# 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

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(n2)

(C) O(ne)

(D) O(n3)

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>}。

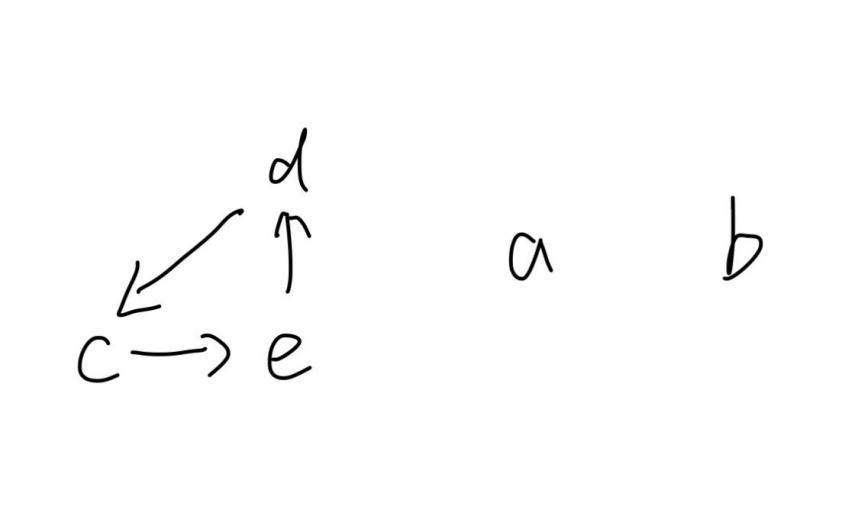
(1)是否存在从c到b的路径?

**不存在从c到b的路径**

(2)计算ID(d)、OD(d)、TD(d)；

ID(d) = 2、OD(d) = 1、TD(D) = 3

(3)画出各个强连通分量。



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

//定义结构体模板，用于储存图中个顶点边的信息

template <class w>

struct Edge

{

    int dstPos = -1;

    w weight; //权值

    Edge<w> \*next = nullptr;

    //结构体构造函数，引用该函数可以给定义的结构体进行数值初始化的操作

    Edge(int \_dstPos, const w &\_weight) : dstPos(\_dstPos), weight(\_weight), next(nullptr) {}

};

// v:节点的值，w:节点的权值  若flag == false，则表示该图为无向图

template <class v, class w, bool flag = false>

class linkTable {

    //定义结构体操作

    typedef Edge<w> Edge;

    private:

        vector<Edge \*> \_matrix;             //结构体指针向量，用来存邻接表

        unordered\_map<v, int> \_indexMap;    //定义键值对保存图节点对应邻接表数组的下标

        vector<v> \_points;                  //顶点集合

        int \_getPointsPos(const v &point) {

            typename unordered\_map<v, int>::iterator pos = \_indexMap.find(point);

            //若没找到，则返回-1

            if (pos == \_indexMap.end())

                return -1;

            //找到了，则返回键值对中的值

            return pos->second;

        }

    public:

        //创建邻接表

        linkTable(const vector<v> &src) {

            int size = src.size();

            //断言处理，保证键值对的键值大于0，防止越界出现

            assert(size > 0);

            //重新调整顶点集合大小

            \_points.resize(size);

            //遍历容器，对容器内元素进行赋值操作，并存下每个元素的键值对

            for (int i = 0; i < size; i++) {

                \_points[i] = src[i];

                \_indexMap[src[i]] = i;

            }

            \_matrix.resize(size, nullptr);

        }

        //添加图中顶点之间的边关系

        void AddEdge(const v &src, const v &dst, const w &weight) {

            //寻找两顶点在哈希表中的位置

            int posSrc = \_getPointsPos(src);

            int posDst = \_getPointsPos(dst);

            //创建新边

            Edge \*edge = new Edge(posDst, weight);

            //先将新边的next指针指向了原先在'posSrc'位置的链表头

            edge->next = \_matrix[posSrc];

            //再将\_matrix[posSrc]指向新创建的边，完成边在邻接链表中的添加

            \_matrix[posSrc] = edge;

            //若为无向图

             if (!flag) {

                edge = new Edge(posSrc, weight);

                edge->next = \_matrix[posDst];

                \_matrix[posDst] = edge;

