AOE 网的关键路径求解算法改进及其应用

The Improvement and Application of the Key path Algorithm to AOE – net

刘小晶　（嘉兴学院信息工程学院 314001）

摘要：在项目企业生产管理中，合理估计工期是一个必不可少的环节。而 AOE 网的关键路径算法是用于此环节的核心方法。文中分析了关键路径算法的传统解决方法，并针对分析结果提出算法的改进方案，从而使算法效率得到提高。并将此算法应用于生产工序的管理中，结果表明了此方案的有效性。

关键词：关键路径 AOE 网 生产工序

1 引言

在项目管理中，合理估计工期，找出影响工程进度的关键活动，从而采取各种措施缩短工期，提高效率是生产管理者的一项核心工作。而 AOE 网的关键路径算法正是用来解决此问题的。在传统的讲解数据结构的教科书中关键路径通常是在拓扑排序的基础上求得的，算法是建立在邻接表的存储结构上。本文通过对传统的关键路径问题分析，提出了一种求关键路径的改进方案，此方案是采有图的十字链表存储结构，在按广度优先搜索的基础上实现的，此算法的时间复杂度为 O(n+e)，较传统的算法效率更高。

AOE 网的关键路径算法可用于对生产工序的完成时间的估算，现在许多企业的生产工艺是一种网络的拓扑结构，各种工序常常是并行的。因此，求出关键路径，提高关键路径上的关键活动的速度，就能缩短生产的时间，提高效率。

2 AOE 网中求关键路径的传统算法

2.1 相关概念

定义 1 给定一个有向网络图 G = (V,E)，顶点表示事件，有向边表示活动，边上的权表示完成这一活动需要的时间，则称此有向网络图为用边活动的网络或 AOE(Activity – On – Edge)—网络。

定义 2 给定 AOE—网络，在网络中，从开始顶点到结束顶点的最长路径称为该 AOE – 网络的关键路径。关键路径上的活动称为关键活动。

在 AOE 网络中，有些活动可以并行的进行。从源点到各个顶点，以至从源点到汇点的有向路径可能不止一条。这些路径的长度也可能不同。完成不同路径的活动所需的时间虽然不同，但只有各条路径上所有活动都完成了，这个工程才算完成。因此，完成整个工程所需的时间取决于从源点到汇点的最长路径，即 AOE 的关键路径(critical path)。如图 1 所示。

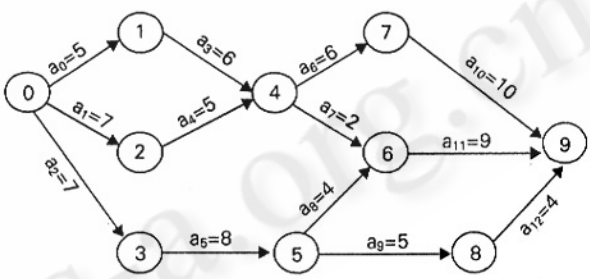

图 1　一个假想工程的 AOE——网络图

图中是一个带权的有向无环图，其中，顶点表示事件(event)，弧表示工序(procedure)，权表示工序的花费时间。整个工序只有一个源点表示生产开始和一个汇点表示生产结束，即网中只有一个入度为零的点和一个出度为零的点。

2.2 算法描述

用 E(i) 表示工序 ai 的最早开始时间，L(i) 表示工序 ai 的最迟开始时间，两者之差 L(i) – E(i) 意味着完成工序 ai 的时间余量。我们把 L(i) = E(i) 的工序叫做关键活动，因而辨别关键活动就是要找 E(i) = L(i) 的工序，为求 AOE 网中工序的 E(i) 和 L(i)，首先应求得事件的最早发生时间 VE(j) 和最迟发生时间 VL(j)。如

果工序 ai 由弧(j,k)表示,其持续时间记为 Dut(j,k),则有如下关系:

E(i) =VE(j)

L(i) =VL(k) - Dut(j,k)

求 VE(j)和 VL(j)需分两步进行:

