

Lab 8 Report

基于图实现的算法

PB22111599 杨映川

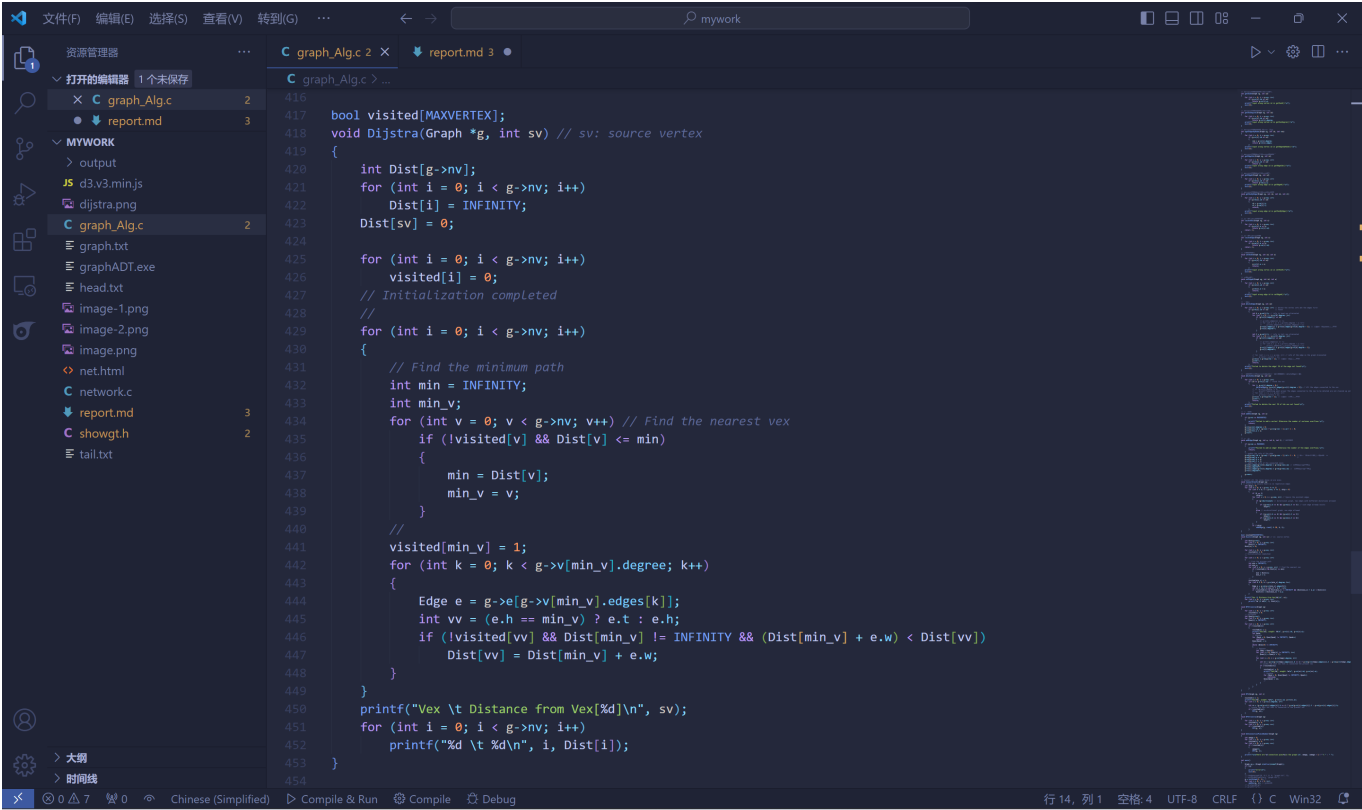
静态结构体

```
C graph_Alg.c 2 ×  report.md 2 ●
C graph_Alg.c > ...
14
15 #define MAXVERTEX 100
16 #define MAXEDGE 20000
17 #define INFINITY 999
18
19 // 顶点结构体
20 typedef struct Vertex
21 {
22     int id;        // 顶点编号（唯一）
23     int w;         // 顶点权值
24     int degree;    // 顶点的度
25     // int in_deg, out_deg; // 入度和出度
26     int edges[MAXVERTEX]; // 边的编号
27 } Vertex;
28
29 // 边结构体
30 typedef struct Edge
31 {
32     int id; // 边的编号（唯一）
33     int h;  // 边头顶点编号
34     int t;  // 边尾顶点编号
35     int w;  // 权值
36 } Edge;
37
38 // 图结构体
39 typedef struct Graph
```

```
39 typedef struct Graph
40 {
41     Vertex v[MAXVERTEX]; // 顶点数组
42     Edge e[MAXEDGE];      // 边数组
43     int nv;               // 顶点数
44     int ne;               // 边数
45     bool dirctional;      // t:有向图, f:无向图
46     bool weighted;       // t:带权图, f:等权图
47 } Graph;
48
```

