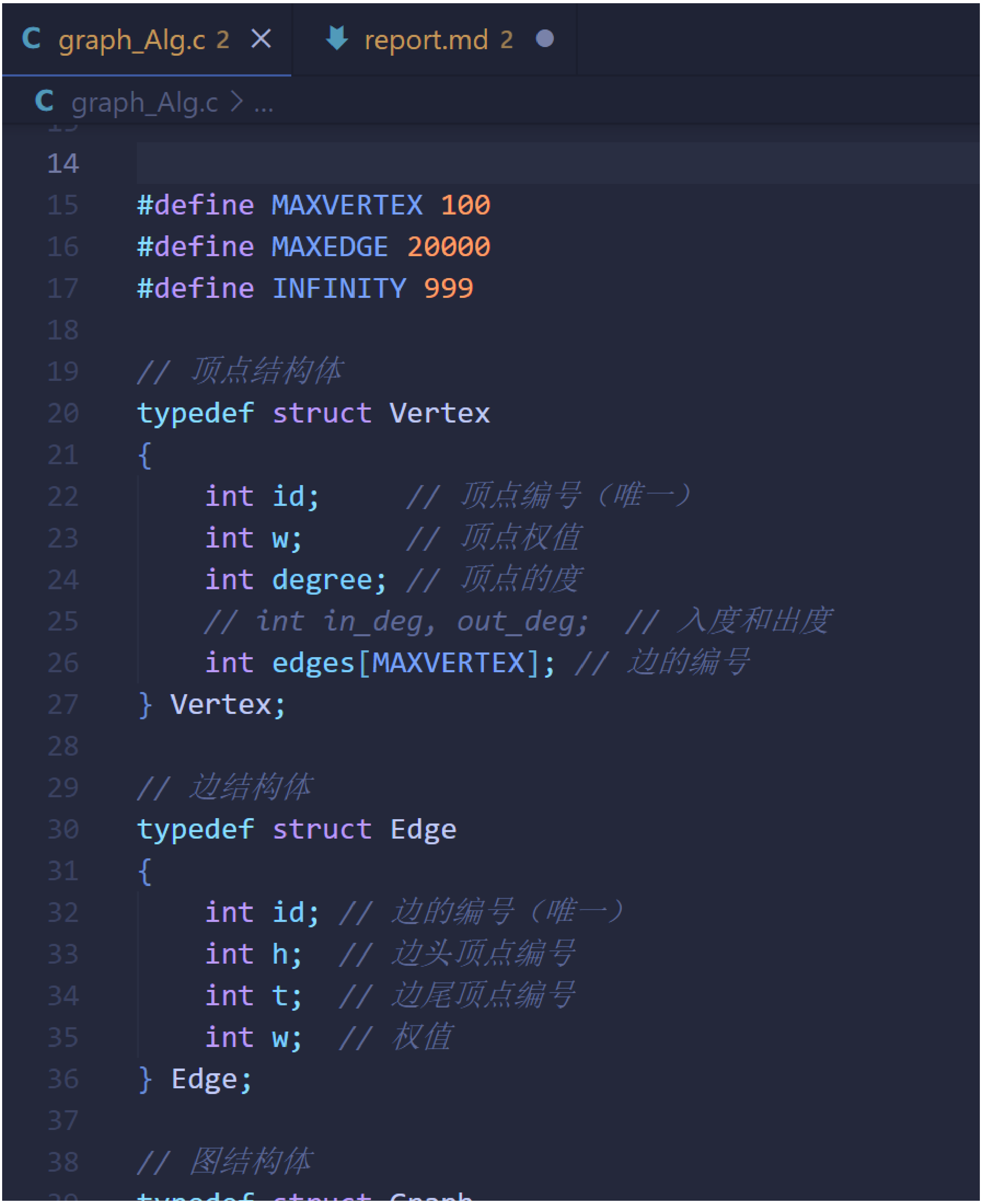
Lab 8 Report

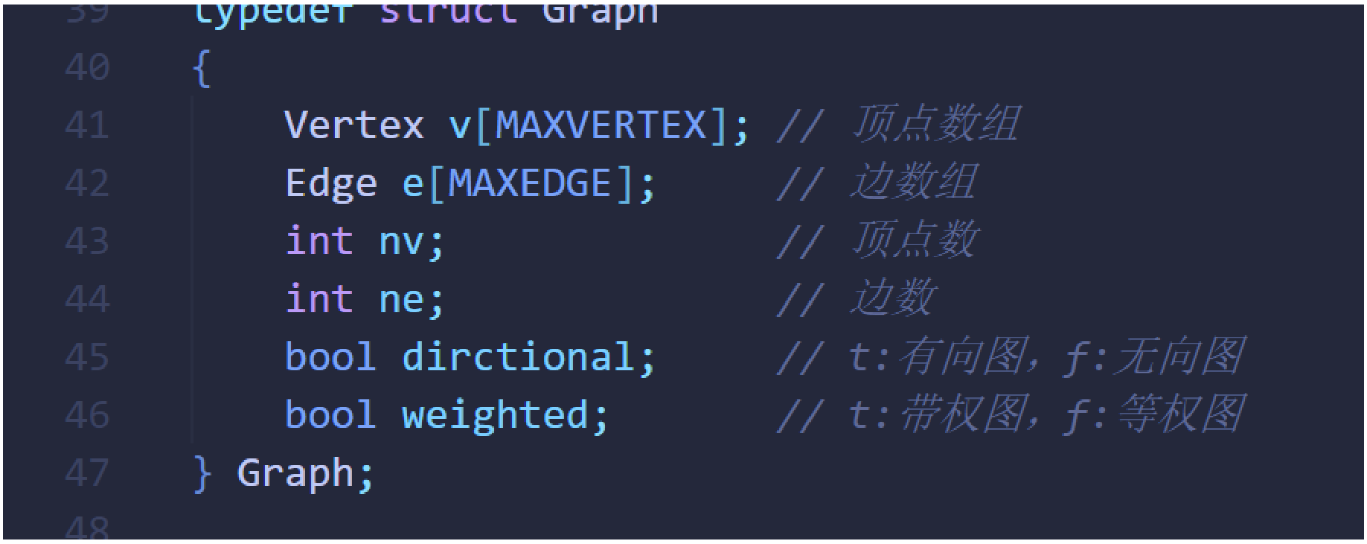
基于图实现的算法

PB22111599 杨映川

静态结构体



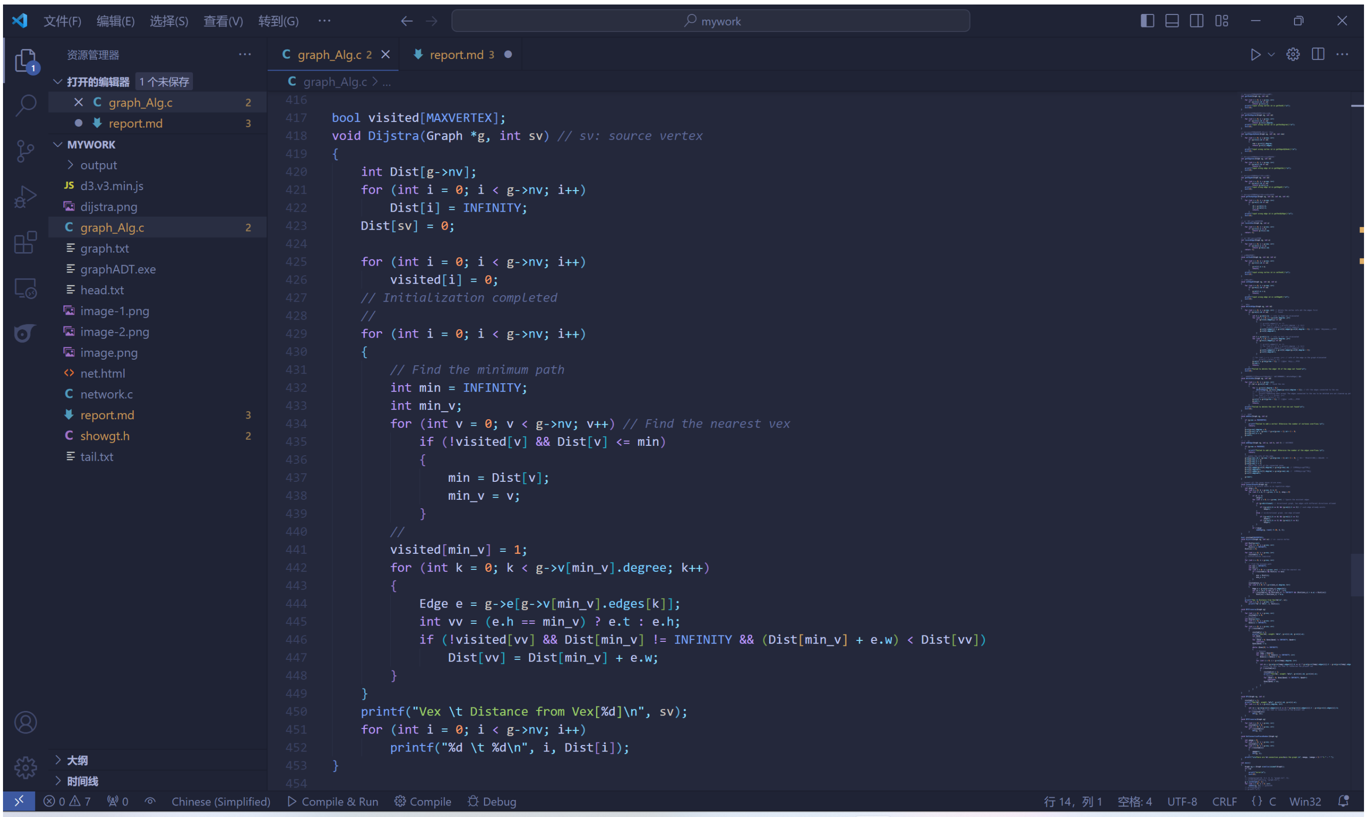
```c

#define MAXVERTEX 100
#define MAXEDGE 20000
#define INFINITY 999

// 顶点结构体
typedef struct Vertex
{
    int id;     // 顶点编号（唯一）
    int w;      // 顶点权值
    int degree; // 顶点的度
    // int in_deg, out_deg;  // 入度和出度
    int edges[MAXVERTEX]; // 边的编号
} Vertex;

// 边结构体
typedef struct Edge
{
    int id; // 边的编号（唯一）
    int h;  // 边头顶点编号
    int t;  // 边尾顶点编号
    int w;  // 权值
} Edge;

// 图结构体
typedef struct Graph
```



```c
typedef struct Graph
{
    Vertex v[MAXVERTEX]; // 顶点数组
    Edge e[MAXEDGE];     // 边数组
    int nv;              // 顶点数
    int ne;              // 边数
    bool dirctional;     // t:有向图, f:无向图
    bool weighted;       // t:带权图, f:等权图
} Graph;
```

Dijstra算法求最短路径

代码



```c
416
417 bool visited[MAXVERTEX];
418 void Dijstra(Graph *g, int sv) // sv: source vertex
419 {
420     int Dist[g->nv];
421     for (int i = 0; i < g->nv; i++)
422         Dist[i] = INFINITY;
423     Dist[sv] = 0;
424
425     for (int i = 0; i < g->nv; i++)
426         visited[i] = 0;
427     // Initialization completed
428     //
429     for (int i = 0; i < g->nv; i++)
430     {
431         // Find the minimum path
432         int min = INFINITY;
433         int min_v;
434         for (int v = 0; v < g->nv; v++) // Find the nearest vex
435             if (!visited[v] && Dist[v] <= min)
436             {
437                 min = Dist[v];
438                 min_v = v;
439             }
440         //
441         visited[min_v] = 1;
442         for (int k = 0; k < g->v[min_v].degree; k++)
443         {
444             Edge e = g->e[g->v[min_v].edges[k]];
445             int vv = (e.h == min_v) ? e.t : e.h;
446             if (!visited[vv] && Dist[min_v] != INFINITY && (Dist[min_v] + e.w) < Dist[vv])
447                 Dist[vv] = Dist[min_v] + e.w;
448         }
449     }
450     printf("Vex \t Distance from Vex[%d]\n", sv);
451     for (int i = 0; i < g->nv; i++)
452         printf("%d \t %d\n", i, Dist[i]);
453 }
454
```