            }

        }

};

int main() {

    //初始化visited数组

    for (int i = 0; i < MAX; ++i)

        visited[i] = false;

    //定义用户输入的顶点个数，同样也是哈希表中最后一个元素的键值

    int vernum;

    cout << "请输入顶点个数：";

    cin >> vernum;

    cout << "请输入顶点，用空格隔开，输入回车表示输入结束" << endl;

    //存输入的顶点字母

    char a;

    //定义容器来存储输入的顶点字母，用来创建邻接表

    vector<char> ver;

    //遍历存储

    for (int i = 0; i < vernum; ++i) {

        cin >> a;

        //消除空格的影响

        cin.ignore();

        ver.push\_back(a);

    }

    //记录第一个顶点

    char first = ver[0];

    //记录顶点数组长度,用于插入操作

    int ver1 = ver.size();

    int flag = -1;

    //提示用户想要输出有向图还是无向图

    cout << "想要输出无向图输入1,想要输出有向图输入0" << endl;

    cin >> flag;

    //消除回车影响

    cin.ignore();

    //提示用户输入顶点之间的边关系与权值

    cout << "请输入需要添加边关系的两顶点并输入边的权值，每组数据之间用分号隔开，每组数据的两个顶点以及边的权值之间用空格隔开，输入回车表示输入结束，具体输入样例如下" << endl;

    cout << "a b 1; c d 2; e f 3(回车)" << endl;

    cout << "以上样例表示a与b、c与d、e与f之间分别建立边关系并确认权值" << endl;

    linkTable<char, int, true> graph2(ver);

        char tmp = '0';

        while (tmp != '\n') {

            int weight1;     //边的权值

            char p, q;       //两个顶点

            scanf ("%c %c %d;", &p, &q, &weight1);

            //tmp用于吸收空格并判断是否为回车

            tmp = getchar();

            //添加两顶点边的关系

            graph2.AddEdge(p, q, weight1);

        }

        cout << "有向图" << endl;

        graph2.PrintGraph();

}

4-7 请给出有向图的

(1) 每个顶点的入度和出度；

ID(V1) = 3 OD(V1) = 0

ID(V2) = 2 OD(V2) = 2

ID(V3) = 1 OD(V3) = 1

ID(V4) = 1 OD(V4) = 3

ID(V5) = 2 OD(V5) = 1

ID(V6) = 1 OD(V6) = 3

(2) 邻接矩阵；

V1 V2 V3 V4 V5 V6

V1 0 0 0 0 0 0

V2 1 0 0 1 0 0

V3 0 1 0 0 0 0

V4 0 0 1 0 1 1

V5 1 0 0 0 0 0

V6 1 1 0 0 1 0

(3) 邻接表。

V1

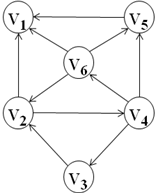
V2->V1->V4

V3->V2

V4->V3->V5->V6

V5->V1

V6->V1->V2->V5



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

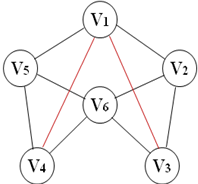
4-9 假设v1为出发点，优先考虑编号的顶点。试给出无向图的

(1)深度优先遍历的顶点序列和边序列；

**V1->V2->V3->V6->V4->V5**

(2)广度优先遍历的顶点序列和边序列。

**V1->V2->V3->V4->V5->V6**



4-10 概念解释：最小生成树。

**在一个连通网的所有生成树中，代价之和最小的生成树**

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

(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。

**1.将G的n个顶点看成n个孤立的连通分支，将所有的边按权从小到大排序**

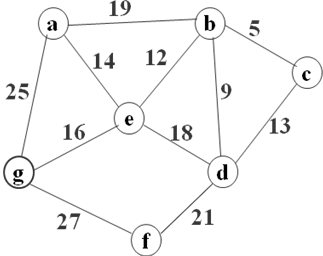
**2.从第一条边开始，依边权递增的顺序查看每一条边，并按下述方法连接2个不同的连通分支：**

