Experiment 21 **Date: 17/12/2024**

Implement Set Data Structure using Bit Strings

Aim:

Create the Abstract Data Type (ADT) using Set and perform the operations Union, Intersection and Difference operations. Implement using Bit Strings.

Algorithm:

Declare the sets

1 Declare a[11], b[11], res[11]
2 Declare and initialize universal set U[11] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

main()

1 Start
2 Declare n, choice.
3 Read the limit of set A
4 Call input(a,n)
5 Read the limit of set B
6 Call input(b,n)
7 Display choices.
8 Read option choice.
    If ch==1 call set_union()
    If ch==2 call set_intersection()
    If ch==3 call set_difference()
    If ch==4 call set_equality())
      If (set_equality())
        Display “Sets are equal”
      Else
        Display “Sets are not equal”
    If ch==5 exit from the program
9 Repeat steps 7 and 8 while(1)
10 Stop

void display(int bs[])

1 Start.
2 For i=1 to 11
    Display bs[i]
3 Exit.

void input(int bs[], int n)

1 Start.
2 Declare x
3 Read the elements
  For i=0 to n
   If (x >= 1 && x <= 10)
     Set bs[x]=1
   Else

Display “Invalid element. Please enter a number between 1 and 10.”
Set i=i-1
4 Exit.

void set_union()
1 Start.
2 For i=1 to 11
Set res[i] = a[i] | b[i]
3 Display Union Set
4 Exit.

void set_intersection()
1 Start.
2 For i=1 to 11
Set res[i] = a[i] & b[i]
3 Display Intersection Set
4 Exit.

void set_difference()
1 Start.
2 For i=1 to 11
Set res[i] = a[i] & ~b[i]
3 Display Difference Set
4 Exit.

bool set_equality()
1 Start.
2 For i=1 to 11
If (a[i] != b[i])
Return false
3 Return true
4 Exit.

Program

```
#include <stdio.h>
#include <stdbool.h>
#include<stdlib.h>
int a[11] = {0}; // Set A
int b[11] = {0}; // Set B
int res[11] = {0}; // Result Set
int U[11] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; // Universal Set
void display(int bs[]) {
   for (int i = 1; i < 11; i++) {
      printf("%d\t", bs[i]);
   }
   printf("\n");
}
void input(int bs[], int n) {
```

```
    int x;
    printf("Enter the elements: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &x);
        if (x >= 1 && x <= 10) {
            bs[x] = 1;
        } else {
            printf("Invalid element. Please enter a number between 1 and 10.\n");
            i--;
        }}}
void set_union() {
    for (int i = 1; i < 11; i++) {
        res[i] = a[i] | b[i];
    }
    printf("\nUnion Set: ");
    display(res);
}
void set_intersection() {
    for (int i = 1; i < 11; i++) {
        res[i] = a[i] & b[i];
    }
    printf("\nIntersection Set: ");
    display(res);
}
void set_difference() {
    for (int i = 1; i < 11; i++) {
        res[i] = a[i] & ~b[i];
    }
    printf("\nDifference Set: ");
    display(res);
}
bool set_equality() {
    for (int i = 1; i < 11; i++) {
        if (a[i] != b[i]) {
            return false;
        }}
    return true;
}
void main() {
    int n;
    printf("Enter the number of elements in set A: ");
    scanf("%d", &n);
    input(a, n);
    printf("Enter the number of elements in set B: ");
    scanf("%d", &n);
    input(b, n);
    int choice;
    while(1) {
        printf("\nMENU");
```

```
        printf("\n1. Union\n2. Intersection\n3. Difference\n4. Check Equality\n5.
Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
           case 1:
              set_union();
              break;
           case 2:
              set_intersection();
              break;
           case 3:
              set_difference();
              break;
           case 4:
              if (set_equality()) {
                 printf("\nSets A and B are equal.\n");
              } else {
                 printf("\nSets A and B are not equal.\n");
              }
              break;
           case 5:
              exit(0);
           default:
              printf("\nInvalid choice! Please enter a number between 1 and 5.\n");
        }} }
```

