***1)some basic page();***

***2)greey code***

***3)build in function***

***4) bitmask***

**Generate all sub set**

**Cunt all ons faster**

***1)some basic;***

// Sometimes, it is low-level optimization, so better time order

// Sometimes, it drop memory by factor of 8! so better memory order

// Sometimes, it make code shorter

/\*

A B !A A&B A|B A^B

0 0 1 0 0 0

0 1 1 0 1 1

1 0 0 0 1 1

1 1 0 1 1 0

X = 1152 = 10010000000

Y = 1428 = 10110010100

X & Y = 10010000000

X | Y = 10110010100

X ^ Y = 00100010100

-----------------------------------------------

A ^ B ^ C ^ D = E For any bit in E with 1, must be ODD number of 1s

X ^ 0 = X X ^ X = 0

(A ^ B ^ C ^ D ^ E) ^ (A ^ B ^ C) = D ^ E

------------------------------------------------------

Sift اسع بكتير من القسمة والضرب

X >> 1 = 1001000000 Equals X/2

X >> 2 = 100100000 Equals X/4

X << 1 = 100100000000 Equals X\*2

X << 2 = 1001000000000 Equals X\*4

X << 1 = 010010000000 Equals X\*2

-----------------------------------------------

اى رقم مكون من 2 اس حاجة

X = 11010 = 26 = 2^1 + 2^3 + 2^4 = 2 + 8 + 16

-------------------------------------------

X = 11010 = 26 // Last Bit shows Parity: Even / Odd

X = 11011 = 27

X % 2 = X & 1 = bit 0

شرح

X=8=1000;8&1=1000&0001=0000

Generally: X%(2^n) = X & (2^n -1)----------------🡪مش فاهم

Even test

if(n%2 == 1) -> Fails of n is negative

if( (n&1) == 1) -> Works always

------------------------------------------------------

void printNumber(int n, int len)

if(!len)return ;

printNumber(n>>1, len-1); // remove last bit

cout<<(n&1); // print last bit

}

// Let's count bits)mean 1)

int countNumBits1(int mask) { // O(bits Length)

int ret = 0; //1101001 -> 110100 -> 11010 -> 1101 -> 110 -> 11 -> 1 -> 0

while (mask) {

if(mask&1)

ret++;

mask >>= 1;

}

return ret;

}

---------------------------------------------------------------------

// Get bit

int getBit(int num, int idx) {

return ((num >> idx) & 1) == 1; // 110100, idx = 4 --> 110 & 1 = 0

}

int setBit1(int num, int idx) {

return num | (1<<idx);

}

int setBit0(int num, int idx) {

return num & ~(1<<idx); // 110100, idx = 3 --> 110100 & ~000100 = 110100 & 111011

}

int flipBit(int num, int idx) {

return num ^ (1<<idx);

}

// Exercise: rotate a bit to right. 0000000000000000000011001101110 -> rotate 5 -> 0111000000000000000000000110011

--------------------------------------------------------------------------------------

/\*

X-1 is very important!

X = 840 = 01101001000

X-1 = 839 = 01101000111 What happened? First bit) ==1 equal 16)16=(2^4) is removed, and 15=2^4-1 is added.

X & (X-1) = 01101000000 //first bit from right removed

X & ~(X-1) = 01101001000 & 10010111000 = 00000001000 value of 1<<SmaintestBitIdx

//to get value of last bit

\*/

***----------------------------------------------------------------------------------------***

int countNumBits2(int mask) { // O(bits Count) \_\_builtin\_popcount

int ret = 0;

while (mask) {

mask &= (mask-1);

++ret; // Simply remove the last bit and so on

}

return ret;

}

***-------------------------------------------------------------------------------------***

// len = 3: 000, 001, 010, 011, 100, 101, 110, 111

void printAllSubsets(int len) // Remember we did it recursively! This is much SIMPLER!

