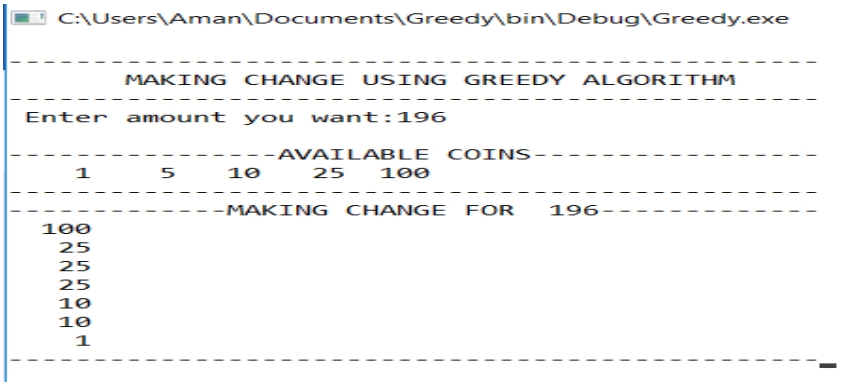
Program No18-Write a program that implements make a change using greedy.

```
#include<stdio.h>
#include<conio.h>
int C[]={1,5,10,25,100};
void make_change(int n);
int bestsol(int,int);
void main()
{
 int n;
 printf("\n-----------------------------------------------");
 printf("\n      MAKING CHANGE USING GREEDY ALGORITHM     ");
 printf("\n-----------------------------------------------");
 printf("\n Enter amount you want:");
 scanf("%d",&n);
 make_change(n);
getch();
}
void make_change(int n)
{
 int S[100],s=0,x,ind=0,i;
 printf("\n----------------AVAILABLE COINS----------------\n");
 for(i=0;i<= 4;i++)
  printf("%5d",C[i]);
 printf("\n-----------------------------------------------");
while(s!=n)
 {
  x=bestsol(s,n);
  if(x==-1)
  {}
  else
  {
   S[ind++]=x;
   s=s+x;
  }
 }
 printf("\n-------------MAKING CHANGE FOR %4d-------------",n);
```

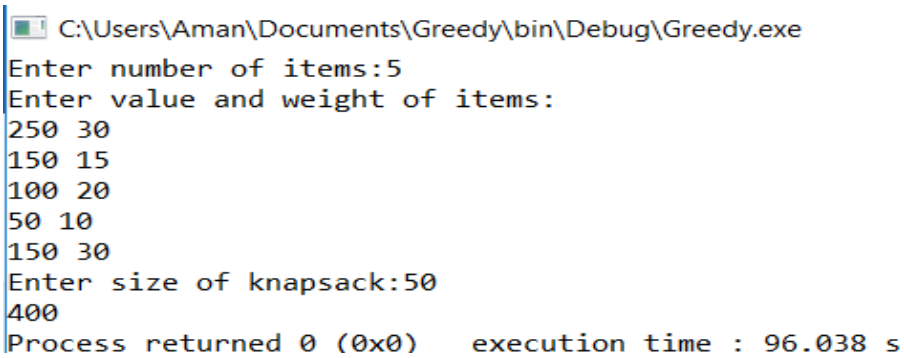
```
 for(i=0;i < ind;i++)
 {
  printf("\n%5d",S[i]);
 }
 printf("\n-----------------------------------------------");
}
int bestsol(int s,int n)
{
 int i;
 for(i=4;i>-1;i--)
 {
  if((s+C[i]) <= n)
   return C[i];
 }
 return -1;
}
```



