LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING

(AUTONOMOUS)

Advanced Data Structures & Algorithm Analysis Lab Manual

B. Tech III Semester (R23)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision of the Department

The Computer Science & Engineering aims at providing continuously stimulating educational environment to its students for attaining their professional goals and meet the global challenges.

Mission of the Department

- **DM1:** To develop a strong theoretical and practical background across the computer science discipline with an emphasis on problem solving.
- **DM2:** To inculcate professional behaviour with strong ethical values, leadership qualities, innovative thinking and analytical abilities into the student.
- **DM3:** Expose the students to cutting edge technologies which enhance their employability and knowledge.
- **DM4:** Facilitate the faculty to keep track of latest developments in their research areas and encourage the faculty to foster the healthy interaction with industry.

Program Educational Objectives (PEOs)

- **PEO1:** Pursue higher education, entrepreneurship and research to compete at global level.
- **PEO2:** Design and develop products innovatively in computer science and engineering and in other allied fields.
- **PEO3:** Function effectively as individuals and as members of a team in the conduct of interdisciplinary projects; and even at all the levels with ethics and necessary attitude.
- **PEO4:** Serve ever-changing needs of society with a pragmatic perception.

PROGRAMME OUTCOMES (POs):

PO 1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO 2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO 3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO 4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO 5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO 6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO 7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO 8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO 9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO 10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO 11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO 12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

PROGRAMME SPECIFIC OUTCOMES (PSOs):

PSO 1	The ability to apply Software Engineering practices and strategies in software project development using open-source programming environment for the success oforganization.
PSO 2	The ability to design and develop computer programs in networking, web applications and IoT as per the society needs.
PSO 3	To inculcate an ability to analyze, design and implement database applications.

1. **Pre-requisites: Data Structures**

2. **Course Educational Objectives (CEOs):**

The objectives of the course is to

- Acquire practical skills in constructing and managing Data structures
- Apply the popular algorithm design methods in problem-solving scenarios

3. **COURSE OUTCOMES (CO):**

CO1: Implement balanced binary trees, heaps and graph traversals using arrays and linked list. **(Apply-L3)**

CO2: Implement Various Sorting Techniques. **(Apply - L3)**

CO3: : Implement optimization problems using greedy, dynamic programming, backtracking and branch-and-bound techniques. **(Apply - L3)**

CO4: Improve individual / teamwork skills, communication & report writing skills with ethical values.

Sample Programs:

1. Implement AVL Tree operations using linked list.
2. Construct B-Tree an order of 5 with a set of 100 random elements stored in array. Implement searching, insertion and deletion operations.
3. Construct Min and Max Heap using arrays, delete any element and display the content of the Heap.
4. Implement BFT and DFT for given graph, when graph is represented by a) Adjacency Matrix b) Adjacency Lists
5. Write a program for finding the biconnected components in a given graph.
6. Write a program to find maximum and minimum element in array using Divide and conquer.
7. Implement Quick sort and Merge sort and observe the execution time for various input sizes (Average, Worst and Best cases).
8. Compare the performance of Single Source Shortest Paths using Greedy method when the graph is represented by adjacency matrix and adjacency lists.
9. Implement Job Sequencing with deadlines using Greedy strategy.
10. Write a program to solve 0/1 Knapsack problem Using Dynamic Programming.
11. Implement N-Queens Problem Using Backtracking.
12. Implement Travelling Sales Person problem using Branch and Bound approach.