{

for (int i = 0; i < (1<<len); ++i)

printNumber(i, len);

// For reversed order. Either reverse each item or work from big to small

//for (int i = (1<<len); i >= 0 ; --i)

// printNumber(i, len);

}

----------------------------------------------------------------------------------------------------------------------------------

/\*

Gray Code:

is a binary numeral system where two successive(السابق (values differ in only one bit

0000 0000 0(number of 1s)

0001 0001 1

0010 0011 2

0011 0010 1

0100 0110 2

0101 0111 3

0110 0101 2

0111 0100 1

1000 1100 2

1001 1101 3

1010 1111 4

1011 1110 3

1100 1010 2

1101 1011 3

1110 1001 2

1111

Build it incrementally, Let's see for len = 1

0

1

Great. For len = 1 we have correct List. For length 2, we know we need to add prefix 0 and 1 so doubling list

00

01

10

11

Great, Len = 2 also works well.

For length 3:

000

001

010

011

100

101

110

111

بص الى فهمت انو بيبنى الجاى كود من من طول مثلا(3)الى (4)هو هيحط للنص الاول 0 والنص التانى 1 فى ال طول 3 وكدة هيكون الطول 4 وفى المشكلة الى بتظهر ان اخر حجة فى النص الاول واول حاجة النص التانى بيكون الفرق كبير

Failed. Note. We know 1st 4 numbers are correct. As the bits count did not change. Same for 2nd 4 numbers. All incremented 1.

Problem when moved from 4th) lenth)to 5th. The point, they will always be 0111111 and 100000.

What about reversing 2nd list, so we got the biggest 111111 one beside end of 1st block 011111.

So all what we need. Given answer of list N-1. To generate N. use 0N + 1N' where N' is reverse list.

If we built the list incrementally including from N=2 we got.

0000 0000 0

0001 0001 1

0010 0011 2

0011 0010 1

0100 0110 2

0101 0111 3

0110 0101 2

0111 0100 1

1000 1100 2

1001 1101 3

1010 1111 4

1011 1110 3

1100 1010 2

1101 1011 3

1110 1001 2

By observation, in gray code, every bit is Xor of its bit and next one.

\*/

int grayCode(int i) {

return i ^ (i>>1);

}

void printAllSubsetsGray(int len)

{

for (int i = 0; i < (1<<len)-1; ++i)

{

printNumber(i, len );

cout<<"\t\t"; //to print the normal number

printNumber( grayCode(i), len );

cout<<"\t"<<\_\_builtin\_popcount(grayCode(i));// Count of 1s in binary of nm

cout<<"\n"; //to print the gry number

}

}

--------------------------------------------------------------

// Be careful with operators precedence problems. http://en.wikipedia.org/wiki/Operators\_in\_C\_and\_C%2B%2B#Operator\_precedence

// X & 7 == 1 is interpreted as X & (7 == 1) --> (X & 7) == 1

// 1<<10 -1 is interpreted as = 1<<9 --> (1<<10) - 1

// Watch out from overflow

// 1 << 60 Fails. 1 is 32 bitInteger --> 1LL << 60

---------------------------------------------------------------------------------------

//مش فاهم من اول هنا

// What about print all subsets of a GIVEN MASK. Easily recursive, let's do it iterative

// E.g. subsets of 101: 101, 100, 001, 000 -> Each one is tried 0, 1

// Let's first assume, mask was complete and we work on it reverse

void PrintAllSubsetsAllOnes(int len)

{

int mask = (1<<len)-1; // mask = 1111 for len = 4;

for (int i = mask; i >= 0 ; i = i - 1)

printNumber(i, len);

}

// So we decrease -1 to get next. Do u remember effect of X-1?

/\*

11111

11110

11101

11100

11011

11010

11001

11000

10111

10110

10101

10100

10011

10010

10001

10000

01111

01110

01101

01100

01011

01010

01001

01000

00111

00110

00101

00100

00011

00010

00001

00000

\*/

***-------------------------------------------------------------------------------------------***

// What about trying the given mask - 1 till zero! Wrong 1s are added! Simply remove them

// 110101000 -1 = 110100111 & 110101000 = 110100000

void getAllSubMasks(int mask) {

for(int subMask = mask ; subMask ; subMask = (subMask - 1) & mask)

printNumber(subMask, 32); // this code doesn't print 0

// For reverse: ~subMask&mask = subMask^mask

}

***--------------------------------------------------------------------------------------------------------------------------***