Dijkstra算法求最短路径

代码



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

注意INFINITY == 999

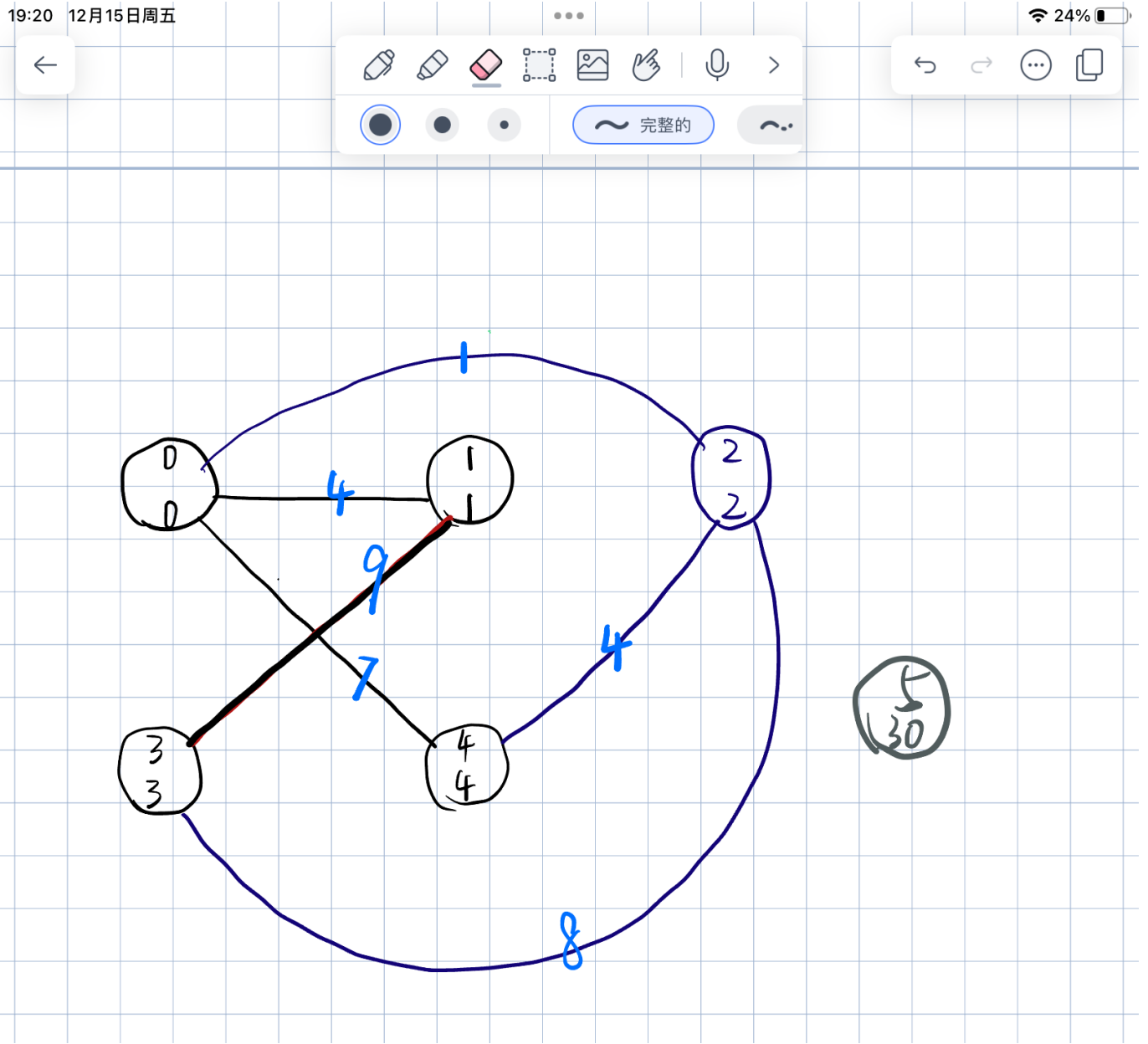
说明

- 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Dijkstra(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>
```

Compile & Run Compile Debug

求联通片个数

代码

