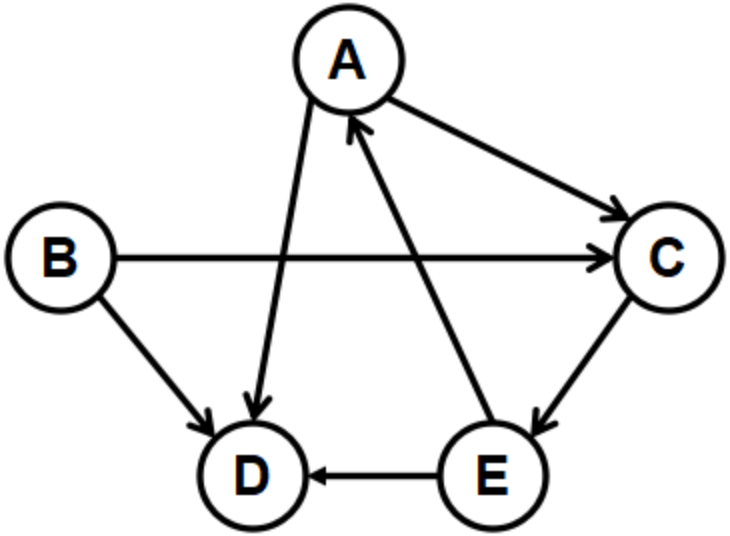
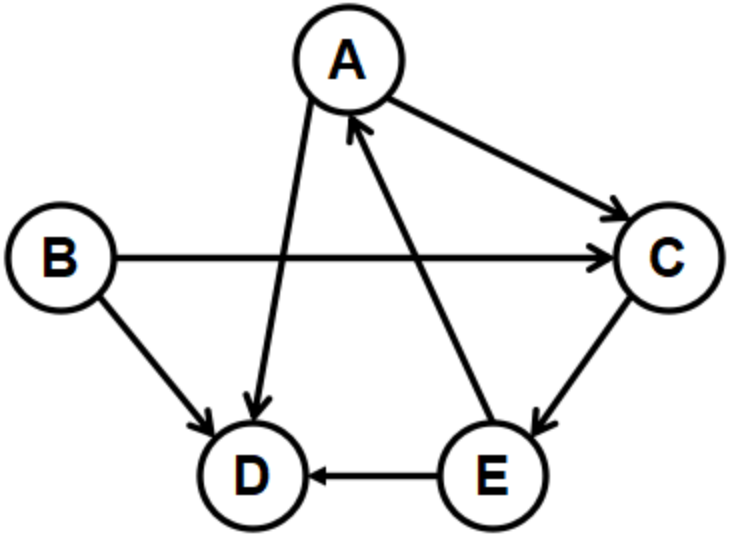
数据结构第6章习题

