1. **MinMax using Divide and Conquer**

```
#include<stdio.h>
void maxmin(int,int,int [],int *,int *);
main()
{
	int a[10],n,p,q,i;
	printf("enter array size");
	scanf("%d",&n);
	printf("enter array elements");
	for(i=0;i<n;i++)
		scanf("%d",&a[i]);
	maxmin(0,n-1,a,&p,&q);
	printf("maximum element is %d\n",p);
	printf("minimum element is %d\n",q);
}
void maxmin(int i,int j,int a[10],int *max,int *min)
{
	int max1,min1,mid;
	if(i==j)
	*max=*min=a[i];
	else if(i==j-1)
	{
		if(a[i]>a[j])
		{
			*max=a[i];
			*min=a[j];
		}
		else
		{
			*max=a[j];
			*min=a[i];
		}
	}
	else
	{
		mid=(i+j)/2;
		maxmin(i,mid,a,max,min);
		maxmin(mid+1,j,a,&max1,&min1);
		if(max1>*max)
			*max=max1;
		if(min1<*min)
			*min=min1;
	}
}
```

2. **K^{th} Smallest element using divide and conquer**

```
#include<stdio.h>
iint partition(int,int);
int ksmall(int,int);
int a[10];
main()
{
        int n,i,k,x;
        printf("enter array size");
        scanf("%d",&n);
        printf("enter array elements");
        for(i=1;i<=n;i++)
                scanf("%d",&a[i]);
        printf("which no of smallest element you want");
        scanf("%d",&k);
        x=ksmall(n,k);
        printf("%d th smallest element in given array is %d\n",k,x);
}
int ksmall(int n,int k)
{
        int lb,ub,j;
        lb=1;
        ub=n;
        while(1)
        {
                j=partition(lb,ub);
                if(j==k)
                        return a[j];
                else if(k<j)
                {
                        ub=j-1;
                }
                else
                {
                        lb=j+1;
                }
        }
}
int partition(int lb,int ub)
{
        int x,down,up,t,j;
        x=a[lb];
        down=lb;
        up=ub;
        while(down<up)
        {
                while(a[down]<=x&&down<=ub)
```

```
                down++;
        while(a[up]>x)
                up--;
        if(down<up)
        {
                t=a[down];
                a[down]=a[up];
                a[up]=t;
        }
    }
    j=up;
    a[lb]=a[j];
    a[j]=x;
    return j;
}
```

3. Knapsack problem using greedy

```
#include<stdio.h>
int mergesort(int,int);
int merge(int,int);
void knapsack(int);
float pr[10],w[10],t[10],x[10],p[10];
main()
{
        int i,n;
        float pro=0;
        printf("enter no of objects");
        scanf("%d",&n);
        printf("enter profits");
        for(i=1;i<=n;i++)
                scanf("%f",&pr[i]);
        printf("enter waits");
        for(i=1;i<=n;i++)
                scanf("%f",&w[i]);
        for(i=1;i<=n;i++)
                t[i]=pr[i]/w[i];
        mergesort(1,n);
        knapsack(n);
        for(i=1;i<=n;i++)
                printf("%.1f\t",x[i]);
        printf("\n");
        for(i=1;i<=n;i++)
                pro=pro+x[i]*pr[i];
        printf("%.1f",pro);
}


void knapsack(int n)
{
        int u,i;
        for(i=0;i<n;i++)
                x[i]=0;
        u=15;
        i=p[0];
        while(i!=0)
        {
                if(w[i]>u)
                        break;
                x[i]=1;
                u=u-w[i];
                i=p[i];
        }
        if(u!=0)
```

```
                    x[i]=u/w[i];
}
int mergesort(int lb,int ub)
{
      int mid,q,r;
      if(lb<ub)
      {
            mid=(lb+ub)/2;
            q=mergesort(lb,mid);
            r=mergesort(mid+1,ub);
            return(merge(q,r));
      }
      else if(lb==ub)
            return lb;
}
int merge(int q,int r)
{
      int i,j,k;
      i=q;j=r;k=0;
      while(i!=0&&j!=0)
      {
            if(t[i]>t[j])
            {
                  p[k]=i;
                  k=i;
                  i=p[i];
           }
           else
            {
                  p[k]=j;
                  k=j;
                  j=p[j];
            }
      }
      if(i==0)
            p[k]=j;
      if(j==0)
            p[k]=i;
      return p[0];
}
```

