LAB MANUAL

For

COMPUTER LABORATORY-III

Subject Code: 410453

ASSIGNMENT NO: A1

1. TITLE

Using Divide and Conquer Strategies and object-oriented software design technique using Modelio to design a software function for Binary Search for an un-ordered data stored in memory. Use necessary USE-CASE diagrams and justify its use with the help of mathematical modelling and related efficiency. Implement the design using Python.

2. PREREQUISITES

- 64-bit Fedora or equivalent OS with 64-bit Intel-i5/i7
- Python 2.7

3. OBJECTIVE

- To Implements the Ordered search approach for given number..
- Implementation search method.

4. MATHEMATICAL MODELS

Let, S be the System Such that,
A={ S, E, I,O, F, DD, NDD, F_min ,F_fri, CPU_Core, Mem_Shared, success, failure }
Where,
S= Start state, E=
End State, I= Set
of Input O= Set of
Out put F =Set of
Function
DD=Deterministic Data
NDD=Non Deterministic Data
F_Min=Main Function
F_Fri= Friend Function CPU_Core=
No of CPU Core.
Mem_ Shared=Shared
Memory. Function:

1) Splitting Function = This function is used for splitting unsorted list.
2) Sorting Function = This function is used for sorting list.
3) Binary Search = This function apply binary search on sorted list.

Success Case: It is the case when all the inputs are given by system are entered correctly. Failure Case: It is the case when the input does not match the validation Criteria.

5. THEORY

Divide and Conquer

The most well-known algorithm design strategy, Given a function to compute on n inputs, the divide-and-conquer strategy consists of:

1. **Divide** the problem into two or more smaller sub-problems. That is splitting the inputs into k distinct subsets, $1 \leq k \leq n$, yielding k sub-problems.
2. **Conquer** the sub problems by solving them recursively.
3. **Combine** the solutions to the sub problems into the solutions for the original problem.
4. if the sub-problems are relatively large, then divide_Conquer is applied again.
5. if the sub-problems are small, then sub-problems are solved without splitting.

A typical Divide and Conquer case:

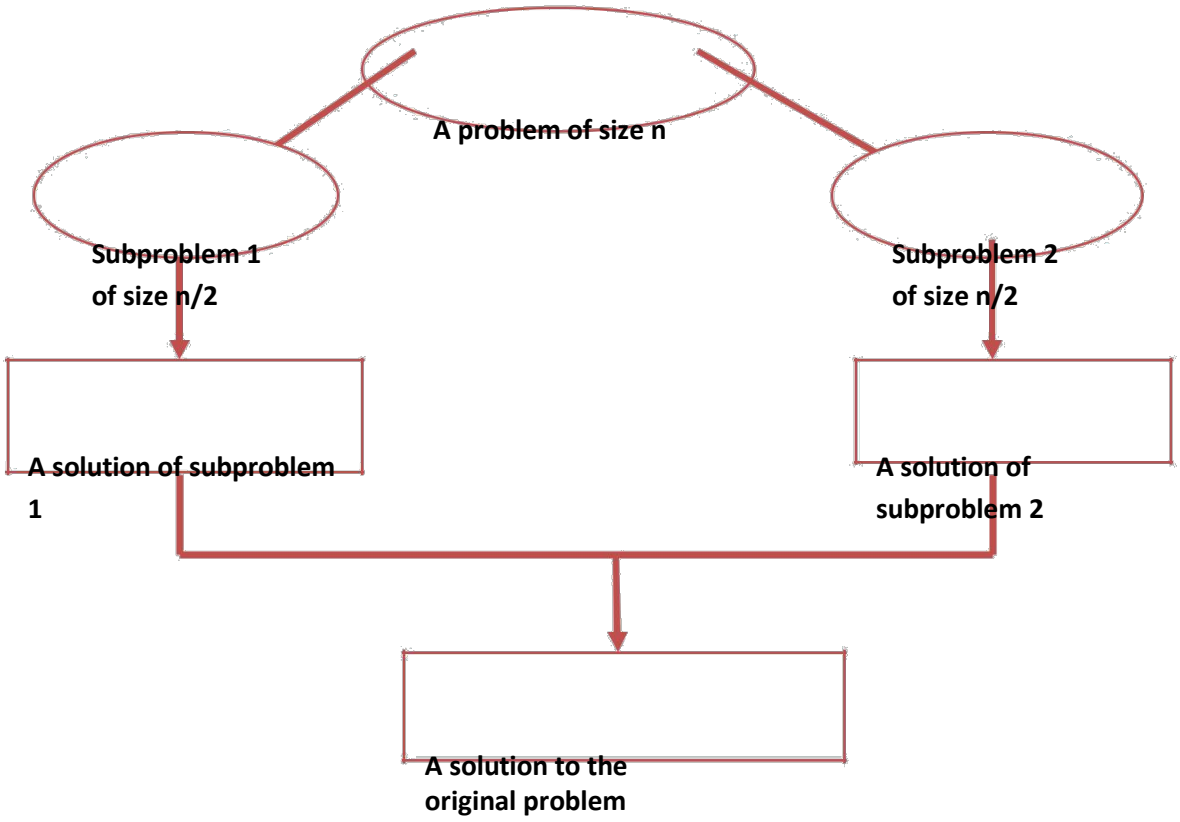


Fig. Divide and Conquer Strategy

General method of Divide and Conquer algorithm

```
Divide_Conquer(problem P)
{
if   Small(P)
return  S(P);
else {
divide P into smaller instances P1, P2, …, Pk, k
Apply Divide Conquer to each of these subproblems ;
return
Combine (Divide_Conque(P1), Divide_Conque (P2),…,……Divide_Conque (Pk));
}
}
```

BINARY SEARCH

```
1.      Algorithm Bin search(a,n,x)
2.      // Given an array a[1:n] of elements in non-decreasing
3.      //order, n>=0,determine whether 'x' is present and
4.      // if so, return 'j' such that x=a[j]; else return 0.
5.      {
6.      low:=1; high:=n;
7.      while (low<=high) do
8.      {
9.      mid:=[(low+high)/2];
10.     if (x<a[mid]) then high; 11.      else if(x>a[mid]) then      low=mid+1;
12.   else return mid;
13.   }
14.   return 0;
15.   }
```

- Algorithm, describes this binary search method, where Binsrch has 4I/ps a[], I , l & x.
- It is initially invoked as Binsrch (a,1,n,x)
- A non-recursive version of Binsrch is given below.
- This Binsearch has 3 i/ps a,n, & x.
- The while loop continues processing as long as there are more elements left to check.
- At the conclusion of the procedure 0 is returned if x is not present, or 'j' is returned, such that a[j]=x.

- We observe that low & high are integer Variables such that each time through the loop either x is found or low is increased by at least one or high is decreased at least one.
- Thus we have 2 sequences of integers approaching each other and eventually low becomes > than high & causes termination in a finite no. of steps if 'x' is not present.

6. APPLICATION FLOW

- start with our root/goal node and check current vertex is the goal state
- treat List as stack
- new search states to explore at front of list
- put new states=use heuristics
- leaf node in search List
- Use Backtrack for higher node.

7. UML Diagrams

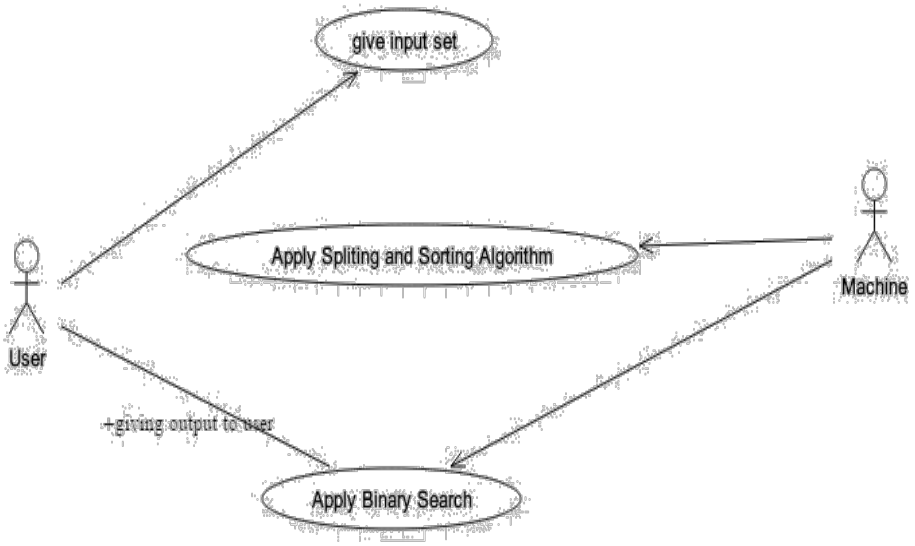


Fig: Use case Diagram

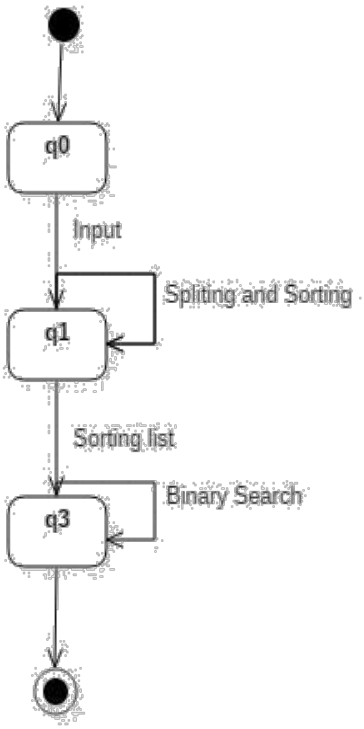


Fig: State Diagram

8. CONCLUSION

Binary search method using divide and conquer strategy is implemented.

ASSIGNMENT NO: A2

1. TITLE:

Using Divide and Conquer Strategies to design an efficient class for Concurrent Quick sort and the input data is stored using XML. Use object oriented software design method and Modelio / StarUML2.x Tool. Perform the efficiency comparison with any two software design methods. Use necessary USE-CASE diagrams and justify its use with the help of mathematical modeling. Implement the design using Scala/ Python/Java/C++.

2. PREREQUISITES:

- 64-bit Fedora or equivalent OS with 64-bit Intel-i5/i7
- Java 1.7.0

3. OBJECTIVE:

- To learn the concept of Divide and Conquer Strategy.
- To study the design and implementation of Quick Sort algorithm

4. INPUT: I=Set of 8 quuens in 8*8 matrix

5. OUTPUT: O1=Success Case: It is the case when all the inputs are given by system are correctly and 8-Queen problem is solved.

O2=Failure Case: It is the case when the input does not match the validation Criteria.

6. MATHEMATICAL MODEL:

Let Z be a system={I,O,T,Sc,Fc}

I=Set of inputs.

O=Set of outputs.

T=Set of transitions function.

Sc=Success Case.

Fc=Failure Case.

I={matrix,8q.json}

O={matrix}

T={issafe,place}

Sc=solution found

Fc=solution not found.

