suraj021's blog

BITMASKS — FOR BEGINNERS

By suraj021, 16 months ago,

I was having trouble understanding Bitmask, then I found an unknown pdf about Bitmask on google. I would like to help the beginners like me in understanding Bitmasks and their uses. Here we go :

MOTIVATION

Suppose you have a set of objects and you want some way to represent which objects to pick and which ones not to
pick. How do you represent that in in a program? More generally, how do you represent a subset of a set?One way is to use a Map to associate with each object a boolean value indicating whether the object is picked. Alternatively,if the object can be indexed by integers, you can use a boolean array. However, this takes up a lot of memory and can be slow due to the overhead of Map and array. If the size of the set is not too large, a bitmask is much more efficient (and convenient)!

WHAT IS BITMASKS ?

Bitmasks a.k.a. lightweight, small sets of Booleans (native support in C/C++/Java). An integer is stored in a computer's memory as a sequence/string of bits. Thus, we can use integers to represent a lightweight small set of Boolean values. All set operations then involve only the bitwise manipulation of the corresponding integer, which makes it a much more efficient choice when compared with the C++ STL vector, bitset, or set options. Such speed is important in competitive programming.

We know an integer is just a bunch of bits stringed together. The 1st bit will represent whether the 1st object is picked, the 2nd bit will represent whether the 2nd object is picked or not, etc. For example, suppose in a set of 5 objects, we have picked the 1st , 3rd , and 4th object. The bitmask to represent this in binary is 01101 or 13 in decimal (in the notes, the 1 st bit will always be the least significant bit and will always appear at the very right).

MANIPULATING BITMASKS

```
1. REPRESENTATION: A 32 (or 64)-bit signed integer for up to 32 (or 64) items. ( To avoid issues with the
   two's complement representation, use a 32-bit/64-bit signed integer to represent bitmasks of up to
   30/62 items only, respectively ).

   For example:                          5| 4  | 3 | 2 | 1 | 0   <- 0-based indexing from right
                                        32| 16 | 8 | 4 | 2 | 1   <- power of 2
                        A= 34 (base 10) =  1 | 0  | 0 | 0 | 1 | 0   <- in binary
                                        F | E  | D | C | B | A   <- alternative alphabet label
  In the example above,the integer A = 34 or 100010 in binary also represents a small set {1, 5} with a
  0-based indexing scheme in increasing digit significance ( or {B, F} using the alternative alphabet
  label )because the second and the sixth bits (counting from the right) of A are on ( 1 ).

 2. To multiply/divide an integer by 2:
                                    We only need to shift the bits in the integer left/right, respectively.
    Notice that the truncation in the shift right operation automatically rounds the division-by-2 down,
    e.g. 17/2  = 8.

    For example:          A = 34 (base 10)                  = 100010 (base 2)
                          A = A << 1 = A * 2 = 68 (base 10) = 1000100 (base 2)
                          A = A >> 2 = A / 4 = 17 (base 10) = 10001 (base 2)
                          A = A > 1 = A / 2 = 8 (base 10) = 1000 (base 2) <- LSB( Least Significant Bit )is
gone

 3. Add the jth object to the subset (set the jth bit from 0 to 1):
    use the bitwise OR operation A |= (1 << j).

     For example:      A = 34 (base 10) = 100010 (base 2)
                       j = 3, 1 << j    = 001000 <- bit '1' is shifted to the left 3 times
                                          -------- OR (true if either of the bits is true)
                       A = 42 (base 10) = 101010 (base 2) // update A to this new value 42

 4. Remove the jth object from the subset (set the jth bit from 1 to 0):
    use the bitwise AND operation A &= ~(1 << j).

     For example:      A = 42 (base 10) = 101010 (base 2)
                       j = 1, ~(1 << j) = 111101 <- '0' is the bitwise NOT operation
                                          --------- AND
                       A = 40 (base 10) = 101000 (base 2) // update A to this new value 40

 5. Check whether the jth object is in the subset (check whether jth bit is 1):
    use the bitwise AND operation T = A & (1 << j).
    If T = 0, then the j-th item of the set is off.
    If T != 0 (to be precise, T = (1 << j)), then the j-th item of the set is on.

     For example:    A = 42 (base 10) = 101010 (base 2)
                     j = 3, 1 << j    = 001000 <- bit '1' is shifted to the left 3 times
                                        -------- AND (only true if both bits are true)
                     T = 8 (base 10)  = 001000 (base 2) -> not zero, the 3rd item is on

 6. To toggle (flip the status of) the j-th item of the set:
    use the bitwise XOR operation A ^= (1 << j).

     For example:      A = 40 (base 10) = 101000 (base 2)
                       j = 2, (1 << j)  = 000100 <- bit '1' is shifted to the left 2 times
                                          -------- XOR <- true if both bits are different
                       A = 44 (base 10) = 101100 (base 2) // update A to this new value 44

 7. To get the value of the least significant bit that is on (first from the right):
    use T = (A & (-A)).

     For example:      A =  40 (base 10) = 000...000101000 (32 bits, base 2)
                      -A = -40 (base 10) = 111...111011000 (two's complement)
                                           ---------------- AND
                       T =   8 (base 10) = 000...000001000 (3rd bit from right is on)

 8. To turn on all bits in a set of size n: (be careful with overflows)
    use A = (1 << n) - 1 ;

 9. Iterate through all subsets of a set of size n:
          for ( x = 0; x < (1 << n), ++x )

 10. Iterate through all subsets of a subset y (not including empty set):
           for ( x = y; x > 0; x = ( y & (x-1) ) )
```