4. Single Source Shortest Path using Greedy (Dijkstras)

```
#include<stdio.h>
void dijkstra(int [][],int,int [],int [],int);
main()
{
	int i,j,cost[10][10],dist[10],prev[10],v,n,m,a[10],k;
	printf("enter no of nodes");
	scanf("%d",&n);
	printf("enter edges");
	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
		{
			printf("enter edge from %d to %d\n",i,j);
			scanf("%d",&cost[i][j]);
			if(cost[i][j]==0)
				cost[i][j]=999;
		}
	printf("enter source node");
	scanf("%d",&v);
	dijkstra(cost,n,dist,prev,v);
	printf("shortest distance from %d to\n",v);
	for(i=1;i<=n;i++)
	{
		printf("%d is%d\n",i,dist[i]);
		printf("path is %d",v);
		for(j=i,k=1;j!=v;j=prev[j],k++)
		{
				a[k]=j;
		}
		for(m=k-1;m>=1;m--)
			printf("->%d",a[m]);
		printf("\n");


	}




}
void dijkstra(int cost[10][10],int n,int dist[10],int prev[10],int v)
{
	int i,j,min,u,w,s[10];
	for(i=1;i<=n;i++)
	{
		s[i]=0;
		dist[i]=cost[v][i];
		prev[i]=v;
```

```
        }
        s[v]=1;
        dist[v]=0;
        for(i=2;i<n;i++)
        {
                min=999;
                for(j=1;j<=n;j++)
                        if(s[j]==0&&min>dist[j])
                        {
                                u=j;
                                min=dist[j];
                        }
                s[u]=1;
                for(w=1;w<=n;w++)
                {
                        if(cost[u][w]!=999&&s[w]==0)
                        {
                                if(dist[w]>dist[u]+cost[u][w])
                                {
                                        dist[w]=dist[u]+cost[u][w];
                                        prev[w]=u;
                                }
                        }
                }
        }
}
```

5. Minimum cost spanning tree Prims algorithm and Kruskal algorithm

Prims Algorithm

```
#include<stdio.h>
main()
{
	int cost[10][10],i,j,n,t[10][2],min;
	printf("enter no of nodes");
	scanf("%d",&n);
	printf("enter costs\n");
	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
		{
			printf("enter cost between %d and %d\n",i,j);
			scanf("%d",&cost[i][j]);
			if(cost[i][j]==0)
				cost[i][j]=9999;
		}
	min=prim(cost,n,t);
	printf("the minimum cost is %d\n",min);
}
int prim(int cost[10][10],int n,int t[10][2])
{
	int mincost,min,p,q,k,i,j,near[10],l,m;
	min=9999;
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
		if(min>cost[i][j])
		{
			min=cost[i][j];
			k=i;l=j;
		}
	t[1][1]=k;
	t[1][2]=l;

	for(i=1;i<=n;i++)
	{
		if(cost[i][k]<cost[i][l])
			near[i]=k;
		else
			near[i]=l;
	}
	mincost=cost[k][l];
	near[k]=0;
	near[l]=0;

	for(i=2;i<n;i++)
	{
		min=9999;
```

```
                for(j=1;j<=n;j++)
                {
                        if(near[j]!=0)
                        {
                        if(min>cost[j][near[j]])
                        {
                        min=cost[j][near[j]];
                        p=j;
                        q=near[j];
                        }
                        }
                }
                t[i][1]=p;
                t[i][2]=q;
                mincost=mincost+cost[p][q];
                near[p]=0;
                for(k=1;k<=n;k++)
                {
                        if(near[k]!=0&&cost[k][p]<cost[k][near[k]])
                                near[k]=p;
                }
        }
        printf("spaaning tree is\n");
        for(i=1;i<n;i++)
        {
        for(j=1;j<=2;j++)
                printf("%d\t",t[i][j]);
        printf("\n");
        }
        return mincost;
}
```