I.a) Write a C program to implement various operations on AVL Tree.

```
 #include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
        int data,height;
        struct node *left,*right;
};
int getheight(struct node *);
int getbalfactor(struct node *);
int largest(int,int);
struct node * insert(struct node *,int);
struct node * createnode(int);
struct node * leftrotate(struct node *);
struct node * rightrotate(struct node *);
void inorder(struct node *);
void preorder(struct node *);
struct node *delete(struct node *,int);
struct node *minnode(struct node *);
int getbalfactor(struct node *r)
{
        if(r==NULL)
                return 0;
        else
                return (getheight(r->left)-getheight(r->right));
}
int getheight(struct node *r)
{
        if(r==NULL)
                return 0;
        else
                return r->height;
}
int largest(int a,int b)
{
        return a>b?a:b;
}
struct node * createnode(int ele)
```

```
{
        struct node *newnode;
        newnode=(struct node *)malloc(sizeof(struct node));
        newnode->data=ele;
        newnode->left=NULL;
        newnode->right=NULL;
        newnode->height=1;
        return newnode;
}
struct node * insert(struct node *r,int ele)
{
        int bf;
        if(r==NULL)
                return createnode(ele);
        else if(r->data>ele)
                r->left=insert(r->left,ele);
        else
                r->right=insert(r->right,ele);
        r->height=1+largest(getheight(r->left),getheight(r->right));
        bf=getbalfactor(r);
        if(bf>1&&ele<r->left->data)
                return rightrotate(r);
        if(bf>1&&ele>r->left->data)
        {
                r->left=leftrotate(r->left);
                return rightrotate(r);
        }
        if(bf<-1&&ele>r->right->data)
                return leftrotate(r);
        if(bf<-1&&ele<r->right->data)
        {
                r->right=rightrotate(r->right);
                return leftrotate(r);
        }
        return r;
}
struct node * leftrotate(struct node *r)
{
        struct node *n1,*n2;
        n1=r->right;
        n2=n1->left;
        n1->left=r;
        r->right=n2;
```

```
        r->height=1+largest(getheight(r->left),getheight(r->right));
        n1->height=1+largest(getheight(n1->left),getheight(n1->right));
        return n1;
}
struct node * rightrotate(struct node * r)
{
        struct node *n1,*n2;
        n1=r->left;
        n2=n1->right;
        n1->right=r;
        r->left=n2;
        r->height=1+largest(getheight(r->left),getheight(r->right));
        n1->height=1+largest(getheight(n1->left),getheight(n1->right));
        return n1;
}
void inorder(struct node *r)
{
        if(r!=NULL)
        {
                inorder(r->left);
                printf("%d ",r->data);
                inorder(r->right);
        }
}
void preorder(struct node *r)
{
        if(r!=NULL)
        {
                printf("%d ",r->data);
                preorder(r->left);
                preorder(r->right);
        }
}

struct node *deletenode(struct node *root,int ele)
{
        int bf;
        struct node *temp;
        if(root==NULL)
                return NULL;
        else if(root->data>ele)
                root->left=deletenode(root->left,ele);
        else if(root->data<ele)
```

```
                root->right=deletenode(root->right,ele);
        else
        {
                if(root->right==NULL&&root->left==NULL)
                {
                    free(root);
                    return NULL;
                }
                else if(root->right==NULL)
                {
                     temp=root->left;
                     free(root);
                     return temp;
                }
                else if(root->left==NULL)
                {
                    temp=root->right;
                    free(root);
                    return temp;
                }
                else
                {
                    temp=minnode(root->right);
                    root->data=temp->data;
                    root->right=deletenode(root->right,temp->data);
                }
        }
        root->height=1+largest(getheight(root->left),getheight(root->right));
        bf=getbalfactor(root);
        if(bf>1&&ele<root->left->data)
                return rightrotate(root);
        if(bf>1&&ele>root->left->data)
        {
                root->left=leftrotate(root->left);
                return rightrotate(root);
        }
        if(bf<-1&&ele>root->right->data)
                return leftrotate(root);
        if(bf<-1&&ele<root->right->data)
        {
                root->right=rightrotate(root->right);
                return leftrotate(root);
        }
```

```
        return root;
}
struct node *minnode(struct node *root)
{
        while(root->left!=NULL)
        {
                root=root->left;
        }
        return root;
}
void main()
{
        struct node *root=NULL;
        int a[20],n,i,ele;
        printf("\nenter size");
        scanf("%d",&n);
        printf("\nenter elements");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
                root=insert(root,a[i]);
        }
        inorder(root);
        printf("\n");
        preorder(root);
        printf("enter element to delete");
        scanf("%d",&ele);
        root=deletenode(root,ele);
        inorder(root);
        printf("\n");
        preorder(root);

}
```