第1题

- 给出下图所示的邻接矩阵和邻接表

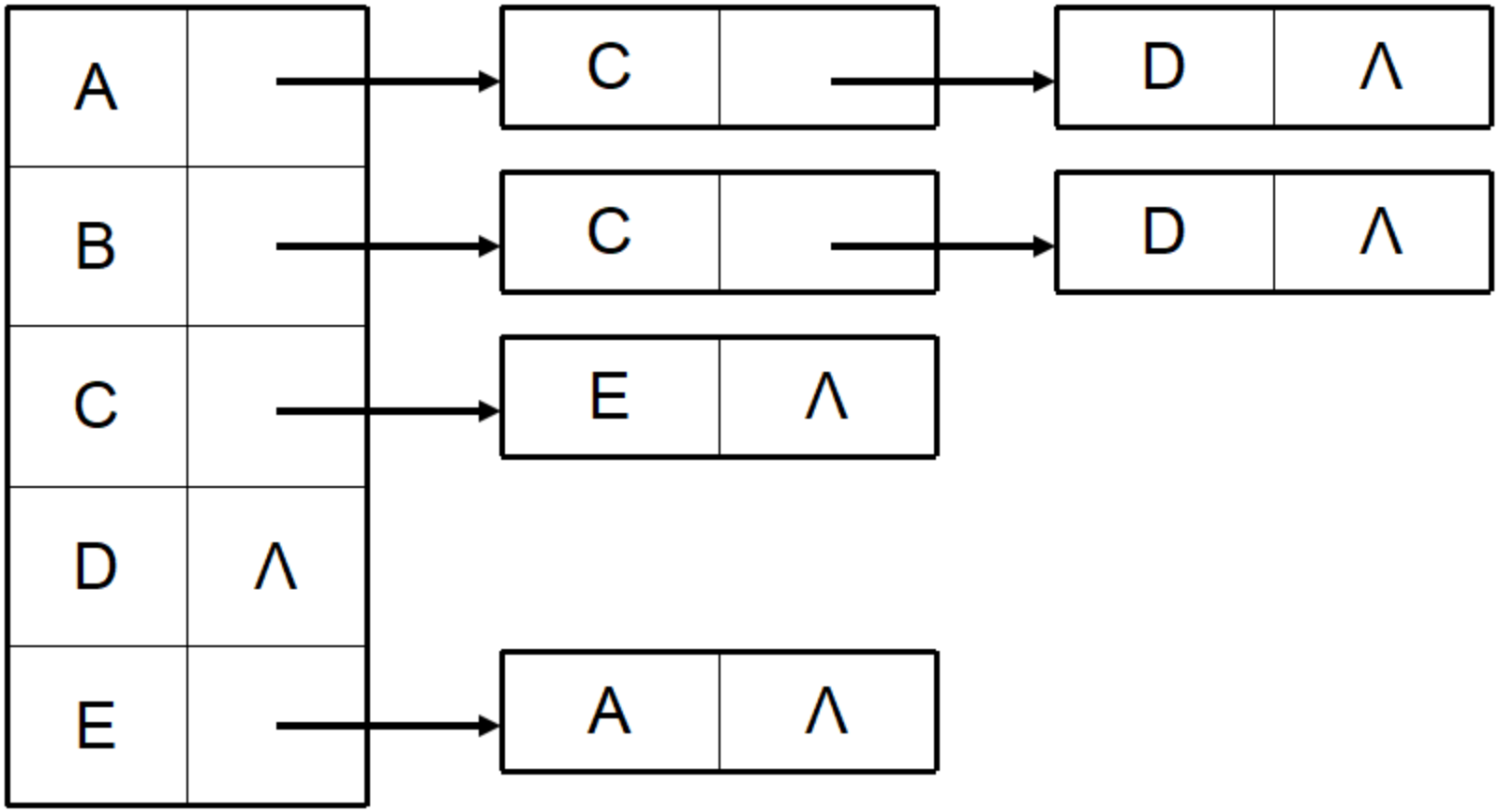


参考答案



	1	2	3	4	5
1	0	0	1	1	0
2	0	0	1	1	0
3	0	0	0	0	1
4	0	0	0	0	0
5	1	0	0	1	0

参考答案：



第2题

- 对于稠密图和稀疏图，分别采用什么存储结构比较好

参考答案

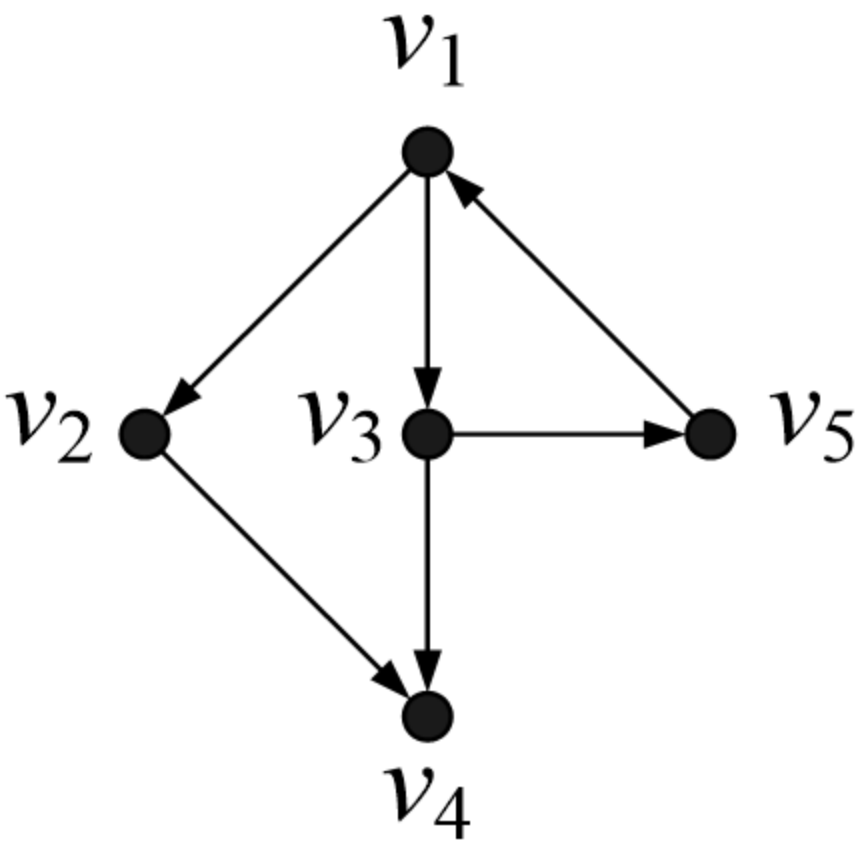
- 从空间利用率的角度考虑，稠密图采用邻接矩阵，稀疏图采用邻接表

第13题

- 设G＝(V, E)是有向图，请给出算法，判断G中是否有回路，并要求算法的复杂性为O(n+e)。

方法一：深度优先搜索

- 思想：深搜时，每个结点有两个状态，标记是否被访问过（0未访问，1已访问过）。判环时，多引入一个状态，标记结点正在访问中（-1正在访问中）。如果一个结点正在访问中，又遍历到该接点，那个存在环路。这种状况是由于出现了反向边，即后代指向祖先的边。
- 如果图中有多个连通分支，需要对每个连通分支都判断。


v_1
v_2
v_3
v_5
v_4

```
算法VCycle(v. flag) /*判断以v为起点的连通分支是
   否有环，若有，flag为TRUE,否则FALSE. */
visited(v) ← -1
p ←adjacent[v].
WHILE p ≠ Λ DO (
   IF visited(vertex(p))=-1 THEN (
        flag=TRUE. RETURN)
   IF visited(vertex(p))=0 THEN (
        VCycle(vertex(p). flag).
        IF flag=TRUE THEN RETURN.
   )
   p ← link(p).
)
visited(v) ← 1. flag ← FALSE. ▌
```

```
算法Cycle(G. flag)
Let v be the first vertex in G
WHILE v is existed DO (
    IF visited(v) = 1 THEN CONTINUE
    VCycle(v. flag)
    IF flag=TRUE THEN RETURN
    Let v be the next vertex
)
```

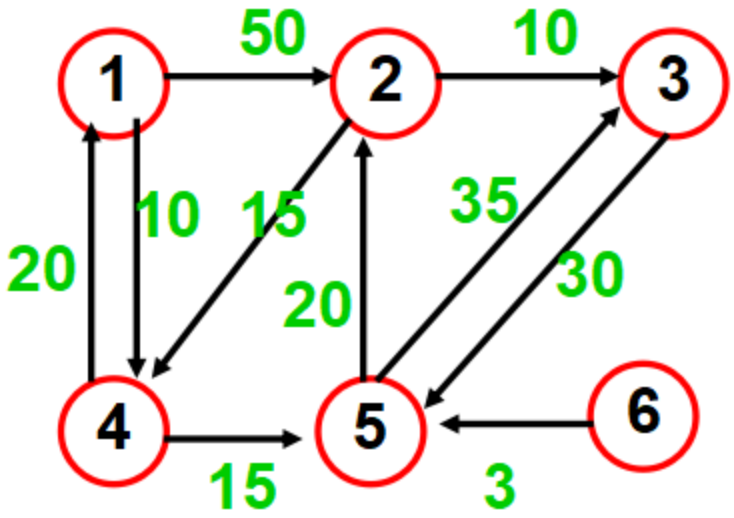
方法二：拓扑排序

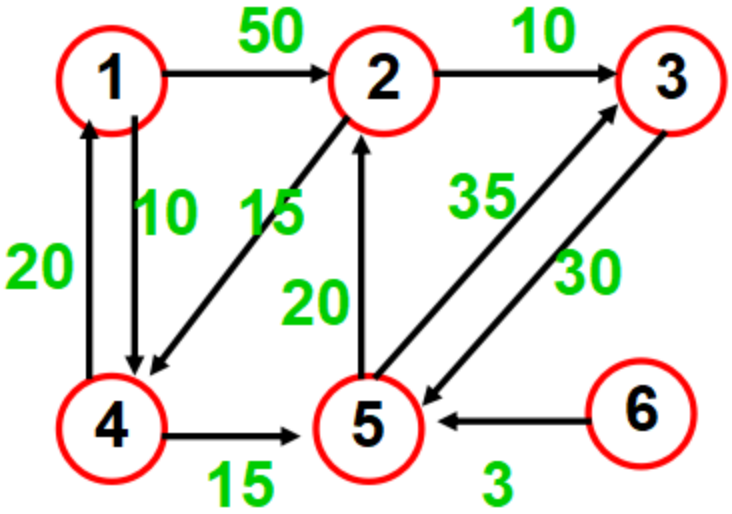
```
void Graph :: TopoOrder( )
{
    int top = - 1 ;
    for ( int i=0 ; i<n ; i + + )
        if ( count[i] = = 0 ) {
                count[i] = top ; top = i ;
        }
```

```
for ( int i=0 ; i<n ; i + + )
  if ( top = = - 1 ) {  //检测图中是否有回路
        cout << " There is a cycle in network ! " << endl;
        return ; }
  else {
     int j=top ; top=count[top] ;
     cout<<j<<endl ;   //输出栈顶j
     Edge *p = head[j].Adjacent ;
     while( p != NULL ) {      //把j的所有邻接顶点的入度减1
          int k=p->VerAdj ;
          if ( --count[k]= = 0 )  { count[k]=top ; top=k ; }
          p=p->link ;
      }
   }
}
```

第14题

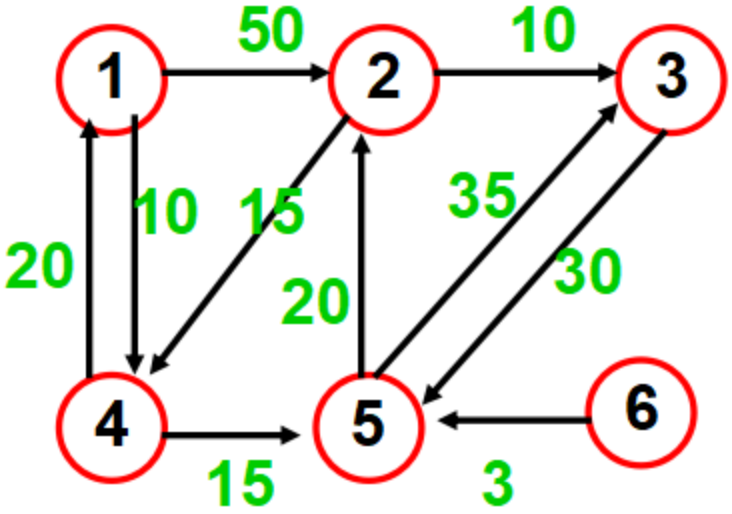
- 对于下图所示的非负有向网络，给出Dijkstra算法产生的由源点2到图中其它顶点的最短路径长度以及路径所经历的顶点，并写出生成过程。





