《数据结构与算法》作业

22920212204392 黄勖

习题 4　图结构

4-1 设某个非连通无向图有 25 条边，问该图至少有(　C　)个顶点。

(A) 7
(B) 8
(C) 9
(D) 10

思路：8 个点 1+2+3+4+5+6+7=28 最多边数 另外一个独立点

4-2 设某无向图中有 n 个顶点 e 条边，则建立该图邻接表的时间复杂度为(　A　)。

(A) O(n+e)
(B) $O(n^2)$
(C) O(ne)
(D) $O(n^3)$

4-3 带权有向图 G 用邻接矩阵 R 存储，则顶点 i 的入度等于 R 中(　　D　　)。

(A) 第 i 行非∞(或非 0)的元素之和
(B) 第 i 列非∞(或非 0)的元素之和
(C) 第 i 行非∞(或非 0)的元素个数
(D) 第 i 列非∞(或非 0)的元素个数

4-4 下面关于无向图的存储结构叙述中，正确的是(　　B　　)。

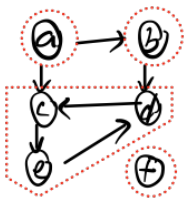
(A) 用邻接表存储图，占用的存储空间大小与图中边数有关，与顶点数无关
(B) 用邻接表存储图，占用的存储空间大小与图中边数和顶点数都有关
(C) 用邻接矩阵存储图，占用的存储空间大小与图中边数和顶点数都有关
(D) 用邻接矩阵存储图，占用的存储空间大小与图中边数有关，与顶点数无关

4-5 设图 G=(V, E)，V={ a, b, c, d, e }，E={<a, b>, <a, c>, <b, d>, <c, e>, <d, c>, <e, d>}。

(1)是否存在从 c 到 b 的路径?
(2)计算 ID(d)、OD(d)、TD(d)；
(3)画出各个强连通分量。

答：

(1) 不存在
(2) ID(d)=2，OD（d）=1，TD（d）=3。(2+1)



(3)

4-6 设计算法，由依次输入的顶点数目、弧的数目、各个顶点元素信息和各条弧信息建立有向图的邻接表。

答：

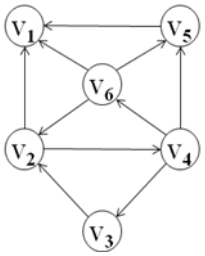
```
#include <iostream>
#define SIZE 1000
using namespace std;
typedef struct ANode{
    int v;
    int weight;
    struct ANode* next;
}ANode,*ALink;
typedef struct VNode{
    char data;
    ALink Vh;
}VNode;
typedef struct List{
    VNode v[SIZE];
}List;
int main()
{
    List L;
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        cin>>L.v[i].data;
    for(int i=1;i<=m;i++)
        L.v[i].Vh=NULL;
    for(int i=1;i<=m;i++)
    {
        int v1,v2,weight;
        cin>>v1>>v2>>weight;
        ALink p=new ANode;
        p->v=v2;
        p->weight=weight;
        p->next=L.v[v1].Vh;
        L.v[v1].Vh=p;
    }
    return 0;
}
```

4-7 请给出有向图的

(1) 每个顶点的入度和出度；

(2) 邻接矩阵；

(3) 邻接表。



答：

4-7 (1)

点	入度	出度
V_1	3	0
V_2	2	2
V_3	1	1
V_4	1	3
V_5	2	1
V_6	1	3

(2) 邻接矩阵：

	V_1	V_2	V_3	V_4	V_5	V_6
V_1	0	0	0	0	0	0
V_2	1	0	0	1	0	0
V_3	0	1	0	0	0	0
V_4	0	0	1	0	1	1
V_5	1	0	0	0	0	0
V_6	1	1	0	0	1	0

(3) 邻接表：

V_1

$V_2 \rightarrow V_1 \rightarrow V_4$

$V_3 \rightarrow V_2$

$V_4 \rightarrow V_3 \rightarrow V_5 \rightarrow V_6$

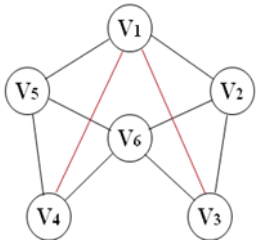
$V_5 \rightarrow V_1$

$V_6 \rightarrow V_1 \rightarrow V_2 \rightarrow V_5$

4-8 设无向图 G=(V，E)，V={a，b，c，d，e，f}，E={(a，b)，(a，e)，(a，c)，(b，e)，(c，f)，(f，d)，(e，d)}。从顶点 a 出发对图 G 进行深度优先搜索遍历，得到的顶点序列是(　D　)。