2. Write a C program to implement various operations on Max Heap

Source Code: -

```
#include<stdio.h>
#include<conio.h>
void maxheap(int[],int);
void heapfy(int[],int,int);
void print(int[],int);
void deletenode(int[],int,int);
void main()
{
        int a[20],n,i,ele,c=0;
        clrscr();
        printf("enter size");
        scanf("%d",&n);
        printf("enter elements");
        for(i=0;i<n;i++)
        {
          scanf("%d",&a[i]);
          c++;
          maxheap(a,c);
          print(a,c);
        }
        printf("enter element to delete");
        scanf("%d",&ele);
        deletenode(a,n,ele);
        print(a,n-1);
        getch();
}
void print(int a[],int n)
{
        int i;
        for(i=0;i<n;i++)
          printf("%d ",a[i]);
}
void maxheap(int a[],int n)
{
        int i;
        for(i=n/2-1;i>=0;i--)
          heapfy(a,n,i);
}
void heapfy(int a[],int n,int i)
{
        int temp,large,left,right;
        large=i;
        left=2*i+1;
        right=2*i+2;
        if(left<n&&a[left]>a[large])
          large=left;
```

```
        if(right<n&&a[right]>a[large])
          large=right;
        if(i!=large)
        {
          temp=a[large];
          a[large]=a[i];
          a[i]=temp;
          heapfy(a,n,large);
        }
}
void deletenode(int a[],int n,int ele)
{
        int i,temp;
        for(i=0;i<n;i++)
        {
          if(a[i]==ele)
          {
              break;
          }
        }
        if(i!=n)
        {
          temp=a[i];
          a[i]=a[n-1];
          a[n-1]=temp;
          n--;
          maxheap(a,n);
        }
}
```

3) Write a C program to implement Min Heap.

Source Code:-

```
#include<stdio.h>
#include<conio.h>
void maxheap(int[],int);
void heapfy(int[],int,int);
void print(int[],int);
void deletenode(int[],int,int);
void main()
{
        int a[20],n,i,ele,c=0;
        clrscr();
        printf("enter size");
        scanf("%d",&n);
        printf("enter elements");
        for(i=0;i<n;i++)
        {
          scanf("%d",&a[i]);
          c++;
          maxheap(a,c);
          print(a,c);
        }
        printf("enter element to delete");
        scanf("%d",&ele);
        deletenode(a,n,ele);
        print(a,n-1);
        getch();
}
void print(int a[],int n)
{
        int i;
        for(i=0;i<n;i++)
          printf("%d ",a[i]);
}
void maxheap(int a[],int n)
{
        int i;
        for(i=n/2-1;i>=0;i--)
          heapfy(a,n,i);
}
void heapfy(int a[],int n,int i)
{
        int temp,large,left,right;
        large=i;
        left=2*i+1;
        right=2*i+2;
        if(left<n&&a[left]<a[large])
          large=left;
```

```
        if(right<n&&a[right]<a[large])
          large=right;
        if(i!=large)
        {
          temp=a[large];
          a[large]=a[i];
          a[i]=temp;
          heapfy(a,n,large);
        }
}
void deletenode(int a[],int n,int ele)
{
        int i,temp;
        for(i=0;i<n;i++)
        {
          if(a[i]==ele)
          {
              break;
          }
        }
        if(i!=n)
        {
          temp=a[i];
          a[i]=a[n-1];
          a[n-1]=temp;
          n--;
          maxheap(a,n);
        }
}
```

4.) Write a C program to find connected components from the given graph.

Source Code:-

```
#include<stdio.h>
#include<conio.h>
int ad[20][20];
int visit[20];
int n;
void DFS(int);
int connected();
void main()
{
        int i,edges,c,v,u;
        clrscr();
        printf("enter no.of vertex");
        scanf("%d",&n);
        printf("enter no.of edges");
        scanf("%d",&edges);
        for(i=0;i<edges;i++)
        {
                printf("enter edge between two vertices");
                scanf("%d%d",&u,&v);
                ad[u][v]=1;
                ad[v][u]=1;
        }
        c=connected();
        printf("no.of connected components are: %d",c);
        getch();
}
int connected()
{
        int i,count=0;
        for(i=0;i<n;i++)
                visit[i]=0;
        for(i=0;i<n;i++)
        {
                if(visit[i]==0)
                {
                    DFS(i);
```

```
                        count++;
                    }
            }
            return count;
}
void DFS(int u)
{
            int i;
            visit[u]=1;
            for(i=0;i<n;i++)
            {
                    if(ad[u][i]==1&&visit[i]==0)
                        DFS(i);
            }
}
```

5) Write a C program to traverse the graph using Depth First Search Traversal.