7. THEORY:

Divide and Conquer strategy:

A divide and conquer algorithm works by recursively breaking down a problem into two or more sub- problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

Quick sort:

The sorting algorithm invented by Hoare, usually known as "quicksort" is also based on the idea of divide-and-conquer. In Quick sort, input is any sequence of numbers and output is sorted array(here we will consider ascending order). We start with one number, mostly the first number, and finds its position in sorted array. This number is called pivot element. Then we divide the array into two parts considering this pivot elements position in sorted array. In each part, separately, we find the pivot elements. This process continues until all numbers are processed and we get the sorted array. In concurrent Quick sort, each part is processed by independent thread i.e. different threads will find out pivot element in each part recursively. Check in following diagram. Here PE stands for Pivot Element.

```
private static void quicksort(int[] arr, int low, int high)
{
    if (arr == null || arr.length == 0)
        return;
    if (low >= high)
        return;
    int middle = low + (high - low) / 2;
    int pivot = arr[middle];
    int i = low, j = high;
```

```
while (i <= j)
{
        while (arr[i] < pivot)
        {
                i++;
        }
while (arr[j] > pivot)
{
        j--;
}
if (i <= j) {
int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
i++;
j--;
}
}
// recursively sort two sub parts
if (low < j)
quicksort(arr, low, j);
if (high > i)
quicksort(arr, i, high);
}
```

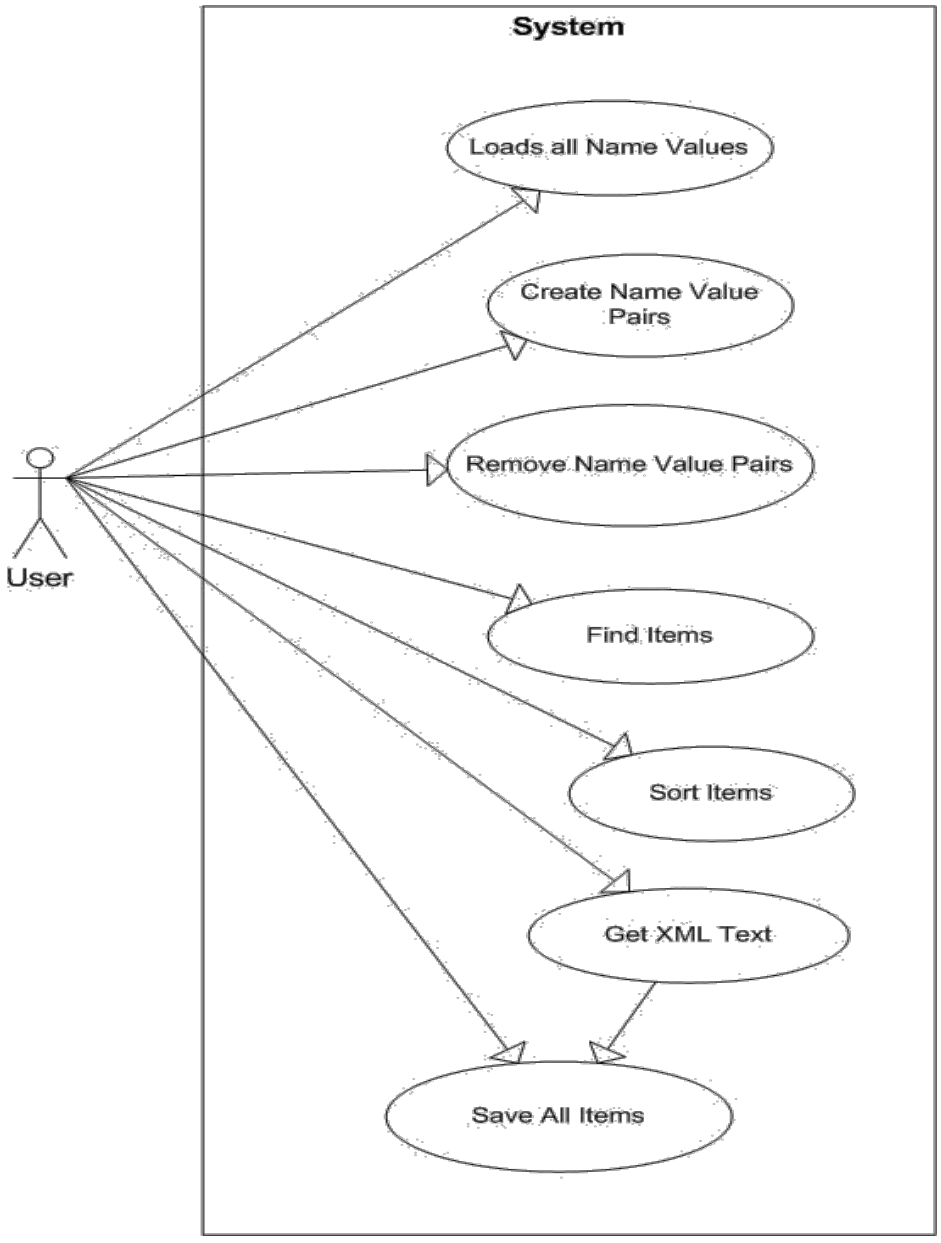
Document Object Model:

The **Document Object Model** (**DOM**) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML, and XML documents. The nodes of every document are organized in a tree structure, called the DOM

tree. Objects in the DOM tree may be addressed and manipulated by using methods on the objects. The public interface of a DOM is specified in its application programming interface (API).

Relative efficiency:

Class	Search algorithm
Worst case performance	**$O(N^2)$**
Best case performance	**O(NlogN)**
Average case performance	**O(NlogN)**



Patterns to be followed in this assignment are FAÇADE pattern and command pattern which includes 2 tier architecture and client server architecture.

CONCLUSION:

Thus we have studied Concurrent Quick Sort using divide and conquer strategy.

ASSIGNMENT NO: A3

1. TITLE

A Web Tool for Booth's multiplication algorithm is used to multiply two numbers located in distributed environment. Use software design client-server architecture and principles for dynamic programming. Perform Risk Analysis. Implement the design using HTML-5/Scala/ Python/Java/C++/ Rubi on Rails. Perform Positive and Negative testing. Use latest open source software modelling, Designing and testing tool/Scrum-it/KADOS and Camel.

2. PREREQUISITES

- 64-bit Fedora or equivalent OS with 64-bit Intel-i5/i7
- Java 1.7.0
- Testing Tool: Scrum-it/KADOS/Camel

3. OBJECTIVE

- To perform Risk Analysis.
- To learn about designing and testing tools.

4. MATHEMATICAL MODELS

Let, S be the System Such that,
A={ S, E, I,O, F, DD, NDD, success, failure }
Where,
S= Start state, E=
End State, I= Set
of Input O= Set of
Out put F =Set of
Function
DD=Deterministic Data
NDD=Non Deterministic Data
Success Case: It is the case when all the inputs are given by system are entered correctly.
Failure Case: It is the case when the input does not match the validation Criteria.

5. THEORY

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. The algorithm was invented by Andrew Donald Booth in 1950.

Booth Multiplication algorithm:

Booth's algorithm examines adjacent pairs of bits of the N-bit multiplier Y in signed two's complement representation, including an implicit bit below the least significant bit, $y_{-1} = 0$. For each bit y_i, for i running from 0 to N-1, the bits y_i and y_{i-1} are considered. Where these two bits are equal, the product accumulator P is left unchanged. Where $y_i = 0$ and $y_{i-1} = 1$, the multiplicand times 2^i is added to P; and where $y_i = 1$ and $y_{i-1} = 0$, the multiplicand times 2^i is subtracted from P. The final value of P is the signed product.

The multiplicand and product are specified; typically, these are both also in two's complement representation, like the multiplier, but any number system that supports addition and subtraction will work as well. As stated here, the order of the steps is not determined. Typically, it proceeds from LSB to MSB, starting at $i = 0$; the multiplication by 2^i is then typically replaced by incremental shifting of the P accumulator to the right between steps; low bits can be shifted out, and subsequent additions and subtractions can then be done just on the highest N bits of P. There are many variations and optimizations on these details.

The algorithm is often described as converting strings of 1's in the multiplier to a high-order +1 and a low-order –1 at the ends of the string. When a string runs through the MSB, there is no high-order +1, and the net effect is interpretation as a negative of the appropriate value.

Implementation:

Booth's algorithm can be implemented by repeatedly adding (with ordinary unsigned binary addition) one of two predetermined values A and S to a product P, then performing a rightward arithmetic shift on P. Let **m** and **r** be the multiplicand and multiplier, respectively; and let x and y represent the number of bits in **m** and **r**.

1. Determine the values of A and S, and the initial value of P. All of these numbers should have a length equal to $(x + y + 1)$.
    1. A: Fill the most significant (leftmost) bits with the value of **m**. Fill the remaining $(y + 1)$ bits with zeros.
    2. S: Fill the most significant bits with the value of $(-\mathbf{m})$ in two's complement notation. Fill the remaining $(y + 1)$ bits with zeros.
    3. P: Fill the most significant x bits with zeros. To the right of this, append the value of **r**. Fill the least significant (rightmost) bit with a zero.
2. Determine the two least significant (rightmost) bits of P.
    1. If they are 01, find the value of $P + A$. Ignore any overflow.
    2. If they are 10, find the value of $P + S$. Ignore any overflow.
    3. If they are 00, do nothing. Use P directly in the next step.
    4. If they are 11, do nothing. Use P directly in the next step.
3. Arithmetically shift the value obtained in the 2nd step by a single place to the right. Let P now equal this new value.
4. Repeat steps 2 and 3 until they have been done y times.
5. Drop the least significant (rightmost) bit from P. This is the product of **m** and **r**.

We will explain Booth's Multiplication with the help of following **example**
Multiply 14 times -5 using 5-bit numbers (10-bit
result). 14 in binary: 01110
-14 in binary: 10010 (so we can add when we need to subtract the
multiplicand) -5 in binary: 11011
Expected result: -70 in binary: 11101 11010 The
Multiplication Process is explained below:

Step	Multiplicand	Action	Multiplier upper 5-bits 0, lower 5-bits multiplier, 1 "Booth bit" initially 0
0	01110	Initialization	00000 11011 0
1	01110	10: Subtract Multiplicand	00000+10010=10010 10010 11011 0
		Shift Right Arithmetic	11001 01101 1
2	01110	11: No-op	11001 01101 1
		Shift Right Arithmetic	11100 10110 1
3	01110	01: Add Multiplicand	11100+01110=01010 (Carry ignored because adding a positive and negative number cannot overflow.) 01010 10110 1
		Shift Right Arithmetic	00101 01011 0
4	01110	10: Subtract Multiplicand	00101+10010=10111 10111 01011 0
		Shift Right Arithmetic	11011 10101 1
5	01110	11: No-op	11011 10101 1
		Shift Right Arithmetic	11101 11010 1

Booth's Multiplication algorithm is an elegant approach to multiplying signed numbers. Using the standard multiplication algorithm, a run of 1s in the multiplier in means that we have to add as many successively shifted multiplicand values as the number of 1s in the run.

