数据结构——C语言描述（慕课版）

05 关键路径

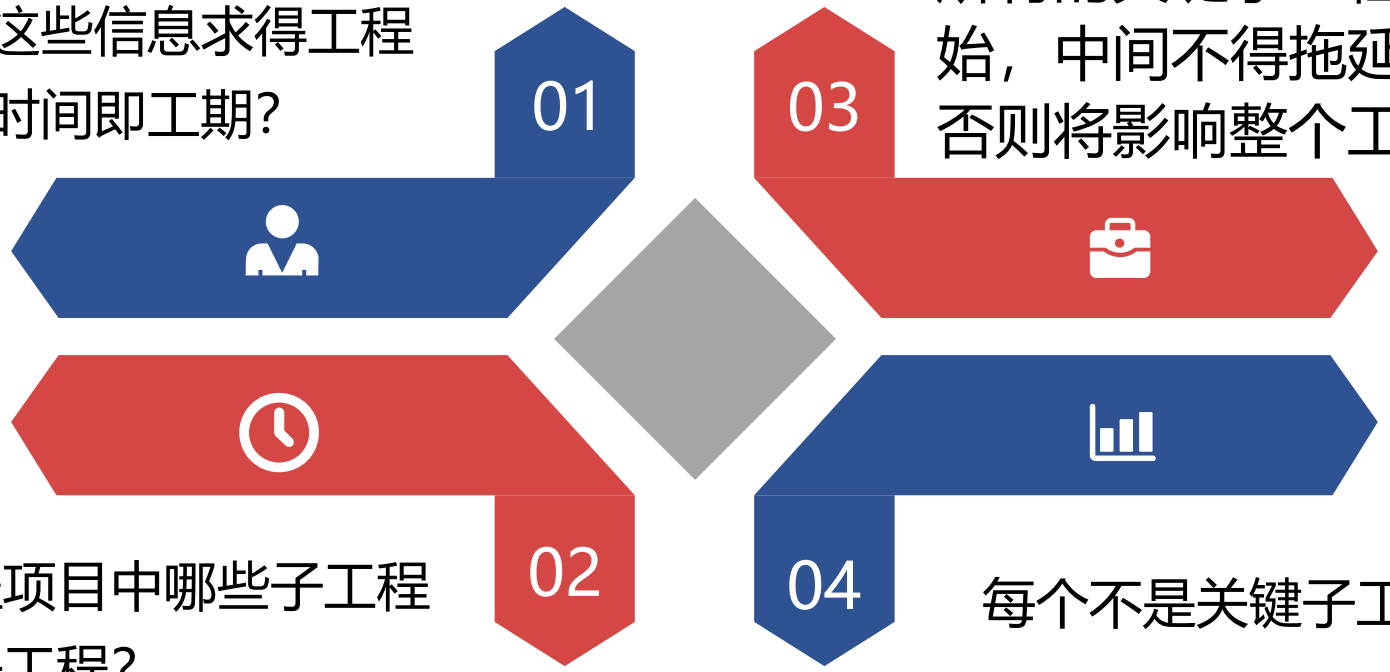
编著：张同珍 & 学校： 上海交通大学

关键路径：

一个工程通常由若干个子工程构成，大多子工程在开始实施时既要有一定的先觉条件，自身也需要一定的时间来完成。

如何根据这些信息求得工程需要的总时间即工期?

01



02

在整个工程项目中哪些子工程是关键的子工程?

03

所有的关键子工程必须在可以开始时马上开始，中间不得拖延，必须按照计划如期完成，否则将影响整个工程工期；

04

每个不是关键子工程的工程有多少时间余量?

这些问题都是工程施工前要精确计算的。

关键路径

在AOE网中，顶点表示事件或者状态，边表示活动（这里就是子工程活动），边上的权值表示完成活动所需要的时间。

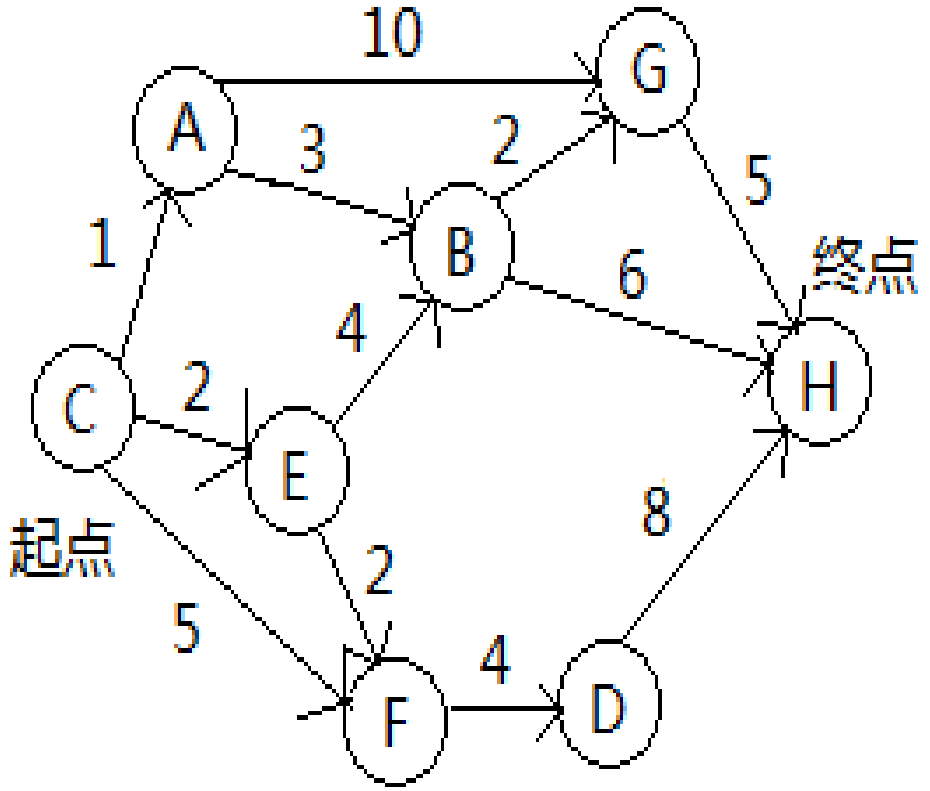
一个顶点如果有n条边射入，则表示当这n个活动全部完成才说该顶点表达的事件发生或者说达到了该顶点表示的状态，之后由该顶点发出的边表示的活动才可以启动。

