LAB MANUAL

Design and Analysis of Algorithm (DAA) 2301CS402

B-Tech 5th Semester

A.Y. 2025-2026

(Darshan Institute of Engineering and Technology)





Table of Content

Lab	Program	Date	Page	Signature
Lab-1	1. Write a recursive program for calculation of factorial of an		3)
	integer.			
	2. Write a program to calculate the sum of numbers from 1 to n		3	
	using recursion.			
	3. Write a program to count the digits of a given number using		4	
	recursion.			
	4. Write a program to calculate the power of any number using		5	
	recursion.			
Lab-2	1. Write a program to implement stack operations (PUSH, POP, PEEP, CHANGE & DISPLAY).		6	
	2. Write a program to implement queue operations (INSERT, DELETE, DISPLAY).		7	
	3. Write a program to implement singly linked list operations		9	
.	(INSERT, DELETE, DISPLAY).			
Lab-3	1. Write a program to sort array elements using bubble sort.		13	
T 1 4	2. Write a program to sort array elements using insertion sort.		14	
Lab-4	1. Write a program to sort array elements using selection sort.		15	
T 1 F	2. Write a program to sort array elements using heap sort.		16	
Lab-5	1. Write a program to implement linear search algorithm.		18	
T.1.0	2. Write a program to implement binary search algorithm.		19	
Lab-6	1. Write a program to implement quick sort algorithm.		21 22	
Lab-7	 Write a program to implement merge sort algorithm. Write a program to study and implement minimum spanning 		24	
Lab-1	tree using Kruskal's algorithm.		24	
	2. Write a program to study and implement minimum spanning		27	
	tree using Prim's algorithm.		2.	
Lab-8	1. Write a program to study and implement Dijkstra's algorithm.		28	
	2. Write a program to implement Knapsack problem using greedy		30	
	approach.			
Lab-9	1. Write a program to implement Huffman code algorithm.		31	
	2. Write a program to implement job scheduling problem using		33	
	greedy approach.			
Lab-10	1. Write a program to implement making a change problem using		35	
	dynamic programming.			
	2. Write a program to Implement 0/1 knapsack problem using		37	
	dynamic programming.			
Lab-11	1. Write a program to solve all-pairs shortest paths problem		38	
	using Floyd's algorithm		40	
	2. Write a program to implement Largest Common Sub-		40	
T a L 10	sequence.		41	
Lab-12	1. Write a program to implement the DFS algorithm.		41	
Tab 19	2. Write a program to implement the BFS algorithm.		42	
Lab-13	Write a Program for N Queen's problem using Backtracking.		43	
Lab-14	Write a program to implement Rabin-Karp method for pattern		45	
	searching.			



Program 1: Write recursive program for calculation of factorial of an integer.

```
Program 1: Code
     # include<stdio.h>
     double fact(double n){
  3
         if(n==1)
  4
              return 1;
  5
  6
         else
  7
              return(n*fact(n-1));
  8
  9
     void main(){
 10
         double n, ans;
 11
 12
         printf("Enter a number:");
 13
         scanf("%lf",&n);
 14
         ans=fact(n);
 15
         printf("\nAns : %lf \n",ans);
 16
     }
 17
Program 1: Output
Enter a number:20
Ans: 2432902008176640000.000000
```

Program 2. Write a program to calculate the sum of numbers from 1 to n using recursion.

```
Program 2: Code
     #include<stdio.h>
     int sumOfRange(int);
  3
     void main(){
  4
  5
        int n1;
  6
        int sum;
  7
        printf(" Input the last number of the range starting from 1 : ");
  8
        scanf("%d", &n1);
  9
        sum = sumOfRange(n1);
 10
        printf("\n The sum of numbers from 1 to %d : %d\n\n", n1, sum);
 11
 12
 13
     int sumOfRange(int n1)
 14
 15
        int res;
 16
 17
        if (n1 == 1)
 18
 19
            return (1);
 20
        } else
 21
 22
 23
            res = n1 + sumOfRange(n1 - 1); //calling the function sumOfRange itself
 24
        }
 25
```



```
Program 2: Output

Input the last number of the range starting from 1: 20
The sum of numbers from 1 to 20: 210
```

Program 3. Write a program to count the digits of a given number using recursion.

```
Program 3: Code
     #include<stdio.h>
     int noOfDigits(int n1);
     void main()
  5
 6
       int n1,ctr;
 7
         printf(" Input a number : ");
 8
         scanf("%d",&n1);
 9
         ctr = noOfDigits(n1);
10
         printf(" The number of digits in the number is : %d \n\n",ctr);
11
12
13
 14
     int noOfDigits(int n1)
15
16
17
         static int ctr=0;
18
          if(n1!=0)
19
20
               ctr++;
21
              noOfDigits(n1/10);
22
23
         }
 24
         return ctr;
 25
     }
 26
```

Program 3: Output

```
Input a number: 12345
The number of digits in the number is: 5
```

Program 4. Write a program to calculate the power of any number using recursion.

```
Program 4: Code

1  #include <stdio.h>
2  long int CalcuOfPower(int x,int y)
4  {
5   long int result=1;
6  if(y == 0)
```



```
return result;
 8
        result=x*(CalcuOfPower(x,y-1));
 9
10
    void main()
11
12
    {
13
        int bNum,pwr;
14
        long int result;
15
        printf(" Input the base value : ");
16
        scanf("%d",&bNum);
17
18
        printf(" Input the value of power : ");
19
        scanf("%d",&pwr);
20
        result=CalcuOfPower(bNum,pwr);
21
        printf(" The value of %d to the power of %d is : %ld\n\n",bNum,pwr,result);
22
23
    }
```

Program 4: Output

```
Input the base value: 2
Input the value of power: 8
The value of 2 to the power of 8 is: 256
```



Program 1. Write a program to implement stack operations (PUSH, POP, PEEP, CHANGE & DISPLAY)

Program 1: Code

```
#include<stdio.h>
 2
    #define size 5
 3
    struct stack{
 4
       int a[size],top;
 5
       int temp[size], tos;
    }s;
 6
 7
    // Push operation....
    void push(int item){
 8
 9
            s.a[++s.top] = item;
10
11
    // Pop operation....
12
    int pop(){
13
        return s.a[s.top--];
14
   // Display operation....
15
    void display(){
16
17
        int i;
        printf("\nThe stack contains: ");
18
19
        for(i = s.top; i>=0; i--){
20
            printf("\n\t%d", s.a[i]);
21
22
    }
    // Peep operation....
23
24
    void peep(){
        printf("\n\tTop : %d", s.top);
25
        printf("\n\tValue: %d",s.a[s.top]);
26
27
    }
    void change(int row, int new_element){
28
        int i;
29
        int j = -1;
30
        printf("\n\tTop: %d", s.top);
31
32
        for(i=s.top; i>row; i--){
            s.temp[++j] = s.a[s.top--];
33
34
35
        s.a[s.top] = new_element;
36
37
        for(i = j; i > -1; i - -) {
38
            s.a[++s.top] = s.temp[j--];
39
        }
40
41
    void main(){
42
        s.top = -1;
43
        int item, choice, row, new_element;
44
        char ans;
45
        do{
46
            printf("\n 1. Push\n 2. Pop\n 3. Display\n 4. Peep\n 5. Change\n6.
47
    Exit\n");
            printf("-----\n");
48
49
            printf("\n Enter your choice: ");
            scanf("%d", &choice);
50
51
            switch(choice){
```



```
52
            case 1:
53
                 if(s.top >= size-1){
                      printf("\nStack overflow..\n");
54
55
                      break;
56
                 printf("\nEnter item to be pushed: ");
57
                 scanf("%d", &item);
58
59
                 push(item);
60
                 break;
61
            case 2:
                 if(s.top == -1){
62
                     printf("\n..Stack underflow..\n");
63
64
                     break;
65
                 }
66
                 pop();
67
                 break;
68
            case 3:
69
                 display();
70
                 break;
71
            case 4:
72
                 peep();
73
                 break;
74
            case 5:
                 printf("\n\tEnter row no : ");
75
76
                 scanf("%d",&row);
                 printf("\n\tEnter new element: ");
77
                 scanf("%d", &new_element);
78
79
                 change(row, new_element );
80
                 break;
81
            case 6:
82
                 return 0;
83
84
        }while(choice != 6);
85
    }
```

