Bit Manipulation

Created	@January 10, 2023 1:53 PM
Class	striver
Type	Lecture
Materials	https://www.youtube.com/watch?v=5rtVTYAk9KQ&t=330s
Reviewed	☐

Bit Manipulation

Binary

```
0 & 1
```

Decimal to Binary

```
 //convert decimal 10 to binary

2 |10 | 0
2 | 5 | 1
2 | 2 | 0
  | 1
//so the in binary 10 is 1010

//convert decimal 14 to binary

2 | 14 | 0
2 | 07 | 1
2 | 03 | 1
  | 1

//so in binary 14 is 1110
```

NOTE: In binary, the bit indexing goes from right to left.

```
3 2 1 0 bits
1 0 1 1
--------
```

```
//so for binary to decimal
//starting from the rightmost bit
/**
  1 * 2 ^ 0 = 1
+ 1 * 2 ^ 1 = 2
+ 0 * 2 ^ 2 = 0
+ 1 * 2 ^ 3 = 8
-----------------
              11

therefore 1011 in decimal is 11.
*/
```

OPERATORS

```
‘&’ - and operator -> all 1 -> 1   a & b & c  a = 5, b =7, c = 8
                      any 0 -> 0   0101 & 0111 & 1000 = 0000
'|' - OR operator  -> any 1 -> 1   a = 101 b = 111
                      all 0 -> 0   101 | 111 = 111

'^' - xor operator -> even 1 -> 0  101 ^ 111 =  010
                      odd 1 -> 1   a = 5 b= 5 ,a ^ b = 0

'~' - negation operator -> 1 -> 0  FLIPS BIT
                           0 -> 1

'>>' - right shift operator -> last one bit will go off
                               a>>1 so for a = 5, 101 >> 1 is 10
                               the right shift means dividing by 2
                               so right shift by 3 means 2 ^ 3 div
                               the number by 8.
'<<' - left shift operator -> the first bit go off
                              5 << 2 means 0000....101 << 2
                                           0000 ...10100
                              left shift of two means (5 * 2^2)
```

Note: integer means 32 bits

Q. Given an array of integers arr[] = {2,1,2,5,6,5,7,7,7,6} every integer occurs twice except one number print the number.

```
#include<bits/stdc++.h>
using namespace std;

int Unique(int arr[], int n){
    int x = 0;
    for(int i = 0; i < n; i++){
        x  = x ^ arr[i];
    }
```

```
    return x;
}

int main(){
    int arr[] = {2,1,2,5,6,5,7,7,6};

    cout<< Unique(arr, 9)<<endl;

}
```

LOGIC: same numbers xor is 0.

Q. Swapping of two numbers using xor

```
/**a = 5 b = 7
Step 1 : a = a ^ b ie. a = 5 ^ 7 ,  b = 7
step 2: b = a ^ b ie.  b = 5 ^ 7 ^ 7 = 5, a = 5 ^ 7
step 3 : a = a ^ b ie. a = 5^7^5 = 7, b = 5*/
```

The code for the above mentioned logic.

```
#include<bits/stdc++.h>
using namespace std;

int main(){
  int a = 5;
  int b = 7;
  cout<< " before swap: a : " << a <<" " << "b: "<< b<<endl;
  a = (a ^ b);
  b = (a ^ b);
  a = (a ^ b)
  cout<< " after swap: a : " << a <<" " << "b: "<< b<<endl;
return 0;
}
```

Q. given N print the xor y of all numbers between (1-N)

```
//approach 1: O(n)

#include<bits/stdc++.h>
using namespace std;

int get_xor(int n){
    int x = 0;
    for(int i = 1; i <= n; i++){
        x = x ^ i;
    }

    return x;
}
```

```
int main(){
  int n;
  cin>>n;

  cout<<get_xor(n)<<endl;



return 0;
}


//approach 2: O(1)
logic: for all multiples of 4 xor value is n  n%4 == 0 ans : n
                                              n% 4 == 1 ans : 1
                                              n % 4 == 2 ans: n + 1
                                              n % 4 == 3 ans: 0

#include<bits/stdc++.h>
using namespace std;
int get_xor(int n){
    if(n % 4 == 0){
        return n;
    }else if(n % 4 == 1){
        return 1;
    }else if(n % 4 == 2){
        return n + 1;
    }else if(n % 4 == 3){
        return 0;
    }

    return -1;
}
int main(){
    int n;
    cin>>n;

    cout<<get_xor(n)<<endl;


return 0;
}
```

Q. Given a range (L-R) print the xor(L ^ L + 1 ^ L+ 2…..)

```
//approach 1: O(n) here, using a for loop to do xor of consecutive
//numbers from l to r and returning x

#include<bits/stdc++.h>
using namespace std;

int Xorer(int l, int r){
    int x = 0;
    for(int i = l; i <= r; i++){
```

```
        x = x ^ i;
    }
    return x;
}

int main(){
int l;
cin>>l;
int r;
cin>>r;

cout<<Xorer(l,r)<<endl;



return 0;
}

//approach 2: O(1)  xor before l that is till l -1
//so that what's left is the required xor result here we also take
//the efficient approach to xoring.
//logic: xor(r) ^ xor(l - 1);

#include<bits/stdc++.h>
using namespace std;

int get_xor(int n){
    if(n % 4 == 0){
        return n;
    }else if(n % 4 == 1){
        return 1;
    }else if(n % 4 == 2){
        return n + 1;
    }else if(n % 4 == 3){
        return 0;
    }

    return -1;
}

int main(){
int l;
cin>>l;
int r;
cin>>r;

int ans = get_xor(l - 1) ^ get_xor(r);

cout<<ans<<endl;

return 0;
}
```

