一个项目进度管理软件的核心算法优化

Optimization of the Core Algorithm of Project Progress Management Software

赵洪兵　陈力军　王　超　潘金贵

（软件新技术国家重点实验室　南京大学计算机科学与技术系　南京210093）

Abstract　This paper discusses the tree——a core data structure of one kind of project progress management software, and the implementation and optimization of algorithms of searching and traversal. In the algorithm D&C method is used throughout the process. We also try to change method of backtracking some key nodes so as to increas the speed of traveling throughout the tree.

Keywords　Tree,　Traversal,　Backtrack

1　引言

Sisfird是日本富士通公司早期开发的用于管理工程开发过程的管理软件，在长期使用中总存在速度太慢的缺点，给用户带来了很大的不便，影响了该软件的推广，为此，我们接受日本富士通公司委托对Sisfird做改进，实现在Win32平台下软件运行速度提高一个数量级的目标。

Sisfird采用基于树的数据结构，程序开销集中在树遍历与回溯计算上。本文在提及运用算法抽象进行程序优化的同时，阐述涉及回溯的树的遍历，根据软件开发的实际提出一种新的树的回溯遍历优化算法，并对该算法进行了分析。

2　数据结构

2.1　数据总体逻辑结构

在一个大型项目中，子项目具有非单一性，子项目之间具有包容性，及由此带来的层次性，子项目的确定具有动态性。考察一下常用的数据结构，可以发现树形结构能很好地表达这些特性。因此，在Sisfird中，核心数据结构就是一棵倒置的多叉树，树的每个结点包含有指向父结点的信息和指向子结点的信息。同时，为了便于顺序访问各结点，在树的实现时采用结合链表结构和数组结构的方法：各结点按先根顺序[2]排列，组成一个全局数组，数组下标是结点在先根遍历过程中的序号。在结点中设一个字段表示结点的层次信息，该字段的值在结点生成时填入。为了叙述方便，在以下文字中，除非有特别说明，把在Sisfird中设计的这棵树称为数据树。

在Sisfird中，涉及到数据树的操作主要分为两部分，即视图部分和文档部分。在视图部分，数据树按倒置树形结构显示（如图1所示）。注意到在视图中每个结点数据都对应着一个序号，即视图序号。因为这个序号与结点是否在视图中显示有关，所以它有别于前面提到的结点数组下标值。从图1中可以看到视图序号为9的结点Task3-3是关闭着的，虽然它有子结点，但是在视图中其子结点将不被显示。所以，后续结点如视图序号为10的结点Task3-4在数组中的下标值（见图2）与视图序号不同。

	作业名
1	[-]Task1
2	.[ ]Task1-1
3	.[ ]Task1-2
4	.[ ]Task1-3
5	[ ]Task2
6	[-]Task3
7	.[ ]Task3-1
8	.[ ]Task3-2
9	.[+]Task3-3
10	.[-]Task3-4
11	.[ ]Task3-4-1

图1　数据树在视图中的表示

在文档部分，涉及数据的存取方式，数据树以成员变量的形式保存在文档类中。为了便于访问树的结点，把数据树封装成一个数组，因而在用户看来对于树的访问完全可通过对数组的随机访问实现。然而这带来了一个问题，就是数组不便于动态扩充，而数据树的结点却是动态变化的。Sisfird早期版本的开发者们考虑到这个问题，就结合链表可动态变化的特点，用顺序链表的方式来实现数组。通过自己定义数组的实现，使该数组具有动态变化的特点，且具有较高的访问命中效率。限于篇幅，本文不对该数组的实现作详细叙述。

2.2　与结点删除、插入有关的操作

图3所示为树结点的简图，图中各字段的含义为：

指向子女的指针数组、结点在树中的层次信息、结点数据信息、结点子女开闭指示、指向父结点的指针。其中bOpen字段只有在该结点是内结点时才有意义，Opened表示子女结点在视图中显示出来(如图1中Task1，Task3、Task3-4)，Closed表示子女结点不在视图中显示(如图1中Task3-3)。图2为数据树的全局表示。在图中，结点按数组顺序排列，这也是文档中数据树的排列方式，左侧数字是结点下标值。结点各字段信息含义同图3所示。