Program 2. Write a program to implement queue operations (INSERT, DELETE, DISPLAY)

```
Program 2: Code
     #include <stdio.h>
  2
     #define MAX 5
  3
     void insert();
  4
     void delete();
  5
     void display();
  6
     int queue_array[MAX];
  7
     int rear = -1;
  8
     int front = - 1;
  9
     main()
 10
     {
 11
         int choice;
 12
         while (1)
 13
          {
 14
              printf("1.Insert element to queue \n");
             printf("2.Delete element from queue \n");
 15
             printf("3.Display all elements of queue \n");
 16
```



```
17
            printf("4.Quit \n");
             printf("Enter your choice : ");
18
             scanf("%d", &choice);
19
20
             switch (choice)
21
                 case 1:
22
23
                 insert();
24
                 break;
25
                 case 2:
26
                 delete();
27
                 break;
28
                 case 3:
29
                 display();
30
                 break;
31
                 case 4:
32
                 exit(1);
33
                 default:
34
                 printf("Wrong choice \n");
35
            }
36
        }
37
38
    void insert()
39
    {
40
        int add item;
41
        if (rear == MAX - 1)
        printf("Queue Overflow \n");
42
43
        else
44
             if (front == - 1)
45
46
             front = 0;
47
             printf("Inset the element in queue : ");
48
             scanf("%d", &add_item);
49
            rear = rear + 1;
50
             queue_array[rear] = add_item;
51
        }
52
    }
    void delete()
53
54
        if (front == - 1 || front > rear)
55
56
             printf("Queue Underflow \n");
57
58
             return ;
59
        }
60
        else
61
        {
             printf("Element deleted from queue is : %d\n", queue_array[front]);
62
63
             front = front + 1;
64
        }
65
    void display()
66
67
        int i;
68
69
        if (front == - 1)
70
             printf("Queue is empty \n");
71
        else
72
        {
             printf("Queue is : \n");
73
74
             for (i = front; i <= rear; i++)</pre>
75
                 printf("%d ", queue_array[i]);
76
             printf("\n");
```



Lab-2 Data Structures

77		}							
78	}								



Program 3. Write a program to implement singly linked list operations (INSERT, DELETE, DISPLAY).

Program 3: Code #include<stdio.h> 2 #include<stdlib.h> struct node 3 4 { 5 int data; 6 struct node *next; 7 **}**; 8 struct node *start; void insert_begin(); 9 void insert_last(); 10 void insert locc(); 11 12 void delete_begin(); void delete_last(); 13 void delete_locc(); 14 15 void print(); 16 void main () 17 { int ch=0; 18 19 while(ch!=8) 20 printf("\nEnter the operation to be performed\n"); 21 printf("\n1.Insert in the begining\n2.Insert at last\n3.Insert at 22 any specified position\n 4.Delete from Beginning\n5.Delete from 23 24 last\n6.Delete node after specified location\n 7.Show\n 8.Exit\n"); 25 scanf("\n%d",&ch); 26 27 switch(ch) /*function calls of all the operations */ 28 { 29 case 1: 30 insert_begin(); 31 break; 32 case 2: 33 insert_last(); 34 break; 35 case 3: 36 insert_locc(); 37 break; 38 case 4: 39 delete_begin(); 40 break; 41 case 5: 42 delete_last(); 43 break; 44 case 6: 45 delete_locc(); 46 break; 47 case 7: 48 print(); 49 break; 50 case 8: 51 exit(0); 52 break; 53 default:



```
54
                 printf("Enter valid option");
 55
             }
 56
         }
 57
                  /*function definition*/
     void insert_begin()
                                            //to insert the node at the beginnning of
 58
 59
     linked list
 60
     {
61
         struct node *p;
62
         int value;
         p=(struct node *) malloc(sizeof(struct node *));
63
 64
         if(p==NULL)
 65
         {
 66
             printf("\nOVERFLOW");
         }
67
68
         else
69
         {
             printf("\nEnter value\n");
70
             scanf("%d",&value);
 71
 72
             p->data=value;
73
             p->next=start;
74
             start=p;
75
         }
 76
 77
     void insert_last()
                                         //to insert the node at the last of linked
 78
     list
79
80
         struct node *p,*temp;
81
         int value;
         p=(struct node*)malloc(sizeof(struct node));
82
83
         if(p==NULL)
84
         {
85
             printf("\nOVERFLOW");
86
         }
87
         else
88
         {
89
             printf("\nEnter value\n");
90
             scanf("%d",&value);
91
             p->data=value;
92
             if(start==NULL)
93
 94
                  p->next=NULL;
 95
                  start=p;
96
             }
97
             else
98
             {
99
                 temp=start;
100
                  while(temp->next!=NULL)
101
                  {
102
                      temp=temp->next;
103
104
                  temp->next=p;
                  p->next=NULL;
105
106
             }
107
         }
108
109
     void insert_locc()
                                        //to insert the node at the specified location
110
     of linked list
111
     {
112
         int i,loc,value;
113
         struct node *p, *temp;
```



```
114
         p=(struct node *)malloc(sizeof(struct node));
115
         if(p==NULL)
116
117
             printf("\nOVERFLOW");
118
119
         else
120
         {
             printf("\nEnter element value");
121
122
             scanf("%d",&value);
123
             p->data=value;
             printf("\nEnter the location after which you want to insert ");
124
125
             scanf("\n%d",&loc);
126
             temp=start;
             for(i=0;i<loc;i++)</pre>
127
128
129
                  temp=temp->next;
130
                  if(temp==NULL)
131
132
                      printf("\ncan't insert\n");
133
                      return;
134
                  }
135
136
             p->next=temp->next;
137
             temp->next=p;
138
         }
139
140
     void delete_begin()
                                //to delete the node present in the beginning of
     the linked list
141
142
     {
         struct node *p;
143
144
         if(start==NULL)
145
         {
             printf("\nList is empty\n");
146
147
         }
148
         else
149
         {
150
             p=start;
151
             start=p->next;
152
             free(p);
         }
153
154
155
     void delete last()
                                 //to delete the node present in the last of the
156
     linked list
157
         struct node *p,*p1;
158
159
         if(start==NULL)
160
161
             printf("\nlist is empty");
162
         else if(start->next==NULL)
163
164
             start=NULL;
165
166
             free(start);
             printf("\nOnly node of the list deleted ...\n");
167
168
         else
169
170
         {
171
             p=start;
172
             while(p->next!=NULL)
173
             {
```



```
174
                  p1=p;
175
                  p=p->next;
176
177
             p1->next=NULL;
178
             free(p);
179
         }
180
181
     void delete_locc()
                           //to delete the node present at the specified of the
182
     linked list
183
     {
184
         struct node *p,*p1;
         int loc,i;
185
         printf("\n Enter the location of the node after which you want to perform
186
     deletion \n");
187
188
         scanf("%d",&loc);
189
         p=start;
190
         for(i=0;i<loc;i++)</pre>
191
192
             p1=p;
193
             p=p->next;
194
195
             if(p==NULL)
196
              {
                  printf("\nCan't delete");
197
198
                  return;
199
             }
200
201
         p1->next=p->next;
202
         free(p);
         printf("\nDeleted node %d ",loc+1);
203
204
                      //to print the values in the linked list
205
     void print()
206
     {
207
         struct node *p;
208
         p=start;
209
         if(p==NULL)
210
         {
             printf("Nothing to print");
211
212
         }
         else
213
214
         {
             printf("\nprinting values\n");
215
216
             while (p!=NULL)
217
             {
218
                  printf("\n%d",p->data);
219
                  p=p->next;
220
             }
221
         }
222
     }
```

