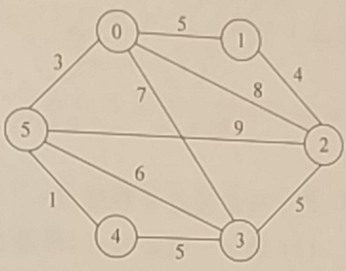
# 实验13 图算法的验证

## 实验要求：

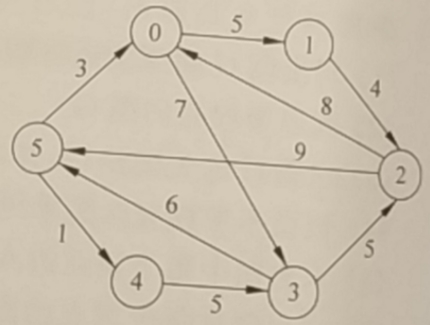
1、用prim算法手工求最小生成树，然后再运算程序进行验证。

要求像教材那样给出步骤，并且在每一步都输出数组closest的值，算法运行也输出该数组的值。



2、用Dijkstra算法求最短路径，然后再运行程序进行验证。

要求像教材那样给出步骤，然后于算法运行的结果进行比较。



## 源代码：

#include <stdio.h>

#include <malloc.h>

#include "exp12-graph.h"

//------------------------------------------------------------

//----邻接矩阵的基本运算算法----------------------------------

//------------------------------------------------------------

void CreateMat(MatGraph& g, int A[MAXV][MAXV], int n, int e) //创建图的邻接矩阵

{

int i, j;

g.n = n; g.e = e;

for (i = 0; i < g.n; i++)

for (j = 0; j < g.n; j++)

g.edges[i][j] = A[i][j];

}

void DispMat(MatGraph g) //输出邻接矩阵g

{

int i, j;

for (i = 0; i < g.n; i++)

{

for (j = 0; j < g.n; j++)

if (g.edges[i][j] != INF)

printf("%4d", g.edges[i][j]);

else

printf("%4s", "∞");

printf("\n");

}

}

//------------------------------------------------------------

//------------------------------------------------------------

//----邻接表的基本运算算法------------------------------------

//------------------------------------------------------------

void CreateAdj(AdjGraph\*& G, int A[MAXV][MAXV], int n, int e) //创建图的邻接表

{

int i, j;

ArcNode\* p;

G = (AdjGraph\*)malloc(sizeof(AdjGraph));

for (i = 0; i < n; i++) //给邻接表中所有头结点的指针域置初值

G->adjlist[i].firstarc = NULL;

for (i = 0; i < n; i++) //检查邻接矩阵中每个元素

for (j = n - 1; j >= 0; j--)

if (A[i][j] != 0 && A[i][j] != INF) //存在一条边

{

p = (ArcNode\*)malloc(sizeof(ArcNode)); //创建一个结点p

p->adjvex = j;

p->weight = A[i][j];

p->nextarc = G->adjlist[i].firstarc; //采用头插法插入结点p

G->adjlist[i].firstarc = p;

}

G->n = n; G->e = n;

}

void DispAdj(AdjGraph\* G) //输出邻接表G

{

int i;

ArcNode\* p;

for (i = 0; i < G->n; i++)

{

p = G->adjlist[i].firstarc;

printf("%3d: ", i);

while (p != NULL)

{

printf("%3d[%d]→", p->adjvex, p->weight);

p = p->nextarc;

}

printf("∧\n");

}

}

void DestroyAdj(AdjGraph\*& G) //销毁图的邻接表

{

int i;

ArcNode\* pre, \* p;

for (i = 0; i < G->n; i++) //扫描所有的单链表

{

pre = G->adjlist[i].firstarc; //p指向第i个单链表的首结点

if (pre != NULL)

{

p = pre->nextarc;

while (p != NULL) //释放第i个单链表的所有边结点

{

free(pre);

pre = p; p = p->nextarc;

}

free(pre);

}

}

free(G); //释放头结点数组

}

//------------------------------------------------------------

//------------------------------------------------------------

//----prim算法------------------------------------------------

//------------------------------------------------------------

void Prim(MatGraph g, int v)