通常AOE网会至少有一个起点或称源点，起点没有边射入、入度为0，说明该状态不需要条件已经到达或者说事件不需要条件就发生了；

AOE网也会有一个终点或称汇点，终点没有边射出、出度为0，说明当到达该顶点表示的状态时就意味着整个工程的结束。

关键路径

右图是一个用AOE网表示的工程图，其中C是起点、H是终点，其他顶点如顶点B表示了一个事件，此事件须当活动<A,B>，<E,B>都完成了才能发生。



利用AOE网求工程中的关键活动可以通过以下几步来完成：

01 求得每个顶点事件的最早发生时间，即到达顶点表示的状态所需要的最短时间。

02 求得每个顶点事件的最迟发生时间，即到达顶点表示的状态所能容忍的最长花费时间。

03 求得每个活动的最早发生时间，即每个边表示的活动最早何时能条件具备并开始。

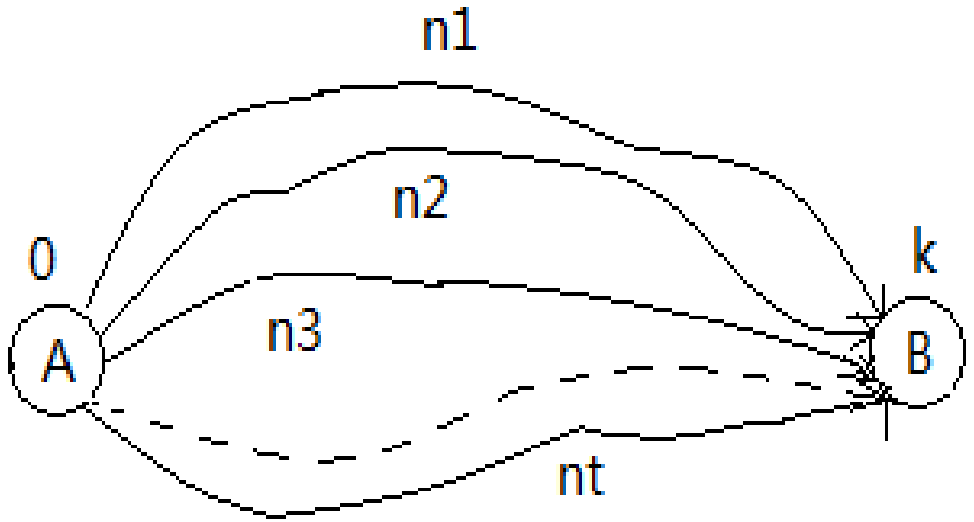
04 求得每个活动的最迟发生时间，即每个边表示的活动最晚何时必须开始，否则影响整个工程工期。

05 当某个活动的最早发生时间和最迟发生时间相同时，表示这些活动一旦条件具备必须马上开始，刻不容缓，否则将延长整个工程的工期，这些活动便是关键活动。

求顶点事件的最早发生时间：

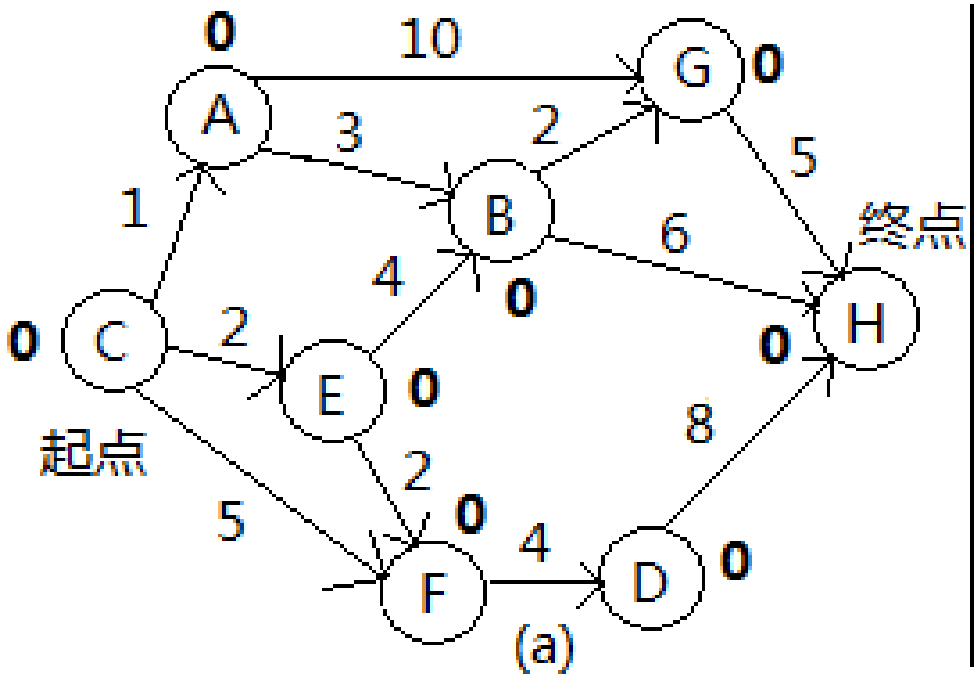
如果从起点到一个顶点有若干条路径，即说明该顶点表示的事件须当每条路径上所有的活动都完成才能发生，因此事件的最早发生时间是最长路径所消耗的时间。