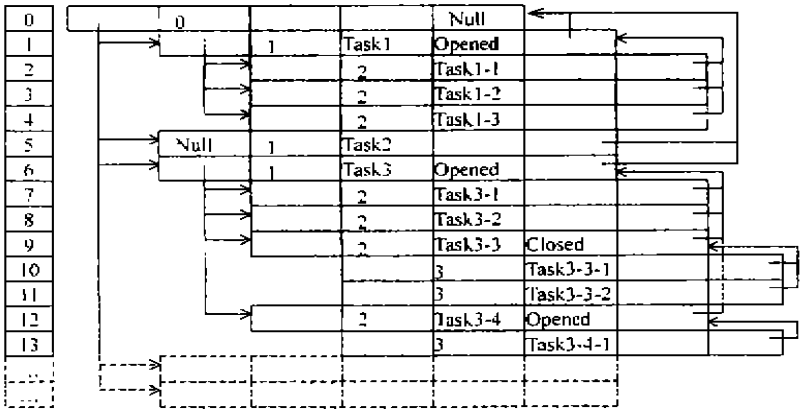

图2 数据树的逻辑结构

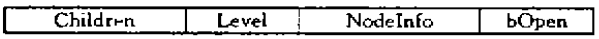

图3 结点信息

出于提高访问速度及便利软件其他计算的考虑，以视图显示的项目列表是用可扩充的指针数组来实现的。用户对结点的操作是在视图方式下完成的，所以结点处理需要把视图方式下得到的结点定位到文档方式中，同时要把文档方式下处理的结点映射到视图中，以图3为例，假如用户要给当前结点Task3增加一个子任务，其处理过程如图4所示。

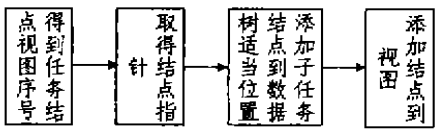

图4 插入结点过程

原来在添加子任务节点时，为了找到其适当的视图位置(要求是当前任务所有的子任务之后)，采用了逐一进行匹配的算法，这样遍历整棵子树的代价很大。现在，通过抽象分析后，问题被简化为找出该子树的所有在视图上可见的子女的总数。

实际中，具体一个项目的子项目层次不会很多，故数据树层次大于4的概率很小。在实现中，我们仍采用了非递归的算法，这样能更好地结合数据树的具体特性加以优化，比如利用子树中存在未被显示的子女来减少子树层次，等等。实际应用证明，经过如此的算法改进及优化，添加子女的速度大大提高。

3 相关计算

Sisfird计算中最重要、也是最耗时的部分是工程进度率的计算。图5、图6所示是数据树结点字段中和进度计算有关的数据量。

计划开始日期
计划结束日期
实际开始日期
实际结束日期(RE)
计划工程量(A)
实际完成工程量(R)
权重(W)
进度率(S)

图5 进度计算用到的结点数据

A	项目计划工程量
A'	子项目计划工程量
R	项目实际完成工作量
R'	子项目实际完成工程量
RE	项目实际结束日期
RE'	子项目实际结束日期
S	项目进度率
S'	子项目进度率
W	项目权重值
W'	子项目权重值
Tq	进度报告日(q)项目进度率
Tq'	进度报告日(q)子项目进度率

图6 进度计算中用到的字符含义

进度计算分两种，一种是计算任务完成后，实际工作量和预定工作量的比例，称为进度率的计算；另一种

是计算在任务实际完成日之前某个工作日的工程实际完成情况，称为进度报告计算。后一种计算结果能体现出工程前后进展情况，是评估项目实施人员作业开展情况的很好依据。

3.1 进度率的计算

表1所示是进度率的计算规则，表中用到的公式及公式中的数据项意义如图7所示。从表中可知，进度率的计算公式有两个，分别表示用项目预定工作量，项目实际工作量计算及用子作业的进度率计算加权平均值两种情况。这两种情况的计算都比较直接，可通过直接对数据树的遍历完成。在遍历过程中不涉及树的回溯，只需简单判断采用公式1还是公式2即可。由文[3]可知，在这种情况下算法的时空复杂度可囿于 n，即假设单个结点操作的时空占有量为 Θ(1)的话，程序的时间复杂度为 t(N)=Θ(N)，s(N)=Θ(N)。该复杂度值对于遍历树的三种基本策略[2]都是一样的，在实现时，由于 T'q 要先于 Tq 计算，所以采用后根遍历策略实现对树的遍历，详细遍历算法可参考文[2]。

表 1 进度率计算规则

进度率直接输入	子项目	子项目进度率	预定工作量实际工作量	计算方法及公式	例外
有直接输入	--	--	--	不计算	--
能被计算	存在	存在	--	根据子项目的进度率计算加权平均值，用 Form1	a)S'n 不包括有 event 属性项目 b)S'n 不包括 Don't Calculate 属性项目
能被计算	存在	不存在（也无法计算）	存在	根据项目本身预定工作量和实际工作量计算，用 Form2	--
能被计算	存在	不存在(也无法计算)	不存在	不计算	--
能被计算	不存在	--	存在	根据项目本身预定工作量和实际工作量计算，用 Form2	--
能被计算	不存在	--	不存在	不计算	--