(A) a b e c d f
(B) a c f e b d
(C) a e b c f d
(D) a e d f c b

4-9 假设 v_1 为出发点，优先考虑编号的顶点。试给出无向图的



(1) 深度优先遍历的顶点序列和边序列；
(2) 广度优先遍历的顶点序列和边序列。

答：

（1）

顶点序列:v1 v2 v3 v6 v4 v5

边序列:e(v1,v2),e(v2,v3),e(v3,v6),e(v6,v4),e(v4,v5)

（2）

顶点序列:v1 v2 v3 v4 v5 v6

边序列:e(v1,v2),e(v1,v3),e(v1,v4),e(v1,v5),e(v2,v6)

4-10 概念解释：最小生成树。

答：无向图 G 有 n 个顶点，生成树是含有无向图 G 的 n 个顶点，n-1 条边的一个连通图。

最小生成树是无向图 G 的所有生成树中各边权值之和最小的一颗生成树。

4-11 设无向图 G=(V, E)，V={a, b, c, d, e}，E={<a, b>, <a, c>, <a, d>, <b, c>, <c, e>, <d, e>}，G1=(V, E1)。如果 G1 是 G 的生成树，则错误的是(D)。

(A) E1={<a, b>，<a, c>，<a, d>，<c, e>}

(B) E1={<a, b>，<a, c>，<c, e>，<d, e>}

(C) E1={<a, c>，<b, c>，<c, e>，<d, e>}

(D) E1={<a, d>，<b, c>，<c, d>，<d, e>}

4-12 判断一个有向图是否存在回路，除了可以利用深度优先遍历算法外，还可以利用(C)。

(A) 广度优先遍历算法

(B) 求最短路径的方法

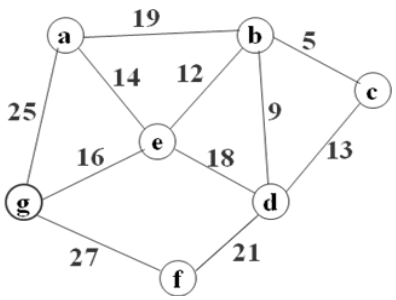
(C) 拓扑排序方法（找无前驱删点输出）

(D) 求关键路径的方法

4-13 设带权无向图 G =(V, E)含有 n 个顶点 m 条边。试描述构造图 G 的最小生成树的克鲁斯卡尔(Kruskal)算法。

答：将 m 条边按权值从小到大排序，从最小的边开始，若加入这条边，图不产生回路，则加入这条边，否则不加入。直至加入 n-1 条边，算法结束。

4-14 假设依据 Prim 算法产生无向网的最小生成树，出发顶点为 a，则被选择的顶点序列是(D)。



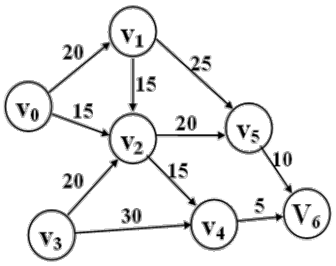
(A) a→b→c→d→e→f→g

(B) a→b→e→g→c→d→f

(C) a→e→d→b→c→f→g

(D) a→e→b→c→d→g→f

4-15 在有向图中，路径(　　C　　)是从 v0 出发的一条最短路径。



(A) $v_0 \rightarrow v_1 \rightarrow v_5$
(B) $v_0 \rightarrow v_2 \rightarrow v_3$
(C) $v_0 \rightarrow v_2 \rightarrow v_4$
(D) $v_0 \rightarrow v_2 \rightarrow v_5 \rightarrow v_6$

