多关键字数据结构及其在参数匹配中的应用①

余　洪　(中国移动(深圳)有限公司 广东 深圳 518048)

摘　要： 介绍常用数据结构的局限性，提出了多关键字数据结构的设想及其匹配算法，并讨论了在参数匹配中的使用。

关键词： 数据结构 关键字 参数

Data Structure of Multi-Keyword

YU Hong

(China Mobile (ShenZhen) Limited, Shenzhen, Shenzhen 518048, China)

Abstract: This Paper is about the limitations of the commonly used data structure. It puts forward an idea for Date Structure of Multi-Keyword and its matching algorithms. Finally, it discusses the use of matching parameters.

Keywords: data structure; multi-keyword; parameter

1　简介

目前常用的数据结构，主要包括数组(静态数组、动态数组)、线性表、链表(单向链表、双向链表、循环链表)、队列、栈、树(二叉树、查找树、平衡树、线索树、堆)、图[1]等 。

这些数据结构一个共同的特点，是都只有一个关键字。在这些数据结构的应用中，往往是通过对这个关键字的比较，来实现各种排序和查找算法，如二分查找法、冒泡排序法等。

然而，在现实生产中，有时候却往往存在不能使用单关键字的情况，在笔者的工作经验中，就存在参数匹配时有多个关键字的情况，且每个关键字还有不同的比较算法，不能使用通用的数据结构，在开发上需要额外花费功夫。

典型的例子，就是带生效时间/失效时间的三关键字的参数模型。如：关键字 a，生效时间 b，失效时间 c，数据信息 d。其中，前三个都是关键字，代表着关键字 a 在生效时间 b 到失效时间 c 之间，使用的数据信息是 d。

这种模型，关键字 a 的匹配算法是“=”，而关键字 b 的匹配算法是“>=”,关键字 c 的匹配算法是“<”。

对于这种模型，常用的数据结构完全不能支持。而由于这三个关键字的匹配算法有不同的方式，因此也不能简单的把三个关键字合成为一个关键字来使用通常的数据结构。

2　多关键字数据结构的设想

为了解决上述问题，需要对目前的数据结构进行改进和加强，来适应这种多关键字的数据模型。

通过对于三关键字的参数模型的仔细分析，对于这种多关键字的数据结构，有两个关键之处，同时也是特殊之处。

(1) 有多个关键字，其个数可以是 2 个、3 个甚至更多。

(2) 每个关键字，都可以有不同的关键字匹配算法。

每个关键字都需要单独为其定义一个匹配规则，前面的例子已经有了“=”、“>=”、“<”，同样，还可以有更多的匹配算法，如“<=”、“>”，甚至还可能有“包含”、“被包含”等(后文中，将使用“{”表示“包含”，使用“}”表示“被包含”)。

