

**MAY  
2021**

**Vol.7  
Issue 1,**

# CORETECHRA



**A TECHNICAL  
MAGAZINE**

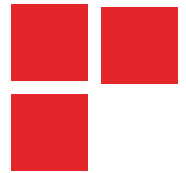


**KARPAGAM**

**INSTITUTE OF TECHNOLOGY**

INSPIRING INNOVATION

# ABOUT OUR DEPARTMENT



## OUR MISSION

To produce technically competent and skilled engineers to meet the challenges of the IT industry

## OUR VISION

- Establishing innovative teaching learning practices with competent faculty and state of the art facilities.
- Enriching knowledge on latest technological advancements through industrial collaborations.
- Developing moral and ethical values through extension activities.

## PROGRAM EDUCATIONAL OBJECTIVES (PEOS)

### PEO 1:

The graduates will have a successful career in the field Information Technology and related domains.

### PEO 2:

The graduates will provide solutions by applying analytical skills for the real-world problems in IT Industry.

### PEO 3:

The graduates will involve in lifelong learning and as part of team in multidisciplinary projects with ethical values.

## PROGRAM SPECIFIC OUTCOMES (PSOS)

### PSO 1:

Identify, formulate and solve engineering problems by applying programming concepts and algorithmic principles in the field of IT.

### PSO 2:

Analyze, design and develop Software applications, Networking and Data Management technologies for efficient IT based systems.

# Table of Contents

1.	A C# LIBRARY FOR PHYSICAL UNITS	01
2.	CRYPTOGRAPHIC HASHES	05
3.	LAZY LOADING	08
4.	Just Try...,,	11
5.	LiteDB Database	13
6.	PATHFINDING ALGORITHMS IN C#	15
7.	RANDOMIZATION AND SAMPLING METHODS	19
8	PROJECT GLIMPSES	25
9.	JustTry it.., Solutions	27

# A C# LIBRARY FOR PHYSICAL UNITS

## Introduction:

The loss of NASA's Mars Climate Orbiter in 1999 served as the project's initial source of inspiration. Due to a conversion error between metric (SI) and US Customary Units, this failed to enter Mars orbit. Measurements in pound-force seconds were being provided by one subsystem to another subsystem that was expecting Newton seconds. The probe came too close to the planet's atmosphere as it braked to enter orbit and either burned up or ricocheted out into solar orbit.



Therefore, I made an effort to create a code library where this kind of error should be prevented by design. It has these characteristics:

- Numerous common physics and engineering calculations can be carried out with it.
- All quantities have a corresponding physical dimension, such as length or mass, because it is based on dimensional analysis.
- Quantities of different dimensions can only be combined in ways that are supported by science since it is firmly coded.
- Internally, S.I. (metric) units are used to store all values.
- Only at its exterior interfaces, such as when converting to and from strings, are values transformed to a specific system of units.

It is created using the .NET 5.0 framework and C# version 9.

Here is an illustration of how to utilise the library:

```
// Tsolkovsky rocket equation
Mass EmptyMass = 120000.Kilograms();
Mass PropellantMass = 1200000.Kilograms();
Mass WetMass = EmptyMass + PropellantMass;
```

```
Velocity ExhaustVelocity = 3700.MetresPerSecond();
Velocity DeltaV = ExhaustVelocity * Math.Log(WetMass / EmptyMass);
// DeltaV = 8872.21251 m/s
```

I've used examples from my old grammar school physics textbook, Nelkon and Parker Advanced Level Physics, in this article as well as the sample code and unit tests. Throughout the 1960s and 1970s, this book served as the de facto sixth form physics textbook in Britain.

## Background:

Dimensional and unit notions serve as the foundation of the library.

## Dimensions:

A physical quantity's Dimension governs how it relates to a group of basic quantities like mass, length, and time. M, L, T, etc. are frequently used as shorthand for these. These fundamental dimensions can be multiplied and divided to create new dimensions. So:

- Area = Length x Length =  $L^2$
- Volume = Length x Length x Length =  $L^3$
- Density = Mass / Volume =  $M/L^3 = ML^{-3}$
- Velocity = Length / Time =  $L/T = LT^{-1}$
- Acceleration = velocity / Time =  $LT^{-2}$
- Force = Mass \* Acceleration =  $MLT^{-2}$

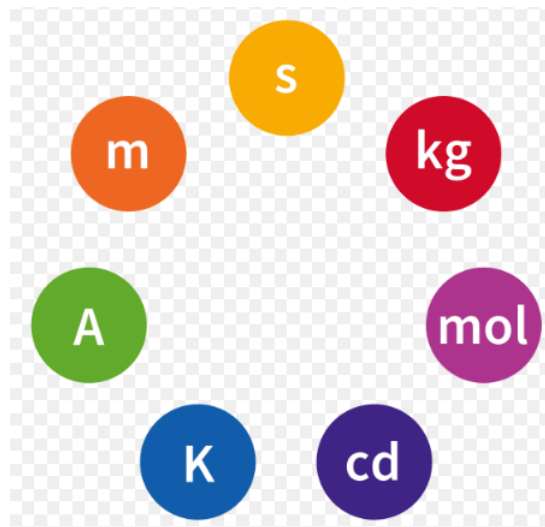
A series of powers of the fundamental dimensions can be used to describe the dimension of any given quantity (for example, Force =  $MLT^{-2}$  above). If the dimensions of two quantities do not match, trying to add or subtract such quantities is invalid. So, for example, adding a mass to a volume is not valid. The following fundamental dimensions are used by the International System of Units (S.I.):

Dimension	Symbol	Unit	Unit Symbol
Mass	M	kilogramme	kg
Length	L	metre	m
Time	T	second	s
Electric Current	I	ampere	A
Thermodynamic Temperature	$\Theta$	kelvin	K
Amount of Substance	N	mole	mol
Luminous Intensity	J	candela	cd

## Units:

Different basic units can be defined by a unit system to correspond to various dimensions. So, whereas the S.I. system uses kilogrammes as the unit of mass, the American and British systems utilise pounds. Similarly, the unit of length is the foot rather than the metre. The American and British systems differ from one another when it comes to how volume is measured. The other fundamental dimensions have the same units across all three systems.