// Efficiency

void intersection(int A[60], int len1, int B[60], int len2)

{

int mask1 = 0;

int mask2 = 0;

for (int i = 0; i < len1; ++i)

mask1 = setBit1(mask1, A[i]);

for (int i = 0; i < len2; ++i)

mask2 = setBit1(mask2, B[i]);

int inter = mask1 & mask2; // In O(1) noes intersection

for (int i = 0; i < max(len1, len2); ++i) {

if(getBit(inter, i))

cout<<i<<" ";

}

cout<<"\n";

}

// In small graphs, E.g. nodes < 64, we could maintain graph node neighbors in a mask

------------------------------------------------------------------------

// Better Memory

// Say you have numbers in range 100 Milion, and you want to know if some number visited before or NOT

// Simply: bool vis[100 \* 1000 \* 1000]; // and mark in it.

// But 100 Milion is too much! What about 100Milion / 8 = 12500000 12 Milion and half!

// You know that each character is 8 bits. Bool too takes 8 bits. UTILIZE byte!

const int MAX = 100000000;

char vis[MAX/8 + 1];

void setVisited(int i) { // in past vis[i] = 1

// Now each byte is responsible for 8 bits.

// So first we need to know which char is for us. Then my byte is in i / 8

// And then which bit with us. Any inside this byte i am is i % 8

vis[i/8] |= ( 1<<(i % 8) );

// Or more efficiently

vis[i>>3] |= ( 1<<(i&7) );

}

bool isVisited(int i) { // in past if(vis[i])return vis[i>>3] & (1<<(i&7));}

-------------------------------------------------------------------------

// Much staff? Let's use STL!

#include <bitset> // The power, speed and memory handling in one place

void STL() {

const int N = 20; // const

string s = "000111";

bitset<N> x(s); // 00000000000000000111

x.set(); // 11111111111111111111

x.flip(); // 00000000000000000000

x = 10; // 00000000000000001010

x |= 3; // 00000000000000001011

x = (x<<3); // 00000000000001011000

x = ~x; // 11111111111110100111

x.set(15, 0); // 11110111111110100111

x.set(15); // 11111111111110100111

x.flip(0); // 11111111111110100110

x.count(); // Returns the number of bits that are set.

x.any(); // Returns true if any bits are set.

x.none(); // Returns true if no bits are set.

x.test(15);

x.to\_ulong(); // Returns an unsigned long represent mask

// The most interesting

if(x[2] == 0)

;

x[0] = 1; // Set bit from most right to 1

x[N-1] = 0; // Set bit from most left to 0

cout<<x<<"\n"; // display a string of N bits

}

***----------------------------------------------------------------------------------------------------------------------***

--------------------------------------------------------------------------------------------------------------------------------------------

***-------------------------------------------------------------------------------------------------------------------------------------------------***

***Counting ones in binary representation***

int cnt = \_\_builtin\_popcount(num);

int cnt = \_\_builtin\_popcountll(num);

will not work with Visual studio

**Bit Masks**

Power Set:

given a set S = {x, y, z} , write all possible subsets from S.

0 → 0 0 0 → { }

1 → 0 0 1 → { z }

2 → 0 1 0 → { y }

3 → 0 1 1 → { y, z }

4 → 1 0 0 → { x }

5 → 1 0 1 → { x, z }

6 → 1 1 0 → { x, y }

7 → 1 1 1 → { x, y, z }

**Bit Masks**

**Code**:

int xyz[] = { ‘x’, ‘y’, ‘z’};

for(int i = 0; i < (1 << 3); i++){

for(int j = 0; j < 3; j++)

if((i >> j) & 1)

cout << xyz[j];

cout << endl;

}

-bitmask:-

--------

vector<int>v;

int n; cin >> n;

ll ans = 0;

forr(i, n)cin >> arr[i];

for (int i = 0; i < (1 << n); i++) {

int curstat = i;

for (int j = 0; j < n; j++) {

if (curstat & (1<<j)) {

b = arr[j];

}

}

}

-----------------------------------------------------------------------------------------------------------