Source Code:-

```
#include<stdio.h>
#include<conio.h>
void DFS(int ad[][20],int[],int,int);
void main()
{
        int i,j,g[20][20],visit[20],s,n;
        clrscr();
        printf("\nenter no.of vertices");
        scanf("%d",&n);
        printf("\nenter adjacent matrix");
        for(i=0;i<n;i++)
        {
                visit[i]=0;
                for(j=0;j<n;j++)
                    scanf("%d",&g[i][j]);
        }
        printf("\nenter source vertex");
        scanf("%d",&s);
        printf("\nDFS order for given graph is: \n");
        DFS(g,visit,s,n);
        getch();
}
void DFS(int ad[][20],int visit[],int s,int n)
{
        int i;
        int top=-1;
        int st[20];
        visit[s]=1;
        st[++top]=s;
        printf("%c-",s+65);
        while(top!=-1)
        {
                s=st[top];
                for(i=0;i<n;i++)
```

```
                {
                    if(ad[s][i]==1&&visit[i]==0)
                    {
                            visit[i]=1;
                            printf("%c-",i+65);
                            st[++top]=i;
                            break;
                    }
                }
                if(i==n)
                    top--;
            }
}
```

6) Write a C program to traverse a graph using Breadth First Search.

Source Code:-

```
#include<stdio.h>
#include<conio.h>
void BFS(int ad[][20],int[],int,int);
void main()
{
        int i,j,g[20][20],visit[20],s,n;
        clrscr();
        printf("\nenter no.of vertices");
        scanf("%d",&n);
        printf("\nenter adjacent matrix");
        for(i=0;i<n;i++)
        {
                visit[i]=0;
                for(j=0;j<n;j++)
                    scanf("%d",&g[i][j]);
        }
        printf("\nenter source vertex");
        scanf("%d",&s);
        printf("\nBFS order for given graph is: \n");
        BFS(g,visit,s,n);
        getch();
}
void BFS(int ad[][20],int visit[],int s,int n)
{
        int i;
        int qu[20];
        int front=-1,rear=-1;
        visit[s]=1;
        front=0;
        qu[++rear]=s;
        printf("%c-",s+65);
        while(front<=rear)
        {
                s=qu[front];
```

```
                for(i=0;i<n;i++)
                {
                    if(ad[s][i]==1&&visit[i]==0)
                    {
                            visit[i]=1;
                            printf("%c-",i+65);
                            qu[++rear]=i;
                    }
                }
                if(i==n)
                    front++;
            }
}
```

7) Write a program to find Articulation Points for the given connected graph using DFS.

Source Code:-

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int ad[10][10],n;
int visit[10],dfn[10],low[10],parent[10];
int ap[10];
int num=0;
void initialize()
{
        int i;
        for(i=0;i<n;i++)
        {
                visit[i]=0;
                dfn[i]=0;
                low[i]=0;
                parent[i]=-1;
                ap[i]=0;
        }
}
int small(int a,int b)
{
        return a<b?a:b;
}
void dfs(int u)
{
        int v;
        int children=0;
        visit[u]=1;
        dfn[u]=low[u]=++num;
        for(v=0;v<n;v++)
        {
                if(ad[u][v]==1)
                {
                    if(visit[v]==0)
                    {
                            children++;
                            parent[v]=u;
                            dfs(v);
                            low[u]=small(low[u],low[v]);
                            if(parent[u]==-1&&children>1)
                            {
                                    ap[u]=1;
                            }
```

```
                                        if(parent[u]!=-1&&low[v]>=dfn[u])
                                        {
                                                ap[u]=1;
                                        }
                                }
                                else if(v!=parent[u])
                                {
                                        low[u]=small(low[u],dfn[v]);
                                }
                        }
                }