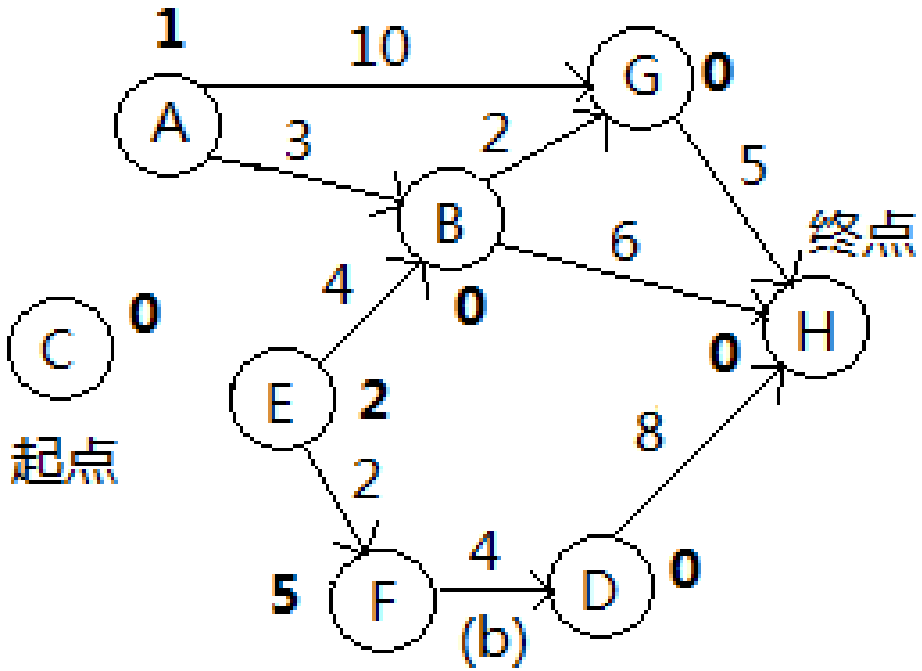
右图中，A顶点是起点，则B顶点事件的最早到达时间是A到B间最长路径上的活动所消耗的时间k=max(n1,n2,n3,…nt)，k时间后由顶点B发出的所有活动才可以开始。