{

int lowcost[MAXV]; //顶点i是否在U中

int min;

int closest[MAXV], i, j, k;

for (i = 0; i < g.n; i++) //给lowcost[]和closest[]置初值

{

lowcost[i] = g.edges[v][i];//存储权值,第0行的第i边

closest[i] = v; //存储结点,v=0，此时存储顶点数个数的零值

//printf("lowcost-%d-", lowcost[i]);

//printf("此时%d", closest[i]);

//for (int i = 0; i < 6; i++) //输出closest 查看数组里面的内容

//{

// printf("-%d-", closest[i]);//只存在6个数值

//}

}

for (i = 1; i < g.n; i++) //找出n-1个顶点

{

min = INF;

for (j = 0; j < g.n; j++) //在(V-U)中找出离U最近的顶点k

if (lowcost[j] != 0 && lowcost[j] < min)

{

min = lowcost[j];

k = j; //k记录最近顶点的编号

//printf("lowcost-%d-",lowcost[j]);

}

printf("closest为：");

for (int i = 0; i < 6; i++) //输出closest 查看数组里面的内容

{

printf(" %d ", closest[i]);//只存在6个数值

}

printf("\n");

printf(" 边(%d,%d)权为:%d\n\n", closest[k], k, min);

lowcost[k] = 0; //标记k已经加入U，改0

for (j = 0; j < g.n; j++) //修改数组lowcost和closest

if (g.edges[k][j] != 0 && g.edges[k][j] < lowcost[j])//更换行

{

lowcost[j] = g.edges[k][j];//将权值更改为另一行的权值

closest[j] = k; //存储结点, v = 5，此时存储顶点数个数的零值

}

}

printf("生成最小生成树后closest为：");

for (int i = 0; i < 6; i++) //输出closest 查看数组里面的内容

{

printf(" %d ", closest[i]);//只存在6个数值

}

printf("\n");

}

//------------------------------------------------------------

//----Dijkstra算法--------------------------------------------

//------------------------------------------------------------

void Dispath(MatGraph g, int dist[], int path[], int S[], int v)

//输出从顶点v出发的所有最短路径

{

int i, j, k;

int apath[MAXV], d; //存放一条最短路径(逆向)及其顶点个数

for (i = 0; i < g.n; i++) //循环输出从顶点v到i的路径

if (S[i] == 1 && i != v)

{

printf(" 从顶点%d到顶点%d的路径长度为:%d\t路径为:", v, i, dist[i]);

d = 0; apath[d] = i; //添加路径上的终点

k = path[i];

if (k == -1) //没有路径的情况

printf("无路径\n");

else //存在路径时输出该路径

{

while (k != v)

{

d++; apath[d] = k;

k = path[k];

}

d++; apath[d] = v; //添加路径上的起点

printf("%d", apath[d]); //先输出起点

for (j = d - 1; j >= 0; j--) //再输出其他顶点

printf(",%d", apath[j]);

printf("\n");

}

}

}

void Dijkstra(MatGraph g, int v) //Dijkstra算法

{

int dist[MAXV], path[MAXV];

int S[MAXV]; //S[i]=1表示顶点i在S中, S[i]=0表示顶点i在U中

int Mindis, i, j, u;

for (i = 0; i < g.n; i++)

{

dist[i] = g.edges[v][i]; //距离初始化

S[i] = 0; //S[]置空

if (g.edges[v][i] < INF) //路径初始化

path[i] = v; //顶点v到顶点i有边时，置顶点i的前一个顶点为v

else

path[i] = -1; //顶点v到顶点i没边时，置顶点i的前一个顶点为-1

}

S[v] = 1; path[v] = 0; //源点编号v放入S中

for (i = 0; i < g.n - 1; i++) //循环直到所有顶点的最短路径都求出

{

Mindis = INF; //Mindis置最大长度初值

for (j = 0; j < g.n; j++) //选取不在S中（即U中）且具有最小最短路径长度的顶点u

if (S[j] == 0 && dist[j] < Mindis)

{

u = j;

Mindis = dist[j];

