2.4 图的存储之广度优先遍历_物联网 / 嵌入式工程师 - 慕课网

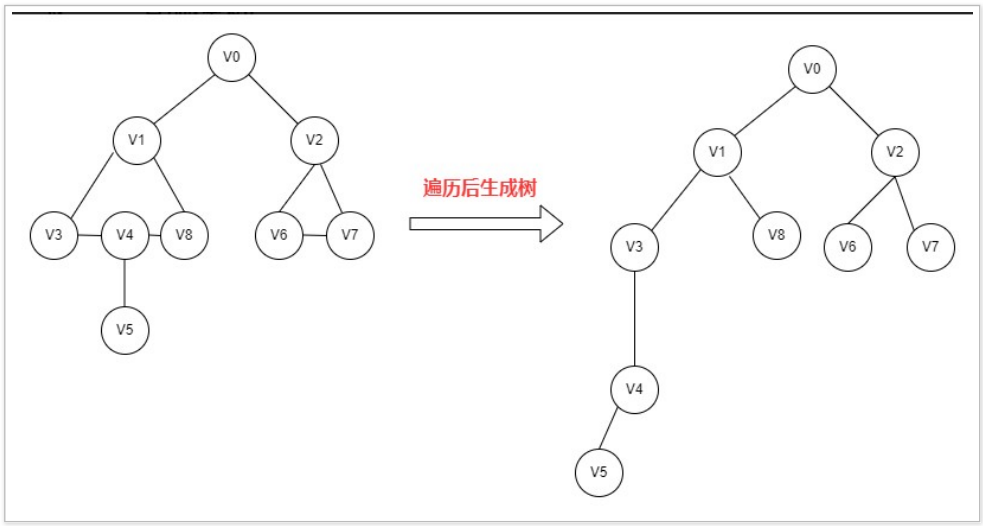
> 慕课网慕课教程 2.4 图的存储之广度优先遍历涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

4. 图的存储之广度优先遍历

我们看了我们的深度优先遍历，我们知道，我们的深度优先遍历类似于我们指定规则的树的先序遍历。而我们的广度优先遍历类于我们的 **** 树的层次遍历。

类似树的层次遍历。初始时，图中各顶点均未被访问，从图中某顶点（设 V0）出发，访问 V0，并依次访问 V0 的各邻接点（广度优先）。然后，分别从这些被访问过的顶点出发，仍按照广度优先的策略搜索其它顶点，……，直到能访问的顶点都访问完毕为止。

为控制广度优先的正确搜索，要用到队列技术，即访问完一个顶点后，让该顶点的序号进队。然后取相应队头（出队），考察访问过的顶点的各邻接点，将未访问过的邻接点访问后再依次进队，……，直到队空为止。



广度优先遍历结果为：V0–>V1–>V2–>V3–>V8–>V6–>V7–>V4–>V8

```
void BFS(graph_t *g,int v)
{
        int t = 0,i = 0;


        linkqueue_t *q = create_empty_linkqueue();
        if(NULL == q)
        {
                printf("malloc si fail\n");
                return ;
        }

        visited[v] = 1;

        enter_linkqueue(q,v);

        while(!is_empty_linkqueue(q))
        {

                t = delete_linkqueue(q);
                printf("V%-3d->",t);
```

```
                for(i = 0;i < N;i++)
                {

                    if(g->maxtrix[t][i] == 1 && visited[i] == 0)
                    {
                        visited[i] = 1;
                        enter_linkqueue(q,i);
                    }
                }
        }

        return ;
}

int main()
{
        graph_t *g = NULL;

        g = create_graph();

        input_edge(g);

        print_matrix(g);


        BFS(g,0);
        putchar('\n');
        return 0;
}
```

linkqueue.h

```
#ifndef __LINKSTACH_H__
#define __LINKSTACH_H__

#include <stdio.h>
#include <string.h>
#include <stdlib.h>


typedef int data_t;

typedef struct node
{
        data_t data;
        struct node *next;
}linknode_t;


typedef struct
{
        linknode_t *front;
        linknode_t *rear;
}linkqueue_t;


extern linkqueue_t *create_empty_linkqueue();


extern int is_empty_linkqueue(linkqueue_t *q);


void enter_linkqueue(linkqueue_t *q,data_t data);


data_t delete_linkqueue(linkqueue_t *q);

#endif
```