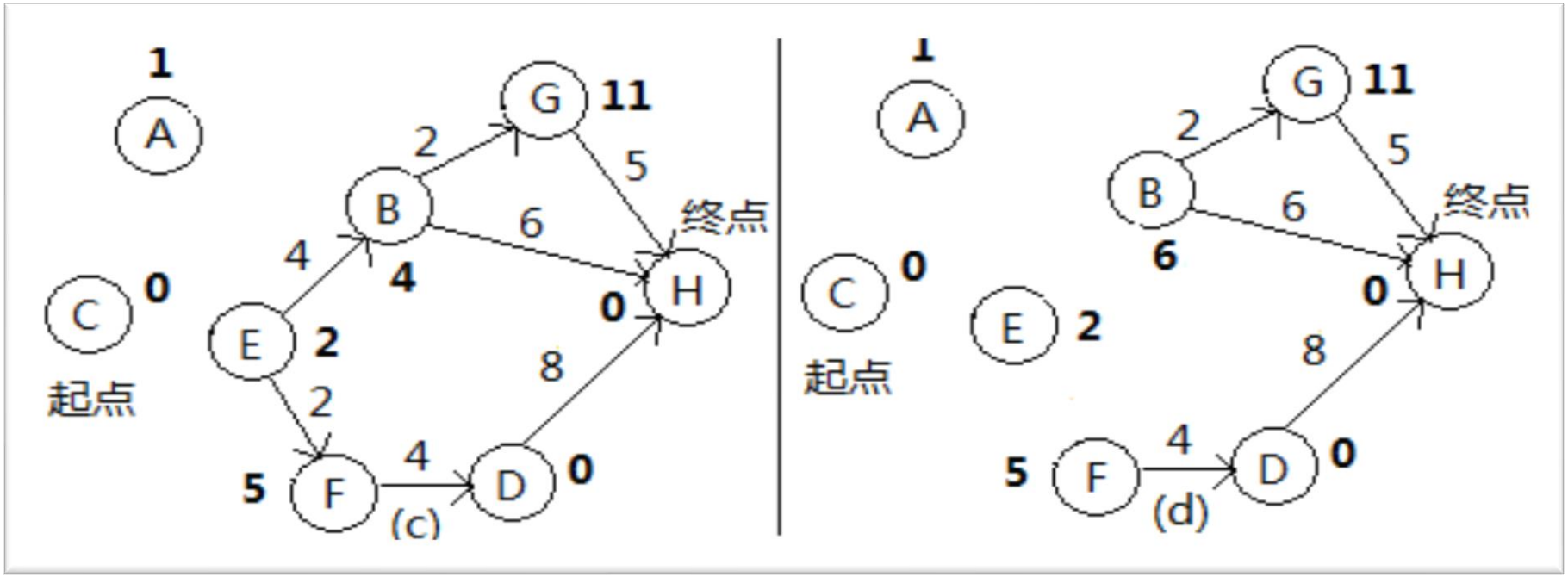


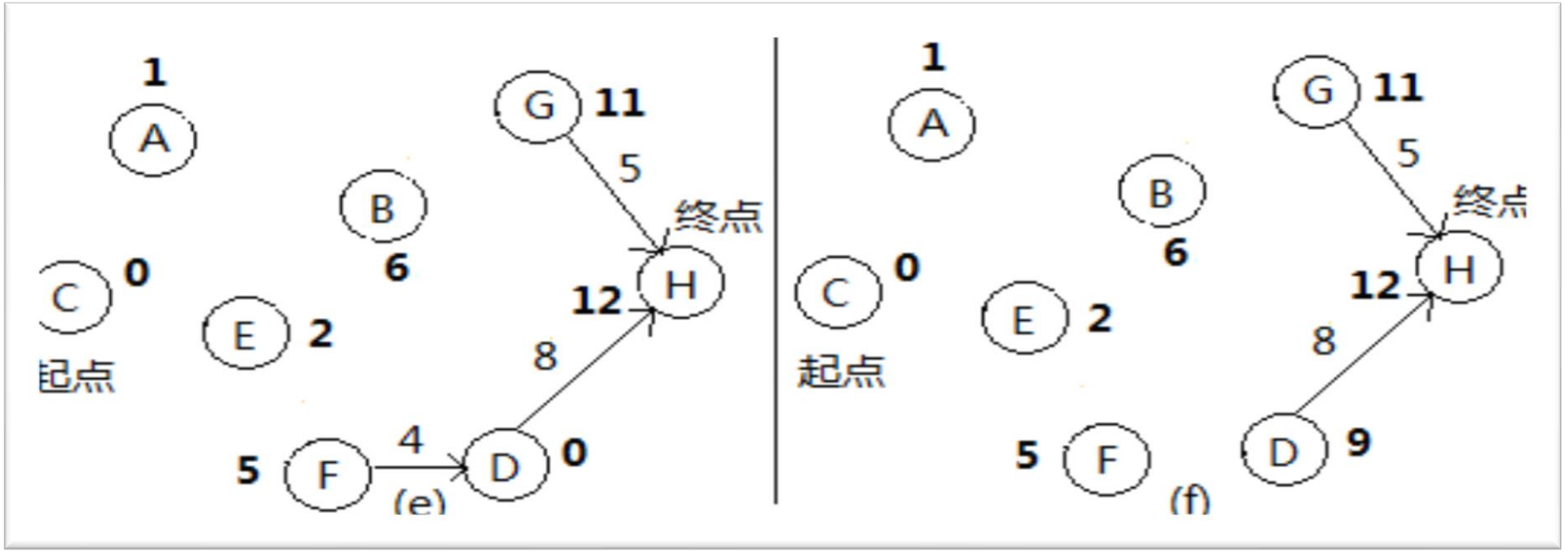
求顶点事件的最早发生时间：

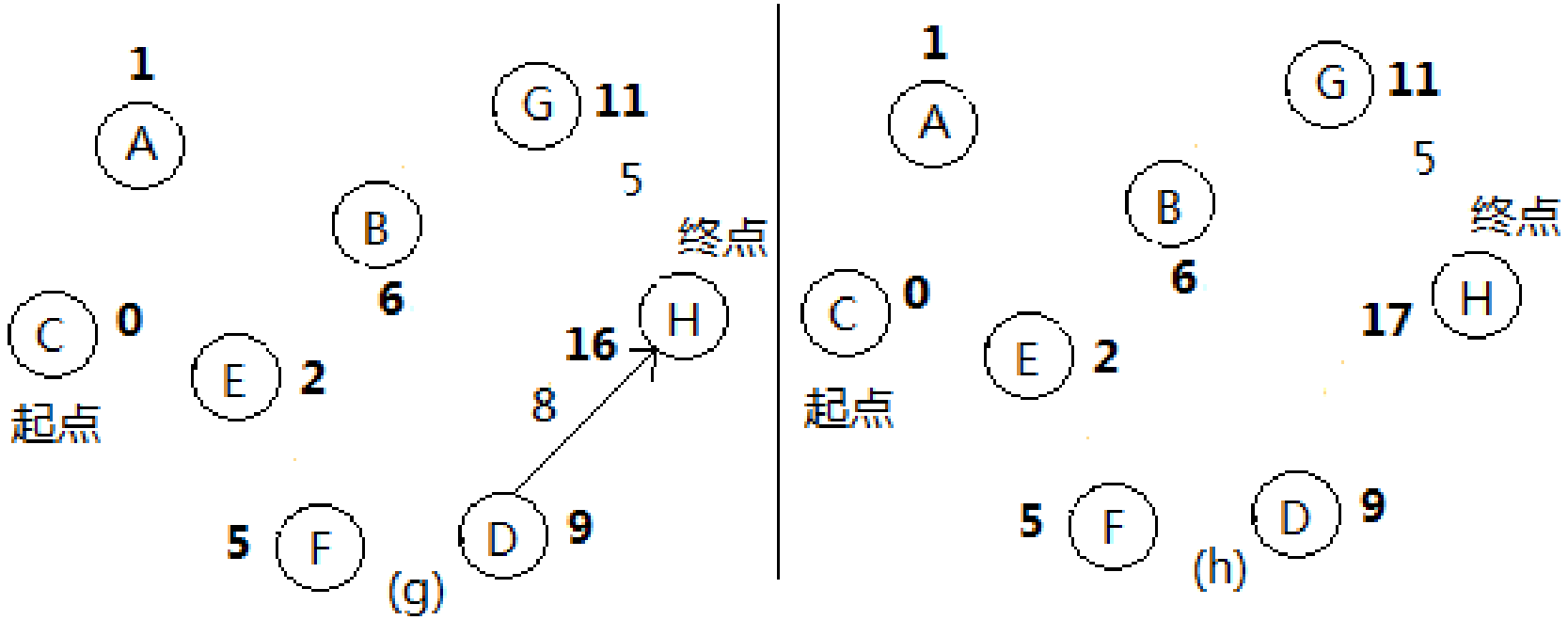
AOE网顶点事件的最早发生时间的具体计算过程，图中顶点旁的加黑数字为该顶点当前的最早发生时间。






