**INTRODUCTION**

**BITMAPS**

Bitmaps are great to compute intersections and unions fast.

They are used to represent a list of distinct integers no larger than N using exactly N bits: if the integer i appears in the list, then i th bit is set to true. Bits for which there are no corresponding integers are set to false.

For example, the integers 3, 4, 7 can be represented as 00011001. As another example, the integers 1, 2, 7 can be represented as 01100001.

To compute the union between 3, 4, 7 and 1, 2, 7, all I need to do is compute the bitwise OR between 00011001 and 01100001 (=01111001) which a computer can do in one CPU cycle. Similarly, the intersection can be computed as the bitwise AND between 00011001 and 01100001 (=00000001).

MERITS:

It is easier & faster to find out the union, intersection and difference of two sets using bitwise operators and bitmaps.

DEMERITS:

There are some downsides to the bitmap approach: first we have to construct the bitmaps and then have to extract the set bits.

This is not much beneficial for larger numbers.

CODE:

A set is passed along with its size and its is converted to Bit Map

int BitMap(int set1[], int set\_size1)

{

int i;

int bitmap =0; //initializing to 0

for( i = 0; i <set\_size1; i++)

bitmap |= 1 << (set1[i]);

return bitmap;

}

E.g. : set A ={1,2,3} size = 3  
LOOPS  
Loop1 :

For (i = 0 to 3)   
bitmap = bitmap | 1<<1  
 = 0 | 10=10  
Loop 2:  
for(i=1 to 3 ),   
bitmap = bitmap | 1 << 2  
 = 10|100 = 110

Loop 3:

For(i= 2 to 3)

Bitmap = bitmap| 1<< 3  
 = 110 | 1000 = 1110

Hence the bitmap gives 1110  
The bit positions are represented as  
For e.g. with a set {1,2,3} the bits in bitmap pointing to 1s 2nd 3rd position are set to

7 6 5 4 3 2 1 0

0 0 0 0 1 1 1 0

Is the produced bitmap. Using this bitmap intersection, union, difference is performed.

CHANGE:   
This function is converting a bitmap into an integer set .  
First it is checked if the jth bit is set in the bitmap or not. If it is a set then corresponding element at the jth position in the set is present   
e.g. bitmap : 00001000  
now j = 3th bit is set   
corresponding element set[0] = 3

for j = 0 to j = LARGE

if((bitmap & (1 << j)) > 0)

set[i++] =j;

**SET THEORY**

**Set theory** is the branch of [mathematical logic](http://en.wikipedia.org/wiki/Mathematical_logic) that studies [sets](http://en.wikipedia.org/wiki/Set_(mathematics)), which are collections of objects. Although any type of object can be collected into a set, set theory is applied most often to objects that are relevant to mathematics. The language of set theory can be used in the definitions of nearly all [mathematical objects](http://en.wikipedia.org/wiki/Mathematical_objects).

**SET OPERATIONS**

1. [**Union**](http://en.wikipedia.org/wiki/Union_(set_theory)) of the sets *A* and *B*, denoted *A* ∪ *B*, is the set of all objects that are a member of *A*, or *B*, or both. The union of {1, 2, 3} and {2, 3, 4} is the set {1, 2, 3, 4}.

CODE:   
void UnionSet(int set1[], int set\_size1,int set2[], int set\_size2)

{

int temp,bitmap1=0,bitmap2=0;

bitmap1=BitMap(set1,set\_size1);

bitmap2=BitMap(set2,set\_size2);

temp=bitmap1 | bitmap2 ;

Change(temp);

}

Explanation : Here set1 set2 with their sizes are passed into the function eg {1,2,3} and {3,4}

Bitmap(set, size) function will generate bitmap for {1,2,3} = 00001110 and also for the set {3,4} = 00011000  
They are then ORed using bitwise 00001110 | 00011000 = 00011110 which will be converted to {1,2,3,4} using the CHANGE function.

**2.** [**Intersection**](http://en.wikipedia.org/wiki/Intersection_(set_theory)) of the sets *A* and *B*, denoted *A* ∩ *B*, is the set of all objects that are members of both *A* and *B*. The intersection of {1, 2, 3} and {2, 3, 4} is the set {2, 3}.

CODE:

void IntersectionSet(int set1[], int set\_size1,int set2[], int set\_size2)

{

int temp,bitmap1=0,bitmap2=0;

bitmap1=BitMap(set1,set\_size1);

bitmap2=BitMap(set2,set\_size2);

temp=bitmap1 & bitmap2 ;

Change(temp);

}

Here bitmap1 will contain set1 represented in the form of bits.

e.g.: for set {1,2} Bitmap will return bitmap1 {00000110} and for set2 {2,3}bitmap will return {00001100}.  
temp = 00000110 & 00001100 = 00000100  
this temp is then passed to Change function which converts the union set back to integer set { 2}.