0010two
x 0111two

+ 0010 multiplicand shifted by 0 bits left
+ 0010 multiplicand shifted by 1 bit left
+ 0010 multiplicand shifted by 2 bits left
+ 0000

00001110two

We can rewrite 2i - 2i-j as:

2i - 2i-j = 2i-j x (2j - 1)

= 2i-j x (2j-1 + 2j-2 + ... + 20)

= 2i-1 + 2i-2 + ... + 2i-j

For example 0111two = 23ten - 20ten. So 0010two x 0111two can be written as: 0010two x (1000two - 0001two)

Or to make it look like the multiplication before:

0010two

x 0111two

\- 0010 = 0010two x - 0001two

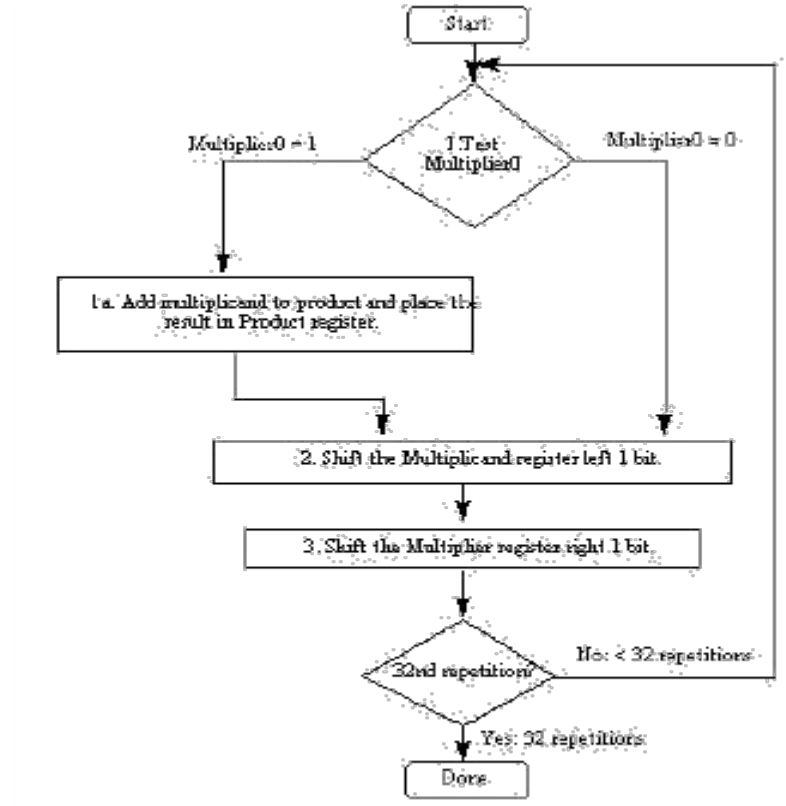
\+ 0000

\+ 0000

\+ 0010 = 0010two x + 1000two

00001110two

Flowchart of booth's multiplication Algorithm :



Multiply Algorithm

KADOS is a web tool for managing Agile projects (Scrum more specifically) through visual boards on which are displayed post-its representing User Stories, Tasks, Activities, Issues, Actions, Bugs and any objects you wanted your project to manage.

RISK ANALYSIS

Risk can be defined as the potential of losses and rewards resulting from an exposure to a hazard or as a result of a risk event. Risk can be viewed to be a multi-dimensional quantity that includes

- event occurrence probability,
- event occurrence consequences,
- consequence significance, and
- the population at risk.

Risk analysis is the process of defining and analyzing the dangers to individuals, businesses and government agencies posed by potential natural and human-caused adverse events. Risk analysis is the review of the risks associated with a particular event or action. In IT, a risk analysis report can be used to align technology-related objectives with a company's business objectives. A risk analysis report can be either quantitative or qualitative. Risk analysis can be defined in many different ways, and much of the definition depends on how risk analysis relates to other concepts. Risk analysis can be "broadly defined to include risk assessment, risk characterization, risk communication, risk management, and policy relating to risk, in the context of risks of concern to individuals, to public- and private-sector organizations, and to society at a local, regional, national, or global level." A useful construct is to divide risk analysis into two components: (1) risk assessment (identifying, evaluating, and measuring the probability and severity of risks) and (2) risk management (deciding what to do about risks).

6. CONCLUSION

Hence we have implemented Booth’s Multiplication algorithm and performed test cases for that.

ASSIGNMENT NO: A4

1. TITLE

In an embedded system application Dining Philosopher's problem algorithm is used to design a software that uses shared memory between neighbouring processes to consume the data. The Data is generated by different Sensors/WSN system Network and stored in Mongo DB (NoSQL). Implementation be done using Scala/ Python/ C++/ Java. Design using Client-Server architecture. Perform Reliability Testing. Use latest open source software modelling, Designing and testing tool/Scrum-it/KADOS, NoSQL Unit and Camel.

2. PREREQUISITES

- 64-bit Fedora or equivalent OS with 64-bit Intel-i5/i7
- Java 1.7.x And MongoDB

3. OBJECTIVE

- Solve the Dining philosopher's problem using Python.

4. MATHEMATICAL MODEL

Let, S be the System Such that,
S={ S, E, I,O, F, DD, NDD, success, failure }
Where,
S = Start state,
E = End state,
I = Input
O = Output
DD = Deterministic Data
NDD = Non-deterministic Data
s1 = Initialization of
database/server.
s2 = Initialization of connection with the server.
e1 = Successful completion of the program and output written in
database. F : Functions:

F1 = main() : Here we create the threads and assign them to each philosopher and start their execution.

F2 = run() : This is a function to be written when we use runnable interface for parallel processing of threads.

F3 = eat() : In this function, we lock the neighboring philosopher and the calling thread(philosopher) stars eating operation.

F4 = thing() : In this function, the calling thread(philosopher) starts thinking.

Success Case: It is the case the connection is Successful with the MongoDB server and threads start running parallely.
Failure Case: It is the case when the connection is failed or there is an exception in threads.

5. THEORY

Dining philosopher's problem

The problem was designed to illustrate the challenges of avoiding deadlock, a system state in which no progress is possible. To see that a proper solution to this problem is not obvious, consider a proposal in which each philosopher is instructed to behave as follows:

- think until the left fork is available; when it is, pick it up;
- think until the right fork is available; when it is, pick it up;
- when both forks are held, eat for a fixed amount of time;
- then, put the right fork down;
- then, put the left fork down; and repeat from the beginning.

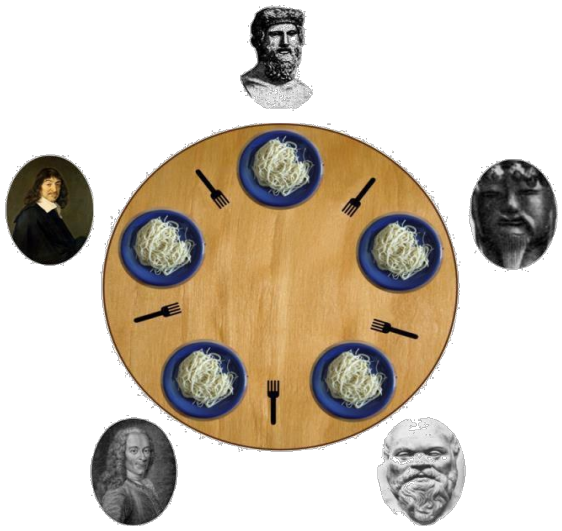


Fig. Dining philosopher's problem

This attempted solution fails because it allows the system to reach a deadlock state, in which no progress is possible. This is a state in which each philosopher has picked up the fork to the left, and is waiting for the fork to the right to become available. With the given instructions, this state can be reached, and when it is reached, the philosophers will eternally wait for each other to release a fork.

Resource starvation might also occur independently of deadlock if a particular philosopher is unable to acquire both forks because of a timing problem. For example there might be a rule that the philosophers put down a fork after waiting ten minutes for the other fork to become available and wait a further ten minutes before making their next attempt. This scheme eliminates the possibility of deadlock (the system can always advance to a different state) but

still suffers from the problem of livelock. If all five philosophers appear in the dining room at exactly the same time and each picks up the left fork at the same time the philosophers will wait ten minutes until they all put their forks down and then wait a further ten minutes before they all pick them up again.

Resource hierarchy solution

This solution to the problem is the one originally proposed by Dijkstra. It assigns a partial order to the resources (the forks, in this case), and establishes the convention that all resources will be requested in order, and that no two resources unrelated by order will ever be used by a single unit of work at the same time. Here, the resources (forks) will be numbered 1 through 5 and each unit of work (philosopher) will always pick up the lower-numbered fork first, and then the higher-numbered fork, from among the two forks he plans to use. The order in which each philosopher puts down the forks does not matter. In this case, if four of the five philosophers simultaneously pick up their lower-numbered fork, only the highest numbered fork will remain on the table, so the fifth philosopher will not be able to pick up any fork. Moreover, only one philosopher will have access to that highest-numbered fork, so he will be able to eat using two forks.

While the resource hierarchy solution avoids deadlocks, it is not always practical, especially when the list of required resources is not completely known in advance. For example, if a unit of work holds resources 3 and 5 and then determines it needs resource 2, it must release 5, then 3 before acquiring 2, and then it must re-acquire 3 and 5 in that order. Computer programs that access large numbers of database records would not run efficiently if they were required to release all higher-numbered records before accessing a new record, making the method impractical for that purpose.

6. THE RELIABILITY TESTING IN SOFTWARE

Reliability Testing is about exercising an application so that failures are discovered and removed before the system is deployed. The purpose of reliability testing is to determine product reliability, and to determine whether the software meets the customer's reliability requirements.

According to ANSI, Software Reliability is defined as: the probability of failure-free software operation for a specified period of time in a specified environment. Software Reliability is not a direct function of time. Electronic and mechanical parts may become "old" and wear out with

time and usage, but software will not rust or wear-out during its life cycle. Software will not change over time unless intentionally changed or upgraded.

- Reliability refers to the consistency of a measure. A test is considered reliable if we get the same result repeatedly. Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. Software Reliability is also an important factor affecting system reliability.
- Reliability testing will tend to uncover earlier those failures that are most likely in actual operation, thus directing efforts at fixing the most important faults.
- Reliability testing may be performed at several levels. Complex systems may be tested at component, circuit board, unit, assembly, subsystem and system levels.
- Software reliability is a key part in software quality. The study of software reliability can be categorized into three parts:

1. Modeling
2. Measurement
3. Improvement

1. Modeling: Software reliability modeling has matured to the point that meaningful results can be obtained by applying suitable models to the problem. There are many models exist, but no single model can capture a necessary amount of the software characteristics. Assumptions and abstractions must be made to simplify the problem. There is no single model that is universal to all the situations.

2. Measurement: Software reliability measurement is naive. Measurement is far from commonplace in software, as in other engineering field. "How good is the software, quantitatively?" As simple as the question is, there is still no good answer. Software reliability cannot be directly measured, so other related factors are measured to estimate software reliability and compare it among products. Development process, faults and failures found are all factors related to software reliability.

3. Improvement: Software reliability improvement is hard. The difficulty of the problem stems from insufficient understanding of software reliability and in general, the characteristics of software. Until now there is no good way to conquer the complexity problem of software. Complete testing of a moderately complex software module is infeasible. Defect-free software product cannot be assured. Realistic constraints of time and budget severely limits the effort put into software reliability improvement.

7. CONCLUSION

The Dining philosopher problem is solved successfully in Java and MongoDB.

ASSIGNMENT NO: A5

1. TITLE

A Mobile App for Calculator having Trigonometry functionality is to be designed and tested. The data storage uses 1.text files, 2. XML Use latest open source software modelling, Designing and testing tool/Scrum-it. Implement the design using HTML-5/Scala/Python/ Java/C++/Rubi on Rails. Perform Positive and Negative testing.

2. PREREQUISITES

- Android Studio/adt-bundle-windows
- Testing tool
- JAVA, XML

3. OBJECTIVE

- To study testing tool.
- To perform Positive and Negative testing.

4. MATHEMATICAL MODEL

Let, S be the System Such that,
A={ S, E, I,O, F, DD, NDD, success, failure }
Where,
S= Start state, E=
End State, I= Set
of Input O= Set of
Out put F =Set of
Function
DD=Deterministic Data
NDD=Non Deterministic Data
Success Case: It is the case when all the inputs are given by system are entered correctly.
Failure Case: It is the case when the input does not match the validation Criteria.

5. THEORY

Android Studio Overview
Android Studio is the official IDE for Android application development, based on IntelliJ IDEA.
On top of the capabilities you expect from IntelliJ, Android Studio offers:

- Flexible Gradle-based build system
- Build variants and multiple apk file generation
- Code templates to help you build common app features
- Rich layout editor with support for drag and drop theme editing
- lint tools to catch performance, usability, version compatibility, and other problems

- Pro Guard and app-signing capabilities
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine
- And much more

Android Project Structure

By default, Android Studio displays your project files in the Android project view. This view shows a flattened version of your project's structure that provides quick access to the key source files of Android projects and helps you work with the Gradle-based build system. The Android project view:

- Shows the most important source directories at the top level of the module hierarchy.
- Groups the build files for all modules in a common folder.
- Groups all the manifest files for each module in a common folder.
- Shows resource files from all Gradle source sets.
- Groups resource files for different locales, orientations, and screen types in a single group per resource type

java/ - Source files for the module.
manifests/ - Manifest files for the
module. res/ - Resource files for the module.
Gradle Scripts/ - Gradle build and property files.

