

深度优先搜索（DFS、深搜）和广度优先搜索（BFS、广搜）

前边介绍了有关图 的 4 种存储方式，本节介绍如何对存储的图中的顶点进行遍历。常用的遍历方式有两种：深度优先搜索和广度优先搜索。

深度优先搜索（简称“深搜”或DFS）

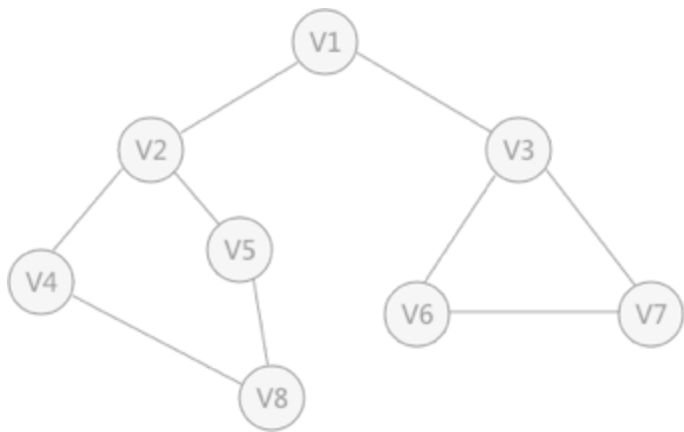


图 1 无向图

深度优先搜索的过程类似于树的先序遍历，首先从例子中体会深度优先搜索。例如图 1 是一个无向图，采用深度优先算法遍历这个图的过程为：

1. 首先任意找一个未被遍历过的顶点，例如从 V1 开始，由于 V1 率先访问过了，所以，需要标记 V1 的状态为访问过；
2. 然后遍历 V1 的邻接点，例如访问 V2，并做标记，然后访问 V2 的邻接点，例如 V4（做标记），然后 V8，然后 V5；
3. 当继续遍历 V5 的邻接点时，根据之前做的标记显示，所有邻接点都被访问过了。此时，从 V5 回退到 V8，看 V8 是否有未被访问过的邻接点，如果没有，继续回退到 V4，V2，V1；
4. 通过查看 V1，找到一个未被访问过的顶点 V3，继续遍历，然后访问 V3 邻接点 V6，然后 V7；
5. 由于 V7 没有未被访问的邻接点，所有回退到 V6，继续回退至 V3，最后到达 V1，发现没有未被访问的；
6. 最后一步需要判断是否所有顶点都被访问，如果还有没被访问的，以未被访问的顶点为第一个顶点，继续依照上边的方式进行遍历。

根据上边的过程，可以得到图 1 通过深度优先搜索获得的顶点的遍历次序为：

```
V1 -> V2 -> V4 -> V8 -> V5 -> V3 -> V6 -> V7
```

所谓深度优先搜索，是从图中的一个顶点出发，每次遍历当前访问顶点的临界点，一直到访问的顶点没有未被访问过的临界点为止。然后采用依次回退的方式，查看来的路上每一个顶点是否有其它未被访问的临界点。访问完成后，判断图中的顶点是否已经全部遍历完成，如果没有，以未访问的顶点为起始点，重复上述过程。