Example of a subset problem: given a set of numbers, we want to find the sum of all subsets.

Sol. This is easy to code using bitmasks. we can use an array to store all the results.

```
int sum_of_all_subset ( vector< int > s ){
          int n = s.size() ;
          int results[ ( 1 << n ) ] ;     // ( 1 << n )= 2^n

       //initialize results to 0
          memset( results, 0, sizeof( results ) ) ;

       // iterate through all subsets

          for( int i = 0 ; i < ( 1 << n ) ; ++ i ) {    // for each subset, O(2^n)
               for ( int j = 0; j < n ; ++ j ) {        // check membership, O(n)
                    if ( ( i & ( 1 << j ) ) != 0 )      // test if bit 'j' is turned on in subset 'i'?
                         results[i] += s [j] ;          // if yes, process 'j'
                    }
               }
          }

 11. LIMITATIONS:
     a. Always check the size of the set to determine whether to use an int or long long or not using bitmask
at all
     b. Always use parenthesis to indicate the precedence of operations when doing bitwise operations!
        When it involves bitwise operators and not putting parenthesis can yield undesirable results!

     For example, let x = 5. Then x - 1 << 2 = 16, but x - (1 << 2) = 1
```

P.S 1. I apologize for bad formatting. 2. If you find something wrong/inappropriate please correct me. 3. Examples are copied from some text book 4. Can anyone please write a blog on Backtracking , i don't get flow of control in recursive calls in backtracking when a certain constraint fails on a configuration (like in N queens problem), how does program backtracks and how control flow takes place then and what happens after that ?

Thank You :

bitmask, bit manipulation, beginners

-7 suraj021 16 months ago 17

Comments (17)

Write comment?

