ADT's based on SET

date 2/22/2023

★ Comparison between Set & Linked List ★

• Elements →	• elements are Unique	• elements can be duplicates
• Order of elements →	• order is NOT SIGN-IFICANT (unless specified)	• order is IMPORTANT (e.g. in Array List)

★ ADT based on Set types

1.) ADT UID (union, intersection, difference)
2.) Dictionary
3.) Priority Queue

[ADT UID]

★ Set ADT operations

1.) set Union
2.) set Intersection
3.) set Difference
4.) MAKENULL
5.) INITIALIZE

★ Implementations

a.) Array Imp. - Static / Dynamic
b.) Linked List - singly or doubly-linked, sorted or unsorted
c.) Cursor-based
d.) Bit-Vector Implementations

Example:

	A	B
0	1	1
1	0	0
2	0	0
3	1	0
4	1	1
5	0	1
6	0	0
7	1	0
8	0	0
9	1	0

$U = \{0, 1, 2, \ldots 9\}$ → Universal set

Set $A = \{3, 7, 0, 4, 9\}$

Set $B = \{0, 4, 5\}$

★ NOTE: List range will be the entirety of the universal set, regardless if the elements are present or not.

★BIT-VECTOR IMPLEMENTATION

date 2/22/2023

★Bit-Vector Implementation: → uses one-dimensional array.
→ also known as Boolean array implementation (T, F)
because the contents of the array is either true or false or their equivalents in the given language.

• NOTE: → Size of set is dependent on the Universal set → all given sets are SUBSETS
→ Best used in:
> When the Universal set U is SMALL.
> Its elements are INTEGERS (or can be mapped to the set of integers)

• Example:

Set A = {3, 7, 0, 4, 9}

• Illustration:

	A
0	1
1	0
2	0
3	1
4	1
5	0
6	0
7	1
8	0
9	1

• Observations:
→ Notice that those in the set are 1 while the rest is 0
→ if set A is implemented using bit-vector, then
$A[x] = 1$ if $x \in A$
$A[x] = 0$ if $x \notin A$

• NOTE: ★ Place a 1 for every element that is = to index in set A.
★ Assign a UNIQUE integer number for each non-integer element

★ADVANTAGES★

1.) Operations member(), insert(), & delete() can be performed in constant time = O(1)
2.) Operations union(), intersection(), & difference can be performed in time proportional to the cardinality (no. of elements) of the universal set = O(N)
3.) If the universal set U is small & can fit in one computer word = USD operations can be performed in one logical operation

-BIT-VECTOR IMPLEMENTATION-

date 2/22/2023

★ D's advantages ★

- The amount of space needed for the set is proportional to the size of the universal set, hence if the universal set is relatively big then the size of each set will also be big. = space needed = size of U

- Illustrations:

★ Insert:

U = {0, 1, 2, ..., 9}

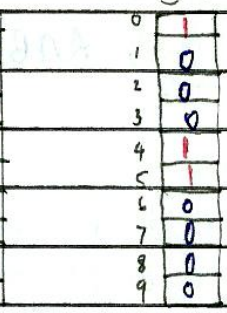
A = {1, 0, 5}

→ Insert(A, 7)

A = {1, 0, 5, 7}

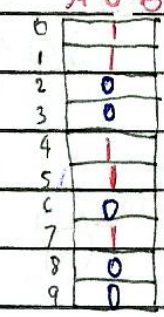
★ Union: U = {0, 1, 2, ..., 9}

A = {1, 0, 5, 7} B = {0, 4, 5}



∴ Union (A, B)

A ∪ B = {0, 1, 4, 5, 7}



- - Basic Set Operations - -

1.) UNION( ) - The union of two sets A & B is the set of all elements which belong to A or B or Both.

→ Notation: A ∪ B (read as "A union B")

Denoted by: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

2.) INTERSECTION( ) - The intersection of two sets A & B is the set of all elements are common to A and B

→ Notation: A ∩ B (read as "A intersection B")

Denoted by: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$

→ Example: A = {1, 3, 4, 6, 9}

B = {2, 3, 6, 8} A ∩ B = {3, 6}

3.) DIFFERENCE( ) - The difference of two sets A & B is the set of all elements which are in A, but not in B.