参考答案

S 2	1	2	3	4	5	6
初始化	∞	0	∞	∞	∞	∞
2	∞	0	10	15	∞	∞
2 3	∞	0	10	15	40	∞
2 3 4	35	0	10	15	30	∞
2 3 4 5	35	0	10	15	30	∞
2 3 4 5 1	35	0	10	15	30	∞
2 3 4 5 1 6	35	0	10	15	30	∞



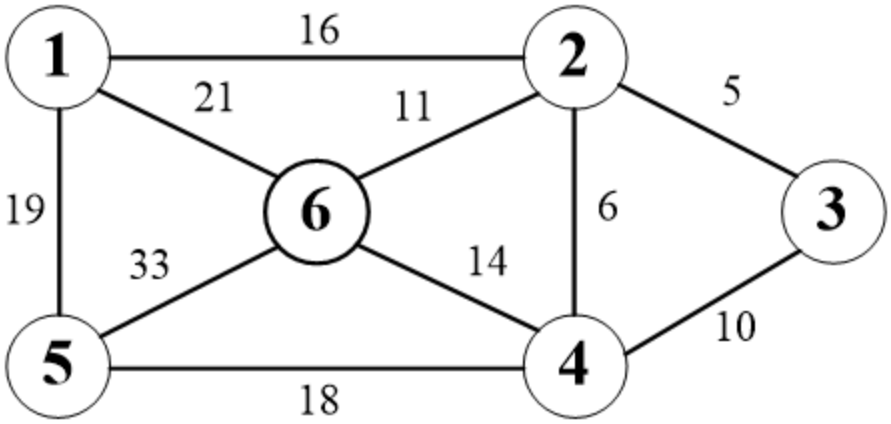
参考答案

	1	2	3	4	5	6
dis	35	0	10	15	30	∞
path	4	-1	2	2	4	-1
s	1	1	1	1	1	0

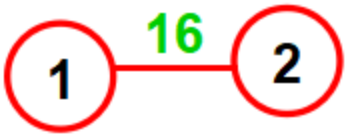
最短路径	最短路径长度
2→3	10
2→4	15
2→4→5	30
2→4→1	35
2→6	∞

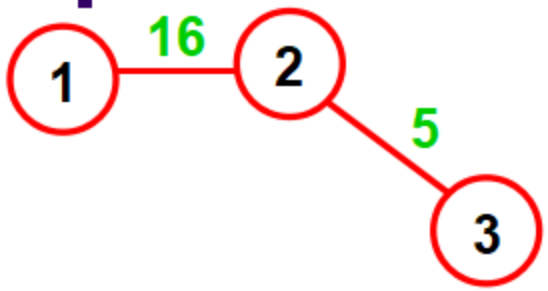
第16题

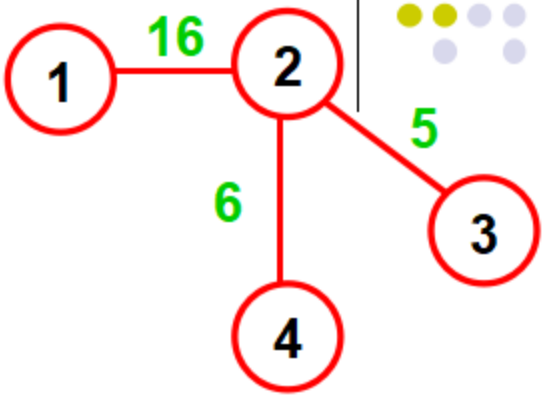
- 对于图5.39所示的无向网络，利用普里姆和克鲁斯卡尔算法，分别求出其最小支撑树，并写出生成过程