1
A
G
11
2
5
B
6
终点
4
4
H
0
C
0
E
2
8
起点
2
5
F
4
D
0
(c)
1
A
G
11
2
5
B
6
终点
6
C
0
H
0
E
2
起点
8
5
F
4
D
0
(d)

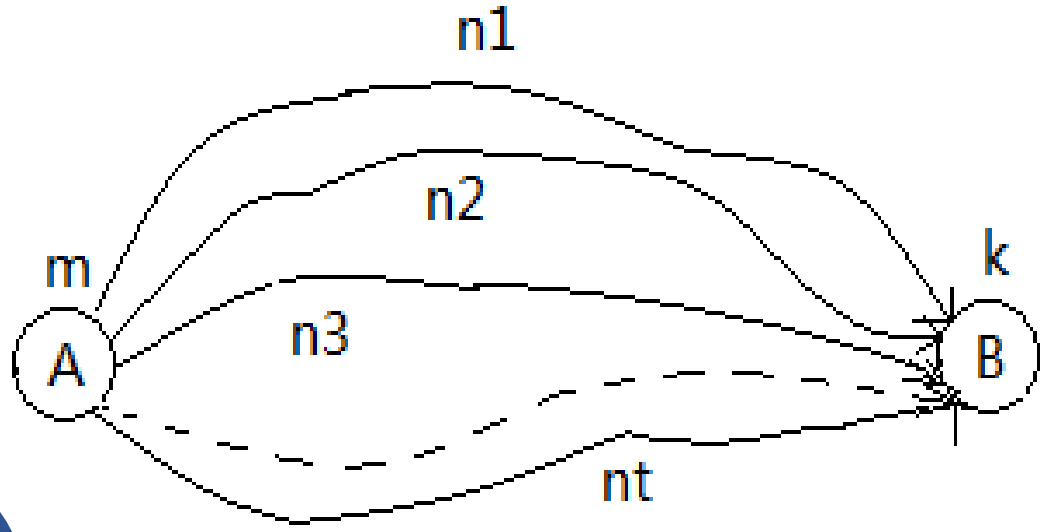

1
A
G
11
5
终点
B
6
H
12
C
0
E
2
起点
8
5
F
4
D
0
(e)
1
A
G
11
5
B
终点
6
C
0
H
12
E
2
起点
8
5
F
D
9
(f)


1
A
G
11
5
B
终点
6
C
0
16
H
E
2
8
起点
5
F
D
9
(g)
1
A
G
11
5
B
终点
6
C
0
17
H
E
2
起点
5
F
D
9
(h)

求顶点事件的最迟发生时间：

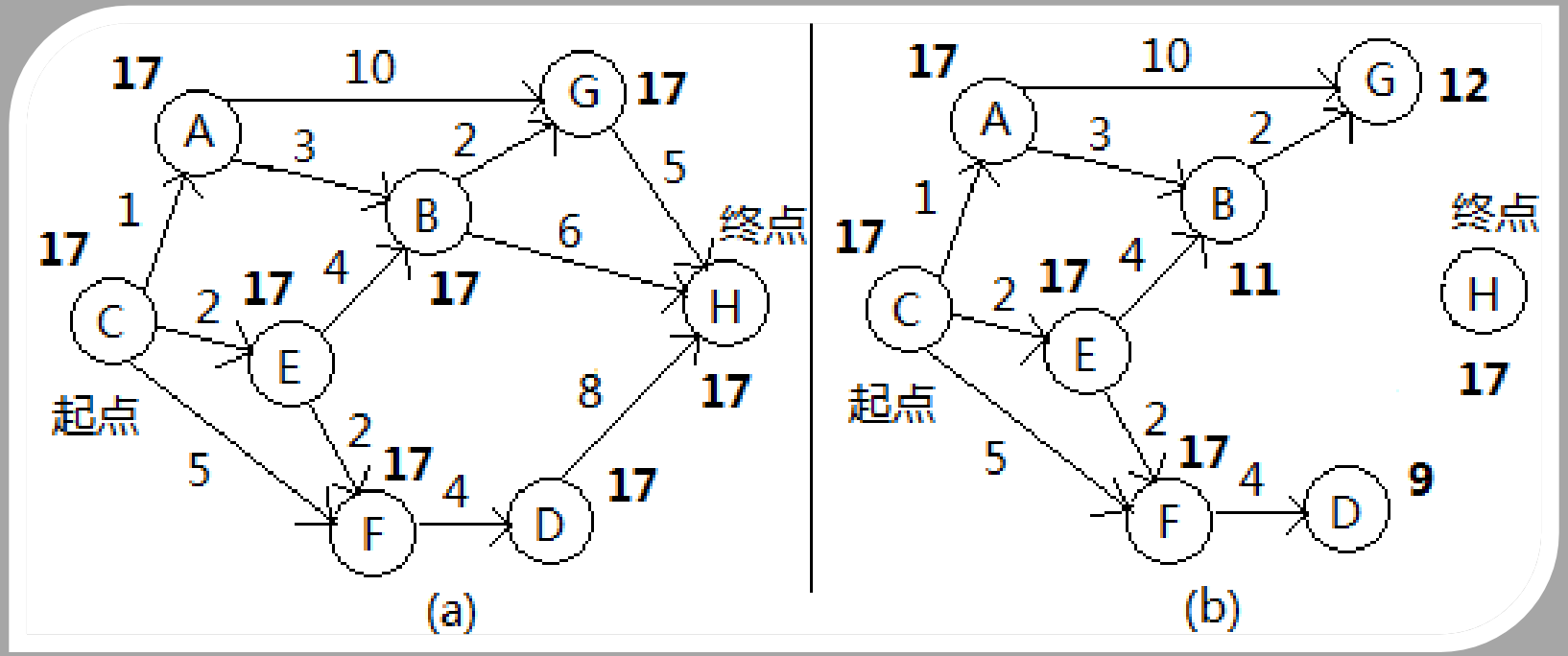
如果一个工程终点的最早时间已知，这个最早时间就是工程需要的总的最短工期，为了达到这个目标，可以设定这个时间就是终点事件的最迟发生时间，然后对余下的顶点倒推回去，可以获得其余顶点事件的最迟发生时间。

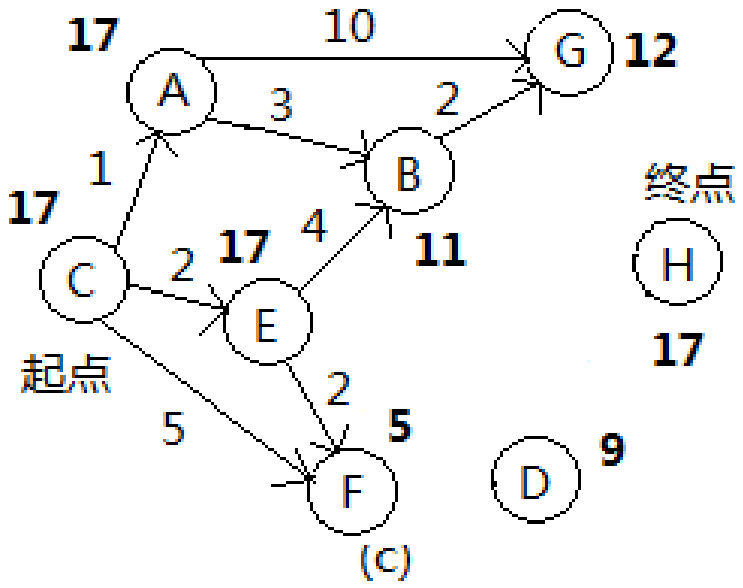
右图中，如果终点事件B的最迟发生时间为k，则顶点事件A的最迟发生时间要满足m=k-max(n1,n2,n3,…nt)，即要保证有足够的时间完成A、B间最长路径上的所有活动。