① 收稿时间:2009-05-11

对于这种多关键字的数据结构，其逻辑结构和匹配算法还需要进行分析。

3　多关键字数据结构的逻辑结构

常用的数据结构只有一个关键字，以二叉树为例，其逻辑结构为：

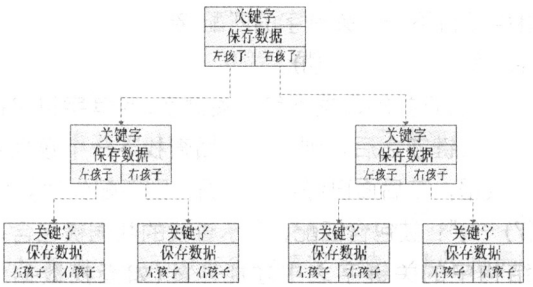

图 1　通常二叉树逻辑结构

而多关键字的数据结构，以二叉树为例，其逻辑结构为：

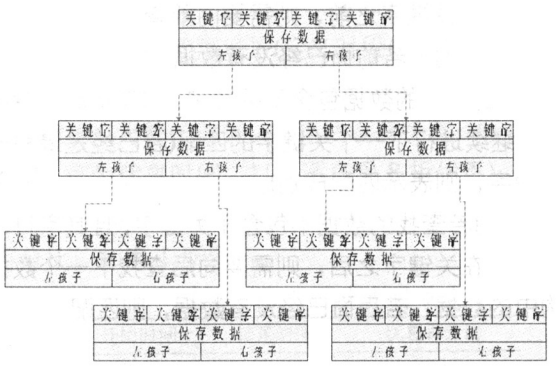

图 2　多关键字二叉树逻辑结构

以上可以看出，多关键字的数据结构，其逻辑结构和常用的数据结构相比，只是一个关键字拆分成了多个关键字，其他的地方可以保持不变。

4　多关键字数据结构的匹配规则

通常单关键字的数据结构，其匹配规则是相等匹配，即：

(1) 若查找数据等于关键字，则匹配成功；

(2) 若查找数据小于关键字，则需要向前继续匹配，若向前已经没有可以匹配的数据，则匹配失败；

(3) 若查找数据大于关键字，则需要向后继续匹配，若向后已经没有可以匹配的数据，则匹配失败。

其参考图如图 3 所示。

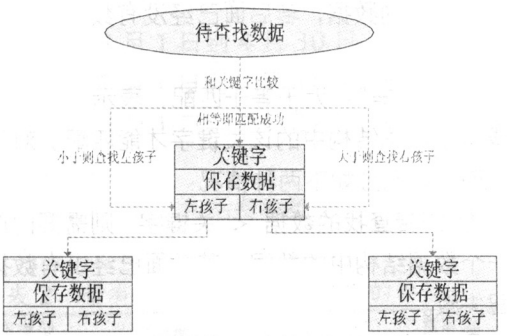

图 3　通常二叉树匹配规则

而多关键字的匹配规则则有所不同，首先是需要进行多个关键字的匹配，其次是在每个关键字匹配时，还可以有非相等匹配的算法，具体解释如下：

从第一个关键字开始，按照每个关键字的匹配算法，依次匹配每个关键字，若每个关键字都匹配成功，则表示匹配成功；若在某个关键字处匹配失败，则按匹配算法向前或者向后查找下一个数据结构中的数据。若向前或向后已经没有可以匹配的数据，则匹配失败。其参考图如图 4 所示。