→ Notation: A - B (read as "A difference B")

Denoted by: $A - B = \{x \mid x \in A \text{ and } x \notin B\}$

→ Example: A = {1, 3, 4, 6, 9}

B = {2, 3, 6, 8}

A - B = {1, 4, 9}

Studocu

UID OPERATIONS VISUALIZATION

date 02/22/2023

★ Union - combine all elements

SET A = {7, 0, 4, 2} A∪B = {0, 1, 2, 4, 5, 7}

B = {0, 1, 5}

OR (∪)

7	6	5	4	3	2	1	0	
1	0	0	1	0	1	0	1	= A
0	0	1	0	0	0	1	1	= B
1	0	1	1	0	1	1	1	= A∪B

★ Intersection - find common elements

SET A = {7, 0, 4, 1} A∩B = {0, 1}

B = {0, 1, 5}

AND (&)

7	6	5	4	3	2	1	0	
1	0	0	1	0	1	1	1	= A
0	0	1	0	0	0	1	1	= B
0	0	0	0	0	0	1	1	= A∩B

★ Difference - element in A, NOT in B

SET A = {7, 0, 4, 1} A - B = {7, 4}

B = {0, 1, 5}

★ get complement of B first.

0 0 1 0 0 0 1 1 → 1 1 0 1 1 1 0 0 (~ COMPLEMENT)

★ Perform AND

7	6	5	4	3	2	1	0	
1	0	0	1	0	0	1	1	= A
1	1	0	1	1	1	0	0	= B
1	0	0	1	0	0	0	0	= A-B

★ Subset - if sets every element is in another set

SET A = {7, 0, 4, 1} B ≤ A

B = {0, 1, 4}

7	6	5	4	3	2	1	0	
1	0	0	1	0	0	1	1	= A
0	0	0	1	0	0	1	1	= B
0	0	0	1	0	0	1	1	!= 0

✓ B is a subset of A

SET OPERATIONS VISUALS

date 2/22/2023

★ member() - returns 1/non-zero if element is a member
0 if not member

SET A = { 2, 4, 5 }

X = 5

*Note: 5 is a mask value that was shifted to that position from 1

7 6 5 4 3 2 1 0

0 0 1 1 0 1 0 0 = A
0 0 1 0 0 0 0 0 = 5
0 0 1 0 0 0 0 0 != 0 ✓
is member

0 0 0 1 0 1 0 0 = A
0 0 1 0 0 0 0 0 = 5
0 0 0 0 0 0 0 0 == 0 ×
is not member

★ insert() - insert element at appropriate position matching bit position.

SET A = {7, 0, 4, 1}

X = 5

OR

7 6 5 4 3 2 1 0

1 0 0 1 0 0 1 1 = A
0 0 1 0 0 0 0 0 = 5
1 0 1 1 0 0 1 1 = {7, 0, 4, 1, 5}

★ delete() - delete element at appropriate position matching bit position

SET A = {7, 0, 4, 1, 5}

X = 5

* get complement of x

~ 0010 0000 = 1101 1111

★ perform AND

1011 0011 = A
1101 1111 = 5
1001 0011 = {7, 0, 4, 1}

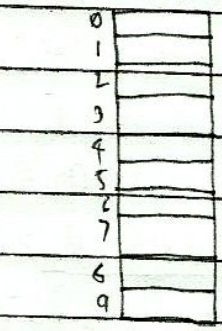
Exercise: Union Set

date 2/22/2023

1.) Write an appropriate definition of datatype SET such that A is an array of integers of size 10. NOTE: SET A is a static array

• Illustration

SET A



• code:

```
#define SIZE 10
typedef int SET [SIZE];
```

SET A; → =A[10]

2.) Using your definition of SET in #1, do the ff.:

a.) function header of function Union() - the func. will return a new set containing element found in the given 2 sets A & B.

• Code: SET * setUnion (SET A, SET B)

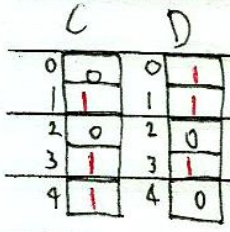
↳ pointer to locally declared array

b.) function call, declare & initialize the call:

• Assumption: U = {0, 1, 2, 3, 4} D = {0, 1, 3} //SIZE shortened to 5 for convenience

C = {1, 3, 4} CUD = {0, 1, 3, 4}

• Illustration:



C={1,3,4} D={0,1,3}

• Declaration & Initialization:

SET C = {0, 1, 0, 1, 1}

SET D = {1, 1, 0, 1, 0}

} NOTE: keep declaration & initialization TOGETHER

SET *retval = setUnion (C, D);

