**Q. WAP for implementation of graph traversals by applying BFS.**

#include<stdio.h>

#include<stdlib.h>

struct queue

{

int size;

int f;

int r;

int\* arr;

};

int isEmpty(struct queue \*q)

{

if(q->r==q->f)

{

return 1;

}

return 0;

}

int isFull(struct queue \*q)

{

if(q->r==q->size-1)

{

return 1;

}

return 0;

}

void enqueue(struct queue \*q, int val)

{

if(isFull(q))

{

printf("This Queue is full\n");

}

else

{

q->r++;

q->arr[q->r] = val;

}

}

int dequeue(struct queue \*q)

{

int a = -1;

if(isEmpty(q))

{

printf("This Queue is empty\n");

}

else

{

q->f++;

a = q->arr[q->f];

}

return a;

}

int main()

{

struct queue q;

q.size = 400;

q.f = q.r = 0;

q.arr = (int\*)malloc(q.size\*sizeof(int));

int node;

int i = 0;

int visited[7] = {0,0,0,0,0,0,0};

int a [7][7] = {

{0,1,0,1,0,0,0},

{1,0,1,1,0,1,1},

{0,1,0,1,1,1,0},

{1,1,1,0,1,0,0},

{0,0,1,1,0,0,1},

{0,1,1,0,0,0,0},

{0,0,0,0,1,0,0}

};

printf("BFS traversal result : ");

printf("\nTraversed : %d", i);

visited[i] = 1;

enqueue(&q, i);

while (!isEmpty(&q))

{

int node = dequeue(&q);

for (int j = 0; j < 7; j++)

{

if(a[node][j] ==1 && visited[j] == 0)

{

printf("\nTraversed : %d", j);

visited[j] = 1;

enqueue(&q, j);

}

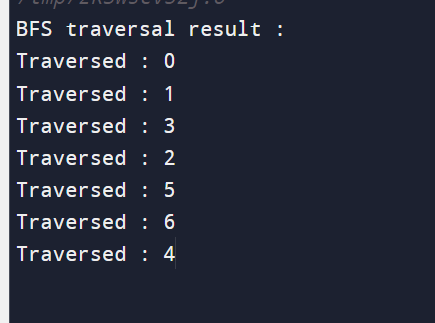
}

}

return 0;

}

**OUTPUT :**

****