> 注意INFINITY == 999

说明

420-427行：**初始化**所有距离为INFINITY，再将起始点到自己的距离改为0。

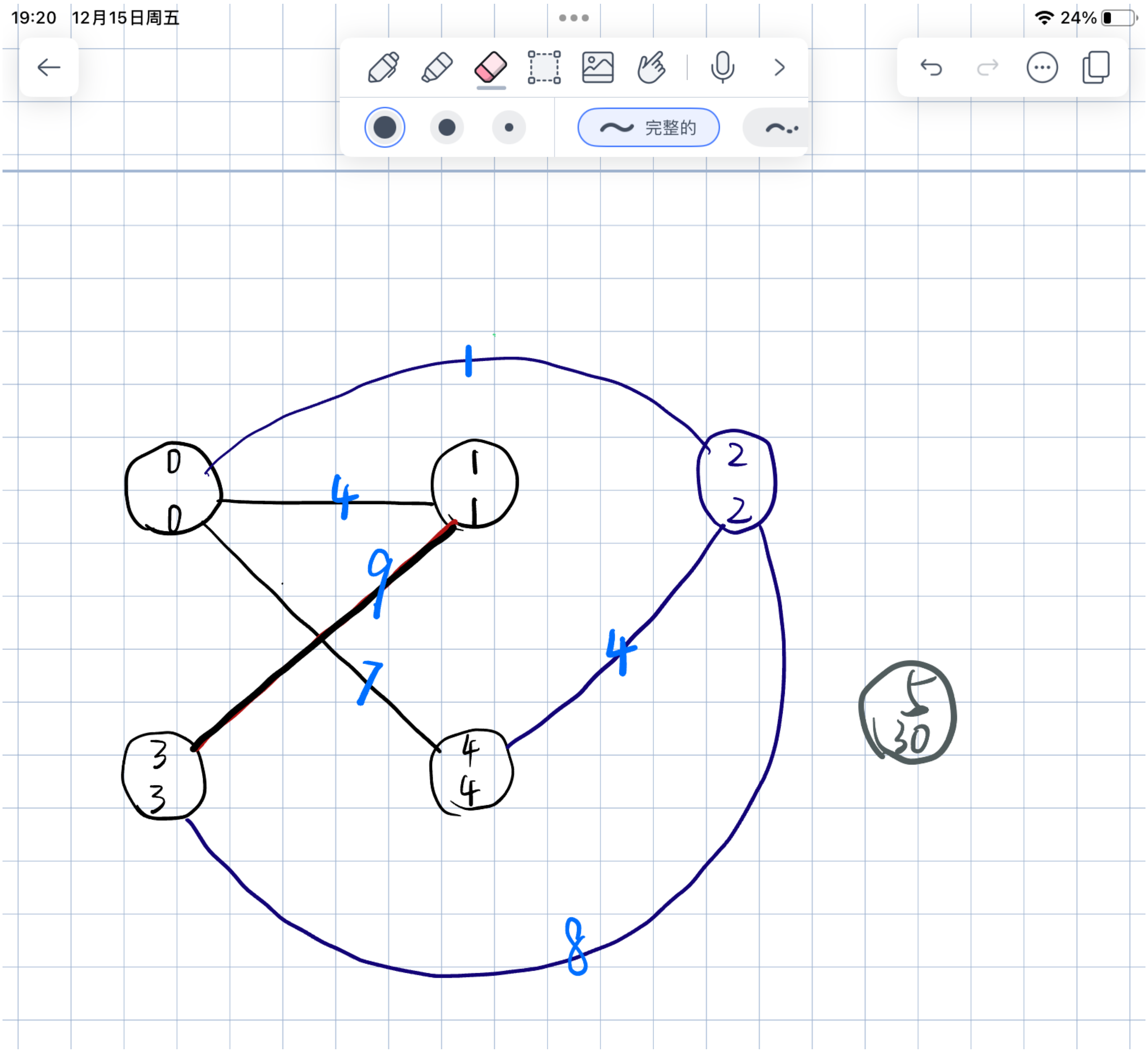
429：设置一个循环，**每次会遍历1个新的顶点**直至遍历完图的所有顶点。

431-441：**找到**此时已遍历了的点集所连接的**最短边**，并将这条边连接到的未遍历的点集中的点赋值给min_v，标记其已被遍历。

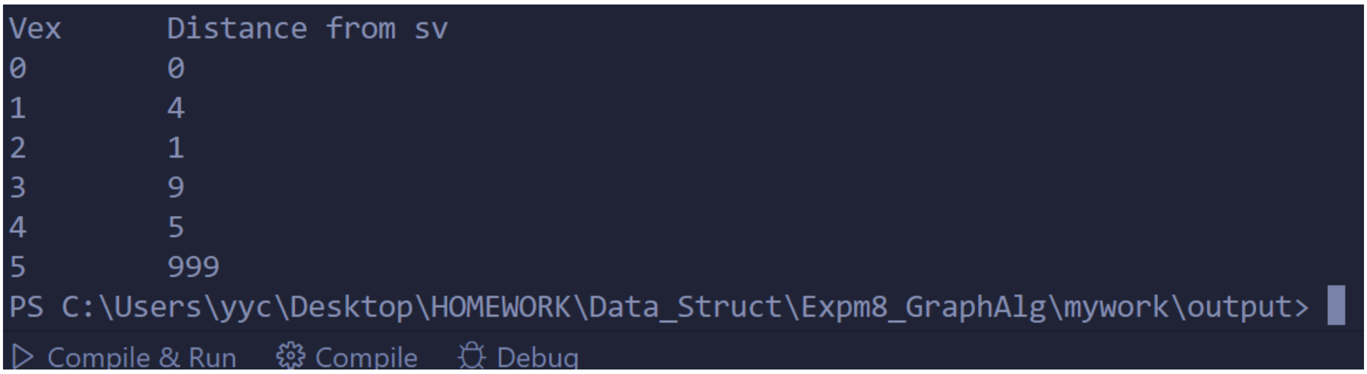
442-449：**遍历min_v**这一点的各条**邻边所连的点**，若此时找到了起始点到这些点的更短的距离则更新Dist[]的值。

