract) u and v into a single vertex (update
         the contracted graph).
      c) Remove self-loops
3) Return cut represented by two vertices.
```

As discussed in the previous post,Karger's algorithmdoesn't always find min cut. In this post, the probability of finding min-cut is discussed.

Probability that the cut produced by Karger's Algorithm is Min-Cut is greater than or equal to 1/(n^2)

Proof:
Let there be a unique Min-Cut of given graph and let there be C edges in the Min-Cut and the edges be {e_1, e_2, e_3, .. e_c}. The Karger's algorithm would produce this Min-Cut if and only if none of the edges in set {e_1, e_2, e_3, .. e_c} is removed in iterations in the main while loop of above algorithm.



```
c is number of edges in min-cut
m is total number of edges
n is total number of vertices

S1 = Event that one of the edges in {e1, e2,
     e3, .. ec} is chosen in 1st iteration.
S2 = Event that one of the edges in {e1, e2,
     e3, .. ec} is chosen in 2nd iteration.
S3 = Event that one of the edges in {e1, e2,
     e3, .. ec} is chosen in 3rd iteration.

..................
..................

The cut produced by Karger's algorithm would be a min-cut if none of the above
events happen.

So the required probability is P[S1'   S2'   S3'    ............]
```

Probability that a min-cut edge is chosen in first iteration:

```
Let us calculate  P[S1']
P[S1]  = c/m
P[S1'] = (1 - c/m)

Above value is in terms of m (or edges), let us convert
it in terms of n (or vertices) using below 2 facts..

1) Since size of min-cut is c, degree of all vertices must be greater
than or equal to c.
```

```
2) As per Handshaking Lemma, sum of degrees of all vertices = 2m

From above two facts, we can conclude below.
  n*c <= 2m
    m >= nc/2

  P[S1] <= c / (cn/2)
        <= 2/n

  P[S1] <= c / (cn/2)
        <= 2/n

  P[S1'] >= (1-2/n) ------------(1)
```

Probability that a min-cut edge is chosen in second iteration:

```
P[S1'   S2'] = P[S2' | S1' ] * P[S1']

In the above expression, we know value of P[S1'] >= (1-2/n)

P[S2' | S1'] is conditional probability that is, a min cut is
not chosen in second iteration given that it is not chosen in first iteration

Since there are total (n-1) edges left now and number of cut edges is still c,
we can replace n by n-1 in inequality (1).  So we get.
  P[S2' | S1' ] >= (1 - 2/(n-1))

  P[S1'   S2'] >= (1-2/n) x (1-2/(n-1))
```

Probability that a min-cut edge is chosen in all iterations:

```
P[S1'   S2'  S3'  ..........  Sn-2']

>= [1 - 2/n] * [1 - 2/(n-1)] * [1 - 2/(n-2)] * [1 - 2/(n-3)] *...
                              ... * [1 - 2/(n - (n-4)] * [1 - 2/(n - (n-3)]

>= [(n-2)/n] * [(n-3)/(n-1)] * [(n-4)/(n-2)] * .... 2/4 * 2/3

>= 2/(n * (n-1))
>= 1/n2
```

How to increase probability of success?

The above probability of success of basic algorithm is very less. For example, for a graph with 10 nodes, the probability of finding the min-cut is greater than or equal to 1/100. The probability can be increased by repeated runs of basic algorithm and return minimum of all cuts found.

Applications:
1) In war situation, a party would be interested in finding minimum number of links that break communication network of enemy.

2) The min-cut problem can be used to study reliability of a network (smallest number of edges that can fail).

3) Study of network optimization (find a maximum flow).

4) Clustering problems (edges like associations rules) Matching problems (an NCalgorithm for min-cut in directed graphs would result in an NCalgorithm for maximum matching in bipartite graphs)

5) Matching problems (an NC algorithm for min-cut in directed graphs would result in an NC algorithm for maximum matching in bipartite graphs)

Sources:
https://www.youtube.com/watch?v=-UuivvyHPas
http://disi.unal.edu.co/~gjhernandezp/psc/lectures/02/MinCut.pdf

Source

https://www.geeksforgeeks.org/kargers-algorithm-for-minimum-cut-set-2-analysis-and-applications/

Chapter 15

K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time)

K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time) - GeeksforGeeks

We recommend reading following post as a prerequisite of this post.

K'th Smallest/Largest Element in Unsorted Array | Set 1

Given an array and a number k where k is smaller than the size of the array, we need to find the k'th smallest element in the given array. It is given that all array elements are distinct.

Examples:

```
Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 3
Output: 7

Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 4
Output: 10
```

We have discussed three different solutions here.

In this post method 4 is discussed which is mainly an extension of method 3 (QuickSelect) discussed in the previouspost. The idea is to randomly pick a pivot element. To implement randomized partition, we use a random function, rand() to generate index between l and r, swap the element at randomly generated index with the last element, and finally call the standard partition process which uses last element as pivot.

Following is an implementation of above Randomized QuickSelect.

C/C++

```
 // C++ implementation of randomized quickSelect
#include<iostream>
#include<climits>
#include<cstdlib>
using namespace std;

int randomPartition(int arr[], int l, int r);

// This function returns k'th smallest element in arr[l..r] using
// QuickSort based method. ASSUMPTION: ELEMENTS IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k)
{
    // If k is smaller than number of elements in array
    if (k > 0 && k <= r - l + 1)
    {
        // Partition the array around a random element and
        // get position of pivot element in sorted array
        int pos = randomPartition(arr, l, r);

        // If position is same as k
        if (pos-l == k-1)
            return arr[pos];
        if (pos-l > k-1)  // If position is more, recur for left subarray
            return kthSmallest(arr, l, pos-1, k);

        // Else recur for right subarray
        return kthSmallest(arr, pos+1, r, k-pos+l-1);
    }

    // If k is more than the number of elements in the array
    return INT_MAX;
}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Standard partition process of QuickSort().  It considers the last
// element as pivot and moves all smaller element to left of it and
// greater elements to right. This function is used by randomPartition()
int partition(int arr[], int l, int r)
{
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++)
    {
        if (arr[j] <= x)
        {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[r]);
    return i;
}

// Picks a random pivot element between l and r and partitions
// arr[l..r] around the randomly picked element using partition()
int randomPartition(int arr[], int l, int r)
{
    int n = r-l+1;
    int pivot = rand() % n;
    swap(&arr[l + pivot], &arr[r]);
    return partition(arr, l, r);
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 4, 19, 26};
    int n = sizeof(arr)/sizeof(arr[0]), k = 3;
    cout << "K'th smallest element is " << kthSmallest(arr, 0, n-1, k);
    return 0;
}
```

Java

```
 // Java program to find k'th smallest element in expected
// linear time
class KthSmallst
{
    // This function returns k'th smallest element in arr[l..r]
    // using QuickSort based method.  ASSUMPTION: ALL ELEMENTS
    // IN ARR[] ARE DISTINCT
    int kthSmallest(int arr[], int l, int r, int k)
    {
        // If k is smaller than number of elements in array
        if (k > 0 && k <= r - l + 1)
        {
            // Partition the array around a random element and
            // get position of pivot element in sorted array
            int pos = randomPartition(arr, l, r);

            // If position is same as k
            if (pos-l == k-1)
                return arr[pos];

            // If position is more, recur for left subarray
            if (pos-l > k-1)
                return kthSmallest(arr, l, pos-1, k);

            // Else recur for right subarray
            return kthSmallest(arr, pos+1, r, k-pos+l-1);
        }

        // If k is more than number of elements in array
        return Integer.MAX_VALUE;
    }

    // Utility method to swap arr[i] and arr[j]
    void swap(int arr[], int i, int j)
    {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    // Standard partition process of QuickSort().  It considers
    // the last element as pivot and moves all smaller element
    // to left of it and greater elements to right. This function
    // is used by randomPartition()
    int partition(int arr[], int l, int r)
    {
        int x = arr[r], i = l;
        for (int j = l; j <= r - 1; j++)
        {
            if (arr[j] <= x)
            {
                swap(arr, i, j);
                i++;
            }
        }
        swap(arr, i, r);
        return i;
    }

    // Picks a random pivot element between l and r and
    // partitions arr[l..r] arount the randomly picked
    // element using partition()
    int randomPartition(int arr[], int l, int r)
    {
        int n = r-l+1;
        int pivot = (int)(Math.random()) * (n-1);
        swap(arr, l + pivot, r);
        return partition(arr, l, r);
    }

    // Driver method to test above
    public static void main(String args[])
    {
        KthSmallst ob = new KthSmallst();
        int arr[] = {12, 3, 5, 7, 4, 19, 26};
        int n = arr.length,k = 3;
        System.out.println("K'th smallest element is "+
                           ob.kthSmallest(arr, 0, n-1, k));
    }
}
/*This code is contributed by Rajat Mishra*/
```

Output:

```
K'th smallest element is 5
```

Time Complexity:
The worst case time complexity of the above solution is still O(n^2). In worst case, the randomized function may always pick a corner element. The expected time complexity of above randomized QuickSelect is ?(n), see CLRS book or MIT video lecture for proof. The assumption in the analysis is, random number generator is equally likely to generate any number in the input range.

Sources:
MIT Video Lecture on Order Statistics, Median
Introduction to Algorithms by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L.

This article is contributed by **Shivam**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : VyshakhNambiar

Source

https://www.geeksforgeeks.org/kth-smallestlargest-element-unsorted-array-set-2-expected-linear-time/

Chapter 16

Linearity of Expectation

Linearity of Expectation - GeeksforGeeks

Prerequisite : Random Variable

This post is about mathematical concepts like expectation, linearity of expectation. It covers one of the required topics to understand Randomized Algorithms.

Let us consider the following simple problem.

Given a fair dice with 6 faces, the dice is thrown n times, find expected value of sum of all results.

For example, if n = 2, there are total 36 possible outcomes.

```
(1, 1), (1, 2), .... (1, 6)
(2, 1), (2, 2), .... (2, 6)
................
................
(6, 1), (6, 2), ..... (6, 6)
```

Expected value of a discrete random variable is R defined as following. Suppose R can take value r_1 with probability p_1, value r_2 with probability p_2, and so on, up to value r_k with probability p_k. Then the expectation of this random variable R is defined as

```
    E[R] = r1*p1 + r2*p2 + ... rk*pk
```

Let us calculate expected value for the above example.

```
Expected Value of sum = 2*1/36 + 3*1/6 + .... + 7*1/36 +
of two dice throws      3*1/36 + 4*1/6 + .... + 8*1/36 +
                        ........................
                        .........................
                        7*1/36 + 8*1/6 + .... + 12*1/36

                      =  7
```

The above way to solve the problem becomes difficult when there are more dice throws.

If we know about linearity of expectation, then we can quickly solve the above problem for any number of throws.

Linearity of Expectation: Let R_1 and R_2 be two discrete random variables on some probability space, then

```
E[R1 + R2] = E[R1] + E[R2]
```

Using the above formula, we can quickly solve the dice problem.

```
Expected Value of sum of 2 dice throws = 2*(Expected value of one dice throw)
                                       = 2*(1/6 + 2/6 + .... 6/6)
                                       = 2*7/2
                                       = 7

Expected value of sum for n dice throws is = n * 7/2 = 3.5 * n
```

Some interesting facts about Linearly of Expectation:
1) Linearity of expectation holds for both dependent and independent events. On the other hand the rule $E[R_1R_2] = E[R_1]*E[R_2]$ is true only for independent events.

2) Linearity of expectation holds for any number of random variables on some probability space. Let R_1, R_2, R_3, … R_k be k random variables, then
$E[R_1 + R_2 + R_3 + … + R_k] = E[R_1] + E[R_2] + E[R_3] + … + E[R_k]$

Another example that can be easily solved with linearity of expectation:
Hat-Check Problem: Let there be group of n men where every man has one hat. The hats are redistributed and every man gets a random hat back. What is the expected number of men that get their original hat back.

Solution: Let R_i be a random variable, the value of random variable is 1 if i'th man gets the same hat back, otherwise 0.

```
So the expected number of men to get the right hat back is
  = E[R1] + E[R2]  +  .. + E[Rn]
  = P(R1 = 1) + P(R2 = 1) + .... + P(Rn = 1)
  [Here P(Ri = 1)  indicates probability that Ri is 1]
  = 1/n + 1/n + ... + 1/n
  = 1
```

So on average 1 person gets the right hat back.

Exercise:
1) Given a fair coin, what is the expected number of heads when coin is tossed n times.

2) Balls and Bins: Suppose we have m balls, labeled i = 1, … , m and n bins, labeled j = 1, .. ,n. Each ball is thrown into one of the bin independently and uniformly at random.
   a) What is the expected number of balls in every bin
   b) What is the expected number of empty bins.

3) Coupon Collector: Suppose there are n types of coupons in a lottery and each lot contains one coupon (with probability 1 = n each). How many lots have to be bought (in expectation) until we have at least one coupon of each type.

See following for solution of Coupon Collector.
Expected Number of Trials until Success

Linearity of expectation is useful in algorithms. For example, expected time complexity of random algorithms like randomized quick sort is evaluated using linearity of expectation (See thisfor reference).

References:
http://www.cse.iitd.ac.in/~mohanty/col106/Resources/linearity_expectation.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-video-lectures/lecture-22-expectation-i/

This article is contributed by **Shubham Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/linearity-of-expectation/

Chapter 17

Load Balancing on Servers (Randomized Algorithm)

Load Balancing on Servers (Randomized Algorithm) - GeeksforGeeks

Consider a high traffic website that receives millions of requests (of different types) per five minutes, the site has k (for example n = 1000) servers to process the requests. How should the load be balanced among servers?

The solutions that we generally think of are
a) Round Robin
b) Assign new request to a server that has minimum load.

Both of the above approaches look good, but they require additional state information to be maintained for load balancing. Following is a simple approach that works better than above approaches.

```
Do following whenever a new request comes in,
        Pick a random server and assign the request to a random server
```

The above approach is simpler, lightweight and surprisingly effective. This approach doesn't calculate existing load on server and doesn't need time management.

Analysis of above Random Approach
Let us analyze the average load on a server when above approach of randomly picking server is used.

Let there be k request (or jobs) J_1, J_2, … J_k

Let there be n servers be S_1, S_2, … S_k.

Let time taken by i'th job be T_i

Let R_{ij} be load on server S_i from Job J_j.

R_{ij} is T_j if j'th job (or J_j) is assigned to S_i, otherwise 0. Therefore, value of R_{ij} is T_j with probability 1/n and value is 0 with probability (1-1/n)

Let R_i be load on i'th server

```
Average Load on i'th server 'Ex(Ri)'
                             [Applying Linearity of Expectation]
                           =
                           =
                           = (Total Load)/n
```

So average load on a server is total load divided by n which is a perfect result.

What is the possibility of deviation from average (A particular server gets too much load)?
The average load from above random assignment approach looks good, but there may be possibility that a particular server becomes too loaded (even if the average is ok).
It turns out that the probability of deviation from average is also very low (can be proved using Chernoff bound). Readers can refer below reference links for proves of deviations. For example, in MIT video lecture, it is shown that if there are 2500 requests per unit time and there are 10 servers, then the probability that any particular server gets 10% more load is at most 1/16000. Similar results are shown at the end of second reference also.

So above simple load balancing scheme works perfect. In-fact this scheme is used in load balancers.

References:
http://www.cs.princeton.edu/courses/archive/fall09/cos521/Handouts/probabilityandcomputing.pdf

MIT Video Lecture

This article is contributed by **Shivam**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/load-balancing-on-servers-random-algorithm/

Chapter 18

Make a fair coin from a biased coin

Make a fair coin from a biased coin - GeeksforGeeks

You are given a function foo() that represents a biased coin. When foo() is called, it returns 0 with 60% probability, and 1 with 40% probability. Write a new function that returns 0 and 1 with 50% probability each. Your function should use only foo(), no other library method.

Solution:
We know foo() returns 0 with 60% probability. How can we ensure that 0 and 1 are returned with 50% probability?
The solution is similar to thispost. If we can somehow get two cases with equal probability, then we are done. We call foo() two times. Both calls will return 0 with 60% probability. So the two pairs (0, 1) and (1, 0) will be generated with equal probability from two calls of foo(). Let us see how.

(0, 1): The probability to get 0 followed by 1 from two calls of foo() = 0.6 * 0.4 = 0.24
(1, 0): The probability to get 1 followed by 0 from two calls of foo() = 0.4 * 0.6 = 0.24

So the two cases appear with equal probability. The idea is to return consider only the above two cases, return 0 in one case, return 1 in other case. For other cases [(0, 0) and (1, 1)], recur until you end up in any of the above two cases.

The below program depicts how we can use foo() to return 0 and 1 with equal probability.