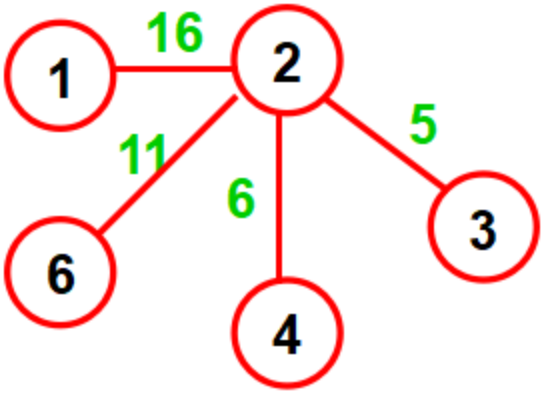


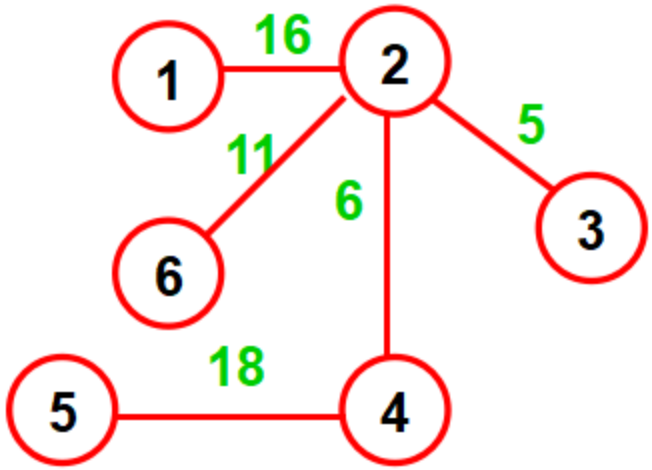
prim算法



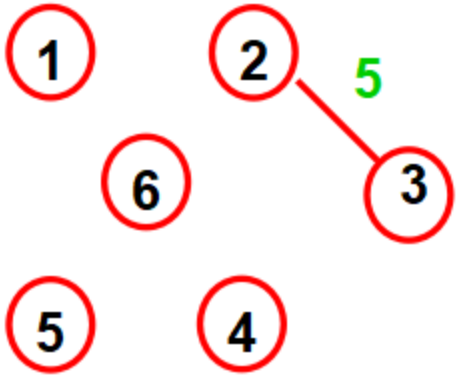


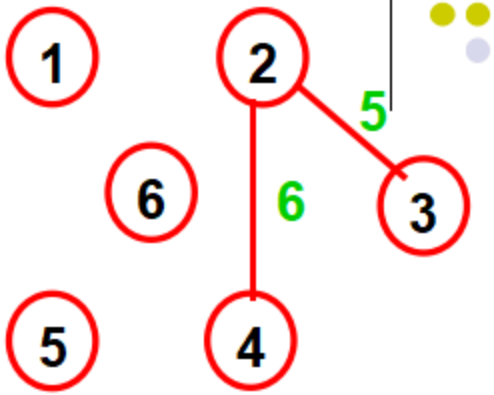


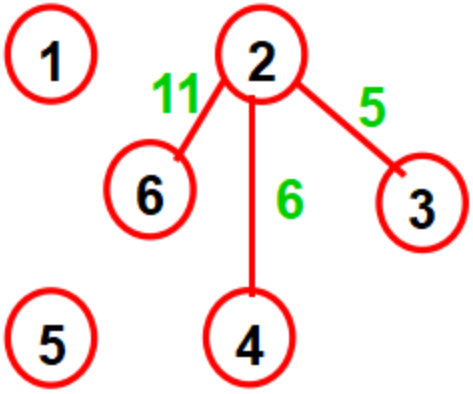


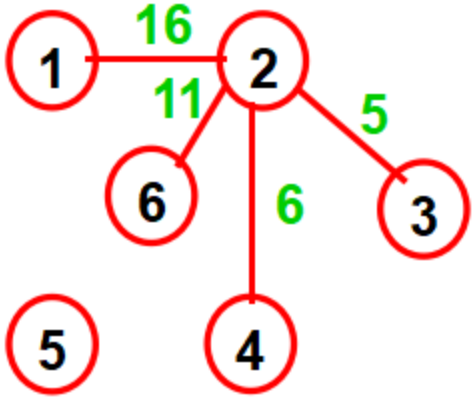


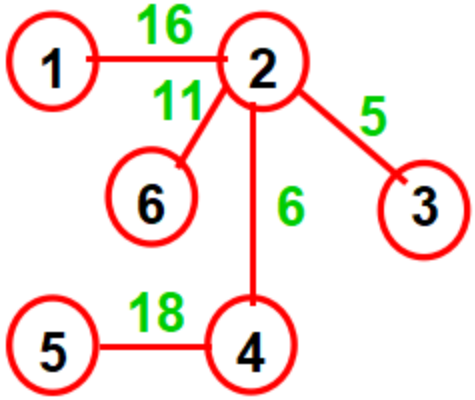
Kruskal算法


1
2
5
6
3
5
4


1
2
5
6
6
3
5
4


1
2
11
5
6
6
3
5
4

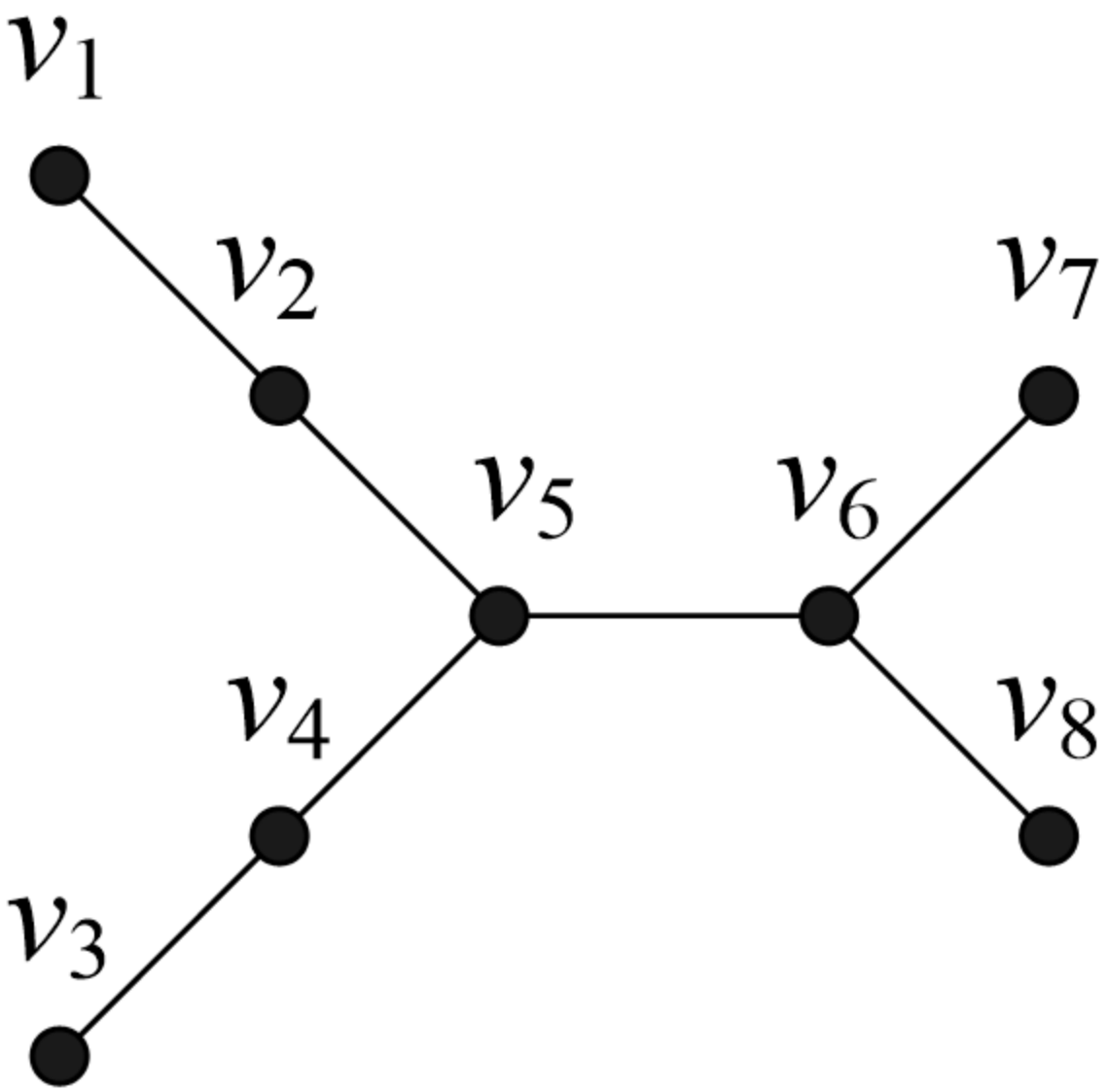

16
1
2
11
5
6
6
3
5
4


16
1
2
11
5
6
6
3
18
5
4

第18题

- 自由树（即无环连通图）T=(V, E)的直径是树中所有顶点之间最短路径的最大值，试设计一个算法求T的直径，并分析算法的时间复杂度。
- 【分级提示】
  - （1）可用邻接表作为存储结构；
  - （2）引入一个辅助数组保存各顶点的度；
  - （3）执行删除过程；
  - （4）并修正各个顶点的度。