Lab-3 Bubble and Insertion sort

Program 1. Write a program to sort array elements using bubble sort.

```
Program 1: Code
     #include<stdio.h>
 2
     void main()
 3
     {
 4
         int array[100], n, i, j, swap;
 5
         printf("Enter number of
 6
     elements:");
 7
         scanf("%d", &n);
 8
         printf("\nEnter %d Numbers:\n",
 9
    n);
10
         for(i = 0; i < n; i++)</pre>
11
12
             scanf("%d", &array[i]);
13
         for(i = 0; i < n - 1; i++)
14
15
             for(j = 0 ; j < n-i-1; j++)</pre>
16
17
18
                  if(array[j] >
19
     array[j+1])
20
21
                      swap=array[j];
22
                      array[j]=array[j+1];
23
                      array[j+1]=swap;
24
                  }
25
             }
26
         printf("Sorted Array:\n");
27
28
         for(i = 0; i < n; i++)</pre>
29
         {
30
             printf("%d\n", array[i]);
         }
31
32
Program 1: Output
Enter number of elements:5
                                              Sorted Array:
Enter 5 Numbers:
                                              5
9
                                              6
8
                                              7
7
                                              8
```

9

Program 2. Write a program to sort array elements using insertion sort.

6



Lab-3 Bubble and Insertion sort

Program 2: Code

```
#include <stdio.h>
 2
    void insertionSort(int arr[], int n)
 3
 4
        int i, key, j;
 5
        for (i = 1; i < n; i++) {
            key = arr[i];
 6
 7
            j = i - 1;
 8
            while (j >= 0 && arr[j] > key) {
 9
                 arr[j + 1] = arr[j];
10
                 j = j - 1;
11
12
            arr[j + 1] = key;
13
        }
14
    void printArray(int arr[], int n)
15
16
17
        int i;
18
        for (i = 0; i < n; i++)</pre>
            printf("%d ", arr[i]);
19
20
        printf("\n");
21
    }
22
    void main()
23
24
        int arr[100], n, i;
        printf("Enter number of elements:");
25
        scanf("%d", &n);
26
        printf("\nEnter %d Numbers:\n", n);
27
28
        for(i = 0; i < n; i++)</pre>
29
30
            scanf("%d", &arr[i]);
31
32
        insertionSort(arr, n); printf("Sorted Array:\n");
33
        printArray(arr, n);
34
    }
```

Program 2: Output

Enter number of elements:5	Sorted Array:
Enter 5 Numbers:	5
9	6
8	7
7	8
6	9
5	

Lab-4 Selection and Heap sort

Program 1. Write a program to sort array elements using selection sort.

```
Program 1: Code
      #include <stdio.h>
   2
      void main()
   3
   4
           int x,arr[100],i,n,pos,count,val,j,minx,minj;
   5
           printf("Enter the number of elements to be sorted: ");
   6
   7
           scanf("%d",&n);
   8
           printf("\nEnter %d Numbers:\n", n);
           for (i = 0; i < n; i++)</pre>
   9
  10
  11
               scanf("%d",&arr[i]);
  12
           }
  13
           for(i=0; i < n; i++)</pre>
  14
  15
  16
            minj=i;
            minx=arr[i];
  17
  18
               for(j=i+1;j<n;j++)</pre>
  19
  20
                    if(arr[j]<minx)</pre>
  21
  22
                      minj=j;
  23
                      minx=arr[j];
  24
  25
  26
            arr[minj]=arr[i];
  27
            arr[i]=minx;
  28
           printf("Sorted Array:\n");
  29
  30
           for(i=0;i<n;++i)</pre>
  31
           {
               printf("%d ",arr[i]);
  32
  33
           }
  34
      }
  35
```

Program 1: Output

Enter number of elements:5	Sorted Array:
Enter 5 Numbers:	5
9	6
8	7
7	8
6	9
5	



Program 2. Write a program to sort array elements using heap sort.

Program 2: Code #include <stdio.h> 2 void main() 3 { 4 int arr[100], n, i, j, c, heap_root, temp; 5 printf("Enter the number of elements to be sorted: "); scanf("%d",&n); 6 printf("\nEnter %d Numbers:\n", n); 7 8 for (i=0; i<n; i++) 9 10 scanf("%d",&arr[i]); 11 12 for (i=1; i<n; i++) 13 14 c = i;15 do 16 { 17 $heap_root = (c-1)/2;$ 18 if(arr[heap_root] < arr[c])</pre> 19 20 temp = arr[heap_root]; arr[heap_root] = arr[c]; 21 22 arr[c] = temp; 23 } 24 c = heap_root; 25 } while (c != 0); 26 printf("Heap array : "); 27 28 for (i=0; i<n; i++) printf("%d ", arr[i]); 29 30 for (j=n-1; j>=0; j--) 31 { 32 temp = arr[0]; arr[0] = arr[j]; 33 arr[j] = temp; 34 35 heap_root = 0; 36 do 37 { $c = 2 * heap_root + 1;$ 38 39 if ((arr[c] < arr[c + 1]) && c < j-1)</pre> 40 41 (arr[heap_root] < arr[c] && c < j)</pre> 42 { 43 temp = arr[heap_root]; 44 arr[heap_root] = arr[c]; 45 arr[c] = temp; 46 47 heap root = c; 48 } while (c<j);</pre> 49 50 printf("\nSorted array : "); 51 for (i=0; i<n; i++) printf("%d ", arr[i]); 52 53 } 54

Lab-4 Selection and Heap sort

55		
22)	

Program 2: Output

Enter the number of elements to be sorted: 5

Enter 5 Numbers:

9

8

7

6

5

Lab-5 Linear and Binary search

Program 1. Write a program to implement linear search algorithm.

```
Program 1: Code
     #include<stdio.h>
  2
     void main()
  3
     {
  4
          int a[20],i,x,n;
  5
          printf("How many elements?");
  6
          scanf("%d",&n);
  7
          printf("Enter array elements:n");
  8
          for(i=0;i<n;++i)</pre>
              scanf("%d",&a[i]);
  9
 10
          printf("nEnter element to search:");
 11
 12
          scanf("%d",&x);
 13
          for(i=0;i<n;++i)</pre>
 14
              if(a[i]==x)
 15
                   break;
          if(i<n)</pre>
 16
              printf("Element found at index %d",i);
 17
 18
          else
 19
              printf("Element not found");
 20
     }
```

```
Program 1: Output

How many elements?5
Enter array elements:
9
8
7
6
5
Enter element to search:5
Element found at index 4
```

Program 2. Write a program to implement binary search algorithm.

```
Program 2: Code

1  #include<stdio.h>
2  #include<stdlib.h>
3  int binsearch(int[], int, int);
void main()
5  {
6    int num, i, key, position;
7    int low, high, list[100];
8    printf("Enter the total number of elements:");
```