深度优先搜索是一个不断回溯的过程。

采用深度优先搜索算法遍历图的实现代码为：

```
01.  #include <stdio.h>
02.
03.  #define MAX_VERTEX_NUM 20 //顶点的最大个数
04.  #define VRType int //表示顶点之间的关系的变量类型
05.  #define InfoType char //存储弧或者边额外信息的指针变量类型
06.  #define VertexType int //图中顶点的数据类型
07.
08.  typedef enum{false,true}bool; //定义bool型常量
09.  bool visited[MAX_VERTEX_NUM]; //设置全局数组，记录标记顶点是否被访问过
10.
11.  typedef struct {
12.      VRType adj; //对于无权图，用 1 或 0 表示是否相邻；对于带权图，直
13.      InfoType * info; //弧或边额外含有的信息指针
14.  }ArcCell,AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
15.
16.  typedef struct {
17.      VertexType vexs[MAX_VERTEX_NUM]; //存储图中顶点数据
18.      AdjMatrix arcs; //二维数组，记录顶点之间的关系
19.      int vexnum,arcnum; //记录图的顶点数和弧（边）数
20.  }MGraph;
21.  //根据顶点本身数据，判断出顶点在二维数组中的位置
22.  int LocateVex(MGraph * G,VertexType v){
23.      int i=0;
24.      //遍历一维数组，找到变量v
25.      for (; i<G->vexnum; i++) {
26.          if (G->vexs[i]==v) {
27.              break;
28.          }
29.      }
30.      //如果找不到，输出提示语句，返回-1
31.      if (i>G->vexnum) {
32.          printf("no such vertex.\n");
33.          return -1;
34.      }
35.      return i;
36.  }
```

```

37. //构造无向图
38. void CreateDN(MGraph *G) {
39.     scanf("%d,%d",&(G->vexnum), &(G->arcnum));
40.     for (int i=0; i<G->vexnum; i++) {
41.         scanf("%d",&(G->vexs[i]));
42.     }
43.     for (int i=0; i<G->vexnum; i++) {
44.         for (int j=0; j<G->vexnum; j++) {
45.             G->arcs[i][j].adj=0;
46.             G->arcs[i][j].info=NULL;
47.         }
48.     }
49.     for (int i=0; i<G->arcnum; i++) {
50.         int v1,v2;
51.         scanf("%d,%d",&v1,&v2);
52.         int n=LocateVex(G, v1);
53.         int m=LocateVex(G, v2);
54.         if (m==-1 || n==-1) {
55.             printf("no this vertex\n");
56.             return;
57.         }
58.         G->arcs[n][m].adj=1;
59.         G->arcs[m][n].adj=1; //无向图的二阶矩阵沿主对角线对称
60.     }
61. }
62.
63. int FirstAdjVex(MGraph G,int v)
64. {
65.     //查找与数组下标为v的顶点之间有边的顶点，返回它在数组中的下标
66.     for(int i = 0; i<G.vexnum; i++){
67.         if( G.arcs[v][i].adj ){
68.             return i;
69.         }
70.     }
71.     return -1;
72. }
73. int NextAdjVex(MGraph G,int v,int w)
74. {
75.     //从前一个访问位置w的下一个位置开始，查找之间有边的顶点
76.     for(int i = w+1; i<G.vexnum; i++){
77.         if(G.arcs[v][i].adj){
78.             return i;
79.         }
80.     }
81.     return -1;
82. }
83. void visitVex(MGraph G, int v){

```

```

84.     printf("%d ",G.vexs[v]);
85. }
86. void DFS(MGraph G,int v){
87.     visited[v] = true;//标记为true
88.     visitVex( G,  v); //访问第v 个顶点
89.     //从该顶点的第一个边开始，一直到最后一个边，对处于边另一端的顶点调用DFS函数
90.     for(int w = FirstAdjVex(G,v); w>=0; w = NextAdjVex(G,v,w)){
91.         //如果该顶点的标记位false，证明未被访问，调用深度优先搜索函数
92.         if(!visited[w]){
93.             DFS(G,w);
94.         }
95.     }
96. }
97. //深度优先搜索
98. void DFSTraverse(MGraph G){//
99.     int v;
100.    //将用做标记的visit数组初始化为false
101.    for( v = 0; v < G.vexnum; ++v){
102.        visited[v] = false;
103.    }
104.    //对于每个标记为false的顶点调用深度优先搜索函数
105.    for( v = 0; v < G.vexnum; v++){
106.        //如果该顶点的标记位为false，则调用深度优先搜索函数
107.        if(!visited[v]){
108.            DFS( G, v);
109.        }
110.    }
111. }
112.
113. int main() {
114.     MGraph G;//建立一个图的变量
115.     CreateDN(&G);//初始化图
116.     DFSTraverse(G);//深度优先搜索图
117.     return 0;
118. }