```
 #include <stdio.h>

int foo() // given method that returns 0 with 60% probability and 1 with 40%
{
    // some code here
}

// returns both 0 and 1 with 50% probability
```

```
int my_fun()
{
    int val1 = foo();
    int val2 = foo();
    if (val1 == 0 && val2 == 1)
        return 0;   // Will reach here with 0.24 probability
    if (val1 == 1 && val2 == 0)
        return 1;   // // Will reach here with 0.24 probability
    return my_fun();  // will reach here with (1 - 0.24 - 0.24) probability
}

int main()
{
    printf ("%d ", my_fun());
    return 0;
}
```

References:
http://en.wikipedia.org/wiki/Fair_coin#Fair_results_from_a_biased_coin

Source

https://www.geeksforgeeks.org/print-0-and-1-with-50-probability/

Chapter 19

Mid-Square hashing

Mid-Square hashing - GeeksforGeeks

Mid-Square hashing is a hashing technique in which unique keys are generated. In this technique, a seed value is taken and it is squared. Then, some digits from the middle are extracted. These extracted digits form a number which is taken as the new seed. This technique can generate keys with high randomness if a big enough seed value is taken. However, it has a limitation. As the seed is squared, if a 6-digit number is taken, then the square will have 12-digits. This exceeds the range of int data type. So, overflow must be taken care of. In case of overflow, use long long int data type or use string as multiplication if overflow still occurs. The chances of a collision in mid-square hashing are low, not obsolete. So, in the chances, if a collision occurs, it is handled using some hash map.

Example:

> Suppose a 4-digit seed is taken. seed = 4765
> Hence, square of seed is = 4765 * 4765 = 22705225
> Now, from this 8-digit number, any four digits are extracted (Say, the middle four).
> So, the new seed value becomes seed = 7052
> Now, square of this new seed is = 7052 * 7052 = 49730704
> Again, the same set of 4-digits is extracted.
> So, the new seed value becomes seed = 7307
> .
> .
> .
> .
> This process is repeated as many times as a key is required.

Mid square technique takes a certain number of digits from the square of a number. This number extracted is a pseudo-random number, which can be used as a key for hashing.

Algorithm:

1. Choose a seed value. This is an important step as for same seed value, the same sequence of random numbers is generated.
2. Take the square of the seed value and update seed by a certain number of digits that occur in the square. **Note**: The larger is the number of digits, larger will be the randomness.
3. Return the key.

Below is the implementation of above algorithm:

```
 // C++ program to illustrate the
// mid-square hashing technique
#include <ctime>
#include <iostream>

using namespace std;

// Returns a seed value based on current system time.
long long int newTime()
{

    // Acquiring number of seconds
    // passed from Jan 1, 1970.
    time_t t = time(NULL);

    // Converting the time to year, month,
    // day, hour, minute, and second.
    struct tm* tm = localtime(&t);
    long long int x;

    // Applying a certain logic to get
    // required number of digits. It may vary.
    x = (tm->tm_hour) * 10000000 + (tm->tm_min) * 100000
        + (tm->tm_sec) * 1000 + (tm->tm_mday) * 10 + (tm->tm_year);

    // Return the calculated number.
    return x;
}

// Returns a random 8-digit key.
long int getKey()
{

    // Taking the key from system time.
    // returns a  8-digit seed value.
    static long long int key = newTime();

    // Squaring the key.
    key = key * key;
    // Extracting required number of digits ( here, 8 ).
    if (key < 1000000000000000) {
        key = key / 10000;
        key = key % 100000000;
    }
    else {
        key = key / 10000;
        key = key % 100000000;
    }

    // Returning the key value.
    return key;
}

// Driver Code
int main()
{
    // get the first key
    std::cout << getKey() << endl;

    // get the second key
    std::cout << getKey() << endl;
    return 0;
}
```

Output:

```
56002270
25424515
```

Note: The output will change according to the date and time.

Source

https://www.geeksforgeeks.org/mid-square-hashing/

Chapter 20

Operations on Sparse Matrices

Operations on Sparse Matrices - GeeksforGeeks

Given two sparse matrices (Sparse Matrix and its representations | Set 1 (Using Arrays and Linked Lists)), perform operations such as add, multiply or transpose of the matrices in their sparse form itself. The result should consist of three sparse matrices, one obtained by adding the two input matrices, one by multiplying the two matrices and one obtained by transpose of the first matrix.

Example: Note that other entries of matrices will be zero as matrices are sparse.

```
Input :

Matrix 1: (4x4)
Row Column Value
1   2       10
1   4       12
3   3       5
4   1       15
4   2       12

Matrix 2: (4X4)
Row Column Value
1   3       8
2   4       23
3   3       9
4   1       20
4   2       25

Output :

Result of Addition: (4x4)
Row Column Value
```

```
1   2      10
1   3      8
1   4      12
2   4      23
3   3      14
4   1      35
4   2      37

Result of Multiplication: (4x4)
Row Column Value
1   1      240
1   2      300
1   4      230
3   3      45
4   3      120
4   4      276

Result of transpose on the first matrix: (4x4)
Row Column Value
1   4      15
2   1      10
2   4      12
3   3      5
4   1      12
```

The sparse matrix used anywhere in the program is sorted according to its row values. Two elements with the same row values are further sorted according to their column values.

Now to **Add** the matrices, we simply traverse through both matrices element by element and insert the smaller element (one with smaller row and col value) into the resultant matrix. If we come across an element with the same row and column value, we simply add their values and insert the added data into the resultant matrix.

To **Transpose** a matrix, we can simply change every column value to the row value and vice-versa, however, in this case, the resultant matrix won't be sorted as we require. Hence, we initially determine the number of elements less than the current element's column being inserted in order to get the exact index of the resultant matrix where the current element should be placed. This is done by maintaining an array index[] whose ith value indicates the number of elements in the matrix less than the column i.

To **Multiply** the matrices, we first calculate transpose of the second matrix to simply our comparisons and maintain the sorted order. So, the resultant matrix is obtained by traversing through the entire length of both matrices and summing the appropriate multiplied values.
Any row value equal to x in the first matrix and row value equal to y in the second matrix (transposed one) will contribute towards result[x][y]. This is obtained by multiplying all such elements having col value in both matrices and adding only those with the row as x in first matrix and row as y in the second transposed matrix to get the result[x][y].

For example: Consider 2 matrices:

```
Row Col Val      Row Col Val
1   2   10       1   1   2
1   3   12       1   2   5
2   1   1        2   2   1
2   3   2        3   1   8
```

The resulting matrix after multiplication will be obtained as follows:

```
Transpose of second matrix:

Row Col Val      Row Col Val
1   2   10       1   1   2
1   3   12       1   3   8
2   1   1        2   1   5
2   3   2        2   2   1

Summation of multiplied values:

result[1][1] = A[1][3]*B[1][3] = 12*8 = 96
result[1][2] = A[1][2]*B[2][2] = 10*1 = 10
result[2][1] = A[2][1]*B[1][1] + A[2][3]*B[1][3] = 2*1 + 2*8 = 18
result[2][2] = A[2][1]*B[2][1] = 1*5 = 5

Any other element cannot be obtained
by any combination of row in
Matrix A and Row in Matrix B.

Hence the final resultant matrix will be:

Row Col Val
1   1   96
1   2   10
2   1   18
2   2   5
```

Following is the Java implementation of above approach:

```
 // Java code to perform add,
// multiply and transpose on sparse matrices

public class sparse_matrix {

    // Maximum number of elements in matrix
```

```
int MAX = 100;

// Array representation
// of sparse matrix
//[][0] represents row
//[][1] represents col
//[][2] represents value
int data[][] = new int[MAX][3];

// dimensions of matrix
int row, col;

// total number of elements in matrix
int len;

public sparse_matrix(int r, int c)
{

    // initialize row
    row = r;

    // initialize col
    col = c;

    // intialize length to 0
    len = 0;
}

// insert elements into sparse matrix
public void insert(int r, int c, int val)
{

    // invalid entry
    if (r > row || c > col) {
        System.out.println("Wrong entry");
    }

    else {

        // insert row value
        data[len][0] = r;

        // insert col value
        data[len][1] = c;

        // insert element's value
        data[len][2] = val;
```

```
            // increment number of data in matrix
            len++;
        }
    }

    public void add(sparse_matrix b)
    {

        // if matrices don't have same dimensions
        if (row != b.row || col != b.col) {
            System.out.println("Matrices can't be added");
        }

        else {

            int apos = 0, bpos = 0;
            sparse_matrix result = new sparse_matrix(row, col);

            while (apos < len && bpos < b.len) {

                // if b's row and col is smaller
                if (data[apos][0] > b.data[bpos][0] ||
                  (data[apos][0] == b.data[bpos][0] &&
                   data[apos][1] > b.data[bpos][1]))

                {

                    // insert smaller value into result
                    result.insert(b.data[bpos][0],
                                  b.data[bpos][1],
                                  b.data[bpos][2]);

                    bpos++;
                }

                // if a's row and col is smaller
                else if (data[apos][0] < b.data[bpos][0] ||
                (data[apos][0] == b.data[bpos][0] &&
                  data[apos][1] < b.data[bpos][1]))

                {

                    // insert smaller value into result
                    result.insert(data[apos][0],
                                  data[apos][1],
                                  data[apos][2]);

                    apos++;
                }

                else {

                    // add the values as row and col is same
                    int addedval = data[apos][2] + b.data[bpos][2];

                    if (addedval != 0)
                        result.insert(data[apos][0],
                                      data[apos][1],
                                      addedval);
                    // then insert
                    apos++;
                    bpos++;
                }
            }

            // insert remaining elements
            while (apos < len)
                result.insert(data[apos][0],
                              data[apos][1],
                              data[apos++][2]);

            while (bpos < b.len)
                result.insert(b.data[bpos][0],
                              b.data[bpos][1],
                              b.data[bpos++][2]);

            // print result
            result.print();
        }
    }

    public sparse_matrix transpose()
    {

        // new matrix with inversed row X col
        sparse_matrix result = new sparse_matrix(col, row);

        // same number of elements
        result.len = len;

        // to count number of elements in each column
        int count[] = new int[col + 1];

        // initialize all to 0
        for (int i = 1; i <= col; i++)
            count[i] = 0;
        for (int i = 0; i < len; i++)
            count[data[i][1]]++;

        int[] index = new int[col + 1];

        // to count number of elements having col smaller
        // than particular i

        // as there is no col with value < 1
        index[1] = 0;

        // initialize rest of the indices
        for (int i = 2; i <= col; i++)

            index[i] = index[i - 1] + count[i - 1];

        for (int i = 0; i < len; i++) {

            // insert a data at rpos and increment its value
            int rpos = index[data[i][1]]++;

            // transpose row=col
            result.data[rpos][0] = data[i][1];

            // transpose col=row
            result.data[rpos][1] = data[i][0];

            // same value
            result.data[rpos][2] = data[i][2];
        }

        // the above method ensures
        // sorting of transpose matrix
        // according to row-col value
        return result;
    }

    public void multiply(sparse_matrix b)
    {

        if (col != b.row) {

            // Invalid muliplication
            System.out.println("Can't multiply, "
                               + "Invalid dimensions");

            return;
```

```
}

// transpose b to compare row
// and col values and to add them at the end
b = b.transpose();
int apos, bpos;

// result matrix of dimension row X b.col
// however b has been transposed, hence row X b.row
sparse_matrix result = new sparse_matrix(row, b.row);

// iterate over all elements of A
for (apos = 0; apos < len;) {

    // current row of result matrix
    int r = data[apos][0];

    // iterate over all elements of B
    for (bpos = 0; bpos < b.len;) {

        // current column of result matrix
        // data[][0] used as b is transposed
        int c = b.data[bpos][0];

        // temporary pointers created to add all
        // multiplied values to obtain current
        // element of result matrix
        int tempa = apos;
        int tempb = bpos;

        int sum = 0;

        // iterate over all elements with
        // same row and col value
        // to calculate result[r]
        while (tempa < len && data[tempa][0] == r
               && tempb < b.len && b.data[tempb][0] == c) {

            if (data[tempa][1] < b.data[tempb][1])

                // skip a
                tempa++;

            else if (data[tempa][1] > b.data[tempb][1])

                // skip b
                tempb++;
            else
```

```
                    // same col, so multiply and increment
                    sum += data[tempa++][2] * b.data[tempb++][2];
                }

                // insert sum obtained in result[r]
                // if its not equal to 0
                if (sum != 0)
                    result.insert(r, c, sum);

                while (bpos < b.len && b.data[bpos][0] == c)

                    // jump to next column
                    bpos++;
            }

            while (apos < len && data[apos][0] == r)

                // jump to next row
                apos++;
        }

        result.print();
    }

    // printing matrix
    public void print()
    {
        System.out.println("Dimension: " + row + "x" + col);
        System.out.println("Sparse Matrix: \nRow Column Value");

        for (int i = 0; i < len; i++) {

            System.out.println(data[i][0] + " "
                             + data[i][1] + " " + data[i][2]);
        }
    }

    public static void main(String args[])
    {

        // create two sparse matrices and insert values
        sparse_matrix a = new sparse_matrix(4, 4);
        sparse_matrix b = new sparse_matrix(4, 4);

        a.insert(1, 2, 10);
        a.insert(1, 4, 12);
        a.insert(3, 3, 5);
        a.insert(4, 1, 15);
        a.insert(4, 2, 12);
        b.insert(1, 3, 8);
        b.insert(2, 4, 23);
        b.insert(3, 3, 9);
        b.insert(4, 1, 20);
        b.insert(4, 2, 25);

        // Output result
        System.out.println("Addition: ");
        a.add(b);
        System.out.println("\nMultiplication: ");
        a.multiply(b);
        System.out.println("\nTranspose: ");
        sparse_matrix atranspose = a.transpose();
        atranspose.print();
    }
}

// This code is contributed by Sudarshan Khasnis
```

Output:

```
Addition:
Dimension: 4x4
Sparse Matrix:
Row Column Value
1 2 10
1 3 8
1 4 12
2 4 23
3 3 14
4 1 35
4 2 37

Multiplication:
Dimension: 4x4
Sparse Matrix:
Row Column Value
1 1 240
1 2 300
1 4 230
3 3 45
4 3 120
4 4 276

Transpose:
```

```
Dimension: 4x4
Sparse Matrix:
Row Column Value
1 4 15
2 1 10
2 4 12
3 3 5
4 1 12
```

Worst case time complexity: Addition operation traverses the matrices linearly, hence, has a time complexity of O(n), where n is the number of non-zero elements in the larger matrix amongst the two. Transpose has a time complexity of O(n+m), where n is the number of columns and m is the number of non-zero elements in the matrix. Multiplication, however, has a time complexity of O(x*n + y*m), where (x, m) is number of columns and terms in the second matrix; and (y, n) is number of rows and terms in the first matrix.

Source

https://www.geeksforgeeks.org/operations-sparse-matrices/

Chapter 21

Primality Test | Set 2 (Fermat Method)

Primality Test | Set 2 (Fermat Method) - GeeksforGeeks

Given a number n, check if it is prime or not. We have introduced and discussed School method for primality testing in Set 1.

Primality Test | Set 1 (Introduction and School Method)

In this post, Fermat's method is discussed. This method is a probabilistic method and is based on below Fermat's Little Theorem.