It is possible to generate and utilise new definitions even though the library already provides ones for the S.I., American, and British systems. For instance, you may develop a system based on the Japanese shakkanho system, which uses the kan ( ) for mass and the shaku ( ) for length.



## Employing the Code:

A Visual Studio solution containing two projects—the library itself and a command-line programme that tests and illustrates the library's features—makes up the provided code in the ZIP file that is attached. Add the library project file to "KMB. Library. Units KMB. Library. Units.csproj" to use it in your own project. Then, add the using statements listed below to your code:

```
using KMB.Library.Units;  
using KMB.Library.Units.Metric;  
using KMB.Library.Units.TimeUnits;           // for hours, minutes etc.  
using KMB.Library.Units.British;             // For feet and pounds. Or use USA if you  
prefer
```

## Materials in the Library:

Numerous classes and interfaces are defined in the Units library. Here are the main ones covered:

### Dimension Class:

This class is used to represent one or more physical dimensions. For the power of each dimension, it has a read-only field:

```
public readonly short M; // Mass
public readonly short L; // Length
public readonly short T; // Time
public readonly short I; // Current
public readonly short Θ; // Temperature
public readonly short N; // Amount of Substance
public readonly short J; // Luminous Intensity
public readonly short A; // Angle.
```

International System of Units - SI	
K	▶ kelvin (temperature)
m	▶ meter (distance)
A	▶ ampere (electric current)
s	▶ second (time)
mol	▶ mole (amount of substance)
kg	▶ kilogram (mass)
cd	▶ candela (intensity of light)

Take note of the angle value. Although technically speaking, angles lack dimensions, it is more practical to treat them as if they did. This allows us to distinguish between angles and dimensionless values, for example, when converting to a string.

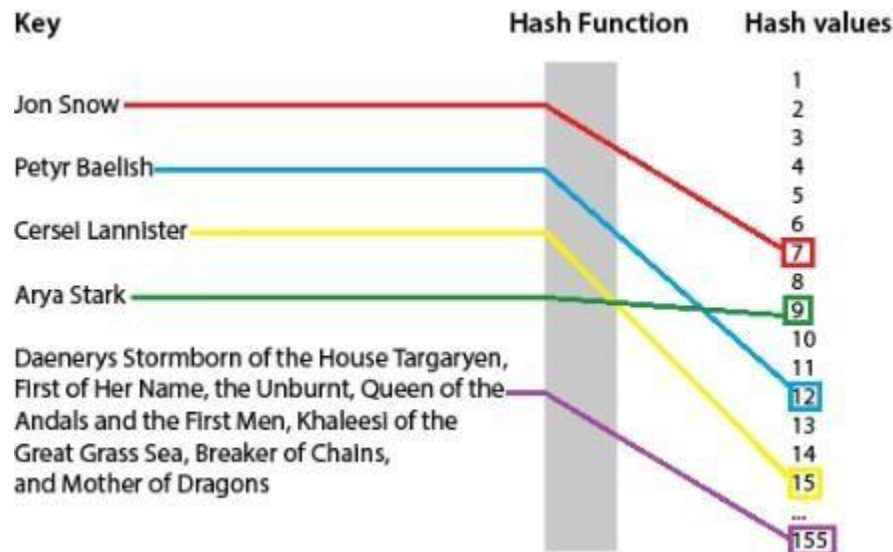
**ARULKUMAR VP**

**AP/IT**

# CRYPTOGRAPHIC HASHES

## What Are Hashes – Introduction:

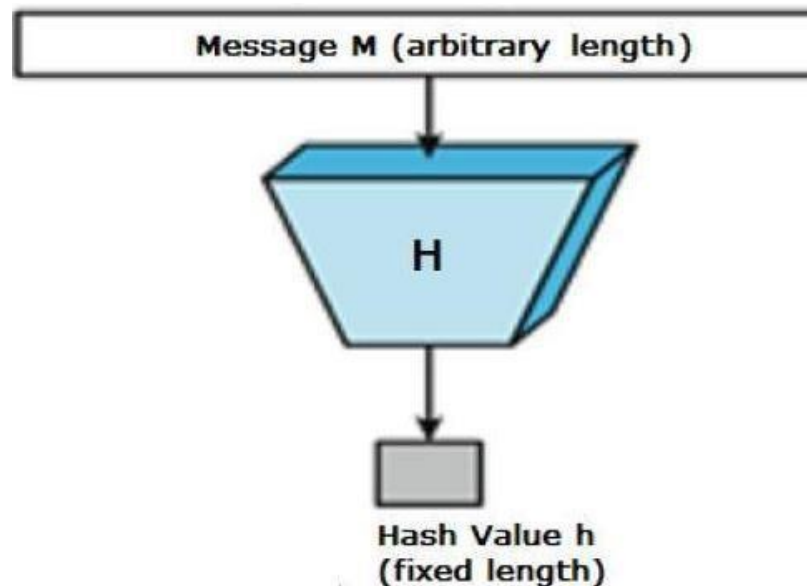
A hash function is a function that converts data of any size to data of a specific size. Consider the illustration below as an illustration:



- ❖ This hash function converts names of any length into integers. In order to determine the corresponding integer, this hash function will simply count the number of characters in the name. It should be noted that examples of different keys that will be mapped into the same integer are easy to locate for this function.
- ❖ Hash tables are a type of data structure seen in some contemporary programming languages like Ruby. They are used to build dictionaries, which map keys to values. These tables compute an index into an array of slots using hash functions. When combined with a strong hash function, hash tables can be extremely effective.
- ❖ A cryptographic hash function is a one-way, or non-invertible, function. This indicates that by focusing simply on the function's output, it is almost impossible to replicate the input of the function (usually referred to as message) (called message digest). Cryptographic hash functions, in contrast to ordinary hash functions, are exceedingly difficult to invert even if the attacker is aware of the theory and the method in use. An



attacker shouldn't know anything about the original message, not even its size, given simply the hash (which, obviously, is not the case of the first example).