Output

```
Enter the number of elements in set A: 4
Enter the elements: 1
2
3
4
Enter the number of elements in set B: 4
Enter the elements: 3
4
5
6
MENU
1. Union
2. Intersection
3. Difference
4. Check Equality
5. Exit
Enter your choice: 1
Union Set: 1    1       1       1       1       1       0       0       0       0

MENU
```

1. Union
2. Intersection
3. Difference
4. Check Equality
5. Exit
Enter your choice: 2
Intersection Set: 0 0 1 1 0 0 0 0 0 0

MENU
1. Union
2. Intersection
3. Difference
4. Check Equality
5. Exit
Enter your choice: 3
Difference Set: 1 1 0 0 0 0 0 0 0 0

MENU
1. Union
2. Intersection
3. Difference
4. Check Equality
5. Exit
Enter your choice: 4
Sets A and B are not equal.

MENU
1. Union
2. Intersection
3. Difference
4. Check Equality
5. Exit
Enter your choice: 5

Experiment 22 **Date: 30/12/2024**

Disjoint Set Data Structures

Aim:

Implement the Disjoint set ADT with Create, Union and Find operations.

Algorithm:

initSets()

1. declare ;
2. i = 0,i<numElements
3. sets[i].parent = i
4. sets[i].rank = 0

find(int)

1. if (sets[element].parent != element)
2. sets[element].parent = find(sets[element].parent)
3. return sets[element].parent

unionSets(int, int)

1. int set1 = find(element1);
2. int set2 = find(element2);
3. if (set1 != set2)
4. if (sets[set1].rank> sets[set2].rank)
5. sets[set2].parent = set1
6. else if (sets[set1].rank< sets[set2].rank)
7. sets[set1].parent = set2
8. else
9. sets[set2].parent = set1;
10. sets[set1].rank++;

displaySets()

1. declare i;
2. i = 0, i<numElements
3. print i
4. for (i = 0; i<numElements; i++) {
5. print sets[i].parent
6. i = 0, i<numElements
7. print sets[i].rank

main()

1. declare i
2. numElements = 6
3. unionSets(0, 1);
4. unionSets(1, 2);
5. unionSets(3, 4);
6. unionSets(4, 5);
7. unionSets(2, 4);

8. set i = 0,
9. if i<numElements
   Print find(i)

Program

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_ELEMENTS 1000
typedef struct Set {
   int parent;
   int rank;
} Set;
Set sets[MAX_ELEMENTS];
int numElements;
void initSets() {
   int i;
   for (i = 0; i<numElements; i++) {
      sets[i].parent = i;
      sets[i].rank = 0;
   }}
int find(int element) {
   if (sets[element].parent != element) {
      sets[element].parent = find(sets[element].parent); // Path compression
   }
   return sets[element].parent;
}
void unionSets(int element1, int element2) {
   int set1 = find(element1);
   int set2 = find(element2);
   if (set1 != set2) {
      if (sets[set1].rank> sets[set2].rank) {
         sets[set2].parent = set1;
      } else if (sets[set1].rank< sets[set2].rank) {
         sets[set1].parent = set2;
      } else {
         sets[set2].parent = set1;
         sets[set1].rank++;
      }}}
void displaySets() {
   int i;
printf("Element:\t");
   for (i = 0; i<numElements; i++) {
printf("%d\t", i);}
printf("\nParent:\t");
   for (i = 0; i<numElements; i++) {
printf("%d\t", sets[i].parent);
   }
printf("\nRank:\t");
```

```
    for (i = 0; i<numElements; i++) {
printf("%d\t", sets[i].rank);
    }
printf("\n\n");
}
int main() {
    int i;
numElements = 6;
initSets();
displaySets();
unionSets(0, 1);
unionSets(1, 2);
unionSets(3, 4);
unionSets(4, 5);
unionSets(2, 4);
displaySets();
    for (i = 0; i<numElements; i++) {
printf("The representative element of element %d is %d\n", i, find(i));
    }
    return 0;
}
```

Output