}

S[u] = 1; //顶点u加入S中

for (j = 0; j < g.n; j++) //修改不在S中（即U中）的顶点的最短路径

if (S[j] == 0)

if (g.edges[u][j] < INF && dist[u] + g.edges[u][j] < dist[j])

{

dist[j] = dist[u] + g.edges[u][j];

path[j] = u;

}

}

Dispath(g, dist, path, S, v); //输出最短路径

}

//主函数----------------------------

int main()

{

MatGraph g,g2;

int A[MAXV][MAXV] = //原调试邻接矩阵

{

{0,28,INF,INF,INF,10,INF},

{28,0,16,INF,INF,INF,14},

{INF,16,0,12,INF,INF,INF},

{INF,INF,12,0,22,INF,18},

{INF,INF,INF,22,0,25,24},

{10,INF,INF,INF,25,0,INF},

{INF,14,INF,18,24,INF,0}

};

int n = 7, e = 9;

int B[MAXV][MAXV] =//prim算法邻接矩阵

{

//0 1 2 3 4 5

{0, 5, 8, 7, INF, 3},//0

{5, 0, 4,INF,INF,INF},//1

{8, 4, 0, 5, INF, 9},//2

{7,INF,5, 0, 5 , 6},//3

{INF,INF,INF,5,0,1},//4

{3,INF,9, 6, 1, 0}//5

};

int n2 = 6, e2 = 10;

int C[MAXV][MAXV] =//Dijkstra算法邻接矩阵

{

//0 1 2 3 4 5

{0,5,INF,7,INF,INF},//0

{INF,0,4,INF,INF,INF},//1

{8,INF,0,INF,INF,9},//2

{INF,INF,5,0,INF,6},//3

{INF,INF,INF,5,0,INF},//4

{3,INF,INF,INF,1,0},//5

};

int n3 = 6, e3 = 10;

CreateMat(g, B, n2, e2); //建立prim算法的邻接矩阵

printf("图G的邻接矩阵:\n");

DispMat(g); //输出邻接矩阵

int v = 0;

printf("\n-----Prim算法求最小生成树-----");

printf("\nPrim算法结果(起始点为%d)\n", v);

Prim(g, v);

printf("\n-----Dijkstra算法求最短路径-----");

CreateMat(g2, C, n3, e3);//建立Dijkstra的邻接矩阵

printf("\n从%d顶点出发的最短路径如下:\n", v);

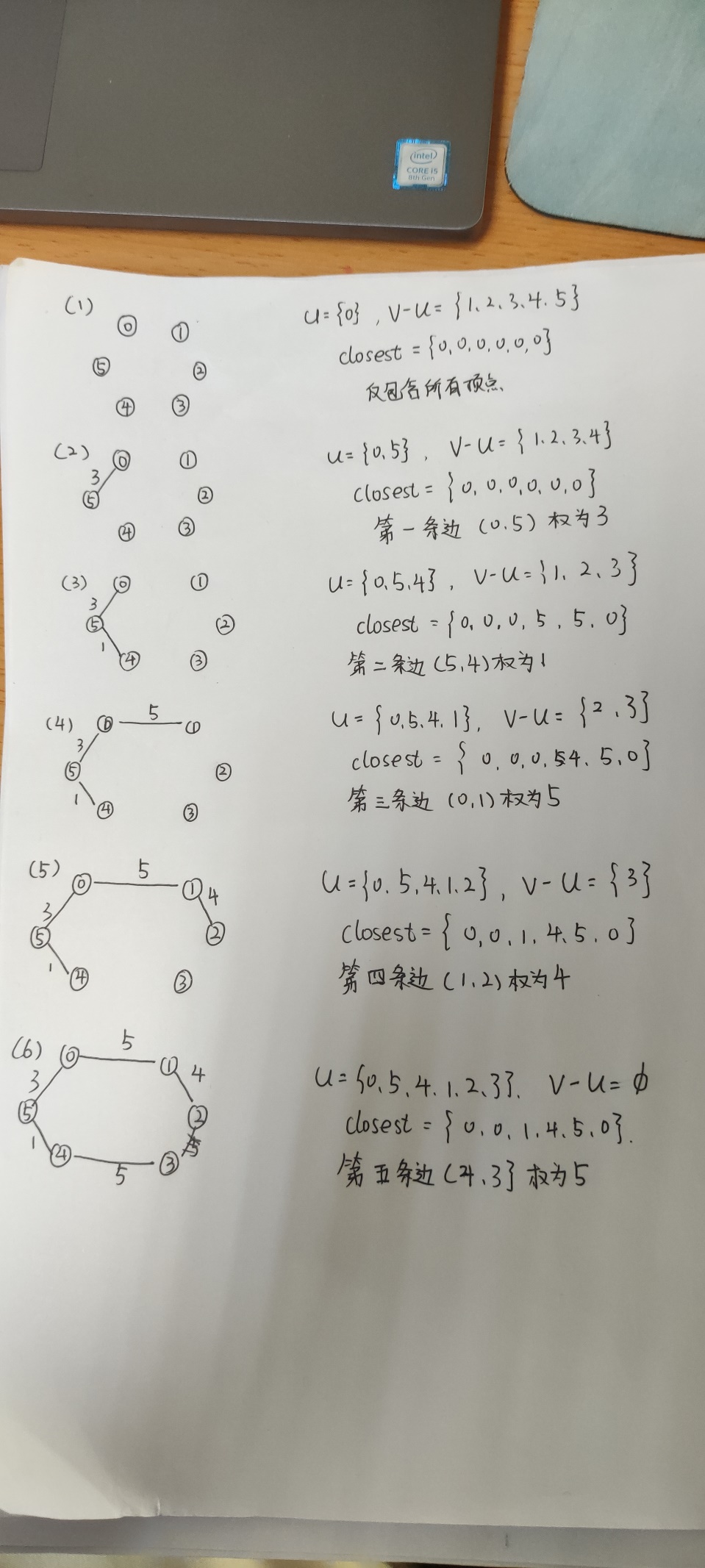
Dijkstra(g2, v);