分析1

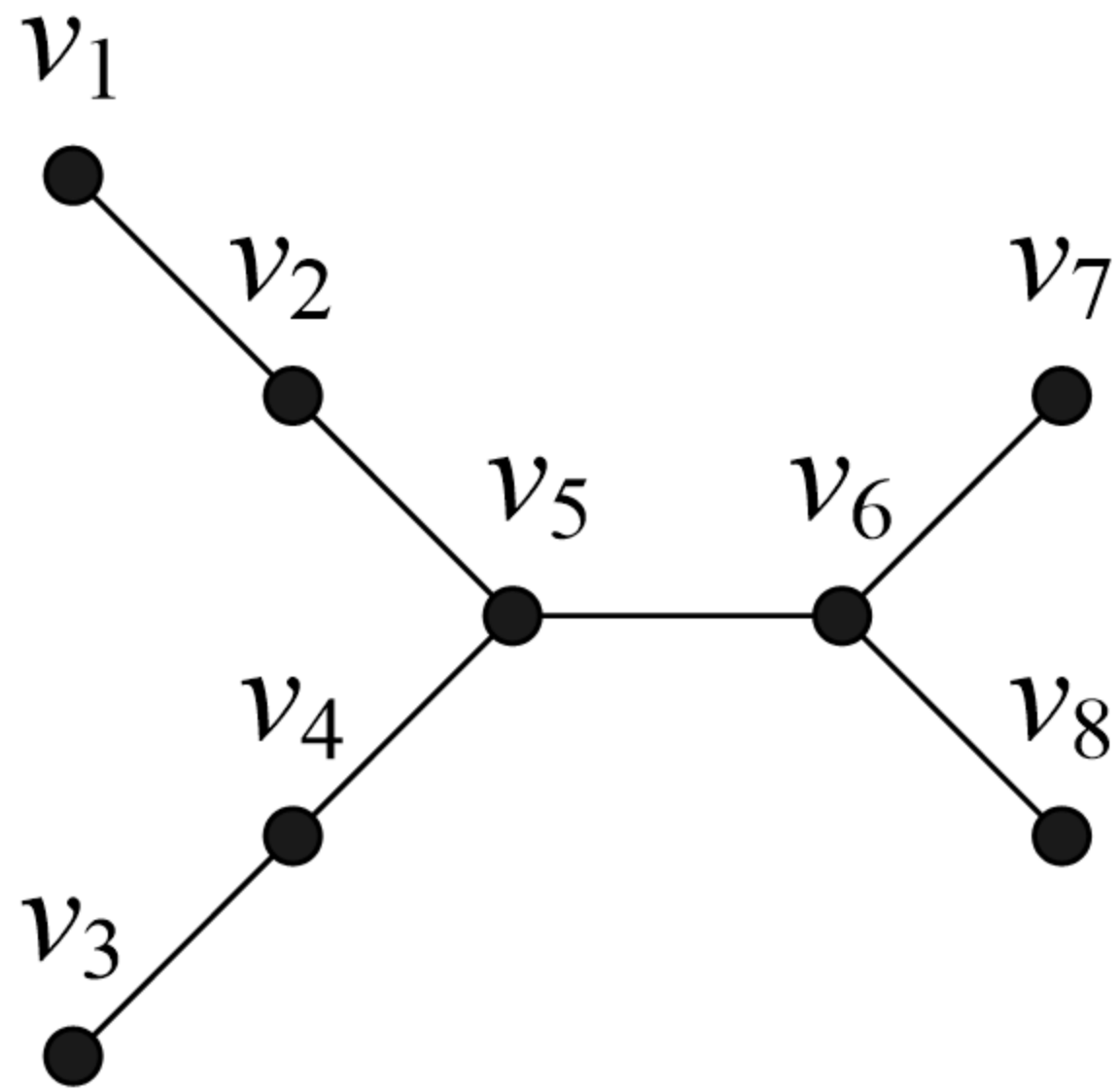


参考答案1

- 借鉴书后答案的思想
- 删除所有叶子（最外层的节点），直径增加2
- 最后剩不多于2个节点的时候直接知道直径
- 最初始的时候计算所有节点的度，并保存所有叶子节点（其度为1）
- 与拓扑排序类似，不是真的删除节点，只是减少节点的度，度减至1为新叶子

```
int Graph_List::TDiameter(int degrees[], int n, Queue q)
{
    for ( int r = 0; n >= 2; r++ ) {  // r表示删除轮数
        n -= q.size();
        for ( int i = q.size(); i > 0; i-- ) {
            Edge * p = vertices[q.QDelete()].adjacent;
            for ( ; p != NULL; p = p->link ) {
                int v = p->verAdj;
                if ( --degrees[v] == 1 ) q.QInsert( v ); }
        }
    }
    return 2*r + ( n == 2 );
}
```

