Can you please help me to modify my program so that it will be able to use hop constraint rule .

```
#include<stdio.h>
#define MAX 10
#define TEMP 0
#define PERM 1
#define infinity 1000

struct node
{
    int predecessor;
    int dist;
    int status;
};
```

```
int adj[MAX][MAX];
int n;
```

```
void create_graph()
{
    int i,max_edges,origin,destin,wt;
```

```
    printf("Enter number of vertices : ");
    scanf("%d",&n);
    max_edges=n*(n-1);
```

```
    for(i=0;i<=max_edges;i++)
    {
        printf("Enter edge %d(-1 -1 to quit) : ",i);
        scanf("%d %d",&origin,&destin);
        if((origin==1) && (destin==-1))
            break;
        printf("Enter weight for this edge : ");
        scanf("%d",&wt);
        if( origin > n || destin > n || origin<=-1 || destin<=-1)
        {
            printf("Invalid edge!\n");
            i--;
        }
        else
            adj[origin][destin]=wt;
    }
}
/*dispay the matrix */
void display()
{
    int i,j;
    for(i=0;i<=n;i++)
    {
        for(j=0;j<=n;j++)
```

```
            printf("%3d",adj[i][j]);
        printf("\n");
    }

}

int findpath(int s,int d,int path[MAX],int *sdist)
{
    struct node state[MAX];
    int i,min,count=0,current,newdist,u,v;
    *sdist=0;
    /* Make all nodes temporary */
    for(i=1;i<=n;i++)
    {
        state[i].predecessor=0;
        state[i].dist = infinity;
        state[i].status = TEMP;
    }

    /*Source node should be permanent*/
    state[s].predecessor=0;
    state[s].dist = 0;
    state[s].status = PERM;

    /*Starting from source node until destination is found*/
    current=s;
    while(current!=d)
    {
        for(i=0;i<=n;i++)
        {
            /*Checks for adjacent temporary nodes */
            if ( adj[current][i] > 0 && state[i].status == TEMP )
            {
                newdist=state[current].dist + adj[current][i];

                if( newdist < state[i].dist )
                {
                    state[i].predecessor = current;
                    state[i].dist = newdist;
                }
            }
        }

        //Search for temporary node with minimum distand make it curren
t node
        min=infinity;
        current=0;
        for(i=0;i<=n;i++)
        {
            if(state[i].status == TEMP && state[i].dist < min)
```

```
            {
                min = state[i].dist;
                current=i;
            }
        }
```

```
        if(current==0) /*If Source or Sink node is isolated*/
            return 0;
        state[current].status=PERM;
    }/*End of while*/
```

```
    /* Getting full path in array from destination to source    */
    while( current!=0 )
    {
        count++;
        path[count]=current;
        current=state[current].predecessor;
    }
```

```
    /*Getting distance and hops from source to destination*/
    for(i=count;i>1;i--)//for finding path
    {
        u=path[i];
        v=path[i-1];
        *sdist+= adj[u][v];
    }
    return (count);
}
/*All outputs are in main function*/
main()
{
    int i,j;
    int source,dest;
    int path[MAX];
    int shortdist,hop;
```

```
    create_graph();
    printf("The adjacency matrix is :\n");
    display();
```

```
    while(1)
    {
        printf("Enter source node(-1 to quit) : ");
        scanf("%d",&source);
        printf("Enter destination node(-1 to quit) : ");
        scanf("%d",&dest);
        printf("Enter destination node(-1 to quit) : ");
        scanf("%d",&hop1);
```

```
        if(source==-1 || dest==-1)
            return 1;
        hop = findpath(source,dest,path,&shortdist);
```

```
        if(shortdist!=0)
        {
            printf("Shortest distance is : %d\n", shortdist);
            printf("Shortest Path is : ");
            for(i=hop;i>1;i--)
                printf("%d->",path[i]);
            printf("%d",path[i]);
            printf("\n");
            printf("hop is : %d\n", hop);
        }
        else
            printf("There is no path from source to destination node\n")
;
    }
}
```