return 1;

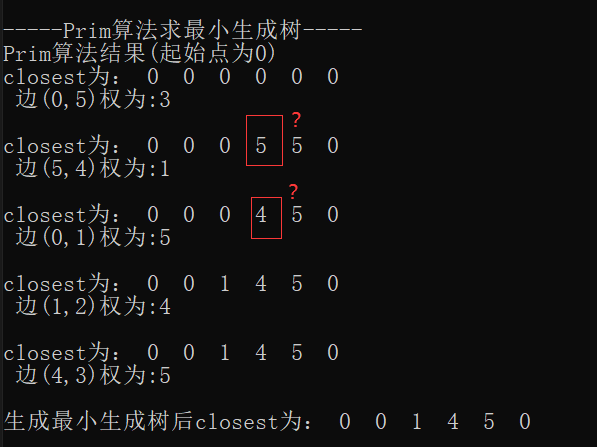
}

## 说明：

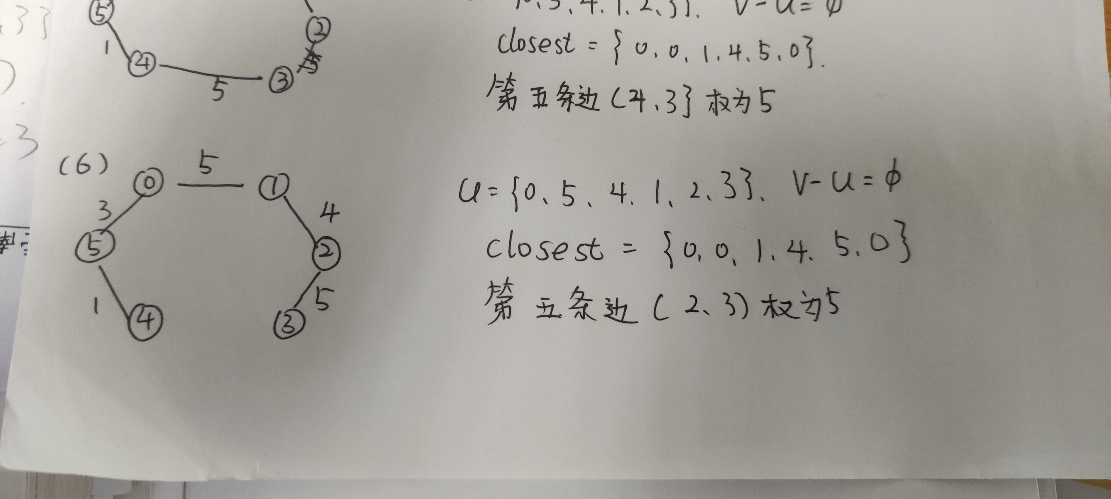
Prim算法：



根据教材书上的Prim算法的图解过程，结合题目要求与题目内容，分析后得到如上图手写的求最小生成树的过程，但也存在问题，在进行分析过程的时候，发现并没有理解到closest{}数组里包含的是什么，以及在何时能修改其值，在运行程序后发现会有无法理解到的修改值的位置，如下图：