Software testing is process of verifying and validating the software or application and checks whether it is working as expected. The intent is to find defects and improve the product quality. There are two ways to test the software viz, **Positive Testing** and **Negative Testing**.

Positive testing can be performed on the system by providing the valid data as input. It checks whether an application behaves as expected with the positive input. This is to test to check the application that does what it is supposed to do so. There is a text box in an application which can accept only numbers. Entering values up to 99999 will be acceptable by the system and any other values apart from this should not be acceptable. To do positive testing, set the valid input values from 0 to 99999 and check whether the system is accepting the values.

Negative Testing can be performed on the system by providing invalid data as input. It checks whether an application behaves as expected with the negative input. This is to test the application that does not do anything that it is not supposed to do so. For example - Negative testing can be performed by testing by entering alphabets characters from A to Z or from a to z. Either system text box should not accept the values or else it should throw an error message for these invalid data inputs.

Positive Testing:

Test Case ID	Expected Result	Actual Result	Status
1	Check if all the numbers are working (0 to 9)	All the numbers are working (0 to 9)	
2	Check if the arithmetic keys (+, -, *, %, /) are working	The arithmetic keys (+, -, *, %, /) are working	

3	Check if the brackets keys are working	The bracket keys are working	
4	Check if the square and square root key is working	The square and square root key is working	
5	Check if the sin, cos, tan, cot keys are working	The sin, cos, tan, cot keys are working	
6	Check if it is showing the correct values for sin, cos, tan and cot	It is showing the correct values for sin, cos, tan and cot	
7	Check the addition of two sin and cos values	The addition of two sin and cos values	
8	Check the addition of two tan and cot values	The addition of two tan and cot values	
9	Check that it is returning the float values or integer values	It is returning the float values or integer values	
10	Check if the functionality using BODMAS/BIDMAS worksas expected	Working Properly	

Negative Testing:

Test Case ID	Expected Result	Actual Result	Status
1	Check if it is allowing letters instead of numbers	It is taking only numbers as input	
2	Check if it is returning float values instead of integer	It is returning integer values only	
3	Check if it is returning integer values instead of float	It is returning float values only	
4	Check if the functionality using BODMAS/BIDMAS works as expected	Functioning Properly	

6. CONCLUSION

A Mobile App for Calculator having Trigonometry functionality is designed and tested.

ASSIGNMENT NO: A6

1. TITLE:

A Write a program in python/ Java/ Scala/ C++/ HTML5 to implement password data encryption. Use encryption method overloading (any to methods studied)

2. PREREQUISITES:

- Latest version of 64 Bit Operating Systems Open Source Fedora-19.
- Python 2.7

3. OBJECTIVE:

- To implement password data encryption.
- To study and Use encryption method overloading.

4. Mathematical Model:

Let S be the solution perspective of the class Weather Report such
that S={s, e, i, o, f, DD, NDD, success, failure}
s=Start of program
e = the end of program
i=Plaintext data password o=
Encrypted data password f=
Found/Not Found
DD=Deterministic data
NDD=NULL
Success- plaintext encrypted
Failure- plaintext not encrypted
Computational Model

5. Theory:

"Encryption" is the conversion of electronic data into another form, called cipher text, which cannot be easily understood by anyone except authorized parties.
The primary purpose of encryption is to protect the confidentiality of digital data stored on computer systems or transmitted via the Internet or other computer networks. Modern encryption algorithms play a vital role in the security assurance of IT systems and communications as they can provide not only confidentiality, but also the following key elements of security:

- Authentication: the origin of a message can be verified.
- Integrity: proof that the contents of a message have not been changed since it was sent.
- Non-repudiation: the sender of a message cannot deny sending the message.
- While security is an afterthought for many PC users, it's a major priority for businesses of any size. It has to be when the Phenomenon Institute tells us that security breaches are costing companies millions every year.
- Even if you don't have millions to lose, protecting what you do have should be a high priority.

- There are several forms of security technology available, but encryption is one that everyday computer users should know about.
- How Encryption Works
- Encryption is an interesting piece of technology that works by scrambling data so it is unreadable by unintended parties. Let's take a look at how it works with the email-friendly software PGP (or GPG for you open source people).

Password Hashing

hash("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
hash("hbllo") = 58756879c05c68dfac9866712fad6a93f8146f337a69afe7dd238f3364946366
hash("waltz") = c0e81794384491161f1777c232bc6bd9ec38f616560b120fda8e90f383853542

Hash algorithms are one way functions. They turn any amount of data into a fixed-length "fingerprint" that cannot be reversed. They also have the property that if the input changes by even a tiny bit, the resulting hash is completely different (see the example above). This is great for protecting passwords, because we want to store passwords in a form that protects them even if the password file itself is compromised, but at the same time, we need to be able to verify that a user's password is correct.
The general workflow for account registration and authentication in a hash-based account system is as follows:

1. The user creates an account.
2. Their password is hashed and stored in the database. At no point is the plain-text (unencrypted) password ever written to the hard drive.
3. When the user attempts to login, the hash of the password they entered is checked against the hash of their real password (retrieved from the database).
4. If the hashes match, the user is granted access. If not, the user is told they entered invalid login credentials.
5. Steps 3 and 4 repeat everytime someone tries to login to their account.

In step 4, never tell the user if it was the username or password they got wrong. Always display a generic message like "Invalid username or password." This prevents attackers from enumerating valid usernames without knowing their passwords.
It should be noted that the hash functions used to protect passwords are not the same as the hash functions you may have seen in a data structures course. The hash functions used to implement data structures such as hash tables are designed to be fast, not secure.
Only **cryptographic hash functions** may be used to implement password hashing. Hash functions like SHA256, SHA512, RipeMD, and WHIRLPOOL are cryptographic hash functions.
It is easy to think that all you have to do is run the password through a cryptographic hash function and your users' passwords will be secure. This is far from the truth. There are many ways to recover passwords from plain hashes very quickly. There are several easy-to-implement techniques that make these "attacks" much less effective. To motivate the need for these techniques, consider this very website. On the front page, you can submit a list of hashes to be cracked, and receive results in less than a second. Clearly, simply hashing the password does not meet our needs for security.

Adding Salt

Lookup tables and rainbow tables are ways to break hashes, only work because each password is hashed the exact same way. If two users have the same password, they'll have the

same password hashes. We can prevent these attacks by randomizing each hash, so that when the same password is hashed twice, the hashes are not the same.

We can randomize the hashes by appending or prepending a random string, called a salt, to the password before hashing. As shown in the example above, this makes the same password hash into a completely different string every time. To check if a password is correct, we need the salt, so it is usually stored in the user account database along with the hash, or as part of the hash string itself.

The salt does not need to be secret. Just by randomizing the hashes, lookup tables, reverse lookup tables, and rainbow tables become ineffective. An attacker won't know in advance what the salt will be, so they can't pre-compute a lookup table or rainbow table. If each user's password is hashed with a different salt, the reverse lookup table attack won't work either.

To Store a Password

1.Generate a long random salt using a CSPRNG.
2.generate hash using standard cryptographic hash function such as SHA256
3.key stretching using PBKDF2 inpust name = standard cryptographic hash function such as SHA256, password = hash generated, salt = salt, round = 10000 aleast, dklen = 64 atleast length of derived key
4.Save both the salt and the hash in the user's database record.

To Validate a Password

1.Retrieve the user's salt and hash from the database.
2.Prepend the salt to the given password and hash it using the same hash function.
3.Compare the hash of the given password with the hash from the database. If they match, the password is correct. Otherwise, the password is incorrect.

Making Password Cracking Harder: Slow Hash Functions

Salt ensures that attackers can't use specialized attacks like lookup tables and rainbow tables to crack large collections of hashes quickly, but it doesn't prevent them from running dictionary or brute-force attacks on each hash individually. High-end graphics cards (GPUs) and custom hardware can compute billions of hashes per second, so these attacks are still very effective. To make these attacks less effective, we can use a technique known as **key stretching**.
The idea is to make the hash function very slow, so that even with a fast GPU or custom hardware, dictionary and brute-force attacks are too slow to be worthwhile. The goal is to make the hash function slow enough to impede attacks, but still fast enough to not cause a noticeable delay for the user.
Key stretching is implemented using a special type of CPU-intensive hash function. Don't try to invent your own–simply iteratively hashing the hash of the password isn't enough as it can be parallelized in hardware and executed as fast as a normal hash. Use a standard algorithm like PBKDF2 orbcrypt.

These algorithms take a security factor or iteration count as an argument. This value determines how slow the hash function will be. For desktop software or smartphone apps, the best way to choose this parameter is to run a short benchmark on the device to find the value

that makes the hash take about half a second. This way, your program can be as secure as possible without affecting the user experience.
If you use a key stretching hash in a web application, be aware that you will need extra computational resources to process large volumes of authentication requests, and that key stretching may make it easier to run a Denial of Service (DoS) attack on your website. I still recommend using key stretching, but with a lower iteration count. You should calculate the iteration count based on your computational resources and the expected maximum authentication request rate. The denial of service threat can be eliminated by making the user solve a CAPTCHA every time they log in. Always design your system so that the iteration count can be increased or decreased in the future.
If you are worried about the computational burden, but still want to use key stretching in a web application, consider running the key stretching algorithm in the user's browser with JavaScript. The Stanford JavaScript Crypto Library includes PBKDF2. The iteration count should be set low enough that the system is usable with slower clients like mobile devices, and the system should fall back to server-side computation if the user's browser doesn't support JavaScript. Client-side key stretching does not remove the need for server-side hashing. You must hash the hash generated by the client the same way you would hash a normal password.

RSA
RSA is a public-key encryption algorithm and the standard for encrypting data sent over the internet. It also happens to be one of the methods used in our PGP and GPG programs. Unlike Triple DES, RSA is considered an asymmetric algorithm due to its use of a pair of keys. You've got your public key, which is what we use to encrypt our message, and a private key to decrypt it. The result of RSA encryption is a huge batch of mumbo jumbo that takes attackers quite a bit of time and processing power to break.

Algorithm
The RSA algorithm involves four steps: key generation, key distribution, encryption and decryption.

RSA involves a *public key* and a *private key.* The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key.

The basic principle behind RSA is the observation that it is practical to find three very large positive integers *e*,*d* and *n* such that with modular exponentiation for all *m*:

$$(m^e)^d \equiv m \pmod{n}$$

and that even knowing *e* and *n* or even *m* it can be extremely difficult to find *d*.

Additionally, for some operations it is convenient that the order of the two exponentiations can be changed and that this relation also implies:

$$(m^d)^e \equiv m \pmod{n}$$

Key distribution[edit]

To enable Bob to send his encrypted messages, Alice transmits her public key (*n*, *e*) to Bob via a reliable, but not necessarily secret route. The private key is never distributed.

Encryption[edit]

Suppose that Bob would like to send message *M* to Alice.

He first turns *M* into an integer *m*, such that $0 \leq m < n$ and gcd(*m*, *n*) = 1 by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext *c*, using Alice's public key *e*, corresponding to

$$c \equiv m^e \pmod{n}$$

This can be done efficiently, even for 500-bit numbers, using modular exponentiation. Bob then transmits *c* to Alice.

Decryption[edit]

Alice can recover *m* from *c* by using her private key exponent *d* by computing

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

Given *m*, she can recover the original message *M* by reversing the padding scheme.