```
Fermat's Little Theorem:
If n is a prime number, then for every a, 1 <= a < n,

an-1   1 (mod n)
 OR
an-1 % n = 1


Example: Since 5 is prime, 24   1 (mod 5) [or 24%5 = 1],
         34   1 (mod 5) and 44   1 (mod 5)

         Since 7 is prime, 26   1 (mod 7),
         36   1 (mod 7), 46   1 (mod 7)
         56   1 (mod 7) and 66   1 (mod 7)

Refer this for different proofs.
```

If a given number is prime, then this method always returns true. If given number is composite (or non-prime), then it may return true or false, but the probability of producing incorrect result for composite is low and can be reduced by doing more iterations.

Below is algorithm:

```
// Higher value of k indicates probability of correct
// results for composite inputs become higher. For prime
// inputs, result is always correct
1)  Repeat following k times:
      a) Pick a randomly in the range [2, n - 2]
      b) If an-1 &nequiv; 1 (mod n), then return false
2) Return true [probably prime].
```

Below is the implementation of above algorithm. The code uses power function from Modular Exponentiation

C++

```
 // C++ program to find the smallest twin in given range
#include <bits/stdc++.h>
using namespace std;

/* Iterative Function to calculate (a^n)%p in O(logy) */
int power(int a, unsigned int n, int p)
{
    int res = 1;      // Initialize result
    a = a % p;  // Update 'a' if 'a' >= p

    while (n > 0)
    {
        // If n is odd, multiply 'a' with result
        if (n & 1)
            res = (res*a) % p;

        // n must be even now
        n = n>>1; // n = n/2
        a = (a*a) % p;
    }
    return res;
}

// If n is prime, then always returns true, If n is
// composite than returns false with high probability
// Higher value of k increases probability of correct
// result.
bool isPrime(unsigned int n, int k)
{
   // Corner cases
   if (n <= 1 || n == 4)  return false;
   if (n <= 3) return true;

   // Try k times
   while (k>0)
   {
       // Pick a random number in [2..n-2]
       // Above corner cases make sure that n > 4
       int a = 2 + rand()%(n-4);

       // Fermat's little theorem
       if (power(a, n-1, n) != 1)
          return false;

       k--;
    }

    return true;
}

// Driver Program to test above function
int main()
{
    int k = 3;
    isPrime(11, k)?  cout << " true\n": cout << " false\n";
    isPrime(15, k)?  cout << " true\n": cout << " false\n";
    return 0;
}
```

Java

```
 // Java program to find the
// smallest twin in given range

import java.io.*;
import java.math.*;

class GFG {

    /* Iterative Function to calculate
    // (a^n)%p in O(logy) */
    static int power(int a,int n, int p)
    {
        // Initialize result
        int res = 1;

        // Update 'a' if 'a' >= p
        a = a % p;
        while (n > 0)
        {
            // If n is odd, multiply 'a' with result
            if ((n & 1) == 1)
                res = (res * a) % p;

            // n must be even now
            n = n >> 1; // n = n/2
            a = (a * a) % p;
        }
        return res;
    }

    // If n is prime, then always returns true,
    // If n is composite than returns false with
    // high probability Higher value of k increases
    //  probability of correct result.
    static boolean isPrime(int n, int k)
    {
    // Corner cases
    if (n <= 1 || n == 4) return false;
    if (n <= 3) return true;

    // Try k times
    while (k > 0)
    {
        // Pick a random number in [2..n-2]
        // Above corner cases make sure that n > 4
        int a = 2 + (int)(Math.random() % (n - 4));

        // Fermat's little theorem
        if (power(a, n - 1, n) != 1)
            return false;

        k--;
        }

        return true;
    }

    // Driver Program
    public static void main(String args[])
    {
        int k = 3;
        if(isPrime(11, k))
            System.out.println(" true");
        else
            System.out.println(" false");
        if(isPrime(15, k))
            System.out.println(" true");
        else
            System.out.println(" false");

    }
}

// This code is contributed by Nikita Tiwari.
```

PHP

```
 <?php
// PHP program to find the
// smallest twin in given range

// Iterative Function to calculate
// (a^n)%p in O(logy)
function power($a, $n, $p)
{

    // Initialize result
    $res = 1;

    // Update 'a' if 'a' >= p
    $a = $a % $p;

    while ($n > 0)
    {

        // If n is odd, multiply
        // 'a' with result
        if ($n & 1)
            $res = ($res * $a) % $p;

        // n must be even now
        $n = $n >> 1; // n = n/2
        $a = ($a * $a) % $p;
    }
    return $res;
}

// If n is prime, then always
// returns true, If n is
// composite than returns
// false with high probability
// Higher value of k increases
// probability of correct
// result.
function isPrime($n, $k)
{

    // Corner cases
    if ($n <= 1 || $n == 4)
    return false;
    if ($n <= 3)
    return true;

    // Try k times
    while ($k > 0)
    {

        // Pick a random number
        // in [2..n-2]
        // Above corner cases
        // make sure that n > 4
        $a = 2 + rand() % ($n - 4);

        // Fermat's little theorem
        if (power($a, $n-1, $n) != 1)
            return false;

        $k--;
    }

    return true;
}

// Driver Code
$k = 3;
$res = isPrime(11, $k) ? " true\n": " false\n";
echo($res);

$res = isPrime(15, $k) ? " true\n": " false\n";
echo($res);

// This code is contributed by Ajit.
?>
```

Output:

```
true
false
```

Time complexity of this solution is O(k Log n). Note that power function takes O(Log n) time.

Note that the above method may fail even if we increase number of iterations (higher k). There exist sum composite numbers with the property that for every a < n, **a^{n-1} 1 (mod n)**. Such numbers are called Carmichael numbers. Fermat's primality test is often used if a rapid method is needed for filtering, for example in key generation phase of the RSA public key cryptographic algorithm.

We will soon be discussing more methods for Primality Testing.

References:

https://en.wikipedia.org/wiki/Fermat_primality_test
https://en.wikipedia.org/wiki/Prime_number
http://www.cse.iitk.ac.in/users/manindra/presentations/FLTBasedTests.pdf
https://en.wikipedia.org/wiki/Primality_test

This article is contributed by **Ajay**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : jit_t

Source

https://www.geeksforgeeks.org/primality-test-set-2-fermet-method/

Chapter 22

Program to generate CAPTCHA and verify user

Program to generate CAPTCHA and verify user - GeeksforGeeks

A CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a test to determine whether the user is human or not.

So, the task is to generate unique CAPTCHA every time and to tell whether the user is human or not by asking user to enter the same CAPTCHA as generated automatically and checking the user input with the generated CAPTCHA.

Examples:

```
CAPTCHA: x9Pm72se
Input: x9Pm62es
Output: CAPTCHA Not Matched

CAPTCHA: cF3yl9T4
Input: cF3yl9T4
Output: CAPTCHA Matched
```

The set of characters to generate CAPTCHA are stored in a character array chrs[] which contains (a-z, A-Z, 0-9), therefore size of chrs[] is 62.

To generate a unique CAPTCHA every time, a random number is generated using rand() function (rand()%62) which generates a random number between 0 to 61 and the generated random number is taken as index to the character array chrs[] thus generates a new character of captcha[] and this loop runs n (length of CAPTCHA) times to generate CAPTCHA of given length.

```
// C++ program to automatically generate CAPTCHA and
// verify user
#include<bits/stdc++.h>
using namespace std;

// Returns true if given two strings are same
bool checkCaptcha(string &captcha, string &user_captcha)
{
    return captcha.compare(user_captcha) == 0;
}

// Generates a CAPTCHA of given length
string generateCaptcha(int n)
{
    time_t t;
    srand((unsigned)time(&t));

    // Characters to be included
    char *chrs = "abcdefghijklmnopqrstuvwxyzABCDEFGHI"
                 "JKLMNOPQRSTUVWXYZ0123456789";

    // Generate n characters from above set and
    // add these characters to captcha.
    string captcha = "";
    while (n--)
        captcha.push_back(chrs[rand()%62]);

    return captcha;
}

// Driver code
int main()
{
    // Generate a random CAPTCHA
    string captcha = generateCaptcha(9);
    cout << captcha;

    // Ask user to enter a CAPTCHA
    string usr_captcha;
    cout << "\nEnter above CAPTCHA: ";
    cin >> usr_captcha;

    // Notify user about matching status
    if (checkCaptcha(captcha, usr_captcha))
        printf("\nCAPTCHA Matched");
    else
        printf("\nCAPTCHA Not Matched");

    return 0;
}
```

Output:

```
CAPTCHA: cF3yl9T4
Enter CAPTCHA: cF3yl9T4
CAPTCHA Matched
```

Source

https://www.geeksforgeeks.org/program-generate-captcha-verify-user/

Chapter 23

QuickSort using Random Pivoting

QuickSort using Random Pivoting - GeeksforGeeks

In this article we will discuss how to implement QuickSort using random pivoting. In QuickSort we first partition the array in place such that all elements to the left of the pivot element are smaller, while all elements to the right of the pivot are greater that the pivot. Then we recursively call the same procedure for left and right subarrays.

Unlike merge sort we don't need to merge the two sorted arrays. Thus Quicksort requires lesser auxillary space than Merge Sort, which is why it is often preferred to Merge Sort.Using a randomly generated pivot we can further improve the time complexity of QuickSort.

We have discussed at two popular methods for partioning the arrays-Hoare's vs Lomuto partition scheme
It is advised that the reader has read that article or knows how to implement the QuickSort using either of the two partition schemes.

Algorithm for random pivoting using Lomuto Partitioning

```
partition(arr[], lo, hi)
    pivot = arr[hi]
    i = lo     // place for swapping
    for j := lo to hi - 1 do
        if arr[j] <= pivot then
            swap arr[i] with arr[j]
            i = i + 1
    swap arr[i] with arr[hi]
    return i

partition_r(arr[], lo, hi)
```

```
    r = Random Number from lo to hi
    Swap arr[r] and arr[hi]
    return partition(arr, lo, hi)

quicksort(arr[], lo, hi)
    if lo < hi
        p = partition_r(arr, lo, hi)
        quicksort(arr, p-1, hi)
        quicksort(arr, p+1, hi)
```

Algorithm for random pivoting using Hoare Partitioning

```
partition(arr[], lo, hi)
   pivot = arr[lo]
   i = lo - 1  // Initialize left index
   j = hi + 1  // Initialize right index

   // Find a value in left side greater
   // than pivot
   do
      i = i + 1
   while arr[i]  pivot

   if i >= j then
      return j

   swap arr[i] with arr[j]

partition_r(arr[], lo, hi)
    r = Random number from lo to hi
    Swap arr[r] and arr[lo]
    return partition(arr, lo, hi)

quicksort(arr[], lo, hi)
    if lo < hi
        p = partition_r(arr, lo, hi)
        quicksort(arr, p, hi)
        quicksort(arr, p+1, hi)
```

Below is the CPP implementation of the Algorithms

Lomuto (C++)

```
 /* C++ implementation QuickSort using Lomuto's partition
   Scheme.*/
```

```
#include <cstdlib>
#include <iostream>
using namespace std;

/* This function takes last element as pivot, places
  the pivot element at its correct position in sorted
  array, and places all smaller (smaller than pivot)
  to left of pivot and all greater elements to right
  of pivot */
int partition(int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i = (low - 1); // Index of smaller element

    for (int j = low; j <= high - 1; j++) {

        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot) {

            i++; // increment index of smaller element
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

// Generates Random Pivot, swaps pivot with
// end element and calls the partition function
int partition_r(int arr[], int low, int high)
{
    // Generate a random number in between
    // low .. high
    srand(time(NULL));
    int random = low + rand() % (high - low);

    // Swap A[random] with A[high]
    swap(arr[random], arr[high]);

    return partition(arr, low, high);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
```

```
{
    if (low < high) {

        /* pi is partitioning index, arr[p] is now
        at right place */
        int pi = partition_r(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = { 10, 7, 8, 9, 1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Hoare (C++)

```
 /* C++ implementation of QuickSort using Hoare's
   partition scheme. */
#include <cstdlib>
#include <iostream>
using namespace std;

/* This function takes last element as pivot, places
   the pivot element at its correct position in sorted
    array, and places all smaller (smaller than pivot)
   to left of pivot and all greater elements to right
   of pivot */
int partition(int arr[], int low, int high)
```

```
{
    int pivot = arr[low];
    int i = low - 1, j = high + 1;

    while (true) {

        // Find leftmost element greater than
        // or equal to pivot
        do {
            i++;
        } while (arr[i] < pivot);

        // Find rightmost element smaller than
        // or equal to pivot
        do {
            j--;
        } while (arr[j] > pivot);

        // If two pointers met.
        if (i >= j)
            return j;

        swap(arr[i], arr[j]);
    }
}

// Generates Random Pivot, swaps pivot with
// end element and calls the partition function
// In Hoare partition the low element is selected
// as first pivot
int partition_r(int arr[], int low, int high)
{
    // Generate a random number in between
    // low .. high
    srand(time(NULL));
    int random = low + rand() % (high - low);

    // Swap A[random] with A[high]
    swap(arr[random], arr[low]);

    return partition(arr, low, high);
}

/* The main function that implements QuickSort
 arr[] --> Array to be sorted,
  low  --> Starting index,
  high  --> Ending index */
void quickSort(int arr[], int low, int high)
```

```
{
    if (low < high) {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition_r(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi);
        quickSort(arr, pi + 1, high);
    }
}

/* Function to print an array */
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = { 10, 7, 8, 9, 1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Output:

```
Sorted array: 1 5 7 8 9 10
```

Analysis of Randomized Quick Sort

Notes

- Using random pivoting we improve the expected or average time complexity to O (N log N). The Worst Case complexity is still O (N^2).

Source

https://www.geeksforgeeks.org/quicksort-using-random-pivoting/

Chapter 24

Random Walk (Implementation in Python)

Random Walk (Implementation in Python) - GeeksforGeeks

Introduction A random walk is a mathematical object, known as a stochastic or random process, that describes a path that consists of a succession of random steps on some mathematical space such as the integers. An elementary example of a random walk is the random walk on the integer number line, which starts at 0 and at each step moves +1 or -1 with equal probability. Other examples include the path traced by a molecule as it travels in a liquid or a gas, the search path of a foraging animal, the price of a fluctuating stock and the financial status of a gambler can all be approximated by random walk models, even though they may not be truly random in reality. As illustrated by those examples, random walks have applications to many scientific fields including ecology, psychology, computer science, physics, chemistry, biology as well as economics. Random walks explain the observed behaviors of many processes in these fields, and thus serve as a fundamental model for the recorded stochastic activity. As a more mathematical application, the value of pi can be approximated by the usage of random walk in agent-based modelling environment.

Enough with the boring theory. Let's take a break while getting some knowledge for the code. So, to code out the random walk we will basically require some libraries in python some to do maths and some others to plot the curve.

Libraries Required

1. **matplotlib** Its a external library which help you to plot the curve. To install this library type the following code in you cmd.

   ```
   pip install matplotlib
   ```

   It would be enough to get you through the installation.

2. **numpy** It's also an external library in python it helps you to work with array and matrices. To install the library type the following code in cmd.