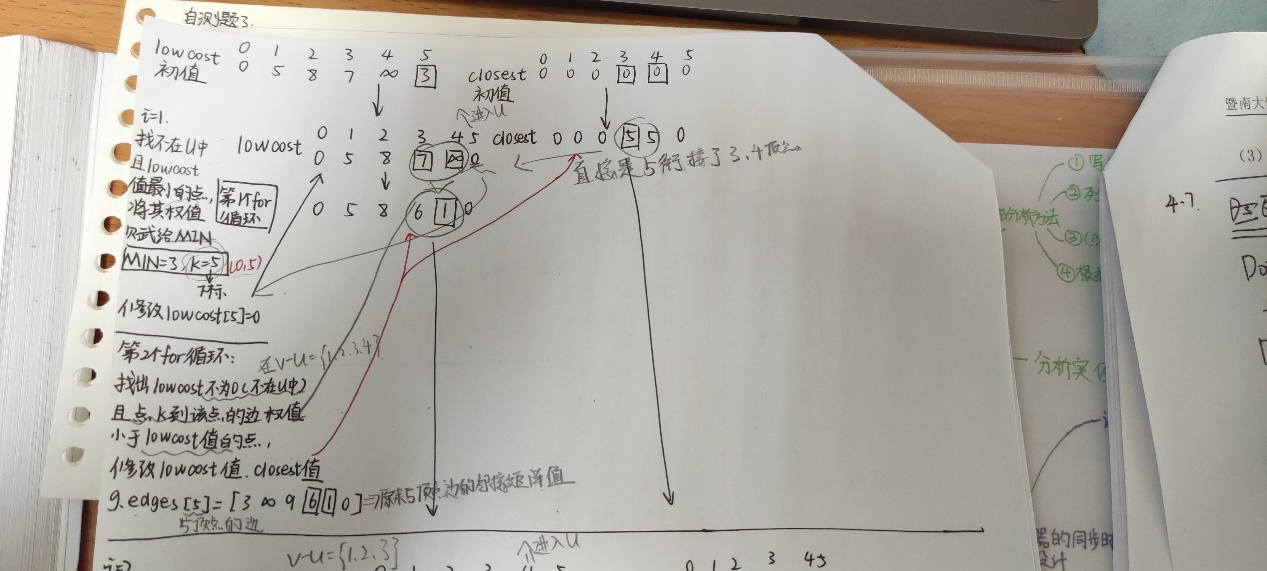


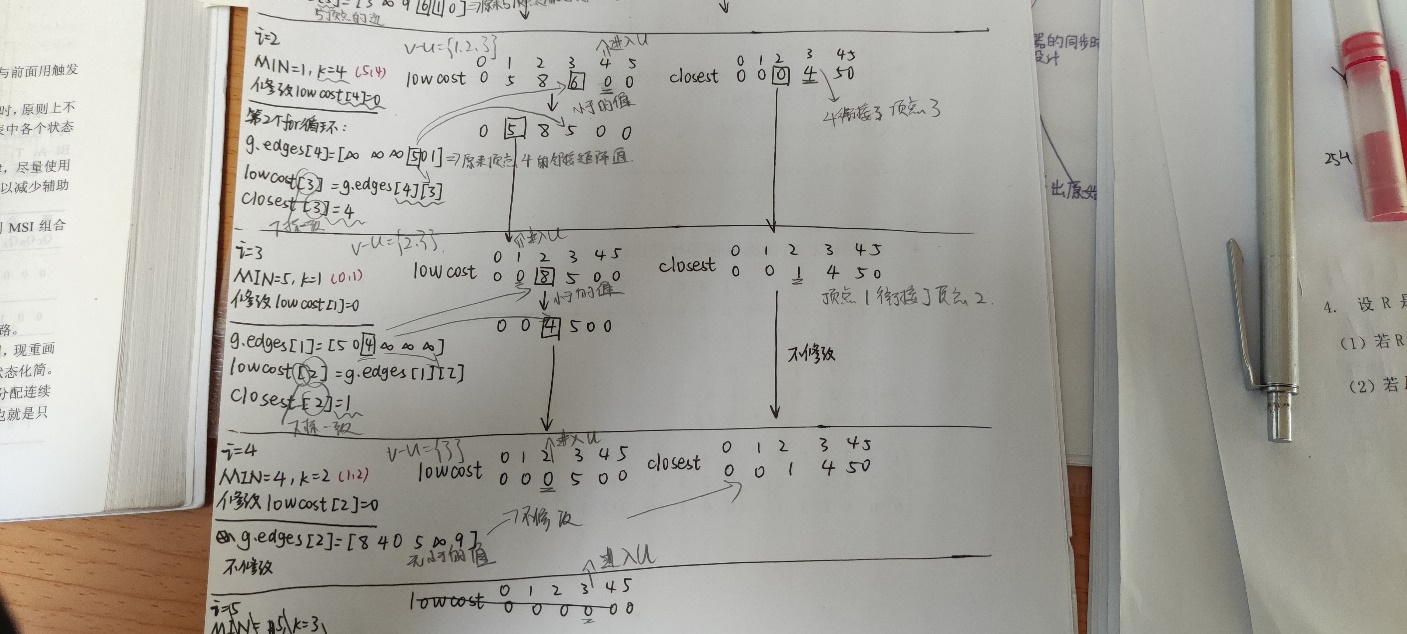
并且在每一步运算的时候发现，图中存在一样数值的路径的时候不知道怎么进行选择，存在如下的选择：