450-452：以表格方式**输出**

实现： 对于以下图，执行函数`Dijstra(g, 0);`



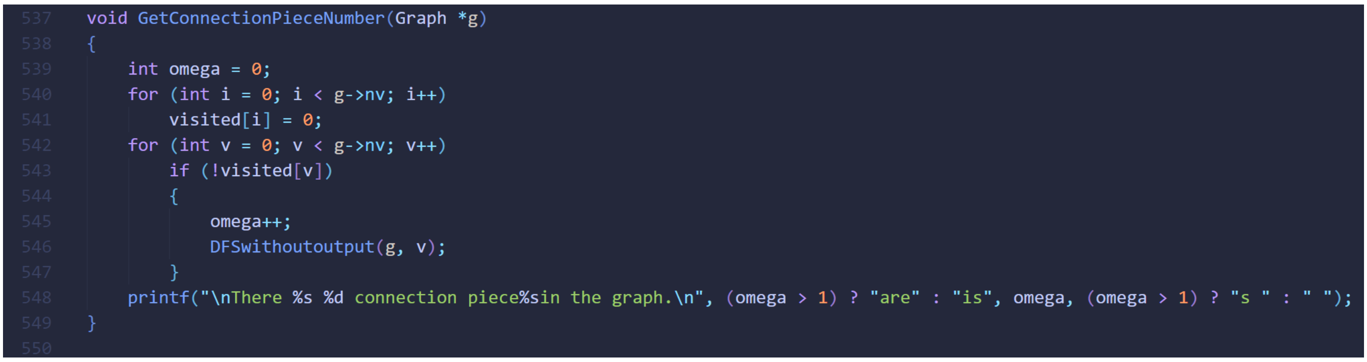
输出结果：



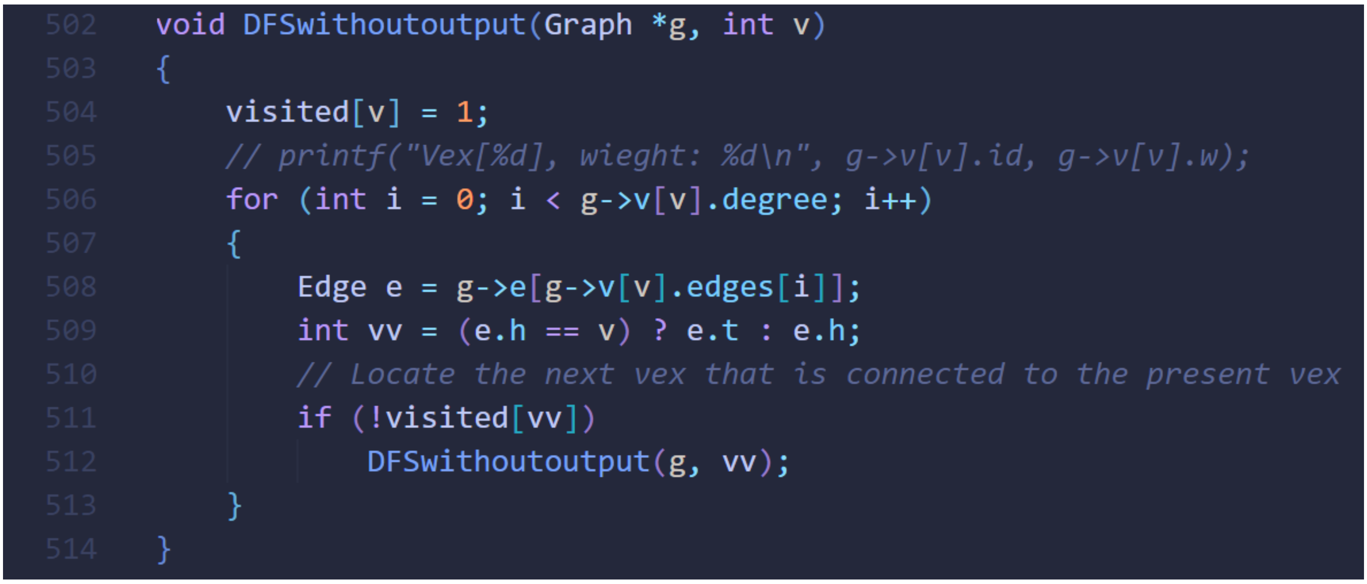
```
Vex       Distance from sv
0         0
1         4
2         1
3         9
4         5
5         999
PS C:\Users\yyc\Desktop\HOMEWORK\Data_Struct\Expm8_GraphAlg\mywork\output> 
```

求联通片个数

代码



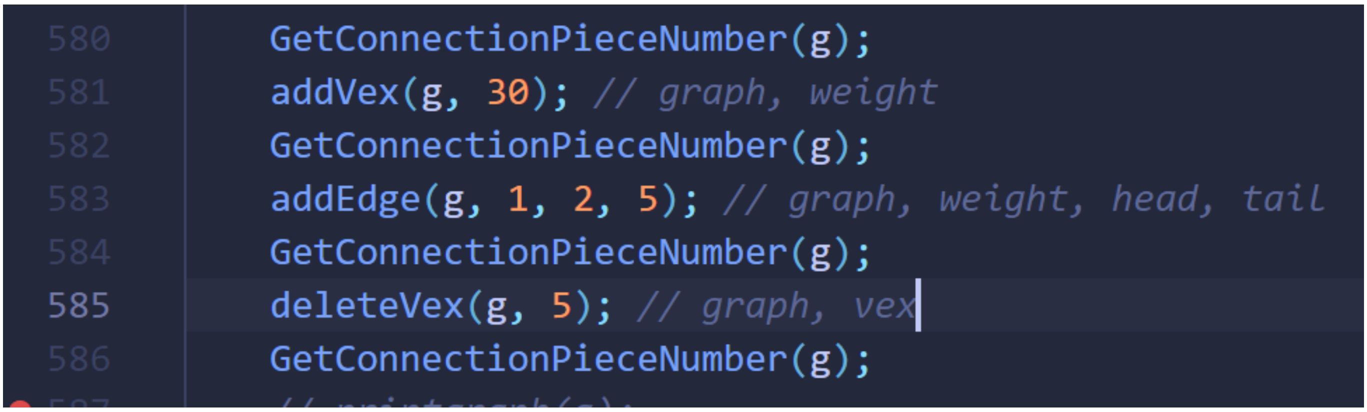
```c
void GetConnectionPieceNumber(Graph *g)
{
    int omega = 0;
    for (int i = 0; i < g->nv; i++)
        visited[i] = 0;
    for (int v = 0; v < g->nv; v++)
        if (!visited[v])
        {
            omega++;
            DFSwithoutoutput(g, v);
        }
    printf("\nThere %s %d connection piece%sin the graph.\n", (omega > 1) ? "are" : "is", omega, (omega > 1) ? "s " : " ");
}
```



```c
void DFSwithoutoutput(Graph *g, int v)
{
    visited[v] = 1;
    // printf("Vex[%d], wieght: %d\n", g->v[v].id, g->v[v].w);
    for (int i = 0; i < g->v[v].degree; i++)
    {
        Edge e = g->e[g->v[v].edges[i]];
        int vv = (e.h == v) ? e.t : e.h;
        // Locate the next vex that is connected to the present vex
        if (!visited[vv])
            DFSwithoutoutput(g, vv);
    }
}
```