```
pip install numpy
```

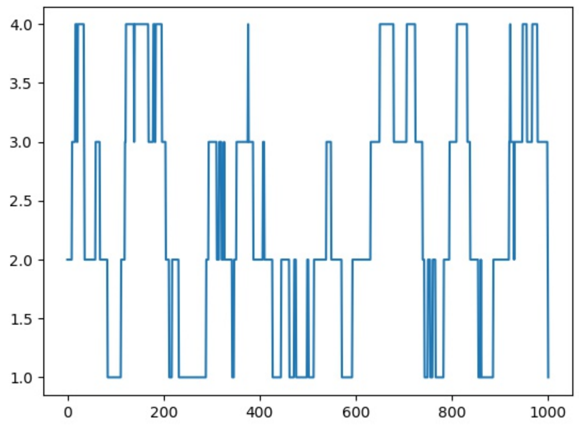
3. **random** It's a built-in library of python we will use it to generate random points.

One-dimensional random walk An elementary example of a random walk is the random walk on the integer number line, which starts at 0 and at each step moves +1 or ?1 with equal probability.

So lets try to implement the 1-D random walk in python.

```
 # Python code for 1-D random walk.
import random
import numpy as np
import matplotlib.pyplot as plt

# Probability to move up or down
prob = [0.05, 0.95]

# statically defining the starting position
start = 2
positions = [start]

# creating the random points
rr = np.random.random(1000)
downp = rr < prob[0]
upp = rr > prob[1]


for idownp, iupp in zip(downp, upp):
    down = idownp and positions[-1] > 1
    up = iupp and positions[-1] < 4
    positions.append(positions[-1] - down + up)

# plotting down the graph of the random walk in 1D
plt.plot(positions)
plt.show()
```

Output:

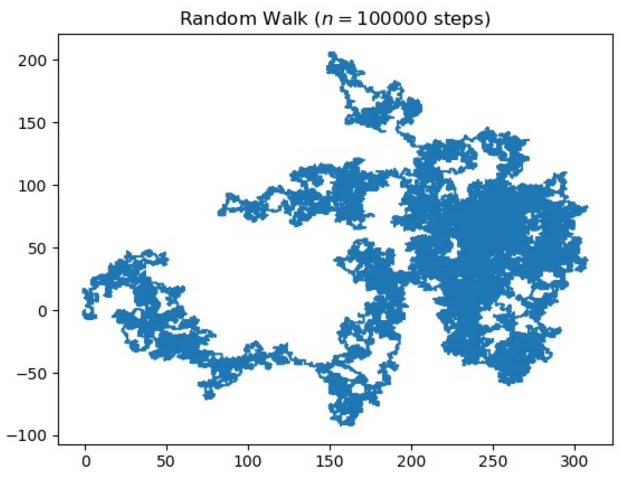


Higher dimensions In higher dimensions, the set of randomly walked points has interesting geometric properties. In fact, one gets a discrete fractal, that is, a set which exhibits stochastic self-similarity on large scales. On small scales, one can observe “jaggedness” resulting from the grid on which the walk is performed. Two books of Lawler referenced below are a good source on this topic. The trajectory of a random walk is the collection of points visited, considered as a set with disregard to when the walk arrived at the point. In one dimension, the trajectory is simply all points between the minimum height and the maximum height the walk achieved (both are, on average, on the order of ?n).

Let’s try to create random walk in 2D.

```
 # Python code for 2D random walk.
import numpy
import pylab
import random

# defining the number of steps
n = 100000

#creating two array for containing x and y coordinate
#of size equals to the number of size and filled up with 0's
x = numpy.zeros(n)
y = numpy.zeros(n)

# filling the coordinates with random variables
for i in range(1, n):
    val = random.randint(1, 4)
    if val == 1:
        x[i] = x[i - 1] + 1
        y[i] = y[i - 1]
    elif val == 2:
        x[i] = x[i - 1] - 1
        y[i] = y[i - 1]
    elif val == 3:
        x[i] = x[i - 1]
        y[i] = y[i - 1] + 1
    else:
        x[i] = x[i - 1]
        y[i] = y[i - 1] - 1


# plotting stuff:
pylab.title("Random Walk ($n = " + str(n) + "$ steps)")
pylab.plot(x, y)
pylab.savefig("rand_walk"+str(n)+".png",bbox_inches="tight",dpi=600)
pylab.show()
```

Output:



Applications

1. In computer networks, random walks can model the number of transmission packets buffered at a server.
2. In population genetics, random walk describes the statistical properties of genetic drift.
3. In image segmentation, random walks are used to determine the labels (i.e., "object" or "background") to associate with each pixel.
4. In brain research, random walks and reinforced random walks are used to model cascades of neuron firing in the brain.
5. Random walks have also been used to sample massive online graphs such as online social networks.

References

1. Wikipedia – Random Walk

2. Stackoverflow – Random Walk 1D
3. matplotlib documentation

Source

https://www.geeksforgeeks.org/random-walk-implementation-python/

Chapter 25

Random number generator in arbitrary probability distribution fashion

Random number generator in arbitrary probability distribution fashion - GeeksforGeeks

Given n numbers, each with some frequency of occurrence. Return a random number with probability proportional to its frequency of occurrence.

Example:

```
Let following be the given numbers.
  arr[] = {10, 30, 20, 40}

Let following be the frequencies of given numbers.
  freq[] = {1, 6, 2, 1}

The output should be
  10 with probability 1/10
  30 with probability 6/10
  20 with probability 2/10
  40 with probability 1/10
```

It is quite clear that the simple random number generator won't work here as it doesn't keep track of the frequency of occurrence.

We need to somehow transform the problem into a problem whose solution is known to us.

One simple method is to take an auxiliary array (say aux[]) and duplicate the numbers according to their frequency of occurrence. Generate a random number(say r) between 0 to Sum-1(including both), where Sum represents summation of frequency array (freq[] in above example). Return the random number aux[r] (Implementation of this method is left as an exercise to the readers).

The limitation of the above method discussed above is huge memory consumption when frequency of occurrence is high. If the input is 997, 8761 and 1, this method is clearly not efficient.

How can we reduce the memory consumption? Following is detailed algorithm that uses O(n) extra space where n is number of elements in input arrays.

1. Take an auxiliary array (say prefix[]) of size n.
2. Populate it with prefix sum, such that prefix[i] represents sum of numbers from 0 to i.
3. Generate a random number(say r) between 1 to Sum(including both), where Sum represents summation of input frequency array.
4. Find index of Ceil of random number generated in step #3 in the prefix array. Let the index be index**c**.
5. Return the random number arr[indexc], where arr[] contains the input n numbers.

Before we go to the implementation part, let us have quick look at the algorithm with an example:
arr[]: {10, 20, 30}
freq[]: {2, 3, 1}
Prefix[]: {2, 5, 6}
Since last entry in prefix is 6, all possible values of r are [1, 2, 3, 4, 5, 6]
1: Ceil is 2. Random number generated is 10.
2: Ceil is 2. Random number generated is 10.
3: Ceil is 5. Random number generated is 20.
4: Ceil is 5. Random number generated is 20.
5: Ceil is 5. Random number generated is 20.
6. Ceil is 6. Random number generated is 30.
In the above example
10 is generated with probability 2/6.
20 is generated with probability 3/6.
30 is generated with probability 1/6.

How does this work?
Any number input[i] is generated as many times as its frequency of occurrence because there exists count of integers in range(prefix[i – 1], prefix[i]] is input[i]. Like in the above example 3 is generated thrice, as there exists 3 integers 3, 4 and 5 whose ceil is 5.

```
 //C program to generate random numbers according to given frequency distribution
#include <stdio.h>
#include <stdlib.h>

// Utility function to find ceiling of r in arr[l..h]
int findCeil(int arr[], int r, int l, int h)
{
    int mid;
    while (l < h)
    {
         mid = l + ((h - l) >> 1);  // Same as mid = (l+h)/2
        (r > arr[mid]) ? (l = mid + 1) : (h = mid);
    }
    return (arr[l] >= r) ? l : -1;
}

// The main function that returns a random number from arr[] according to
// distribution array defined by freq[]. n is size of arrays.
int myRand(int arr[], int freq[], int n)
{
    // Create and fill prefix array
    int prefix[n], i;
    prefix[0] = freq[0];
    for (i = 1; i < n; ++i)
        prefix[i] = prefix[i - 1] + freq[i];

    // prefix[n-1] is sum of all frequencies. Generate a random number
    // with value from 1 to this sum
    int r = (rand() % prefix[n - 1]) + 1;

    // Find index of ceiling of r in prefix arrat
    int indexc = findCeil(prefix, r, 0, n - 1);
    return arr[indexc];
}

// Driver program to test above functions
int main()
{
    int arr[]  = {1, 2, 3, 4};
    int freq[] = {10, 5, 20, 100};
    int i, n = sizeof(arr) / sizeof(arr[0]);

    // Use a different seed value for every run.
    srand(time(NULL));

    // Let us generate 10 random numbers accroding to
    // given distribution
    for (i = 0; i < 5; i++)
      printf("%d\n", myRand(arr, freq, n));

    return 0;
}
```

Output: May be different for different runs

```
4
3
```

```
4
4
4
```

This article is compiled by Aashish Barnwal. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/random-number-generator-in-arbitrary-probability-distribution-fashion/

Chapter 26

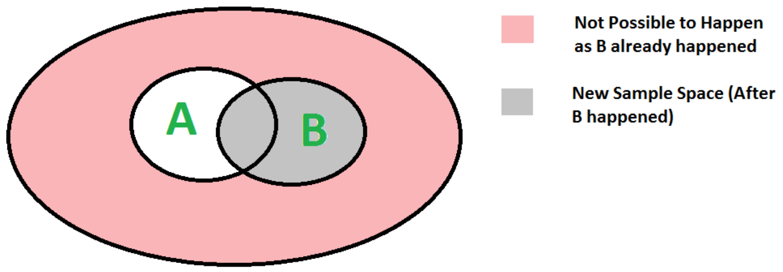
Randomized Algorithms | Set 0 (Mathematical Background)

Randomized Algorithms | Set 0 (Mathematical Background) - GeeksforGeeks

Conditional Probability Conditional probability P(A | B) indicates the probability of even 'A' happening given that the even B happened.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

We can easily understand above formula using below diagram. Since B has already happened, the sample space reduces to B. So the probability of A happening becomes P(A   B) divided by P(B)


A
B
Not Possible to Happen as B already happened
New Sample Space (After B happened)

Below is Bayes's formula for conditional probability.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

The formula provides relationship between P(A|B) and P(B|A). It is mainly derived form conditional probability formula discussed in the previous post.

Consider the below forrmulas for conditional probabilities P(A|B) and P(B|A)

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$

Since P(B A) = P(A B), we can replace P(A B) in first formula with P(B|A)P(A) After replacing, we get the given formula. Refer this for examples of Bayes's formula.

Random Variables:
A random variable is actually a function that maps outcome of a random event (like coin toss) to a real value.

Example :

```
Coin tossing game :
A player pays 50 bucks if result of coin
toss is "Head"

The person gets 50 bucks if the result is
Tail.

A random variable profit for person can
be defined as below :

Profit = +50 if Head
         -50 if Tail

Generally gambling games are not fair for players,
the organizer takes a share of profit for all
arrangements. So expected profit is negative for
a player in gambling and positive for the organizer.
That is how organizers make money.
```

Expected Value of Random Variable :
Expected value of a random variable R can be defined as following

```
    E[R] = r1*p1 + r2*p2 + ... rk*pk

    ri ==> Value of R with probability pi
```

Expected value is basically sum of product of following two terms (for all possible events)
a) Probability of an event.
b) Value of R at that even

```
Example 1:
In above example of coin toss,
Expected value of profit = 50 * (1/2) +
                          (-50) * (1/2)
                         = 0

Example 2:
Expected value of six faced dice throw is
  = 1*(1/6) + 2*(1/6) + .... + 6*(1/6)
  = 3.5
```

Linearity of Expectation:
Let R_1 and R_2 be two discrete random variables on some probability space, then

```
     E[R1 + R2] = E[R1] + E[R2]
```

For example, expected value of sum for 3 dice throws is = 3 * 7/2 = 7

Refer this for more detailed explanation and examples.

Expected Number of Trials until Success
If probability of success is p in every trial, then expected number of trials until success is 1/p. For example, consider 6 faced fair dice is thrown until a '5' is seen as result of dice throw. The expected number of throws before seeing a 5 is 6. Note that 1/6 is probability of getting a 5 in every trial. So number of trials is 1/(1/6) = 6.
As another example, consider a QuickSort version that keeps on looking for pivots until one of the middle n/2 elements is picked. The expected time number of trials for finding middle pivot would be 2 as probability of picking one of the middle n/2 elements is 1/2. This example is discussed in more detail in Set 1.
Refer this for more detailed explanation and examples.

More on Randomized Algorithms:

- Randomized Algorithms | Set 1 (Introduction and Analysis)
- Randomized Algorithms | Set 2 (Classification and Applications)
- Randomized Algorithms | Set 3 (1/2 Approximate Median)

All Randomized Algorithm Topics

This article is contributed by **Shivam Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : ChandanKumar9

Source

https://www.geeksforgeeks.org/randomized-algorithms-set-0-mathematical-background/

Chapter 27

Randomized Algorithms | Set 1 (Introduction and Analysis)

Randomized Algorithms | Set 1 (Introduction and Analysis) - GeeksforGeeks

An algorithm that uses random numbers to decide what to do next anywhere in its logic is called Randomized Algorithm.. For example, in Randomized Quick Sort, we use random number to pick the next pivot (or we randomly shuffle the array). And in Karger's algorithm, we randomly pick an edge.

Some randomized algorithms have deterministic time complexity. For example, this implementation of Karger's algorithm has time complexity as O(E). Such algorithms are called Monte Carlo Algorithms and are easier to analyse for worst case.
On the other hand, time complexity of other randomized algorithms (other than Las Vegas) is dependent on value of random variable. Such Randomized algorithms are calledLas Vegas Algorithms. These algorithms are typically analysed for expected worst case. To compute expected time taken in worst case, all possible values of the used random variable needs to be considered in worst case and time taken by every possible value needs to be evaluated. Average of all evaluated times is the expected worst case time complexity. Below facts are generally helpful in analysis os such algorithms.
Linearity of Expectation
Expected Number of Trials until Success.

For example consider below a randomized version of QuickSort.

A **Central Pivot** is a pivot that divides the array in such a way that one side has at-least 1/4 elements.

```
// Sorts an array arr[low..high]
randQuickSort(arr[], low, high)

1. If low >= high, then EXIT.
```

```
2. While pivot 'x' is not a Central Pivot.
  (i)   Choose uniformly at random a number from [low..high].
        Let the randomly picked number number be x.
  (ii)  Count elements in arr[low..high] that are smaller
        than arr[x]. Let this count be sc.
  (iii) Count elements in arr[low..high] that are greater
        than arr[x]. Let this count be gc.
  (iv)  Let n = (high-low+1). If sc >= n/4 and
        gc >= n/4, then x is a central pivot.

3. Partition arr[low..high] around the pivot x.

4. // Recur for smaller elements
   randQuickSort(arr, low, sc-1)

5. // Recur for greater elements
   randQuickSort(arr, high-gc+1, high)
```

The important thing in our analysis is, time taken by step 2 is O(n).

How many times while loop runs before finding a central pivot?
The probability that the randomly chosen element is central pivot is 1/2.

Therefore, expected number of times the while loop runs is 2 (See this for details)

Thus, the expected time complexity of step 2 is O(n).

What is overall Time Complexity in Worst Case?
In worst case, each partition divides array such that one side has n/4 elements and other side has 3n/4 elements. The worst case height of recursion tree is Log $_{3/4}$ n which is O(Log n).