**Additionally, cryptographic hash functions possess the following features:**

- The computation of the hash value of any given message is simple computationally.
- Integrity of the message: The message cannot be changed without also changing the message digest.
- Collision resistance: Finding two different messages that create the identical digest is not possible.

Here are some of the most popular hashing operations:

- MD5: A 128-bit digest size that was introduced in 1992. It is now regarded as cryptographically flawed.
- SHA-1, which was created by the NSA and standardized in 1995. The digest is 160 bits in size. New projects should use SHA-2 or SHA-3 instead because it is no longer thought to be secure. By 2017, the browsers for Internet Explorer, Chrome, and Firefox will all no longer support SHA-1 SSL certificates. The first SHA-1 collision was discovered in February 2017, and according to CWI Amsterdam and Google, this "emphasizes the necessity of sun setting SHA-1 usage." In the section on Practical Examples, we'll see evidence for this collision ourselves.
- Six cryptographic hash algorithms in the SHA-2 family come in four different digest sizes: 224-bit, 256-bit, 384-bit, and 512-bit. was also created by the NSA and released in 2001 as a federal standard in the United States (FIPS).

- NIST published the SHA-3 standard in August 2015. Although they play a significant part in Public Key Cryptography, we will focus on the usage of cryptographic hash functions in computing and verifying checksums in this article.

**Security:** You can tell if a file you obtained from the internet is real, or that it hasn't been tampered with, by performing a hash operation on it, then checking to see if the digest you generated matches the one published on the internet. However, one could argue that receiving hashes from the same website as your files is not particularly secure because a hacker who has altered the file is likely also able to alter the hash listing. Hash listings can be found on HTTPS-secured websites and in PGP-signed emails from mailing list announcements.

**Integrity:** Due to the fact that even the smallest change to the original file during transmission would result in a completely different digest, hash values are useful for spotting problems. However, as it is not possible to tell exactly what has changed by just glancing at the digest, it is best to delete the downloaded file and begin the download process again listing hashes.

A list of checksums looks like this:

#### SHA-1 Digest

bc9e476c8d17d609eb97aee7d38e0bdf49af78f2	npp.6.8.5.bin.7z
14ecc1afa8f2737a49f19dda8671c6efe5d08f68	npp.6.8.5.bin.minimalist.7z
6a9bb771cacblc6ed64a263c706477d755f71cc9	npp.6.8.5.bin.zip
88024d292c6b15df8be0341cf338d6f2059d8771	npp.6.8.5.Installer.exe

**One such illustration is downloading an operating system.**

- ❖ It's possible that you've concluded that checking every download you make is excessive. If so, no problem; the decision is yours. But what if the file is extremely large or crucial?
- ❖ This image shows the download page for Ubuntu Vivid Vervet. There is always a list of the file digests available for download along with the various file format options

**GUNAL R**

**II-IT**

## LAZY LOADING

Regarding the concept of "lazy loading," this postpones loading an object until we really need it.



### Introduction:

The idea behind lazy loading is to wait to load an item until we actually need it. Simply said, on demand object loading as opposed to needless object loading.

Consider the example below, where we have a straightforward Customer class with numerous Order objects included within it. Take a hard look at the Customer class' function Object() { [native code] }. The Order object is loaded at the same time the Customer object is generated. Therefore, the Order object is loaded regardless of whether we need it or not. But what if you initially only load the Customer object and then load the Order object as needed?

```
public class Customer
{
    private List<Order> _Orders= null;
    ...
    ...
    public Customer()
    {
        _CustomerName = "Shiv";
        _Orders = LoadOrders(); // Loads the order object even though //not
needed
    }

    private List<Order> LoadOrders()
    {
        List<Order> temp = new List<Order>();
    }
}
```

```

    Order o = new Order();
    o.OrderNumber = "ord1001";
    temp.Add(o);
    o = new Order();
    o.OrderNumber = "ord1002";
    temp.Add(o);
    return temp;
}

```



### So how can we put lazy loading into practise?

If we wish to use lazy loading in the aforementioned example, we must make the following adjustments: Eliminate the loading of the Order object from the function `Object () { [native code] }`. Only load the Order object into the Order get property if it isn't already loaded.

```

public class Customer
{
    private List<Order> _Orders= null;
    ...
    ...
    public Customer()
    {
        _CustomerName = "Shiv";
    }
    public List<Order> Orders
    {
        get
        {
            if (_Orders == null)
            {
                _Orders = LoadOrders();
            }
        }
    }
}

```

```
    }  
    return _Orders;  
  }  
}
```

Now, you can see the Orders object is null if you execute the client code and halt your debugger immediately before the foreach loop goes through it (i.e., not loaded). However, the Order object collection is created as soon as the foreach loop passes over the Order object.

### What are the Advantages and Disadvantages of Lazy Loading?



#### The advantages of lazy loading are as follows:

- Minimises the application's startup time.
- On-demand loading reduces memory use for the application.
- SQL execution on databases is avoided if possible.

The coding becoming difficult is the only drawback. There is a minor loss in performance because we must determine whether the loading is necessary or not. But the benefits outweigh the drawbacks by a wide margin. As a reminder, eager loading is the reverse of lazy loading. As a result, during eager loading, we load every object into memory as soon as it is created.

**PRITHIKA M**

**II-IT**

## Just Try...,,

### Problem 1:

A Dictionary is containing of different words. The problem is you need to find the longest word from the dictionary words.