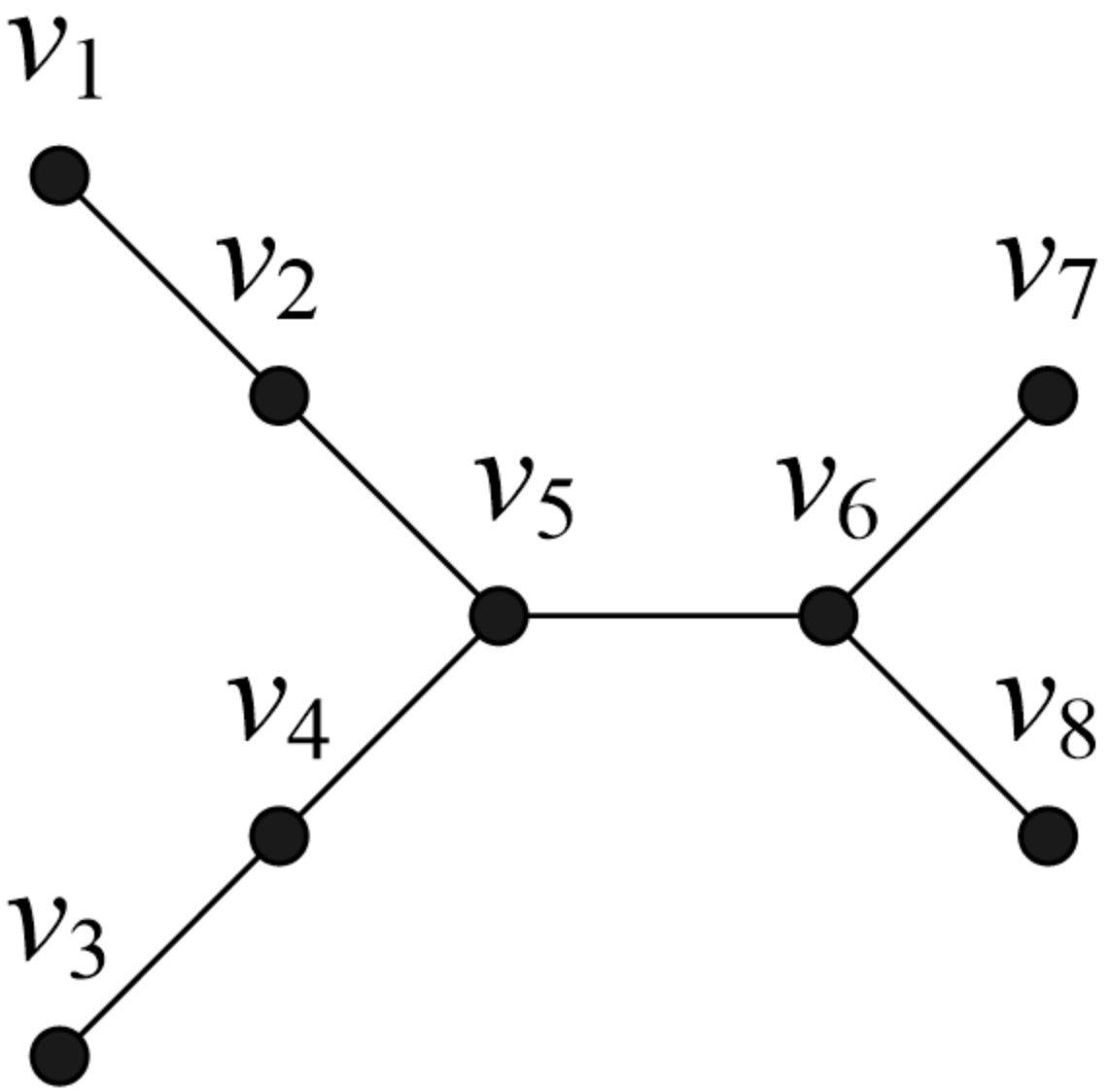
分析2



参考答案2

- 思路：直径是最长路径，从某点出发找距离最远的节点，该节点必然位于直径的端点，从该端点出发找最长路径即为直径。
- 具体算法略

分析3



参考答案

```
struct Vertex
{
    bool visited;
    int depth; //初始值为-1
};
```

```
function DepthExp( v, r)
    p ← adjacent(v). max ← -1
    WHILE vertex(p) ≠ r DO (
        IF depth(v) = -1 THEN DepthExp( vertex(p), v)
        IF depth(v) > max THEN d ← depth(v)
        p ← next(p)
    )
    p ← next(p)
    WHILE p ≠ Λ DO (
        IF depth(v) = -1 THEN DepthExp( vertex(p), v)
        IF depth(v) > max THEN d ← depth(v)
        p ← next(p)
    )
    depth(v) ← max + 1
end DepthExp
```

```
function FirstSecond( v. first, second)
    p ← adjacent(v). first ← vertex(p)
    IF next(p) = Λ THEN second ← Λ
    p ← next(p). second ← vertex(p). p ← next(p).
    IF depth(first) < depth(second) THEN
        first ↔ second
    WHILE p ≠ Λ DO (
        IF depth(vertex(p)) ≥ depth(first) THEN
            second ← first. first ← vertex(p)
        ELSE IF depth(p) > depth(second) THEN
            second ← vertex(p)
        p ← next(p)
    )
END
```

```
Algorithm Diameter(G. d)
    IF |V(G)| < 2 THEN d ← 0. RETURN
    v ← a vertex with more than two neighbors
    p ← adjacent(v)
    WHILE p ≠ Λ DO (
        DepthExp(vertex(p), v)
        p ← next(p)
    )
    FirstSecond(v. first, second)
    WHILE depth(first) > depth(second) + 1 DO (
        DepthExp(v, first). v ← first
        FirstSecond(v. first, second)
    )
    d ← depth(first) + depth(second) + 2
```