实现

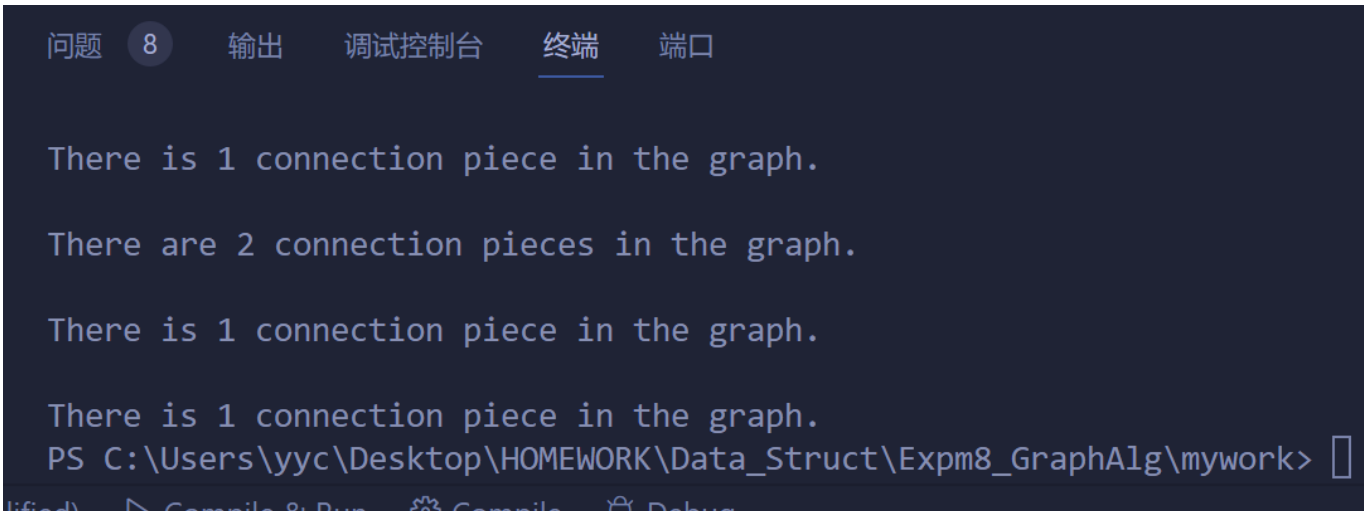
在主函数中使用以下代码进行测试：



```c
580        GetConnectionPieceNumber(g);
581        addVex(g, 30); // graph, weight
582        GetConnectionPieceNumber(g);
583        addEdge(g, 1, 2, 5); // graph, weight, head, tail
584        GetConnectionPieceNumber(g);
585        deleteVex(g, 5); // graph, vex
586        GetConnectionPieceNumber(g);
587        // printgraph(g);
```

> 581行代码添加的点的编号为5

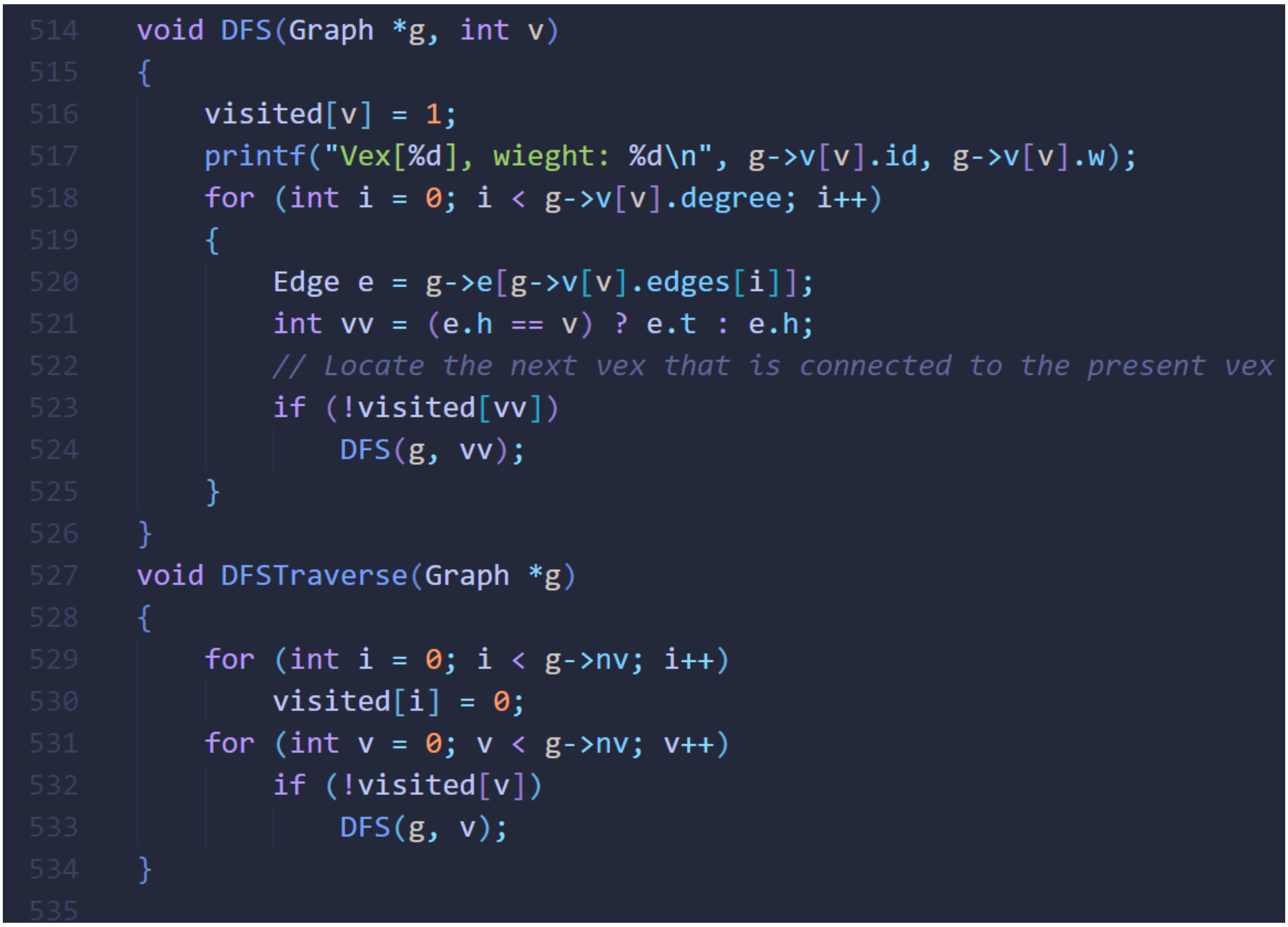
输出结果如下：



```
There is 1 connection piece in the graph.

There are 2 connection pieces in the graph.

There is 1 connection piece in the graph.

There is 1 connection piece in the graph.
PS C:\Users\yyc\Desktop\HOMEWORK\Data_Struct\Expm8_GraphAlg\mywork>
```