Key generation[edit]

The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers *p* and *q*.
    - For security purposes, the integers *p* and *q* should be chosen at random, and should be similar in magnitude but 'differ in length by a few digits'[2] to make factoring harder. Prime integers can be efficiently found using a primality test.
2. Compute *n* = *pq*.
    - *n* is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
3. Compute $\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1) = n - (p+q-1)$, where φ is Euler's totient function. This value is kept private.
4. Choose an integer *e* such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$; i.e., *e* and $\varphi(n)$ are coprime.
5. Determine *d* as $d \equiv e^{-1} \pmod{\varphi(n)}$; i.e., *d* is the modular multiplicative inverse of *e* (modulo $\varphi(n)$)
    - This is more clearly stated as: solve for *d* given $d \cdot e \equiv 1 \pmod{\varphi(n)}$
    - *e* having a short bit-length and small Hamming weight results in more efficient encryption – most commonly $2^{16} + 1 = 65{,}537$. However, much smaller values of *e* (such as 3) have been shown to be less secure in some settings.[13]
    - *e* is released as the public key exponent.
    - *d* is kept as the private key exponent.

The *public key* consists of the modulus *n* and the public (or encryption) exponent *e*. The *private key* consists of the modulus *n* and the private (or decryption) exponent *d*, which must be kept secret. *p*, *q*, and φ(*n*) must also be kept secret because they can be used to calculate *d*.

- An alternative, used by PKCS#1, is to choose *d* matching $de \equiv 1 \pmod{\lambda}$ with $\lambda = \text{lcm}(p-1, q-1)$, where lcm is the least common multiple. Using λ instead of φ(*n*) allows more choices for *d*. λ can also be defined using the Carmichael function, λ(*n*).

Since any common factors of (p − 1) and (q − 1) are present in the factorisation of pq − 1,[14] it is recommended that (p − 1) and (q − 1) have only very small common factors, if any besides the necessary 2.[15][2][16]

Note: The authors of RSA carry out the key generation by choosing d and then computing e as the modular multiplicative inverse of d (modulo φ(n)). Since it is beneficial to use a small value for e (i.e. 65,537) in order to speed up the encryption function, current implementations of RSA, such as PKCS#1 choose e and compute d instead.[2][17]

Example[edit]

Here is an example of RSA encryption and decryption. The parameters used here are artificially small, but one can also use OpenSSL to generate and examine a real keypair.

1. Choose two distinct prime numbers, such as

$p = 61$ and $q = 53$

2. Compute $n = pq$ giving

$$n = 61 \times 53 = 3233$$

3. Compute the totient of the product as $\varphi(n) = (p-1)(q-1)$ giving

$$\varphi(3233) = (61-1)(53-1) = 3120$$

4. Choose any number $1 < e < 3120$ that is coprime to 3120. Choosing a prime number for *e* leaves us only to check that *e* is not a divisor of 3120.

Let $e = 17$

5. Compute *d*, the modular multiplicative inverse of *e* (mod φ(*n*)) yielding,

$$d = 2753$$

Worked example for the modular multiplicative inverse:

$$d \times e \bmod \varphi(n) = 1$$

$$2753 \times 17 \bmod 3120 = 1$$

The **public key** is (*n* = 3233, *e* = 17). For a padded plaintext message *m*, the encryption function is

$$c(m) = m^{17} \mod 3233$$

The **private key** is (d = 2753). For an encrypted ciphertext c, the decryption function is

$$m(c) = c^{2753} \mod 3233$$

For instance, in order to encrypt m = 65, we calculate

$$c = 65^{17} \mod 3233 = 2790$$

To decrypt c = 2790, we calculate

$$m = 2790^{2753} \mod 3233 = 65$$

Both of these calculations can be computed efficiently using the square-and-multiply algorithm for modular exponentiation. In real-life situations the primes selected would be much larger; in our example it would be trivial to factor n, 3233 (obtained from the freely available public key) back to the primes p and q. Given e, also from the public key, we could then compute d and so acquire the private key.

Practical implementations use the Chinese remainder theorem to speed up the calculation using modulus of factors (mod pq using mod p and mod q).

The values d_p, d_q and q_{inv}, which are part of the private key are computed as follows:

$$\begin{aligned} d_p &= d \mod (p-1) = 2753 \mod (61-1) = 53 \\ d_q &= d \mod (q-1) = 2753 \mod (53-1) = 49 \\ q_{\text{inv}} &= q^{-1} \mod p = 53^{-1} \mod 61 = 38 \\ &\Rightarrow (q_{\text{inv}} \times q) \mod p = 38 \times 53 \mod 61 = 1 \end{aligned}$$

Here is how d_p, d_q and q_{inv} are used for efficient decryption. (Encryption is efficient by choice of a suitable d and e pair)

$$\begin{aligned} m_1 &= c^{d_p} \mod p = 2790^{53} \mod 61 = 4 \\ m_2 &= c^{d_q} \mod q = 2790^{49} \mod 53 = 12 \\ h &= (q_{\text{inv}} \times (m_1 - m_2)) \mod p = (38 \times -8) \mod 61 = 1 \\ m &= m_2 + h \times q = 12 + 1 \times 53 = 65 \end{aligned}$$

Conclusion:

Hence we have designed to design a program to implement password data encryption using encryption method overloading.

ASSIGNMENT NO: B1

1. TITLE:

Using Divide and Conquer Strategies to 8-Queens Matrix is Stored using JSON/XML having first Queen placed, use back-tracking to place remaining Queens to generate final 8-queen's Matrix using Python.

2. PREREQUISITES:

- 64-bit Fedora or equivalent OS with 64-bit Intel-i5/i7
- Python 2.7

3. OBJECTIVE:

- To learn the concept of Divide and Conquer Strategy.
- To study the design and implementation of 8-Queens Matrix
- To learn 8-queens matrix.

4. INPUT: Unsorted Numbers in array

5. OUTPUT: Sorted array is generated.

6. MATHEMATICAL MODEL:

Let S be the solution perspective of the class Weather Report such that

S={s, e, i, o, f, DD, NDD, success, failure}

s=Start of program

e = the end of program

i=Unsorted numbers.

o=Result of sorted data

f= Found/Not Found

DD=Deterministic data

NDD=NULL

Success-Number is found.

Failure-Number is not found

7. THEORY :

The eight queens puzzle is the problem of placing eight chess queens on an 8 8 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens

share the same row, column, or diagonal. The eight queens puzzle is an example of the more general n-queens problem of placing n queens on an n n chessboard, where solutions exist for all natural numbers n with the exception of n=2 and n=3.

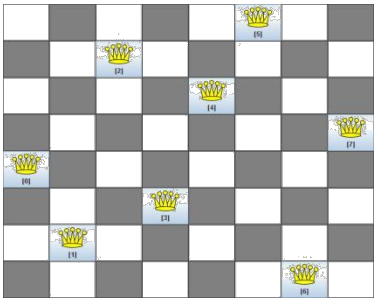


Figure 1: Queen Placing

The problem can be quite computationally expensive as there are 4,426,165,368 possible arrangements of eight queens on an 8X8 board, but only 92 solutions. The eight queens puzzle has 92 distinct solutions. If solutions that differ only by symmetry operations (rotations and reflections) of the board are counted as one, the puzzle has 12 fundamental solution.

- Find an arrangement of 8 queens on a single chess board such that no two queens are attacking one another.

- In chess, queens can move all the way down any row, column or diagonal (so long as no pieces are in the way).

Due to the first two restrictions, it's clear that each row and column of the board will haveexactly one queen.

Conditions For 8 Queen Problem :

1) No two Queens should be in Same rows.

2) No two Queens should be in Same column.

3) No two Queens should be in Same diagonal.

The backtracking strategy is as follows :

1) Place a queen on the first available square in row 1.

2) Move onto the next row, placing a queen on the first available square there (that doesn'tconflict with the previously placed queens).

3) Continue in this fashion until either :

a) you have solved the problem, or

b) You get stuck.

When you get stuck, remove the queens that got you there, until you get to a row where there is another valid square to try.

- When we carry out backtracking, an easy way to visualize what is going on is a tree that shows all the different possibilities that have been tried.

Approach :

- Create a solution matrix of the same structure as chess board.
- Whenever place a queen in the chess board, mark that particular cell in solution matrix.
- At the end print the solution matrix, the marked cells will show the positions of the queens in the chess board.

Conditions For 8 Queen Problem :

1) No two Queens should be in Same rows.
2) No two Queens should be in Same column.
3) No two Queens should be in Same diagonal.

Algorithm :

1) Place the queens column wise, start from the left most column
2) If all queens are placed.
    a) return true and print the solution matrix
3) Else
    a) Try all the rows in the current column.
    b) Check if queen can be placed here safely if yes mark the current cell in solution matrix as 1 and try to solve the rest of the problem recursively.
    c) If placing the queen in above step leads to the solution return true.
    d) If placing the queen in above step does not lead to the solution , BACKTRACK, mark the current cell in solution matrix as 0 and return false.
4. If all the rows are tried and nothing worked, return false and print NO SOLUTION

CONCLUSION:

Thus we have successfully studied Queens Matrix is Stored using JSON/XML having first Queen placed, use back-tracking to place remaining Queens to generate final 8-queen's Matrix

Assignment No. : B2

1. TITLE

Concurrent implementation of ODD-EVEN SORT is to be implemented as a web application using HTML5/ Scala/ Python/ Java. Write a debugger to test the performance of White-box testing.

2. PREREQUISITES

- 64-bit Fedora or equivalent OS with 64-bit Intel-i5/i7
- Java 1.7.0

3. OBJECTIVE

- Understand and Implement the Meaning of Odd-Even sort.
- Write test cases for White Box Testing

4. THEORY

In computing, an odd–even sort is a relatively simple sorting algorithm, developed originally for use on parallel processors with local interconnections. It is a comparison sort related to bubble sort, with which it shares many characteristics. It functions by comparing all odd/even indexed pairs of adjacent elements in the list and, if a pair is in the wrong order (the first is larger than the second) the elements are switched. The next step repeats this for even/odd indexed pairs (of adjacent elements). Then it alternates between odd/even and even/odd steps until the list is sorted.

Consider the concurrent odd-even transposition sort of an even-sized array.For 8 elements, 4 threads operate concurrently during the odd phase, and 3 threads operate concurrently during the even phase. Assume that threads, which execute during the odd phase, are named O1, O2, O3, and O4 and threads, which execute during the even phase, are named E1, E2and E3. During an odd phase, each odd-phase thread compares and exchanges a pair of numbers. Thread O1 compares the first pair of numbers, thread O2 compares the second pair of numbers, threadO3 compares the third pair, and O4 compares the fourth pair of numbers.During an even phase, the first and last elements of the array are not processed; thread E1 compares the second and third array elements, thread E2 compares the fourth and fifth array elements, and thread E3 compares the sixth and seventh elements. The algorithm runs for a total of 8 phases (i.e., equal to the number of values in the data file).

In Odd Even Sort compare the Element available on Even Index sort and for Odd sort compare the Element available on Odd index. in this assignment JSP used for Web application development purpose the remaining logic for sorting purpose is bubble sort.JSP technology is used to createweb application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc. A JSP page consists of HTML tags and JSP tags. The jsp pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc

Advantage of JSP over Servlet

There are many advantages of JSP over servlet. They are as follows

1) Extension to Servlet

JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain
JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

3) Fast Development
No need to recompile and redeploy.If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet
In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

Life cycle of a JSP Page
The JSP pages follows these phases:

- Translation of JSP Page.
- Compilation of JSP Page.
- Classloading (class file is loaded by the classloader).
- Instantiation (Object of the Generated Servlet is created).
- Initialization (jspInit() method is invoked by the container).
- Reqeust processing (_jspService() method is invoked by the container).
- Destroy (jspDestroy() method is invoked by the container).