}
void findap()
{
                int i;
                for(i=0;i<n;i++)
                {
                                if(visit[i]==0)
                                {
                                    dfs(i);
                                }
                }
                printf("articulation point in the graph\n");
                for(i=0;i<n;i++)
                {
                                if(ap[i])
                                {
                                    printf("%d",i);
                                }
                }
}
void main()
{
                int edges,u,v,i,j;
                printf("enter no of vertices:");
                scanf("%d",&n);
                printf("enter no of edges");
                scanf("%d",&edges);
                for(i=0;i<edges;i++)
                {
                                printf("enter edge(u,v):");
                                scanf("%d%d",&u,&v);
                                ad[u][v]=1;
                                ad[v][u]=1;
                }
                printf("\n the adjacency  matrix:");
```

```
        for(i=0;i<n;i++)
        {
                        for(j=0;j<n;j++)
                        {
                            printf("%d",ad[i][j]);
                        }
                        printf("\n");
        }
        initialize();
        findap();
}
getch();
}
```

8) Write a program for finding Minimum- Maximum element using Divide and Conquer.

Source code:-

```
#include<stdio.h>
#include<conio.h>
void min_max(int[],int,int,int*,int*);
int main()
{
	int a[20],n,i,min,max;
	clrscr();
	printf("enter size");
	scanf("%d",&n);
	printf("enter elements");
	for(i=0;i<n;i++)
		scanf("%d",&a[i]);
	min_max(a,0,n-1,&min,&max);
	printf("Minimum element is %d\nMaximum element is %d",min,max);
	getch();
	return 0;
}
void min_max(int a[],int low,int high,int *min,int *max)
{
	int min1,max1;
	if(low==high)
	{
		*min=*max=a[low];
	}
	else if(low==high-1)
	{
		if(a[low]>a[high])
		{
			*min=a[high];
			*max=a[low];
		}
		else
		{
```

```
                *min=a[low];
                *max=a[high];
        }
    }
    else
    {
        int mid=(low+high)/2;
        min_max(a,low,mid,min,max);
        min_max(a,mid+1,high,&min1,&max1);
        if(min1<*min)
            *min=min1;
        if(max1>*max)
            *max=max1;
    }
}
```

9) Write a C program to sort the array elements using Merge sort.

Source code: -

```
#include<stdio.h>
#include<conio.h>
void merge_sort(int[],int,int);
void combine(int[],int,int,int);
int main()
{
        int a[20],n,i;
        clrscr();
        printf("enter size");
        scanf("%d",&n);
        printf("enter elemnts");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        printf("\nbefore sorting elements are\n");
        for(i=0;i<n;i++)
                printf("%d ",a[i]);
        merge_sort(a,0,n-1);
        printf("\nafter sorting elements are\n");
        for(i=0;i<n;i++)
                printf("%d ",a[i]);
        getch();
        return 0;
}
void merge_sort(int a[],int low,int high)
{
        int mid;
        if(low<high)
        {
                mid=(low+high)/2;
                merge_sort(a,low,mid);
                merge_sort(a,mid+1,high);
                combine(a,low,mid,high);
        }
}
```

```
void combine(int a[],int low,int mid,int high)
{
        int i,l,j,k,temp[20];
        i=low;
        j=mid+1;
        l=low;
        while(i<=mid&&j<=high)
        {
                if(a[i]<a[j])
                {
                        temp[l]=a[i];
                        i++;
                }
                else
                {
                        temp[l]=a[j];
                        j++;
                }
                l++;
        }
        if(i>mid)
        {
                for(k=j;k<=high;k++)
                {
                        temp[l]=a[k];
                        l++;
                }
        }
        else
        {
                for(k=i;k<=mid;k++)
                {
                        temp[l]=a[k];
                        l++;
                }
        }
```

```
        for(i=low;i<=high;i++)
                a[i]=temp[i];
}
```

10) Write a C Program to sort the elements using Quick sort.

Source code:-

```
#include<stdio.h>
#include<conio.h>
void quick_sort(int[],int,int);
int partition(int[],int,int);
void main()
{
        int a[20],n,i;
        clrscr();
        printf("\nenter size");
        scanf("%d",&n);
        printf("\nenter elements");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        printf("\nbefore sorting elements are:\n");
        for(i=0;i<n;i++)
                printf("%d ",a[i]);
        quick_sort(a,0,n-1);
        printf("\nafter sorting elements are:\n");
        for(i=0;i<n;i++)
                printf("%d ",a[i]);
        getch();
}
void quick_sort(int a[],int low,int high)
{
        int j;
        if(low<high)
        {
                j=partition(a,low,high);
                quick_sort(a,low,j-1);
                quick_sort(a,j+1,high);
        }
}
int partition(int a[],int low,int high)
{
```