深度优先搜索

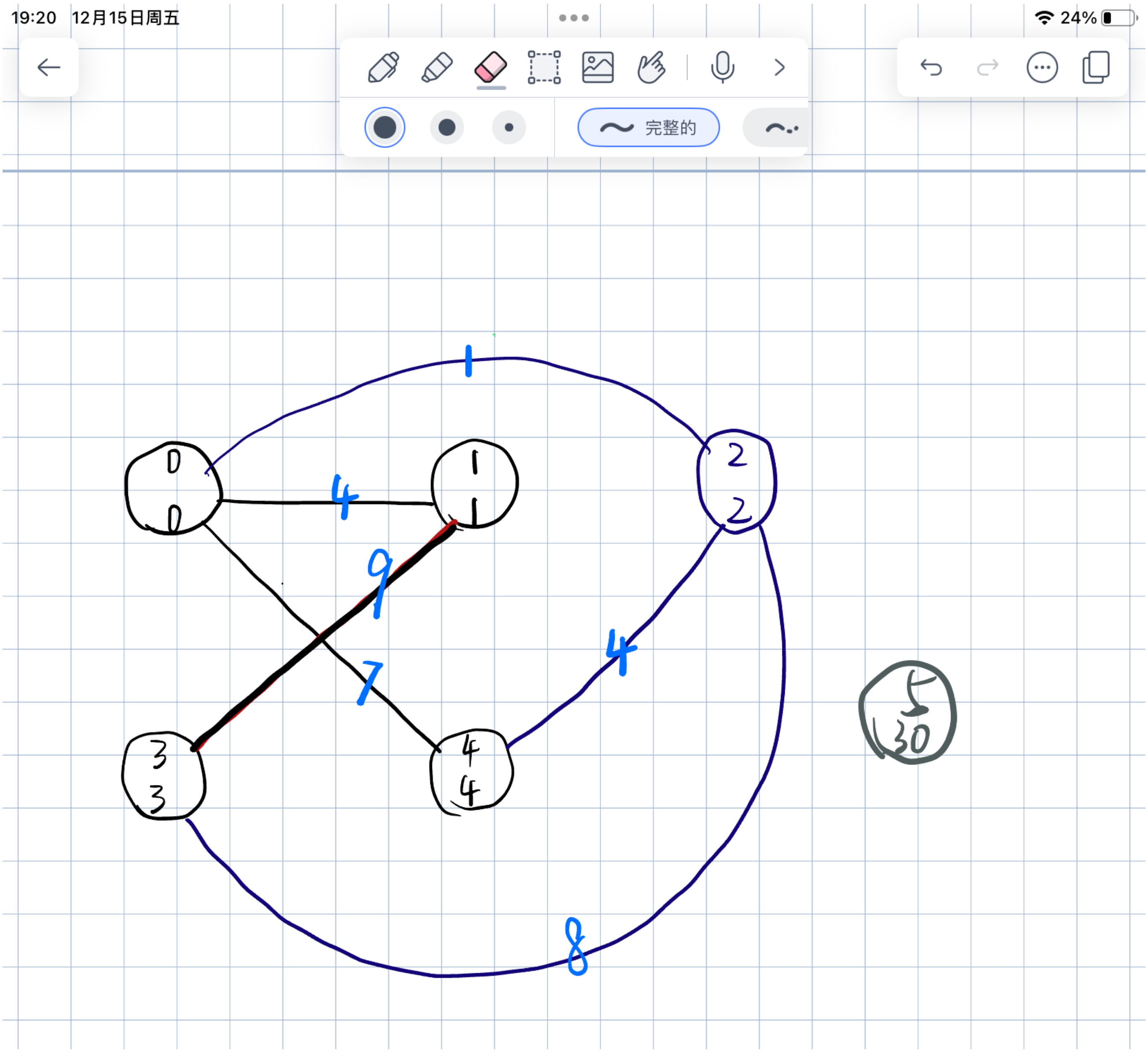
代码



```c
void DFS(Graph *g, int v)
{
    visited[v] = 1;
    printf("Vex[%d], wieght: %d\n", g->v[v].id, g->v[v].w);
    for (int i = 0; i < g->v[v].degree; i++)
    {
        Edge e = g->e[g->v[v].edges[i]];
        int vv = (e.h == v) ? e.t : e.h;
        // Locate the next vex that is connected to the present vex
        if (!visited[vv])
            DFS(g, vv);
    }
}
void DFSTraverse(Graph *g)
{
    for (int i = 0; i < g->nv; i++)
        visited[i] = 0;
    for (int v = 0; v < g->nv; v++)
        if (!visited[v])
            DFS(g, v);
}
```

