(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 107431660 B
(45)授权公告日 2020.08.18

(21)申请号 201680014265.4

(22)申请日 2016.01.29

(65)同一申请的已公布的文献号
申请公布号 CN 107431660 A

(43)申请公布日 2017.12.01

(30)优先权数据
2015-048657 2015.03.11 JP

(85)PCT国际申请进入国家阶段日
2017.09.07

(86)PCT国际申请的申请数据
PCT/JP2016/052664 2016.01.29

(87)PCT国际申请的公布数据
WO2016/143405 JA 2016.09.15

(73)专利权人 NTT通信公司
地址 日本东京都

(72)发明人 浅井大史 小原泰弘

(74)专利代理机构 北京三友知识产权代理有限公司 11127
代理人 黄纶伟 欧阳琴

(51)Int.Cl.
H04L 12/745(2006.01)

(56)对比文件
CN 102741841 A,2012.10.17,
CN 102741841 A,2012.10.17,
CN 101484895 A,2009.07.15,
CN 104052669 A,2014.09.17,
CN 102739551 A,2012.10.17,
CN 101911068 A,2010.12.08,
US 7831626 B1,2010.11.09,
US 7539153 B1,2009.05.26,
US 7523218 B1,2009.04.21,
WO 2004040400 A2,2004.05.13,
US 6522632 B1,2003.02.18,

审查员 楼芃雯

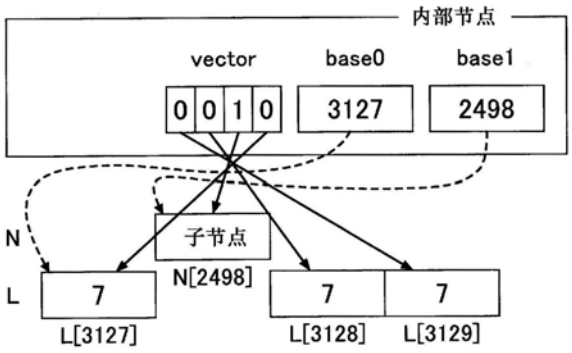
权利要求书3页 说明书12页 附图14页

(54)发明名称

检索装置、检索方法及记录介质

(57)摘要

一种检索装置，具有：存储单元，其存储检索对象数据；以及运算单元，其根据密钥数据进行对所述检索对象数据的检索处理，在所述存储单元中存储的所述检索对象数据是具有内部节点排列和叶节点排列的多叉树结构的数据，所述检索对象数据中的各内部节点包括用比特表示转移目的地是内部节点还是叶节点的比特向量，所述运算单元反复执行如下处理，一直到转移目的地成为叶节点为止，所述处理是：从密钥数据中取得规定比特长度的块，根据所访问的内部节点的所述比特向量中与该块的值对应的比特，判定从该内部节点起的转移目的地是内部节点还是叶节点并访问转移目的地的节点。



CN 107431660 B

1.一种检索装置,具有:存储单元,其存储检索对象数据;以及运算单元,其根据密钥数据进行对所述检索对象数据的检索处理,其特征在于,

在所述存储单元中存储的所述检索对象数据是具有内部节点排列和叶节点排列的多叉树结构的数据,

所述检索对象数据中的各内部节点包括用比特表示转移目的地是内部节点还是叶节点的比特向量、表示转移目的地的1个内部节点的存储位置的第1基础信息以及表示转移目的地的1个叶节点的存储位置的第2基础信息,

所述运算单元反复执行如下处理,一直到转移目的地成为叶节点为止,所述处理是:从密钥数据中取得规定比特长度的块,根据所访问的内部节点的所述比特向量中与该块的值对应的比特,判定从该内部节点起的转移目的地是内部节点还是叶节点,在判定出的转移目的地是内部节点的情况下,使用所述第1基础信息访问该转移目的地的内部节点,在判定出的转移目的地是叶节点的情况下,使用所述第2基础信息访问该转移目的地的叶节点。

2.根据权利要求1所述的检索装置,其特征在于,

对于所述检索对象数据中的各内部节点,成为转移目的地的内部节点在所述内部节点排列中按照存储位置连续的方式进行存储,成为转移目的地的叶节点在所述叶节点排列中按照存储位置连续的方式进行存储,