Sample input:

```
{  
"cat",  
"dog",  
"red",  
"is",  
"am"  
}
```

Sample Output:

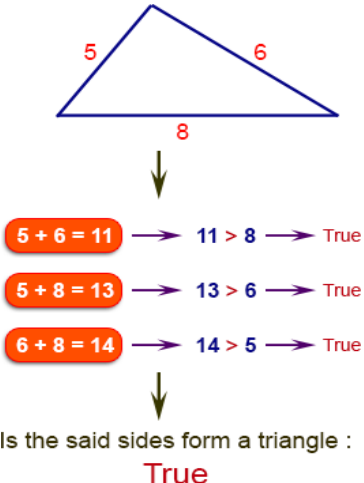
Original Dictionary:[cat, dog, red, is, am]

Longest word(s) of the above dictionary: [cat, dog, red]

For Solutions Refer Pg No:27

### Problem 2:

User needs to enter the 3sides to form triangle and display True if it is a triangle or False if it's not. Pictorial representation is given below,

 <p>5 + 6 = 11 → 11 &gt; 8 → True</p> <p>5 + 8 = 13 → 13 &gt; 6 → True</p> <p>6 + 8 = 14 → 14 &gt; 5 → True</p> <p>Is the said sides form a triangle : <b>True</b></p>	<p><b>Sample Input:</b></p> <p>Input side1: 5</p> <p>Input side2: 6</p> <p>Input side3: 8</p>	<p><b>Sample Output:</b></p> <p>Is the said side forms a triangle: True</p>
---	---	---

For Solutions Refer Pg No:28

**Problem 3:**

Let us use the letter H to mean "hundred", the letter T to mean "ten" and "1, 2, . . . n" to represent the ones digit n (<10). Write a Java program to convert 3 digits positive number in above format. For example, 234 should be output as BBSSS1234 because it has 2 "hundreds", 3 "ten", and 4 of the ones

**Sample input:**

Input a positive number(max three digits):

235

**Sample Output:**

Result:

HHTTT12345

For Solutions Refer Pg No:28

**Problem 4:**

Your task is to develop a small part of spreadsheet software. Write a Java program which adds up columns and rows of given table as shown in the specified figure:

**Sample Input:**

Input number of rows/columns (0 to exit)

4

25 69 51 26

68 35 29 54

54 57 45 63

61 68 47 59

**Sample Output:**

25 69 51 26 171

68 35 29 54 186

54 57 45 63 219

61 68 47 59 235

208 229 172 202 811

For Solutions Refer Pg No:29

This article will show how a desktop application produces numerous objects and looks for text within them. I recently searched for a software storage system. This desktop application makes numerous objects and looks for text within them. I so reasoned, "Why don't I try something new?" I could use some sort of document database instead of a SQL database. But I didn't want a separate server; I just wanted a straightforward file to be able to access this database. I found LiteDB really quickly when searching the Internet for these types of databases for .NET apps. And now I'd like to talk about my experience using this database.

### Inheritance

Storing object this way:

```
internal class Item
{
    public string Title { get; set; }

    public string Description { get; set; }

    public List<Field> Fields { get; set; } = new List<Field>();
}
```

The Field class, however, is abstract. It also has numerous descendants:

```
internal abstract class Field
{
}
internal sealed class TextField : Field
{
    public string Text { get; set; }
}
internal sealed class PasswordField : Field
{
    public string Password { get; set; }
}
internal sealed class DescriptionField : Field
{
    public string Description { get; set; }
}
```



I had to set up the storing of several Field class descendants when working with SQL databases. I mistakenly believed that using LiteDB would require me to create my own BSON serialisation technique; however LiteDB actually gives me the option to do so. But I received a welcome surprise. There is nothing I have to do. Different methods of serialisation and deserialization have previously been put into place. Simply make the required objects.

```
var items = new Item[]
{
    new Item
    {
        Title = "item1",
        Description = "description1",
        Fields =
        {
            new TextField
            {
                Text = "text1"
            },
            new PasswordField
            {
                Password = "123"
            }
        }
    },
    new Item
    {
        Title = "item2",
        Description = "description2",
        Fields =
        {
            new TextField
            {
                Text = "text2"
            },
            new DescriptionField
            {
                Description = "description2"
            }
        }
    }
};
```

**and insert them into the database:**

```
using (var db = new
LiteDatabase(connectionString))
{
    var collection = db.GetCollection<Item>();
    collection.InsertBulk(items);
}
```

**Just that. You can view the contents of your database using the LiteDB.Studio tool.**



**ABITHA  
III-IT**

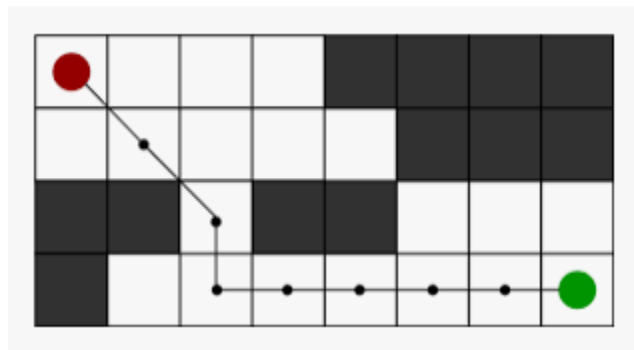
# PATHFINDING ALGORITHMS IN C#

## Introduction

Have you ever wondered how GPS programmes determine the quickest route to a specific location? You'll notice that it's actually fairly easy. This article discusses it and offers some sample code that you may use whenever you see fit. Dijkstra and A\*, two popular fundamental algorithms, are also contrasted in the paper.

### The Issue:

- Assume you own a map. You are aware of your location and your goals. The margins of the map serve as the roads that link the nodes (places with coordinates).
- You can travel to one or more edges from every node. The cost of an advantage (e.g. length or time it takes to travel it).
- One might possibly analyse all potential paths to the destination on small maps and choose the shortest one. However, given the exponential growth of the combinations, that is not particularly practical for maps with numerous nodes.