实现

对以下图使用深度优先搜索



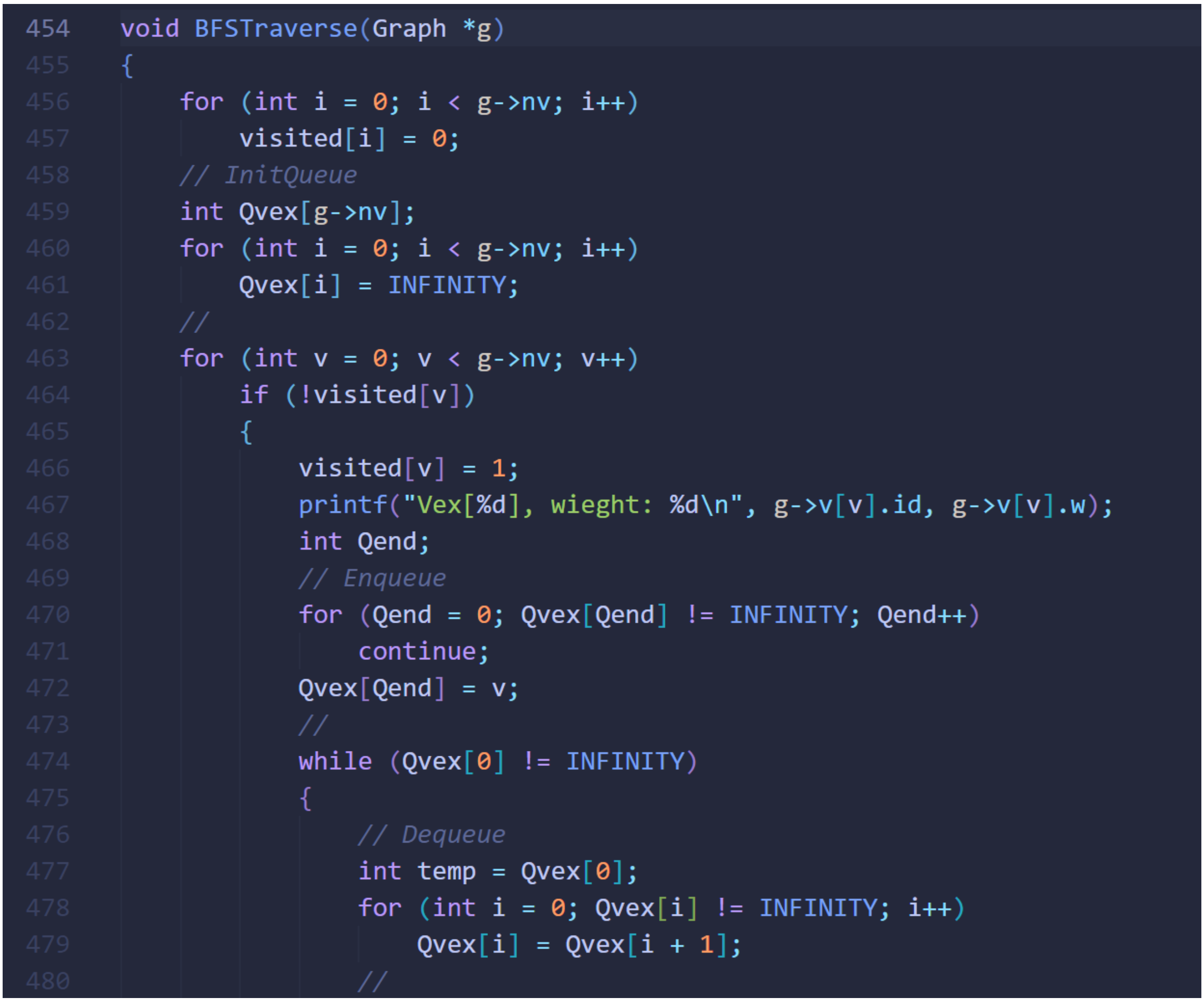
输出：

```
Deep First Search:
Vex[0], wieght: 0
Vex[2], wieght: 2
Vex[4], wieght: 4
Vex[3], wieght: 3
Vex[1], wieght: 1
Vex[5], wieght: 30
```

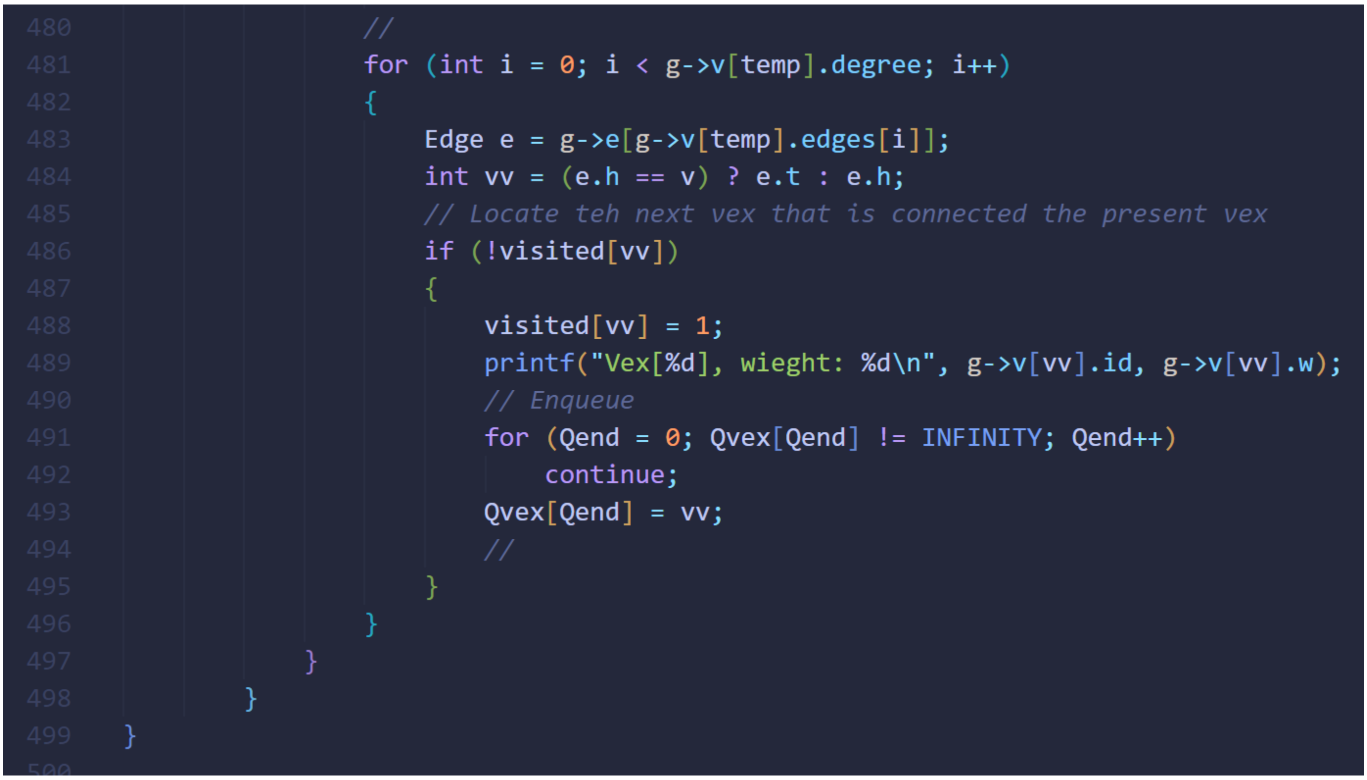
> 0 -> 2 -> 4 -> (回到2) -> 3 -> 1 -> (回到0) -> (此联通片遍历完成) -> (下一个联通片) -> 5