```

以图 1 为例，运行结果为：

8,9

1

2

3

4

5

6

7

```
8
1,2
2,4
2,5
4,8
5,8
1,3
3,6
6,7
7,3
1 2 4 8 5 3 6 7
```

广度优先搜索

广度优先搜索类似于树的层次遍历。从图中的某一顶点出发，遍历每一个顶点时，依次遍历其所有的邻接点，然后再从这些邻接点出发，同样依次访问它们的邻接点。按照此过程，直到图中所有被访问过的顶点的邻接点都被访问到。

最后还需要做的操作就是查看图中是否存在尚未被访问的顶点，若有，则以该顶点为起始点，重复上述遍历的过程。

还拿图 1 中的无向图为例，假设 V1 作为起始点，遍历其所有的邻接点 V2 和 V3，以 V2 为起始点，访问邻接点 V4 和 V5，以 V3 为起始点，访问邻接点 V6、V7，以 V4 为起始点访问 V8，以 V5 为起始点，由于 V5 所有的起始点已经全部被访问，所有直接略过，V6 和 V7 也是如此。

以 V1 为起始点的遍历过程结束后，判断图中是否还有未被访问的点，由于图 1 中没有了，所以整个图遍历结束。遍历顶点的顺序为：

```
V1 -> V2 -> v3 -> V4 -> V5 -> V6 -> V7 -> V8
```

广度优先搜索的实现需要借助[队列](#)这一特殊数据结构，实现代码为：

```
01.  #include <stdio.h>
02.  #include <stdlib.h>
03.  #define MAX_VERTEX_NUM 20 //顶点的最大个数
04.  #define VRType int //表示顶点之间的关系的变量类型
05.  #define InfoType char //存储弧或者边额外信息的指针变量类型
06.  #define VertexType int //图中顶点的数据类型
07.  typedef enum{false,true}bool; //定义bool型常量
08.  bool visited[MAX_VERTEX_NUM]; //设置全局数组，记录标记顶点是否被访问过
09.  typedef struct Queue{
10.      VertexType data;
11.      struct Queue * next;
```

```

12. }Queue;
13. typedef struct {
14.     VRType adj; //对于无权图, 用 1 或 0 表示是否相邻; 对于带权图, 直
15.     InfoType * info; //弧或边额外含有的信息指针
16. }ArcCell, AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
17.
18. typedef struct {
19.     VertexType vexs[MAX_VERTEX_NUM]; //存储图中顶点数据
20.     AdjMatrix arcs; //二维数组, 记录顶点之间的关系
21.     int vexnum, arcnum; //记录图的顶点数和弧(边)数
22. }MGraph;
23. //根据顶点本身数据, 判断出顶点在二维数组中的位置
24. int LocateVex(MGraph * G, VertexType v) {
25.     int i=0;
26.     //遍历一维数组, 找到变量v
27.     for (; i<G->vexnum; i++) {
28.         if (G->vexs[i]==v) {
29.             break;
30.         }
31.     }
32.     //如果找不到, 输出提示语句, 返回-1
33.     if (i>G->vexnum) {
34.         printf("no such vertex.\n");
35.         return -1;
36.     }
37.     return i;
38. }
39. //构造无向图
40. void CreateDN(MGraph *G) {
41.     scanf("%d,%d",&(G->vexnum), &(G->arcnum));
42.     for (int i=0; i<G->vexnum; i++) {
43.         scanf("%d",&(G->vexs[i]));
44.     }
45.     for (int i=0; i<G->vexnum; i++) {
46.         for (int j=0; j<G->vexnum; j++) {
47.             G->arcs[i][j].adj=0;
48.             G->arcs[i][j].info=NULL;
49.         }
50.     }
51.     for (int i=0; i<G->arcnum; i++) {
52.         int v1,v2;
53.         scanf("%d,%d",&v1,&v2);
54.         int n=LocateVex(G, v1);
55.         int m=LocateVex(G, v2);
56.         if (m==-1 || n==-1) {
57.             printf("no this vertex\n");
58.             return;