Kruskal Algorithm

```
#include<stdio.h>
void kruskal(int [][],int,int [][]);
void Union(int,int);
void sort(int [],int);
int find(int);
int id[10];
main()
{
        int cost[10][10],t[10][2],i,j,p,n;
        printf("enter no of nodes");
        scanf("%d",&n);
        printf("enter cost of edges");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
```

```
		{
			printf("enter edge from %d to %d\n",i,j);
			scanf("%d",&cost[i][j]);
		}
	}
	kruskal(cost,n,t);
	for(i=1;i<n;i++)
	{
		for(j=1;j<=2;j++)
			printf("%d\t",t[i][j]);
		printf("\n");
	}
}
void kruskal(int cost[10][10],int n,int t[10][2])
{

	int x,y,b=0,a[50],i,j,g,k,u,v,mincost=0,m=1,s=1,p;
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
		if(cost[i][j]!=0)
			a[b++]=cost[i][j];
	sort(a,b);
	for(i=0;i<b;i++)
		printf("%d\t",a[i]);
	for(i=1;i<=n;i++)
	{
		id[i]=-1;
	}
	while(m<n)
	{
		g=a[0];
		s=1;
		for(i=1;i<=n&&s;i++)
		{
			for(j=1;j<=n&&s;j++)
			{
				if(cost[i][j]==g)
				{
					u=i;
					v=j;
					cost[i][j]=cost[j][i]=0;
					s=0;

				}
			}
		}
		b=b-2;
		for(p=0;p<b;p++)
```

```
                        a[p]=a[p+2];
                x=find(u);
                y=find(v);
                if(x!=y)
                {
                        t[m][1]=u;
                        t[m][2]=v;
                        mincost=mincost+g;
                        Union(x,y);
                        m++;
                }
        }
        printf("\nminimum cost is %d\n",mincost);
}
void Union(int u,int v)
{
        id[u]=v;
}
int find(int i)
{
 while(id[i]>=0)
        i=id[i];
 return i;
}
void sort(int a[10],int n)
{
        int i,j,temp;
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        if(a[i]<a[j])
                        {
                                temp=a[i];
                                a[i]=a[j];
                                a[j]=temp;
                        }
}
```

6. **Multistage forward and backward**

Forward approach

```
#include<stdio.h>
int forward(int [][],int,int,int []);
main()
{
	int n,cost[20][20],k,p[20],i,j,min,stage[20];
	printf("enter no of stages");
	scanf("%d",&k);
	printf("enter no of nodes");
	scanf("%d",&n);
	for(i=1;i<=n;i++)
	{
	printf("enter stage of %d node",i);
	scanf("%d",&stage[i]);
	}
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		{
			if(i<j&&stage[i]+1==stage[j])
			{
				printf("enter cost from %d node to %dnode",i,j);
				scanf("%d",&cost[i][j]);

				if(cost[i][j]==0)
					cost[i][j]=999;
			}
			else
				cost[i][j]=999;
		}
	}
	min=forward(cost,n,k,p);
	printf("minimum cost from source to sink is %d\n",min);
	printf("the path is");
	for(i=1;i<=k;i++)
		printf("%d->",p[i]);

}
int forward(int c[20][20],int n,int k,int p[20])
{
	int cost[20],min,l,i,j,d[20];
	cost[n]=0;
	for(j=n-1;j>=1;j--)
	{
		min=999;
		for(i=j;i<=n;i++)
		{
```

```
                        if(c[j][i]!=999&&min>c[j][i]+cost[i])
                        {
                                min=c[j][i]+cost[i];
                                l=i;
                        }
                }
                cost[j]=c[j][l]+cost[l];
                d[j]=l;
        }

        p[1]=1;
        p[k]=n;
        for(j=2;j<k;j++)
                p[j]=d[p[j-1]];
        return cost[1];
}
```