linkqueue.c

```
#include "linkqueue.h"


linkqueue_t *create_empty_linkqueue()
{
        linkqueue_t *q = NULL;
        linknode_t *head = NULL;


        head = (linknode_t *)malloc(sizeof(linknode_t));
        head->next = NULL;
```

```
        q = (linkqueue_t *)malloc(sizeof(linkqueue_t));
        q->front = q->rear = head;


        return q;
}


int is_empty_linkqueue(linkqueue_t *q)
{
        return q->front == q->rear ? 1 : 0;
}


void enter_linkqueue(linkqueue_t *q,data_t data)
{
        linknode_t *temp = NULL;
        temp = (linknode_t *)malloc(sizeof(linknode_t));

        temp->data = data;

        temp->next = q->rear->next;
        q->rear->next = temp;


        q->rear = temp;
        return ;
}


data_t delete_linkqueue(linkqueue_t *q)
{
        linknode_t *temp = NULL;
        data_t data;


        temp = q->front->next;
        data = temp->data;


        q->front->next = temp->next;
        free(temp);
        temp = NULL;


        if(q->front->next == NULL)
        {
                q->rear = q->front;
        }

        return data;
}
```

graph.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "linkqueue.h"

typedef  int vertex_t;

#define N 9

typedef struct
{

        vertex_t v[N];
        int maxtrix[N][N];
}graph_t;

int visited[N];




graph_t *create_graph()
{
        graph_t *g = NULL;
        int i = 0;
```

```
        g = (graph_t *)malloc(sizeof(graph_t));
        memset(g,0,sizeof(graph_t));


        for(i = 0;i < N;i++)
        {
                g->v[i] = i;
        }

        return g;
}


void input_edge(graph_t *g)
{
        int i = 0,j = 0;

        printf("please input link (v0,v1) (v0,v2) ...\n");



        while(scanf("(V%d,V%d)",&i,&j) == 2)
        {
                g->maxtrix[i][j] = g->maxtrix[j][i] = 1;
                getchar();
        }


        while(getchar() != '\n');
}


void print_matrix(graph_t *g)
{
        int i = 0,j = 0;
        printf("%3c",' ');

        for(i = 0;i < N;i++)
        {
                printf("V%-2d",i);
        }
        putchar('\n');

        for(i = 0;i < N;i++)
        {
                printf("V%-2d",i);
                for(j = 0;j < N;j++)
                {
                        printf("%-3d",g->maxtrix[i][j]);
                }
                putchar('\n');
        }

        return ;
}


int first_adj(graph_t *g,int v)
{
        int i = 0;

        for(i = 0;i < N;i++)
        {
                if(g->maxtrix[v][i] != 0)
                        return i;
        }

        return -1;
}

int next_adj(graph_t *g,int v,int u)
{
        int i = 0;

        for(i =u + 1;i < N;i++)
        {
                if(g->maxtrix[v][i] != 0)
                        return i;
        }
        return -1;
}




void DFS(graph_t *g,int v)
{
```

```
        int u = 0;

        printf("V%-d->",v);


        visited[v] = 1;


        u = first_adj(g,v);



        while(u >= 0)
        {

                if(visited[u] == 0)
                {
                        DFS(g,u);
                }


                u = next_adj(g,v,u);
        }
        return ;
}

void BFS(graph_t *g,int v)
{
        int t = 0,i = 0;


        linkqueue_t *q = create_empty_linkqueue();
        if(NULL == q)
        {
                printf("malloc si fail\n");
                return ;
        }

        visited[v] = 1;

        enter_linkqueue(q,v);

        while(!is_empty_linkqueue(q))
        {

                t = delete_linkqueue(q);
                printf("V%-3d->",t);

                for(i = 0;i < N;i++)
                {

                        if(g->maxtrix[t][i] == 1 && visited[i] == 0)
                        {
                                visited[i] = 1;
                                enter_linkqueue(q,i);
                        }
                }
        }

        return ;
}

int main()
{
        graph_t *g = NULL;

        g = create_graph();

        input_edge(g);

        print_matrix(g);


        BFS(g,0);
        putchar('\n');
        return 0;
}
```

运行结果：

```
please input link (v0,v1) (v0,v2) ...
(V0,V1) (V0,V2) (V1,V3) (V1,V5) (V3,V4) (V4,V8) (V4,V5) (V0,V2) (V2,V6) (V6,V7) (V2,V7)
```

```
   V0 V1 V2 V3 V4 V5 V6 V7 V8
V0 0  1  1  0  0  0  0  0  0
V1 1  0  0  1  0  0  0  0  1
V2 1  0  0  0  0  0  1  1  0
V3 0  1  0  0  1  0  0  0  0
V4 0  0  0  1  0  1  0  0  1
V5 0  0  0  0  1  0  0  0  0
V6 0  0  1  0  0  0  0  1  0
V7 0  0  1  0  0  0  1  0  0
V8 0  1  0  0  1  0  0  0  0
V0  ->V1  ->V2  ->V3  ->V5  ->V6  ->V7  ->V4  ->V8  ->
```

全文完