NOTES:

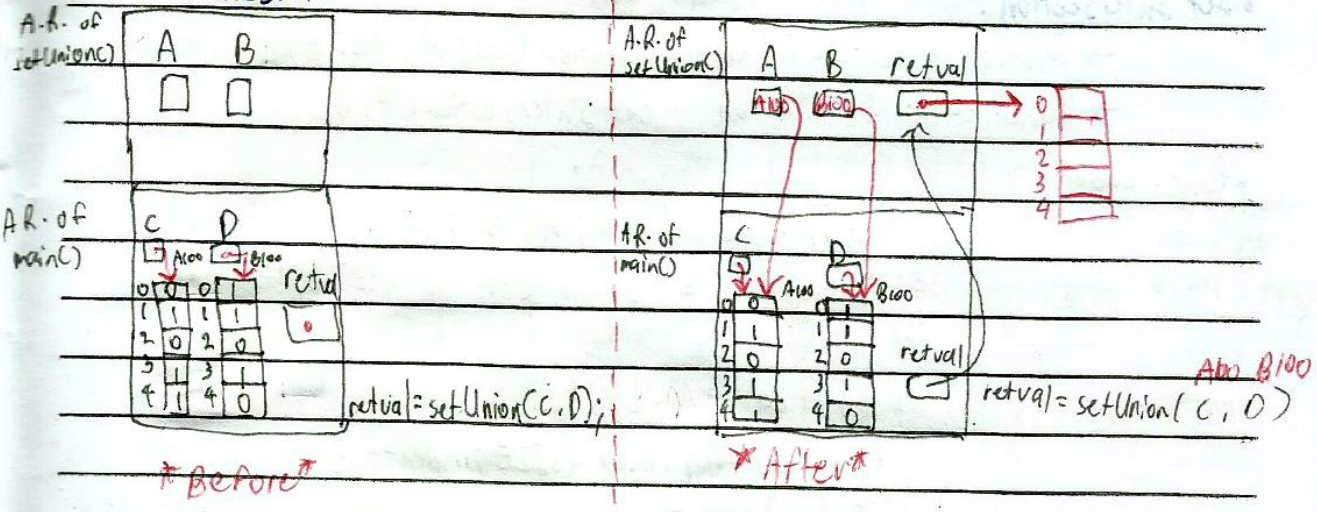
- make a catcher variable since you're returning a new set.
- Its a pointer since it will point to the locally created dynamically allocated array in the function.

Strandmore

Exercise: Union set

date 2/22/2023

★ Execution stack:



★ The name of an array is a POINTER to five components.

★ A & B will get the address of C[0] & D[0] respectively.

★ Array must be dynamically allocated when returned.

★ Code:

```
SET *setUnion(SET A, SET B){
    SET *retval = (SET *) calloc( SIZE, sizeof(SET));
    int index;                    // initializes array to be all 0

    if (retval != NULL){
        for (index = 0; index < SIZE; index++){
            (*retval)[index] = (A[index] == 1 || B[index] == 1)? 1:0;
        }
    }
    return retval;
}
```

OR

can also be:

```
(*retval)[index] = A[index] || B[index];
```

Strathmore

Practice Stuff

date _____

- Set Intersection:
  - → function header: SET *setIntersection(SET A, SET B)
  - → func. call: SET *retval = setIntersection(A, B);

- func. code:

```
SET *setIntersection(SET A, SET B){
  SET *retval = (SET*)malloc(sizeof(SET));
  int index;
  if (retval != NULL){
    for(index=0; index<SIZE; index++){
    (*retval)[index] = (A[index]==1 && B[index]==1)? 1:0;
  }
  }
  return retval;
}
```

- Set Difference:
  - → func. header: SET *setDifference(SETA, SETB)
  - → func. call: SET *retval = setDifference(A, B);

- func. code:

```
SET *setDifference(SET A, SETB){
  SET *retval = (SET*)malloc(sizeof(SET));
  int index;
  if (retval != NULL){
    for(index=0; index<SIZE; index++){
    (*retval)[index] = (A[index] && !B[index])? 1:0;
  }
  }
  return retval;
}
```