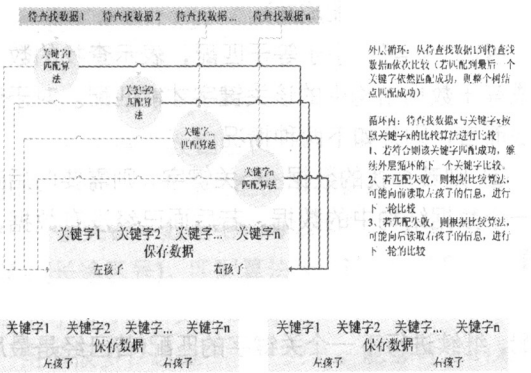

图 4　多关键字二叉树的匹配规则

其中每个关键字的匹配算法如下：

(1) “=”：等号匹配，表示要查找的数据与数据结构中的该关键字必须完全相等才能匹配，否则表示不匹配。对于查找算法而言，则分如下三种情况：

① 若查找的数据 < 关键字，则向前查找前一个数据结构中的数据，若前面已经没有数据，则匹配失败。

② 若查找的数据与 = 关键字，则表示此关键字匹配，需要继续进行下一个关键字的匹配(若已经是最后一个关键字，则表示匹配成功)；

③ 若查找的数据 > 关键字，则向后查找下一个

数据结构中的数据，若后面已经没有数据，则匹配失败。

(2) ">=": 大于等于匹配，表示查找的数据大于或等于数据结构中的该关键字才能匹配。对于查找算法而言，则分如下两种情况：

① 则若查找的数据 < 关键字，则需要向前查找前一个数据结构中的数据，若前面已经没有数据，则匹配失败。

② 若查找的数据 >= 关键字，则表示此关键字匹配，继续进行下一个关键字的匹配(若已经是最后一个关键字，则表示匹配成功)。

(3) ">": 大于匹配，表示查找的数据大于数据结构中的该关键字才能匹配。对于查找算法而言，则分如下两种情况：

① 则若查找的数据 <= 关键字，则需要向前查找前一个数据结构中的数据，若前面已经没有数据，则匹配失败。

② 若查找的数据 > 关键字，则表示此关键字匹配，继续进行下一个关键字的匹配(若已经是最后一个关键字，则表示匹配成功)。

(4) "<=": 小于等于匹配，表示查找的数据小于或等于数据结构中的该关键字才能匹配。对于查找算法而言，则分如下两种情况：

① 则若查找的数据 > 关键字，则需要向后查找下一个数据结构中的数据，若后面已经没有数据，则匹配失败。

② 若查找的数据 <= 关键字，则表示此关键字匹配，继续进行下一个关键字的匹配(若已经是最后一个关键字，则表示匹配成功)。

(5) "<": 小于匹配，表示查找的数据小于数据结构中的该关键字才能匹配。对于查找算法而言，则分如下两种情况：

① 则若查找的数据 >= 关键字，则需要向后查找下一个数据结构中的数据，若后面已经没有数据，则匹配失败。

② 若查找的数据 < 关键字，则表示此关键字匹配，继续进行下一个关键字的匹配(若已经是最后一个关键字，则表示匹配成功)。

(6) "{": 包含匹配，表示查找的数据包含了数据结构中的该关键字才能匹配(如查找数据为 "1232"，数据结构中的关键字为 "123"，则匹配，本算法仅仅能用于字符串比较)。对于查找算法而言，则分如下三种情况：

① 则若查找的数据不包含关键字，且其字符串排列顺序在关键字之前，则需要向前查找前一个数据结构中的数据，若前面已经没有数据，则匹配失败。

② 若查找的数据包含关键字，则表示此关键字匹配，继续进行下一个关键字的匹配(若已经是最后一个关键字，则表示匹配成功)。

③ 则若查找的数据不包含关键字，且其字符串排列顺序在关键字之后，则需要向后查找下一个数据结构中的数据，若后面已经没有数据，则匹配失败。

(7) "}": 被包含匹配，表示查找的数据被包含在数据结构中的关键字之中才能匹配(如查找数据为 "123x2"，数据结构中的关键字为 "123x25"，则匹配，本算法仅仅能用于字符串比较)。对于查找算法而言，则分如下三种情况：

① 则若查找的数据不包含于关键字，且其字符串排列顺序在关键字之前，则需要向前查找前一个数据结构中的数据，若前面已经没有数据，则匹配失败。

② 若查找的数据包含于关键字，则表示此关键字匹配，继续进行下一个关键字的匹配(若已经是最后一个关键字，则表示匹配成功)。

③ 则若查找的数据不包含于关键字，且其字符串排列顺序在关键字之后，则需要向后查找下一个数据结构中的数据，若后面已经没有数据，则匹配失败。

5 多关键字数据结构的设计

多关键字数据结构，可以参考目前常用的数据结构来设计，包括数组、线形表、链表、队列、栈、树等，其基本的逻辑结构都保持不变，但是需要调整其关键字的设计和匹配算法，使其支持上述的多关键字模型和多关键字匹配算法即可。

以通常的树结构为例：

其初始化为： tree tree1

其增加节点为 tree1.addnode(关键字，内容)

其查找函数为 treenode1 = tree1. findnode ("查找数据")