Lab-5 Linear and Binary search

```
scanf("%d", &num);
10
        printf("\nEnter the elements of list:\n");
11
        for (i = 0; i < num; i++)
12
           scanf("%d", &list[i]);
13
14
        }
15
        for (i = 0; i < num; i++)</pre>
16
17
             if(list[i] < list[i - 1])</pre>
18
                 printf("Given input is not sorted\n");
19
20
                 return 0;
21
             }
22
23
        low = 0;
        high = num - 1;
24
25
        printf("\nEnter element to be searched: ");
26
        scanf("%d", &key);
        position = binsearch(list, key, low, high);
27
28
        if (position != -1)
29
          printf("\nNumber present at %d", (position + 1));
30
31
32
        else
33
        {
34
          printf("\n The number is not present in the list");
35
36
        return (0);
37
    int binsearch(int a[], int x, int low, int high)
38
39
40
        int mid;
        if (low > high)
41
42
          return -1;
43
        mid = (low + high) / 2;
        if (x == a[mid])
44
45
46
          return (mid);
47
        }
48
        else if (x < a[mid])</pre>
49
          binsearch(a, x, low, mid - 1);
50
51
        }
        else
52
53
        {
           binsearch(a, x, mid + 1, high);
54
55
        }
56
    }
```

```
Program 2: Output

Enter the total number of elements:5
Enter the elements of list:
10
20
30
```



Lab-5 Linear and Binary search

40
50
Enter element to be searched: 20
Number present at 2



Program 1. Write a program to implement quick sort algorithm.

```
Program 1: Code
     #include<stdio.h>
     int pivot(int[],int,int);
     void swap(int[],int,int);
     void quicksort(int[],int,int);
  5
     int pivot(int arr[],int i,int j)
  6
  7
          int p = arr[i],temp;
          int k = i;
  8
  9
          int l = j+1;
 10
          do
 11
          {
 12
 13
              while(arr[k]<=p && k<j);</pre>
          }
 14
          do
 15
          {
 16
 17
              while(arr[1]>p);
 18
          while(k<1)
 19
          {
 20
              temp=arr[k];
 21
              arr[k]=arr[l];
              arr[1]=temp;
 22
 23
              do
 24
              {
 25
                   k++;
              }
                  while(arr[k] <= p);</pre>
 26
 27
              do{
 28
                   1--;
                  while(arr[1] > p);
 29
 30
 31
          temp=arr[i];
 32
          arr[i]=arr[1];
 33
          arr[1]=temp;
 34
          return 1;
 35
     }
 36
     void quicksort(int arr[],int i,int j)
 37
 38
          if(i<j)</pre>
 39
 40
              int position = pivot(arr,i,j);
 41
              quicksort(arr,i,position-1);
 42
              quicksort(arr,position+1,j);
 43
          }
 44
 45
     void main()
 46
     {
 47
          int arr[1000],n,i;
 48
          printf("Enter the number of elements to be sorted: ");
 49
 50
          scanf("%d",&n);
          printf("\nEnter %d Numbers:\n", n);
 51
 52
          for (i=0; i<n; i++)
 53
          {
```



Program 1: Output

```
Enter the number of elements to be sorted: 5
Enter 5 Numbers:
9
8
7
6
5
Sorted Array:
5 6 7 8 9
```

Program 2. Write a program to implement merge sort algorithm.

Program 2: Code

```
#include<stdio.h>
 2
    void disp( );
 3
    void mergesort(int,int,int);
 4
    void msortdiv(int,int);
 5
    int arr[100],n;
 6
    void main( )
 7
    {
 8
 9
        printf("Enter the number of elements to be sorted: ");
        scanf("%d",&n);
10
        printf("\nEnter %d Numbers:\n", n);
11
        for (i=0; i<n; i++)</pre>
12
13
            scanf("%d",&arr[i]);
14
15
        printf("\nBefore Sorting the elements are:");
16
17
18
        msortdiv(0,n-1);
19
        printf("\nAfter Sorting the elements are:");
20
        disp( );
21
22
    void disp( )
23
24
        int i;
25
        for(i=0;i<n;i++)</pre>
26
        printf("%d ",arr[i]);
27
28
    void mergesort(int low,int mid,int high)
```

Lab-6 Quick and Merge sort

```
29
    {
30
         int t[50],i,j,k;
31
         i=low;
32
         j=mid+1;
33
         k=low;
         while((i<=mid) && (j<=high))</pre>
34
35
36
             if(arr[i]>=arr[j])
37
                  t[k++]=arr[j++];
38
             else
39
                  t[k++]=arr[i++];
40
         }
41
         while(i<=mid)</pre>
42
43
             t[k++]=arr[i++];
44
45
         while(j<=high)</pre>
46
         {
47
             t[k++]=arr[j++];
48
49
         for(i=low;i<=high;i++)</pre>
50
         {
51
             arr[i]=t[i];
         }
52
53
54
    void msortdiv(int low,int high)
55
56
         int mid;
57
         if(low!=high)
58
59
             mid=((low+high)/2);
60
             msortdiv(low,mid);
61
             msortdiv(mid+1,high);
             mergesort(low,mid,high);
62
63
         }
64
    }
```

Program 2: Output

```
Enter the number of elements to be sorted: 8
Enter 8 Numbers:
9
8
7
6
5
4
3
2
Before Sorting the elements are:9 8 7 6 5 4 3 2
After Sorting the elements are:2 3 4 5 6 7 8 9
```



Program 1. Write a program to study and implement minimum spanning tree using Kruskal's algorithm.

Program 1: Code

```
#include <stdio.h>
 2
    #define MAX 30
 3
    typedef struct edge {
      int u, v, w;
 4
   } edge;
 6
    typedef struct edge_list {
 7
      edge data[MAX];
 8
      int n;
 9
    } edge list;
10
    edge_list elist;
11
    int Graph[MAX][MAX], n;
12
    edge_list spanlist;
13
    void kruskalAlgo();
    int find(int belongs[], int vertexno);
14
    void applyUnion(int belongs[], int c1, int c2);
15
16
    void sort();
17
    void print();
18
    void kruskalAlgo() {
19
      int belongs[MAX], i, j, cno1, cno2;
20
      elist.n = 0;
      for (i = 1; i < n; i++)
21
22
        for (j = 0; j < i; j++) {
23
          if (Graph[i][j] != 0) {
24
            elist.data[elist.n].u = i;
25
            elist.data[elist.n].v = j;
26
            elist.data[elist.n].w = Graph[i][j];
27
            elist.n++;
28
          }
29
        }
30
        sort();
31
      for (i = 0; i < n; i++)
32
        belongs[i] = i;
33
      spanlist.n = 0;
34
      for (i = 0; i < elist.n; i++) {</pre>
35
        cno1 = find(belongs, elist.data[i].u);
36
        cno2 = find(belongs, elist.data[i].v);
37
        if (cno1 != cno2) {
38
39
          spanlist.data[spanlist.n] = elist.data[i];
40
          spanlist.n = spanlist.n + 1;
41
          applyUnion(belongs, cno1, cno2);
42
        }
43
44
45
    int find(int belongs[], int vertexno) {
46
      return (belongs[vertexno]);
47
48
    void applyUnion(int belongs[], int c1, int c2) {
49
      int i;
50
51
      for (i = 0; i < n; i++)
52
        if (belongs[i] == c2)
```



```
53
           belongs[i] = c1;
 54
     void sort() {
55
 56
       int i, j;
 57
       edge temp;
 58
 59
       for (i = 1; i < elist.n; i++)</pre>
         for (j = 0; j < elist.n - 1; j++)</pre>
60
           if (elist.data[j].w > elist.data[j + 1].w) {
 61
             temp = elist.data[j];
 62
             elist.data[j] = elist.data[j + 1];
 63
64
             elist.data[j + 1] = temp;
65
66
     void print() {
67
68
       int i, cost = 0;
 69
 70
       for (i = 0; i < spanlist.n; i++) {</pre>
         printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v,
71
72
     spanlist.data[i].w);
73
         cost = cost + spanlist.data[i].w;
74
 75
       printf("\nSpanning tree cost: %d", cost);
 76
77
     void main()
78
79
       int i, j, total_cost;
80
       n = 6;
81
       Graph[0][0] = 0;
       Graph[0][1] = 4;
82
83
       Graph[0][2] = 4;
84
       Graph[0][3] = 0;
85
       Graph[0][4] = 0;
86
       Graph[0][5] = 0;
87
       Graph[0][6] = 0;
88
89
       Graph[1][0] = 4;
90
       Graph[1][1] = 0;
91
       Graph[1][2] = 2;
92
       Graph[1][3] = 0;
93
       Graph[1][4] = 0;
 94
       Graph[1][5] = 0;
95
       Graph[1][6] = 0;
96
97
       Graph[2][0] = 4;
98
       Graph[2][1] = 2;
99
       Graph[2][2] = 0;
100
       Graph[2][3] = 3;
101
       Graph[2][4] = 4;
102
       Graph[2][5] = 0;
103
       Graph[2][6] = 0;
104
105
       Graph[3][0] = 0;
106
       Graph[3][1] = 0;
107
       Graph[3][2] = 3;
108
       Graph[3][3] = 0;
109
       Graph[3][4] = 3;
110
       Graph[3][5] = 0;
111
       Graph[3][6] = 0;
112
```

