广度优先算法的实现

1. 数据结构：图的邻接表，队列
2. 广度优先遍历无向图思路：

**广度优先搜索类似于树的层次遍历过程。它需要借助一个队列来实现**

 1.准备工作：创建一个visited数组，用来记录已被访问过的顶点；创建一个队列，用来存放每一层的顶点；初始化图G。

  2.从图中的v0开始访问，将的visited[v0]数组的值设置为true，同时将v0入队。

3.只要队列不空，则重复如下操作：

    (1)队头顶点u出队。

    (2)依次检查u的所有邻接顶点w，若visited[w]的值为false，则访问w，并将visited[w]置为true，同时将w入队。

3.实验用到的代码如下

#include <stdio.h>

#include <stdint.h>

#include <queue>

using namespace std;

#define MAX\_NODE 26

typedef int InfoType;

typedef int ElemType;

typedef int VRType;

typedef char VertexType;

typedef enum

{

DG = 0,

DN,

UDG,

UDN

}GraphKind;

typedef struct ArcCell

{

VRType adj;

InfoType \*info;

}ArcCell, AdjMatrix[MAX\_NODE][MAX\_NODE];

typedef struct

{

VertexType vexs[MAX\_NODE];

AdjMatrix arcs;

int vexnum;

int arcnum;

GraphKind kind;

}MGraph;

void BuildUDG(MGraph \*G)

{

int i = 0, j = 0, k = 0;

char c = 'a';

int n1[] = {0, 0, 1, 1, 2, 2, 3, 4, 5};

int n2[] = {1, 2, 3 ,4, 5, 6, 7, 7, 6};

if (!G)

return;

G->vexnum = 8;

G->arcnum = 9;

G->kind = UDG;

for (i=0; i<G->vexnum; ++i)

G->vexs[i] = c++;

for (i=0; i<G->vexnum; ++i)

{

for (j=0; j<G->vexnum; ++j)

{

G->arcs[i][j].adj = 0;

G->arcs[i][j].info = NULL;

}

}

for (k=0; k<G->arcnum; ++k)

{

G->arcs[n1[k]][n2[k]].adj = 1;

G->arcs[n2[k]][n1[k]].adj = G->arcs[n1[k]][n2[k]].adj;

}

}

static unsigned char visited[MAX\_NODE] = {0};

int NextAdjVex(MGraph \*G, int v, int w)

{

int i = 0;

if (!G || v < 0)

return -1;

for (i=++w; i<G->vexnum; ++i)

if (G->arcs[v][i].adj != 0 && G->arcs[v][i].adj != INT32\_MAX)

return i;

return -1;

}

void DFS(MGraph \*G, int v)

{

int i = 0;

if (!G)

return;

visited[v] = 1;

printf("%c ", G->vexs[v]);

for (i=NextAdjVex(G, v, -1); i>=0; i=NextAdjVex(G, v, i))

{

if (!visited[i])

DFS(G, i);

}

}

void BFSQueue(MGraph \*G)

{

int i = 0, j = 0, k = 0;

queue<int> s;

if (!G)

return;

for (i=0; i<G->vexnum; ++i)

visited[i] = 0;

for (i=0; i<G->vexnum; ++i)

{

if (!visited[i])

{

visited[i] = 1;

printf("%c ", G->vexs[i]);

s.push(i);

while (!s.empty())

{

j = s.front();

s.pop();

for (k=NextAdjVex(G, j, -1); k>=0; k=NextAdjVex(G, j, k))

{

if (!visited[k])

{

visited[k] = 1;

printf("%c ", G->vexs[k]);

s.push(k);

}

}

}

}

}

printf("\n");

}

int main()

{

MGraph G = {0};

BuildUDG(&G);

BFSQueue(&G);

getchar();

}