

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,

$A = \{ S, E, I, O, F, DD, NDD, \text{success}, \text{failure} \}$

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 \mathbf{m} and \mathbf{r} be the multiplicand and multiplier, respectively; and let x and y represent the number of bits in \mathbf{m} and \mathbf{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 \mathbf{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 \mathbf{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 $2^i - 2^{i-j}$ as:

$$\begin{aligned}
 2^i - 2^{i-j} &= 2^{i-j} \times (2^j - 1) \\
 &= 2^{i-j} \times (2^{j-1} + 2^{j-2} + \dots + 2^0) \\
 &= 2^{i-1} + 2^{i-2} + \dots + 2^{i-j}
 \end{aligned}$$

For example $0111_{\text{two}} = 23_{\text{ten}} - 20_{\text{ten}}$. So $0010_{\text{two}} \times 0111_{\text{two}}$ can be written as:

$0010_{\text{two}} \times (1000_{\text{two}} - 0001_{\text{two}})$

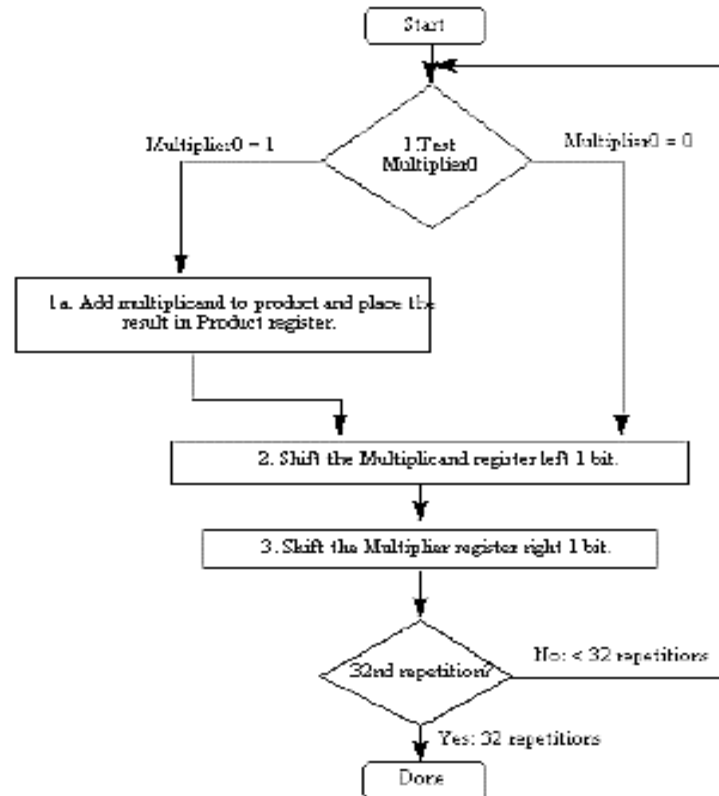
Or to make it look like the multiplication before:

```

    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.

File: Booth.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.*;

public interface Booth extends Remote {
    public TreeMap<String, int[]> multiply(int n1, int n2)
        throws RemoteException;
    public int getDecimal(int[] B)
        throws RemoteException;
    public void rightShift(int[] A)
        throws RemoteException;
    public void add(int[] A, int[] B)
        throws RemoteException;
    public int[] binary(int n)
        throws RemoteException;
}
```

File: BoothImpl.java

```
import java.rmi.RemoteException;
import java.util.*;

public class BoothImpl implements Booth {

    public TreeMap<String, int[]> multiply(int n1, int n2)
    {
        int[] m = binary(n1);
        int[] m1 = binary(-n1);
        int[] r = binary(n2);
        int[] A = new int[9];
        int[] O = new int[9];
        int[] P = new int[9];
        for (int i = 0; i < 4; i++)
        {
            A[i] = m[i];
            O[i] = m1[i];
            P[i + 4] = r[i];
        }

        TreeMap<String, int[]> map = new TreeMap<String, int[]>();
        map.put("A", A);

        map.put("O", O);
        map.put("P", P);

        for (int i = 0; i < 4; i++)
        {
```

```

        if (P[7] == 0 && P[8] == 0);
            // do nothing
        else if (P[7] == 1 && P[8] == 0)
            add(P, 0);
        else if (P[7] == 0 && P[8] == 1)
            add(P, A);
        else if (P[7] == 1 && P[8] == 1);
            // do nothing
        rightShift(P);
        map.put("P"+Integer.toString(i), P);
    }
    int [] res = {getDecimal(P)};
    map.put("Result",res);
    return map;
}
/** Function to get Decimal equivalent of P */
public int getDecimal(int[] B)
{
    int p = 0;
    int t = 1;
    for (int i = 7; i >= 0; i--, t *= 2)
        p += (B[i] * t);
    if (p > 64)
        p = -(256 - p);
    return p;
}
/** Function to right shift array */
public void rightShift(int[] A)
{
    for (int i = 8; i >= 1; i--)
        A[i] = A[i - 1];
}
/** Function to add two binary arrays */
public void add(int[] A, int[] B)
{
    int carry = 0;
    for (int i = 8; i >= 0; i--)
    {
        int temp = A[i] + B[i] + carry;
        A[i] = temp % 2;
        carry = temp / 2;
    }
}
/** Function to get binary of a number */
public int[] binary(int n)
{
    int[] bin = new int[4];
    int ctr = 3;
    int num = n;
    /** for negative numbers 2 complement */
    if (n < 0)
        num = 16 + n;
    while (num != 0)
    {
        bin[ctr--] = num % 2;
        num /= 2;
    }
    return bin;
}