所述运算单元在根据所述比特向量的比特的值判定出的转移目的地是内部节点的情况下,使用所述第1基础信息和所述比特向量中表示内部节点的比特的数量访问该转移目的地的内部节点,

所述运算单元在转移目的地是叶节点的情况下,使用所述第2基础信息和所述比特向量中表示叶节点的比特的数量访问该转移目的地的叶节点。

3.根据权利要求1所述的检索装置,其特征在于,

对于所述检索对象数据中的各内部节点,成为转移目的地的叶节点在所述叶节点排列中按照存储位置连续的方式进行存储,具有相同值的叶节点被压缩,多个叶节点不包括具有相同值的多个叶节点,

所述检索对象数据中的各内部节点包括具有表示压缩前的叶节点的值变化的存储位置的比特的叶向量,

所述运算单元在根据所述比特向量的比特的值判定出的转移目的地是叶节点的情况下,使用所述第2基础信息和所述叶向量中表示所述存储位置的比特的数量访问该转移目的地的叶节点。

4.根据权利要求3所述的检索装置,其特征在于,

所述运算单元先调查所述比特向量和所述叶向量中的所述比特向量,根据该比特向量的比特的值使用所述叶向量。

5.根据权利要求1所述的检索装置,其特征在于,

对于所述检索对象数据中的各内部节点,成为转移目的地的叶节点在所述叶节点排列中按照存储位置连续的方式进行存储,具有相同值的叶节点被压缩,多个叶节点不包括具有相同值的多个叶节点,

所述检索对象数据中的各内部节点包括具有表示压缩前的叶节点的值变化的存储位置的比特的掩码向量,

所述运算单元在根据所述比特向量的比特的值判定出的转移目的地是叶节点的情况下，使用所述第2基础信息和用所述掩码向量掩蔽后的所述比特向量中的表示叶节点的比特的数量来访问该转移目的地的叶节点。

6.根据权利要求1所述的检索装置，其特征在于，

对于所述检索对象数据中的各内部节点，成为转移目的地的内部节点在所述内部节点排列中按照存储位置连续的方式进行存储，具有相同值的内部节点被压缩，多个内部节点不包括具有相同值的多个内部节点，

所述检索对象数据中的各内部节点包括具有表示压缩前的内部节点的值变化的存储位置的比特的掩码向量，

所述运算单元在根据所述比特向量的比特的值判定出的转移目的地是内部节点的情况下，使用所述第1基础信息和用所述掩码向量掩蔽后的所述比特向量中的表示内部节点的比特的数量来访问该转移目的地的内部节点。

7.根据权利要求1所述的检索装置，其特征在于，

对于所述检索对象数据中的各内部节点，成为转移目的地的叶节点中的规定值被掩蔽，该被掩蔽的值被变更为另一个值，然后将具有相同值的叶节点压缩，由此多个叶节点不包括具有相同值的多个叶节点，并在所述叶节点排列中使存储位置连续进行存储，

所述检索对象数据中的各内部节点包括所述被掩蔽的规定值、具有表示具有该规定值的叶向量的压缩前的位置的比特的叶掩码、以及由表示压缩前的叶节点的值变化的存储位置的比特构成的叶向量，

所述运算单元在根据所述比特向量的比特的值判定出的转移目的地是叶节点的情况下，判定是否在所述比特向量中与该比特的位置相同的位置对所述叶掩码设定了比特，在设定了比特的情况下，取得所述规定值作为该转移目的地的叶节点的值，在未设定比特的情况下，使用所述第2基础信息和所述叶向量中表示所述存储位置的比特的数量访问该转移目的地的叶节点。