```
T(n) < T(n/4) + T(3n/4) + O(n)
T(n) < 2T(3n/4) + O(n)

Solution of above recurrence is O(n Log n)
```

Note that the above randomized algorithm is not the best way to implement randomized Quick Sort. The idea here is to simplify the analysis as it is simple to analyse.
Typically, randomized Quick Sort is implemented by randomly picking a pivot (no loop). Or by shuffling array elements. Expected worst case time complexity of this algorithm is also O(n Log n), but analysis is complex, the MIT prof himself mentions same in his lecture here.

Randomized Algorithms | Set 2 (Classification and Applications)

References:
http://www.tcs.tifr.res.in/~workshop/nitrkl_igga/randomized-lecture.pdf

This article is contributed by **Ashish Sharma**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/randomized-algorithms-set-1-introduction-and-analysis/

Chapter 28

Randomized Algorithms | Set 2 (Classification and Applications)

Randomized Algorithms | Set 2 (Classification and Applications) - GeeksforGeeks

We strongly recommend to refer below post as a prerequisite of this.

Randomized Algorithms | Set 1 (Introduction and Analysis)

Randomized algorithms are classified in two categories.

Las Vegas: These algorithms always produce correct or optimum result. Time complexity of these algorithms is based on a random value and time complexity is evaluated as expected value. For example, Randomized QuickSort always sorts an input array and expected worst case time complexity of QuickSort is O(nLogn).

Monte Carlo: Produce correct or optimum result with some probability. These algorithms have deterministic running time and it is generally easier to find out worst case time complexity. For example this implementation of Karger's Algorithm produces minimum cut with probability greater than or equal to $1/n^2$ (n is number of vertices) and has worst case time complexity as O(E). Another example is Fermet Method for Primality Testing.

Example to Understand Classification:

```
Consider a binary array where exactly half elements are 0
and half are 1. The task is to find index of any 1.
```

A Las Vegas algorithm for this task is to keep picking a random element until we find a 1.
A Monte Carlo algorithm for the same is to keep picking a random element until we either find 1 or we have tried maximum allowed times say k.
The Las Vegas algorithm always finds an index of 1, but time complexity is determined as expect value. The expected number of trials before success is 2, therefore expected time complexity is O(1).
The Monte Carlo Algorithm finds a 1 with probability $[1 - (1/2)^k]$. Time complexity of Monte Carlo is O(k) which is deterministic

- Consider a tool that basically does sorting. Let the tool be used by many users and there are few users who always use tool for already sorted array. If the tool uses simple (not randomized) QuickSort, then those few users are always going to face worst case situation. On the other hand if the tool uses Randomized QuickSort, then there is no user that always gets worst case. Everybody gets expected O(n Log n) time.
- Randomized algorithms have huge applications in Cryptography.
- Load Balancing.
- Number-Theoretic Applications: Primality Testing
- Data Structures: Hashing, Sorting, Searching, Order Statistics and Computational Geometry.
- Algebraic identities: Polynomial and matrix identity verification. Interactive proof systems.
- Mathematical programming: Faster algorithms for linear programming, Rounding linear program solutions to integer program solutions
- Graph algorithms: Minimum spanning trees, shortest paths, minimum cuts.
- Counting and enumeration: Matrix permanent Counting combinatorial structures.
- Parallel and distributed computing: Deadlock avoidance distributed consensus.
- Probabilistic existence proofs: Show that a combinatorial object arises with non-zero probability among objects drawn from a suitable probability space.
- Derandomization: First devise a randomized algorithm then argue that it can be derandomized to yield a deterministic algorithm.

Sources:
http://theory.stanford.edu/people/pragh/amstalk.pdf
https://en.wikipedia.org/wiki/Randomized_algorithm

This article is contributed by **Ashish Sharma**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/randomized-algorithms-set-2-classification-and-applications/

Chapter 29

Randomized Algorithms | Set 3 (1/2 Approximate Median)

Randomized Algorithms | Set 3 (1/2 Approximate Median) - GeeksforGeeks

We strongly recommend to refer below articles as a prerequisite of this.

Randomized Algorithms | Set 1 (Introduction and Analysis)
Randomized Algorithms | Set 2 (Classification and Applications)

In this post, a Monte Carlo algorithm is discussed.

Problem Statement : Given an unsorted array A[] of n numbers and > 0, compute an element whose rank (position in sorted A[]) is in the range $[(1 - \ )n/2, (1 + \ )n/2]$.
For ½ Approximate Median Algorithm &epsilom; is 1/2 => rank should be in the range [n/4, 3n/4]

We can find k'th smallest element in O(n) expected time and O(n) worst case time.

What if we want in less than O(n) time with low probable error allowed?
Following steps represent an algorithm that is O((Log n) x (Log Log n)) time and produces incorrect result with probability less than or equal to $2/n^2$.

1. Randomly choose k elements from the array where k=c log n (c is some constant)
2. Insert then into a set.
3. Sort elements of the set.
4. Return median of the set i.e. (k/2)th element from the set

Source

https://www.geeksforgeeks.org/randomized-algorithms-set-3-12-approximate-median/

Chapter 30

Randomized Binary Search Algorithm

Randomized Binary Search Algorithm - GeeksforGeeks

We are given a sorted array A[] of n elements. We need to find if x is present in A or not.In binary search we always used middle element, here we will randomly pick one element in given range.

In Binary Search we had

```
middle = (start + end)/2
```

In Randomized binary search we do following

```
Generate a random number t
Since range of number in which we want a random
number is [start, end]
Hence we do, t = t % (end-start+1)
Then, t = start + t;
Hence t is a random number between start and end
```

It is a Las Vegas randomized algorithm as it always finds the correct result.

Expected Time complexity of Randomized Binary Search Algorithm
For n elements let say expected time required be T(n), After we choose one random pivot, array size reduces to say k. Since pivot is chosen with equal probability for all possible pivots, hence p = 1/n.

T(n) is sum of time of all possible sizes after choosing pivot multiplied by probability of choosing that pivot plus time take to generate random pivot index.Hence

```
T(n) = p*T(1) + p*T(2) + ..... + p*T(n) + 1
putting p = 1/n
T(n) = ( T(1) + T(2) + ..... + T(n) ) / n + 1
n*T(n) = T(1) + T(2) + .... + T(n) + n      .... eq(1)
Similarly for n-1
(n-1)*T(n-1) = T(1) + T(2) + ..... + T(n-1) + n-1    .... eq(2)
Subtract eq(1) - eq(2)
n*T(n) - (n-1)*T(n-1) = T(n) + 1
(n-1)*T(n) - (n-1)*T(n-1) =  1
(n-1)*T(n) = (n-1)*T(n-1) + 1
T(n) = 1/(n-1) + T(n-1)
T(n) = 1/(n-1) + 1/(n-2) + T(n-2)
T(n) = 1/(n-1) + 1/(n-2) + 1/(n-3) + T(n-3)
Similarly,
T(n) = 1 + 1/2 + 1/3 + ... + 1/(n-1)
Hence T(n) is equal to (n-1)th Harmonic number,
n-th harmonic number is O(log n)
Hence T(n) is O(log n)
```

Recursive C++ implementation of Randomized Binary Search

```
 // C++ program to implement recursive
// randomized algorithm.
#include <iostream>
#include <ctime>
using namespace std;

// To generate random number
// between x and y ie.. [x, y]
int getRandom(int x, int y)
{
    srand(time(NULL));
    return (x + rand() % (y-x+1));
}

// A recursive randomized binary search function.
// It returns location of x in
// given array arr[l..r] is present, otherwise -1
int randomizedBinarySearch(int arr[], int l,
                            int r, int x)
{
    if (r >= l)
    {
        // Here we have defined middle as
        // random index between l and r ie.. [l, r]
        int mid = getRandom(l, r);
        // If the element is present at the
        // middle itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
          return randomizedBinarySearch(arr, l,
                                    mid-1, x);

        // Else the element can only be present
        // in right subarray
        return randomizedBinarySearch(arr, mid+1,
                                         r, x);
    }

    // We reach here when element is not present
    // in array
    return -1;
}

// Driver code
int main(void)
{
    int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof(arr)/ sizeof(arr[0]);
    int x = 10;
    int result = randomizedBinarySearch(arr, 0, n-1, x);
    (result == -1)? printf("Element is not present in array")
    : printf("Element is present at index %d", result);
    return 0;
}
```

Output:

```
Element is present at index 3
```

Iterative C++ implementation of Randomized Binary Search

```
 // C++ program to implement iterative
// randomized algorithm.
#include <iostream>
#include <ctime>
using namespace std;
```

```
// To generate random number
// between x and y ie.. [x, y]
int getRandom(int x, int y)
{
    srand(time(NULL));
    return (x + rand()%(y-x+1));
}

// A iterative randomized binary search function.
// It returns location of x in
// given array arr[l..r] if present, otherwise -1
int randomizedBinarySearch(int arr[], int l,
                               int r, int x)
{
    while (l <= r)
    {
        // Here we have defined middle as
        // random index between l and r ie.. [l, r]
        int m = getRandom(l, r);

        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
            r = m - 1;
    }

    // if we reach here, then element was
    // not present
    return -1;
}

// Driver code
int main(void)
{
    int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof(arr)/ sizeof(arr[0]);
    int x = 10;
    int result = randomizedBinarySearch(arr, 0, n-1, x);
    (result == -1)? printf("Element is not present in array")
        : printf("Element is present at index %d", result);
    return 0;
}
```

Output:

```
Element is present at index 3
```

Source

https://www.geeksforgeeks.org/randomized-binary-search-algorithm/

Chapter 31

Reservoir Sampling

Reservoir Sampling - GeeksforGeeks

Reservoir sampling is a family of randomized algorithms for randomly choosing *k* samples from a list of *n* items, where *n* is either a very large or unknown number. Typically *n* is large enough that the list doesn't fit into main memory. For example, a list of search queries in Google and Facebook.

So we are given a big array (or stream) of numbers (to simplify), and we need to write an efficient function to randomly select *k* numbers where *1 <= k <= n*. Let the input array be *stream[]*.

A **simple solution** is to create an array *reservoir[]* of maximum size *k*. One by one randomly select an item from *stream[0..n-1]*. If the selected item is not previously selected, then put it in *reservoir[]*. To check if an item is previously selected or not, we need to search the item in *reservoir[]*. The time complexity of this algorithm will be *O(k^2)*. This can be costly if *k* is big. Also, this is not efficient if the input is in the form of a stream.

It **can be solved in *O(n)* time**. The solution also suits well for input in the form of stream. The idea is similar to thispost. Following are the steps.

1) Create an array *reservoir[0..k-1]* and copy first *k* items of *stream[]* to it.
2) Now one by one consider all items from *(k+1)*th item to *n*th item.
…**a)** Generate a random number from 0 to *i* where *i* is index of current item in *stream[]*. Let the generated random number is *j*.
…**b)** If *j* is in range 0 to *k-1*, replace *reservoir[j]* with *arr[i]*

Following is implementation of the above algorithm.

C/C++

```
 // An efficient program to randomly select k items from a stream of items

#include <stdio.h>
#include <stdlib.h>
```

```
#include <time.h>

// A utility function to print an array
void printArray(int stream[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", stream[i]);
    printf("\n");
}

// A function to randomly select k items from stream[0..n-1].
void selectKItems(int stream[], int n, int k)
{
    int i;  // index for elements in stream[]

    // reservoir[] is the output array. Initialize it with
    // first k elements from stream[]
    int reservoir[k];
    for (i = 0; i < k; i++)
        reservoir[i] = stream[i];

    // Use a different seed value so that we don't get
    // same result each time we run this program
    srand(time(NULL));

    // Iterate from the (k+1)th element to nth element
    for (; i < n; i++)
    {
        // Pick a random index from 0 to i.
        int j = rand() % (i+1);

        // If the randomly  picked index is smaller than k, then replace
        // the element present at the index with new element from stream
        if (j < k)
          reservoir[j] = stream[i];
    }

    printf("Following are k randomly selected items \n");
    printArray(reservoir, k);
}

// Driver program to test above function.
int main()
{
    int stream[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int n = sizeof(stream)/sizeof(stream[0]);
    int k = 5;
    selectKItems(stream, n, k);
    return 0;
}
```

Java

```
 // An efficient Java program to randomly
// select k items from a stream of items
import java.util.Arrays;
import java.util.Random;
public class ReservoirSampling
{
    // A function to randomly select k items from stream[0..n-1].
    static void selectKItems(int stream[], int n, int k)
    {
        int i;   // index for elements in stream[]

        // reservoir[] is the output array. Initialize it with
        // first k elements from stream[]
        int reservoir[] = new int[k];
        for (i = 0; i < k; i++)
            reservoir[i] = stream[i];

        Random r = new Random();

        // Iterate from the (k+1)th element to nth element
        for (; i < n; i++)
        {
            // Pick a random index from 0 to i.
            int j = r.nextInt(i + 1);

            // If the randomly  picked index is smaller than k,
            // then replace the element present at the index
            // with new element from stream
            if(j < k)
                reservoir[j] = stream[i];
        }

        System.out.println("Following are k randomly selected items");
        System.out.println(Arrays.toString(reservoir));
    }

    //Driver Program to test above method
    public static void main(String[] args) {
        int stream[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
        int n = stream.length;
        int k = 5;
        selectKItems(stream, n, k);
    }
}
//This code is contributed by Sumit Ghosh
```

Python3

```
 # An efficient Python3 program
# to randomly select k items
# from a stream of items
import random
# A utility function
# to print an array
def printArray(stream,n):
    for i in range(n):
        print(stream[i],end=" ");
    print();

# A function to randomly select
# k items from stream[0..n-1].
def selectKItems(stream, n, k):
        i=0;
        # index for elements
        # in stream[]

        # reservoir[] is the output
        # array. Initialize it with
        # first k elements from stream[]
        reservoir = [0]*k;
        for i in range(k):
            reservoir[i] = stream[i];

        # Iterate from the (k+1)th
        # element to nth element
        while(i < n):
            # Pick a random index
            # from 0 to i.
            j = random.randrange(i+1);

            # If the randomly picked
            # index is smaller than k,
            # then replace the element
            # present at the index
            # with new element from stream
            if(j < k):
                reservoir[j] = stream[i];
            i+=1;

        print("Following are k randomly selected items");
        printArray(reservoir, k);
```

```
# Driver Code

if __name__ == "__main__":
    stream = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12];
    n = len(stream);
    k = 5;
    selectKItems(stream, n, k);

# This code is contributed by mits
```

PHP

```
 <?php
// An efficient PHP program
// to randomly select k items
// from a stream of items

// A utility function
// to print an array
function printArray($stream,$n)
{
    for ($i = 0; $i < $n; $i++)
        echo $stream[$i]." ";
    echo "\n";
}

// A function to randomly select
// k items from stream[0..n-1].
function selectKItems($stream, $n, $k)
    {
        $i; // index for elements
            // in stream[]

        // reservoir[] is the output
        // array. Initialize it with
        // first k elements from stream[]
        $reservoir = array_fill(0, $k, 0);
        for ($i = 0; $i < $k; $i++)
            $reservoir[$i] = $stream[$i];

        // Iterate from the (k+1)th
        // element to nth element
        for (; $i < $n; $i++)
        {
            // Pick a random index
            // from 0 to i.
            $j = rand(0,$i + 1);
            // If the randomly picked
            // index is smaller than k,
            // then replace the element
            // present at the index
            // with new element from stream
            if($j < $k)
                $reservoir[$j] = $stream[$i];
        }

        echo "Following are k randomly ".
                     "selected items\n";
        printArray($reservoir, $k);
    }

// Driver Code
$stream = array(1, 2, 3, 4, 5, 6, 7,
                8, 9, 10, 11, 12);
$n = count($stream);
$k = 5;
selectKItems($stream, $n, $k);

// This code is contributed by mits
?>
```

Output:

```
Following are k randomly selected items
6 2 11 8 12
Note: Output will differ every time as it selects and prints random elements
```

Time Complexity: O(n)

How does this work?
To prove that this solution works perfectly, we must prove that the probability that any item *stream[i]* where *0 <= i < n* will be in final *reservoir[]* is *k/n.* Let us divide the proof in two cases as first *k* items are treated differently.

Case 1: For last *n-k* stream items, i.e., for *stream[i]* where *k <= i < n*
For every such stream item *stream[i]*, we pick a random index from 0 to *i* and if the picked index is one of the first *k* indexes, we replace the element at picked index with *stream[i]*

To simplify the proof, let us first consider the *last item.* The probability that the last item is in final reservoir = The probability that one of the first *k* indexes is picked for last item = *k/n* (the probability of picking one of the *k* items from a list of size *n*)

Let us now consider the *second last item.* The probability that the second last item is in final *reservoir[]* = [Probability that one of the first *k* indexes is picked in iteration for *stream[n-2]*] X [Probability that the index picked in iteration for *stream[n-1]* is not same as index picked for *stream[n-2]*] = [*k/(n-1)]*[(n-1)/n*] = *k/n.*

Similarly, we can consider other items for all stream items from *stream[n-1]* to *stream[k]* and generalize the proof.

Case 2: For first *k* stream items, i.e., for *stream[i]* where *0 <= i < k*
The first *k* items are initially copied to *reservoir[]* and may be removed later in iterations for *stream[k]* to *stream[n]*.
The probability that an item from *stream[0..k-1]* is in final array = Probability that the item is not picked when items *stream[k], stream[k+1], …. stream[n-1]* are considered = *[k/(k+1)] x [(k+1)/(k+2)] x [(k+2)/(k+3)] x … x [(n-1)/n] = k/n*

References:
http://en.wikipedia.org/wiki/Reservoir_sampling

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/reservoir-sampling/

Chapter 32

Select a Random Node from a Singly Linked List

Select a Random Node from a Singly Linked List - GeeksforGeeks

Given a singly linked list, select a random node from linked list (the probability of picking a node should be 1/N if there are N nodes in list). You are given a random number generator.

Below is a Simple Solution
1) Count number of nodes by traversing the list.
2) Traverse the list again and select every node with probability 1/N. The selection can be done by generating a random number from 0 to N-i for i'th node, and selecting the i'th node node only if generated number is equal to 0 (or any other fixed number from 0 to N-i).

We get uniform probabilities with above schemes.

