**《数据结构与算法》作业**

# 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( B )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( B  )。

(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)；

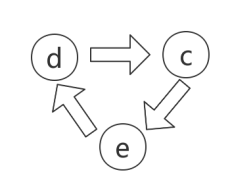
(3)画出各个强连通分量。

答：

1. 不存在

(2)ID(d)=2、OD(d)=1、TD(d)=3；

(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。

void insertedge(ALGragh& G, int i, int j)//插入一条边,头插

{

AdjNode\* s = (AdjNode\*)malloc(sizeof(AdjNode));

s->v = j;

s->next = NULL;

if (G.Vex[i].first == NULL)

{

G.Vex[i].first = s;

G.Vex[i].last = s;

}

else

{

G.Vex[i].last->next = s;

G.Vex[i].last = s;

}

}

void CreateALGraph(ALGragh& G)//创建有向图邻接表

{

int i, j;

VexType u, v;

VexType s[MaxVnum];

printf("请输入顶点数：\n");

scanf("%d", &G.vexnum);

printf("请输入边数:\n");

scanf("%d", &G.edgenum);

printf("请输入顶点信息:\n");

scanf("%s", s);

for (i = 0; i < G.vexnum; i++)

{

G.Vex[i].data = s[i];

}

for (i = 0; i < G.vexnum; i++)

G.Vex[i].first = NULL;

printf("请输入每条边的两个顶点：\n");

int num = G.edgenum;

while (num--)

{

char str[3];

scanf("%s", str);

i = FindVex(G, str[0]);//查找顶点u的存储下标

j = FindVex(G, str[1]);//查找顶点v的存储下标

if (i != -1 && j != -1)

{

insertedge(G, i, j);

//insertedge(G, j, i);//无向图多插入一条边

}

else

{

printf("输入错误！请重新输入！\n");

num++;

}

}

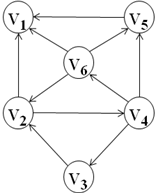
}

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。



1. 每个顶点的入度和出度：

V1： 入度为3，出度为0

V2： 入度为2，出度为2

V3： 入度为1，出度为1

V4： 入度为1，出度为3

V5： 入度为2，出度为1

V6： 入度为1，出度为3

1. 邻接矩阵：

0 0 0 0 0 0

1 0 0 1 0 0

0 1 0 0 0 0

0 0 1 0 1 1

1 0 0 0 0 0

1 1 0 0 1 0

1. 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

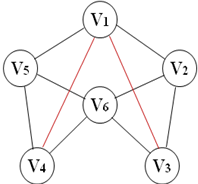
(C) a e b c f d

(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；

(2)广度优先遍历的顶点序列和边序列。



答：

1. 深度优先遍历的顶点序列和边序列

v1, v2, v3, v6, v4, v5

v1-v2, v2-v3, v3-v6, v6-v4, v4-v5

1. 广度优先遍历的顶点序列和边序列。

v1, v2, v3, v4, v5, v6

v1-v2, v1-v3, v1-v4, v1-v5,v2-v6

4-10 概念解释：最小生成树。

在一给定的无向图G = (V, E) 中，(u, v) 代表连接顶点 u 与顶点 v 的边，而 w(u, v) 代表此边的权重，若存在 T 为 E 的子集且为无循环图，使得联通所有结点的的 w(T) 最小，则此 T 为 G 的最小生成树。

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

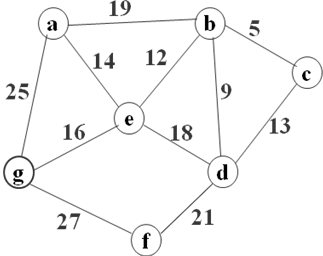
(C) 拓扑排序方法

(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。

按照权值从小到大的顺序选择n-1条边，并保证这n-1条边不构成回路。克鲁斯卡尔算法在所有连接森林的两个不同树的边里面，寻找权值最小的边将最小的边加入一个不相交的集合，每次加入时判断该边的起始顶点与结束顶点是否属于同一个树，即是否使森林产生回路，如果不产生回路则加入集合，最后即构成最小生成树。