Program 2: Code

21

22

23

24

Lab-7 Kruskal's and Prim's

```
113
        Graph[4][0] = 0;
114
        Graph[4][1] = 0;
115
        Graph[4][2] = 4;
116
        Graph[4][3] = 3;
117
        Graph[4][4] = 0;
118
        Graph[4][5] = 0;
119
        Graph[4][6] = 0;
120
121
        Graph[5][0] = 0;
122
        Graph[5][1] = 0;
123
        Graph[5][2] = 2;
124
        Graph[5][3] = 0;
125
        Graph[5][4] = 3;
126
        Graph[5][5] = 0;
127
        Graph[5][6] = 0;
128
        kruskalAlgo();
129
        print();
Program 1: Output
2 - 1 : 2
5 - 2 : 2
3 - 2 : 3
4 - 3 : 3
1 - 0 : 4
Spanning tree cost: 14
```

Program 2. Write a program to study and implement minimum spanning tree using Prim's algorithm.

```
#include<stdio.h>
    int a,b,u,v,n,i,j,ne=1;
 3
    int visited[10]={0}, min, mincost=0, cost[10][10];
 4
    void main(){
 5
         printf("\nEnter the number of nodes:");
         scanf("%d",&n);
 6
 7
         printf("\nEnter the adjacency matrix:\n");
 8
         for(i=1;i<=n;i++)</pre>
 9
             for(j=1;j<=n;j++)</pre>
10
                  scanf("%d",&cost[i][j]);
11
                  if(cost[i][j]==0)
12
13
                      cost[i][j]=999;
14
             }
15
            visited[1]=1;
         printf("\n");
16
17
         while(ne < n)</pre>
18
             for(i=1,min=999;i<=n;i++)</pre>
19
             for(j=1;j<=n;j++)</pre>
20
             if(cost[i][j]< min)</pre>
```

{

if(visited[i]!=0)

a=u=i;

b=v=j;

min=cost[i][j];

Lab-7 Kruskal's and Prim's

```
if(visited[u]==0 || visited[v]==0)
  26
                     printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
  27
                     mincost+=min;
  28
                     visited[b]=1;
  29
                cost[a][b]=cost[b][a]=999;
  30
           printf("\n Minimum cost=%d",mincost);
  31
  32
       }
Program 2: Output
Enter the number of nodes:6
                                                  Edge 1:(1 3) cost:1
Enter the adjacency matrix:
                                                  Edge 2:(1 2) cost:3
0\ 3\ 1\ 6\ 0\ 0
                                                  Edge 3:(2 5) cost:3
 305030
                                                  Edge 4:(3 6) cost:4
 1\ 5\ 0\ 5\ 6\ 4
                                                  Edge 5:(6 4) cost:2
6\ 0\ 5\ 0\ 0\ 2
                                                  {\rm Minimum\ cost}{=}13
036006
0\ 0\ 4\ 2\ 6\ 0
```

Lab-8 Dijkstra's and Knapsack

Program 1. Write a program to study and implement Dijkstra's algorithm.

Program 1: Code #include<stdio.h> #define INFINITY 9999 3 #define MAX 10 4 void dijikstra(int G[MAX][MAX], int n, int startnode); 5 void main(){ int G[MAX][MAX], i, j, n, u; 6 7 printf("\nEnter the no. of vertices:: "); 8 scanf("%d", &n); 9 printf("\nEnter the adjacency matrix::\n"); 10 for(i=0;i < n;i++)</pre> 11 for(j=0;j < n;j++)</pre> scanf("%d", &G[i][j]); 12 printf("\nEnter the starting node:: "); 13 scanf("%d", &u); 14 15 dijikstra(G,n,u); 16 17 void dijikstra(int G[MAX][MAX], int n, int startnode) 18 19 int cost[MAX][MAX], distance[MAX], pred[MAX]; int visited[MAX], count, mindistance, nextnode, i,j; 20 21 for(i=0;i < n;i++)</pre> 22 for(j=0;j < n;j++)</pre> 23 if(G[i][j]==0) 24 cost[i][j]=INFINITY; 25 else 26 cost[i][j]=G[i][j]; 27 28 for(i=0;i< n;i++)</pre> 29 { 30 distance[i]=cost[startnode][i]; 31 pred[i]=startnode; 32 visited[i]=0; 33 34 distance[startnode]=0; visited[startnode]=1; 35 36 count=1; 37 while(count < n-1){</pre> 38 mindistance=INFINITY; 39 for(i=0;i < n;i++)</pre> 40 if(distance[i] < mindistance&&!visited[i])</pre> 41 { 42 mindistance=distance[i]; 43 nextnode=i; 44 visited[nextnode]=1; 45 46 for(i=0;i < n;i++)</pre> 47 if(!visited[i]) 48 if(mindistance+cost[nextnode][i] < distance[i])</pre> 49 50 distance[i]=mindistance+cost[nextnode][i]; 51 pred[i]=nextnode; 52 53 count++; 54 }

Lab-8 Dijkstra's and Knapsack

```
55
        for(i=0;i < n;i++)</pre>
56
57
             if(i!=startnode)
58
                  printf("\nDistance of %d = %d", i, distance[i]);
59
                  printf("\nPath = %d", i);
60
61
                  j=i;
62
                  do
63
                      j=pred[j];
64
                      printf(" <-%d", j);</pre>
65
66
67
                  while(j!=startnode);
68
             }
69
```

Program 1: Output

```
Enter the no. of vertices:: 4

Enter the adjacency matrix:: 0 \ 1 \ 1 \ 1
1 \ 0 \ 1 \ 0
1 \ 1 \ 0 \ 1
1 \ 0 \ 1 \ 0

Enter the starting node:: 1

Distance of 0 = 1
Path = 0 < -1
Distance of 2 = 1
Path = 2 < -1
Distance of 3 = 2
Path = 3 < -0 < -1
```

