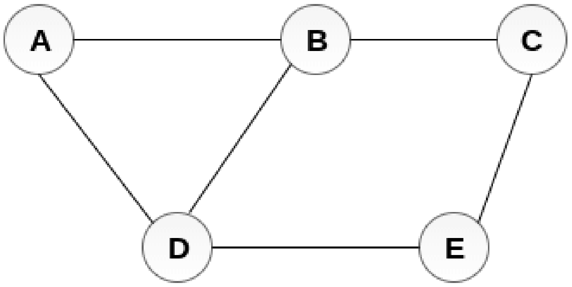
Depth First Search (DFS) and Breadth First Search (BFS)

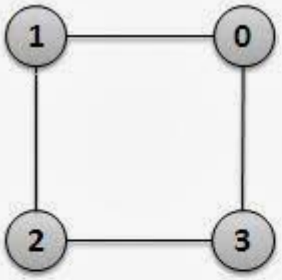
Dr. Rahul Das Gupta



Undirected Graph

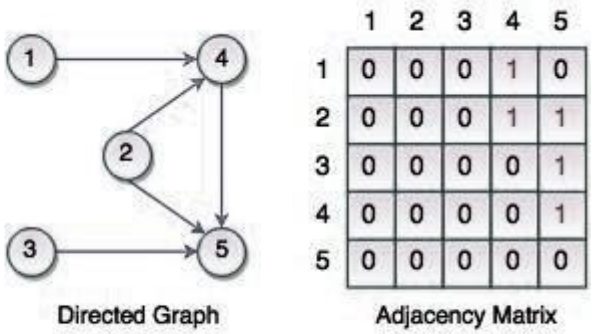
$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 0 & 1 & 0 \\ B & 1 & 0 & 1 & 1 & 0 \\ C & 0 & 1 & 0 & 0 & 1 \\ D & 1 & 1 & 0 & 0 & 1 \\ E & 0 & 0 & 1 & 1 & 0 \end{array}$$

Adjacency Matrix



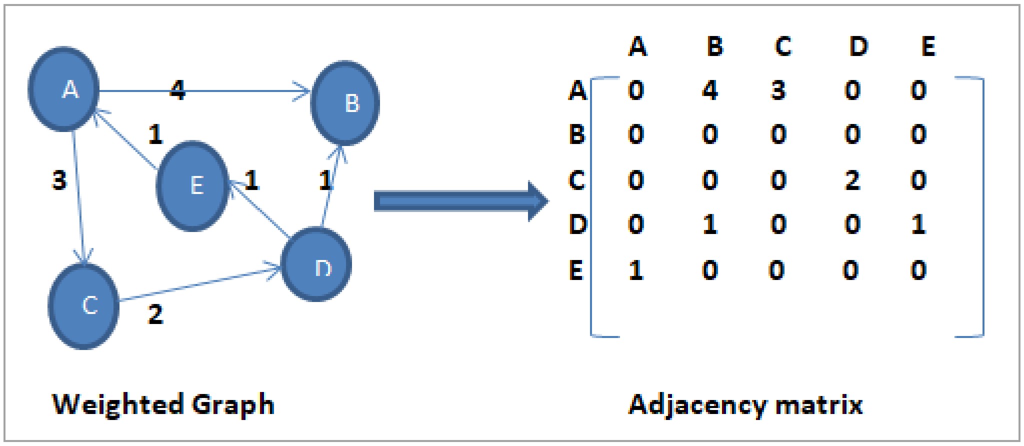
	0	1	2	3
0	0	1	0	1
1	1	0	1	0
2	0	1	0	1
3	1	0	1	0

