Experiment 1: Error Detection using Parity

Aim: To apply Parity check rules for error detection

Objective: After carrying out this experiment, students will be able to:

- Apply 1D and 2D parity rules for error detection
- Analyze the difference between 1D and 2D parity and their limitations

Problem statement: You are required to write separate programs to demonstrate the use of 1D and 2D parity. Take the input bit streams from the user. Your programs should calculate the parity and display the input and output bit streams.

Analysis: While analyzing your program, you are required to address the following points:

- Why can this method not be used to correct errors?
- How are 1D and 2D parity different?
- What are the limitations of this method of error detection?

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	8	
Results	8	
Viva	4	
Total	**20**	

Submitted by:

Register No:

© Ramaiah University of Applied Sciences

1. Algorithm/Flowchart:-

1-D parity:-

Step 1:- Start
Step2:-define a function “countOnes” which counts number of ones in an array.
Step3:-input number of bits and the bits.
Step4:-input type of parity (even or odd) from user
Step5:- if parity is even:

```
        if (countOnes(arr,n)%2==0):  //no. of ones is even
                arr[n]=0            // parity bit is “0”
        else:
                arr[n]=1            // parity bit is 1
if parity is odd:
        if (countOnes(arr,n)%2==0):  //no. of ones is even
                arr[n]=1            // parity bit is “1”
        else:
                arr[n]=0            // parity bit is “0”
```

Step6: display parity bit
Step7:-display whole array of parity bits
Step8:-Stop

© Ramaiah University of Applied Sciences

2-D parity:-

Step 1:- Start
Step2:-define a function “countOnes” which counts number of ones in an array.
Step3:-input Data size and the bits using for loop.
Step4:-input type of parity (even or odd) from user
Step5: run for loop from i=0 to Data size:

5.1)Declare 2 arrays(temp,temp2) for rows and columns.
5.2)Run for loop from j=0 to Data size:

Temp[j]=arr[i][j] // collect data of i^{th} row
Temp2[j]=arr[j][i] // collect data of i^{th} column

5.3) if parity is even:

if (countOnes(temp,n)%2==0): //no. of ones is even in i^{th} row
arr[i][n]=0 // parity bit at row end is “0”
else:
arr[i][n]=1 // parity bit at row end is “1”

if (countOnes(temp2,n)%2==0): //no. of ones is even in i^{th}column
arr[i][n]=0 // parity bit at column end is “0”
else:
arr[i][n]=1 // parity bit at column end is “1”

similarly,
we can do if parity is odd.

Step6:-display whole array of parity bits
Step7:-Stop

© Ramaiah University of Applied Sciences

2. Program:-

1-d Parity:-

```c
#include <stdio.h>
#include <stdlib.h>
int countOnes(int arr[],int n){
    int count=0;
    for (int i = 0; i < n; i++)
    {
        if (arr[i]==1)count++;
    }
    return count;
}
int main() {
    int n,choice;
    printf("Enter Data Size: ");
    scanf("%d",&n);

    int arr[n+1][n+1];
    printf("Enter Bits:- \n");
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            scanf(" %d",&arr[i][j]);
        }
    };
    printf("\nChoose Parity:- \n\t\t1)Even Parity\n\t\t2)Odd Parity\n");

    printf("Enter choice: ");
    scanf("%d",&choice);
    // check number of ones accordingly
    int i,j;
    for (i = 0; i < n; i++)
    {
        int temp[n];// temp array for rows
        int temp2[n];// temp array for columns
        for (j = 0; j < n; j++)
        {
            temp[j]=arr[i][j];
            temp2[j]=arr[j][i];
        }
```

© Ramaiah University of Applied Sciences

```
        switch (choice)
        {
        case 1:arr[i][n]=(countOnes(temp,n)%2==0)?0:1;// horizontal check
               arr[n][i]=(countOnes(temp2,n)%2==0)?0:1;// vertical check
            break;
        case 2:arr[i][n]=(countOnes(temp,n)%2==0)?1:0;// horizontal check
               arr[n][i]=(countOnes(temp2,n)%2==0)?1:0;// vertical check
            break;
        default:printf("Enter valid choice \n");
            break;
        }
    }
    arr[n][n]=2;
    //display
    printf("\nParity bits are : \n");
     for (int i = 0; i < n+1; i++)
    {
        for (int j = 0; j < n+1; j++)
        {
            printf(" %d",arr[i][j]);
        }
        printf("\n");

    };
    return 0;
}
```