8.根据权利要求2至7中任一项所述的检索装置，其特征在于，

所述运算单元使用由该运算单元构成的CPU的popcnt命令计算所述比特的数量。

9.根据权利要求1～7中任意一项所述的检索装置，其特征在于，

所述运算单元和所述存储单元是在64比特CPU中构成的。

10.根据权利要求1～7中任意一项所述的检索装置，其特征在于，

所述块是6比特长度，所述比特向量是64比特长度。

11.根据权利要求2至7中任一项所述的检索装置，其特征在于，

所述运算单元和所述存储单元是在64比特CPU中构成的，所述块是6比特长度，所述比特向量是64比特长度，

所述运算单元使用所述64比特CPU的popcnt命令计算所述比特的数量，使用来自基础信息的基于该比特的数量的偏置进行对所述转移目的地的节点的访问。

12.根据权利要求1～7中任意一项所述的检索装置，其特征在于，

所述运算单元从所述密钥数据中取得比所述规定比特长度长的比特长度的块，使用该块进行搜索，由此直接到达叶节点。

13.一种存储有程序的计算机可读取的存储介质，该程序使计算机作为权利要求1～7

中任意一项所述的所述检索装置中的各单元发挥作用。

14.一种由检索装置执行的检索方法，所述检索装置，具有：存储单元，其存储检索对象数据；以及运算单元，其根据密钥数据进行对所述检索对象数据的检索处理，其特征在于，

所述存储单元中存储的所述检索对象数据是具有内部节点排列和叶节点排列的多叉树结构的数据，

所述检索对象数据中的各内部节点包括用比特表示转移目的地是内部节点还是叶节点的比特向量、表示转移目的地的1个内部节点的存储位置的第1基础信息以及表示转移目的地的1个叶节点的存储位置的第2基础信息，

所述检索方法具有如下步骤：

反复执行如下处理，一直到转移目的地成为叶节点为止，所述处理是：从密钥数据中取得规定比特长度的块，根据所访问的内部节点的所述比特向量中与该块的值对应的比特，判定从该内部节点起的转移目的地是内部节点还是叶节点，在判定出的转移目的地是内部节点的情况下，使用所述第1基础信息访问该转移目的地的内部节点，在判定出的转移目的地是叶节点的情况下，使用所述第2基础信息访问该转移目的地的叶节点。

检索装置、检索方法及记录介质

技术领域

[0001] 本发明涉及通过搜索用树结构表现的检索对象数据来取得期望的数据的检索技术。

背景技术

[0002] 在路由器等装置中进行如下处理:根据所接收到的分组的目的地地址检索路径表而决定分组的转发目的地。在该处理中进行最长一致检索,为此以往使用Patricia前缀树(Trie)、基数树(Radix树)等。以往,二叉树的方式成为主流,性能即使高也就是数Mlps(Mega Lookup per second)。虽然也研究了多叉树(N-ary/Multiway)的方式,但是在实际应用中不是主流。这些树的性能未达到所期望的,因而实现数百 Mlps的TCAM这样的硬件成为事实上的标准。TCAM在经济性、集成度/规模性、功耗/发热方面存在难点。

[0003] 为了打破TCAM的问题,近年来出现了将市售品的器件和软件组合起来进行路径检索的技术。PacketShader、GPU Click、GAMT等利用GPU实现较高的路径检索性能,但是由于利用GPU,因而与TCAM一样存在热问题等。另外,作为现有技术文献有专利文献1。

[0004] 现有技术文献

[0005] 专利文献

[0006] 专利文献1:日本特开2000－083054号公报

发明内容

[0007] 发明要解决的问题

[0008] 如上所述,在利用TCAM或GPU等特定的器件时存在发热等问题,因而不期望利用特定的器件使路径检索高速化。

[0009] 也提出了不以特定的硬件的利用为前提,而在通用的硬件(例如市售的CPU等) 中利用软件使路径检索高速化的技术(例如DXR、SAIL),然而该技术在路径表内的路径数成为大规模时或地址长度变长时,存在性能下降的问题。