SET OPERATIONS: UNION (LL imple)

date 2/22/2023

→ uses size of set represented, not size of Universal set

→ better if list is SORTED

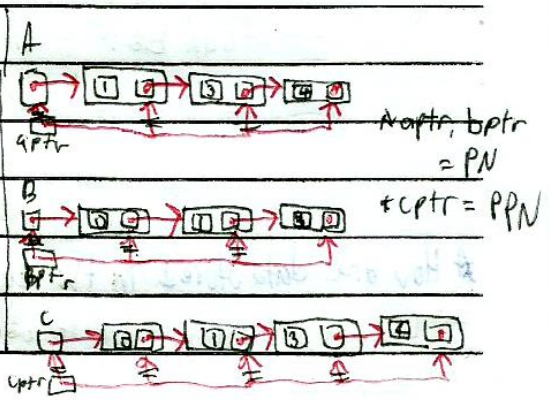
• Exercise 2:

• Def:

```
typedef struct node {
    int data;
    struct node *link;
} nodetype, *SET
```

• func call:

```
SET A;
SET B;
SET C;
C = setUnion(A,B);
```



• code:

```
SET setUnion(SET A, SETB){
    SET C, *cptr, aptr, bptr;
    C=NULL; cptr = &C;
    aptr=A; bptr=B;

    While(aptr != NULL && bptr != NULL){
        *cptr = (SET) malloc (sizeof (struct node));
        if (aptr->data < bptr->elem) {
            (*cptr)->data = aptr->elem;
            aptr = aptr->next;
        }else{
            if(aptr->elem == bptr->elem){
                aptr = aptr->next;
            }
            (*cptr)->elem = bptr->elem;
            bptr = bptr->next;
        }
        cptr = &(*cptr)->next
    }
    if (bptr != NULL){
        aptr = bptr;
    }
    While(aptr != NULL){
        *cptr = (SET) malloc(sizeof(nodetype));
        (*cptr)->data = aptr->data;
        aptr = aptr->link;
        cptr = &(*cptr)->link;
    }
    *cptr = NULL;
}
*cptr = NULL;
return C;
}
```

(1) Declare - aptr, bptr (PN) for traversing
- cptr (PPN) for inserting
- C (new set)

(2) Loop checking until either ptr becomes NULL

(3) check if a < b, if a is smaller, insert first to C.
- update *cptr & aptr.

COMPUTER WORD IMPLEMENTATION

date 2/22/2023

★ Computer Word - a unit of data of a defined bit length.
- Can be moved & addressed
- Can be:
  - Char - 8 bits
  - Short int - 16 bits
  - int - 32 bits

★ operations:
1.) UID
2.) member()
3.) insert()
4.) delete()

★ How are data stored in the Computer Memory?

→ Characters & integers are stored in the computer's memory as a sequence of binary digits (bits).

•Example:

a.) char ch = 'A'

Note: A → 65 (ASCII Value)

8 bits → ch = 0 1 0 0 0 0 0 1 = 65
since its a character 128 64 32 16 8 4 2 1

b.) char x = -25; Note: do 2's complement if given is negative

= 0001 1001 → leave first 0 & 1 unchanged (none so skip)
= 1110 0110 → reverse bits
= 1110 0111 → plus 1
-25 = 1110 0111 (in computer's memory

Note: • bit patterns of positive integers start w/ 0;
• of negative integers start w/ 1
• Range value of char is -128 to 127

Recall: Bit-Vector Implementation

date 2/22/2023

$U = \{0, 1, 2, 3, 4, 5, 6, 7\}$ SET $A = \{7, 0, 4, 2\}$

	A
0	1
1	0
2	1
3	0
4	1
5	0
6	0
7	1

Recall: If set A is implemented using Bit-Vector, then

$A[x] = 1$ if $x \in A$

$A[x] = 0$ if $x \notin A$

Facts:
- Contents are bits (0 or 1)
- Since C does not have the "bit" datatype, char or int can be used instead.
- Elements of Universal set are indices of the array
- Can FIT INSIDE char since char = 8 bits.

* Bit-Vector to Computer Word:

> using same given:

7	6	5	4	3	2	1	0	
1	0	0	1	0	1	0	1	A (BVI) ⇒ Elements are index

↓

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	0	0	1	0	1	0	1	A (Computer Word) ⇒ Elements are Exponents

Set A is implemented using computer word:

- if $x \in A$, then the bit position w/ place value 2^x is 1.
  ex. $4 \in A$, bit position: 2^4 is 1

- if $x \notin A$, bit position w/ place value 2^x is 0.
  ex. $5 \notin A$, bit position: 2^5 is 0.