4-16 采用邻接表存储结构，设计一个算法，判别无向图 G 中指定的两个顶点之间是否存在一条长度为 k 的简单路径。

注：简单路径是指顶点序列中不含有重复的顶点。

答：

```
/* 简单路径的寻找，判断两个顶点是否存在长度为 k 的简单路径 */
#include <iostream>
using namespace std;
#define MAXSIZE 100
typedef struct ALink {
    int Vi;
    int Wi;
    struct ALink *next;
}ALink;
typedef struct {
    char data;
    ALink *Vh;
}VNode;
typedef struct {
    VNode v[MAXSIZE];
    int n;              // 顶点个数
    int m;              // 边数量
    int isVisit[MAXSIZE];   // 访问标识
}AList;
void InitialAList(AList &L)
{
    cout << "----------您正在建立邻接表----------" << endl;
    cout << "请输入您想要建立的顶点总数: ";
    cin >> L.n;
    for (int i = 1; i <= L.n; i++)
    {
        cout << "请输入第" << i << "个顶点的数据:";
        cin >> L.v[i].data;
        L.v[i].Vh = NULL;
        L.isVisit[i] = 0;
```

```
    }
    cout << "请输入您想要建立的关系总数(边数):";
    cin >> L.m;
    for (int i = 1; i <= L.m; i++)
    {
        char ch1, ch2;
        int v1, v2,weight;
        cout << "------您正在建立第" << i << "条关系------" << endl;
        cout << "请输入建立关系的第一个顶点数据: ";
        cin >> ch1;
        cout << "请输入建立关系的第二个顶点数据: ";
        cin >> ch2;
        cout << "请输入两顶点之间的权值: ";
        cin >> weight;
        for (int j = 1; j <= L.n; j++)
        {
            if (ch1 == L.v[j].data)
                v1 = j;
            if (ch2 == L.v[j].data)
                v2 = j;
        }
        ALink *p1 = new ALink;                  // 创建结点，加入邻接表中
        p1->Vi = v2;
        p1->Wi = weight;
        p1->next = L.v[v1].Vh;
        L.v[v1].Vh = p1;
        ALink *p2 = new ALink;                  // 创建结点，加入邻接表中
        p2->Vi = v1;
        p2->Wi = weight;
        p2->next = L.v[v2].Vh;
        L.v[v2].Vh = p2;
    }
    cout << "----------邻接表建立完成----------" << endl;
}
void OutputAList(AList &L)
{
    for (int i = 1; i <= L.n; i++)
    {
        ALink *p = L.v[i].Vh;
        cout << L.v[i].data << ":"<<endl;
        while (p)
        {
            cout << L.v[i].data << "—" << L.v[p->Vi].data << ":" << p->Wi <<
endl;
            p = p->next;
        }
    }
```

```
    return;
}
void InitVisited(AList &L)
{
    for (int i = 1; i <= L.n; i++)        // 初始化标记数组
        L.isVisit[i] = 0;
    return;
}
int DFS_RoadK(AList &L, int v1, int v2, int k, int x)
{
    if (x == k && v1 == v2)
        return 1;
    if (x != k && v1 == v2)
        return 0;
    L.isVisit[v1] = 1;
    ALink *p = L.v[v1].Vh;
    while (p)
    {
        if (L.isVisit[p->Vi] == 0)
        {
            L.isVisit[p->Vi] = 1;
            if (DFS_RoadK(L, p->Vi, v2, k, x + 1) == 1)
                return 1;
            L.isVisit[p->Vi] = 0;
        }
        p = p->next;
    }
    return 0;
}
int SearchRoad_K(AList &L)
{
    char ch1, ch2;
    int v1, v2, k;
    cout << "请输入您想要查询简单路径的第一个顶点: ";
    cin >> ch1;
    cout << "请输入您想要查询简单路径的第二个顶点: ";
    cin >> ch2;
    cout << "请输入您想要查询的简单路径的距离 k:";
    cin >> k;
    for (int i = 1; i <= L.n; i++)
    {
        if (L.v[i].data == ch1)
            v1 = i;
        if (L.v[i].data == ch2)
            v2 = i;
    }
    InitVisited(L);
    return DFS_RoadK(L, v1, v2, k, 0);
```

```
}
void Search_Road_Process(AList &L)
{
    cout << "------正在执行简单路径查询功能------" << endl;
    cout << "请输入您想要查询的次数:";
    int times;
    cin >> times;
    for (int i = 1; i <= times; i++)
    {
        cout << "------正在执行第" << i << "次查询操作------" << endl;
        if (SearchRoad_K(L) == 1)
            cout << "存在" << endl;
        else
            cout << "不存在" << endl;
    }
    return;
}
void OutputRoad(AList &L)
{
    cout << "下面输出各个顶点之间所有简单路径的距离:" << endl;
    for (int i = 1; i <= L.n; i++)
    {
        for (int j = i + 1; j <= L.n; j++)
        {
            cout << L.v[i].data << "-" << L.v[j].data << ":";
            for (int k = 1; k <= 10; k++)
            {
                InitVisited(L);
                if (DFS_RoadK(L, i, j, k, 0) == 1)
                    cout << k << " ";
            }
            cout << endl;
        }
    }
    return;
}
int main()
{
    AList L;
    InitialAList(L);
    OutputAList(L);
    OutputRoad(L);
    Search_Road_Process(L);
    return 0;
}
```