(1) 从 VE(1) =0 开始向前递推

VE(j) =Max{VE(i) +Dut(i,j)} 其中(i,j) ∈T,2≤j≤n,T 是所有以 j 为头的弧的集合。

(2) 从 VL(n) =Ve(n)起向后递推

VL(i,j) =Max{VL(j) -Dut(i,j)} 其中(i,j) ∈S,1≤i≤n -1,S 是所有以 i 为尾的弧的集合。

传统算法中这两个递推公式的计算分别是在拓扑有序和逆拓扑有序的前提下进行,并利用图的一般表示方式邻接表的开式来解决的。

仔细考虑关键路径的含义发现其概念非常简单,就是网络图中的最长路径,因此如果能够从开始活动遍历到结束活动的所有可能的路径,在这些路径中求出最长的路径就是关键路径,关键路径上的活动自然就是关键活动。由此分析,下面给出求关键路径算法在按广度优先遍历的基础上予以实现。

3 AOE 网中求关键路径的算法改进

3.1 算法的理论基础

定 义:在图的一条路中,若出现的结点互不相同,则称其为基本路。

定理 1:在一个具有 n 个结点的图中,任何基本路的秩(弧的数目)均为不大于 n -1

证明:因任何基本路中各结点互不相同且最多为 n 个,因此其秩均不大于 n -1

定理 2:设 L 为简单有向加权图 G 中顶点 vi 到 vj 的关键路径,则 L 一定是 vi 到 vj 的基本路。

证明:若 L 不是基本路,则 L 上至少存在一个重复出现的顶点 u,于是经过结点 u 一定存在一个回路,与 G 为简单有向加权图矛盾,所以 L 一定是基本路。

定理 3:给定简单有向加权图 G,v 为源点,u 为汇点,对于图 G 的任意顶点 w,如果 w 到 v 的最长距离和 w 到 u 的最长距离之和等于关键路径的长度,则 w 一定在从源点 v 到汇点 u 的某一条关键路径上。

3.2 算法设计

基于上述事实,给出:

3.2.1 图的存储结构

图的存储结构采用十字链表形式,,在十字链表中,对应有向图中每一条弧有一个结点,对应于每个顶点也有一个结点。这些结点的结构如下:

弧结点:

Tailvex	Headvex	Hlink	Tlink	Info

顶点结点:

Data	Firstin	Firstout

C 语言关于图的十字链表存储形式描述如下:

```
#define Max 20
Typedef struct ArcBox {
    Int Tailvex, Headvex; //该弧的尾和头顶点的位置
    Int Weight; //该弧的相关信息
    Struct ArcBox *Hlink, *Tlink; //分别为弧头相同和弧尾相同的弧的链接域
}ArcBox;
typedef struct VexNode {
    Vertextype Data;
    ArcBox *Firstin, *Firstout; //分别指向该顶点第一条入弧和出弧
}VexNode;
typedef struct{
    VexNode Xlist[Max]; //表头向量
    Int Vexnum, Arcnum; //有向图的当前顶点数和弧数
}OLGraph;
```

为了求出每个源点和汇点的最长距离,分别设置辅助数组 path1[Max]和 path2[Max],其中 path1[Max]存储源点到结点 i 的最长距离,path2[Max]存储结点 i 到汇点的最长距离。为了判断图是否有环,设置辅助数组 count[Max], count[k]存储从发点到结点 k 的路径的秩。

3.2.2 算法描述

(1) 对数组 path1[ ]、path2[ ]和 count[ ]进行初始化,初始化值为 0;

(2) 对图 G 从源点开始进行广度优先搜索,求出源点到其余各顶点的最长距离 path1[i](i =0,1,…,n

-1). 若在求解过程中某个 count[i]大于 n-1,则说明该图有环,算法终止;否则转 3);

(3) 对图 G 从汇点开始进行广度优先遍历,求出各顶点 k 到汇点的最长距离;