antivenom — 16 months ago, # |
Can there be anything better than exponential in subset sum ?
→ Reply

slicedclementines — 3 months ago, # ^ | ← Rev. 2 +5
There is a simple formula for the sum of all subsets. Consider a set $\{a_1, a_2, ..., a_n\}$. There are 2^n subsets, and each a_i exists in exactly 2^{n-1} of them. Therefore, the sum of all subsets is equal to $2^{n-1}(a_1 + a_2 + ... + a_n)$
→ Reply

antivenom — 16 months ago, # | 0
SEE this link—> beneficial to all
→ Reply

SmartCoder — 16 months ago, # | 0
Copy & paste from Competitive programming book :D
→ Reply

suraj021 — 16 months ago, # ^ | 0
I have already written that in P.S :D
→ Reply

VioletNash — 7 months ago, # ^ | 0
And which book might that be?
→ Reply

suraj021 — 6 months ago, # ^ | 0
Just google "Competitive Programming 3". Best of Luck
→ Reply

cup_of_tea — 4 months ago, # ^ | 0
You seems to know which book it comes from, why not replace in your post "some text book" by the title, and making it more visible? Stealing ideas like this is a ugly thing to do
→ Reply

JAYPEE — 4 months ago, # | 0
thanks ,such type of activity really helps beginners like me.
→ Reply

tweety — 4 months ago, # | +30
I think this type of post is illegal
→ Reply

suraj021 — 4 months ago, # ^ | 0
Look, you have an option of ignoring the post. Just do it. By the way JAYPEE found it useful. I helped a person. :P
→ Reply

charl407 — 4 months ago, # | 0
Hello, How to make sure that at all times, there are only k bits set to 1 in an n bit array?
→ Reply

anubhakurah — 3 months ago, # ^ | 0
@charl407 What operation do you want to do with that n bit array,it depends upon that,after what manipulations you want to have the number of set bits same?
→ Reply

invincibleirm — 4 months ago, # | 0
Good WORK DUDE
→ Reply

elmaystro — 5 weeks ago, # | 0
Can someone explain this line for me ? results[i] += s [j] ; // if yes, process j
I can't figure what it is doing i exactly and cant figure the relation between if jth bit is turned on in "i" with the array itself.
→ Reply

— Top rated

#	User	Rating
1	tourist	3580
2	Petr	3481
3	TooDifficult	3285
4	dotorya	3094
5	Um_nik	3047
6	PavelKunyavskiy	3007
7	matthew99	3001
8	jcvb	2999
9	vepifanov	2992
10	RomaWhite	2982

Countries | Cities | Organizations View all →

— Top contributors

#	User	Contrib.
1	gKseni	181
2	Enrichto	175
3	Petr	167
4	Swistakk	156
5	rng_58	156
6	Zlobober	156
7	Edvard	154
8	Xellos	146
9	csacademy	145
10	Um_nik	143

View all →

— Find user

Handle:
Find

— Recent actions

MikeMirzayanov → 2016-2017 CT SO3E05: Codeforces Trainings Season 3 Episode 5

Shafaet → HackerRank HourRank 13

pablosking → Need Help — problem: Olympic Games, PointTime Duality, Convexhull, Corner Cases, Tricky Implementation

fadi_yns1 → The Palindromic day

ajith → Slack team

Kareem_Mohamed95 → Problem D-Lakes in Berland

spiral_out → There is something that's been bothering me for a while now

AbdAllah_Rodo → Recursion tree solved examples to practice :)

sankat407 → how to solve equation ??

fcspartakm → Codeforces Round #297 (Div.2) Editorial

zscoder → Codeforces Round #372 Editorial

igdor99 → Codeforces Round #321 Editorial

vepifanov → Intel Code Challenge Elimination Round (Div.1 + div.2 combined)

math-is-stupid → Need help in SPOJ problem's.

Kuga_samsu → ****What category of problem is this ???????*****

Bogman → Codeforces Round #276 — Editorial

P1kachu → Codeforces Round #374 (Div. 2) Editorial

Hamada14 → Sum of the minimum of all subarrays

hogloid → About ICPC Bangkok Regional Contest(Any Info Appreciated)

Sonechko → Codeforces Round #371 Editorial

Um_nik → P.A.Q. (in PM)

Tobby_And_Friends → Segment Tree problem from CODECHEF

spontaneously → Google APAC 2017 round A problem B Rain

Am1en → SGU

saba_tavdgiridze → TIMUS 1960

Detailed →