```
i = 1, probability of selecting first node = 1/N
i = 2, probability of selecting second node =
                   [probability that first node is not selected] *
                   [probability that second node is selected]
                  = ((N-1)/N)* 1/(N-1)
                  = 1/N
```

Similarly, probabilities of other selecting other nodes is 1/N

The above solution requires two traversals of linked list.

How to select a random node with only one traversal allowed?
The idea is to use Reservoir Sampling. Following are the steps. This is a simpler version of Reservoir Sampling as we need to select only one key instead of k keys.

```
(1) Initialize result as first node
```

```
   result = head->key
(2) Initialize n = 2
(3) Now one by one consider all nodes from 2nd node onward.
    (3.a) Generate a random number from 0 to n-1.
         Let the generated random number is j.
    (3.b) If j is equal to 0 (we could choose other fixed number
          between 0 to n-1), then replace result with current node.
    (3.c) n = n+1
    (3.d) current = current->next
```

Below is the implementation of above algorithm.

C

```
 /* C program to randomly select a node from a singly
   linked list */
#include<stdio.h>
#include<stdlib.h>
#include <time.h>

/* Link list node */
struct Node
{
    int key;
    struct Node* next;
};

// A reservoir sampling based function to print a
// random node from a linked list
void printRandom(struct Node *head)
{
    // IF list is empty
    if (head == NULL)
       return;

    // Use a different seed value so that we don't get
    // same result each time we run this program
    srand(time(NULL));

    // Initialize result as first node
    int result = head->key;

    // Iterate from the (k+1)th element to nth element
    struct Node *current = head;
    int n;
    for (n=2; current!=NULL; n++)
    {
        // change result with probability 1/n
        if (rand() % n == 0)
           result = current->key;

        // Move to next node
        current = current->next;
    }

    printf("Randomly selected key is %d\n", result);
}

/* BELOW FUNCTIONS ARE JUST UTILITY TO TEST  */

/* A utility function to create a new node */
struct Node *newNode(int new_key)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    /* put in the key  */
    new_node->key  = new_key;
    new_node->next =  NULL;

    return new_node;
}


/* A utility function to insert a node at the beginning
  of linked list */
void push(struct Node** head_ref, int new_key)
{
    /* allocate node */
    struct Node* new_node = new Node;

    /* put in the key  */
    new_node->key  = new_key;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}


// Driver program to test above functions
int main()
```

```
{
    struct Node *head = NULL;
    push(&head, 5);
    push(&head, 20);
    push(&head, 4);
    push(&head, 3);
    push(&head, 30);

    printRandom(head);

    return 0;
}
```

Java

```
 // Java program to select a random node from singly linked list

import java.util.*;

// Linked List Class
class LinkedList {

    static Node head;  // head of list

    /* Node Class */
    static class Node {

        int data;
        Node next;

        // Constructor to create a new node
        Node(int d) {
            data = d;
            next = null;
        }
    }

    // A reservoir sampling based function to print a
    // random node from a linked list
    void printrandom(Node node) {

        // If list is empty
        if (node == null) {
            return;
        }

        // Use a different seed value so that we don't get
        // same result each time we run this program
        Math.abs(UUID.randomUUID().getMostSignificantBits());

        // Initialize result as first node
        int result = node.data;

        // Iterate from the (k+1)th element to nth element
        Node current = node;
        int n;
        for (n = 2; current != null; n++) {

            // change result with probability 1/n
            if (Math.random() % n == 0) {
                result = current.data;
            }

            // Move to next node
            current = current.next;
        }

        System.out.println("Randomly selected key is " + result);
    }

    // Driver program to test above functions
    public static void main(String[] args) {

        LinkedList list = new LinkedList();
        list.head = new Node(5);
        list.head.next = new Node(20);
        list.head.next.next = new Node(4);
        list.head.next.next.next = new Node(3);
        list.head.next.next.next.next = new Node(30);

        list.printrandom(head);

    }
}

// This code has been contributed by Mayank Jaiswal
```

Python

```
 # Python program to randomly select a node from singly
# linked list

import random

# Node class
class Node:
    # Constructor to initialize the node object
    def __init__(self, data):
        self.data= data
        self.next = None

class LinkedList:

    # Function to initialize head
    def __init__(self):
        self.head = None

    # A reservoir sampling based function to print a
    # random node from a linkd list
    def printRandom(self):

        # If list is empty
        if self.head is None:
            return

        # Use a different seed value so that we don't get
        # same result each time we run this program
        random.seed()

        # Initialize result as first node
        result = self.head.data

        # Iterate from the (k+1)th element nth element
        current = self.head
        n = 2
        while(current is not None):

            # change result with probability 1/n
            if (random.randrange(n) == 0 ):
                result = current.data

            # Move to next node
            current = current.next
            n += 1

        print "Randomly selected key is %d" %(result)

    # Function to insert a new node at the beginning
    def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node
    # Utility function to print the linked LinkedList
    def printList(self):
        temp = self.head
        while(temp):
            print temp.data,
            temp = temp.next


# Driver program to test above function
llist = LinkedList()
llist.push(5)
llist.push(20)
llist.push(4)
llist.push(3)
llist.push(30)
llist.printRandom()

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Note that the above program is based on outcome of a random function and may produce different output.

How does this work?
Let there be total N nodes in list. It is easier to understand from last node.

The probability that last node is result simply 1/N [For last or N'th node, we generate a random number between 0 to N-1 and make last node as result if the generated number is 0 (or any other fixed number]

The probability that second last node is result should also be 1/N.

```
The probability that the second last node is result
          = [Probability that the second last node replaces result] X
            [Probability that the last node doesn't replace the result]
          = [1 / (N-1)] * [(N-1)/N]
          = 1/N
```

Similarly we can show probability for 3rd last node and other nodes.

This article is contributed by **Rajeev**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/select-a-random-node-from-a-singly-linked-list/

Chapter 33

Select a Random Node from a tree with equal probability

Select a Random Node from a tree with equal probability - GeeksforGeeks

Given a Binary Tree with children Nodes, Return a random Node with equal Probability of selecting any Node in tree.

Consider the given tree with root as 1.

```
      10
    /    \
   20     30
  / \      / \
40   50   60   70
```

Examples:

```
Input : getRandom(root);
Output : A Random Node From Tree : 3

Input : getRandom(root);
Output : A Random Node From Tree : 2
```

A **simple solution** is to store Inorder traversal of tree in an array. Let the count of nodes be n. To get a random node, we generate a random number from 0 to n-1, use this number as index in array and return the value at index.

An **alternate solution** is to modify tree structure. We store count of children in every node. Consider the above tree. We use inorder traversal here also. We generate a number smaller than or equal count of nodes. We traverse tree and go to the node at that index.

We use counts to quickly reach the desired node. With counts, we reach in O(h) time where h is height of tree.

```
      10,6
    /      \
  20,2      30,2
 /   \      /   \
40,0 50,0  60,0  70,0
The first value is node and second
value is count of children.
```

We start traversing the tree, on each node we either go to left subtree or right subtree considering whether the count of children is less than random count or not.

If the random count is less than the count of children then we go left else we go right.

Below is the implementation of above Algorithm. getElements will return count of children for root, InsertChildrenCount inserts children data to each node, RandomNode return the random node with the help of Utility Function RandomNodeUtil.

Source

https://www.geeksforgeeks.org/select-random-node-tree-equal-probability/

C++

```
 // CPP program to Select a Random Node from a tree
#include <bits/stdc++.h>
using namespace std;

struct Node {
    int data;
    int children;
    Node *left, *right;
};

Node* newNode(int data)
{
    Node *temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    temp->children = 0;
    return temp;
}

// This is used to fill children counts.
int getElements(Node* root)
```

```
{
    if (!root)
        return 0;
    return getElements(root->left) +
          getElements(root->right) + 1;
}

// Inserts Children count for each node
void insertChildrenCount(Node*& root)
{
    if (!root)
        return;

    root->children = getElements(root) - 1;
    insertChildrenCount(root->left);
    insertChildrenCount(root->right);
}

// returns number of children for root
int children(Node* root)
{
    if (!root)
        return 0;
    return root->children + 1;
}

// Helper Function to return a random node
int randomNodeUtil(Node* root, int count)
{
    if (!root)
        return 0;

    if (count == children(root->left))
        return root->data;

    if (count < children(root->left))
        return randomNodeUtil(root->left, count);

    return randomNodeUtil(root->right,
              count - children(root->left) - 1);
}

// Returns Random node
int randomNode(Node* root)
{
    srand(time(0));

    int count = rand() % (root->children + 1);
    return randomNodeUtil(root, count);
}

int main()
{
    // Creating Above Tree
    Node* root = newNode(10);
    root->left = newNode(20);
    root->right = newNode(30);
    root->left->right = newNode(40);
    root->left->right = newNode(50);
    root->right->left = newNode(60);
    root->right->right = newNode(70);

    insertChildrenCount(root);

    cout << "A Random Node From Tree : "
         << randomNode(root) << endl;

    return 0;
}
```

Time Complexity of randomNode is O(h) where h is height of tree. Note that we are either moving to right or to left at a time.

Chapter 34

Select a random number from stream, with O(1) space

Select a random number from stream, with O(1) space - GeeksforGeeks

Given a stream of numbers, generate a random number from the stream. You are allowed to use only O(1) space and the input is in the form of stream, so can't store the previously seen numbers.

So how do we generate a random number from the whole stream such that the probability of picking any number is 1/n. with O(1) extra space? This problem is a variation of Reservoir Sampling. Here the value of k is 1.

1) Initialize 'count' as 0, 'count' is used to store count of numbers seen so far in stream.
2) For each number 'x' from stream, do following
…..**a)** Increment 'count' by 1.
…..**b)** If count is 1, set result as x, and return result.
…..**c)** Generate a random number from 0 to 'count-1'. Let the generated random number be i.
…..**d)** If i is equal to 'count – 1', update the result as x.

C/C++

```
 // An efficient C program to randomly select a number from stream of numbers.
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// A function to randomly select a item from stream[0], stream[1], .. stream[i-1]
int selectRandom(int x)
{
    static int res;    // The resultant random number
    static int count = 0;  //Count of numbers visited so far in stream
    count++;  // increment count of numbers seen so far

    // If this is the first element from stream, return it
    if (count == 1)
        res = x;
    else
    {
        // Generate a random number from 0 to count - 1
        int i = rand() % count;

        // Replace the prev random number with new number with 1/count probability
        if (i == count - 1)
            res  = x;
    }
    return res;
}

// Driver program to test above function.
int main()
{
    int stream[] = {1, 2, 3, 4};
    int n = sizeof(stream)/sizeof(stream[0]);

    // Use a different seed value for every run.
    srand(time(NULL));
    for (int i = 0; i < n; ++i)
        printf("Random number from first %d numbers is %d \n",
                                i+1, selectRandom(stream[i]));
    return 0;
}
```

Java

```
 //An efficient Java program to randomly select a number from stream of numbers.

import java.util.Random;

public class GFG
{
    static int res = 0;    // The resultant random number
    static int count = 0;  //Count of numbers visited so far in stream

    //A method to randomly select a item from stream[0], stream[1], .. stream[i-1]
    static int selectRandom(int x)
    {
        count++; // increment count of numbers seen so far

        // If this is the first element from stream, return it
        if (count == 1)
            res = x;
        else
        {
             // Generate a random number from 0 to count - 1
            Random r = new Random();
            int i = r.nextInt(count);

            // Replace the prev random number with new number with 1/count probability
            if(i == count - 1)
                res = x;
        }
        return res;
    }

    // Driver program to test above function.
    public static void main(String[] args)
    {
        int stream[] = {1, 2, 3, 4};
        int n = stream.length;
        for(int i = 0; i < n; i++)
            System.out.println("Random number from first " + (i+1) +
                               " numbers is " + selectRandom(stream[i]));
    }
}
//This code is contributed by Sumit Ghosh
```

C#

```
 // An efficient C# program to randomly
// select a number from stream of numbers.
using System;

class GFG
{
    // The resultant random number
    static int res = 0;

    // Count of numbers visited
    // so far in stream
    static int count = 0;

    // A method to randomly select
    // a item from stream[0],
    // stream[1], .. stream[i-1]
    static int selectRandom(int x)
    {
        // increment count of
        // numbers seen so far
        count++;

        // If this is the first
        // element from stream,
        // return it
        if (count == 1)
            res = x;
        else
        {
            // Generate a random number
            // from 0 to count - 1
            Random r = new Random();
            int i = r.Next(count);

            // Replace the prev random
            // number with new number
            // with 1/count probability
            if(i == count - 1)
                res = x;
        }
        return res;
    }

// Driver Code
public static void Main()
{
        int[] stream = {1, 2, 3, 4};
        int n = stream.Length;
        for(int i = 0; i < n; i++)
            Console.WriteLine("Random number from " +
                              "first {0} numbers is {1}" ,
                          i + 1, selectRandom(stream[i]));
    }
}

// This code is contributed by mits
```

PHP

```
 <?php
// An efficient php program to randomly
// select a number from stream of numbers.

// A function to randomly select a item
// from stream[0], stream[1], .. stream[i-1]
function selectRandom($x)
{
    // The resultant random number
    $res;

    // Count of numbers visited so far
    // in stream
    $count = 0;

    // increment count of numbers seen
    // so far
    $count++;

    // If this is the first element
    // from stream, return it
    if ($count == 1)
        $res = $x;
    else
    {

        // Generate a random number from
        // 0 to count - 1
        $i = rand() % $count;

        // Replace the prev random number
        // with new number with 1/count
        // probability
        if (i == $count - 1)
            $res = $x;
    }
    return $res;
}

// Driver program to test above function.
    $stream = array(1, 2, 3, 4);
    $n = sizeof($stream)/sizeof($stream[0]);

    // Use a different seed value for
    // every run.
    srand(time(NULL));

    for ($i = 0; $i < $n; ++$i)
        echo "Random number from first ",
                    $i+1, "numbers is " ,
        selectRandom($stream[$i]), "\n" ;

// This code is contributed by nitin mittal.
?>
```

Output:

```
Random number from first 1 numbers is 1
Random number from first 2 numbers is 1
Random number from first 3 numbers is 3
Random number from first 4 numbers is 4
```

Auxiliary Space: O(1)

How does this work
We need to prove that every element is picked with 1/n probability where n is the number of items seen so far. For every new stream item x, we pick a random number from 0 to 'count -1', if the picked number is 'count-1', we replace the previous result with x.