S 进度率
n 子项目数目

Form1 $S=(\Sigma S'n \times W'n)/\Sigma W'n$
$S''=(\Sigma S''n \times W''n)/\Sigma W''$
Form2 $S=R/A$

图7 进度率计算公式及公式数据项意义

3.2 进度报告日报告率计算

进度报告日报告率的计算公式有三个，图8所示是进度报告计算使用到的三个公式和公式中符号的含义。

其中公式1用于父结点项目的进度率未知，而其子结点进度报告率可计算时；公式2用于父结点项目的进度率是直接输入得到的，并且其子结点的进度报告率可计算；公式3用于直接用结点本身的数据值计算进度报告率。表2列出了计算进度报告的规则。

由表2可知，进度报告的计算体现在 Sisfird 的数据树中就是按照树的每个结点的相应数据（如图5所示），先计算子结点的进度率(Progress Rate，T'q)，然后按照结点的层次分布，根据子结点的进度情况和子结点与父结点工程开展日期的关系，推算父结点的进度率(Progress Rate，Tq)。对于树的遍历，这儿仍采用后根次序。设进度报告计算时每个结点的时间占有量为 Θ(f(k))，则由前面的叙述可知，无回溯遍历 N 结点树的时间复杂度为 T(N)=Θ(N*f(k))。

n 子项目数目
m 进度报告日之前完的子项目数目
n>m

S=ΣTqm
S'=ΣT'qm

Form1 $Tq=(\Sigma T'qm \times W'm)/\Sigma W'n$
$T'q=(\Sigma T''qm \times W''m)/\Sigma W''n$

Form2 $Tq=(\Sigma R'm)/A \times 100$
$T'q=(\Sigma R''m)/A \times 100$

Form3 $Tq=(\Sigma T'qm \times W'm)/\Sigma W'n \times S/((\Sigma S'n \times W'n)/\Sigma W'n)$
$T'q=(\Sigma T''qm \times W''m)/\Sigma W''n \times S/((\Sigma S''n \times W''n)/\Sigma W''n)$

图 8 进度报告计算公式及符号意义

表 2 进度报告率计算规则

子项目进度率	进度率直接输入	项目及子项目实际工作量	计算过程描述	计算公式
存在(或可计算得到)	存在	--	把直接输入的进度率按各报告日进度进展比例分配	Form3: Tq
存在(或可计算得到)	不存在	--	直接按子项目在各进度报告日的完成情况计算	Form1: Tq
不存在	--	存在	无子项目情况选择符合条件的报告日直接计算	Form2: Tq
不存在	--	不存在	不用计算	--

进度报告率的三个计算公式可分成两类，一是针对对应于数据树叶结点的项目的计算，即公式3；另一个是针对对应于数据树内结点的项目的计算，即公式1和2。用公式1和公式3计算时，都需要用到 T'q。由前面的定义知，T'q 是子项目的进度报告率，在采用树的后根遍历的算法中，T'q 由前一次树的遍历过程计算而得，因而这种计算需要进行树的回溯，由于这两类计算的计算过程截然不同，并且对应于树回溯操作的处理也不同，故在实现中对它们实行"分而治之"的策略，如图9。

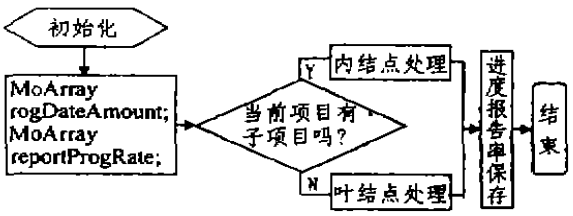

图9 分而治之策略

在计算某结点时，所有对树的回溯操作看作遍历树时的结点运算的一部分，它不影响树的全局遍历次

序，本文中将不再赘述。

下面将阐述在树的全局遍历过程中处理结点运算时涉及到的回溯操作，在具体的处理结点的计算流程中，按分治策略把对'叶结点'和'根结点'的处理分开来，以实现处理时的高效率。在处理叶结点时，关键是找到当前报告日，并计算此报告日的报告进度率。具体报告日定位和报告日进度率的计算策略由于涉及很多软件的琐碎细节，在此不赘述。当树的遍历回到该叶结点的上层父结点时，需要回溯以便取得此时计算的结果。在早期的 Sisfurd 版本中，所有计算所得的子项目进度报告日报告率保存在全局数据树中，回溯时通过对当前内结点的子树再次实行遍历而实现。对一棵有 N 个结点的树，我们假设每个内结点平均有 k 个子女，则这棵树的层数 $n=\log_k(k-1)N+1$，如果在每个内结点上都需要对其所有直接子女进行回溯，设在每个内结点上的回溯运算量为 O(k*s)(s 是回溯访问一个子结点时的运算量)，那么对于高度为 n 的树回溯运算量就为：