(4) 对图 G 从源点开始进行广度优先搜索,依据条件 path1[i]+path2[i]=path2[0]求出所有关键活动。

3.2.3 算法实现

```
void CirticalPath( OLGraph G)
{ int i, j, count[Max], path1[Math], path2[Math];
 ArcBox *p;
 LinkQueue Q; //Q 存储遍历顶点系列;
 For (i=0; i<G.Vexnum; i++)
 { count=0; path1[i]=0; path2[i]=0};
 //下面开始求 0 号顶点到其余各个顶点的最长距离
 InitQueue (Q) ;
 EnQueue(Q);
 While ( ! QueueEmpty(Q)
 { DeQueue(Q,j);
  for (p=G.Xlist[j].Firstout; p=p->Tlink)
  { i=p->Headvex;
   count[i]=cout[j]+1;
   If (count[i]>G.Vexnum-1)
   { printf("该图有存在环,算法终止!");
    exit(-2);
   }
   If (path1[j]+p->Weight>path1[i])
path1[i]=path1[j]+p->Weight;
   Enqueue(Q,i);
  }
 }
 //下面开始求其余各个顶点到 G.Vexnum 号顶点的最长距离
 EnQueue(Q, G.Vexnum-1);
 While ( ! QueueEmpty(Q))
 { DeQueue(Q,j);
  for (p=g.xlist[j].Firstin; p; p=p->hlink)
  { i=p->Tailvex;
   If (path2[j]+p->Weight>path2[i]) path2
[i]=path2[j]+p->Weight;
   EnQueue(Q,i);
  }
 }
 //下面求所有关键活动
 EnQueue(Q,0);
 While ( ! QueueEmpth(Q))
 { DeQueue(Q,j);
  for(p=G.Xlist[j].Firstout; p; p=p->Tlink)
  { i=p-<Headvex;
   If ( path1[i]+path2[i]=path2[0]&&(path1
(i)==path1[j]+p->Weight))
   {printf("(%d%d)",j,i);
    EnQueue(Q,i); }
  }
 }
 DestroyQueue(Q);
}
```

此算法的时间复杂度与广度优先遍历图的时间复杂度相同为 O(n+e),通过实验比较,该算法较传统的算法效率明显提高。

4 算法在生产工序中的应用

此算法已成功应用于一个车辆修理信息管理系统中。因车辆修理工艺实际上就是一个网络的工序图,虽然不同车间的修车工序各不相同,但它们都是 AOE 网,并且用户需要掌握的信息也一样,所以运用了面向对象的方法将此算法进行封装以便于代码复用。

4.1 算法的实现方式

采用 java 这种面向对象的程序设计语言及后台数据库 SQL-SERVER。根据 AOE 网的关键路径算法还需做如下改进:将生产工艺中的数据信息存储在表中;用 Java 在前台对数据进发生行采集和使用;将算法和数据分离封装在中间件中。这样用户可以方便数据的维护。例如:工序发生变化,用户仅需将表中的工序信息进行调整而程序不用修改;另外将程序扩充为一个分布式组件可以平衡整个系统的负载,例如不同车间对生产工序的计算,虽然工序不同,而且是用不同的机器进行处理,但是,通过调用封装后的分布式组件,可以将这些同类的计算功能用一个服务器进行处理,而

(下转第 53 页)

(上接第 49 页)

不同车间的机器则专门处理前台的交互,这样真正实现了系统工程分布式体系结构。

4.2 算法的考虑

后台采用数据库 SQL Server,因此顶点信息用表的形式建立 AOE 的存储结构

字段名	后继结点	当前结点	工序代码	工序名称	工序耗时	入度
字段类型	整型	整型	字符型	字符型	整型	整型

然后用 java 语言编写好算法代码,再根据用户需求,从预检车辆的故障出发,自动调整各工序的修时,最后根据计算,得出不同车辆的最长修时,自动安排车辆的修理台位,进一步实现计算机的辅助决策功能。

5 小结

本文在广度优先遍历的基础上,给出了一种新的求解关键路径的算法,该算法采用图的十字链表的结构形式,不需要进行拓扑排序只需在遍历的基础上既可求出所有关键活动,其算法性能也较传统算法效率更高。同时也尝试用面向对象语言在一个信息管理系统中实现了此算法。

参考文献

1 严蔚敏、吴伟民, 数据结构(C 语言版)[M] 北京 清华大学出版社,1997。

2 Bruce ckel . inking in Java (Third Edition),机械工业出版社,2005 。