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 )。



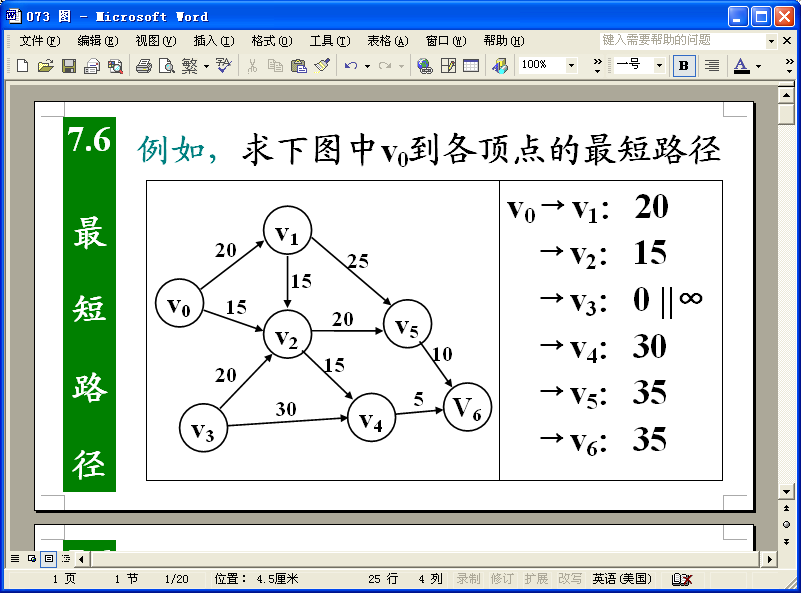
(A) a→b→c→d→e→f→g

(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。

typedef struct AdjNode { //定义邻接点类型

int v; //邻接点下标

struct AdjNode\* next; //指向下一个邻接点

}AdjNode;

typedef struct VexNode { //定义顶点类型

VexType data; // VexType为顶点的数据类型，根据需要定义

AdjNode\* first = NULL; //指向第一个邻接点

AdjNode\* last = NULL; //指向最后一个邻接点

}VexNode;

typedef struct {//定义邻接表类型

VexNode Vex[MaxVnum];

int vexnum;

int edgenum; //顶点数，边数

}ALGragh;

void DFS(ALGraph G, int i, int j, int k, int visited[] , bool Result)

{

static int d = 0;//记录当前路径的长度

visited[i] = 1; //访问标记

d++;

if (i == j && d == k + 1)//找到一条路径

Result = true;

else{

AdjNode\* p;

p = G.Vex[i].first; //指针指向第一个邻接点

while (!Result && p != NULL){

int w = p->v; // 若w结点未访问，递归的访问它

if (visited[w] == 0){

DFS(G, w, j, k,visited ,Result);

}

p = p->next;

}

}

visited[i] = 0;//回退操作

d--;

}

int SimplePath(ALGraph G, int i, int j, int k)

{

static bool Result = false;

DFS(G, i, j, k,visited,Result);

return Result;

}

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。

# define MAXV 1000; //最大顶点数

#define INF 1000000000; //设INF为一个很大的数

int n, G[MAXV][MAXV]; //n为顶点数

int d[MAXV]; //起点到达各点的最短路径长度

bool visit[MAXV] = {false}; //访问标记数组

void Dijkstra(int v0，int n) //v0为起点

{

fill(d, d + MAXV, INF); //将整个d数组赋为INF

d[v0] = 0;

for(int i = 0; i < n; i++)

{

int u = -1, MIN = INF; //u使d[u]最小，MIN存最小的d[u]

for(int j = 0; j < n; j++) //找到未访问的顶点中d[ ]最小的

{

if(visit[j] == false && d[j] < MIN)

{

u = j;

MIN = d[j];

}

}

if(u == -1)

return;

Visit[u] = true; //标记u为已访问

for(int v = 0; v < n; v++)

{

//如果v未访问 && u能到达v && 以u为中介点可以使d[v]更优

if(visit[v] == false && G[u][v] != INF && d[u] + G[u][v] < d[v])

{

d[v] = d[u] + G[u][v]; //优化d[v]

}

}

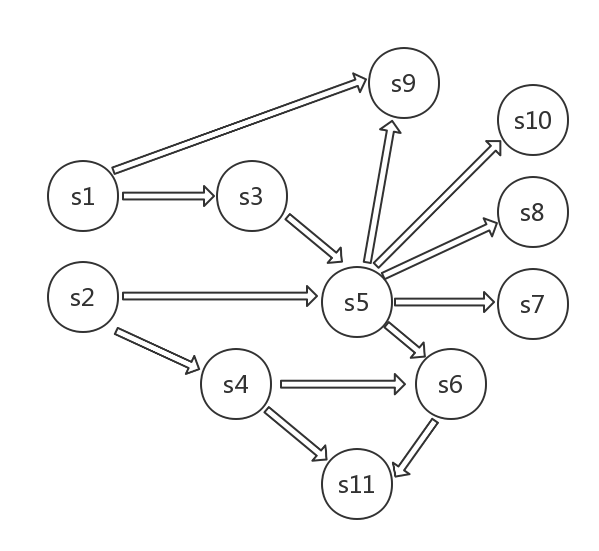
}

}

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 |

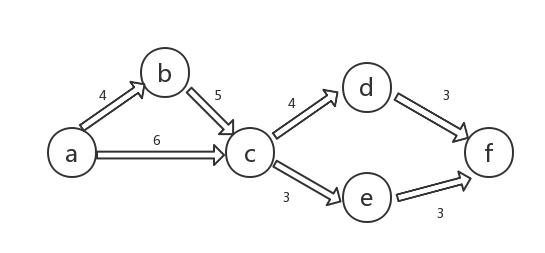
要求：

(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。

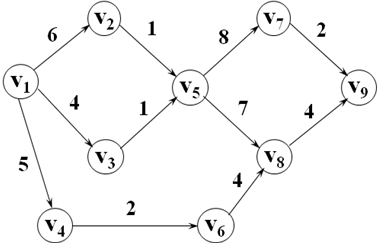
答：

(1):



(2): a->b->c->d->f, 长度为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