**Bitwise Hacks for Competitive Programming**

* **How to set a bit in the number ‘num’ :**

If we want to set a bit at nth position in number ‘num’, it can be done using ‘OR’ operator( | ).

#include<iostream>

using namespace std;

void set(int & num,int pos){

     num |= (1 << pos);

}

int main(){

     int num = 4, pos = 1;

     set(num, pos);

     cout << (int)(num) << endl;

     return 0;

}

* **How to unset/clear a bit at n’th position in the number ‘num’ :**

Suppose we want to unset a bit at nth position in number ‘num’ then we have to do this with the help of ‘AND’ (&) operator.

* First we left shift ‘1’ to n position via (1<<n) than we use bitwise NOT operator ‘~’ to unset this shifted ‘1’.
* Now after clearing this left shifted ‘1’ i.e making it to ‘0’ we will ‘AND'(&) with the number ‘num’ that will unset bit at nth position position.

#include <iostream>

using namespace std;

void unset(int &num,int pos){

num &= (~(1 << pos));

}

int main(){

int num = 7;

int  pos = 1;

unset(num, pos);

cout << num << endl;

return 0;

}

* **Toggling a bit at nth position :**

Toggling means to turn bit ‘on'(1) if it was ‘off'(0) and to turn ‘off'(0) if it was ‘on'(1) previously. We will be using ‘XOR’ operator here which is this ‘^’. The reason behind ‘XOR’ operator is because of its properties.

Properties of ‘XOR’ operator.

* + 1^1 = 0
  + 0^1 = 1

If two bits are different then ‘XOR’ operator returns a set bit(1) else it returns an unset bit(0).

#include <iostream>

using namespace std;

void toggle (int &num, int pos ){

    num ^= (1 << pos);

}

int main()

{

    int num = 4;

    int pos = 1;

    toggle(num, pos);

    cout << num << endl;

    return 0;

}

* **Checking if bit at nth position is set or unset:**

It is quite easily do able using ‘AND’ operator. Left shift ‘1’ to given position and then ‘AND'(‘&’).

#include <iostream>

using namespace std;

bool at\_position(int num,int pos){

bool bit = num & (1<<pos);

return bit;

}

int main(){

int num = 5;

int pos = 2;

bool bit = at\_position(num, pos);

cout << bit << endl;

return 0;

}

* **Inverting every bit of a number/1’s complement:**

If we want to invert every bit of a number i.e change bit ‘0’ to ‘1’ and bit ‘1’ to ‘0’. We can do this with the help of ‘~’ operator. For example : if number is num=00101100 (binary representation) so ‘~num’ will be ‘11010011’.

***This is also the ‘1s complement of number’****.*

#include <iostream>

using namespace std;

int main(){

    int num = 4;

    // Inverting every bit of number num

    cout << (~num); // Output : -5

    return 0;

}

* **Two’s complement of the number:**

2’s complement of a number is 1’s complement + 1.So formally we can have 2’s complement by finding 1s complement and adding 1 to the result i.e (~num+1) or what else we can do is using ‘-‘ operator.

#include <iostream>

using namespace std;

int main(){

    int num = 4;

    int twos\_complement = -num;

    cout << "Two's complement : " << twos\_complement << endl;

    cout << "Two's complement : " << (~num+1) << endl;

    return 0;

}

* **Stripping off the lowest set bit :**

we want to strip off the lowest set bit for example in Binary Indexed tree data structure, counting number of set bit in a number.

We do something like this:

**X = X & (X-1)**

But how does it even work ?

Let us see this by taking an example, let X = 1100.

(X-1)  inverts all the bits till it encounter lowest set ‘1’ and it also invert that lowest set ‘1’.

X-1 becomes 1011. After ‘ANDing’ X with X-1 we get lowest set bit stripped.

#include <iostream>

using namespace std;

void strip\_last\_set\_bit(int &num){

num = num & (num-1);

}

int main(){

int num = 14;

strip\_last\_set\_bit(num);

cout << num << endl;

return 0;

}

* **Getting lowest set bit of a number:**

This is done by using expression ‘X &(-X)’Let us see this by taking an example:Let X = 00101100. So ~X(1’s complement) will be ‘11010011’ and 2’s complement will be (~X+1 or -X) i.e  ‘11010100’.So if we ‘AND’ original number ‘X’ with its two’s complement which is ‘-X’, we get lowest set bit.

#include <iostream>

using namespace std;

int lowest\_set\_bit(int num){

    int ret = num & (-num);

    return ret;

}

int main(){

    int num = 10;

    int ans = lowest\_set\_bit(num);

    cout << ans << endl; // output : 2

    return 0;

}