Program No-19 Write a program that implements 0/1 Knapsack Problem using dynamic Programming.

```
#include<stdio.h>
int max(int a, int b) { return (a > b)? a : b; }
int knapSack(int W, int wt[], int val[], int n)
{
  int i, w;
  int K[n+1][W+1];

  for (i = 0; i <= n; i++)
  {
     for (w = 0; w <= W; w++)
     {
        if (i==0 || w==0)
           K[i][w] = 0;
        else if (wt[i-1] <= w)
             K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
        else
             K[i][w] = K[i-1][w];
     }
  }

  return K[n][W];
}

int main()
{
   int i, n, val[20], wt[20], W;

   printf("Enter number of items:");
   scanf("%d", &n);

   printf("Enter value and weight of items:\n");
   for(i = 0;i < n; ++i){
      scanf("%d%d", &val[i], &wt[i]);
   }
```

```
    printf("Enter size of knapsack:");
    scanf("%d", &W);

    printf("%d", knapSack(W, wt, val, n));
    return 0;
}
```

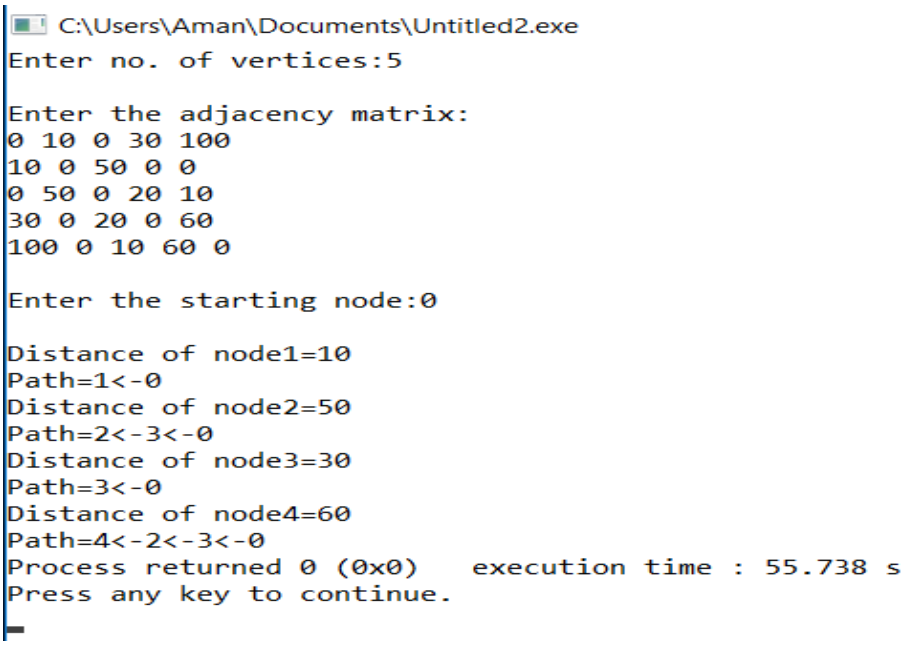


Program No-20 Write a program that implements Dijkstra's Algorithm.

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
   int G[MAX][MAX],i,j,n,u;
   printf("Enter no. of vertices:");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix:\n");

   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         scanf("%d",&G[i][j]);

   printf("\nEnter the starting node:");
   scanf("%d",&u);
   dijkstra(G,n,u);

   return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{

   int cost[MAX][MAX],distance[MAX],pred[MAX];
   int visited[MAX],count,mindistance,nextnode,i,j;


                                                  //pred[] stores the predecessor of each node
                                                    //count gives the number of nodes seen so far
                                                  //create the cost matrix

   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         if(G[i][j]==0)
            cost[i][j]=INFINITY;
         else
            cost[i][j]=G[i][j];

                                                  //initialize pred[],distance[] and visited[]
   for(i=0;i<n;i++)
   {
      distance[i]=cost[startnode][i];
```

```
   pred[i]=startnode;
   visited[i]=0;
}

distance[startnode]=0;
visited[startnode]=1;
count=1;

while(count<n-1)
{
   mindistance=INFINITY;

                                                    //nextnode gives the node at minimum distance

   for(i=0;i<n;i++)
      if(distance[i]<mindistance&&!visited[i])
      {
         mindistance=distance[i];
         nextnode=i;
      }