Real-Time Object-Oriented Modeling (ROOM)
Model real time systems based on timeliness, dynamic internal structure, reactiveness, concurrency and distribution, using the ROOM notation.ROOM is an object orientedmethodology for real-time systems developed originally at Bell-Northern Research. ROOM is based upon a principle of using the same model for all phases of the development process.

ROOM models are composed of actors which communicate with each other by sending messages along protocols. Actors may be hierarchically decomposed, and may have behaviors described by ROOM charts, a variant of Harel's state charts. Descriptions of actors, protocols, and behaviors can all be reused through inheritance.

5. MATHEMATICAL MODELS
Let, S be the System Such that,
A={ S, E, I,O, F, DD, NDD, success, failure } Where,
S= Start state, E= End State, I= Set of Input
O= Set of Out put F =Set of Function
DD=Deterministic Data NDD=Non Deterministic Data

Success Case: It is the case when all the inputs are given by system are entered correctly.

Failure Case: It is the case when the input does not match the validation Criteria6.

CONCLUSION

A Web application is created for Concurrent implementation of ODD-EVEN SORT using Real time Object Oriented Modeling (ROOM).

Assignment No.: B5

1. TITLE

Write a web application using Scala/ Python/ Java /HTML5 to check the plagiarism in the given text paragraph written/ copied in the text box. Give software Modelling, Design, UML and Test cases for the same using Analysis Modelling (Static Modelling, Object Modelling, Dynamic Modelling).

2. PREREQUISITES

- 64-bit Fedora or equivalent OS with 64-bit Intel-5/i7
- Java 1.7.0

3. OBJECTIVE

- To implement the logic for Check the Plagiarism in the given text.
- Understand the Meaning of Software modeling using COMET.

4. MATHEMATICAL MODELS

Let, S be the System Such that,
A={ S, E, I,O, F, DD, NDD, success, failure }
Where,
S= Start state, E=
End State, I= Set
of Input O= Set of
Out put F =Set of
Function
DD=Deterministic Data
NDD=Non Deterministic Data
Success Case: It is the case when all the inputs are given by system are entered correctly.
Failure Case: It is the case when the input does not match the validation Criteria.

5. THEORY

Observations of plagiarism behavior in practice reveal a number of commonly found methods for illegitimate text usage, which can briefly be summarized as follows. Copy & Paste (c & p) plagiarism specifies the act of taking over parts or the entirety of a text verbatim from another author. Disguised plagiarism includes practices intended to mask literally copied segments. Undue paraphrasing defines the intentional rewriting of foreign thoughts, in the vocabulary and

style of the plagiarist without giving due credit in order to conceal the original source. Translated plagiarism is the manual or automated conversion of content from one language to another intended to cover its origin. Idea plagiarism encompasses the usage of a broader foreign concept without appropriate source acknowledgement. An Example is the appropriation of research approaches, methods, experimental setups, argumentative structures, background sources etc.

COMET

COMET is a highly iterative object-oriented software development method that addresses the requirements, analysis, and design modeling phases of the object-oriented development life cycle. The functional requirements of the system are defined in terms of actors and use cases. Each use case defines a sequence of interactions between one or more actors and the system. A use case can be viewed at various levels of detail. In a *requirements* model, the functional requirements of the system are defined in terms of actors and use cases. In an *analysis* model, the use case is realized to describe the objects that participate in the use case, and their interactions. In the *design* model.

Test Case Id	**Test case**	**Test case Description**	**Status**
1	Title Test	Title test passed	
2	Fileone input	Fileone input Passed	
3	Filetwo input	Filetwo input Passed	
4	ButtonCheck clicked	ButtonCheck passed	

Java Scanner class

To Check the Plagiarism in the given text first of all perform string compare operation also need to understand the operation related to scanner classes. There are various ways to read input from the keyboard, the java.util.Scanner class is one of them.

- The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace bydefault. It provides many methods to read and parse various primitive values.
- Java Scanner class is widely used to parse text for string and primitive types using regular expression.
- Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

java.io.PrintStream class: The PrintStream class provides methods to write data to another stream. The PrintStream class automatically flushes the data so there is no need to call flush() method. Moreover, its methods don't throw IOException.

Stream: A stream is a sequence of data.In Java a stream is composed of bytes. It's called a stream because it's like a stream of water that continues to flow. In java, 3 streams are created for us automatically. All these streams are attached with console.

1) System.out: standard output stream

2) System.in: standard input stream

3) System.err: standard error stream

OutputStream

Java application uses an output stream to write data to a destination, it may be a file,an array,peripheral device or socket.

InputStream:

Java application uses an input stream to read data from a source, it may be a file,an array,peripheral device or socket.

Output Stream class: OutputStream class is an abstract class.It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Reading data from keyboard: There are many ways to read data from the keyboard. For example:

- InputStreamReader
- Console
- Scanner
- DataInputStream etc.

InputStreamReader class: InputStreamReader class can be used to read data from keyboard.It performs two tasks:

- connects to input stream of keyboard
- converts the byte-oriented stream into character-oriented stream

The Java Console class is be used to get input from console. It provides methods to read text and password. If you read password using Console class, it will not be displayed to the user. The java.io.Console class is attached with system console internally.

6. CONCLUSION

A web application is created to check the plagiarism in the given text paragraph written/ copied in the text box.

ASSIGNMENT NO: BD1

TITLE:

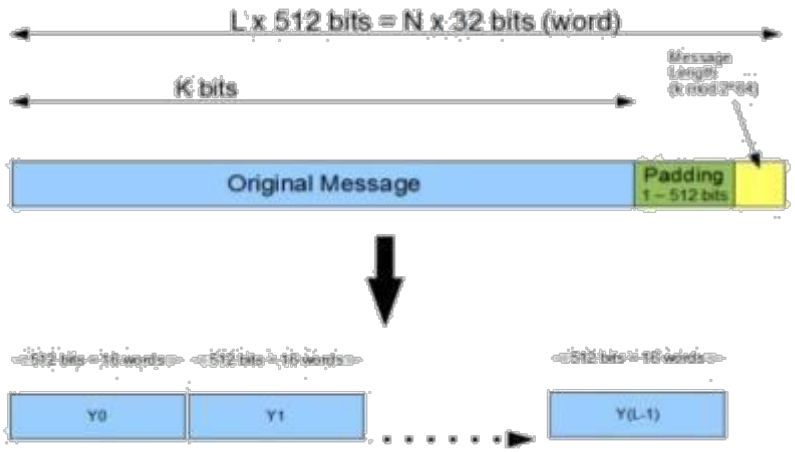
A message is to be transmitted using network resources from one machine to another, calculate and demonstrate the use of a Hash value equivalent to SHA-1. Develop program in C++/Python/Scala/Java using Eclipse.

OBJECTIVES:

To understand SHA-1 algorithm and implement a program to transmit a message from one machine to another by using a hash value equivalent to SHA-1.

THEORY:

SHA-1 (Secure Hash Algorithm) is a most commonly used from SHA series of cryptographic hash functions, designed by the National Security Agency of USA and published as their government standard. SHA-1 produces the 160-bit hash value. Original SHA (or SHA-0) also produces 160-bit hash value, but SHA-0 has been withdrawn by the NSA shortly after publication and was superseded by the revised version commonly referred to as SHA-1. The other functions of SHA series produce 224-, 256-, 384- and 512-bit hash values.



MATHEMATICAL MODEL:

Let U = {s, e, f, S, F, I, O, DD, NDD } be a universal set where,

s = start

e = end

f = set of functions

I = Input set

O = Output set

DD = Deterministic Data

ND = Non-Deterministic Data

S = Cases of Success

F = Cases of Failure

In our program, f=sha1() // which calculates the hash value of the text given as a parameter

Sha1() also consists of functions for chunking i.e. chunk(), Rot() for rotating

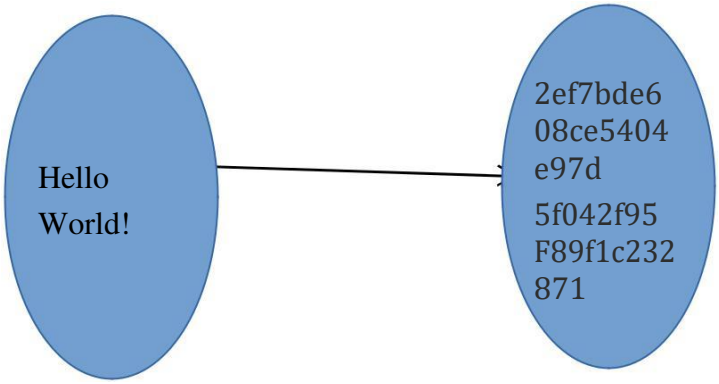
I= input text whose hash value has to be calculated

O= Result that Message has been tampered or not along with the hash value

DD= h0,h1, h2,h3,h4

Venn Diagram :

Every file will correspond to a unique hash value. Thus there is 1:1 relationship between the input file and hash values calculated. A small change in the input file makes a significant difference in the calculated hash value.



IMPLEMENTATION DETAILS / DESIGN LOGIC:

SHA1 Algorithm:-

Consider an input text to be hashed “Hello World!”

Step 0: Initialize variables

There are five variables that need to be initialized.

h0 = 01100111010001010010001100000001

h1 = 11101111110011011010101110001001

h2 = 10011000101110101101110011111110

h3 = 00010000001100100101010001110110

h4 = 11000011110100101110000111110000

Step 1: Select the input text

In this example I am going to use the string: 'Hello World!'.

Step 2: Break it into characters

H e l l o W o r l d !
Note that spaces count as characters.

Step 3: Convert characters to ASCII codes

Each character should now be converted from text into ASCII. ASCII or the 'American Standard Code for Information Interchange' is a standard that allows computers to communicate different symbols by assigning each one a number. No matter what font or language you use, the same character will always have the same ASCII code.

72 101 108 108 111 32 87 111 114 108 100 33

Step 4: Convert numbers into binary

All base ten numbers are now converted into 8-bit binary numbers. The eight-bit part means that if they don't actually take up a full eight place values, simply append zeros to the beginning so that they do.

Step 5: Add '1' to the end

Put the numbers together and add the number '1' to the end :

Step 6: Append '0's' to the end

In this step you add zeros to the end until the length of the message is congruent to 448 mod 512. That means that after dividing the message length by 512, the remainder will be 448.

Step 6.1: Append original message length

This is the last of the 'message padding' steps. You will now add the 64-bit representation of the original message length, in binary, to the end of the current message.
The message length should now be an exact multiple of 512.

Step 7: 'Chunk' the message

We will now break the message up into 512 bit chunks. In this case the message is only 512 bit's long, so there will be only one chunk that will look exactly the same as the last step.

Step 8: Break the 'Chunk' into 'Words'

Break each chunk up into sixteen 32-bit words

Step 9: 'Extend' into 80 words

This is the first sub-step. Each chunk will be put through a little function that will create 80 words from the 16 current ones.
This step is a loop. What that means is that every step after this will be repeated until a certain condition is true.
In this case we will start by setting the variable 'i' equal to 16. After each run through of the loop we will add 1 to 'i' until 'i' is equal to 79.

We begin by selecting four of the current words. The ones we want are: [i-3], [i-8], [i-14] and [i-16]. That means for the first time through the loop we want the words numbered: 13, 8, 2 and 0.

0: 01000001001000000101010001100101

2: 00000000000000000000000000000000

8: 00000000000000000000000000000000

13: 00000000000000000000000000000000

Now that we have our words selected we will start by performing what's known as an 'XOR' or 'Exclusive OR' on them. In the end all four words will be XOR'ed together.