Backward approach

```
#include<stdio.h>
int backward(int [][],int,int,int []);
main()
{
        int n,cost[20][20],k,p[20],i,j,min,stage[20];
        printf("enter no of stages");
        scanf("%d",&k);
        printf("enter no of nodes");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
        printf("enter stage of %d node",i);
        scanf("%d",&stage[i]);
        }
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        if(i<j&&stage[i]+1==stage[j])
                        {
                                printf("enter cost from %d node to %dnode",i,j);
                                scanf("%d",&cost[i][j]);

                                if(cost[i][j]==0)
                                        cost[i][j]=999;
                        }
                        else
                                cost[i][j]=999;
                }
        }
```

```
	min=backward(cost,n,k,p);
	printf("minimum cost from source to sink is %d\n",min);
	printf("the path is");
	for(i=1;i<=k;i++)
		printf("%d->",p[i]);

}
int backward(int c[20][20],int n,int k,int p[20])
{
	int cost[20],min,l,i,j,d[20];
	cost[1]=0;
	for(j=2;j<=n;j++)
	{
		min=999;
		for(i=1;i<=j;i++)
		{
			if(c[i][j]!=999&&min>c[i][j]+cost[i])
			{
				min=c[i][j]+cost[i];
				l=i;
			}
		}
		cost[j]=c[l][j]+cost[l];
		d[j]=l;
	}

	p[1]=1;
	p[k]=n;
	for(j=k-1;j>=2;j--)
		p[j]=d[p[j+1]];
	return cost[n];
}
```

7. All Pair Shortest Paths

```
#include<stdio.h>
void allpair(int [][],int n,int a[][]);
int min(int,int);
main()
{
        int i,j,n,cost[10][10],a[10][10];
        printf("enter no of nodes");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
                printf("enter cost from %d to %d node",i,j);
                scanf("%d",&cost[i][j]);
                if(cost[i][j]==0)
                        cost[i][j]=999;
        }
        allpair(cost,n,a);
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
                if(i!=j)
                printf("the shortest distance from %d to %d is %d\n",i,j,a[i][j]);
}
}

void allpair(int cost[10][10],int n,int a[10][10])
{
        int i,j,k;
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
                a[i][j]=cost[i][j];
        for(k=1;k<=n;k++)
                for(i=1;i<=n;i++)
                        for(j=1;j<=n;j++)
                                a[i][j]=min(a[i][k]+a[k][j],a[i][j]);
}
int min(int a,int b)
{
if(a>b)
return b;
return a;
}
```

8. **BFS &DFS**

BFS

```
#include<stdio.h>
void bfs(int);
int cost[10][10],visit[10],n;
main()
{
	int i,j;
	printf("enter no of nodes");
	scanf("%d",&n);
	printf("enter costs 1 if there is edge\n");
	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
		{
			printf("%d-%d:",i,j);
			scanf("%d",&cost[i][j]);
		}
	for(i=1;i<=n;i++)
		visit[i]=0;
	bfs(1);
}
void bfs(int v)
{
	int u,q[20],f=0,r=0,w;
	u=v;
	visit[v]=1;
	printf("%d\n",v);
	while(1)
	{
		for(w=1;w<=n;w++)
			if(visit[w]==0)
			{
				q[r++]=w;
				visit[w]=1;
				printf("%d",w);
			}
		if(f==r)
			return;
		u=q[f++];
	}
}
```