补充1

已知一个有向图的边集为{<a,b>, <a,c>, <a,d>, <b,d>, <b,e>, <d,e>}，则由该图产生的一种可能的拓扑序列为(　　)。

A. a,b,c,d,e　　B. a,b,d,e,b

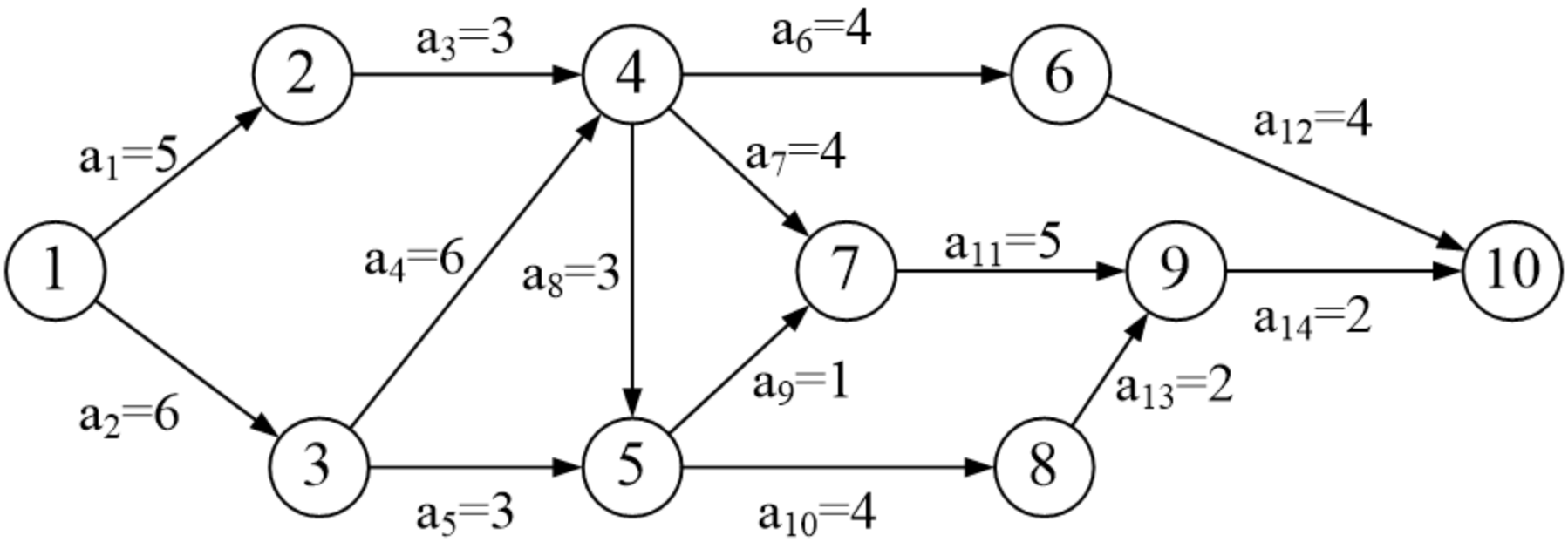
C. a,c,b,e,d　　D. a,c,d,b,e

参考答案

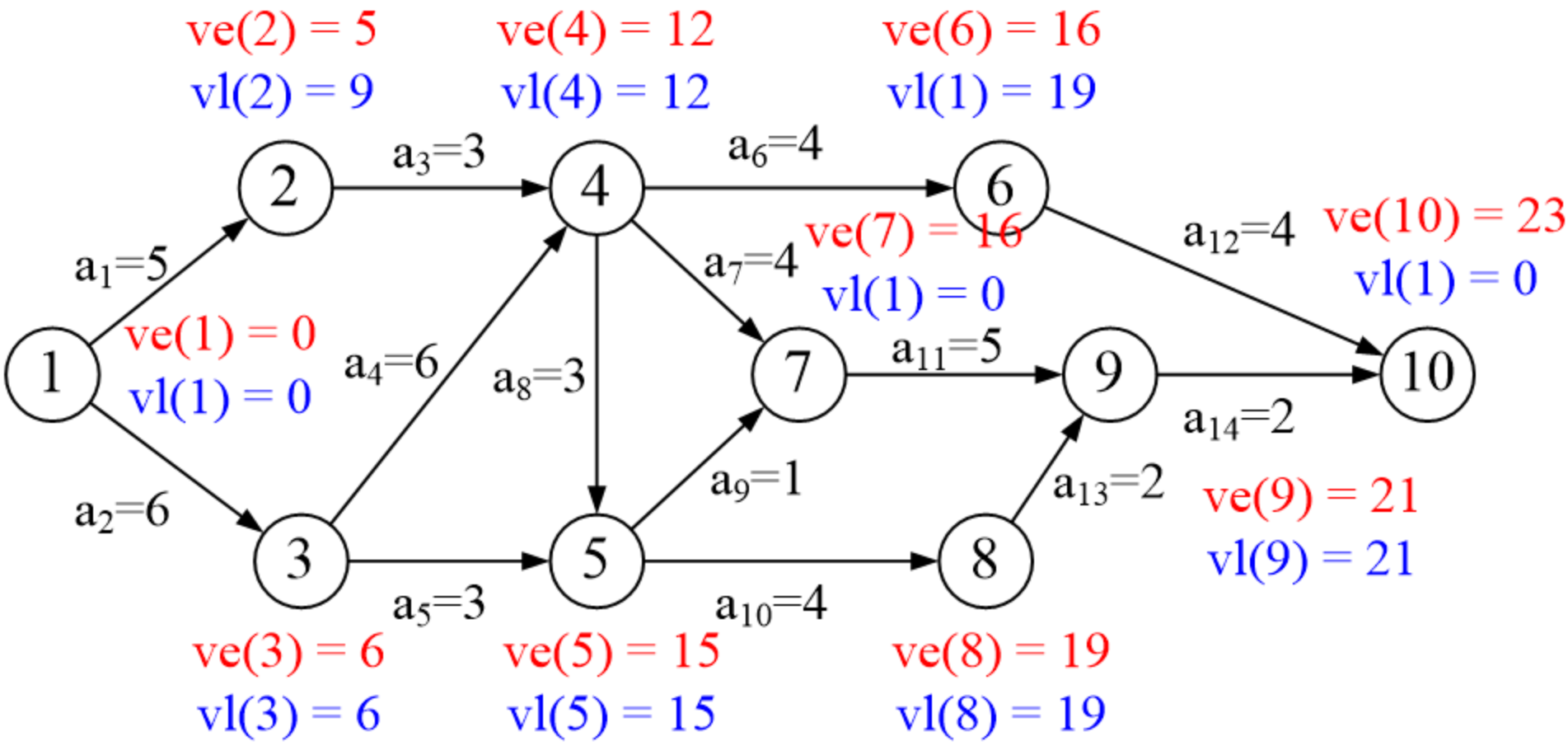
- A

补充2

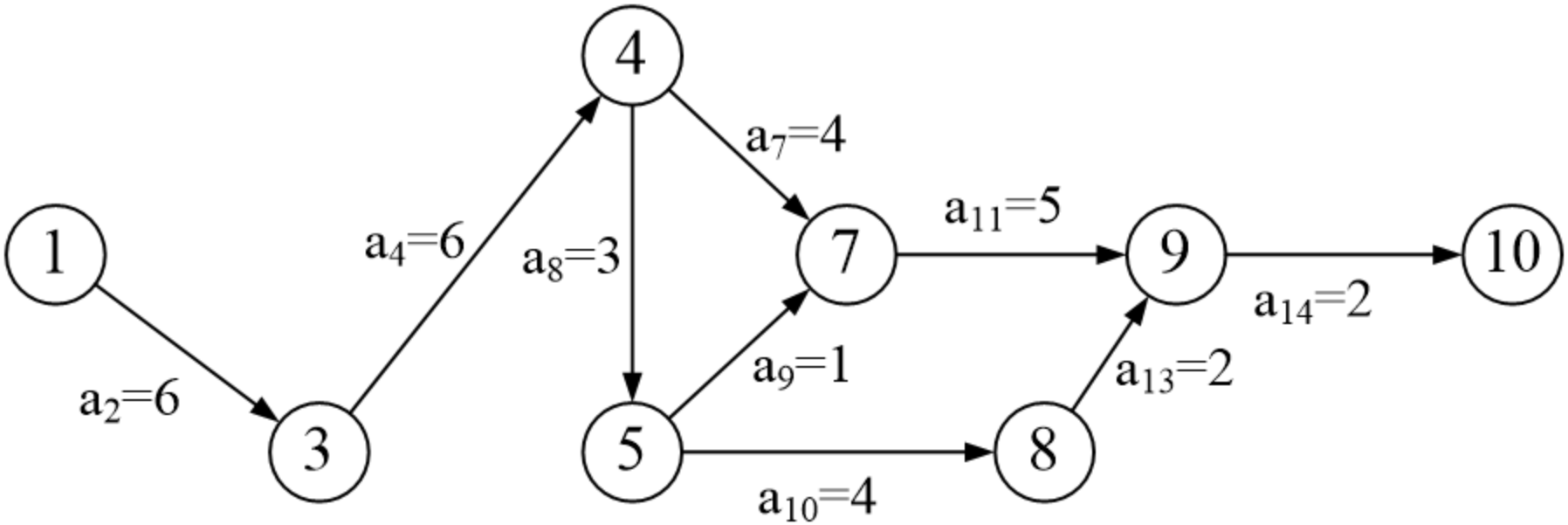
- 求下面AOE网的关键路径。



参考答案


ve(2) = 5
vl(2) = 9
ve(4) = 12
vl(4) = 12
ve(6) = 16
vl(1) = 19
a3=3
a6=4
2
4
6
a1=5
ve(7) = 16
vl(1) = 0
a12=4
ve(10) = 23
vl(1) = 0
a7=4
ve(1) = 0
vl(1) = 0
a4=6
a8=3
7
a11=5
9
10
1
a14=2
a9=1
a13=2
ve(9) = 21
vl(9) = 21
a2=6
3
5
8
a5=3
a10=4
ve(3) = 6
vl(3) = 6
ve(5) = 15
vl(5) = 15
ve(8) = 19
vl(8) = 19

参考答案



补充3：Google笔试题

- 含n个顶点，m条边的连通图，至少去掉几条边才能构成一棵树?

参考答案

- m-n+1

补充4：Google笔试题

- 在一个无向图中，寻找是否有一条距离为k的路径，描述算法即可，不用实现，分析算法的时间和空间复杂度，尽量优化算法

分析1

- 有环必然存在长度为k的路径
- 无环则是自由森林，直径就是最长路径

参考答案1

- 以每个入度为0的点为起点，在深度优先遍历的基础上，检测是否存在环路，若存在，直接返回true，图中包含若干个自由树，计算每个自由树的最大直径
- 无向图中边数大于等于节点数n，则必然存在环
- 时间复杂度O(n)

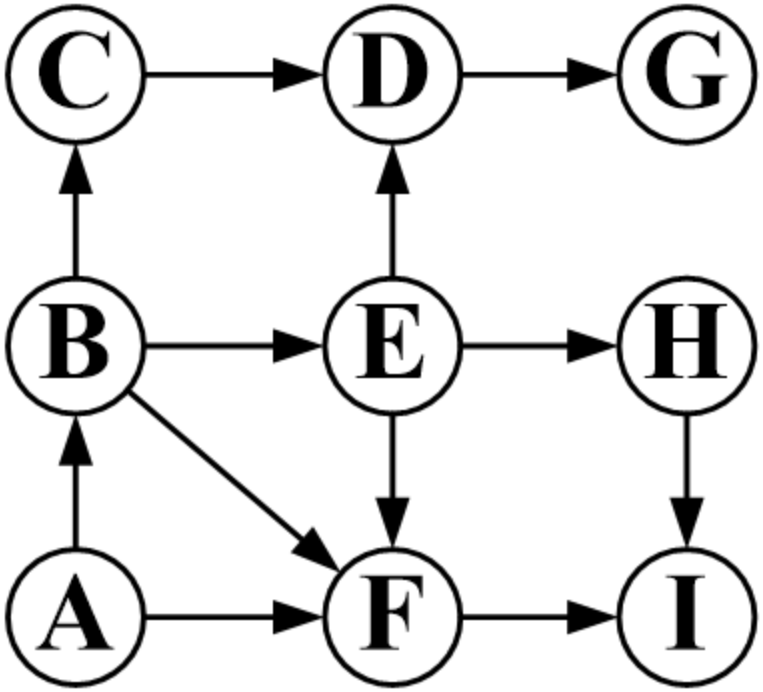
补充5：金山软件笔试题

- 给定一个有向图：
  1、要确定该图中是否存在环路，给出算法思路和主要部分代码实现；
  2、如果该图无环，要获取给定起始和终止节点间的路径数目，给出算法思路和主要部分代码实现；
  3、如果该图有环，给出上述算法调整方案和主要部分代码实现。