2-D Parity:-

```
#include <stdio.h>
#include <stdlib.h>
int countOnes(int arr[],int n){
    int count=0;
    for (int i = 0; i < n; i++)
    {
        if (arr[i]==1)count++;
    }
    return count;
}
```

© Ramaiah University of Applied Sciences

```c
int main() {
    int n,choice;
    printf("Enter Data Size: ");
    scanf("%d",&n);

    int arr[n+1][n+1];
    printf("Enter Bits:- \n");
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            scanf(" %d",&arr[i][j]);
        }
    };
    printf("\nChoose Parity:- \n\t\t1)Even Parity\n\t\t2)Odd Parity\n");

    printf("Enter choice: ");
    scanf("%d",&choice);
    // check number of ones accordingly
    int i,j;
    for (i = 0; i < n; i++)
    {
        int temp[n];// temp array for rows
        int temp2[n];// temp array for columns
        for (j = 0; j < n; j++)
        {
            temp[j]=arr[i][j];
            temp2[j]=arr[j][i];
        }
        switch (choice)
        {
        case 1:arr[i][n]=(countOnes(temp,n)%2==0)?0:1;// horizontal check
               arr[n][i]=(countOnes(temp2,n)%2==0)?0:1;// vertical check
            break;
        case 2:arr[i][n]=(countOnes(temp,n)%2==0)?1:0;// horizontal check
               arr[n][i]=(countOnes(temp2,n)%2==0)?1:0;// vertical check
            break;
        default:printf("Enter valid choice \n");
            break;
        }
    }
```

© Ramaiah University of Applied Sciences

```
    arr[n][n]=2;
    //display
    printf("\nParity bits are : \n");
     for (int i = 0; i < n+1; i++)
    {
        for (int j = 0; j < n+1; j++)
        {
            printf(" %d",arr[i][j]);
        }
        printf("\n");
    };
    return 0;
}
```

3. Results

1-D Parity:-

```
Enter number of bits: 5
Enter Bits:-
1 0 1 1 0

 Choose Parity:-
 1. Even Parity
 2. Odd Parity
Enter choice: 1
Parity is : 1
Parity bits are :
1 0 1 1 0 1
```

© Ramaiah University of Applied Sciences

2-D Parity:-

```
Enter Data Size: 3
Enter Bits:-
1 0 1
1 1 1
0 1 0

Choose Parity:-
               1)Even Parity
               2)Odd Parity
Enter choice: 1

Parity bits are :
 1 0 1 0
 1 1 1 1
 0 1 0 1
 0 0 0 2
PS F:\RUAS\sem 5\Computer networks\CN Lab\lab_1\Programs>
```

4. Analysis and Discussions:-

Why can this method not be used to correct errors?

If an odd number of bits (including the parity bit) are transmitted incorrectly, the parity bit will be incorrect, thus indicating that a parity error occurred in the transmission. The parity bit is only suitable for detecting errors; it cannot correct any errors, as there is no way to determine which particular bit is corrupted. The data must be discarded entirely, and re-transmitted from scratch

2d parity over 1d parity:-

Two Dimensional Parity can detect as well as correct one or more bit errors. If a one or more bit error takes place then the receiver will receive the message with the changed parity bit. It indicates that some error has taken place which means the error is detected.

5. Comments

   a. Limitations of the experiment

   There are two conditions which can cause a parity checking mechanism to fail to detect errors properly.

© Ramaiah University of Applied Sciences

- **Parity Bit Corruption** - Corruption of the parity bit itself during transmission can cause the receiving station to discard valid data.
- **Data Bit Corruption** - Corruption of more than one bit in the byte/word will leave the number of binary 1's in the byte the same, but change the data's actual value. This corrupts the data but prevents the receiving device from detecting the error because parity still matches
- **Combination** - Both the parity bit and a bit in the data is corrupted such that the data and parity bit match, but the data is corrupted.

6. Conclusions

All the above conditions occur frequently enough to make parity checking somewhat unreliable. Parity checks are often disabled and more effective methods are used to verify the integrity of the data such as cyclic redundancy checks.

© Ramaiah University of Applied Sciences