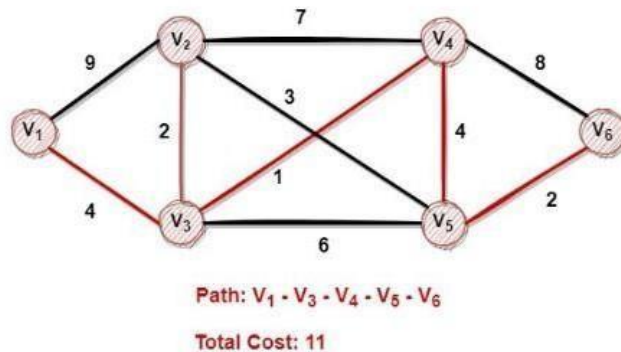
### Dijkstra:

Edsger Dijkstra made the discovery of the Dijkstra algorithm in 1959. The process is as follows:

- Add all linked nodes to the priority queue starting at the start node.
- Make the first node the current node and order the priority queue by lowest cost.
- Choose the optimal child node for each one that starts you out on the shortest path possible.
- A node is considered "Visited" when all of its edges have been examined from it, at which point you are no longer required to visit it.
- Each linked child node to the current node should be added to the priority queue. until the backlog is empty, proceed to step 2.

- Make a list of all the nodes in recursive fashion that are on the shortest route from start to finish.
- The quickest route can be discovered by reversing the list.

## Dijkstra's Algorithm



	Distance	Previous Node	Visited
V <sub>1</sub>	0	None	True
V <sub>2</sub>	6	V <sub>3</sub>	True
V <sub>3</sub>	4	V <sub>1</sub>	True
V <sub>4</sub>	5	V <sub>3</sub>	True
V <sub>5</sub>	9	V <sub>4</sub>	True
V <sub>6</sub>	11	V <sub>5</sub>	True

In other words, calculate the distance to the start for each child of a node iteratively. Save the distance and the node that was the starting point of the shortest path. Reverse that list after you get to the final node and travel back to the beginning the shortest route to find the shortest path.

Here is the C# code I used to implement the Dijkstra algorithm. It might be simpler to comprehend than the aforementioned.

```
public List<Node> GetShortestPathDijkstra()
{
    DijkstraSearch();
    var shortestPath = new List<Node>();
    shortestPath.Add(End);
    BuildShortestPath(shortestPath, End);
    shortestPath.Reverse();
    return shortestPath;
}

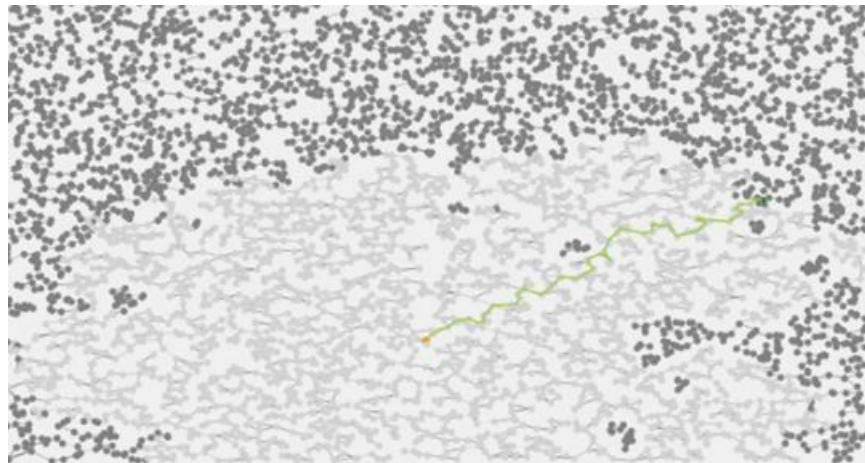
private void BuildShortestPath(List<Node> list, Node node)
{
    if (node.NearestToStart == null)
        return;
    list.Add(node.NearestToStart);
    BuildShortestPath(list, node.NearestToStart);
}

private void DijkstraSearch()
{
    Start.MinCostToStart = 0;
```

```

var prioQueue = new List<Node>();
prioQueue.Add(Start);
do {
    prioQueue = prioQueue.OrderBy(x => x.MinCostToStart).ToList();
    var node = prioQueue.First();
    prioQueue.Remove(node);
    foreach (var cnn in node.Connections.OrderBy(x => x.Cost))
    {
        var childNode = cnn.ConnectedNode;
        if (childNode.Visited)
            continue;
        if (childNode.MinCostToStart == null ||
            node.MinCostToStart + cnn.Cost < childNode.MinCostToStart)
        {
            childNode.MinCostToStart = node.MinCostToStart + cnn.Cost;
            childNode.NearestToStart = node;
            if (!prioQueue.Contains(childNode))
                prioQueue.Add(childNode);
        }
    }
    node.Visited = true;
    if (node == End)
        return;
} while (prioQueue.Any());
}

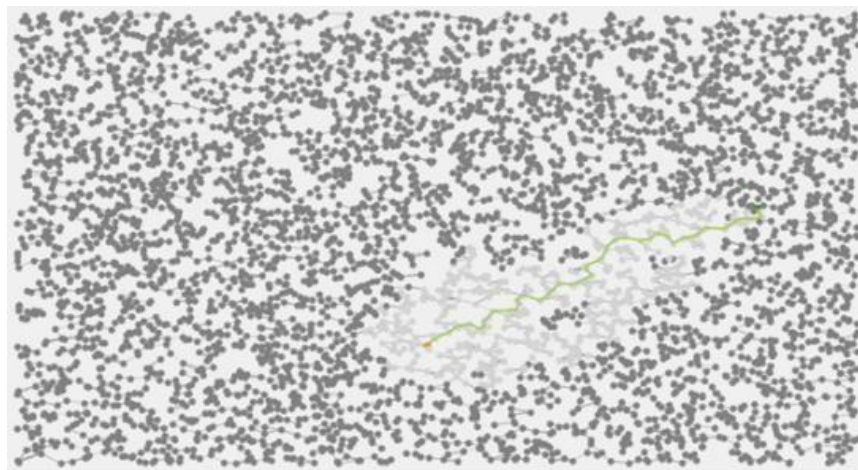
```



In my test software, I generated this map at random. The lines between the dots, which symbolise edges, are nodes. There are 5000 nodes and 15000 edges on this map. The search algorithm visits lighter coloured dots, and the optimum path is shown in green.