$$T(n)=O(\sum_{i=0}^{n-1}k^i(k,s))=O(k^{n+1},s)$$

当 s 的值比较大时，由上述公式可知 T 值将以几何级数增长。当然 T(n)的值还与 k、n 有关，但要实现树的遍历运算，k、n 的值是没办法作优化的。要使 T(n)的值控制在能容忍的范围内，关键在于设法控制 s 的值，使之不会增长过快。

为了控制 s 的值，必须设法降低对树的回溯时的运算量。在早期版的 Sisfurd 中，回溯是通过遍历所有直接子女的方法实现的。树的回溯操作是基于链表的，对于图3所示的数据树，在后根遍历到某个父结点时，可通过指向其子女链表的指针取得子女信息，从而对子女的内容进行回溯访问。这个操作包括两个过程，首先取得指向子女链表的指针，然后再根据子女序号取得子女结点指针。因为在实现子女链表时用的是系统定义的数组式链表[1]，而这个操作相对而言是比较费时的。考虑到对子女结点的回溯访问是按子女的序号随机访问的，并且在对某个结点处理时所有回溯仅限于其直接子女，在回溯完成后，被回溯访问过的结点不会再在其他回溯过程中被访问，所以我们在改进程序中把原来的保存在子树中的进度报告日报告率临时保存在一个动态数组中，每次回溯不用再到数据树中去查找，直接在动态数组中通过下标映射实现随机存取。

3.3 算法分析

下面分析一下 s 的值是如何有效地被控制的。先定义两个数据结构，第一个是长度为 n 的单链表 L，另一个是有 n 个元素的一维数组 A，在程序中，对 L 和 A 的访问完全是随机的，假设处理 L 和 A 中的某个元素的时间是相同的，设为 t，则由文[1]可知，遍历 L 所需时间 T(L)=t*n*(n+1)/2，遍历 A 所需时间 T(A)=n*t。

在 Sisfurd 早期版本中，由于要实现树的动态生长，很直观地就采用单链表作为计算数据的中间存储池。但在实际使用中，树的每一层上结点数目一般总是在500以内，对于超出500的部分，我们仍用链表作为存储池，但这种情况在实际应用中几乎不出现。对于500个结点随机遍历，单链表花费时间 T(L)=125250t；而一维数组只需时间 T(A)=500t，因而核心计算部分的效率的提高是很显著的。

结束语 本文所论述的具体问题的算法抽象及带有回溯的树的运算的算法优化在新版本的 Sisfurd 中得到运用，在日本富士通公司与南大富士通软件技术有限公司的评测中，证明其具有较高的效率，获得了用户满意的效果。

参考文献

1 Microsoft MSDN
2 陈本林，陈佩佩. 数据结构. 南京大学出版社，1999
3 余祥宣，崔国华，邹海明. 计算机算法基础. 华中理工大学出版社，2000
4 Kruglinski D J 著，潘爱民，王国印 译. Visual C++技术内幕(第四版). 清华大学出版社，1998
5 Sisfurd 项目文档. 富士通公司，1996

(上接第104页)

模型，包括对信度网结构的修改及对条件概率表的调整。最简单的方法是利用浏览数据库，脱机对信度网进行更新，称为批量更新。当然，也可以进行在线更新，但由于信度网的更新所需的计算量比较大，因此可能会对在线导航造成影响。

本文中的在线导航系统存在的一个问题是，它没有利用浏览者在网站上的浏览顺序这一信息。可以设想，通过分析用户浏览各个主题的先后顺序，可能会产生更准确的导航信息。对此，还可做进一步的研究。

参考文献

1 Heckerman D. Learning Bayesian Network: The Conbination of Knowledge and Statistical Data. [Technical Report MSR-94-09]
2 Rissanen J. Stochastic Complexity in Statistical Inquiry. World Scientific, River Edge, NJ, 1989
3 Chen J, Bell D A, Liu W. Learning Belief network from data. An information thoery based approach. In Proc of ACM CIKM'97
4 Jensen F V, Lauritzen S L, Olesen K G. Bayesian updating in causal probabilistic networks by local computations, Comp Stat Quart, 1994, 4: 269～282