SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJS22ITL502 **DATE: 23-08-24**

COURSE NAME: Advanced Data Structures Laboratory **CLASS:** TY B. TECH

NAME: Anish Sharma **DIV:** IT1-1 **ROLL:** I011

EXPERIMENT NO. 1

CO/LO: Carry out Amortized Analysis of Algorithms.

AIM / OBJECTIVE: To implement a k-bit Binary Counter that counts upward and perform Amortized Analysis.

DESCRIPTION OF EXPERIMENT:

We use an array A [0, …, k – 1) of bits, where A.length = k, as the counter. A binary number x that is stored in the counter has its lowest-order bit in A[0] and its highest-order bit in A[k – 1], so that

$$x = \sum_{i=0}^{k-1} A[i].\, 2^{i}$$

Initially, x = 0, and thus A[i] = 0 for i = 0, 1, … k - 1.

ALGORITHM:

The algorithm begins by initializing the binary string B of length k and reading the number of bits k. A menu is displayed with options to either increment the binary counter or exit the program. When the user opts to increment the counter, the current binary string B is displayed. The program then performs an amortized analysis by setting the initial cost to 1 and traversing the binary string from right to left. For each 0 encountered, the cost is incremented, and the traversal stops upon encountering a 1. The total cumulative cost is updated and displayed. The binary number is then

incremented by flipping 1s to 0s from the rightmost bit until a 0 is found and flipped to 1. After incrementing, the updated binary string B is displayed. The program continues to loop back to the menu, allowing the user to increment the counter or exit.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Function to increment the binary number
void incrementBinary(char *binary, int k) {
int i = k - 1; while (i >= 0 && binary[i] ==
'1') { binary[i] = '0'; i--; } if (i >= 0) {
binary[i] = '1';
  } } void displayBinary(const char *label, char
*binary) { printf("%s %s\n", label, binary);
} void amortizedAnalysis(char *binary, int k) { static int
totalCost = 0; int cost = 1; // The cost of incrementing by
flipping bits


  for (int i = k - 1; i >= 0 && binary[i] == '0'; i--) { cost++;
  }


  totalCost += cost;
```

```
printf("Operation cost: %d, Total cost so far: %d\n", cost, totalCost);
}

// Menu-driven program void menu() { int k,
choice; char binary[33]; while (1) {
printf("\nMenu:\n"); printf("1. Increment
Binary Counter\n");

printf("2. Exit\n"); printf("Enter
    your choice: "); scanf("%d",
    &choice);

    switch (choice) { case
      1:
      printf("Enter the value of k (number of bits): ");
  scanf("%d", &k);

  printf("Enter the initial binary number: ");
  scanf("%s", binary); displayBinary("Original
  Binary:", binary); amortizedAnalysis(binary, k);
  incrementBinary(binary, k); displayBinary("After
  Incrementing:", binary); break;
      case 2:
        exit(0);
        break;
      default:
        printf("Invalid choice! Please try again.\n");
    }
  }
}
```

```
int main() {
    menu();
    return 0;
}
```

```
Menu:
1. Increment Binary Counter
2. Exit
Enter your choice: 1
Enter the value of k (number of bits): 5
Enter the initial binary number: 11001
Original Binary: 11001
Operation cost: 1, Total cost so far: 1
After Incrementing: 11010

Menu:
1. Increment Binary Counter
2. Exit
Enter your choice: 1
Enter the value of k (number of bits): 5
Enter the initial binary number: 11010
Original Binary: 11010
Operation cost: 2, Total cost so far: 3
After Incrementing: 11011
```

TECHNOLOGY STACK USED: C, C++, JAVA

Original Binary: 11001

After Incrementing: 11010

Original Binary: 11010

After Incrementing: 11011 Original

Binary: 11011

After Incrementing: 11100

AMORTIZED ANALYSIS:

Counter Value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18

Not all bits in the counter flip in each increment operation. The 0th bit flips in each increment and there are $\lfloor n \rfloor$ flips. The 1st bit gets flipped alternately and thus $\lfloor ^{n} \quad \rfloor$ flips in total. Thus, the ith bit gets

$\overline{}$
2

flipped ^{n} times in total.

$\lfloor \, i \rfloor$

2

he total number of flips in a sequence of n increments is thus $\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$.

The worst-case time for a sequence of n increment operations on an initially zero counter is therefore O(n). The average cost of each operation, and therefore the amortized cost per operation, is O(n)/n = O (1).

OBSERVATIONS:

The worst-case time for a sequence of n increment operations on an initially zero counter is consequently O(n) according to the Aggregate Method for Amortized Analysis (n). The average cost of each operation, and hence the amortized cost per operation, is O(n)/n = O (1).

CONCLUSION:

In this experiment, we understood how to implement a k-bit Binary Counter that counts upward and perform Amortized Analysis.

REFERENCES:

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, “Introduction to Algorithms”, 3rd Edition, The MIT Press, 2009.