Adjacency Matrix Representation of Undirected Graph



	1	2	3	4	5
1	0	0	0	1	0
2	0	0	0	1	1
3	0	0	0	0	1
4	0	0	0	0	1
5	0	0	0	0	0

Fig. Adjacency Matrix Representation of Directed Graph



	A	B	C	D	E
A	0	4	3	0	0
B	0	0	0	0	0
C	0	0	0	2	0
D	0	1	0	0	1
E	1	0	0	0	0

```
#include<stdio.h>
#include<stdlib.h>
#define INFINITE 10000
#define EMPTY_ERROR -9999

typedef struct
{
 int n;
 char *V;
 int **wt;
}GRAPH;

typedef struct
{
 int top;
 int *vertex;
}STACK;

typedef struct
{
 int front, rear;
 int *vertex;
}QUEUE;


void input_graph(GRAPH *);
void initialisation_stack(STACK *, int);
void initialisation_queue(QUEUE *, int );
void push(STACK *, int );
void insert_q(QUEUE *, int );
int pop(STACK *);
int delete_q(QUEUE *);
void DFS(GRAPH , int);
void BFS(GRAPH , int);
void free_graph (GRAPH *);

void input_graph(GRAPH *aG)
{
 int i,j;
 char ans;
 printf("\n Enter the no. of vertices :");
 scanf("%d",&aG->n);
 getchar();
 aG->V=(char *)malloc(sizeof(char)*aG->n);
 for(i=0;i<aG->n;i++)
    aG->V[i]='A'+i;
 aG->wt=(int **)malloc(sizeof(int *)*aG->n);
 for(i=0; i<aG->n; i++)
    aG->wt[i]=(int *)malloc(sizeof(int)*aG->n);
/*
 for(i=0; i<aG->n; i++)
    for(j=0; j<aG->n; j++)
    {
       printf("\n Is any edge between %d and %d? Answer (Y/N):",i+1,j+1);
```

```
      scanf("%c",ans);
      if(ans=='y'||ans=='Y')
      {
        printf("\n Enter edge cost between %d and %d :",i+1,j+1);
        scanf("%d",&aG->wt[i][j]);
      }
      else
        aG->wt[i][j]=INFINITE;
    }
*/
aG->wt[0][0]=0;
aG->wt[0][1]=5;
aG->wt[0][2]=5;
aG->wt[0][3]=INFINITE;
aG->wt[0][4]=INFINITE;
aG->wt[1][0]=5;
aG->wt[1][1]=0;
aG->wt[1][2]=INFINITE;
aG->wt[1][3]=5;
aG->wt[1][4]=5;
aG->wt[2][0]=5;
aG->wt[2][1]=INFINITE;
aG->wt[2][2]=0;
aG->wt[2][3]=INFINITE;
aG->wt[2][4]=INFINITE;
aG->wt[3][0]=INFINITE;
aG->wt[3][1]=5;
aG->wt[3][2]=INFINITE;
aG->wt[3][3]=0;
aG->wt[3][4]=INFINITE;
aG->wt[4][0]=INFINITE;
aG->wt[4][1]=5;
aG->wt[4][2]=INFINITE;
aG->wt[4][3]=INFINITE;
aG->wt[4][4]=0;
 }
```

```
void initialisation_stack(STACK *s, int stack_size)
{
 s->top=-1;
 s->vertex=(int *)malloc(sizeof(int)*stack_size);
}
```

```
void initialisation_queue(QUEUE *q, int queue_size)
{
 q->front=-1;
 q->rear=-1;
 q->vertex=(int *)malloc(sizeof(int)*queue_size);
}

void push(STACK *s, int v)
{
    s->vertex[++(s->top)]=v;
}

void insert_q(QUEUE *q, int v)
{
    q->vertex[++(q->rear)]=v;
}


int pop(STACK *s)
{
 if(s->top==-1)
 {
   printf("\n Empty stack...");
   return EMPTY_ERROR;
 }
 return s->vertex[(s->top)--];
}

int delete_q(QUEUE *q)
{
    if(q->front==q->rear)
    {
       printf("\n Empty queue...");
       q->front=-1;
       q->rear=-1;
```

```
      return EMPTY_ERROR;
    }
    else
      return q->vertex[++(q->front)];
}


void DFS(GRAPH G, int v)
{
 STACK stk;
 int *visited;
 int i, p;
 printf("\n");
 initialisation_stack(&stk, G.n); /* Stack Initialisation */
 visited = (int *)malloc(sizeof(int)*G.n);
 for(i=0;i<G.n;i++) /* Mark all the node as unvisited. */
        visited[i]=0;
 visited[v]=1;  /* Mark the Starting node as visited. */
 push(&stk,v); /* Insert the starting vertex in Stack.*/
 while(stk.top!=-1)  /*Continue until Stack is not empty. */
 {
    p=pop(&stk); /*p : the current node just remove from the Stack.*/
    printf("Visited %c  ", G.V[p]);
/*Insert those nodes into Stack which are adjacent to p and not
visited earlier. */
    for (i=0; i<G.n; i++)
       if(G.wt[p][i]!=0 && G.wt[p][i]!=INFINITE && visited[i]==0)
         {
            push(&stk, i);
            visited[i]=1;
         }
 }
free(visited);
printf("\n");
}
```

```
void BFS(GRAPH G, int v)
{
 QUEUE queue;
 int *visited;
 int i, p;
 printf("\n");
 initialisation_queue(&queue, G.n); /* Queue Initialisation */
 visited=(int *)malloc(sizeof(int)*G.n);
 for(i=0; i<G.n; i++) /* Mark all the node as unvisited. */
         visited[i]=0;
 visited[v]=1; /* Mark the Starting node as visited. */
 insert_q(&queue, v); /* Insert the starting vertex in Queue.*/
 while(queue.front != queue.rear)
/*Continue until Queue is not empty. */
 {
     p=delete_q(&queue);
   /*p : the current node just remove from the Queue.*/
     printf("Visited %c  ", G.V[p]);
     //printf("\n %d %d", queue.front, queue.rear);
/*Insert those nodes into Queue which are adjacent to p and not
visited earlier. */
     for (i=0; i<G.n; i++)
        if(G.wt[p][i]!=0 && G.wt[p][i]!=INFINITE && visited[i]==0)
          {
            insert_q(&queue,i);
            visited[i]=1;
          }

 }
 free(visited);
 printf("\n");
}
```

```
void free_graph (GRAPH *G)
{
 int i,j;

 free(G->V);
 for(i=0; i<G->n; i++)
    free(G->wt[i]);
 free(G->wt);
}


void main()
{
 GRAPH G;
 //G=(GRAPH *)malloc(sizeof(GRAPH));
 input_graph(&G);
 DFS(G, 0);
 BFS(G, 0);
 free_graph(&G);
}
```