To simplify proof, let us first consider the last element, the last element replaces the previously stored result with 1/n probability. So probability of getting last element as result is 1/n.

Let us now talk about second last element. When second last element processed first time, the probability that it replaced the previous result is 1/(n-1). The probability that previous result stays when nth item is considered is (n-1)/n. So probability that the second last element is picked in last iteration is [1/(n-1)] * [(n-1)/n] which is 1/n.

Similarly, we can prove for third last element and others.

References:
Reservoir Sampling

Improved By : nitin mittal, Mithun Kumar

Source

https://www.geeksforgeeks.org/select-a-random-number-from-stream-with-o1-space/

Chapter 35

Shuffle a deck of cards

Shuffle a deck of cards - GeeksforGeeks

Given a a of cards, task is to shuffle them.

Asked in Amazon Interview

Prerequisite : Shuffle a given array

Algorithm:

```
1. First fill the array with the values in order.
2. Go through the array and exchange each element
   with the randomly chosen element in the range
   from itself to the end.

// It is possible that an element will be swap
// with itself, but there is no problem with that.
```

C++

```
 // C++ program for shuffling desk of cards.
#include <bits/stdc++.h>
using namespace std;

// Function which shuffle and print the array
void shuffle(int card[], int n)
{
    // Initialize seed randomly
    srand(time(0));

    for (int i=0; i<n ;i++)
    {
        // Random for remaining positions.
        int r = i + (rand() % (52 -i));

        swap(card[i], card[r]);
    }
}

// Driver code
int main()
{
    // Array from 0 to 51
    int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8,
               9, 10, 11, 12, 13, 14, 15,
               16, 17, 18, 19, 20, 21, 22,
               23, 24, 25, 26, 27, 28, 29,
               30, 31, 32, 33, 34, 35, 36,
               37, 38, 39, 40, 41, 42, 43,
               44, 45, 46, 47, 48, 49, 50,
               51};

    shuffle(a, 52);

    // Printing all shuffled elements of cards
    for (int i=0; i<52; i++)
        cout << a[i] << " ";
    cout << endl;

    return 0;
}
```

Java

```
 // Java Code for Shuffle a deck of cards
import java.util.Random;

class GFG {

    // Function which shuffle and print the array
    public static void shuffle(int card[], int n)
    {

        Random rand = new Random();

        for (int i = 0; i < n; i++)
        {
            // Random for remaining positions.
            int r = i + rand.nextInt(52 - i);
            //swapping the elements
            int temp = card[r];
            card[r] = card[i];
            card[i] = temp;

        }
    }

    // Driver code
    public static void main(String[] args)
    {
        // Array from 0 to 51
        int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8,
                   9, 10, 11, 12, 13, 14, 15,
                   16, 17, 18, 19, 20, 21, 22,
                   23, 24, 25, 26, 27, 28, 29,
                   30, 31, 32, 33, 34, 35, 36,
                   37, 38, 39, 40, 41, 42, 43,
                   44, 45, 46, 47, 48, 49, 50,
                   51};

        shuffle(a, 52);

        // Printing all shuffled elements of cards
        for (int i = 0; i < 52; i ++)
           System.out.print(a[i]+" ");

    }
}
// This code is contributed by Arnav Kr. Mandal
```

C#

```
 // C# Code for Shuffle a deck of cards
using System;

class GFG {

    // Function which shuffle and
    // print the array
    public static void shuffle(int []card,
                               int n)
    {

        Random rand = new Random();

        for (int i = 0; i < n; i++)
        {
            // Random for remaining positions.
            int r = i + rand.Next(52 - i);

            //swapping the elements
            int temp = card[r];
            card[r] = card[i];
            card[i] = temp;

        }
    }

    // Driver code
    public static void Main()
    {
        // Array from 0 to 51
        int []a = {0, 1, 2, 3, 4, 5, 6, 7, 8,
                   9, 10, 11, 12, 13, 14, 15,
                   16, 17, 18, 19, 20, 21, 22,
                   23, 24, 25, 26, 27, 28, 29,
                   30, 31, 32, 33, 34, 35, 36,
                   37, 38, 39, 40, 41, 42, 43,
                   44, 45, 46, 47, 48, 49, 50,
                   51};

        shuffle(a, 52);

        // Printing all shuffled elements of cards
        for (int i = 0; i < 52; i ++)
        Console.Write(a[i]+" ");

    }
}

// This code is contributed by Nitin Mittal.
```

Output:

```
29 27 20 23 26 21 35 51 15 18 46 32 33 19
24 30 3 45 40 34 16 11 36 50 17 10 7 5 4
39 6 47 38 28 13 44 49 1 8 42 43 48 0 12
37 41 25 2 31 14 22
```

Note : Output will be different each time because of the random function used in the program.
Please refer Shuffle a given array for details.

Improved By : nitin mittal

Source

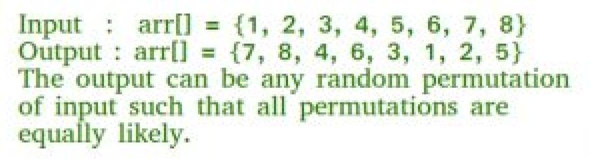
https://www.geeksforgeeks.org/shuffle-a-deck-of-cards-3/

Chapter 36

Shuffle a given array

Shuffle a given array - GeeksforGeeks

Given an array, write a program to generate a random permutation of array elements. This question is also asked as “shuffle a deck of cards” or “randomize a given array”. Here shuffle means that every permutation of array element should equally likely.



```
Input  :  arr[] = {1, 2, 3, 4, 5, 6, 7, 8}
Output : arr[] = {7, 8, 4, 6, 3, 1, 2, 5}
The output can be any random permutation
of input such that all permutations are
equally likely.
```

Let the given array be *arr[]*. A simple solution is to create an auxiliary array *temp[]* which is initially a copy of *arr[]*. Randomly select an element from *temp[]*, copy the randomly selected element to *arr[0]* and remove the selected element from *temp[]*. Repeat the same process n times and keep copying elements to *arr[1], arr[2], … .* The time complexity of this solution will be O(n^2).

Fisher–Yates shuffle Algorithm works in O(n) time complexity. The assumption here is, we are given a function rand() that generates random number in O(1) time.
The idea is to start from the last element, swap it with a randomly selected element from the whole array (including last). Now consider the array from 0 to n-2 (size reduced by 1), and repeat the process till we hit the first element.

Following is the detailed algorithm

```
To shuffle an array a of n elements (indices 0..n-1):
  for i from n - 1 downto 1 do
       j = random integer with 0 <= j <= i
       exchange a[j] and a[i]
```

Following is implementation of this algorithm.

C

```
 // C Program to shuffle a given array

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// A utility function to swap to integers
void swap (int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// A utility function to print an array
void printArray (int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// A function to generate a random permutation of arr[]
void randomize ( int arr[], int n )
{
    // Use a different seed value so that we don't get same
    // result each time we run this program
    srand ( time(NULL) );

    // Start from the last element and swap one by one. We don't
    // need to run for the first element that's why i > 0
    for (int i = n-1; i > 0; i--)
    {
        // Pick a random index from 0 to i
        int j = rand() % (i+1);

        // Swap arr[i] with the element at random index
        swap(&arr[i], &arr[j]);
    }
}

// Driver program to test above function.
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
    int n = sizeof(arr)/ sizeof(arr[0]);
    randomize (arr, n);
    printArray(arr, n);

    return 0;
}
```

Java

```
 // Java Program to shuffle a given array
import java.util.Random;
import java.util.Arrays;
public class ShuffleRand
{
    // A Function to generate a random permutation of arr[]
    static void randomize( int arr[], int n)
    {
        // Creating a object for Random class
        Random r = new Random();

        // Start from the last element and swap one by one. We don't
        // need to run for the first element that's why i > 0
        for (int i = n-1; i > 0; i--) {

            // Pick a random index from 0 to i
            int j = r.nextInt(i);

            // Swap arr[i] with the element at random index
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
        // Prints the random array
        System.out.println(Arrays.toString(arr));
    }

    // Driver Program to test above function
    public static void main(String[] args)
    {

         int[] arr = {1, 2, 3, 4, 5, 6, 7, 8};
         int n = arr.length;
         randomize (arr, n);
    }
}
// This code is contributed by Sumit Ghosh
```

Python

```
 # Python Program to shuffle a given array
import random

# A function to generate a random permutation of arr[]
def randomize (arr, n):
    # Start from the last element and swap one by one. We don't
    # need to run for the first element that's why i > 0
    for i in range(n-1,0,-1):
        # Pick a random index from 0 to i
        j = random.randint(0,i)

        # Swap arr[i] with the element at random index
        arr[i],arr[j] = arr[j],arr[i]
    return arr

# Driver program to test above function.
arr = [1, 2, 3, 4, 5, 6, 7, 8]
n = len(arr)
print(randomize(arr, n))


# This code is contributed by Pratik Chhajer
```

C#

```
 // C# Code for Number of digits
// in the product of two numbers
using System;

class GFG
{
// A Function to generate a
// random permutation of arr[]
    static void randomize(int []arr, int n)
    {
        // Creating a object
        // for Random class
        Random r = new Random();

        // Start from the last element and
        // swap one by one. We don't need to
        // run for the first element
        // that's why i > 0
        for (int i = n - 1; i > 0; i--)
        {

            // Pick a random index
            // from 0 to i
            int j = r.Next(0, i);

            // Swap arr[i] with the
            // element at random index
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
        // Prints the random array
        for (int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");
    }


// Driver Code
static void Main()
{
    int[] arr = {1, 2, 3, 4,
                 5, 6, 7, 8};
    int n = arr.Length;
    randomize (arr, n);
}
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP Program to shuffle
// a given array

// A function to generate
// a random permutation of arr[]
function randomize ($arr, $n)
{
    // Start from the last element
    // and swap one by one. We
    // don't need to run for the
    // first element that's why i > 0
    for($i = $n - 1; $i >= 0; $i--)
    {
        // Pick a random index
        // from 0 to i
        $j = rand(0, $i);

        // Swap arr[i] with the
        // element at random index
        $tmp = $arr[$i];
        $arr[$i] = $arr[$j];
        $arr[$j] = $tmp;
    }
    for($i = 0; $i < $n; $i++)
    echo $arr[$i]." ";
}

// Driver Code
$arr = array(1, 2, 3, 4,
             5, 6, 7, 8);
$n = count($arr);
randomize($arr, $n);

// This code is contributed by mits
?>
```

Output :

```
7 8 4 6 3 1 2 5
```

The above function assumes that rand() generates a random number.

Time Complexity: O(n), assuming that the function rand() takes O(1) time.

How does this work?
The probability that ith element (including the last one) goes to last position is 1/n, because we randomly pick an element in first iteration.

The probability that ith element goes to second last position can be proved to be 1/n by dividing it in two cases.
Case 1: i = n-1 (index of last element):
The probability of last element going to second last position is = (probability that last element doesn't stay at its original position) x (probability that the index picked in previous step is picked again so that the last element is swapped)
So the probability = ((n-1)/n) x (1/(n-1)) = 1/n
Case 2: 0 < i < n-1 (index of non-last):
The probability of ith element going to second position = (probability that ith element is not picked in previous iteration) x (probability that ith element is picked in this iteration)
So the probability = ((n-1)/n) x (1/(n-1)) = 1/n

We can easily generalize above proof for any other position.

Improved By : Sam007, Mithun Kumar

Source

https://www.geeksforgeeks.org/shuffle-a-given-array/

Chapter 37

Shuffle or Randomize a list in Java

Shuffle or Randomize a list in Java - GeeksforGeeks

- **Shuffling a list**
  Collections.shuffle() is used to shuffle lists in java.

  Class hierarchy:

```
java
    util
       Collections
```

  Syntax:

```
Collections.shuffle(list);
```

  Examples:

```
 // Java program to demonstrate working of shuffle()
import java.util.*;

public class GFG {
    public static void main(String[] args)
    {
        ArrayList<String> mylist = new ArrayList<String>();
        mylist.add("ide");
        mylist.add("quiz");
        mylist.add("geeksforgeeks");
        mylist.add("quiz");
        mylist.add("practice");
        mylist.add("qa");

        System.out.println("Original List : \n" + mylist);

        Collections.shuffle(mylist);

        System.out.println("\nShuffled List : \n" + mylist);
    }
}
```

Output:

```
Original List :
[ide, quiz, geeksforgeeks, quiz, practice, qa]

Shuffled List :
[ide, practice, quiz, qa, geeksforgeeks, quiz]
```

- **Shuffling a list using user provided Random Obbject**
  Syntax:

```
Collections.shuffle(list, Random object);
```

Examples:

```
 // Java program to demonstrate working of shuffle()
// with user provided source of randomness.
import java.util.*;

public class GFG {
    public static void main(String[] args)
    {
        ArrayList<String> mylist = new ArrayList<String>();
        mylist.add("ide");
        mylist.add("quiz");
        mylist.add("geeksforgeeks");
        mylist.add("quiz");
        mylist.add("practice");
        mylist.add("qa");

        System.out.println("Original List : \n" + mylist);
        // Here we use Random() to shuffle given list.
        Collections.shuffle(mylist, new Random());
        System.out.println("\nShuffled List with Random() : \n"
                           + mylist);

        // Here we use Random(3) to shuffle given list. Here 3
        // is seed value for Random.
        Collections.shuffle(mylist, new Random(3));
        System.out.println("\nShuffled List with Random(3) : \n"
                           + mylist);

        // Here we use Random(5) to shuffle given list.  Here 5
        // is seed value for Random.
        Collections.shuffle(mylist, new Random(5));
        System.out.println("\nShuffled List with Random(5) : \n"
                           + mylist);
    }
}
```

Output:

```

Original List :
[ide, quiz, geeksforgeeks, quiz, practice, qa]

Shuffled List with Random() :
[geeksforgeeks, practice, qa, ide, quiz, quiz]

Shuffled List with Random(3) :
[quiz, ide, practice, quiz, geeksforgeeks, qa]

Shuffled List with Random(5) :
[ide, quiz, geeksforgeeks, quiz, practice, qa]
```

- **How to write our own Shuffle method?**
  We can use Fisher–Yates shuffle Algorithmthat works in O(n) time.

```
 // Java Program to shuffle a given array
import java.util.Random;
import java.util.Arrays;
public class ShuffleRand
{
    // A Function to generate a random permutation of arr[]
    static void randomize( int arr[], int n)
    {
        // Creating a object for Random class
        Random r = new Random();
        // Start from the last element and swap one by one. We don't
        // need to run for the first element that's why i > 0
        for (int i = n-1; i > 0; i--) {

            // Pick a random index from 0 to i
            int j = r.nextInt(i);

            // Swap arr[i] with the element at random index
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }

        // Prints the random array
        System.out.println(Arrays.toString(arr));
    }

    // Driver Program to test above function
    public static void main(String[] args)
    {
         int[] arr = {1, 2, 3, 4, 5, 6, 7, 8};
         int n = arr.length;
         randomize (arr, n);
    }
}
```

Output:

```
[5, 7, 1, 3, 6, 8, 4, 2]
```

Source

https://www.geeksforgeeks.org/shuffle-or-randomize-a-list-in-java/

Chapter 38

Strong Password Suggester Program

Strong Password Suggester Program - GeeksforGeeks

Given a password entered by the user, check its strength and suggest some password if it is not strong.