```

```

59.         }
60.         G->arcs[n][m].adj=1;
61.         G->arcs[m][n].adj=1;//无向图的二阶矩阵沿主对角线对称
62.     }
63. }
64.
65. int FirstAdjVex(MGraph G,int v)
66. {
67.     //查找与数组下标为v的顶点之间有边的顶点，返回它在数组中的下标
68.     for(int i = 0; i<G.vexnum; i++){
69.         if( G.arcs[v][i].adj ){
70.             return i;
71.         }
72.     }
73.     return -1;
74. }
75. int NextAdjVex(MGraph G,int v,int w)
76. {
77.     //从前一个访问位置w的下一个位置开始，查找之间有边的顶点
78.     for(int i = w+1; i<G.vexnum; i++){
79.         if(G.arcs[v][i].adj){
80.             return i;
81.         }
82.     }
83.     return -1;
84. }
85. //操作顶点的函数
86. void visitVex(MGraph G, int v){
87.     printf("%d ",G.vexs[v]);
88. }
89. //初始化队列
90. void InitQueue(Queue ** Q){
91.     (*Q)=(Queue*)malloc(sizeof(Queue));
92.     (*Q)->next=NULL;
93. }
94. //顶点元素v进队列
95. void EnQueue(Queue **Q,VertexType v){
96.     Queue * element=(Queue*)malloc(sizeof(Queue));
97.     element->data=v;
98.     Queue * temp=(*Q);
99.     while (temp->next!=NULL) {
100.         temp=temp->next;
101.     }
102.     temp->next=element;
103. }
104. //队头元素出队列
105. void DeQueue(Queue **Q,int *u){

```

```

106.     (*u)=(*Q)->next->data;
107.     (*Q)->next=(*Q)->next->next;
108. }
109. //判断队列是否为空
110. bool QueueEmpty(Queue *Q){
111.     if (Q->next==NULL) {
112.         return true;
113.     }
114.     return false;
115. }
116. //广度优先搜索
117. void BFSTraverse(MGraph G){//
118.     int v;
119.     //将用做标记的visit数组初始化为false
120.     for( v = 0; v < G.vexnum; ++v){
121.         visited[v] = false;
122.     }
123.     //对于每个标记为false的顶点调用深度优先搜索函数
124.     Queue * Q;
125.     InitQueue(&Q);
126.     for( v = 0; v < G.vexnum; v++){
127.         if(!visited[v]){
128.             visited[v]=true;
129.             visitVex(G, v);
130.             EnQueue(&Q, G.vexs[v]);
131.             while (!QueueEmpty(Q)) {
132.                 int u;
133.                 DeQueue(&Q, &u);
134.                 u=LocateVex(&G, u);
135.                 for (int w=FirstAdjVex(G, u); w>=0; w=NextAdjVex(G, u, w)) {
136.                     if (!visited[w]) {
137.                         visited[w]=true;
138.                         visitVex(G, w);
139.                         EnQueue(&Q, G.vexs[w]);
140.                     }
141.                 }
142.             }
143.         }
144.     }
145. }
146. int main() {
147.     MGraph G;//建立一个图的变量
148.     CreateDN(&G);//初始化图
149.     BFSTraverse(G);//广度优先搜索图
150.     return 0;
151. }

```


例如，使用上述程序代码遍历图 1 中的无向图，运行结果为：

```
8,9
1
2
3
4
5
6
7
8
1,2
2,4
2,5
4,8
5,8
1,3
3,6
6,7
7,3
1 2 3 4 5 6 7 8
```

总结

本节介绍了两种遍历图的方式：深度优先搜索算法和广度优先搜索算法。深度优先搜索算法的实现运用的主要是回溯法，类似于树的先序遍历算法。广度优先搜索算法借助队列的先进先出的特点，类似于树的层次遍历。

推荐阅读

图的深度优先搜索和广度优先搜索	图文并茂的介绍了两种搜索算法的具体实现过程
深度优先搜索（DNS）和广度优先搜索（BFS）	详细介绍了两种搜索算法的实现过程
图的遍历算法（深度优先算法DFS和广度优先算法BFS）	详细介绍了两种算法的实现过程，并配备的 C++ 的实现代码
广度优先搜索（BFS）和深度优先搜索（DFS）的应用实例	从实例出发介绍两种搜索算法