DFS

```
#include<stdio.h>
void dfs(int);
int n,cost[10][10],visit[10];
main()
{
	int i,j;
```

```
	printf("enter no of nodes");
	scanf("%d",&n);
	printf("enter 1 if there is edge");
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
	{
		printf("%d-%d:",i,j);
		scanf("%d",&cost[i][j]);
	}
	for(i=1;i<=n;i++)
		visit[i]=0;
	dfs(1);
}
void dfs(int v)
{
	int u;
	visit[v]=1;
	printf("%d\n",v);
	for(u=1;u<=n;u++)
	{
		if(cost[u][v]==1)
		if(visit[u]==0)
			dfs(u);
	}
}
```

9. Bi Connected Components

```
#include<stdio.h>
void bicomp(int,int);
int min(int,int);
int dfn[20],l[20],n,cost[10][10],num=1,sta[20],top=-1;
main()
{
	int i,j;
	printf("enter no of nodes");
	scanf("%d",&n);
	printf("enter 1 if there is edge");
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
	{
		printf("%d-%d:",i,j);
		scanf("%d",&cost[i][j]);
	}
	for(i=1;i<=n;i++)
	{
		dfn[i]=0;
		l[i]=0;
	}
	bicomp(1,0);
}
void bicomp(int u,int v)
{
	int w,x,y;
	dfn[u]=num;
	l[u]=num;
	num++;
	for(w=1;w<=n;w++)
	{
		if(cost[w][u]!=0)
		{
			if(w!=v&&dfn[w]<dfn[u])
			{
				sta[++top]=u;
				sta[++top]=w;
			}
			if(dfn[w]==0)
			{
				bicomp(w,u);
				if(l[w]>=dfn[u])
				{
					printf("bi connected component is\n");
					do
					{
						x=sta[top--];
```

```
                                        y=sta[top--];
                                        printf("%d\t%d\n",x,y);
                                }
                                while(!((x==u&&y==w)||(x==w&&y==u)));
                        }
                        l[u]=min(l[u],l[w]);
                }
                else if(w!=v)
                        l[u]=min(l[u],dfn[w]);
        }
    }
}
int min(int a,int b)
{
    if(a>b)
        return b;
    return a;
}
```

10. **N Queens**

```
#include<stdio.h>
#include<math.h>
void nqueen(int,int);
int place(int,int);
int x[10];
main()
{
        int n;
        printf("enter no of queens");
        scanf("%d",&n);
        nqueen(1,n);
}
void nqueen(int k,int n)
{
        int i,p;
        for(i=1;i<=n;i++)
        {
                if(place(k,i))
                {
                        x[k]=i;
                        if(k==n)
                        {
                                for(p=1;p<=k;p++)
                                        printf("%d\t",x[p]);
                                printf("\n");
                        }
                        else
                                nqueen(k+1,n);
                }
        }
}
int place(int k,int i)
{
        int j;
for(j=1;j<k;j++)
        if(x[j]==i||abs(j-k)==abs(x[j]-i))
                return 0;
        return 1;
}
```

11. **Write a program to color the nodes in a given graph such that no two adjacent can have the same color using backtracking.**

```
#include<stdio.h>
int G[10][10],x[10],n,m;
void mcolor(int);
```

```
int main()
{
	int i,j;
	printf("enter no of nodes");
	scanf("%d",&n);
	printf("enter 1 if there is edge");
	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
		{
			printf("%d-%d:",i,j);
			scanf("%d",&G[i][j]);
		}
	printf("Enter no. of colors\n");
	scanf("%d",&m);
	mcolor(1);
	return(0);
}
void write()
{
	int i;
	for(i=1;i<=n;i++)
		printf("%d",x[i]);
	printf("\n");
}
void nextValue(int k)
{
	int j;
	while(1)
	{
		x[k]=(x[k]+1)%(m+1);
		if(x[k]==0)
			return;
		for(j=1;j<=n;j++)
			if((G[k][j]!=0)&&(x[k]==x[j]))
				break;
		if(j==(n+1))
			return;
	}
}
```

```
void mcolor(int k)
{
	while(1)
	{
		nextValue(k);
		if(x[k]==0)
			return;
		if(k==n)
			write();
		else
			mcolor(k+1);
	}
}
```