## Algorithm A\*

- There are numerous ways to improve Dijkstra's algorithm. A\* is one of the most typical. With one small change, it is essentially identical to Dijkstra.
- Additionally, edges are given priority based on how much closer they bring the target in a straight line. The straight-line distance to each node's final destination must therefore be calculated before beginning an A\* search, which is simple to do if you know each node's coordinate. This is the most basic version of A\*, and its definition permits modifications to the heuristics function. Straight Line Distance To End in this instance.
- Due to the fact that this algorithm visits fewer nodes when the direction of the path end is known, it performs much better.
- See the example I used below. What has been added to Dijkstra's algorithm is in bold.



## Conclusion:

### Which of Dijkstra and Apath \*'s finding algorithms is therefore the best?

It is A\* if you are simply concerned with finding the shortest route. It is far quicker and produces the same outcome as Dijkstra. However, Dijkstra is more effective at determining the optimum path than this iteration of A\* if the cost of an edge has characteristics other than its length. After all, it continues to move quickly. I consider 500,000 nodes to be a sizable data set. My implementation, in my opinion, might be greatly improved.

**FATHEEM MEERAN S**

**III -IT**

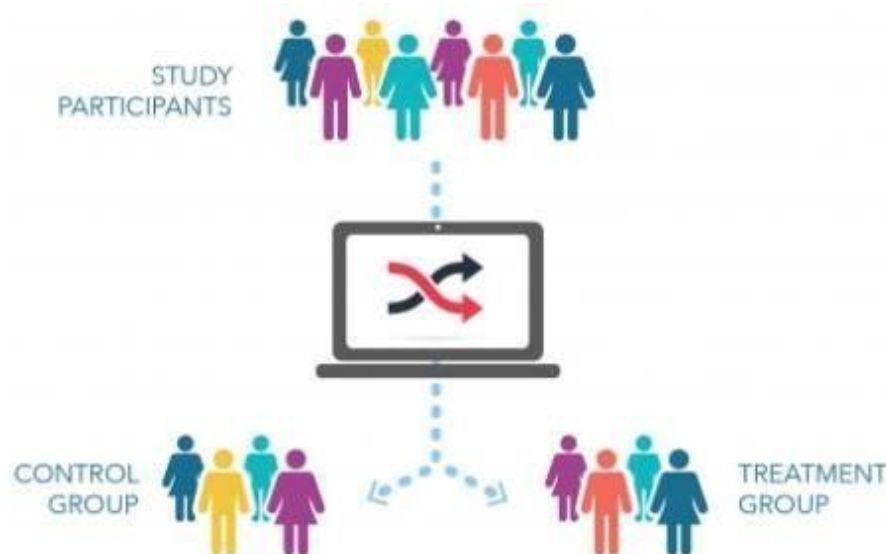
# RANDOMIZATION AND SAMPLING METHODS

Contains pseudocode for many of them and provides several different methods programmes can sample using an underlying (pseudo-)random number generator. This article explores many ways that applications can sample randomised information by modifying the results of an underlying source of random numbers, such as the results of a pseudorandom number generator, and provides pseudocode and sample Python code for many of these techniques.

## Introduction:

Methods for sampling and randomization are listed on this page. A randomization or sampling method generates numbers or other values known as random variates when it is driven by a "source of random numbers." The randomization led to the creation of these variations. In reality, so-called pseudorandom number generators, or PRNGs, frequently replicate the "source of random numbers." This document discusses a variety of techniques, like as,

- Ways to generate randomised content and conditions, such as true/false conditions, shuffling, and sampling unique items from a list
- Ways to sample integers or real numbers from a uniform distribution (such as the core method, `RNDINT(N)`);
- And non-uniform distributions, such as weighted choice, the Poisson distribution, and other probability distributions.





This page focuses on sampling and randomization techniques that precisely sample from the stated distribution without adding any further errors on top of the ones already present in the inputs (and assuming that an ideal "source of random numbers" is available). If there are only a finite amount of available values, then this will be the case. However, there will always be some degree of approximation when dealing with distributions like the normal distribution and others that can take on an unlimited number of values; in this case, the emphasis of this page is on techniques that reduce the mistake they impose.

The documentation for the code is given together with sample Python code that implements many of the methods in this document. This document also includes pseudocode for many of the methods.

The methods for randomization described on this page presuppose that we have an infinite supply of numbers that have been independently picked at random and have a uniform distribution. See "Sources of Random Numbers" in the appendix for more details.



**The following generally fall outside the purview of this document:**

- The selection of an underlying PRNG (or machine or software that simulates a "source of random numbers") for a specific application, including considerations of security, performance, and quality, is not covered in this paper. In another document, I go into greater detail on recommendations.

- Algorithms for particular PRNGs, such as Mersenne Twister, PCG, xorshift, linear congruential generators, or generators based on hash functions, are not included in this document.
- The same is true for other hardware and software that act as a "source of random numbers," as this paper does not describe how to verify PRNGs for accuracy or sufficiency. For instance, "Testing PRNGs for High-Quality Randomness" discusses testing.
- The creation or specification of "seeds" for usage in PRNGs is not covered in this document. Detail about this is provided elsewhere.
- The creation of random security parameters, such as encryption keys, is not demonstrated in this document.
- Randomness extraction is not covered in this paper (also known as unbiassing, deskewing, or whitening). See my note on the extraction of randomness.
- Techniques for reducing variation, such as common random numbers or importance sampling, are outside the scope of this work.

Additionally, sampling techniques used for computer graphics rendering (such as Poisson disc sampling, multiple importance sampling, blue noise, and gradient noise) are not the focus of this page because they tend to prioritise performance and aesthetic appeal over accuracy and precise sampling in this application. In rendering graphics, rejection sampling is virtually ever employed.