[0010] 在使用通用的硬件的检索处理中,在检索对象数据的数据规模成为大规模、密钥数据长度变长的情况下,不仅路径检索,而且也产生检索性能下降的问题。

[0011] 本发明正是鉴于上述情况而完成的,其目的在于,提供在使用通用的硬件的情况下,也能够高速地检索用树结构表现的检索对象数据的技术。

[0012] 用于解决问题的手段

[0013] 根据本发明的实施方式提供检索装置,具有:存储单元,其存储检索对象数据;以及运算单元,其根据密钥数据进行对所述检索对象数据的检索处理,其特征在于,

[0014] 在所述存储单元中存储的所述检索对象数据是具有内部节点排列和叶节点排列的多叉树结构的数据,

[0015] 所述检索对象数据中的各内部节点包括用比特表示转移目的地是内部节点还是叶节点的比特向量,

[0016] 所述运算单元反复执行如下处理，一直到转移目的地成为叶节点为止，所述处理是：从密钥数据取得规定比特长度的块，根据所访问的内部节点的所述比特向量中与该块的值对应的比特，判定从该内部节点起的转移目的地是内部节点还是叶节点并访问转移目的地的节点。

[0017] 另外，根据本发明的实施方式提供由检索装置执行的检索方法，该检索装置具有：存储单元，其存储检索对象数据；运算单元，其根据密钥数据进行对所述检索对象数据的检索处理，其特征在于，

[0018] 在所述存储单元中存储的所述检索对象数据是具有内部节点排列和叶节点排列的多叉树结构的数据，

[0019] 所述检索对象数据中的各内部节点包括用比特表示转移目的地是内部节点还是叶节点的比特向量，

[0020] 所述检索方法具有如下步骤：反复执行如下处理，一直到转移目的地成为叶节点为止，所述处理是：从密钥数据中取得规定比特长度的块，根据所访问的内部节点的所述比特向量中与该块的值对应的比特，判定从该内部节点起的转移目的地是内部节点还是叶节点并访问转移目的地的节点。

[0021] 发明效果

[0022] 根据本发明的实施方式，即使是在使用通用的硬件的情况下，也能够高速地检索用树结构表现的检索对象数据。

附图说明

[0023] 图1是用于说明多路径基数搜索方法的图。

[0024] 图2是本发明的实施方式的检索装置10的结构图。

[0025] 图3是示出本发明的实施方式的检索装置20的图。

[0026] 图4是示出存储部12中存储的检索对象数据的例子的图。

[0027] 图5是用于说明本实施方式的检索对象数据的结构及检索处理的概要的图。

[0028] 图6是示出内部节点和叶节点的更具体的例子的图。

[0029] 图7是用于说明检索处理的步骤的流程图。

[0030] 图8A是用于说明直接指向(direct pointing)的图。

[0031] 图8B是用于说明直接指向的图。

[0032] 图9是用于说明叶节点的数据的压缩例的图。

[0033] 图10是用于说明压缩例中的数据结构的例子的图。

[0034] 图11是用于说明叶节点的数据的压缩例的图。

[0035] 图12是用于说明压缩例中的数据结构的例子的图。

[0036] 图13是用于说明内部节点的数据的压缩例的图。

[0037] 图14是示出应用leaf mask时的内部节点的数据结构的图。

[0038] 图15是示出在使用leaf mask时取得叶的值的处理的流程图。

[0039] 图16是用于说明有关leaf mask的数据生成方法的图。

具体实施方式

[0040] 下面，参照附图说明本发明的实施方式。另外，下面说明的实施方式只不过是一个例子，本发明所应用的实施方式不限于以下的实施方式。

[0041] (关于搜索方法)

[0042] 在本实施方式中，作为本发明的检索技术的应用对象的例子假定如下的处理：在路由器中，将所接收到的分组的目的地地址作为密钥，按照最长一致来检索路由表(更具体地讲是转发表(forwarding table))，取得作为该分组的转发目的地的下一跳(next hop)的信息。然而，本发明的应用对象不限于此，本发明不限于最长一致，可以应用于完全一致等各种类型的检索。下面，将作为检索(搜索)的对象的数据称为检索对象数据。另外，将目的地地址等成为检索的密钥的数据称为密钥数据。

[0043] 在本实施方式中，使用利用多叉树表现的多路径基数搜索法作为对检索对象数据进行检索的方法，因而首先参照图1说明多路径基数搜索法的概要。

[0044] 在多路径基数搜索法中，从开头起按照每规定数量的多个比特(以下称为块(chunk))划分密钥数据，按照该多个比特进行树的转移。图1是将每2比特作为块的例子。各块可以取4种类型的值(在图1中表示为a、b、c、d)，因而将树中的各节点分支成4个方向。分支点是内部节点(在图1中用圆圈示出的节点)或者叶节点 (在图1中用四方形示出的节点)。