Program 2. Write a program to implement Knapsack problem using greedy approach.

```
Program 2: Code
     #include<stdio.h>
     int max(int a, int b) {
        if(a>b){
  3
  4
            return a;
  5
        } else {
  6
            return b;
  7
  8
  9
     int knapsack(int W, int wt[], int val[], int n) {
 10
        int i, w;
 11
        int knap[n+1][W+1];
 12
        for (i = 0; i <= n; i++) {
            for (w = 0; w <= W; w++) {
 13
 14
               if (i==0 || w==0)
 15
                  knap[i][w] = 0;
```

Lab-8 Dijkstra's and Knapsack

```
16
             else if (wt[i-1] <= w)</pre>
17
                 knap[i][w] = max(val[i-1] + knap[i-1][w-wt[i-1]], knap[i-1][w]);
18
19
                 knap[i][w] = knap[i-1][w];
20
          }
21
22
       return knap[n][W];
23
24
    int main() {
       int val[] = {20, 25, 40};
25
26
       int wt[] = \{25, 20, 30\};
27
       int W = 50;
28
       int n = sizeof(val)/sizeof(val[0]);
29
       printf("The solution is : %d", knapsack(W, wt, val, n));
30
       return 0;
31
    }
```

Program 2: Output

The solution is : 65



Program 1. Write a program to implement Huffman code algorithm.

```
Program 1: Code
  1
     #include<string.h>
     #include<stdio.h>
  2
  3
     #include<stdlib.h>
  4
     typedef struct node
  5
  6
              char ch;
  7
              int freq;
  8
              struct node *left;
  9
              struct node *right;
 10
     }node;
 11
 12
     node * heap[100];
 13
     int heapSize=0;
 14
 15
     void Insert(node * element)
 16
     {
 17
              heapSize++;
              heap[heapSize] = element;
 18
              int now = heapSize;
 19
              while(heap[now/2] -> freq > element -> freq)
 20
 21
 22
                      heap[now] = heap[now/2];
 23
                      now /= 2;
 24
 25
              heap[now] = element;
 26
     node * DeleteMin()
 27
 28
     {
 29
              node * minElement,*lastElement;
 30
              int child, now;
 31
              minElement = heap[1];
 32
              lastElement = heap[heapSize--];
 33
              for(now = 1; now*2 <= heapSize ;now = child)</pre>
 34
              {
 35
                   child = now*2;
 36
                        if(child != heapSize && heap[child+1]->freq < heap[child] ->
     freq )
 37
 38
                       {
 39
                               child++;
 40
                       }
 41
                       if(lastElement -> freq > heap[child] -> freq)
 42
                       {
 43
                               heap[now] = heap[child];
 44
                      }
 45
                      else
 46
                      {
 47
                               break;
 48
                       }
 49
 50
              heap[now] = lastElement;
 51
              return minElement;
 52
 53
     void print(node *temp,char *code)
```



```
54
     {
 55
             if(temp->left==NULL && temp->right==NULL)
 56
              {
                      printf("char %c code %s\n",temp->ch,code);
 57
 58
                      return;
 59
             int length = strlen(code);
 60
 61
             char leftcode[10], rightcode[10];
 62
             strcpy(leftcode,code);
 63
             strcpy(rightcode,code);
 64
              leftcode[length] = '0';
              leftcode[length+1] = '\0';
 65
             rightcode[length] = '1';
 66
             rightcode[length+1] = '\0';
 67
 68
             print(temp->left,leftcode);
 69
             print(temp->right,rightcode);
 70
 71
     int main()
 72
     {
73
             heap[0] = (node *)malloc(sizeof(node));
 74
 75
             heap[0]->freq = 0;
 76
             int n;
 77
             printf("Enter the no of characters: ");
             scanf("%d",&n);
 78
 79
             printf("Enter the characters and their frequencies: ");
 80
             char ch;
             int freq,i;
 81
 82
             for(i=0;i<n;i++)</pre>
 83
 84
              {
 85
                      scanf(" %c",&ch);
                      scanf("%d",&freq);
 86
 87
                      node * temp = (node *) malloc(sizeof(node));
 88
                      temp -> ch = ch;
 89
                      temp -> freq = freq;
 90
                      temp -> left = temp -> right = NULL;
 91
                      Insert(temp);
 92
 93
              if(n==1)
 94
              {
 95
                      printf("char %c code 0\n",ch);
 96
                      return 0;
 97
 98
             for(i=0;i<n-1;i++)</pre>
 99
100
                      node * left = DeleteMin();
101
                      node * right = DeleteMin();
                      node * temp = (node *) malloc(sizeof(node));
102
103
                      temp \rightarrow ch = 0;
                      temp -> left = left;
104
105
                      temp -> right = right;
106
                      temp -> freq = left->freq + right -> freq;
107
                      Insert(temp);
108
              }
109
             node *tree = DeleteMin();
110
              char code[10];
111
             code[0] = '\0';
112
             print(tree,code);
```



```
Program 1: Output

Enter the no of characters: 6
Enter the characters and their frequencies: A 5
B 9
C 12
D 13
E 16
F 45
char F code 0
char C code 100
char D code 101
char A code 1100
char B code 1101
char E code 111
```

Program 2. Write a program to implement job scheduling problem using greedy approach.

```
Program 2: Code
     #include <stdio.h>
  2
  3
     typedef struct {
  4
          char id;
  5
          int deadline;
  6
          int profit;
  7
     } Job;
  8
  9
     void jobScheduling(Job jobs[], int n) {
 10
          int i, j, maxDeadline = 0;
 11
 12
          // Find maximum deadline
          for (i = 0; i < n; i++) {</pre>
 13
 14
              if (jobs[i].deadline > maxDeadline)
 15
                  maxDeadline = jobs[i].deadline;
          }
 16
 17
          // Sort jobs by profit (descending order)
 18
          for (i = 0; i < n - 1; i++) {
 19
              for (j = i + 1; j < n; j++) {
 20
                  if (jobs[j].profit > jobs[i].profit) {
 21
                       Job temp = jobs[i];
 22
 23
                       jobs[i] = jobs[j];
                       jobs[j] = temp;
 24
 25
                  }
 26
              }
          }
 27
 28
          // Create time slots
 29
          char slot[maxDeadline + 1];
 30
 31
          for (i = 1; i <= maxDeadline; i++)</pre>
```



```
slot[i] = '-'; // empty slot
32
33
34
        // Fill slots using greedy method
        for (i = 0; i < n; i++) {
35
             for (j = jobs[i].deadline; j > 0; j--) {
36
                 if (slot[j] == '-') {
37
38
                      slot[j] = jobs[i].id;
39
                      break;
40
                 }
41
             }
        }
42
43
44
        // Print job sequence
45
        printf("\nOptimal Job Sequence: ");
46
        for (i = 1; i <= maxDeadline; i++) {</pre>
             if (slot[i] != '-')
47
                 printf("%c ", slot[i]);
48
49
        }
50
    }
51
52
    int main() {
53
        Job jobs[] = {
54
             {'A', 2, 100},
             {'B', 1, 19},
55
             {'C', 2, 27},
{'D', 1, 25},
{'E', 3, 15}
56
57
58
59
        };
60
        int n = sizeof(jobs) / sizeof(jobs[0]);
61
62
63
        printf("Job Scheduling Problem using Greedy Approach\n");
        printf("Jobs: (ID, Deadline, Profit)\n");
64
65
        for (int i = 0; i < n; i++) {
66
             printf("(%c, %d, %d)\n", jobs[i].id, jobs[i].deadline, jobs[i].profit);
        }
67
68
        jobScheduling(jobs, n);
69
70
71
        return 0;
72
    }
```

Program 2: Output

```
Job Scheduling Problem using Greedy Approach
Jobs: (ID, Deadline, Profit)
(A, 2, 100)
(B, 1, 19)
(C, 2, 27)
(D, 1, 25)
(E, 3, 15)
Optimal Job Sequence: A C E
```



Lab-10 Making a change and 0/1 Knapsack problem

Program 1. Write a program to implement making a change problem using dynamic programming.