```
    int i,j,pivot;
    i=low;
    j=high;
    pivot=low;
    do
    {
        while(a[j]>a[pivot])
            j--;
        while(a[i]<a[pivot])
            i++;
        if(i<j)
        {
            int temp=a[i];
            a[i]=a[j];
            a[j]=temp;
        }
    }while(i<j);
    int temp=a[j];
    a[j]=a[pivot];
    a[pivot]=temp;
    return j;
}
```

12) Implement Fractional Knapsack Problem using Greedy strategy

Source code:-

```
#include<stdio.h>
#include<conio.h>
struct object
{
        int id;
        float p,w,r;
};
void main()
{
        struct object o[20],temp;
        int i,j,n,m;
        float x[20],total=0;
        clrscr();
        printf("\nenter no.of objects");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nenter object number");
                scanf("%d",o[i].id);
                printf("\nenter profit & weight");
                scanf("%f%f",&o[i].p,&o[i].w);
                o[i].r=o[i].p/o[i].w;
                x[i]=0;
        }
        printf("\nenter capacity of knapsack");
        scanf("%d",&m);
        for(i=0;i<n-1;i++)
        {
                for(j=0;j<n-1-i;j++)
                {
                        if(o[j].r<o[j+1].r)
                        {
                                temp=o[j];
```

```
                        o[j]=o[j+1];
                        o[j+1]=temp;
                }
        }       }
    for(i=0;i<n;i++)
    {
        if(o[i].w>m)
            break;
        else
        {
            m=m-o[i].w;
            total=total+o[i].p;
            x[o[i].id]=1;
        }       }
    if(i<n)
    {
        x[o[i].id]=m/o[i].w;
        total=total+(x[o[i].id]*o[i].p);
    }
    printf("knapsack vector is:\n(");
    for(i=0;i<n;i++)
    {
        printf("X%d",i+1);
        if(i<n-1)
            printf(" , ");
    }
    printf(") = (");
    for(i=1;i<=n;i++)
    {
        printf("%.2f",x[i]);
        if(i<n)
            printf(" , ");
    }
    printf(")\nTotal profit:%.2f",total);
    getch();
}
```

13) Implement Job Sequencing with deadlines using Greedy strategy.

Source code:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
	int i,j,p[20],d[20],a[20],slot[20],temp,r,total=0,max,n;
	clrscr();
	printf("\nenter no.of objects");
	scanf("%d",&n);
	printf("\nenter profit");
	for(i=0;i<n;i++)
	{

		scanf("%d",&p[i]);
	}
	printf("\nenter deadline");
	for(i=0;i<n;i++)
	{

		scanf("%d",&d[i]);
	}

	max=d[0];
	for(i=0;i<n;i++)
	{
		a[i]=i;
		if(d[i]>max)
			max=d[i];
	}

	for(i=0;i<max;i++)
		slot[i]=-1;
	for(i=0;i<n-1;i++)
	{
```

```
        for(j=0;j<n-1-i;j++)
        {
                if(p[j]<p[j+1])
                {
                        temp=p[j];
                        p[j]=p[j+1];
                        p[j+1]=temp;

                        temp=a[j];
                        a[j]=a[j+1];
                        a[j+1]=temp;
                }               }       }
for(i=0;i<n;i++)
{
        j=a[i];
        r=d[j];
        while(r>=1)
        {
                if(slot[r-1]==-1)
                {
                        slot[r-1]=j;
                        total=total+p[i];
                        break;
                }
                r--;
        }       }
printf("optimal solution is:\n(");
for(i=0;i<max;i++)
{
        printf("j%d",slot[i]+1);
        if(i<max-1)
                printf(" , ");
}
printf(")");
printf("\nTotal profit:%d",total);
getch(); }
```

14) Write a C program to implement f Single Source Shortest Paths using Greedy method when the graph is represented by adjacency matrix