- Computer Word Operations (Visualization)

date 2/22/2023

- typedef unsigned char SET;

no negatives, ranged value: 0 to 255

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

1 1 1 1 1 1 1 1 = 255 | SET A = {0, 1, 2, 3, 4, 5, 6, 7}

★ How to Extract a Bit

→ can be done using bitwise operators (& ^ | ~)
& shifting operators (<< >>)

//member() - VISUALIZATION

- must EXTRACT to retrieve element (mask value)
- checks if element (exponent) is a MEMBER of set (bits)

Example: A = { 5, 4, 2 }
7 6 5 4 3 2 1 0
= 0 0 1 1 0 1 0 0

bitwise AND

0011 0100 & 0000 0100 → mask value

0000 0100 != 0 → Member

0011 0000 & 0000 0100 → Mask value

0000 0000 == 0 → Not member

* How shifting works: mask value: 0000 0001 << 1

★ Move/SHIFT the mask value & use it to check against the elements in set A to check if elements are present.

☆ Mask value should always start at 1 (00000001) & shift LEFT.
func - will check against each bit per iteration.

★ If resulting bit pattern != 0 (means it found an element in that bit position)
== 0 (means element does not exist in that bit position).

★Ex: 0000 0001 (<<2)
0000 0100

Strathmore

A support page

date 2/22/2023

• Logical Right shift & Arithmetic Right shift.

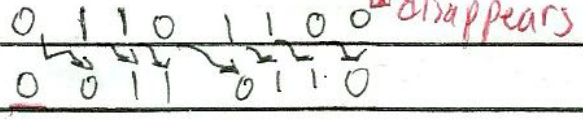
• Logical Right Shift - done as >>> in Java
- best done w/ unsigned data types

• Comparison between Left & Right Shift.

$N << 3 = N * 2^3$

$N >> 3 = N / 2^3$

• Some Examples:



• 25 right shift 2

0 0 0 1 1 0 0 1 = 25

0 0 0 0 0 1 1 0 = 25 >> 2

• -25 right shift 2

1 1 1 0 0 1 1 1 = -25

1 1 1 1 1 0 0 1 = -25 >> 2

• Arithmetic Right Shift - Automatic 0

Exercise: Finding the Bit-Pattern

date 2/22/2023

- Write the code of the function that will accept a short integer as parameter. The function will display the bit pattern of this integer.
  a.) use shift operators (>> <<) and
  b.) use bitwise operators (& ^ | ~)

NOTE: It should be platform independent (use sizeof()), do not assume any size. Use sizeof to compute the size of short int - the number of bytes, then you can compute the no. of bits

- Code:

```
typedef short int SET;
void displayBitPattern(SET A){
  int index;
  int size = sizeof(short int)*8;
  for(index=1; index<size; index++){
    if((A&1<<index)>0){
      printf("%d", index);
    }
  }
}
```

bitwise AND

// to print what numbers are IN the set.

or

```
if((A&1<<index)>0){
  printf("%d", 1);
}else{
  printf("%d", 0);
}
```

// to print 0000 1111 pattern of the set

- Simulation:

1. Set x = 1 → MAsk Value, func. will shift from here
2. Calculate size of datatype. — must be platform independent
3. Check bit pattern of A against 1 << x. (shift 1 place, 2 places)
4. Print

Staedtmore

- Comp-Word OPERATIONS CODE -

date _____

* UID

- Definition: typedef unsigned char SET;

- Union(A, B) - returns A ∪ B

```
SET setUnion(SET A, SET B){
    return (A | B);
}
```

- Intersection(A, B) - returns A ∩ B

```
SET setIntersaction(SET A, SET B){
    return (A & B);
}
```

- Difference(A, B) - returns A - B

```
SET setDifference(SET A, SET B){
    return (A & ~B);
}
```

- Subset(A, B) - returns 1/nonzero if B ⊆ A; 0 otherwise

```
int isSubset(SET A, SET B){
    return (A & B) == B;
}
```

* Set functions

- initialize(*A)

```
Void initialize(SET *A){
    *A = 0;
}
```

- member(A, x)

```
int isMember(SET A, int elem){
    return (A & 1 << elem) > 0;
}
```

- insert(*A, x)

```
SET insert(SET *A, int elem){
    *A = *A | 1 << elem;
}
```

- delete(*A, x)

```
SET delete(SET *A, int elem){
    *A = *A & ~(1 << elem);
}
```