Criteria for strong password is as follows :
A password is strong if it has :
1. At least 8 characters
2. At least one special char
3. At least one number
4. At least one upper and one lower case char.

Examples :

```
Input : keshav123
Output : Suggested Password
k(eshav12G3
keshav1@A23
kesh!Aav123
ke(shav12V3
keGshav1$23
kesXhav@123
keAshav12$3
kesKhav@123
kes$hav12N3
$kesRhav123

Input :rakesh@1995kumar
Output : Your Password is Strong
```

Approach :
Check the input password for its strength if it fulfills all the criteria, then it is strong password else check for the absent characters in it, and return the randomly generated password to the user.

CPP

```
 // CPP code to suggest strong password
#include <bits/stdc++.h>

using namespace std;

// adding more characters to suggest strong password
string add_more_char(string str, int need)
{
    int pos = 0;

    // all 26 letters
    string low_case = "abcdefghijklmnopqrstuvwxyz";

    for (int i = 0; i < need; i++) {
        pos = rand() % str.length();
        str.insert(pos, 1, low_case[rand() % 26]);
    }
    return str;
}

// make powerfull string
string suggester(int l, int u, int d, int s, string str)
{

    // all digits
    string num = "0123456789";

    // all lower case, uppercase and special characters
    string low_case = "abcdefghijklmnopqrstuvwxyz";
    string up_case = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    string spl_char = "@#$_()!";

    // position at which place a character
    int pos = 0;

    // if there is no lowercase char in input string, add it
    if (l == 0) {
        // generate random integer under string length
        pos = rand() % str.length();

        // generate random integer under 26 for indexing of a to z
        str.insert(pos, 1, low_case[rand() % 26]);
    }

    // if there is no upper case char in input string, add it
    if (u == 0) {
        pos = rand() % str.length();
        str.insert(pos, 1, up_case[rand() % 26]);
    }

    // if there is no digit in input string, add it
    if (d == 0) {
        pos = rand() % str.length();
        str.insert(pos, 1, num[rand() % 10]);
    }

    // if there is no special character in input string, add it
    if (s == 0) {
        pos = rand() % str.length();
        str.insert(pos, 1, spl_char[rand() % 7]);
    }

    return str;
}

/* make_password function :This function is used to check
strongness and if input string is not strong, it will suggest*/
void generate_password(int n, string p)
{
    // flag for lower case, upper case, special
    // characters and need of more characters
    int l = 0, u = 0, d = 0, s = 0, need = 0;

    // password suggestions.
    string suggest;

    for (int i = 0; i < n; i++) {
        // password suggestions.
        if (p[i] >= 97 && p[i] <= 122)
            l = 1;
        else if (p[i] >= 65 && p[i] <= 90)
            u = 1;
        else if (p[i] >= 48 && p[i] <= 57)
            d = 1;
        else
            s = 1;
    }

    // Check if input string is strong that
    // means all flag are active.
    if ((l + u + d + s) == 4) {
        cout << "Your Password is Strong" << endl;
        return;
    }
    else
        cout << "Suggested passowrd " << endl;

    /*suggest 10 strong strings */
    for (int i = 0; i < 10; i++) {
        suggest = suggester(l, u, d, s, p);
        need = 8 - suggest.length();
        if (need > 0)
            suggest = add_more_char(suggest, need);
        cout << suggest << endl;
    }
}

// Driver code
int main()
{
    string input_string = "geek@2018";
    srand(time(NULL));

    // srand function set Seed for rand().
    generate_password(input_string.length(), input_string);
    return 0;
}
```

Java

```
 // Java code to suggest strong password
import java.io.*;
import java.util.*;
import java.util.Random;

public class GFG {

    // adding more characters to suggest
    // strong password
    static StringBuilder add_more_char(
                        StringBuilder str, int need)
    {
        int pos = 0;
        Random randm = new Random();

        // all 26 letters
        String low_case = "abcdefghijklmnopqrstuvwxyz";
        for (int i = 0; i < need; i++) {
            pos = randm.nextInt(1000) % str.length();
            str.setCharAt(pos,low_case.charAt(
                            randm.nextInt(1000) % 26));
        }
        return str;
    }

    // make powerfull String
    static StringBuilder suggester(int l, int u, int d,
                              int s, StringBuilder str)
    {
        Random randm = new Random();

        // all digits
        String num = "0123456789";

        // all lower case, uppercase and special
        // characters
        String low_case = "abcdefghijklmnopqrstuvwxyz";
        String up_case = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        String spl_char = "@#$_()!";

        // position at which place a character
        int pos = 0;

        // if there is no lowercase char in input
        // String, add it
        if (l == 0) {

            // generate random integer under
            // String length()
            pos = randm.nextInt(1000) % str.length();

            // generate random integer under 26 for
            // indexing of a to z
            str.setCharAt(pos,low_case.charAt(randm.nextInt(1000)
                                        % 26));
        }

        // if there is no upper case char in input
        // String, add it
        if (u == 0) {
            pos = randm.nextInt(1000) % str.length();
            str.setCharAt(pos,low_case.charAt(randm.nextInt(1000)
                                        % 26));
        }
        // if there is no digit in input String, add it
        if (d == 0) {
            pos = randm.nextInt(1000) % str.length();
            str.setCharAt(pos,low_case.charAt(randm.nextInt(1000)
                                        % 10));
        }

        // if there is no special character in input
        // String, add it
        if (s == 0) {
            pos = randm.nextInt(1000) % str.length();
            str.setCharAt(pos,low_case.charAt(randm.nextInt(1000)
                                        % 7));
        }

        return str;
    }

    /* make_password function :This function is used
    to check strongness and if input String is not
    strong, it will suggest*/
    static void generate_password(int n, StringBuilder p)
    {

        // flag for lower case, upper case, special
        // characters and need of more characters
        int l = 0, u = 0, d = 0, s = 0, need = 0;

        // password suggestions.
        StringBuilder suggest;

        for (int i = 0; i < n; i++) {

            // password suggestions.
            if (p.charAt(i) >= 97 && p.charAt(i) <= 122)
                l = 1;
            else if (p.charAt(i) >= 65 && p.charAt(i) <= 90)
                u = 1;
            else if (p.charAt(i) >= 48 && p.charAt(i) <= 57)
                d = 1;
            else
                s = 1;
        }

        // Check if input String is strong that
        // means all flag are active.
        if ((l + u + d + s) == 4) {
            System.out.println("Your Password is Strong");
            return;
        }
        else
            System.out.println("Suggested password ");

        /*suggest 10 strong Strings */
        for (int i = 0; i < 10; i++) {
            suggest = suggester(l, u, d, s, p);
            need = 8 - suggest.length();
            if (need > 0)
                suggest = add_more_char(suggest, need);
            System.out.println(suggest);;
        }
    }

    // Driver code
    public static void main(String[] args)
    {
        StringBuilder input_String = new StringBuilder("geek@2018");
        generate_password(input_String.length(), input_String);
    }
}

// This code is contributed by Manish Shaw (manishshaw1)
```

C#

```
 // C# code to suggest strong password
using System;
using System.Collections.Generic;

class GFG {

    // adding more characters to suggest
    // strong password
    static string add_more_char(string str, int need)
    {
        int pos = 0;
        Random randm = new Random();

        // all 26 letters
        string low_case = "abcdefghijklmnopqrstuvwxyz";

        for (int i = 0; i < need; i++) {
            pos = randm.Next(1,1000) % str.Length;
            str = str.Insert(pos,low_case[randm.Next(1000)
                                        % 26].ToString());
        }
        return str;
    }

    // make powerfull string
    static string suggester(int l, int u, int d, int s,
                                             string str)
    {
        Random randm = new Random();

        // all digits
        string num = "0123456789";

        // all lower case, uppercase and special
        // characters
        string low_case = "abcdefghijklmnopqrstuvwxyz";
        string up_case = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        string spl_char = "@#$_()!";

        // position at which place a character
        int pos = 0;

        // if there is no lowercase char in input
        // string, add it
        if (l == 0) {

            // generate random integer under
            // string length
            pos = randm.Next(1,1000) % str.Length;

            // generate random integer under 26 for
            // indexing of a to z
            str = str.Insert(pos,low_case[randm.Next(1,
                                 1000) % 26].ToString());
        }

        // if there is no upper case char in input
        // string, add it
        if (u == 0) {
            pos = randm.Next(1,1000) % str.Length;
            str = str.Insert(pos,up_case[randm.Next(
                              1,1000) % 26].ToString());
        }

        // if there is no digit in input string, add it
        if (d == 0) {
            pos = randm.Next(1,1000) % str.Length;
            str = str.Insert(pos,num[randm.Next(1,1000)
                                      % 10].ToString());
        }

        // if there is no special character in input
        // string, add it
        if (s == 0) {
            pos = randm.Next(1,1000) % str.Length;
            str = str.Insert(pos,spl_char[randm.Next(
                              1,1000) % 7].ToString());
        }

        return str;
    }

    /* make_password function :This function is used
    to check strongness and if input string is not
    strong, it will suggest*/
    static void generate_password(int n, string p)
    {

        // flag for lower case, upper case, special
        // characters and need of more characters
        int l = 0, u = 0, d = 0, s = 0, need = 0;

        // password suggestions.
        string suggest;

        for (int i = 0; i < n; i++) {

            // password suggestions.
            if (p[i] >= 97 && p[i] <= 122)
                l = 1;
            else if (p[i] >= 65 && p[i] <= 90)
                u = 1;
            else if (p[i] >= 48 && p[i] <= 57)
                d = 1;
            else
                s = 1;
        }

        // Check if input string is strong that
        // means all flag are active.
        if ((l + u + d + s) == 4) {
            Console.WriteLine("Your Password is Strong\n");
            return;
        }
        else
            Console.WriteLine("Suggested password\n ");
        /*suggest 10 strong strings */
        for (int i = 0; i < 10; i++) {
            suggest = suggester(l, u, d, s, p);
            need = 8 - suggest.Length;
            if (need > 0)
                suggest = add_more_char(suggest, need);
            Console.WriteLine(suggest + "\n");;
        }
    }

    // Driver code
    public static void Main()
    {
        string input_string = "geek@2018";
        generate_password(input_string.Length, input_string);
    }
}

// This code is contributed by Manish Shaw (manishshaw1)
```

PHP

```
 <?php
// php code to suggest
// strong password

// adding more characters to
// suggest strong password
function add_more_char( $str, $need)
{
    $pos = 0;

    // all 26 letters
    $low_case = "abcdefghijklmnopqrstuvwxyz";

    for ($i = 0; $i < $need; $i++)
    {
        $pos = rand() % strlen($str);
        $str[$pos]=$low_case[rand() % 26];
    }
    return $str;
}

// make powerfull string
function suggester($l, $u, $d, $s, $str)
{
```

```
// all digits
$num = "0123456789";

// all lower case, uppercase and
// special characters
$low_case = "abcdefghijklmnopqrstuvwxyz";
$up_case = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

// $spl_char1 = "@#$_()!";
// position at which place
// a character
$pos = 0;

// if there is no lowercase char
// in input string, add it
if ($l == 0)
{

    // generate random integer
    // under string length
    $pos = rand() % strlen($str);

    // generate random integer under
    // 26 for indexing of a to z
    $str[$pos]=$low_case[rand() % 26];
}

// if there is no upper case
// char in input string, add it
if ($u == 0)
{
    $pos = rand() % strlen($str);
    $str[$pos]=$up_case[rand() % 26];
}

// if there is no digit
// in input string, add it
if ($d == 0) {
    $pos = rand() % strlen($str);
    $str[$pos]=$num[rand() % 10];
}

// if there is no special character
// in input string, add it
if ($s == 0)
{
    $pos = rand() % strlen($str);
    $str[$pos]=$spl_char1[rand() % 7];
    }

    return $str;
}

// Make_password function : This
// function is used to check
// strongness and if input string
// is not strong, it will suggest
function generate_password($n, $p)
{

    // flag for lower case, upper case,
    // special characters and need
    // of more characters
    $l = 0;
    $u = 0;
    $d = 0;
    $s = 0;
    $need = 0;

    // password suggestions.
    $suggest;

    for ($i = 0; $i < $n; $i++)
    {
        // password suggestions.
        if ($p[$i] >= 97 && $p[$i] <= 122)
            $l = 1;
        else if ($p[$i] >= 65 && $p[$i] <= 90)
            $u = 1;
        else if ($p[$i] >= 48 && $p[$i] <= 57)
            $d = 1;
        else
            $s = 1;
    }

    // Check if input string is strong
    // that means all flag are active.
    if (($l + $u + $d + $s) == 4)
    {
        echo "Your Password is Strong.\n";
        return;
    }
    else
        echo "Suggested passowrd";

    // suggest 10 strong strings
    for ($i = 0; $i < 10; $i++)
    {
        $suggest = suggester($l, $u, $d, $s, $p);
        $need = 8 - strlen($suggest);
        if ($need > 0)
            $suggest = add_more_char($suggest, $need);
        echo "\n".$suggest;
    }
}

    // Driver code
    $input_string = "geek@2018";
    srand(mktime(NULL));

    // srand function set Seed for rand().
    generate_password(strlen($input_string),
                      $input_string);

// This code is contributed by mits
?>
```

Output:

```
Suggested password
geek@201K8
geek@201S8
g0eek@2018
geek@201N8
geek@2P018
geek@D2018
geUek@2018
geek@2Q018
geek@2F018
geekZ@2018
```

Time complexity : O(n).

Improved By : Mithun Kumar, manishshaw1

Source

https://www.geeksforgeeks.org/strong-password-suggester-program/

Chapter 39

Write a function that generates one of 3 numbers according to given probabilities

Write a function that generates one of 3 numbers according to given probabilities - GeeksforGeeks

You are given a function rand(a, b) which generates equiprobable random numbers between [a, b] inclusive. Generate 3 numbers x, y, z with probability P(x), P(y), P(z) such that P(x) + P(y) + P(z) = 1 using the given rand(a,b) function.

The idea is to utilize the equiprobable feature of the rand(a,b) provided. ***Let the given probabilities be in percentage form, for example P(x)=40%, P(y)=25%, P(z)=35%.***.

Following are the detailed steps.
1) Generate a random number between 1 and 100. Since they are equiprobable, the probability of each number appearing is 1/100.
2) Following are some important points to note about generated random number 'r'.
a) 'r' is smaller than or equal to P(x) with probability P(x)/100.
b) 'r' is greater than P(x) and smaller than or equal P(x) + P(y) with P(y)/100.
c) 'r' is greater than P(x) + P(y) and smaller than or equal 100 (or P(x) + P(y) + P(z)) with probability P(z)/100.

```
 // This function generates 'x' with probability px/100, 'y' with
// probability py/100  and 'z' with probability pz/100:
// Assumption: px + py + pz = 100 where px, py and pz lie
// between 0 to 100
int random(int x, int y, int z, int px, int py, int pz)
{
        // Generate a number from 1 to 100
        int r = rand(1, 100);
        // r is smaller than px with probability px/100
        if (r <= px)
            return x;

         // r is greater than px and smaller than or equal to px+py
         // with probability py/100
        if (r <= (px+py))
            return y;

         // r is greater than px+py and smaller than or equal to 100
         // with probability pz/100
        else
            return z;
}
```

This function will solve the purpose of generating 3 numbers with given three probabilities.

This article is contributed by **Harsh Agarwal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/write-a-function-to-generate-3-numbers-according-to-given-probabilities/