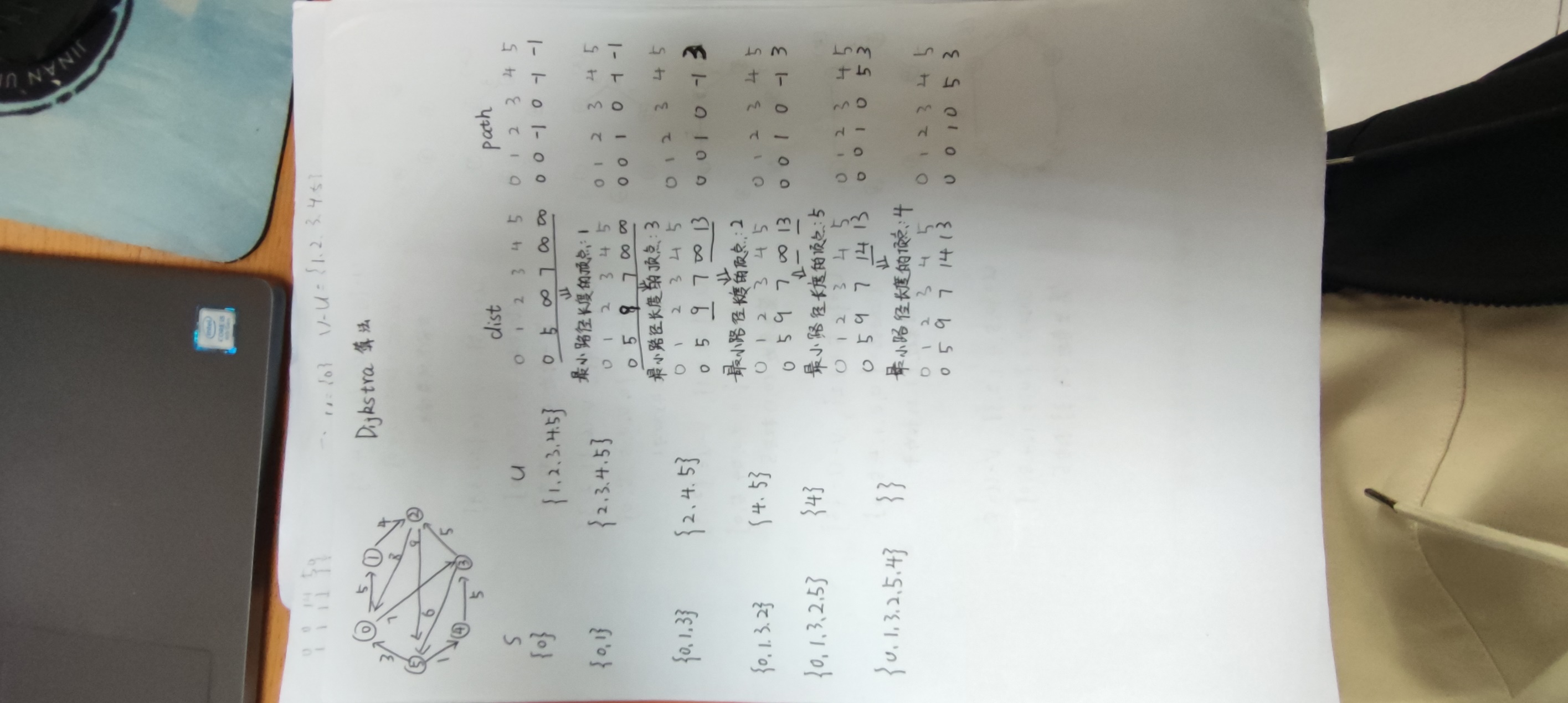
但是程序运行的结果显然是另外一条的路径。

在询问同学之后，可以得知其上不解修改值修改过程如下：



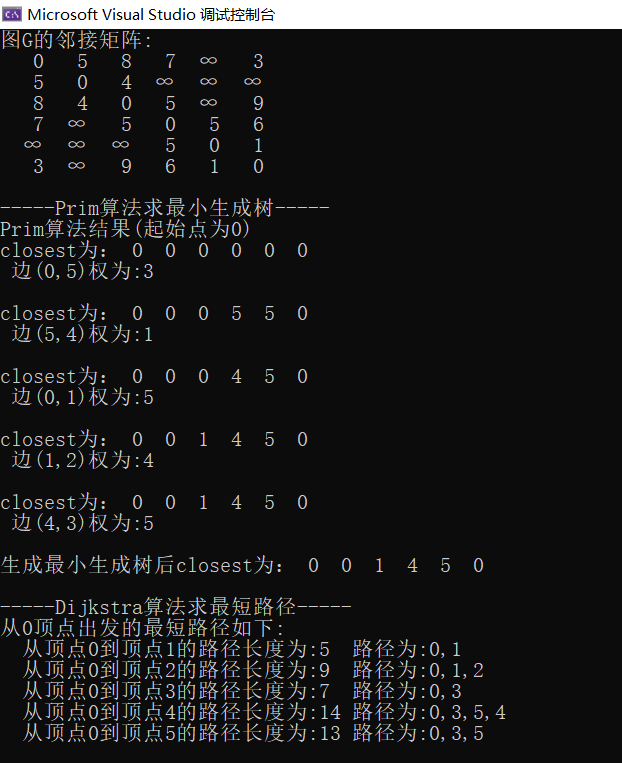


Dijkstra算法：



通过对教材的理解以及教材Dijkstra算法的分析过程结合题目要求以及题目内容可以得出如上图的分析过程，期间存在对path{}数组的不理解，在分析过程中不知道在何时修改path{}数组对应的值，以及在选择路径的时候会存在已选择路径的重复分析和其之后顶点的衔接问题，就导致path{}数组路径选择后修改的前一个顶点的问题。

## 程序运行截图：



实验输出与实验要求一致。

## 实验小结：

通过这次实验，认识到了求最小生成树的Prim算法和从一个顶点到其余各顶点的最短路径的Dijkstra算法，能够在其中通过对图的算法的验证和对两种算法的分析，对图算法有了更好的认识，并且在其中存在的问题可以反映出自己是否对两种算法的掌握程度。