[0045] 在从密钥数据中的第一个块起第一段的节点中开始搜索，分支成相应的值的子节点，使密钥进入下一个块，由此依次进行搜索，在到达叶节点时搜索结束。

[0046] 在图1的例子中，例如在密钥数据是d××××(×表示任意的值)的情况下，到达由5示出的叶节点。另外，在密钥数据是bb××的情况下，到达用由6示出的叶节点。在叶节点存储有例如表示下一跳的信息(例如地址、IF信息等)，在到达叶节点的情况下，取得与密钥数据对应的下一跳的信息。

[0047] 上述的例子是将块长度设为2比特的例子，然而例如在使用64比特CPU结构的情况下，能够使用将块长度设为6比特、在各节点进行64分支的树的数据结构，以便将比特宽度设为相同宽度而有效地进行运算。

[0048] 在如上所述的多路径基数搜索法中，通常各节点具有对应分支数量的、指向子节点用的指针(存储子节点的地址等)，然而各指针例如用64比特指定子节点，因而存在整体的树的数据量非常庞大的问题。因此，在这样使用指针的结构中，树的数据不能完全存储在通用CPU的高速闪存中，不得不存储在位于CPU之外的存储器中，因而存在检索速度下降的问题。

[0049] 另一方面，在本实施方式的技术中，与上述的技术相比，能够大幅削减各内部节点的数据量，并且能够压缩具有相同数据的节点，因而能够减小整体的树的数据量，能够在通用CPU的高速闪存中存储树的数据而进行处理。因此，即使是使用通用CPU 等通用的硬件的情况下，也能够进行高速的检索处理。下面，更详细地说明本实施方式的技术。

[0050] (装置结构例)

[0051] 首先，说明执行本实施方式的检索处理的检索装置的结构例。图2是示出本实施方式的检索装置10的结构例的图。

[0052] 如图2所示，检索装置10具有运算部11、存储部12、输入部13、输出部14。运算部11

是利用后述的方法对使用了密钥数据的检索对象数据执行检索处理的功能部。存储部12是存储检索对象数据的功能部。输入部13是输入密钥数据的功能部。输出部14是输出检索结果的功能部。

[0053] 例如，检索装置10是通用计算机，由运算部11和存储部12构成CPU。在这种情况下，存储部12相当于CPU内的高速闪存。另外，存储部12的一部分也可以是 CPU以外的存储器。该CPU按照具有本实施方式的处理的逻辑的程序进行动作。该程序存储在存储部12中。另外，该程序也可以存储在存储部12以外的存储器等的存储部中。

[0054] 通过将该程序存储在可移动存储器等记录介质中，并从该可移动存储器加载到通用计算机中，能够使用该计算机作为检索装置10。

[0055] 另外，也可以将运算部11和存储部12构成为将本实施方式的处理的逻辑作为硬件电路装配而成的装置。

[0056] 图3是示出本实施方式的检索装置另一例的检索装置20的图。检索装置20例如是作为路由器发挥作用的装置。如图3所示，检索装置20具有分组处理部21、目的地决定部22及多个接口(IF)。分组处理部21经由IF接收分组，从与通过目的地决定部22决定的目的地(下一跳)对应的IF输出分组。在图3中示出了从IF-A接收分组并从IF-B输出的例子。

[0057] 目的地决定部22具有存储路由表(转发表)的存储部，从分组处理部21接收分组的目的地地址作为密钥数据，根据该密钥数据检索转发表，由此决定该分组的下一跳，将该下一跳的信息输出给分组处理部21。目的地决定部22例如能够使用图2所示的检索装置10。

[0058] 下面，详细说明由检索装置10执行的检索处理。下面，将进行基础处理的方式作为实施例1进行说明，将对实施例1追加了能够进行节点的压缩的功能的方式的例子作为实施例2～4进行说明。