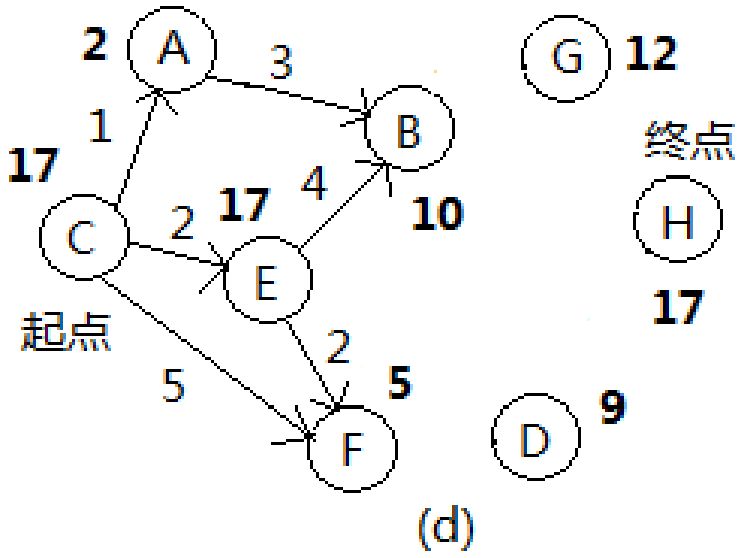


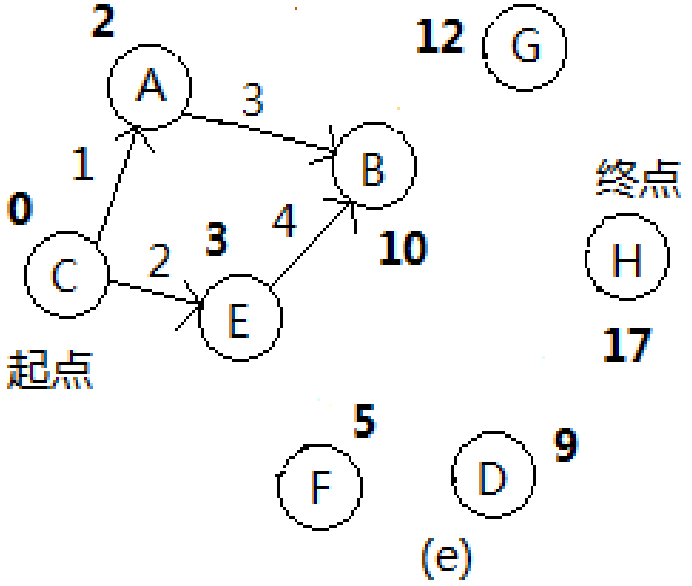
求顶点事件的最迟发生时间：

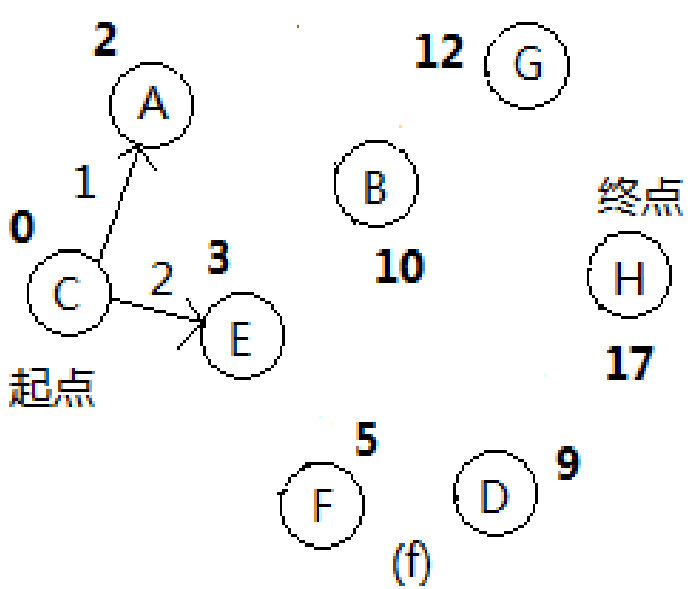
终点的最迟发生时间即其最早发生时间，所有其他顶点的最迟发生时间因取最小值，故首先赋予一个和终点H一样的最迟发生时间，然后按照求最早发生时的逆序H、D、G、F、B、E、A、C顺序进行计算：

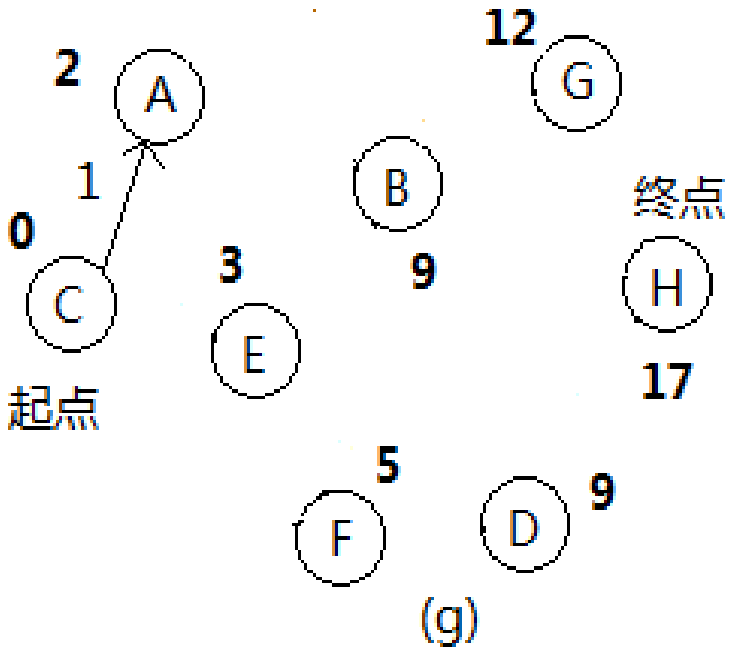



17
10
G
12
A
3
2
1
17
B
11
C
2
17
4
E
起点
5
2
5
F
D
9
终点
H
17
(c)

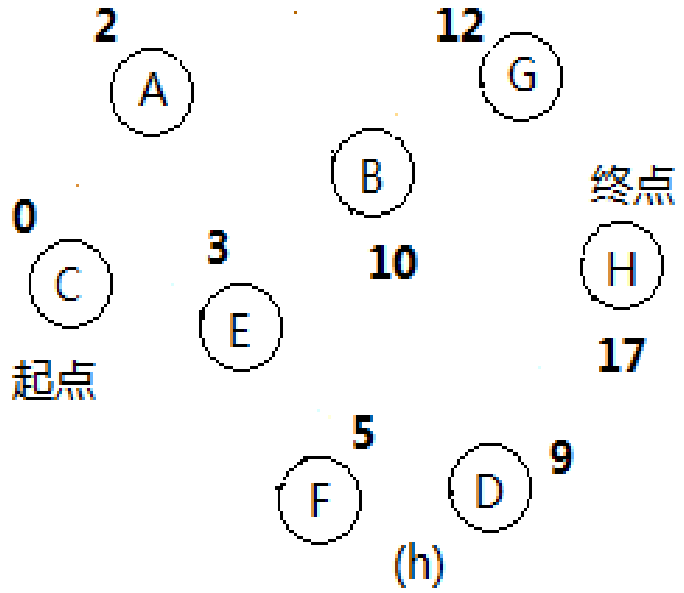

2
A
3
G
12
1
B
17
C
2
17
4
10
E
终点
H
17
起点
2
5
5
F
D
9
(d)


2
12
G
A
3
1
B
0
C
2
3
4
10
E
终点
H
17
起点
5
F
D
9
(e)