广度优先搜索

代码



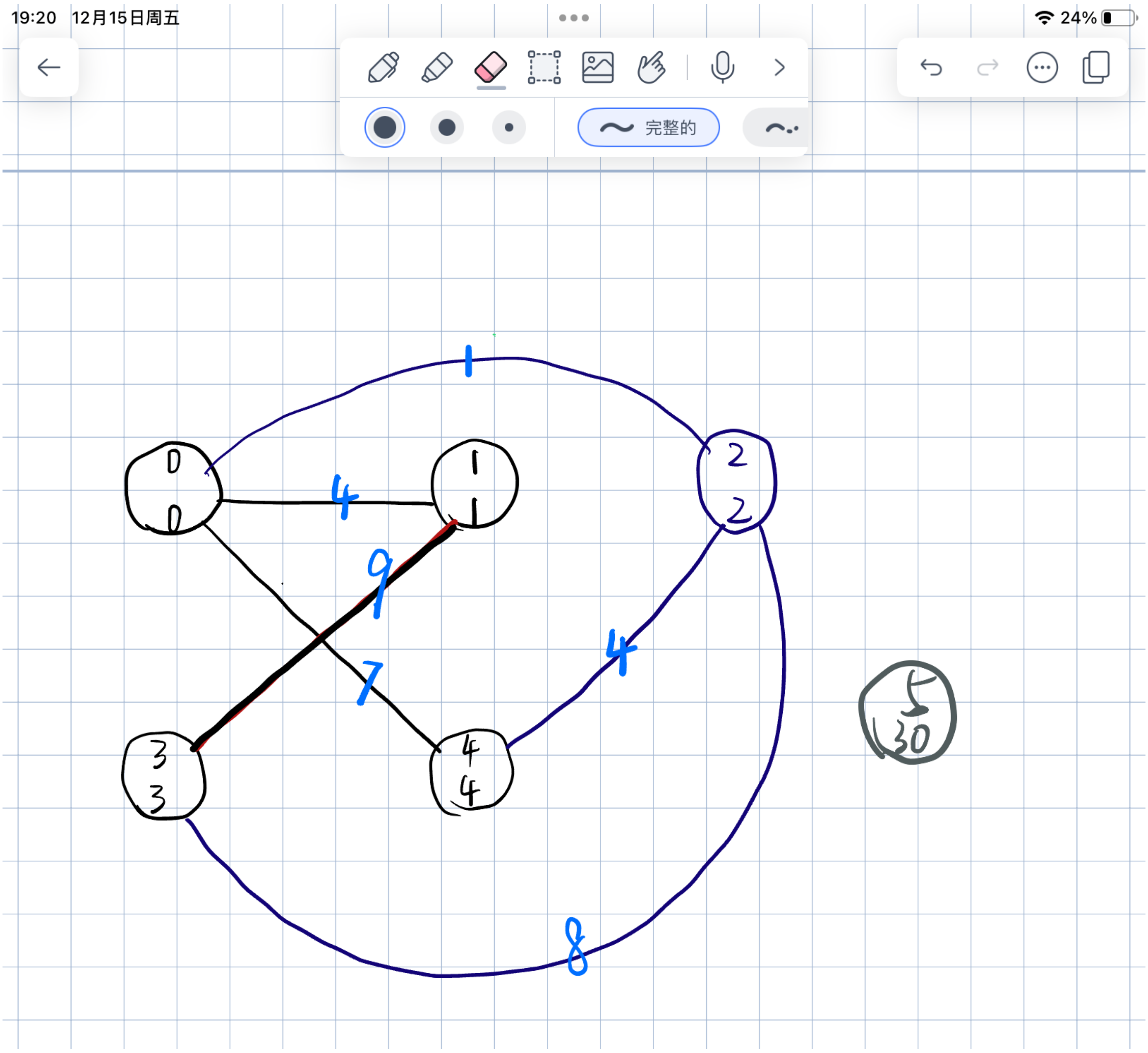
```c
void BFSTraverse(Graph *g)
{
    for (int i = 0; i < g->nv; i++)
        visited[i] = 0;
    // InitQueue
    int Qvex[g->nv];
    for (int i = 0; i < g->nv; i++)
        Qvex[i] = INFINITY;
    //
    for (int v = 0; v < g->nv; v++)
        if (!visited[v])
        {
            visited[v] = 1;
            printf("Vex[%d], wieght: %d\n", g->v[v].id, g->v[v].w);
            int Qend;
            // Enqueue
            for (Qend = 0; Qvex[Qend] != INFINITY; Qend++)
                continue;
            Qvex[Qend] = v;
            //
            while (Qvex[0] != INFINITY)
            {
                // Dequeue
                int temp = Qvex[0];
                for (int i = 0; Qvex[i] != INFINITY; i++)
                    Qvex[i] = Qvex[i + 1];
                //
```



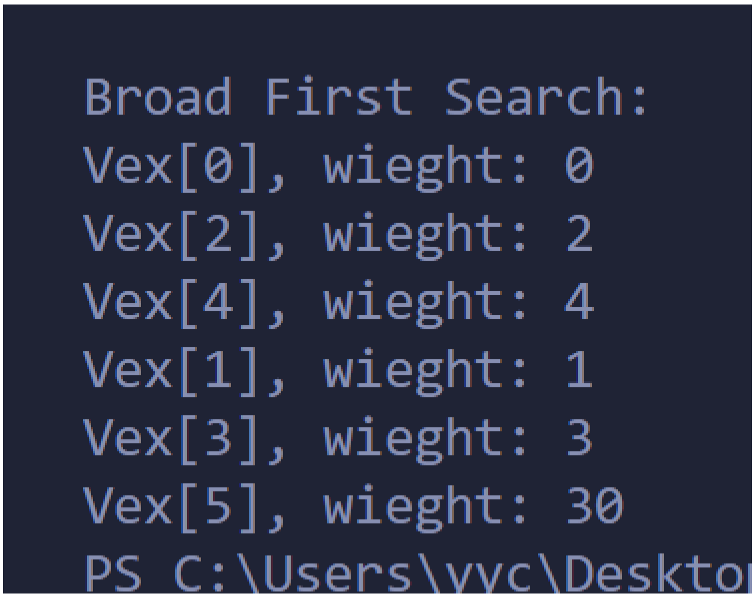
```c
                //
                for (int i = 0; i < g->v[temp].degree; i++)
                {
                    Edge e = g->e[g->v[temp].edges[i]];
                    int vv = (e.h == v) ? e.t : e.h;
                    // Locate teh next vex that is connected the present vex
                    if (!visited[vv])
                    {
                        visited[vv] = 1;
                        printf("Vex[%d], wieght: %d\n", g->v[vv].id, g->v[vv].w);
                        // Enqueue
                        for (Qend = 0; Qvex[Qend] != INFINITY; Qend++)
                            continue;
                        Qvex[Qend] = vv;
                        //
                    }
                }
            }
        }
}
```

实现

对以下图使用广度优先搜索



输出：



```
Broad First Search:
Vex[0], wieght: 0
Vex[2], wieght: 2
Vex[4], wieght: 4
Vex[1], wieght: 1
Vex[3], wieght: 3
Vex[5], wieght: 30
PS C:\Users\yyc\Deskto
```

> 0 -> 2, 4, 1 -> 3 -> (此联通片遍历完成) -> (下一个联通片) -> 5

问题

无。