                                                    //check if a better path exists through nextnode

      visited[nextnode]=1;
      for(i=0;i<n;i++)
         if(!visited[i])
            if(mindistance+cost[nextnode][i]<distance[i])
            {
               distance[i]=mindistance+cost[nextnode][i];
               pred[i]=nextnode;
            }
   count++;
}

//print the path and distance of each node
for(i=0;i<n;i++)
   if(i!=startnode)
   {
      printf("\nDistance of node%d=%d",i,distance[i]);
      printf("\nPath=%d",i);

      j=i;
      do
      {
         j=pred[j];
         printf("<-%d",j);
      }while(j!=startnode);
}
```

}


C:\Users\Aman\Documents\Untitled2.exe
Enter no. of vertices:5
Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0
Enter the starting node:0
Distance of node1=10
Path=1<-0
Distance of node2=50
Path=2<-3<-0
Distance of node3=30
Path=3<-0
Distance of node4=60
Path=4<-2<-3<-0
Process returned 0 (0x0)   execution time : 55.738 s
Press any key to continue.

Program No-21 Write a program that implements Longest Common SubSequence.

```
#include<stdio.h>
#include<string.h>

int i,j,m,n,c[20][20];
char x[20],y[20],b[20][20];

void print(int i,int j)
{
        if(i==0 || j==0)
                return;
        if(b[i][j]=='c')
        {
                print(i-1,j-1);
                printf("%c",x[i-1]);
        }
        else if(b[i][j]=='u')
                print(i-1,j);
        else
                print(i,j-1);
}

void lcs()
{
        m=strlen(x);
        n=strlen(y);
        for(i=0;i<=m;i++)
                c[i][0]=0;
        for(i=0;i<=n;i++)
                c[0][i]=0;


                        //c, u and l denotes cross, upward and downward directions respectively

        for(i=1;i<=m;i++)
                for(j=1;j<=n;j++)
                {
                        if(x[i-1]==y[j-1])
                        {
                                c[i][j]=c[i-1][j-1]+1;
                                b[i][j]='c';
                        }
                        else if(c[i-1][j]>=c[i][j-1])
                        {
                                c[i][j]=c[i-1][j];
                                b[i][j]='u';
```

```
                                        }
                                        else
                                        {
                                                c[i][j]=c[i][j-1];
                                                b[i][j]='l';
                                        }
                        }
}

int main()
{
        printf("Enter 1st sequence:");
        scanf("%s",x);
        printf("Enter 2nd sequence:");
        scanf("%s",y);
        printf("\nThe Longest Common Subsequence is ");
        lcs();
        print(m,n);
    return 0;
}
```

```
C:\Users\Aman\Documents\Untitled3.exe
Enter 1st sequence:ACFGHDEI
Enter 2nd sequence:ABFHDI

The Longest Common Subsequence is AFHDI
Process returned 0 (0x0)   execution time : 42.105 s
Press any key to continue.
```

Program No-22 Write a program that implements N Queen Problem.

```
#include<stdio.h>
#include<math.h>

int board[20],count;

int main()
{
 int n,i,j;
 void queen(int row,int n);

 printf(" - N Queens Problem Using Backtracking -");
 printf("\n\nEnter number of Queens:");
 scanf("%d",&n);
 queen(1,n);
 return 0;
}

//function for printing the solution
void print(int n)
{
 int i,j;
 printf("\n\nSolution %d:\n\n",++count);

 for(i=1;i<=n;++i)
  printf("\t%d",i);

 for(i=1;i<=n;++i)
 {
  printf("\n\n%d",i);
  for(j=1;j<=n;++j)                                    //for nxn board
  {
   if(board[i]==j)
    printf("\tQ");                                     //queen at i,j position
   else
    printf("\t-");                                    //empty slot
  }
 }
}
```

/*funtion to check conflicts
If no conflict for desired postion returns 1 otherwise returns 0*/