```

537 void GetConnectionPieceNumber(Graph *g)
538 {
539     int omega = 0;
540     for (int i = 0; i < g->nv; i++)
541         visited[i] = 0;
542     for (int v = 0; v < g->nv; v++)
543         if (!visited[v])
544         {
545             omega++;
546             DFSwithoutoutput(g, v);
547         }
548     printf("\nThere %s %d connection piece%s in the graph.\n", (omega > 1) ? "are" : "is", omega, (omega > 1) ? "s " : " ");
549 }
550

```

```

502 void DFSwithoutoutput(Graph *g, int v)
503 {
504     visited[v] = 1;
505     // printf("Vex[%d], wieght: %d\n", g->v[v].id, g->v[v].w);
506     for (int i = 0; i < g->v[v].degree; i++)
507     {
508         Edge e = g->e[g->v[v].edges[i]];
509         int vv = (e.h == v) ? e.t : e.h;
510         // Locate the next vex that is connected to the present vex
511         if (!visited[vv])
512             DFSwithoutoutput(g, vv);
513     }
514 }

```

实现

在主函数中使用以下代码进行测试：

```

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     // printf("graph: %s\n", g->name);

```

581行代码添加的点的编号为5

输出结果如下：

```

问题 8 输出 调试控制台 终端 端口

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

```

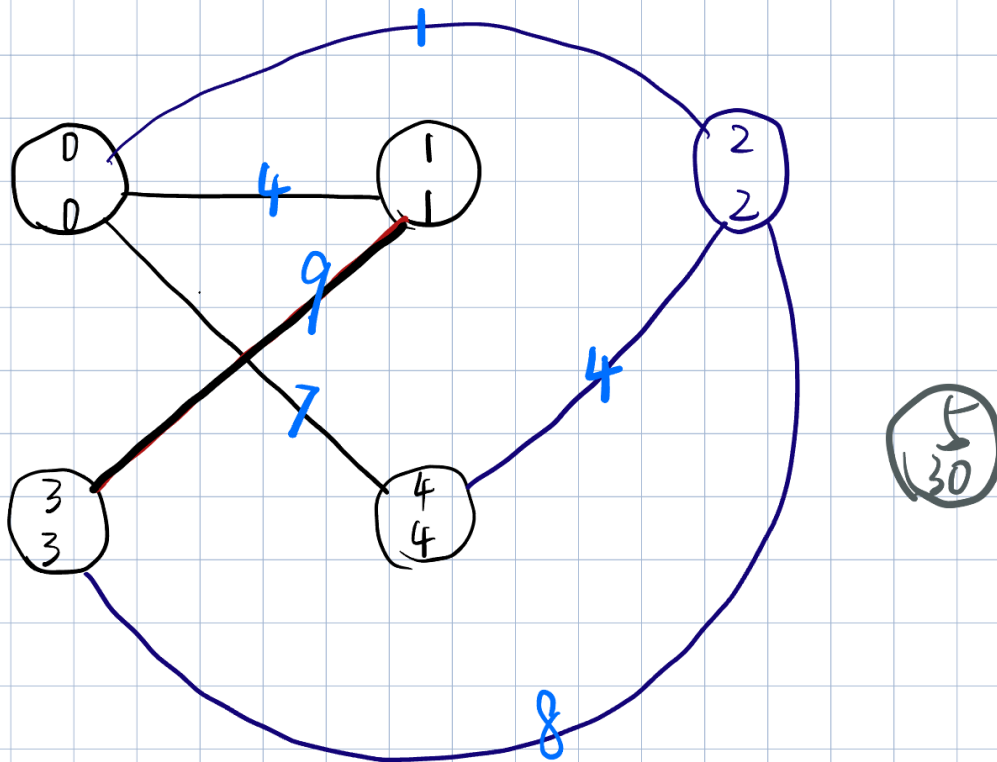
514 void DFS(Graph *g, int v)
515 {
516     visited[v] = 1;
517     printf("Vex[%d], wieght: %d\n", g->v[v].id, g->v[v].w);
518     for (int i = 0; i < g->v[v].degree; i++)
519     {
520         Edge e = g->e[g->v[v].edges[i]];
521         int vv = (e.h == v) ? e.t : e.h;
522         // Locate the next vex that is connected to the present vex
523         if (!visited[vv])
524             DFS(g, vv);
525     }
526 }
527 void DFSTraverse(Graph *g)
528 {
529     for (int i = 0; i < g->nv; i++)
530         visited[i] = 0;
531     for (int v = 0; v < g->nv; v++)
532         if (!visited[v])
533             DFS(g, v);
534 }
535

