//CODE FOR SERIAL EXECUTION

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

#define MAX 20
#define INFINITY 10000
int n;//Number of nodes
int **adj;//Will contain entire adjacency as read from the file

void define_input();
int breakup(int **,int,int);
int min_no(int *,int);


int main()
{
	int i,j;
	define_input();//function to read input from file
	//printf("define-input done\n");
	//printf("n=%d\n",n);
	int ans=calc(1,n);
	printf("The maximum flow is: %d\n",ans);
	return 0;
}


int calc(int start_index,int end_index)//Shall calculate max flow only for the adjacency matrix  passed as argument
{

int i,j=0;
int temp_n;
//Initialization and matrix loading
temp_n=(end_index-start_index)+1;

int **temp_adj,*temp_h;
temp_adj=malloc(sizeof (int *)*(temp_n+1));
for(i=0;i<=temp_n;i++)
{
	temp_adj[i]=malloc(sizeof (int)*(temp_n+1));
}

temp_h=malloc(sizeof(int)*(temp_n+1));

for(i=1;i<=temp_n;i++)
{
	for(j=1;j<=temp_n;j++)
	{
		temp_adj[i][j]=adj[start_index+i-1][start_index+j-1];
	}

}

for(i=1;i<=temp_n;i++)
	temp_h[i]=breakup(temp_adj,i,temp_n);


int k;//Scaling factor
int delta;//for iterations of each scaling phase
int f[temp_n+1][temp_n+1],res[temp_n+1][temp_n+1],e[temp_n+1];//Adjacency,flow,residual,excess,height matrices
int maxflow=0; //The final solution


for(i=1;i<=temp_n;i++)
{
	e[i]=0;
}
```

```
//Initializing residual matrix
for(i=1;i<=temp_n;i++)
for(j=1;j<=temp_n;j++)
{
	res[i][j]=temp_adj[i][j];//Initializing residual matrix to adjacency matrix
}

//Preflow operation
for(i=1;i<=temp_n;i++)
{
	if(temp_adj[1][i]!=0)
	{
		f[1][i]=temp_adj[1][i];
		e[i]=temp_adj[1][i];//Updation of excess
		e[1]=e[1]-temp_adj[1][i];//updation of excess of source
		//Updation of residual matrix
		res[1][i]=temp_adj[1][i]-f[1][i];
		res[i][1]=f[1][i];
	}
}

//end of preflow operation


//Calculation of k
i=0;
while(1)
{
	if(pow(2,i)>temp_n)
	{
		k=i;
		break;
	}
	else
		i++;
}

//.............WHILE LOOP BEGINS HERE.............


int counter=0;
delta=pow(2,k);//Initializing delta

while(delta>0)//while loop for iterations on different scaling phases
{
counter++;

int an=-1;//active node
int tn=-1;//Node to send the flow to(target node)
int minht=-1;//Needed for relabel operation
int minflow;//Stores the amount of flow to be pushed in each case

//Selection of active node
for(i=1;i<temp_n;i++)
{
	if(e[i]>delta/2)
	{
		if(an==-1)
			an=i;
		else
		{
			if(temp_h[i]<temp_h[an])
				an=i;
		}
	}
}//end of for loop
```

```
if(an==-1)//Go to next scaling phase
{
	//printf("Goes to condition an==-1");
	delta=delta/2;
	continue;
}

//Selection of active node complete
//printf("Active node is: %d \n",an);

//Selection of node to send flow to
for(i=1;i<=temp_n;i++)
{
	if(res[an][i]!=0 && temp_h[an]==temp_h[i]+1)
	{
		tn=i;
		break;
	}

}//end of for loop

//if no such node exists, relabel

//...RELABEL OPERATION....
if(tn==-1)
{

for(i=1;i<=temp_n;i++)//finding a neighbouring node of active node, with minimum height
{
	if(res[an][i]!=0)
	{
		//printf("Edge [%d][%d] exists. \n",an,i);
		if(minht==-1)
		{
			minht=temp_h[i];
			tn=i;
		}
		else
		{
			if(temp_h[i]<minht)
			{
				minht=temp_h[i];
				tn=i;
			}
		}
	}
}//end of for loop

temp_h[an]=minht+1;//Relabeling the height of active node
//printf("Height of node %d relabeled to : %d \n",an,temp_h[an]);

}//end of if


//Relabel operation complete, height of active node increased and also 'tn' contains the node to push the flow to.