**（1）当查看到第k条边(v,w)时，如果端点v和w分别是当前2个不同的连通分支T1和T2中的顶点时，就用边(v,w)将T1和T2连接成一个连通分支，然后继续查看第k+1条边；**

**（2）如果端点v和w在当前的同一个连通分支中，就直接再查看第k+1条边。**

**3.这个过程一直进行到只剩下一个连通分支时为止，选取到的所有边恰好构成G的一棵最小生成树**

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** )。



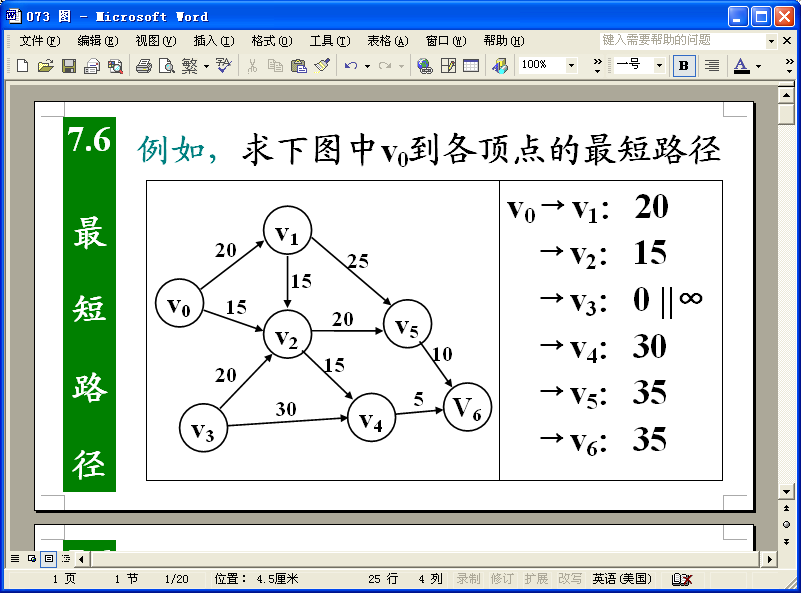
(A) a→b→c→d→e→f→g

(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) v0→v1→v5

(B) v0→v2→v3

(C) v0→v2→v4

(D) v0→v2→v5→v6

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。

int visited[MAXSIZE];

int exist\_path\_len(ALGraph G,int i,int j,int k) {

    if (i==j && k==0) //找到了一条路径,且长度符合要求

        return true;

    else if (k > 0) {

        visited[i]=1;

        for(p=G.vertices[i].firstarc; p!=NULL; p=p->nextarc) {

            m=p->adjvex;

            if(!visited[m])

            if(exist\_path\_len(G,m,j,k-1)) return true; //剩余路径长度减一

        }

        visited[i]=0;

    //这里需要把已经访问的点重新置为0，因为如果当前不存在长度为k

    }

    return false; //没找到

}

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 V0，int n)，用于计算从源点V0到其它各顶点的最短路径。

void Dijkstra(vector<vector<int> > g) {

    int vex = g.size() - 1; //顶点数

    int start;  //计算最短距离的起点

    cout << "输入计算最短路径的点：";

    cin >> start;

    vector<Point> points(vex + 1);  //记录点的信息

    for (int i = 1; i <= vex; i++)

        points[i].dis = g[start][i];    //初始化dis，记录起点到各点的直接距离

    points[start].visit = true;     //起点访问

    for (int i = 1; i <= vex; i++) {

        int MINdis = INT\_MAX;

        int MINvex = 0;

        for (int j = 1; j <= vex; j++) {

            if (!points[j].visit && points[j].dis < MINdis) {   //找出距离最近的点

                MINdis = points[j].dis;

                MINvex = j;

            }

        }

        points[MINvex].visit = true;    //距离最近的点的最短距离确定

        for (int i = 1; i <= vex; i++)  //更新dis数组

            if(g[MINvex][i]<INT\_MAX)

                points[i].dis = min(points[i].dis, g[MINvex][i] + points[MINvex].dis);