```
Element:    0    1    2    3    4    5
Parent:     0    1    2    3    4    5
Rank:       0    0    0    0    0    0
Element:    0    1    2    3    4    5
Parent:     0    0    0    0    3    3
Rank:       2    0    0    1    0    0
The representative element of element 0 is 0
The representative element of element 1 is 0
The representative element of element 2 is 0
The representative element of element 3 is 0
The representative element of element 4 is 0
The representative element of element 5 is 0
```

Experiment 23 **Date: 30/12/2024**

Heap Data Structure

Aim:

Create a Max-Heap and Min-Heap from the array.

Algorithm:

main()

1 Start
2 Declare n, arr[n].
3 Read the limit of array
4 Read the elements of the array
5 Call buildMaxHeap(arr, n)
6 Call printArray(arr, n)
7 Call buildMinHeap(arr, n)
8 Call printArray(arr, n)
9 Stop

void maxHeapify(int arr[], int n, int i)

1 Start.
2 Set largest=i
3 Set left = 2 * i + 1
4 Set right = 2 * i + 2
5 If (left < n && arr[left] > arr[largest])
  Set largest = left
6 If (right < n && arr[right] > arr[largest])
  Set largest = right
7 If (largest != i)
  Set temp = arr[i]
  Set arr[i] = arr[largest]
  Set arr[largest] = temp
  Call maxHeapify(arr, n, largest)
8 Exit.

void buildMaxHeap(int arr[], int n)

1 Start.
2 For i=n/2-1 to 0
  Call maxHeapify(arr, n, i)
3 Exit.

void minHeapify(int arr[], int n, int i)

1 Start.
2 Set smallest=i
3 Set left = 2 * i + 1
4 Set right = 2 * i + 2
5 If (left < n && arr[left] < arr[smallest])
  Set smallest= left

6 If (right < n && arr[right] < arr[smallest])
Set smallest= right
7 If (smallest!= i)
Set temp = arr[i]
Set arr[i] = arr[smallest]
Set arr[smallest] = temp
Call minHeapify(arr, n, smallest)
8 Exit.

void buildMinHeap(int arr[], int n)

1 Start.
2 For i=n/2-1 to 0
Call minHeapify(arr, n, i)
3 Exit.

void printArray(int arr[], int n)

1 Start.
2 For i=0 to n
Display arr[i]
3 Exit.

Program

```
#include <stdio.h>
#include <stdlib.h>
void maxHeapify(int arr[], int n, int i) {
   int largest = i;
   int left = 2 * i + 1;
   int right = 2 * i + 2;
   if (left < n && arr[left] > arr[largest])
      largest = left;
   if (right < n && arr[right] > arr[largest])
      largest = right;
   if (largest != i) {
      int temp = arr[i];
      arr[i] = arr[largest];
      arr[largest] = temp;
      maxHeapify(arr, n, largest);
   }
}
void buildMaxHeap(int arr[], int n) {
   for (int i = n / 2 - 1; i >= 0; i--)
      maxHeapify(arr, n, i);
}
void minHeapify(int arr[], int n, int i) {
   int smallest = i;
   int left = 2 * i + 1;
   int right = 2 * i + 2;
   if (left < n && arr[left] < arr[smallest])
```

```
        smallest = left;

    if (right < n && arr[right] < arr[smallest])
        smallest = right;
    if (smallest != i) {
        int temp = arr[i];
        arr[i] = arr[smallest];
        arr[smallest] = temp;
        minHeapify(arr, n, smallest);
    }
}
void buildMinHeap(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        minHeapify(arr, n, i);
}
void printArray(int arr[], int n) {
    for (int i = 0; i < n; ++i)
        printf("%d ", arr[i]);
    printf("\n");
}
void main() {
    int n;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements of the array: ");
    for (int i = 0; i < n; ++i)
        scanf("%d", &arr[i]);
    buildMaxHeap(arr, n);
    printf("Max-Heap: ");
    printArray(arr, n);
    buildMinHeap(arr, n);
    printf("Min-Heap: ");
    printArray(arr, n);
}
```

Output

```
Enter the number of elements in the array: 5
Enter the elements of the array: 5
0
9
2
3
Max-Heap: 9 3 5 2 0
Min-Heap: 0 2 5 9 3
```