### **Notation:**

- The rules for pseudocode apply to this document.
- For intervals, the notation is as follows: "A or greater, but less than b" is denoted by  $[a, b]$ . "Greater than a, but less than b" is what  $(a, b)$  means. "Greater than a and less than or equal to b" is what  $(a, b]$  means. "A or greater and b or less" is what  $[a, b]$  denotes.
- Where  $x$  is a floating-point number,  $\log_{1p}(x)$  is a "robust" substitute for  $\ln(1 + x)$  that is equivalent to  $\ln(1 + x)$  (Pedersen 2018) <sup>[1]</sup>.
- With the supplied numerator  $n$  and denominator  $d$ ,  $\text{MakeRatio}(n, d)$  generates a rational number.



- Sum(list) computes the total of the integers or rational numbers in the supplied list. (Naively rounding off floating-point figures can result in mistakes.)

### Uniform Random Integers:

The rules for pseudocode apply to this document. This section demonstrates how to use a "source of random numbers" to generate independent, uniform random integers (or a device or programme that simulates that source).

The four approaches RNDINT, RNDINTEXC, RNDINTRANGE, and RNDINTEXCRANGE are discussed in this section. RNDINT, which is discussed next, can be used as the foundation for the other techniques.



### RNDINT: Random Integers in [0, N]:

The main method in this document, RNDINT(maxInclusive), obtains independent uniform integers in the range [0, maxInclusive] from a "source of random numbers" [2]. The following pseudocode, which uses RNDINT, says:

1. According to the appendix, the function NEXTRAND() reads the subsequent number produced by a "source of (uniform) random numbers" (an endless source of numbers, each of which is chosen independently of any other choice and with a uniform distribution). A pseudorandom number generator can actually replicate this source, as mentioned in the appendix. Instead of having a uniform distribution, the source in this method could have a non-uniform one.
2. The amount of conceivable outcomes using that source is known as MODULUS.

```
3. METHOD RndIntHelperNonPowerOfTwo(maxInclusive)
4.   if maxInclusive <= MODULUS - 1:
5.     // NOTE: If the programming language implements
6.     // division with two integers by discarding the result's
7.     // fractional part, the division can be used as is without
8.     // using a "floor" function.
```

```

9.      nPlusOne = maxInclusive + 1
10.     maxexc = floor((MODULUS - 1) / nPlusOne) * nPlusOne
11.     while true // until a value is returned
12.         ret = NEXTRAND()
13.         if ret < nPlusOne: return ret
14.         if ret < maxexc: return rem(ret, nPlusOne)
15.     end
16. else
17.     cx = floor(maxInclusive / MODULUS) + 1
18.     while true // until a value is returned
19.         ret = cx * NEXTRAND()
20.         // NOTE: The addition operation below should
21.         // check for integer overflow and should reject the
22.         // number if overflow would result.
23.         ret = ret + RNDINT(cx - 1)
24.         if ret <= maxInclusive: return ret
25.     end
26. end
27. END METHOD
28.
29. METHOD RndIntHelperPowerOfTwo(maxInclusive)
30.     // NOTE: Finds the number of bits minus 1 needed
31.     // to represent MODULUS (in other words, the number
32.     // of random bits returned by NEXTRAND() ). This will
33.     // be a constant here, though.
34.     modBits = ln(MODULUS)/ln(2)
35.     // Lumbroso's Fast Dice Roller.
36.     x = 1
37.     y = 0
38.     nextBit = modBits
39.     rngv = 0
40.     while true // until a value is returned
41.         if nextBit >= modBits
42.             nextBit = 0
43.             rngv = NEXTRAND()
44.         end
45.         x = x * 2
46.         y = y * 2 + rem(rngv, 2)
47.         rngv = floor(rngv / 2)
48.         nextBit = nextBit + 1
49.         if x > maxInclusive
50.             if y <= maxInclusive: return y
51.             x = x - maxInclusive - 1
52.             y = y - maxInclusive - 1
53.         end
54.     end
55. END METHOD
56.
57. METHOD RNDINT(maxInclusive)
58.     // maxInclusive must be 0 or greater
59.     if maxInclusive < 0: return error
60.     if maxInclusive == 0: return 0

```

```
61. if maxInclusive == MODULUS - 1: return NEXTRAND()
62. // NOTE: Finds the number of bits minus 1 needed
63. // to represent MODULUS (if it's a power of 2).
64. // This will be a constant here, though.
65. modBits=ln(MODULUS)/ln(2)
66. if floor(modBits) == modBits // Is an integer
67.     return RndIntHelperPowerOfTwo(maxInclusive)
68. else
69.     return RndIntHelperNonPowerOfTwo(maxInclusive)
70. end
71. END METHOD
```