& operator

Q. How to check whether a number is odd or even using bit manipulation?

```
/**logic: The last bit of an odd number is set and the last bit of
an even number is always 0 so if we perform the & operation between the
number and 1 if the answer is 0 then it's even else odd.*/

#include<bits/stdc++.h>
using namespace std;

 void oddEve(int n){
    if((n & 1) == 0){
        cout<<"even"<<endl;
    }else{
        cout<<"odd"<<endl;
    }
}

int main(){
int n;
cin>>n;

oddEve(n);


return 0;
}
```

Q. Check if the ith bit is set:

```
approach 1: first right shift n ,i places then do the and operation
then perform & operation with 1 if the answer is 1 then the last bit
is set.
#include<bits/stdc++.h>
using namespace std;

void checkSetBit(int n, int i){
    if(n > i){
        n = n>>i;
    }else{
        cout<<"invalid input"<<endl;
    }

    if(n & 1 == 1){
        cout<<"The bit is set"<<endl;
    }else{
        cout<<"The bit is not set"<<endl;
    }
}

int main(){
int n;
cin>>n;
int i;
cin>>i;
```

```
checkSetBit(n , i);


return 0;
}

approach 2: making a mask: (1 << i) then (mask & n != 0) bit is set

#include<bits/stdc++.h>
using namespace std;
void checkSetBit(int n, int i){
    if(n & (1 << i) == 1){
        cout<<"The bit is set"<<endl;
    }else{
        cout<<"The bit is not set"<<endl;
    }
}

int main(){
    int n;
    cin>>n;
    int i;
    cin>>i;
    checkSetBit(n , i);
    return 0;
}

/**The second approach is better and more correct because we are not
altering the input value which is given to us. We should always try to
never alter the given value*/
```

Using OR operator:

Q. set the ith bit of a number:

```
/**logic: to make the mask and perform the OR operation as I can
remember in the OR operation mantra any 1 is one and all 0 is 0 so
if the bit is already set then nothing will be done and if its 0
then it will change to one*/

#include<bits/stdc++.h>
using namespace std;

int setIthbit(int n, int i){
    int mask = 1 << i;
    int x;
    x = n | mask;

    return x;
}

int main(){
    int n;
    cin>>n;
    int i;
    cin>>i;
```

```
    int ans = setIthbit(n,i);
    cout<<ans<<endl;
    return 0;
}
```

Q. Clear the ith bit.

```
/**logic: Make the mask like the previous question and then negate it
now we perform the and operator and if the bit is set it will become 0
and if it's 0 it will remain 0.*/
#include<bits/stdc++.h>
using namespace std;

int clearIthbit(int n, int i){
    int mask = ~(1 << i);
    int x;
    x = n & mask;

    return x;
}

int main(){
int n;
cin>>n;
int i;
cin>>i;

int ans = clearIthbit(n,i);
cout<<ans<<endl;
return 0;
}
```

Q. Remove the last set bit.

```
/**logic: perform the and operation of n with n -1 number
  8 = 1000
&
  7 = 0111
------------
  0 = 0000
*/
#include<bits/stdc++.h>
using namespace std;

int clearLastSetBit(int n){
    int x = n & (n - 1);
    return x;
}
```

```
int main(){
    int n;
    cin>>n;
    cout<<clearLastSetBit(n);

return 0;
}
```

Q. How to check if a number is a power of 2.

```
/**logic: if 'AND' the result of the number and the preceding number
is 0 then the number is a power of two
8: 1000
&
7: 0111
---------
0: 0000*/
#include<bits/stdc++.h>
using namespace std;

void powerOfTwo(int n){
    if((n & (n - 1)) == 0){
        cout<<"power of 2"<<endl;

    }else{
        cout<<"not a power of two"<<endl;
    }
}

int main(){
  int n;
  cin>>n;

  powerOfTwo(n);

return 0;
}
```

Q. Count the number of set bits?

```
/** approach 1: 0(number of bits) in this approach we perform and
operation on the last bit of the number and if its = 1 then we
increment the count variable;and then we right shift the input value n
till it becomes 0.
*/
#include<bits/stdc++.h>
using namespace std;
```

```
int countSetBits(int n){
    while(n != 0){
        int count = 0;
        if(n & 1 == 1){
            count++;
        }
        n = n>>1;
    }
}

int main(){
    int n;
    cin>>n;

    cout<<countSetBits(n)<<endl;

return 0;
}

/** approach 2: O(number of set bits): here we keep clearing the
rightmost set bit and incrementing the count variable till n becomes 0.
*/

#include<bits/stdc++.h>
using namespace std;

int countSetBits(int n){
    int count = 0;
    while(n != 0){
        n = (n & (n -1));
        count++;
    }
    return count;
}

int main(){
    int n;
    cin>>n;

    cout<<countSetBits(n)<<endl;

return 0;
}
```