4-17 设带权有向图 G =(V, E)含有 n 个顶点、e 条边，采用邻接矩阵 Graph[n][n]作为存储结构。试设计算法 Dijkstra(int V_0，int n)，用于计算从源点 V_0 到其它

各顶点的最短路径。

答：

```
//最短路径(dijkstra算法)
#include <iostream>
#define MAXSIZE 100
using namespace std;
typedef struct {
    int n;          // 顶点数
    int m;          // 边数
    char data[MAXSIZE];   //数据元素
    int Vr[MAXSIZE][MAXSIZE];     //邻接矩阵
}Amatrix;
typedef struct {
    int n;       //顶点数
    int Wi[MAXSIZE];   //最短路径权值之和
    int Vi[MAXSIZE];     //访问序列
    int isVisited[MAXSIZE]; //访问标识，1为已经访问，0为未访问
}WeightMatrix;
void Init_Amatrix_WeightMatrix(Amatrix &M, WeightMatrix &W)
{
    cout << "请输入顶点数:";
    cin >> M.n;
    W.n = M.n;
    for (int i = 1; i <= M.n; i++)
    {
        cout << "请输入第" << i << "个数据元素: ";
        cin >> M.data[i];
    }
    cout << "请输入边数:";
    cin >> M.m;
    for (int i = 1; i <= M.n; i++)
    {
        for (int j = 1; j <= M.n; j++)
        {
            M.Vr[i][j] = 0;
        }
    }
    for (int i = 1; i <= M.m; i++)
    {
        char ch1, ch2;
        int v1, v2, weight;
        cout << "------正在建立第" << i << "条关系------" << endl;
        cout << "请输入第一个顶点:";
        cin >> ch1;
        cout << "请输入第二个顶点:";
        cin >> ch2;
```

```
            cout << "请输入权值:";
            cin >> weight;
            for (int j = 1; j <= M.n; j++)
            {
                if (ch1 == M.data[j])
                    v1 = j;
                if (ch2 == M.data[j])
                    v2 = j;
            }
            M.Vr[v1][v2] = weight;
            M.Vr[v2][v1] = weight;
        }
        for (int i = 1; i <= W.n; i++)
        {
            W.Wi[i] = 0;
            W.isVisited[i] = 0;
            W.Vi[i] = i;
        }
        return;
}
void OutputAmatrix(Amatrix &M, WeightMatrix &W)
{
        for (int i = 1; i <= M.n; i++)
        {
            for (int j = 1; j <= M.n; j++)
            {
                cout << M.Vr[i][j] << " ";
            }
            cout << endl;
        }
        return;
}
int FindLocation(Amatrix &M, char ch)
{
        for (int i = 1; i <= M.n; i++)
        {
            if (M.data[i] == ch)
                return i;
        }
        return 0;
}
void Dijkstra(Amatrix &M, WeightMatrix &W,char ch)
{
        int v = FindLocation(M,ch);
        W.Wi[v] = 0;
        W.isVisited[v] = 1;
        W.Vi[v] = v;
        for (int i = 1; i <= M.n - 1; i++)
```

```
    {
        for (int j = 1; j <= M.n; j++)
        {
            if (W.isVisited[j] == 0 && M.Vr[v][j]!=0&&((W.Wi[v] + M.Vr[v][j] <
W.Wi[j])||W.Wi[j]==0))
            {
                W.Wi[j] = W.Wi[v] + M.Vr[v][j];
                W.Vi[j] = v;
            }
        }
        int min, number;
        int flag = 1;
        for (int j = 1; j <= M.n; j++)
        {
            if (W.isVisited[j] == 0&&W.Wi[j]!=0)
            {
                if (flag == 1)
                {
                    min = W.Wi[j];
                    number = j;
                    flag = 0;
                }
                else
                {
                    if (W.Wi[j] < min)
                    {
                        min = W.Wi[j];
                        number = j;
                    }
                }

            }
        }
        W.isVisited[number] = 1;
        v = number;
    }
    return;
}
void OutputDijkstra(Amatrix &M, WeightMatrix &W, char ch)
{
    int v = FindLocation(M,ch);
    int num = 0;
    for (int i = 1; i <= M.n; i++)
    {
        if (i == v)
            continue;
        num++;
        cout << "Path" << num << ":";
```

```
            int a[MAXSIZE];
            int num1 = 0;
            int v1 = i;
            num1++;
            a[num1] = v1;
            while (v1 != v)
            {
                v1 = W.Vi[v1];
                num1++;
                a[num1] = v1;
            }
            for (int j = num1; j >= 1; j--)
            {
                if (j != num1)
                    cout << "->";
                cout << M.data[a[j]];
            }
            cout << " (" << W.Wi[i] << ")" << endl;
        }
        return;
}
void Process_dijkstra(Amatrix &M, WeightMatrix &W)
{
        char ch;
        cout << "请输入想要作为起点的元素:";
        cin >> ch;
        Dijkstra(M, W, ch);
        OutputDijkstra(M, W, ch);
        return;
}
int main()
{
        Amatrix M;
        WeightMatrix W;
        Init_Amatrix_WeightMatrix(M, W);
        OutputAmatrix(M, W);
        Process_dijkstra(M, W);
        return 0;
}
```