2
12
G
A
1
B
0
C
2
3
10
E
终点
H
17
起点
5
F
D
9
(f)



(g)



(h)

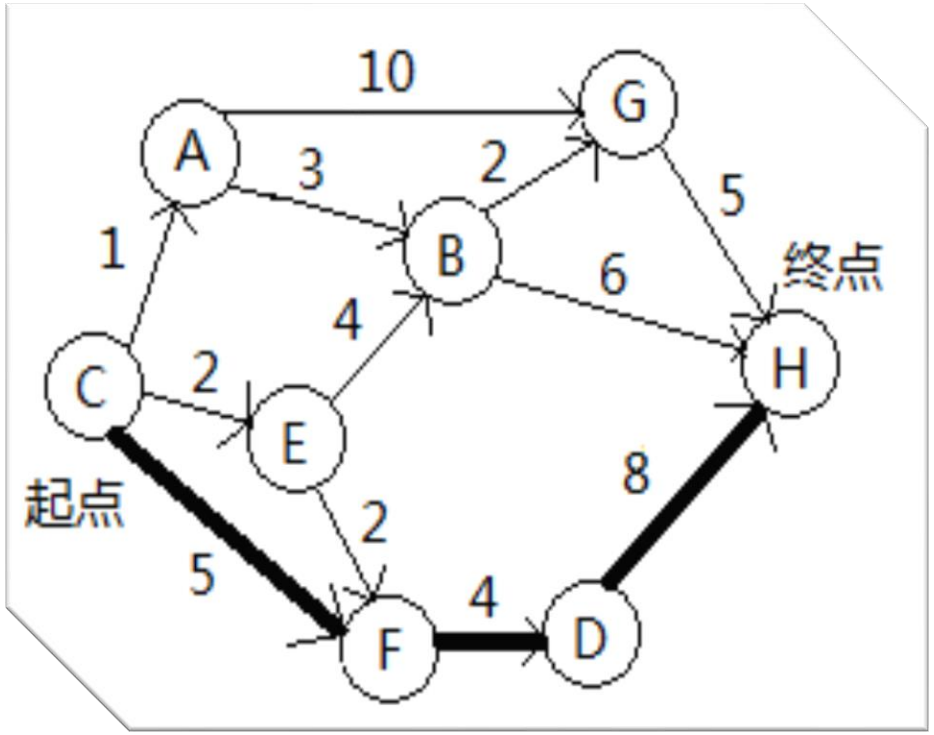
顶点	最早发生时间	最迟发生时间
C	0	0
A	1	2
E	2	3
B	6	10
F	5	5
G	11	12
D	9	9
H	17	17

求活动的最早发生时间和最迟发生时间：

AOE网中的一个活动<u,v>，活动的最早发生时间是顶点u事件的最早发生时间，活动的最迟发生时间是顶点v的最迟发生时间减去边<u,v>的权值。

顶点	最早发生时间	最迟发生时间
<C，A>	0	2-1=1
<C，E>	0	3-2=1
<C，F>	0	5-5=0
<A，G>	1	12-10=2
<A，B>	1	10-3=7
<E，B>	2	10-4=6
<E，F>	2	5-2=3
<B，G>	6	12-2=10
<B，H>	6	17-6=11
<F，D>	5	9-4=5
<G，H>	11	17-5=12
<D，H>	9	17-8=9

求关键路径：



当活动的最早发生时间和最迟发生时间一致时，表示该活动为关键活动（如表5-3中旁边加黑块的边），这些关键活动组成的由起点到终点的路径成为关键路径。

求关键路径(用邻接矩阵表示有向图)算法实现：

01
OPTION

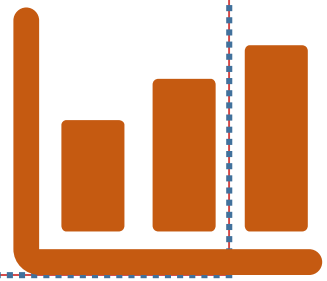
程序定义边结点结构，包括了和边相邻的两个顶点u和v、权重weight、活动的最早发生时间early以及最迟发生时间last，定义了记录顶点入度的数组indegree、记录顶点的最早发生时间数组verEarly和最迟发生时间数组verLast。

02
OPTION

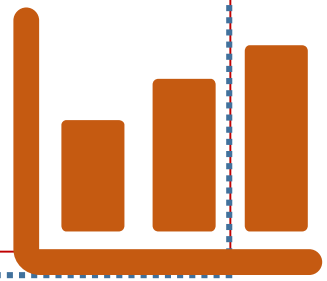
定义了两个栈s1和s2，其中s1保存了**入度为0的顶点**，s2保存了s1**出栈的顶点序列**，s1中顶点出栈的顺序就是顶点计算最早发生时间的**顺序**；s2中顶点出栈是计算最早发生时间时顶点序列的**逆序**，此逆序用于计算顶点的最迟发生时间。

求关键路径(用邻接矩阵表示有向图)算法实现：

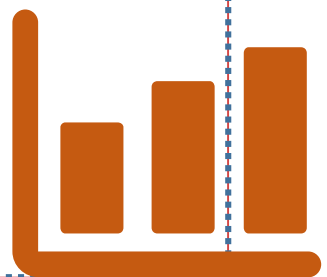
```
typedef struct
{
    int u, v, weight;
    int early, last;
}Edges;

void keyActivity (Graph *g, verType start, verType end)
{   int *inDegree;
    int *verEarly, *verLast; //事件-顶点的最早发生时间、最迟发生时间
    Edges *edgeEL; //活动-边的最早发生时间、最迟发生时间
    stack s1,s2;
    int i, j, k;
    int u, v;
    int intStart, intEnd;
    int total;
```



