图的遍历

张晓平

November 28, 2016

定义 1. 从图中某一顶点出发访遍图中其余顶点，且使每一个顶点仅被访问一次，这一过程叫做图的遍历*(Traversing Graph)*。

图的遍历分为两种：

- 深度优先遍历
- 广度优先遍历

1 深度优先遍历(Depth First Search, DFS)

定义 2 (连通图的深度优先遍历)**.** 从图中某个顶点v出发，访问此顶点，然后从v的未被访问的邻接点出发深度优先遍历图，直到图中所有与v有路径相同的顶点都被访问到。

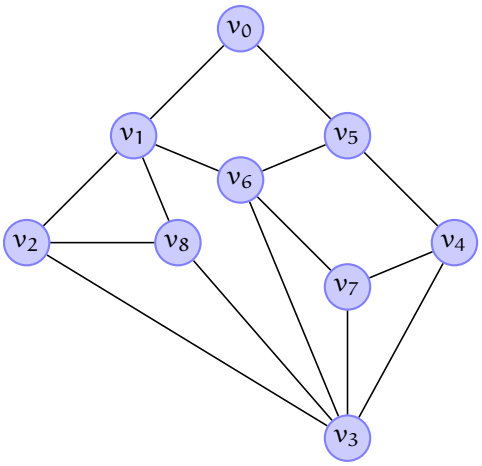

Figure 1: DFS示意图

深度优先搜索遍历类似于树的先序遍历。假定给定图G的初态是所有顶点均未被访问过，在G中任选一个顶点v作为遍历的初始点，则深度优先搜索的过程可通过栈来实现（入栈节点将被访问），具体过程如下：

首先让v入栈，然后在其邻接点中选择一个节点w，接着进行深度优先搜索。搜索过程中如果遇到一个节点u，其邻接点均被访问过，则从栈中弹出该节点u。搜索将在栈空时结束。

DFS看似复杂，其实可以很简单地递归实现。

定义 3 (非连通图的深度优先遍历)**.** 只需要对它的连通分量分别进行深度优先遍历，即在先前一个顶点进行一次深度优先遍历后，若图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作为起始点，重复上述过程，直到图中所有顶点都被访问到为止。

DFSTraverse.c (adjacent matrix)

```
#include "adjmatrix.h"
void DFS(Graph G, int i)
{
  int j;
```

Table 1: 图(1)深度优先搜索的栈操作

栈底→栈顶	弹出元素	打印元素
v_0		v_0
v_0, v_1		v_1
v_0, v_1, v_2		v_2
v_0, v_1, v_2, v_3		v_3
v_0, v_1, v_2, v_3, v_8		v_8
v_0, v_1, v_2, v_3	v_8	
v_0, v_1, v_2, v_3, v_6		v_6
$v_0, v_1, v_2, v_3, v_6, v_5$		v_5
$v_0, v_1, v_2, v_3, v_6, v_5, v_4$		v_4
$v_0, v_1, v_2, v_3, v_6, v_5, v_4, v_7$		v_7
$v_0, v_1, v_2, v_3, v_6, v_5, v_4$	v_7	
$v_0, v_1, v_2, v_3, v_6, v_5$	v_4	
v_0, v_1, v_2, v_3, v_6	v_5	
v_0, v_1, v_2, v_3	v_6	
v_0, v_1, v_2	v_3	
v_0, v_1	v_2	
v_0	v_1	
栈空	v_0	

```
  visited[i] = TRUE;
  printf("%c␣", G.v[i]);
  for(j = 0; j < G.numVertices; j++)
    if(G.e[i][j] == 1 && !visited[j])
      DFS(G,j);
}

void DFSTraverse(Graph G)
{
  int i;
  for(i = 0; i < G.numVertices; i++)
    visited[i] = FALSE;
  for(i = 0; i < G.numVertices; i++)
    if(!visited[i])
      DFS(G, i);
}
```

2 广度优先遍历(Breadth First Search, BFS)

广度优先搜索的过程如下：

先访问节点v，并标记它已被访问，然后访问v的所有邻接点，这些节点访问之后，接着访问第一个邻接点的所有邻接点。为实现广度优先搜索，每次将当前节点入列保存，在处理完所有邻接点之后，出列一个节点，然后处理该节点的所有邻接点，每个邻接点如未被访问则访问后入队列，已访问过的节点忽略，直到队列为空。

BFSTraverse.c (adjacent matrix)

```
#include "adjmatrix.h"
void BFSTraverse(Graph G)
{
  int i, j;
  Queue Q;
  for (i = 0; i < G.numVertices; i++)
    visited[i] = FALSE;
  InitQueue(&Q);
  for (i = 0; i < G.numVertices; i++) {
    if (!visited[i]) {
```

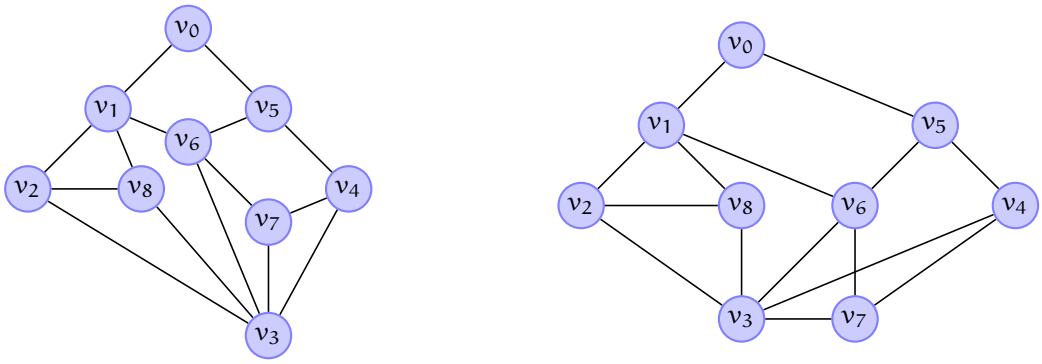

Figure 2: BFS示意图

Table 2: 图(2)广度优先搜索的队列操作

队头←队尾	出队列元素（打印元素）
v_0 v_1, v_5	v_0
v_5 v_5, v_2, v_6, v_8	v_1
v_2, v_6, v_8 v_2, v_6, v_8, v_4	v_5
v_6, v_8, v_4 v_6, v_8, v_4, v_3	v_2
v_8, v_4, v_3 v_8, v_4, v_3, v_7	v_6
v_4, v_3, v_7	v_8
v_3, v_7	v_4
v_7	v_3
队列空	v_7

```
      visited[i] = TRUE;
      printf("%c␣", G.v[i]);
      EnQueue(&Q, i);
      while (!QueueEmpty(Q)) {
        DeQueue(&Q, &i);
        for(j = 0; j < G.numVertices; j++) {
          if(G.e[i][j] == 1 && !visited[j]) {
            visited[j] = TRUE;
            printf("%c␣", G.v[j]);
            EnQueue(&Q, j);
          }
        }
      }
    }
  }
}
```