00000000000000000000000000000000 :word 13

00000000000000000000000000000000 :word 8

00000000000000000000000000000000 :word 13 XOR word 8

00000000000000000000000000000000 :word 13 XOR word 8
01110010011011000110010000100001 :word 2

01110010011011000110010000100001 :(word 13 XOR word 8) XOR word 2

01110010011011000110010000100001 (word 13 XOR word 8) XOR word 2

01001000011001010110110001101100 :word 0

00111010000010010000100001001101
((word 13 XOR word 8) XOR word 2) XOR word 0

01110100000100100001000010011010 :Left Rotated 1

After step nine is complete we will now have 80 words. The last 5 have been listed below:

75: 01100011110010111001110001110101
76: 01001100000101110100111011110000
77: 01001000000100111100011010100001
78: 01011100010101101011111100111010
79: 10101011100010110110010010110100

Step 10: Initialize variables

Set the letters A-->E equal to the variables h0-->h4.

A = h0

B = h1

C = h2

D = h3

E = h4

Step 11: The main loop

This loop will be run once for each word in succession.

Step 11.1: Four choices

Depending on what number word is being input, one of four functions will be run on it.

Words 0-19 go to function 1.
Words 20-39 go to function 2
Words 40-59 go to function 3
Words 60-79 go to function 4

Function1 : (B AND C) or (!B AND D)

Function2 : B XOR C XOR D
Function3 : (B AND C) OR (B AND D) OR (C AND D)
Function4 : B XOR C XOR D

After completing one of the four functions above, each variable will move on to this sub step before restarting the loop with the next word. For this step we are going to create a new variable called 'temp' and set it equal to: (A left rotate 5) + F + E + K + (the current word).

Step 12:

h0 = h0 + A

h1 = h1 + B

h2 = h2 + C

h3 = h3 + D

h4 = h4 + E

If these variables are longer than 32 bits they should be truncated.

Finally the variables are converted into base 16 (hex) and joined together.

2ef7bde6	:h0 in hex
8ce5404	:h1 in hex
e97d5f04	:h2 in hex
2f95f89f	:h3 in hex
1c232871	:h4 in hex

2ef7bde608ce5404e97d5f042f95f89f1c232871 :digest

Input :- Hello World!

Expected Output :- 2ef7bde608ce5404e97d5f042f95f89f1c232871
If hash values match at both machines then message is not tampered.

TEST CASES:

hello World!	788245B4dad73c1e5a630c126c484c7a2464f280
Hello World!	2ef7bde608ce5404e97d5f042f95f89f1c232871

For every file , hash value is calculated at the sending machine and appended with the file. This appended file is sent over the network to the destination machine and hash value is calculated again at the destination. If the hash values match then message is securely transferred.

CONCLUSION:

We have successfully implemented this program to use of a Hash value equivalent to SHA-1.

ASSIGNMENT NO: BD2

TITLE:
Write a program to generate a pseudorandom number generator for generating the long-term private key and the ephemeral keys used for each signing based on SHA-1 using Python/Java/C++. Disregard the use of existing pseudorandom number generators available.

PREREQUISITES

- 64-bit Fedora or equivalent OS with 64-bit Intel-i5/i7
- Python 2.7

OBJECTIVES:

1. To develop problem solving abilities using Mathematical Modeling.
2. To understand the use and working of Pseudorandom number generator.

MATHEMATICAL MODEL:

Let P be the solution perspective.

Let, S be the System Such that,
A= {S, E, I, O, F, DD, NDD, success, failure}
Where,
S= Start state, E=
End State, I= Set
of Input O= Set of
Out put F =Set of
Function
DD=Deterministic Data
NDD=Non Deterministic Data
Success Case: It is the case a pseudorandom number is generated.
Failure Case: It is the case when some exception occurs and pseudorandom number is not generated.

THEORY:

For the purpose of generating pseudorandom number we are using Mersenne Twister Algorithm.

The **Mersenne Twister** is a pseudorandom number generator (PRNG). It is by far the most widely used general-purpose PRNG.[1] Its name derives from the fact that its period length is chosen to be a Mersenne prime.

For a w-bit word length, the Mersenne Twister generates integers in the range $[0, 2^w-1]$.

The Mersenne Twister algorithm is based on a matrix linear recurrence over a finite binary field F_2. The algorithm is a twistedgeneralised feedback shift register[41] (twisted GFSR, or TGFSR) of rational normal form (TGFSR(R)), with state bit reflection and tempering. The basic idea is to define a series x_i through a simple recurrence relation, and then output numbers of the form $x_i T$, where T is an invertible F_2 matrix called a tempering matrix.

The general algorithm is characterized by the following quantities (some of these explanations make sense only after reading the rest of the algorithm):

- w: word size (in number of bits)
- n: degree of recurrence
- m: middle word, an offset used in the recurrence relation defining the series $\boldsymbol{x}$, $1 \le m < n$
- r: separation point of one word, or the number of bits of the lower bitmask, $0 \le r \le w - 1$
- a: coefficients of the rational normal form twist matrix
- b, c: TGFSR(R) tempering bitmasks
- s, t: TGFSR(R) tempering bit shifts
- u, d, l: additional Mersenne Twister tempering bit shifts/masks

with the restriction that $2^{nw - r} - 1$ is a Mersenne prime. This choice simplifies the primitivity test and k-distribution test that are needed in the parameter search.

The series $\boldsymbol{x}$ is defined as a series of w-bit quantities with the recurrence relation:

$$x_{k+n} := x_{k+m} \oplus (x_k^u \mid x_{k+1}^l) A \qquad k = 0, 1, \ldots$$

where $\mid$ denotes the bitwise or, $\oplus$ the bitwise exclusive or (XOR), x_k^u means the upper $w - r$ bits of x_k, and x_{k+1}^l means the lower r bits of x_{k+1}. The twist transformation A is defined in rational normal form as:

$$A = \begin{pmatrix} 0 & I_{w-1} \\ a_{w-1} & (a_{w-2}, \ldots, a_0) \end{pmatrix}$$

with I_{n-1} as the $(n-1) \times (n-1)$ identity matrix. The rational normal form has the benefit that multiplication by A can be efficiently expressed as: (remember that here matrix multiplication is being done in F_2, and therefore bitwise XOR takes the place of addition)

$$\boldsymbol{x}A = \begin{cases} \boldsymbol{x} \gg 1 & x_0 = 0 \\ (\boldsymbol{x} \gg 1) \oplus \boldsymbol{a} & x_0 = 1 \end{cases}$$

where x_0 is the lowest order bit of x.

As like TGFSR(R), the Mersenne Twister is cascaded with a tempering transform to compensate for the reduced dimensionality of equidistribution (because of the choice of *A*being in the rational normal form). Note that this is equivalent to using the matrix *A* ***where*** $\mathbf{A} = T^{-1}AT$ for *T* an invertible matrix, and therefore the analysis of characteristic polynomial mentioned below still holds.

As with *A*, we choose a tempering transform to be easily computable, and so do not actually construct *T* itself. The tempering is defined in the case of Mersenne Twister as

$y := x \oplus ((x >> u) \,\&\, d)$ $y := y \oplus ((y << s) \,\&\, b)$ $y := y \oplus ((y << t) \,\&\, c)$ $z := y \oplus (y >> l)$

where ***x*** is the next value from the series, ***y*** a temporary intermediate value, ***z*** the value returned from the algorithm, with <<, >> as the bitwise left and right shifts, and & as the bitwise and. The first and last transforms are added in order to improve lower-bit equidistribution. From the property of TGFSR, $s + t \geq \lfloor w/2 \rfloor - 1$ is required to reach the upper bound of equidistribution for the upper bits.

The coefficients for MT19937 are:

- $(w, n, m, r) = (32, 624, 397, 31)$
- $a = 9908B0DF_{16}$
- $(u, d) = (11, FFFFFFFF_{16})$
- $(s, b) = (7, 9D2C5680_{16})$
- $(t, c) = (15, EFC60000_{16})$
- $l = 18$

Note that 32-bit implementations of the Mersenne Twister generally have $d = FFFFFFFF_{16}$. As a result, the *d* is occasionally omitted from the algorithm description, since the bitwise and with *d* in that case has no effect.

The coefficients for MT19937-64 are:[42]

- $(w, n, m, r) = (64, 312, 156, 31)$
- $a = B5026F5AA96619E9_{16}$
- $(u, d) = (29, 5555555555555555_{16})$
- $(s, b) = (17, 71D67FFFEDA60000_{16})$
- $(t, c) = (37, FFF7EEE000000000_{16})$
- $l = 43$

CONCLUSION:

Hence, we have written program which is capable of generating pseudorandom number without using existing pseudorandom number generator available.

ASSIGNMENT NO: BD4

TITLE:
Write a Python/ Java program to validate the parameter tuple for the security of the DSA. Design necessary classes. Use Miller-Rabin primality testing may be used.

Learning Objective:
1 Implementation of the problem statement using Object oriented programming. Understand the concept of DSA and Miller-Rabin primality Test.

Theory:
Introduction
The Miller–Rabin primality test or Rabin–Miller primality test is a primality test: an algorithm which determines whether a given number is prime or not. After receiving the message user wants to know that whether it is the original message he uses verification algorithm. The message and the digital signature is the input for this algorithm and decrpts the message using senders public key. If the sign matches the message is original.
Signature verification Algorithm To verify a message signature, the receiver of the message and the digital signature can follow these steps:

- ✓ Generate the message digest h, using the same hash algorithm.
- ✓ Compute w, such that s*w mod q = 1. w is called the modular multiplicative inverse of s modulo q.

- ✓ Compute u1 = h*w mod q.
- ✓ Compute u2 = r*w mod q.
- ✓ Compute v = (((g**u1)*(y**u2)) mod p) mod q.
- ✓ If v == r, the digital signature is valid

Miller–Rabin primality test Algorithm:- Input :A number N to be tested and a variable iteration-the number of 'a' for which algorithm will test N. **Output :**0 if N is definitely a composite and 1 if N is probably a prime. Write N as For each iteration Pick a random a in [1,N-1] x = mod n if x =1 or x = n-1 Next iteration for r = 1 to s-1 x = mod n if x = 1 return false if x = N-1 Next iteration return false return true Here's a python implementation :
import random
def modulo(a,b,c): x = 1 y = a while b>0:
if b%2==1:
x = (x*y)%c y = (y*y)%c
b = b/2
return x%c def millerRabin(N,iteration):
if N<2:
return False if N!=2 and N%2==0:
return False d=N-1 while d%2==0:
d = d/2 for i in range(iteration):
a = random.randint(1, N-1) temp = d x = modulo(a,temp,N) while (temp!=N-1 and x!=1 and x!=N-1):
x = (x*x)%N temp = temp*2 if (x!=N-1 and temp%2==0):
return False

return True

MATHEMATICAL MODEL:

Let S be the solution perspective of the class Weather Report such that
S= {s, e, i, o, f, success, failure}
s=Start of program
e = the end of program
i=Message
o= Message True or false after Verifying digital signature.
Success- Data receive with digital signature
Failure- Signature not Verified

CONCLUSION:

Hence, we have written program to validate the parameter tuple for the security of the DSA. Miller-Rabin primarily testing may be used.

Experiment Number: C2

TITLE:
Install and Use Latest IDS (Open Source)

OBJECTIVES:

1. To develop problem solving abilities using Mathematical Modeling.
2. To apply algorithmic strategies while solving problems.

THEORY:

An intrusion detection system (IDS) inspects all inbound and outbound network activity and identifies suspicious patterns that may indicate a network or system attack from someone attempting to break into or compromise a system.