```

实现

对下图使用深度优先搜索

19:20 12月15日周五



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

```

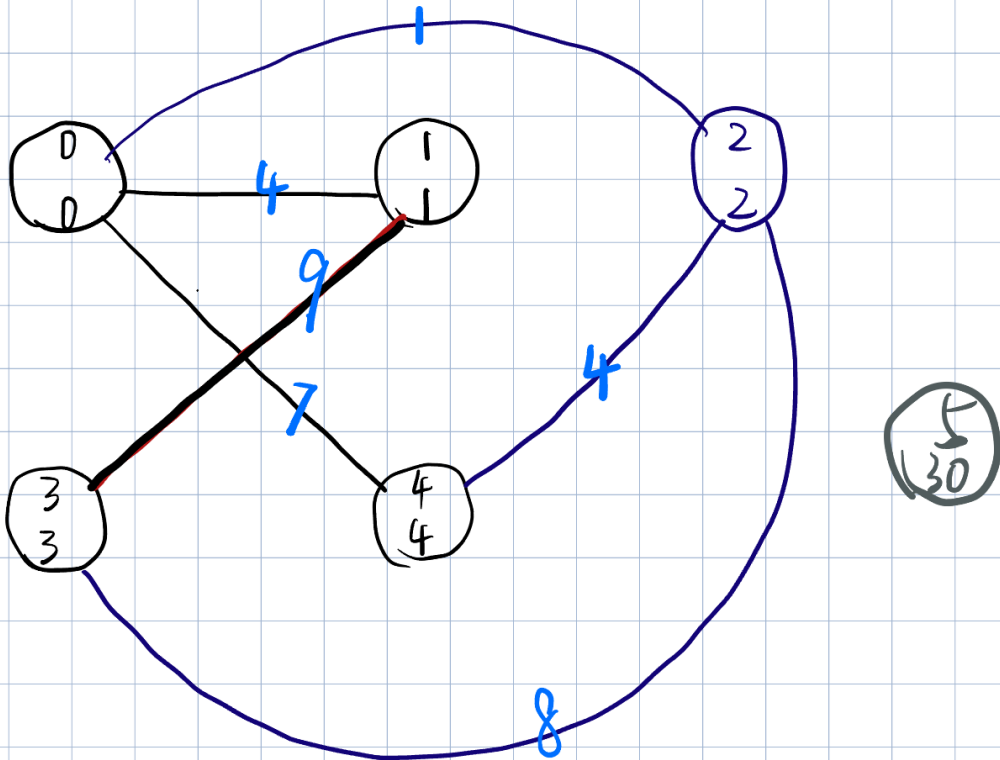
454 void BFSTraverse(Graph *g)
455 {
456     for (int i = 0; i < g->nv; i++)
457         visited[i] = 0;
458     // InitQueue
459     int Qvex[g->nv];
460     for (int i = 0; i < g->nv; i++)
461         Qvex[i] = INFINITY;
462     //
463     for (int v = 0; v < g->nv; v++)
464         if (!visited[v])
465         {
466             visited[v] = 1;
467             printf("Vex[%d], wieght: %d\n", g->v[v].id, g->v[v].w);
468             int Qend;
469             // Enqueue
470             for (Qend = 0; Qvex[Qend] != INFINITY; Qend++)
471                 continue;
472             Qvex[Qend] = v;
473             //
474             while (Qvex[0] != INFINITY)
475             {
476                 // Dequeue
477                 int temp = Qvex[0];
478                 for (int i = 0; Qvex[i] != INFINITY; i++)
479                     Qvex[i] = Qvex[i + 1];
480                 //
481                 for (int i = 0; i < g->v[temp].degree; i++)
482                 {
483                     Edge e = g->e[g->v[temp].edges[i]];
484                     int vv = (e.h == v) ? e.t : e.h;
485                     // Locate teh next vex that is connected the present vex
486                     if (!visited[vv])
487                     {
488                         visited[vv] = 1;
489                         printf("Vex[%d], wieght: %d\n", g->v[vv].id, g->v[vv].w);
490                         // Enqueue
491                         for (Qend = 0; Qvex[Qend] != INFINITY; Qend++)
492                             continue;
493                         Qvex[Qend] = vv;
494                         //
495                     }
496                 }
497             }
498         }
499 }
500

```

实现

对下图使用广度优先搜索

19:20 12月15日周五



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\yvc\Desktop
```

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

问题

无。