Program 1: Code

```
#include <stdio.h>
    #define INF 999
 3
    //total different denominations of coins available
    //amount for which we are making change
    #define A 8
 7
    void display(int arr[A+1]);
    void coinChange(int d[N+1], int C[A+1], int S[A+1]);
    void coinSet(int d[N+1], int S[A+1]);
 9
10
    void main() {
11
      //denomination d of the coins
12
      //we will start from index 1
13
      //so, index 0 is set to 0
      int d[N+1] = \{0, 1, 4, 6\};
14
15
      int C[A+1];
16
      int S[A+1];
      coinChange(d, C, S);
17
      printf("\nC[p]\n");
18
19
      display(C);
20
      printf("\nS[p]\n");
21
      display(S);
      printf("\nMin. no. of coins required to make change for amount %d = %d\n", A,
22
23
    C[A]);
      printf("\nCoin Set\n");
24
25
      coinSet(d, S);
26
    }
27
    void coinChange(int d[N+1], int C[A+1], int S[A+1]) {
28
      int i, p, min, coin;
29
      C[0] = 0;
30
      S[0] = 0;
      for(p = 1; p <= A; p++) {</pre>
31
32
        min = INF;
33
        for(i = 1; i <= N; i++) {
          if(d[i] <= p) {
34
            if(1 + C[p - d[i]] < min) {
35
              min = 1 + C[p - d[i]];
36
37
              coin = i;
38
39
          }
40
41
        C[p] = min;
        S[p] = coin;
42
43
44
    void coinSet(int d[N+1], int S[A+1]) {
45
      int a = A;
46
      while(a > 0) {
47
48
        printf("Use coin of denomination: %d\n", d[S[a]]);
49
        a = a - d[S[a]];
50
      }
51
    void display(int arr[A+1]) {
```



Lab-10 Making a change and 0/1 Knapsack problem

```
53
        int c;
 54
        for(c = 0; c <= A; c++) {
          printf("%5d", arr[c]);
 55
 56
        printf("\n");
 57
 58
Program 1: Output
C[p]
  0
      1
         ^{2}
             3
                1
                     ^{2}
                       1
                            2
S[p]
     1
        1
            1
                ^{2}
                                2
                    1
                         3
                            1
Min. no. of coins required to make change for amount 8=2
Use coin of denomination: 4
Use coin of denomination: 4
```

Program 2. Write a program to Implement 0/1 knapsack problem using dynamic programming.

```
Program 2: Code
     #include<stdio.h>
  2
     int max(int a, int b)
  3
          return (a > b)? a : b;
  4
  5
     int knapSack(int W, int wt[], int val[], int n)
  6
  7
         int i, w;
  8
  9
         int K[n+1][W+1];
         for (i = 0; i <= n; i++)
 10
 11
 12
             for (w = 0; w \le W; w++)
 13
                 if (i==0 || w==0)
 14
                     K[i][w] = 0;
 15
                 else if (wt[i-1] <= w)</pre>
 16
                        K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
 17
 18
                 else
 19
                        K[i][w] = K[i-1][w];
 20
             }
 21
         return K[n][W];
 22
 23
 24
     void main()
 25
 26
          int i, n, val[20], wt[20], W;
 27
          printf("Enter number of items:");
```



Lab-10 Making a change and 0/1 Knapsack problem

```
28
         scanf("%d", &n);
         printf("Enter value and weight of items:\n");
29
30
         for(i = 0;i < n; ++i){</pre>
          scanf("%d%d", &val[i], &wt[i]);
31
32
         printf("Enter size of knapsack:");
33
         scanf("%d", &W);
printf("%d", knapSack(W, wt, val, n));
34
35
36
    }
```

Program 2: Output

Enter number of items:3

Enter value and weight of items:

100 20

50 10

150 30

Enter size of knapsack:50

250

Lab-11 Floyd's algorithm and LCS

Program 1. Write a program to solve all-pairs shortest paths problem using Floyd's algorithm

Program 1: Code

```
1
   #include <stdio.h>
    #define INF 99999
3
   #define V 4
                  // Number of vertices in the graph
4
5
   void floydWarshall(int graph[V][V]) {
6
7
        int dist[V][V], i, j, k;
8
9
        // Initialize distance matrix same as input graph
10
        for (i = 0; i < V; i++)
11
12
            for (j = 0; j < V; j++)
13
                dist[i][j] = graph[i][j];
14
15
        // Floyd-Warshall algorithm
16
        for (k = 0; k < V; k++) {
17
18
            for (i = 0; i < V; i++) {
19
                for (j = 0; j < V; j++) {
20
                     if (dist[i][k] + dist[k][j] < dist[i][j])</pre>
21
                         dist[i][j] = dist[i][k] + dist[k][j];
22
23
                }
24
            }
25
        }
26
27
28
        // Print the shortest distance matrix
29
        printf("\nAll-Pairs Shortest Paths (Floyd's Algorithm):\n");
30
        for (i = 0; i < V; i++) {
31
            for (j = 0; j < V; j++) {
32
                if (dist[i][j] == INF)
33
34
                     printf("%7s", "INF");
35
36
                    printf("%7d", dist[i][j]);
37
38
            printf("\n");
39
40
        }
41
   }
42
43
   int main() {
44
45
        /* Example Graph */
46
        int graph[V][V] = {
47
                  5, INF, 10},
            {0,
48
                        3, INF},
            {INF, ∅,
49
            {INF, INF, ∅,
                             1},
50
51
            {INF, INF, INF, ∅}
52
```

Lab-11 Floyd's algorithm and

LCS

```
53
54
55
56
56
57
floydWarshall(graph);
78
59
}

Program 1: Output

Floyd's Algorithm - All Pairs Shortest Path

Floyd's Algorithm - All Pairs Shortes
```

All-Pairs Shortest Paths (Floyd's Algorithm):

INF INF 0 1
INF INF INF 0

Program 2. Write a program to implement Largest Common Sub-sequence.

Program 2: Code

```
#include <stdio.h>
2
    #include <string.h>
3
    int i, j, m, n, LCS_table[20][20];
    char S1[20] = "ACADB", S2[20] = "CBDA", b[20][20];
4
5
    void lcsAlgo() {
6
      m = strlen(S1);
                       n = strlen(S2);
7
      for (i = 0; i <= m; i++)
8
        LCS_table[i][0] = 0;
9
      for (i = 0; i <= n; i++)
10
        LCS table [0][i] = 0;
      for (i = 1; i <= m; i++)
11
12
        for (j = 1; j <= n; j++) {
13
          if (S1[i - 1] == S2[j - 1]) {
            LCS_{table[i][j]} = LCS_{table[i - 1][j - 1] + 1;
14
15
          } else if (LCS_table[i - 1][j] >= LCS_table[i][j - 1]) {
            LCS_table[i][j] = LCS_table[i - 1][j];
16
17
          } else {
18
            LCS_table[i][j] = LCS_table[i][j - 1];
                                                          }
19
      int index = LCS_table[m][n];
20
21
      char lcsAlgo[index + 1];
      lcsAlgo[index] = '\0';
22
23
      int i = m, j = n;
24
      while (i > 0 \&\& j > 0) {
25
        if (S1[i - 1] == S2[j - 1]) {
          lcsAlgo[index - 1] = S1[i - 1];
26
27
          i--;
                                 index--;
                     j--;
28
        else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
29
30
          i--;
31
        else
```