**3.** [**Set difference**](http://en.wikipedia.org/wiki/Set_difference) of *U* and *A*, denoted *U* \ *A*, is the set of all members of *U* that are not members of *A*. The set difference {1,2,3} \ {2,3,4} is {1} , while, conversely, the set difference{2,3,4} \ {1,2,3} is {4} . When *A* is a subset of *U*, the set difference *U* \ *A* is also called the [complement](http://en.wikipedia.org/wiki/Complement_(set_theory)) of *A* in *U*. In this case, if the choice of *U* is clear from the context, the notation *Ac*is sometimes used instead of *U* \ *A*, particularly if *U* is a [universal set](http://en.wikipedia.org/wiki/Universal_set) as in the study of [Venn diagrams](http://en.wikipedia.org/wiki/Venn_diagram).

CODE:

void DifferenceSet(int set1[], int set\_size1,int set2[], int set\_size2)

{

int temp,bitmap1=0,bitmap2=0;

bitmap1=BitMap(set1,set\_size1);

bitmap2=BitMap(set2,set\_size2);

temp=bitmap1 ^~ bitmap2 ;

Change(temp);

}

Here bitmap1 will contain set1 represented in the form of bits.

e.g.: for set {1, 2, 3} Bitmap will return bitmap1 {00001110} and for set2 {2,3}bitmap will return {00001100}.  
negation of bitmap2 will be 11110011  
 temp = 00000110 & 11110011 = 00000010   
this temp is then passed to Change which converts the union set back to integer set { 1}.

**4.**[**Cartesian product**](http://en.wikipedia.org/wiki/Cartesian_product) of *A* and *B*, denoted *A* × *B*, is the set whose members are all possible [ordered pairs](http://en.wikipedia.org/wiki/Ordered_pair) (*a*,*b*) where *a* is a member of *A* and *b* is a member of *B*. The Cartesian product of{1, 2} and {red, white} is {(1, red), (1, white), (2, red), (2, white)}

CODE:.

void Cartesian Product(int set1[],int set\_size1,int set2[],int set\_size2)

{

/\*Function to Find Cartesian Product of two Sets\*/

int index1,index2;

for(index1=0;index1<set\_size1;index1++)

for(index2=0;index2<set\_size2;index2++

printf("{%d",set1[index1]);

printf(",");

printf("%d",set2[index2]);

}

**5.**[**Power set**](http://en.wikipedia.org/wiki/Power_set) of a set *A* is the set whose members are all possible subsets of *A*. For example, the power set of {1, 2} is { {}, {1}, {2}, {1,2} }   
For counter = 0 to power\_set\_size   
[this loop runs till 2^n ,that is the total number of subsets]  
For index = 0 to size\_of\_set  
  
logic :

counter & 1 << set[index]

eg: counter : 000 and set = {1,2,3}  
so counter goes from 000 to 111  
Check if jth bit in the counter is set .If set then print jth element from set  
j = 0  
000 & 1 << 0  
000 & 1 = 000  
j = 1   
000 & 1 << 1  
000 & 10 = 000  
j = 2  
000 & 1<< 2  
000 & 100  
000  
  
j > 3(greater than size of set )  
This loop gives {} set  
now for counter 001  
j = 0  
001 & 1 << 0  
001 & 1  
001 ---- which indicates jth bit is set in the counter   
thus set[0] is a subset ={1}  
j = 1  
001 & 1 << 1  
001 & 10  
000  
j = 2  
001 & 1 << 2  
001 & 100  
000

similarly for other subsets

**FUNCTIONS TABLE:**

|  |  |  |  |
| --- | --- | --- | --- |
| RETURN TYPE | FUNCTION | ARGUMENTS | DESCRIPTION |
| Void | InsertSet | int set1[],int \*set\_size1 | Function to Insert Elements Into Set |
| Int | BitMap | Int set1[],int set\_size1 | Function to Convert Integer Set into Bitmap |
| Void | UnionSet | int set1[], int set\_size1,int set2[], int set\_size2 | Function to Perform Union of Two Sets |
| Void | IntersectionSet | int set1[], int set\_size1,int set2[], int set\_size2 | Function to Perform Intersection of Two Sets |
| Void | Change | int bitmap | Function to Convert Bitmap into Set |
| Void | printPowerSet | int set1[], int set\_size1 | Function To Print PowerSet of a Given Set |
| Void | CartesianProduct | int set1[],int set\_size1,int set2[],int set\_size2 | Function to Find Cartesian Product of two Sets |
| Void | DisplaySet | int set1[], int set\_size1 | Function to Display the Set |
| Void | DeleteSet | int set1[],int \*set\_size1 | Function to Delete element from Set |