Source code:-

```
#include<stdio.h>
#include<conio.h>
#define INF 9999
void disjkstra(int[][10],int,int);
void main()
{
	int n,s,i,j,ad[10][10];
	clrscr();
	printf("\nenter no.of vertices");
	scanf("%d",&n);

	printf("\nenter adjacent matrix");
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
			scanf("%d",&ad[i][j]);
	}
	printf("\nenter source vertex");
	scanf("%d",&s);
	disjkstra(ad,n,s);
	getch();
}
void disjkstra(int ad[][10],int n,int s)
{
	int cost[10][10],visit[10],pre[10],dist[10],min,i,j,next,count=0;
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
		{
			if(ad[i][j]==0)
				cost[i][j]=INF;
			else
				cost[i][j]=ad[i][j];
```

```
        }
    }
    for(i=0;i<n;i++)
    {
        visit[i]=0;
        dist[i]=cost[s][i];
        pre[i]=s;
    }
    visit[s]=1;
    dist[s]=0;
    count=1;
    while(count<n-1)
    {
        min=INF;
        for(i=0;i<n;i++)
        {
            if(visit[i]==0&&min>dist[i])
            {
                min=dist[i];
                next=i;
            }
        }
        visit[next]=1;
        for(i=0;i<n;i++)
        {
            if(visit[i]==0&&dist[i]>dist[next]+cost[next][i])
            {
                dist[i]=dist[next]+cost[next][i];
                pre[i]=next;
            }
        }
        count++;
    }
    for(i=0;i<n;i++)
    {
        if(i!=s)
```

```
            {
                printf("\nDistance to reach %d is %d\n",i,dist[i]);
                printf("path: %d",i);
                j=i;
                do
                {
                    j=pre[j];
                    printf("<-%d",j);
                }while(j!=s);
            }
        }
}
```

15) Write a program to solve 0/1 Knapsack problem Using Dynamic Programming.

Source code: -

```
#define MAX_ITEMS 100
#include<stdio.h>
#include<conio.h>

int max(int a,int b)
{
	return (a>b)?a:b;
}

int knapsack(int values[],int weights[],int n,int capacity)
{
	int dp[10][10];
	int i,w;
	for(i=0;i<=n;i++)
	{
		for(w=0;w<=capacity;w++)
		{
			if(i==0||w==0)
			{
				dp[i][w]=0;
			}
			else if(weights[i-1]<=w)
			{
				dp[i][w]=max(values[i-1]+dp[i-1][w-weights[i-1]],dp[i-1][w]);
			}
			else
			{
				dp[i][w]=dp[i-1][w];
			}
		}
	}

	return dp[n][capacity];
```

```
}

void main()
{
        int n,capacity,values[100],weights[100],i,result;
        clrscr();
        printf("Enter number of items : \n");
        scanf("%d",&n);
        printf("Enter the profit values of items :\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&values[i]);
        }
        printf("Enter the weights of items : \n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&weights[i]);
        }
        printf("Enter the capacity of the knapsack :\n");
        scanf("%d",&capacity);
        result=knapsack(values,weights,n,capacity);
        printf("Maximum profit of the knapsack : %d\n",result);
        getch();
}
```

16) Write a C program to implement N Queens Problem

Source code: -

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void nqueens(int,int);
int place(int,int);
int x[20],count=0;
int main()
{
        int n,k=1;
        clrscr();
        printf("\nEnter no.of queens");
        scanf("%d",&n);
        nqueens(k,n);
        getch();
        return 0;
}
void nqueens(int k,int n)
{
        int i,j;
        for(i=1;i<=n;i++)
        {
                if(place(k,i))
                {
                        x[k]=i;
                        if(k==n)
                        {
                                count++;
                                printf("\nsolution %d\n",count);
                                for(j=1;j<=n;j++)
                                {
                                        printf("%d ",x[j]);
                                }
                        }
```

```
                else
                        nqueens(k+1,n);
                }
        }
}
int place(int k,int i)
{
        int j;
        for(j=1;j<k;j++)
        {
                if(x[j]==i || abs(k-j)==abs(i-x[j]))
                        return 0;
        }
        return 1;
}
```