4-18 设软件工程专业开设的主要课程如表所示：

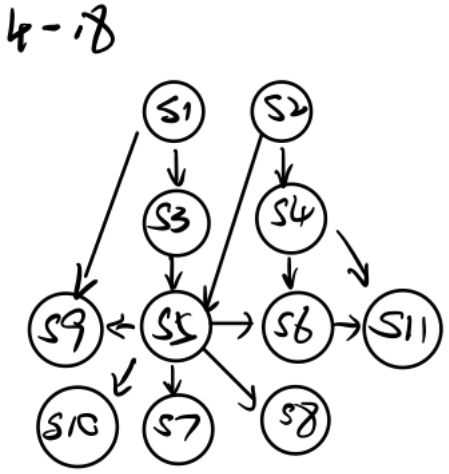
代码	课程名称	先修课程
S1	高等数学	无
S2	程序设计基础	无
S3	离散数学	S1
S4	计算机组成原理	S2
S5	数据结构与算法	S2, S3

代码	课程名称	先修课程
S7	数据库系统	S5
S8	编译技术	S5
S9	算法分析	S1, S5
S10	软件工程导论	S5
S11	计算机网络	S4, S6

S6	操作系统	S4, S5			

试根据先修课程要求绘制课程体系拓扑结构图(结点用课程代码表示)。

答;



4-19 设含有 6 个顶点 a, b, c, d, e, f 的有向带权图 G，其邻接矩阵如下：

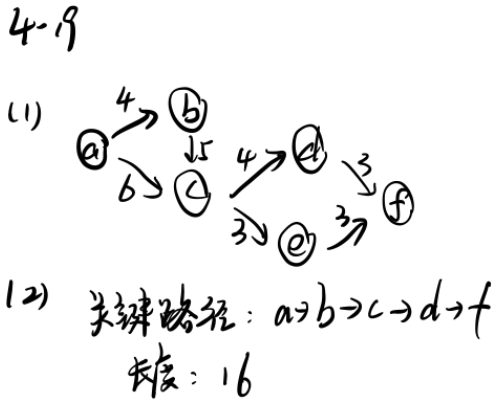
\	a	b	c	d	e	f
a	0	4	6	0	0	0
b	0	0	5	0	0	0
c	0	0	0	4	3	0
d	0	0	0	0	0	3
e	0	0	0	0	0	3
f	0	0	0	0	0	0

要求：

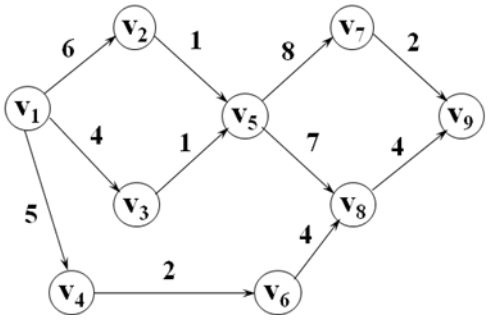
(1)画出有向带权图 G;

(2)求图 G 的关键路径，并计算关键路径长度。

答：



4-20 设 v_1 是源点、v_9 是汇点，则在有向图中，(C)是一条关键路径。



(A) $v_1 \rightarrow v_4 \rightarrow v_6 \rightarrow v_8 \rightarrow v_9$

(B) $v_1 \rightarrow v_3 \rightarrow v_5 \rightarrow v_7 \rightarrow v_9$

(C) $v_1 \rightarrow v_2 \rightarrow v_5 \rightarrow v_8 \rightarrow v_9$

(D) $v_1 \rightarrow v_2 \rightarrow v_5 \rightarrow v_7 \rightarrow v_9$