分析

- (1) 前面已经讲过
- (2) 深度优先遍历，使用动态规划记录节点到目的节点的路径数，u到v的路径数等于u的后继到v的路径之和

分析



分析

- (3) 前两步都是基于深度优先的，进行结合即可

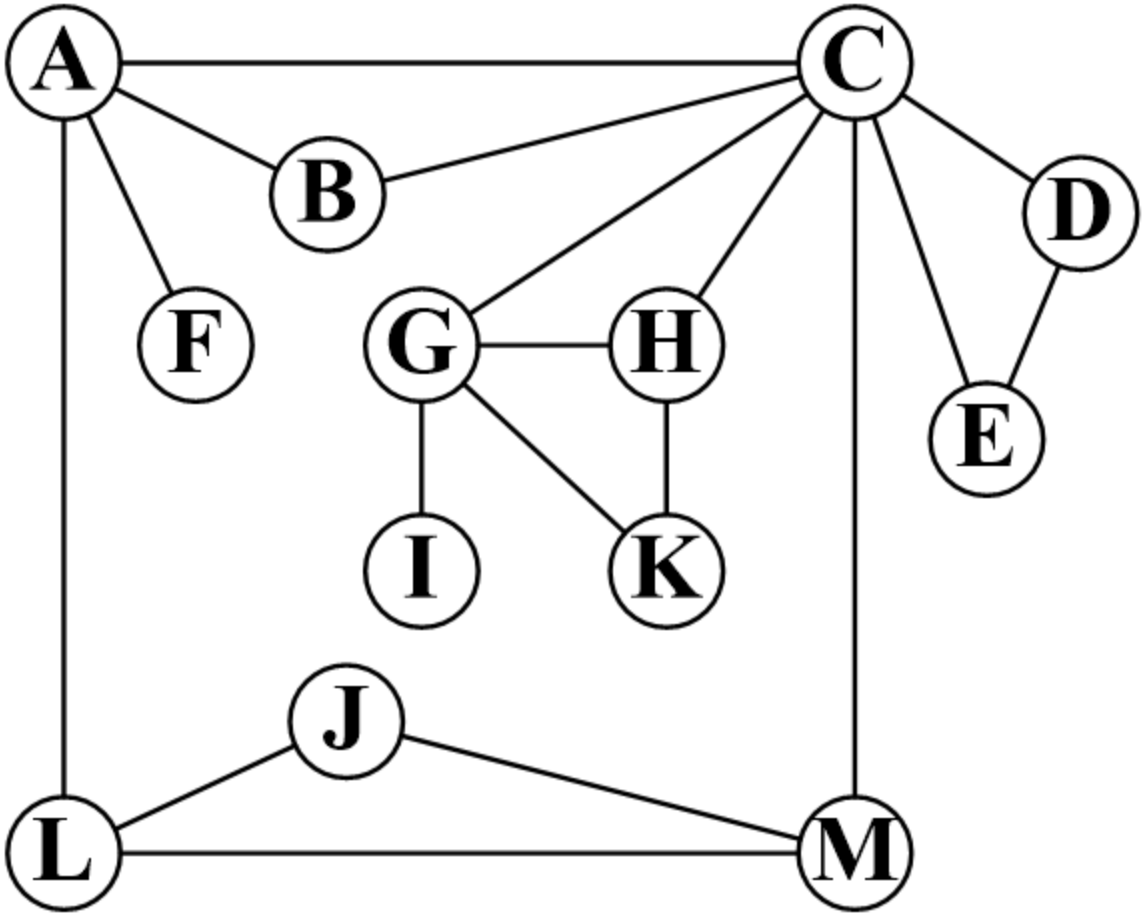
参考答案

- 略

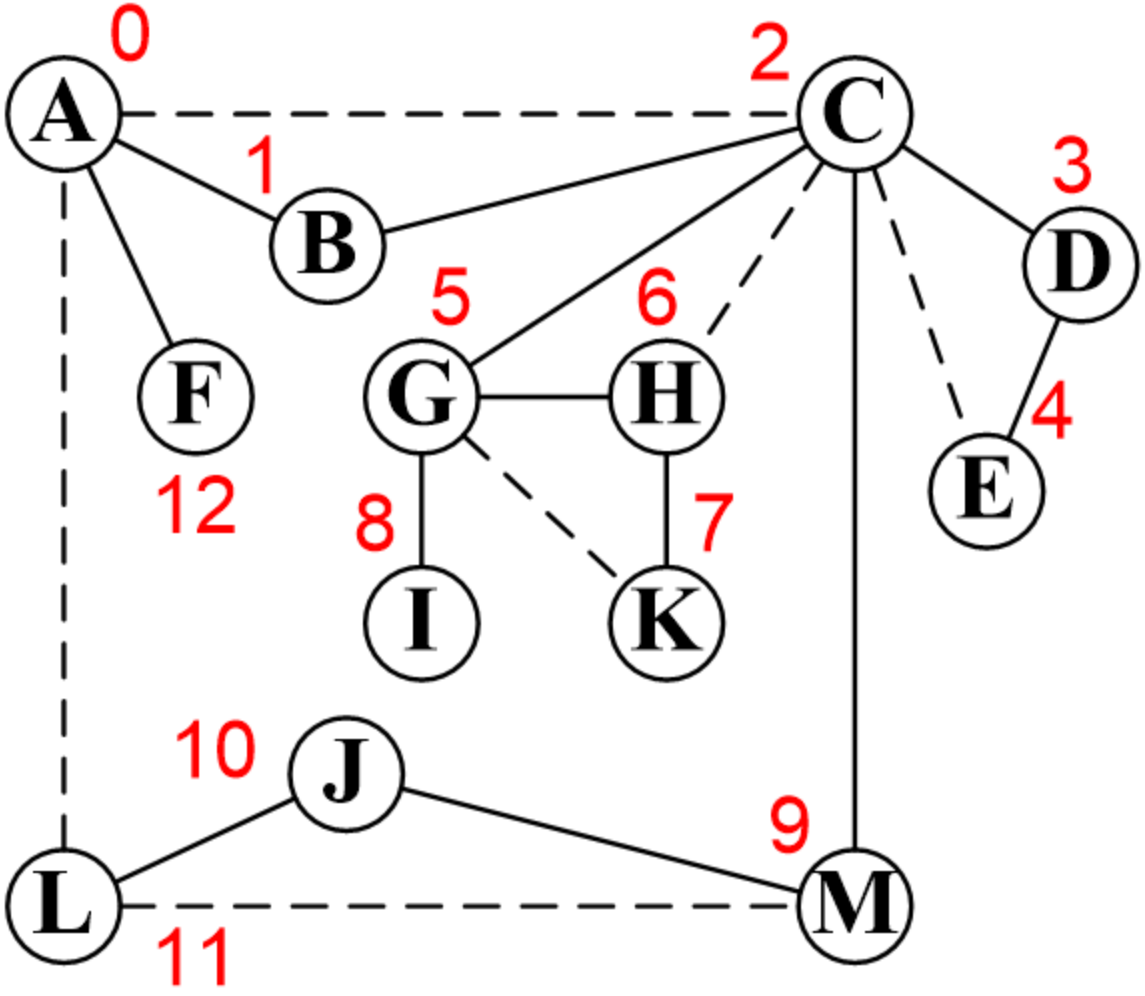
补充6：网易有道笔试题

- 求一个有向连通图的割点，割点的定义是，如果除去此节点和与其相关的边， 有向图不再连通，描述算法。

分析



- 利用深度优先遍历，在深度优先搜索树中，根节点是割点当且仅当有多棵子树，非根节点是割点当且仅当它有子节点，且它的子节点不能访问到它的祖先节点


0
A
2
C
1
B
3
D
5
6
F
G
H
4
E
12
8
7
I
K
10
J
9
L
M
11

参考答案

```
void dfs(int u)
{
    static int counter = 0; //记录节点u的遍历次序
    static int root = u; //记录源点
    int num = 0; //记录子节点数目
    Edge *p = vertices[u].adjacent;
    if ( p == NULL ) return;
    visit[u] = 1;
    order[u] = low[u] = ++counter; //初始化
    for(; p != NULL; p = p->link) {
```

```
        if (!visit[v]) {  // (u,v)为树边
            num++;
            dfs(v);
            low[u] = min(low[u], low[v]);
            if ( u == root && num > 1)
                cout << “articulation point:” << u << endl;
            if ( u != NIL && low[v] >= order[u])
                cout << “articulation point:” << u << endl;
         }
        else if(v != parent[u])
            low[u] = min(low[u], order[v]); // (u,v)为回边
    } //end-for
}
```

THE END