若在树中发现关键字与 "查找数据" 完全匹配的数据，则认为查找成功，将此结果返回。

若设计为多关键字的数据结构，则需要其支持多关键字及多匹配算法，因此，其设计如下：

其初始化为 ntree tree1(n,“匹配算法 1、匹配算法 2、匹配算法 3、…匹配算法 n”) --其中 n 表示有 n 个关键字,“匹配算法 x”表示是第 x 个关键字的取值算法。(在具体实现的时候,从“匹配算法 1”到“匹配算法 n”可以用一个变量来输入,每个“匹配算法”之间使用特定的分隔符分隔,以达到实现关键字个数任意扩展的要求,这里假定为“、”)

其增加节点为:tree1.addnote(“关键字 1、2、3…n”,内容)

其查找为 tree1.findnode(“查找数据 1、2、3…n”)

这样,只要设计好一套多关键字的数据结构及其查找算法,就可以在多种情况下使用这些通用的多关键字数据结构,来满足各种不同的需求和用途。

6 多关键字数据结构在参数匹配中的应用

多关键字的数据结构有多种用途,特别是在参数匹配上,有着丰富的用途。举例如下:

(1) 分时段优惠场景(定义时段优惠如下:每天 0 点到 8 点为特惠时间,打电话享受 5 折优惠;每天 8 点到 20 点为繁忙时间,打电话没有优惠;每天 20 点到 24 点为优惠时间,打电话享受 8 折优惠)

① 整理其信息,包括开始时间、结束时间、优惠百分比,如表 1 所示:

表 1 分时段优惠场景表

开始时间	结束时间	优惠百分比
0 点	8 点	50%
8 点	20 点	0%
20 点	24 点	20%

其中,开始时间和结束时间都是关键字,优惠百分比为数据信息。

② 这种场景,单关键字数据结构无法支持。

③ 多关键字数据结构可以实现。实现方式为:两关键字(开始时间、结束时间),匹配算法为(>=、<)。即满足当前时间>=开始时间,且当前时间<结束时间,则匹配成功。

(2) 带有生效日期和失效日期的参数(定义费率参数表如下:从 3 月 1 日到 4 月 30 日,通话费为 0.6 元每分钟;从 5 月 1 日到 7 月 31 日,通话费为每分钟 0.5 元;从 3 月 1 日到 7 月 31 日,长途费为每分钟 0.4 元)

① 整理其信息,包括费率类型、生效日期、失效日期、费率值,如下:

表 2 带有生效日期和失效日期的参数表

费率类型	生效日期	失效日期	费率值
通话费	3 月 1 日	4 月 30 日	0.6
通话费	5 月 1 日	7 月 31 日	0.5
长途费	3 月 1 日	7 月 31 日	0.4

其中,费率类型、生效日期、失效日期为关键字。

② 这种场景,单关键字数据结构无法支持。

③ 多关键字数据结构可以实现。实现方式为:三关键字(费率类型、生效时间、失效时间),匹配算法为(=、>=、<)。即满足费率类型匹配,且话单日期>=生效时间,且话单日期<失效时间,则匹配成功。

(3) 长途区号模糊匹配(在电信领域中,往往需要判断呼叫号码中的长途区号,即若呼叫号码为 12345,某地的长途区号为 123,则呼叫号码匹配成功,体现为呼叫号码拨打了该地的长途电话。在长度不定的国际长途区号匹配时尤为有用)

① 单关键字数据结构没有实现模糊匹配的匹配算法(因为其只有相等匹配的匹配算法)。

② 多关键字数据结构可以实现。实现方式为:一关键字(长途区号),匹配算法为“}”。即满足长途区号包含于呼叫号码,则匹配成功。

用途很多,这里不一一列举。能够肯定的是,若有一套完整的多关键字的数据结构,并通过面向对象的实现,就可以方便的应用于多种软件生产活动中,带来更高的效率和可靠性。

参考文献

1 许卓群,张乃孝,杨冬青,唐世渭.数据结构.北京:高等教育出版社.