There are several ways to categorize IDS:

- **Misuse detection** vs. **anomaly detection**: in misuse detection, the IDS analyzes the information it gathers and compares it to large databases of attack signatures. Essentially, the IDS looks for a specific attack that has already been documented. Like a virus detection system, misuse detection software is only as good as the database of attack signatures that it uses to compare packets against. In anomaly detection, the system administrator defines the baseline, or normal, state of the networks traffic load, breakdown, protocol, and typical packet size. The anomaly detector monitors network segments to compare their state to the normal baseline and look for anomalies.
- **Network-based** vs. **host-based systems**: in a network-based system, or NIDS, the individual packets flowing through a network are analyzed. The NIDS can detect malicious packets that are designed to be overlooked by a firewall simplistic filtering rules. In a host-based system, the IDS examines at the activity on each individual computer or host.
- **passive system** vs. **reactive system**: in a passive system, the IDS detects a potential security breach, logs the information and signals an alert. In a reactive system, the IDS responds to the suspicious activity by logging off a user or by reprogramming the firewall to block network traffic from the suspected malicious source.
- IPS and IDS:

  Ans:

Sr no.	Parameters	Intrusion Prevention System	Intrusion Detection System
1.	Placement in network infrastructure	Part of the direct line of communication(inline)	Outside direct line of communication(Out of band)
2.	System Type	Active and/or passive	Passive
3.	Detection Mechanisms	1.Stastical anomaly based detection	Signature detection

		2. Signature detection	
4.	Usefulness	Ideal for blocking web destruction	Ideal for identifying blocking attacks
5.	Traffic Requirement	"Original" traffic is required	Traffic replication is required

Types of IDS:

1. Active and passive IDS:

An **active Intrusion Detection Systems (IDS)** is also known as Intrusion Detection and Prevention System (IDPS). Intrusion Detection and Prevention System (IDPS) is configured to automatically block suspected attacks without any intervention required by an operator.

A **passive IDS** is a system that's configured to only monitor and analyze network traffic activity and alert an operator to potential vulnerabilities and attacks. A passive IDS is not capable of performing any protective or corrective functions on its own.

2. Network Intrusion detection systems (NIDS) and Host Intrusion detection systems (HIDS)

Network Intrusion Detection Systems (NIDS) usually consists of a network appliance (or sensor) with a Network Interface Card (NIC) operating in promiscuous mode and a separate management interface. The IDS is placed along a network segment or boundary and monitors all traffic on that segment.

A **Host Intrusion Detection Systems (HIDS)** and software applications (agents) installed on workstations which are to be monitored. The agents monitor the operating system and write data to log files and/or trigger alarms.

3. Knowledge-based (Signature-based) IDS and behavior-based (Anomaly-based) IDS

A **knowledge-based (Signature-based) Intrusion Detection Systems (IDS)** references a database of previous attack signatures and known system vulnerabilities. The meaning of word signature, when we talk about Intrusion Detection Systems (IDS) is recorded evidence of an intrusion or attack. Each intrusion leaves a footprint behind (e.g., nature of data packets, failed attempt to run an application, failed logins, file and folder access etc.). These footprints are called signatures.

A Behavior-based (Anomaly-based) Intrusion Detection Systems (IDS) references a baseline or learned pattern of normal system activity to identify active intrusion attempts. Deviations from this baseline or pattern cause an alarm to be triggered.

Advantages:

1. It can detect the unauthorized user
2. It can detect password cracking and denial of services
3. It can catch illegal data manipulations
4. Monitors the operations of firewalls
5. Allows administrator to tune, organize operating system audit trails other logs

Disadvantages:

1. Host-based IDS works well for a single machine, extremely labor-intensive of monitor multiple machines.
2. If host is compromised then no more alerts will be generated
3. There is no detection with unknown signatures

Example:

Examples of Network IDS:

- SNORT

Examples of HIDS:

- OSSEC - Open Source Host-based Intrusion Detection System
- Tripwire
- AIDE - Advanced Intrusion Detection Environment
- Prelude Hybrid IDS

Snort is a free and open source network intrusion prevention system (NIPS) and network intrusion detection system (NIDS) created by Martin Roesch in 1998. In 2009, Snort entered InfoWorld's Open Source Hall of Fame as one of the "greatest [pieces of] open source software of all time".

Snort's open source network-based intrusion detection system (NIDS) has the ability to perform real-time traffic analysis and packet logging on Internet Protocol (IP) networks. Snort performs protocol analysis, content searching and matching. These basic services have many purposes including application-aware triggered quality of service, to de-prioritize bulk traffic when latency-sensitive applications are in use.

The program can also be used to detect probes or attacks, including, but not limited to, operating system fingerprinting attempts, common gateway interface, buffer overflows, server message block probes, and stealth port scans.

Snort can be configured in three main modes: sniffer, packet logger, and network intrusion detection. In sniffer mode, the program will read network packets and display them on the console. In packet logger mode, the program will log packets to the disk. In intrusion detection mode, the program will monitor network traffic and analyze it against a rule set

defined by the user. The program will then perform a specific action based on what has been identified.

Installation steps for snort:

First we need to install all the prerequisites from the Ubuntu repositories:

```
sudo apt-get install -y build-essential libpcap-dev libpcre3-dev libdumbnet-dev
bison flex zlib1g-dev liblzma-dev openssl libssl-dev
```

Breakdown of the packages you are installing:

build-essential: provides the build tools (GCC and the like) to compile software.

bison, flex: parsers required by DAQ (DAQ is installed later below).

libpcap-dev: Library for network traffic capture required by Snort.

libpcre3-dev: Library of functions to support regular expressions required by Snort.

libdumbnet-dev: the libdnet library provides a simplified, portable interface to several low-level networking routines. Many guides for installing Snort install this library from source, although that is not necessary.

zlib1g-dev: A compression library required by Snort.

liblzma-dev: Provides decompression of swf files (adobe flash)

openssl and libssl-dev: Provides SHA and MD5 file signatures

Next install ethtool:

```
sudo apt-get install -y ethtool
```

now edit /etc/network/interfaces as an admin and append the following two lines for each network interface we will have Snort listen on

(in our case eth0):

```
post-up ethtool -K eth0 gro off
```

```
post-up ethtool -K eth0 lro off
```

Important note for people running Ubuntu 15.10: In Ubuntu 15.10, for new installations (not upgrades), network interfaces no longer follow the ethX standard (eth0, eth1, …). Instead, interfaces names are assigned as Predictable Network Interface Names. This means you need to check the names of your interfaces using ifconfig -a and replace eth0 with whatever network interface you find.

Next we will create a directory to save the downloaded tarball files:

```
mkdir snort_src
```

Snort uses the Data Acquisition library (DAQ) to abstract calls to packet capture libraries. DAQ is downloaded and installed from the Snort website:

```
cd snort_src
wget https://www.snort.org/downloads/snort/daq-
2.0.6.tar.gz tar -xvzf daq-2.0.6.tar.gz
cd daq-2.0.6
./configure
make
sudo make install
```

Now we are ready to install Snort from source. When we configure the build of Snort, we use the –enable-sourcefire flag, which enables Packet Performance Monitoring (PPM), and matches the way the sourcefire team builds Snort.

```
cd snort_src
wget https://www.snort.org/downloads/snort/snort-
2.9.8.0.tar.gz tar -xvzf snort-2.9.8.0.tar.gz
cd snort-2.9.8.0
./configure --enable-
sourcefire make
```

```
sudo make install
```

Run the following command to update shared libraries:

```
sudo ldconfig
```

Since the Snort installation places the Snort binary at /usr/local/bin/snort, it is a good policy to create a symlink to /usr/sbin/snort:

```
sudo ln -s /usr/local/bin/snort /usr/sbin/snort
```

The last step of our Snort installation is to test that the Snort Binary runs. Execute Snort with the -V flag, which causes Snort to show the version number:

```
/usr/sbin/snort -V
```

you should see output similar to the following:

```
user@snortserver:~$ /usr/sbin/snort -V

   ,,_     -*&amp;gt; Snort! &amp;lt;*-
  o&amp;quot;  )~   Version 2.9.8.0 GRE (Build 229)
   ''''    By Martin Roesch &amp;amp; The Snort Team:
http://www.snort.org/contact#team
           Copyright (C) 2014 Cisco and/or its affiliates. All rights
           reserved. Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.5.3
           Using PCRE version: 8.31 2012-07-
           06 Using ZLIB version: 1.2.8

user@snortserver:~$
```

For all the following instructions become a superuser and then execute them.

Create the following directories and files:

```
mkdir /etc/snort

mkdir /etc/snort/rules

mkdir /var/log/snort

mkdir /etc/snort/preproc_rules

touch /etc/snort/rules/white_list.rules

touch /etc/snort/rules/black_list.rules

touch /etc/snort/rules/local.rules
```

Change permissions:

```
chmod -R 5775 /etc/snort/

chmod -R 5775 /var/log/snort/

chmod -R 5775 /usr/local/lib/snort
```

Copy *.conf and *.map files from snort download directory to /etc/snort:

```
cp /home/snort_src/snort-2.9.8.0/etc/*.conf* /etc/snort/

cp -v /home/snort_src/snort-2.9.8.0/etc/*.map* /etc/snort/
```

Before editing snort.conf get the backup of that file first:

```
cp /etc/snort/snort.conf /etc/snort/snort.conf_orig
```

Give following Command:

```
sed -i 's/include \$RULE\_PATH/#include \$RULE\_PATH/' /etc/snort/snort.conf
```

Note:Above Command will comment all rulesets which we will edit line by line

Go to line 45 of /etc/snort/snort.conf, edit to make like below

```
ipvar HOME_NET 192.168.1.0/24
ipvar EXTERNAL_NET !$HOME_NET
```

(Note: replace above ip address with your ip address)

MATHEMATICAL MODEL:

Let P be the solution perspective.

P ={ S , E , I , O, F }

S = { Initial state of the IDS, Initial snort configuration file }

I = Input of the system → { I1 , I2
} where,
I1 = { List of local rules }
I2 = { List of blocked IP's }

O = Output of the system → { O1 , O2, O3, O4 }
where,
O1 = { list_blocked_ip }
O2 = { display_firewall_rules}
O3 = {Display network traffic}
O4 = {Allow_IDS_ON/OFF}

F = Functions used → { f1 , f2 , f3
} where
f1 = { IDS on / off }
f2 = { Blocked_ip's }
f3 = { Display network traffic }
f4 = {Display_rules}
E = End state of the system shows successfully installing snort and handling and updating the IDS and firewall rules of the system.

Input :-
List of blocked IP's and list of firewall rules

Expected Output :-

1. Allow IDS on/off
2. Display Blocked IP list
3. Display Firewall rules
4. Display network traffic

Execute the program with the following commands.
Note: Make sure you should be sudo user.
>sudo su
>Now start snort as a daemon process with the help of -D option and set the file path where the logs will be generated using following command :
snort -D -c /etc/snort/snort.conf -l /var/log/snort/
>To view the current network traffic along with packet headers and the data
do: snort -vd
>To view the list of blocked IP addresses do:
cat /etc/snort/rules/iplists/black_list.rules
>To view the list of firewall rules do:
cat /etc/snort/rules/local.rules
>To kill the snort daemon
do: pkill snort

TEST CASES:

Test ID	Description	Input	Expected output	Actual output
1.	List all blocked IPs	List of IP's to be blocked	Should list all blocked IPs	Listing all blocked IPs
2.	Start IDS	IDS install	Should start the IDS	IDS got started
3.	Start IDS	IDS not install	Should provide the exception and not able to start IDS	IDS not started
4.	List Firewall rules	Local rules	Should display Firewall rules	Displaying firewall rules

CONCLUSION:

Hence based on the rule of firewall and IDS, we have learnt about the blocking and unblocking the IPs from the IP tables and configuring the IDS.