//Determining the amt of flow to be pushed
minflow=e[an];
if(res[an][tn]<minflow && res[an][tn]>0)
	minflow=res[an][tn];
if((delta-e[tn])<minflow && (delta-e[tn])>0)
	minflow=delta-e[tn];

//Carrying out the flow and corresponding updations
f[an][tn]=f[an][tn]+minflow;
```

```
if(tn!=temp_n && tn!=1)
{
        e[tn]=e[tn]+minflow; //Increasing flow on dest
}
if(an!=temp_n && an!=1)
{
        e[an]=e[an]-minflow;
}
//printf("e[an] after update is    : %d \n",e[an]);

res[an][tn]=res[an][tn]-minflow;
res[tn][an]=res[tn][an]+minflow;

}//end of outermost while loop
```

//Finding the solution i.e maxflow from final residual matrix

```
for(i=1;i<=temp_n;i++)
{
        if(res[i][1]>0)
                maxflow=maxflow+res[i][1];
}

return maxflow;

}//end of calc().....
```

//.....READING INPUT FROM FILE.....

```
void define_input()
{
const char delim[]=" ";
FILE *fp;

fp=fopen("Graph6","r");

if(fp==NULL)
        printf("Error reading file");


char ch[MAX];
int numberofnodes,fromnode,tonode,capacity;
int i=0,temp1=0;

fgets(ch,10,fp);
while(*(ch+i)!='\n')
{
        i++;
}
char chnew1[i];
strcpy(chnew1,ch);
n=atoi(chnew1);

//Creating an adjacency matrix using the pointer adj and initializing its size
adj=malloc(sizeof (int *)*(n+1));
 for(temp1=0;temp1<=n;temp1++)
{
        adj[temp1]=malloc(sizeof (int)*(n+1));
}

i=0;//Reinitialize variable i.
while(fgets(ch,10,fp)!=NULL)//Loop for getting one line as one string for every iteration
{
```

```
		char chnew[MAX];
		strcpy(chnew,ch);
		//printf("%s\n",chnew);
		i++;

		//Tokenizing the string
		char *t;
		int j=0;
		t=strtok(chnew,delim);
		while(t!=NULL)
		{
			j++;
			if(j==1)
			{fromnode=atoi(t);}
			else if(j==2)
			{tonode=atoi(t);}
			else
			{capacity=atoi(t);}
			t=strtok(NULL,delim);
		}
		adj[fromnode][tonode]=capacity; //presence of an edge in adjacency matrix is indicated by a non zero value, i.e its capacity

}//end of outer while loop
fclose(fp);
}//end of define_input() function

int breakup(int **adj,int node_no,int n)//to find heights of nodes based on the graph
{
int i,t=0;
int temp[n+1];

if(node_no!=1 && node_no!=n)//sink
{
		for(i=1;i<=n;i++)
		{
		if(adj[node_no][i]!=0)
		{
		temp[i]=1+breakup(adj,i,n);
	}
		else
		temp[i]=INFINITY;
		}//end of for loop
return min_no(temp,n);
}//end of if
else if(node_no==n)
		return 0;
else
		return n;

}//end of breakup() function

int min_no(int *temp,int n)
{
int min,loc,c;
min=temp[1];
loc=1;
for(c=1;c<=n;c++)
{

		if(temp[c]<min)
		{
			min=temp[c];
			loc=c;
	}
}
return min;
}//end of min_no() function
```