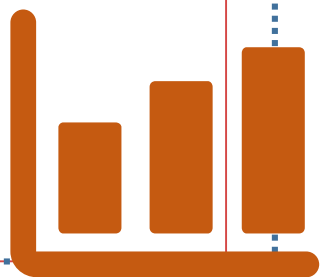
```
inDegree    = (int *) malloc (sizeof(int)*g->verts);
verEarly    = (int *) malloc (sizeof(int)*g->verts);
verLast     = (int *) malloc (sizeof(int)*g->verts);
edgeEL      = (Edges *) malloc (sizeof(Edges)*g->edges);
initialize(&s1);
initialize(&s2);

//找到起点和终点的下标
intStart = intEnd = -1;
for (i=0; i<g->verts; i++)
{   if (g->verList[i]==start)
        intStart = i;
    if (g->verList[i]==end)
        intEnd = i;
}
if ((intStart==-1)||(intEnd==-1)) exit(1);
```



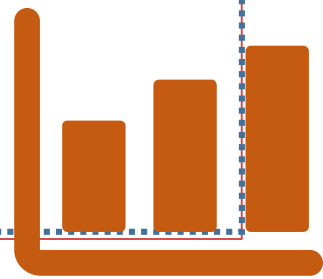
```
//计算每个顶点的入度，计数邻接矩阵每一列中非无穷大元素
for (j=0; j<g->verts; j++)
{   inDegree[j] = 0;
    for (i=0; i<g->verts; i++)
    {
        if (g->edgeMatrix[i][j]!=g->noEdge)
          inDegree[j]++;
    }
}

//初始化顶点最早发生时间
for (i=0; i<g->verts; i++) {
    verEarly[i] = 0;
}
```



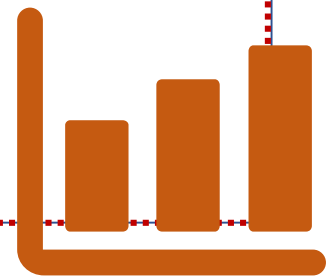
```
//计算每个顶点的最早发生时间
verEarly[intStart] = 0;
//push(&s1,intStart);
i = intStart;
while (i!=intEnd) //当终点因为入度为零压栈、出栈时，则计算结束
{   for (j=0; j<g->verts; j++) {
        if (g->edgeMatrix[i][j]!=g->noEdge)
        {   inDegree[j]--;
            if (inDegree[j]==0) push(&s1, j);
            if (verEarly[j]<verEarly[i]+g->edgeMatrix[i][j])
                verEarly[j] = verEarly[i]+g->edgeMatrix[i][j];
        }
    }
    i = top(&s1); pop(&s1); push(&s2,i);
}
```



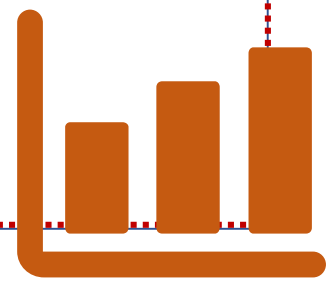
```
 printf("顶点的最早发生时间：\n");
for (i=0; i<g->verts; i++)
    printf("%d ",verEarly[i]);
 printf("\n");

//初始化顶点最迟发生时间
total = verEarly[intEnd];
for (i=0; i<g->verts; i++)
{     verLast[i] = total;
}
verLast[intStart] = 0;

 //按照计算顶点最早发生时间逆序依次计算顶点最迟发生时间
while (!isEmpty(&s2))
{
    j = top(&s2); pop(&s2);
```



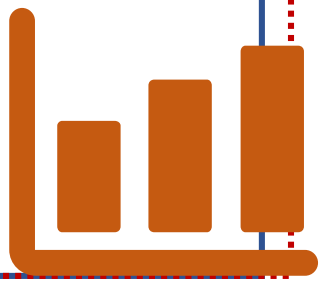
```
//修改所有射入顶点j的边的箭尾顶点的最迟发生时间
for (i=0; i<g->verts; i++)
        if (g->edgeMatrix[i][j]!=g->noEdge)
            if (verLast[i] > verLast[j] - g->edgeMatrix[i][j])
                verLast[i] = verLast[j] - g->edgeMatrix[i][j];
}

printf("顶点的最迟发生时间：\n");
for (i=0; i<g->verts; i++)
    printf("%d ",verLast[i]);
printf("\n");

   //建立边信息数组
   for (i=0; i<g->verts; i++)
    for (j=0; j<g->verts; j++)
        if (g->edgeMatrix[i][j]!=g->noEdge)
```



```
        {
            edgeEL[k].u = i;
            edgeEL[k].v = j;
            edgeEL[k].weight = g->edgeMatrix[i][j];
        }

//将边的最早发生时间<u,v>设置为箭尾顶点u的最早发生时间
//将边的最迟发生时间<u,v>设置为箭头顶点v的最迟发生时间-<u,v>边的权重
for (k=0; k<g->edges; k++)
{
    u = edgeEL[k].u;
    v = edgeEL[k].v;
    edgeEL[k].early = verEarly[u];
    edgeEL[k].last  = verLast[v] - edgeEL[k].weight;
}
```



```
//活动的最早发生时间，最迟发生时间
    printf("活动的最早、最迟发生时间:\n");
    for (k=0; k<g->edges; k++)
    {   u = edgeEL[k].u;
        v = edgeEL[k].v;
        printf("%d ->  %d :  %d  %d\n", g->verList[u],
        g->verList[v], edgeEL[k].early, edgeEL[k].last);
    }
    //输出关键活动
    printf("关键活动：\n");
    for (k=0; k<g->edges; k++)
        if (edgeEL[k].early == edgeEL[k].last)
        {   u = edgeEL[k].u;
            v = edgeEL[k].v;
            printf("%d ->  %d :  %d\n", g->verList[u], g->verList[v], edgeEL[k].early);
        }
}
```



时间复杂度分析：

找起点和终点下标花费时间O(n);

计算顶点入度花费时间O(n^2);

计算顶点最早发生时间花费时间O(n^2);

计算顶点最迟发生时间花费时间O(n^2);

建立边信息花费时间O(n^2);

计算活动的最早发生时间花费时间O(e);

计算活动的最迟发生时间花费时间O(e);

输出关键活动花费时间O(e);

因$e \leq n(n-1)$，所以总的时间代价为$O(n^2)$。

谢　谢　观　看

数据结构——C语言描述（慕课版）

人民邮电出版社
POSTS & TELECOM PRESS