Lab-11 Floyd's algorithm and

LCS

Program 2: Output

S1 : ACADB S2 : CBDALCS: CB

Lab-12 DFS and BFS algorithms

Program 1. Write a program to implement the DFS algorithm.

```
Program 1: Code
      #include<stdio.h>
  2
      int a[20][20],reach[20],n;
  3
      void dfs(int v) {
  4
          int i;
  5
          reach[v]=1;
  6
          for (i=1;i<=n;i++)</pre>
  7
            if(a[v][i] && !reach[i]) {
               printf("\n %d->%d",v,i);
  8
  9
               dfs(i);
          }
 10
 11
      void main() {
 12
 13
          int i,j,count=0;
 14
          printf("\n Enter number of vertices:");
 15
          scanf("%d",&n);
 16
          for (i=1;i<=n;i++) {</pre>
 17
               reach[i]=0;
 18
               for (j=1;j<=n;j++)</pre>
 19
                  a[i][j]=0;
 20
          }
          printf("\n Enter the adjacency matrix:\n");
 21
 22
          for (i=1;i<=n;i++)</pre>
 23
            for (j=1;j<=n;j++)</pre>
             scanf("%d",&a[i][j]);
 24
 25
          dfs(1);
          printf("\n");
 26
 27
          for (i=1;i<=n;i++) {
 28
               if(reach[i])
 29
                  count++;
 30
          if(count==n)
 31
            printf("\n Graph is connected"); else
 32
            printf("\n Graph is not connected");
 33
 34
      }
Program 1: Output
Enter number of vertices:4
                                                 1 -> 2
Enter the adjacency matrix:
                                                 2 -> 3
0 1 1 1
                                                 3->4
1010
1 1 0 1
                                                Graph is connected
1\ 0\ 1\ 0
```

Program 2. Write a program to implement the BFS algorithm.

```
Program 2: Code

1 #include<stdio.h>
```



Lab-12 DFS and BFS algorithms

```
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
    void bfs(int v) {
 4
         for (i=1;i<=n;i++)</pre>
 5
           if(a[v][i] && !visited[i])
 6
            q[++r]=i;
 7
         if(f<=r) {</pre>
 8
             visited[q[f]]=1;
 9
             bfs(q[f++]);
10
         }
11
12
    void main() {
13
        int v;
        printf("\n Enter the number of vertices:");
14
15
        scanf("%d",&n);
16
         for (i=1;i<=n;i++) {</pre>
17
             q[i]=0;
18
             visited[i]=0;
19
20
        printf("\n Enter graph data in matrix form:\n");
21
        for (i=1;i<=n;i++)</pre>
22
           for (j=1;j<=n;j++)</pre>
            scanf("%d",&a[i][j]);
23
         printf("\n Enter the starting vertex:");
24
25
         scanf("%d",&v);
26
         bfs(v);
         printf("\n The node which are reachable are:\n");
27
28
         for (i=1;i<=n;i++)</pre>
29
           if(visited[i])
            printf("%d\t",i); else
30
            printf("\n Bfs is not possible");
31
32
```

Program 2: Output

```
Enter the number of vertices:4
Enter graph data in matrix form:
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0
Enter the starting vertex:1
The node which are reachable are:
1 2 3 4
```



Program 1: Write a Program for N Queen's problem using Backtracking.

Program 1: Code #include <stdio.h> #include <stdbool.h> 3 4 #define N 4 // Change N to any size of chessboard 5 6 7 // Function to print solution 8 void printSolution(int board[N][N]) { 9 for (int i = 0; i < N; i++) {</pre> 10 for (int j = 0; j < N; j++) 11 12 printf(" %d ", board[i][j]); 13 printf("\n"); 14 15 printf("\n"); 16 17 18 19 // Check if queen can be placed on board[row][col] 20 bool isSafe(int board[N][N], int row, int col) { 21 int i, j; 22 23 24 // Check row on left 25 for (i = 0; i < col; i++)</pre> 26 if (board[row][i]) 27 return false; 28 29 30 // Check upper diagonal on left 31 for (i = row, j = col; i >= 0 && j >= 0; i--, j--)32 if (board[i][j]) 33 34 return false; 35 36 // Check lower diagonal on left 37 for (i = row, j = col; j >= 0 && i < N; i++, j--)38 if (board[i][j]) 39 40 return false; 41 42 return true; 43 44 45 46 // Solve N-Queens using backtracking 47 bool solveNQUtil(int board[N][N], int col) { 48 // If all queens placed 49 if (col >= N) 50 51 return true; 52 53 // Try placing queen in each row of this column 54 for (int i = 0; i < N; i++) {

Lab-13 N Queen's problem

```
55
            if (isSafe(board, i, col)) {
56
                board[i][col] = 1;
57
58
                 if (solveNQUtil(board, col + 1))
59
60
                     return true;
61
62
                // BACKTRACK
63
                board[i][col] = 0;
64
            }
65
66
        }
67
        return false; // No place found
68
    }
69
70
71
    void solveNQ() {
72
        int board[N][N] = {0};
73
74
        if (!solveNQUtil(board, 0)) {
75
            printf("Solution does not exist\n");
76
77
            return;
78
        }
79
80
        printSolution(board);
81
    }
82
83
84
    int main() {
85
        printf("N-Queens Problem Solution (N = %d)\n\n", N);
86
        solveNQ();
87
88
        return 0;
89
    }
90
```

Program 1: Output

Lab-14 Rabin-Karp string matching

Program 1: Write a program to implement Rabin-Karp method for pattern searching.

Program 2: Code

```
1
    #include <stdio.h>
 2
    #include <string.h>
 3
 4
    #define d 256 // number of characters in input alphabet
 5
 7
    // Rabin-Karp search function
 8
    void rabinKarp(char text[], char pattern[], int q) {
 9
        int m = strlen(pattern);
10
        int n = strlen(text);
11
12
        int i, j;
13
        int p = 0; // hash value for pattern
14
        int t = 0; // hash value for text
15
        int h = 1;
16
17
18
        // h = pow(d, m-1) % q
19
        for (i = 0; i < m - 1; i++)
20
            h = (h * d) % q;
21
22
23
        // Calculate hash value for pattern and first window of text
24
        for (i = 0; i < m; i++) {
25
            p = (d * p + pattern[i]) % q;
26
            t = (d * t + text[i]) % q;
27
28
        }
29
30
        // Slide the pattern over text one by one
31
        for (i = 0; i <= n - m; i++) {
32
            // Check the hash values
33
34
            if (p == t) {
35
                // Check characters one by one
36
                for (j = 0; j < m; j++) {
37
                     if (text[i + j] != pattern[j])
38
                         break;
39
40
                }
41
                if (j == m)
42
                     printf("Pattern found at index %d\n", i);
43
            }
44
45
46
            // Calculate hash value for next window of text
47
            if (i < n - m) {</pre>
48
                t = (d * (t - text[i] * h) + text[i + m]) % q;
49
50
51
                // We might get negative value of t, convert it
52
                if (t < 0)
53
                    t = (t + q);
```

Lab-14 Rabin-Karp string matching

```
54
            }
55
        }
56
    }
57
58
59
    int main() {
60
        char text[] = "ABCCDDAEFG";
61
        char pattern[] = "CDD";
62
        int q = 101; // A prime number
63
64
65
        printf("Text: %s\n", text);
66
        printf("Pattern: %s\n", pattern);
67
68
        rabinKarp(text, pattern, q);
69
70
71
        return 0;
72
    }
73
```

Program 1: Output

Text: ABCCDDAEFG

Pattern: CDD

Pattern found at index 2