```
int place(int row,int column)
{
 int i;
 for(i=1;i<=row-1;++i)
```

```
 {
                                                    //checking column and digonal conflicts
  if(board[i]==column)
   return 0;
  else
   if(abs(board[i]-column)==abs(i-row))
    return 0;
 }

 return 1; //no conflicts
}

                                                    //function to check for proper positioning of queen
void queen(int row,int n)
{
 int column;
 for(column=1;column<=n;++column)
 {
  if(place(row,column))
  {
   board[row]=column;                                //no conflicts so place queen
   if(row==n)                                        //dead end
    print(n);                                        //printing the board configuration
   else                                              //try queen with next position
    queen(row+1,n);
  }
 }
}
```

```
C:\Users\Aman\Documents\Untitled4.exe
 - N Queens Problem Using Backtracking -
Enter number of Queens:4

Solution 1:
          1         2         3         4
1         -         Q         -         -
2         -         -         -         Q
3         Q         -         -         -
4         -         -         Q         -
Solution 2:
          1         2         3         4
1         -         -         Q         -
2         Q         -         -         -
3         -         -         -         Q
4         -         Q         -         -
Process returned 0 (0x0)   execution time : 14.654 s
Press any key to continue.
```

Program No-23 Write a Program that implements knapsack using backtracking

```
#include<stdio.h>
#include<conio.h>
#define MAX 20
float final_profit;
int w[MAX];
int p[MAX];
int n,m;
int temp[MAX],x[MAX];
float final_wt;

float Bound_Calculation(int,int,int);
void BackTracking(int,int,int);

void main()
{
 int i;

 printf("\n-----------------------------------------------------------");
 printf("\n     KNAPSACK PROBLEM USING BACKTRACKING");
 printf("\n-----------------------------------------------------------");
 printf("\n Enter number of Objects you want:");
 scanf("%d",&n);
 printf("\n-----------------------------------------------------------");
 for(i=1;i<=n;i++)
 {
  printf("\n Enter Weight and value for object%d:",i);
  scanf("%3d %3d",&w[i],&p[i]);
 }
 printf("\n Enter Capacity of Knapsack:");
 scanf("%d",&m);
 getch();
 printf("\n-----------------------------------------------------------");
 printf("\n Weight\tProfit");
 printf("\n-----------------------------------------------------------");

 for(i=1;i<=n;i++)
 {
  printf("\n %d \t %d",w[i],p[i]);
 }

 BackTracking(1,0,0);

 printf("\n-----------------------------------------------------------");
 printf("\n Following Objects are included:");
 printf("\n-----------------------------------------------------------");
 for(i=1;i<=n;i++)
```

```
 {
  if(x[i]==1)
   printf("\n%d",i);
 }
 printf("\n-----------------------------------------------------------");
 printf("\n Final Weight:%0.2f",final_wt);
 printf("\n Final Profit:%0.2f",final_profit);
getch();
}
float Bound_Calculation(int cp,int cw,int k)
{
 int ub,c,i;
 ub=cp;
 c=cw;
 for(i=k+1;i<=n;i++)
 {
  c=c+w[i];
  if(c < m)
   ub=ub+p[i];
  else
   return (ub+(1-(c-m)/w[i])*p[i]);
 }
 return ub;
}
void BackTracking(int k,int cp,int cw)
{
 int new_k,new_cp,new_cw,j;
 if(cw+w[k]<=m)
 {
  temp[k]=1;
  if(k < n)
  {
   new_k=k+1;
   new_cp=cp+p[k];
   new_cw=cw+w[k];
   BackTracking(new_k,new_cp,new_cw);
  }
  if((new_cp>final_profit)&&(k==n))
  {
   final_profit=new_cp;
   final_wt=new_cw;
   for(j=1;j<=k;j++)
   {
    x[j]=temp[j];
   }
  }
 }
```

```
 if(Bound_Calculation(cp,cw,k)>=final_profit)
 {
  temp[k]=0;
  if(k < n)
   BackTracking(k+1,cp,cw);
  if((cp>final_profit)&&(k==n))
  {
   final_profit=cp;
   final_wt=cw;
   for(j=1;j<=n;j++)
    x[j]=temp[j];
  }
 }
}
```