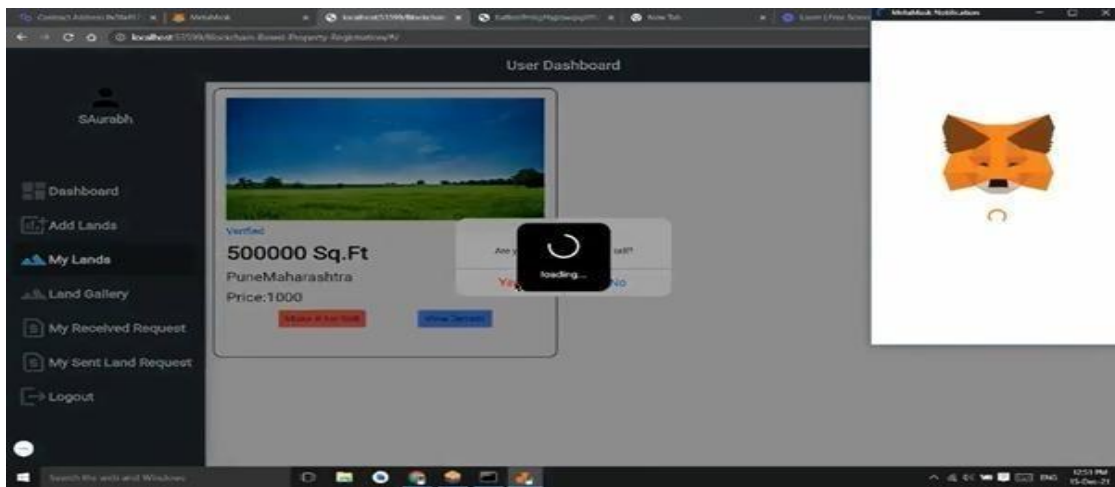
**SRIJENA N**

**IV-IT**

# PROJECT GLIMPSES

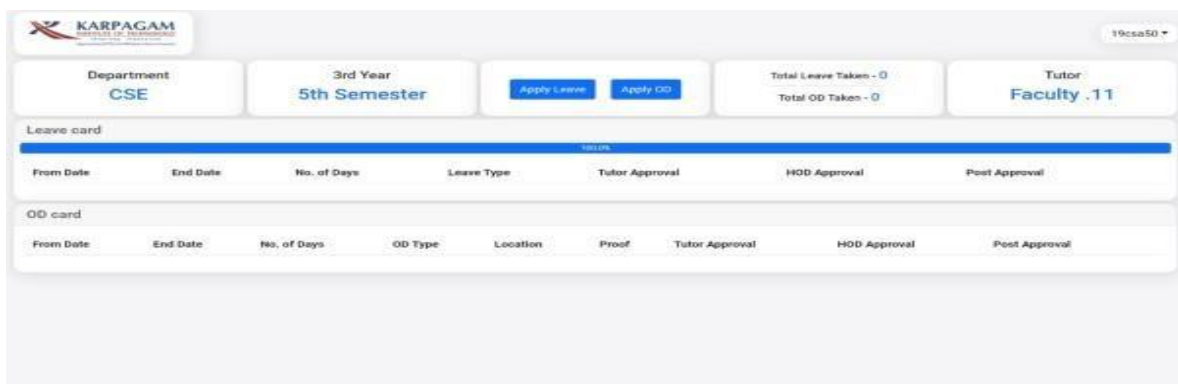
## Land record management system

Land Record Management System is used for recording, indexing, accessing and maintaining the land records. It is maintaining the record by using the Blockchain Technology and it is working based on the SHA 256 algorithm. The platform used in this system is SHA256 algorithm.



## ERP: Leave and OD management module

Leave and OD management system is a web application created using Django. It is an interface between the students and their tutors in which a student can apply leave and OD through this application, then it is get validated and approved by corresponding tutor. With the help of this app college can maintain the students leave and OD records and they can access at any time.



### **ERP: Event Management System**

Event management system is used to maintain the details of events conducted in college. This application is used to manage the event process, event feedback and event report. If any event like workshop or seminar is conducted in college it is easy to collect feedback from students, also it is easy to get the event report by using this application.

## Just Try it.., Solutions

### Problem 1:

```
import java.util.*;

public class Solution {

    static ArrayList<String> longestWords(String[] dictionary) {

        ArrayList<String> list = new ArrayList<String>();

        int longest_length = 0;

        for (String str : dictionary) {

            int length = str.length();

            if (length > longest_length) {

                longest_length = length;

                list.clear();

            }

            if (length == longest_length) {

                list.add(str);

            }

        }

        return list;

    }

    public static void main(String[] args) {

        //String [] dict = {"cat", "flag", "green", "country", "w3resource"};

        String [] dict = {"cat", "dog", "red", "is", "am"};

        System.out.println("Original dictionary : "+Arrays.toString(dict));

        System.out.println("Longest word(s) of the above dictionary:

"+longestWords(dict)); }}
```

**Problem 2:**

```
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Input side1: ");
        int s1 = in.nextInt();
        System.out.print("Input side2: ");
        int s2 = in.nextInt();
        System.out.print("Input side3: ");
        int s3 = in.nextInt();
        System.out.print("Is the said sides form a triangle: " + isValidTriangle(s1, s2, s3));
    }

    public static boolean isValidTriangle(int a, int b, int c) {
        return (a + b > c && b + c > a && c + a > b);
    }
}
```

**Problem 3:**

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("Input a positive number(max three digits):");
        char[] num = String.format("%03d", in.nextInt()).toCharArray();
        StringBuilder tm = new StringBuilder();
        for (int i = 0; i < num[0] - '0'; i++) {
            tm.append("H");
        }
        for (int i = 0; i < num[1] - '0'; i++) {
            tm.append("T");
        }
    }
}
```

```

    }
    for (int i = 0; i < num[2] - '0'; i++) {
        tm.append(i + 1);
    }
    System.out.println("Result:");
    System.out.println(tm.toString());
}
}

```

#### Problem 4:

```

import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Input number of rows/columns (0 to exit)");
        while(true){
            int n = sc.nextInt();
            if(n==0)break;
            int[][] map = new int[n+1][n+1];
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    map[i][j] = sc.nextInt();
                    map[i][n] += map[i][j];
                }
                map[n][n] += map[i][n];
            }
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    map[n][i] += map[j][i];
                }
            }
        }
    }
}

```



```
System.out.println("Result:");
for(int i=0;i<n+1;i++){
    for(int j=0;j<n+1;j++){
        System.out.printf("%5d", map[i][j]);
    }
    System.out.println();
}
}
```

# **EDITORIAL MEMBERS**

## **EDITORIAL FACULTY MEMBER**

**Dr.B.CHELLAPRABHA Prof./IT**

## **EDITORIAL STUDENT MEMBERS**

M. PRITHIKA	(II - IT)
D. VISHAL	(III - IT)
N. SRIJENA	(IV - IT)
E.HARIPRASATH	(IV - IT)



**Web:** [www.karpagamtech.ac.in](http://www.karpagamtech.ac.in)

**Address:** L&T Bypass Road,  
Bodipalayam post,  
Coimbatore 641 105.