```



```
}
```

File: Server.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server {

    public Server() {}

    public static void main(String args[]) {

        try {
            LocateRegistry.createRegistry(1099);
            BoothImpl robj = new BoothImpl();
            Booth stub = (Booth) UnicastRemoteObject.exportObject(robj, 0);

            Registry registry = LocateRegistry.getRegistry();
            registry.bind("boothmul", stub);
            System.out.println("Server is ready to listen... ");

        } catch (Exception e) {
            System.err.println("Server exception thrown: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

File: Client.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.*;

public class Client {
    private static Booth stub = null;
    private Client() {}

    public static void main(String[] args) {

        int n1=7,n2=-7;
        try {
            Registry reg = LocateRegistry.getRegistry("localhost");
            stub = (Booth) reg.lookup("boothmul");

        } catch (Exception e) {
            System.err.println("Client exception thrown: " + e.toString());
            e.printStackTrace();
        }
        if (args.length == 2) {
            try {
                n1 = Integer.parseInt(args[0]);
            }
        }
    }
}
```

```
        n2 = Integer.parseInt(args[1]);
    }
    catch (Exception e) {
        System.out.println("Wrong input " + e.getMessage() );
        System.exit(0);
    }
    print(n1,n2);

} else {
    System.out.println("java Client int1<-7 to 7> int2<-7 to 7>");
    System.out.println("\nFor example: java Client 7 -7 ");
    System.out.println("\nYou will get the output like the following: \n");
    print(n1,n2);
    System.exit(0);
}

}

public static void print(int n1,int n2){
    TreeMap<String, int[]> result;
    try {
        result = stub.multiply(n1,n2);
        for(Map.Entry m:result.entrySet()){
            System.out.print(m.getKey());
            System.out.println(Arrays.toString(result.get(m.getKey())));
        }
    }catch(Exception e) {
        System.out.println("Remote method exception thrown: " + e.getMessage());
    }
}

}
```

```
/*
cipher@blackfury-HP-eNVy:~/be-2/A3$ java Server
Server is ready to listen...
cipher@blackfury-HP-eNVy:~/be-2/A3$ java Client
Usage: java Client int1<-7 to 7> int2<-7 to 7>
```

For example: java Client 7 -7

You will get the output like the following:

```
A[0, 1, 1, 1, 0, 0, 0, 0, 0]
O[1, 0, 0, 1, 0, 0, 0, 0, 0]
P[1, 1, 0, 0, 1, 1, 1, 1, 1]
P0[1, 1, 0, 0, 1, 1, 1, 1, 1]
P1[1, 1, 0, 0, 1, 1, 1, 1, 1]
P2[1, 1, 0, 0, 1, 1, 1, 1, 1]
P3[1, 1, 0, 0, 1, 1, 1, 1, 1]
Result[-49]
cipher@blackfury-HP-eNVy:~/be-2/A3$ java Client 5 6
A[0, 1, 0, 1, 0, 0, 0, 0, 0]
O[1, 0, 1, 1, 0, 0, 0, 0, 0]
P[0, 0, 0, 1, 1, 1, 1, 0, 0]
P0[0, 0, 0, 1, 1, 1, 1, 0, 0]
P1[0, 0, 0, 1, 1, 1, 1, 0, 0]
P2[0, 0, 0, 1, 1, 1, 1, 0, 0]
P3[0, 0, 0, 1, 1, 1, 1, 0, 0]
Result[30]

*/
```