    }

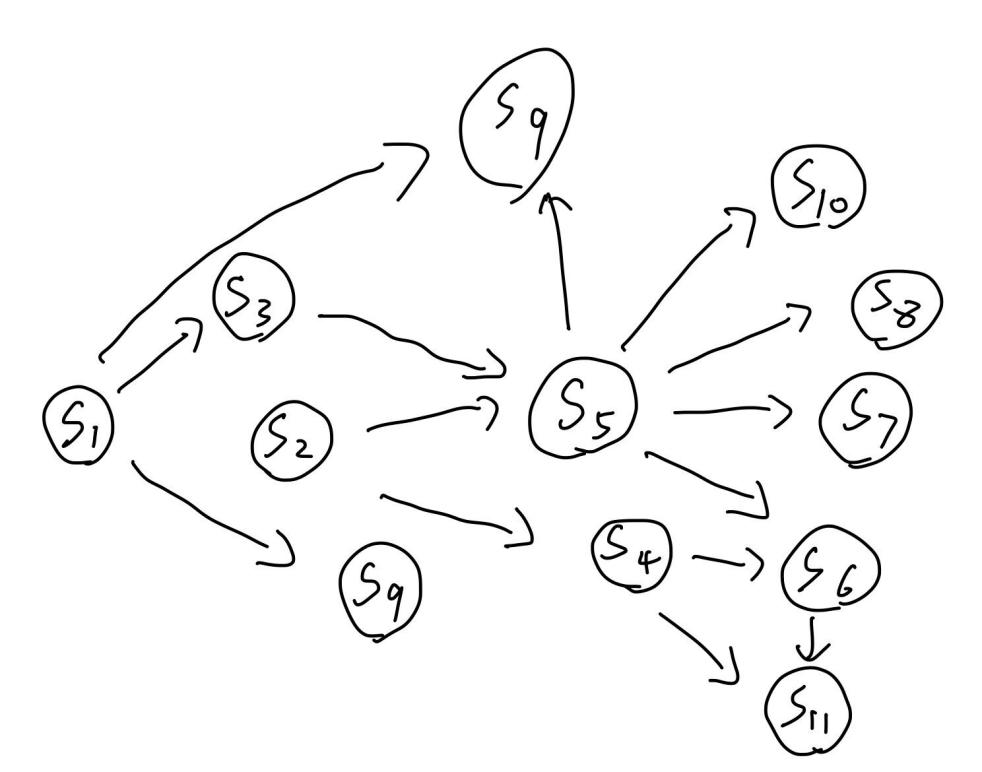
    output(points, start);  //输出结果

}

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 |  | 代码 | 课程名称 | 先修课程 |
| S1 | 高等数学 | 无 |  | S7 | 数据库系统 | S5 |
| S2 | 程序设计基础 | 无 |  | S8 | 编译技术 | S5 |
| S3 | 离散数学 | S1 |  | S9 | 算法分析 | S1, S5 |
| S4 | 计算机组成原理 | S2 |  | S10 | 软件工程导论 | S5 |
| S5 | 数据结构与算法 | S2, S3 |  | 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)。

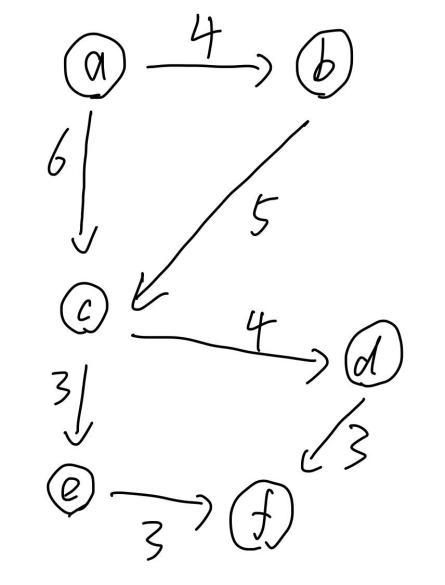


4-19 设含有6个顶点a, b, c, d, e, f的有向带权图G，其邻接矩阵如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ＼ | 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；

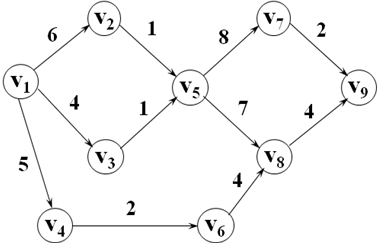


(2)求图G的关键路径，并计算关键路径长度。

**关键路径：a->b->c->d->f**

**关键路径长度：16**

4-20 设v1是源点、v9是汇点，则在有向图中，( **C** )是一条关键路径。



(A) v1→v4→v6→v8→v9

(B) v1→v3→v5→v7→v9

(C) v1→v2→v5→v8→v9

(D) v1→v2→v5→v7→v9