[0059] (实施例1)

[0060] 图4是示出在检索装置10的存储部12中存储的检索对象数据的例子的图。图4 对于实施例1～4是相同的。如前面所述，在本实施方式中，进行以多路径基数搜索法为基础的检索处理，因而检索对象数据具有保存树中的各内部节点的数据的node array(节点排列)、和树中的各叶节点的数据即leaf array(叶排列)。通过指定各排列的Index，能够访问作为排列而存储的各节点的数据。

[0061] 由leaf array和node array构成的检索对象数据存储在例如可移动存储器等记录介质中，并从该可移动存储器加载到检索装置10中，由此能够使用检索装置10作为针对检索对象数据的检索装置10。另外，也能够从某一计算机经由网络将检索对象数据加载到检索装置10中。

[0062] 参照图5对实施例1的内部节点的数据结构进行说明。图5是在块的比特长度是 2、即从树的各节点向4个方向进行分支的例子，无论块的比特长度是几比特都是同样的结构。

[0063] 如图5所示，内部节点具有vector、base0、base1。vector是由来自该内部节点的分支数量的比特构成的比特向量。在块的比特长度是2比特的情况下，可以取00、01、10、11这4种值。vector的各比特从右端起顺序地对应于上述4种的各值。另外，“从右端起”是一个例子，也可以是“从左端起”。例如，在使用低位优先(little endian) 的CPU的情况下是从右端数起，在使用高位优先(big endian)的CPU的情况下是从左端数起。

[0064] 在图5的例子中，例如vector的右端(第0个)比特对应于块00，第1个比特对应于块

01,第2个比特对应于块10,第3个比特对应于块11。vector的各比特示出来自该内部节点的转移目的地(子节点)是内部节点还是叶节点。在本实施方式中，1表示内部节点,0表示叶节点,但这是示例,也可以构成为1表示叶节点、0表示内部节点。

[0065] 例如,在与图5所示的内部数据对应的块是00、01、10、11中的01的情况下,运算部11通过参照从vector的第0个数起的第1个比特(1),能够掌握下一个节点是内部节点。另外,例如在块是00、01、10、11中的00的情况下,运算部11通过参照vector的第0个比特(0),能够掌握下一个节点是叶节点。

[0066] 如上所述,运算部11能够根据vector掌握转移目的地的节点是内部节点还是叶节点,但在这种状态下,为了取得内部节点/叶节点的数据,不知道访问node array/leaf array中的哪个Index的要素为好。因此,在本实施方式中,内部节点保存base0、base1。

[0067] base1保存node array中与该内部节点的vector的比特1对应的子的内部节点的存储开始Index。base0保存leaf array中与该内部节点的vector的比特0对应的子的叶节点的存储开始Index。

[0068] 在本实施方式中,在node array中,对于各内部节点按照Index的顺序存储成为该内部节点的子的内部节点的数据。例如,对于某一内部节点,在子的内部节点有3 个的情况下,在node array中将该子的内部节点的3个数据存储为Index连续的3个数据。这3个数据中Index成为开头(最小)的数据的Index是base1。

[0069] 另外,在leaf array中,对于各内部节点按照Index的顺序存储成为该内部节点的子的叶节点的数据。例如,对于某一内部节点,在子的叶节点有3个的情况下,在leaf array中将该子的叶节点的3个数据存储为Index连续的3个数据。这3个数据中Index 成为开头(最小)的数据的Index是base0。另外,在本实施方式中使用的Index是指示存储部位的指标,可以将其改称为“地址”。

[0070] 按照以上所述在node array/leaf array中存储有数据,因而运算部11按照以下所述使用base0/base1访问子的内部节点/叶节点的数据。

[0071] 关于对vector所在的比特位置(从第0个数起第v个)的子的内部节点的访问,运算部11计算(计数)从vector的第0个到第v个的比特位置的1的个数。即,从 vector的右端开始计算(v+1)比特中的1的个数。在设该个数为bc(bit count)时,运算部11通过在node array中访问将bc与base1相加后的值减1而得的值 (bc+base1-1)的Index,能够取得该内部节点的数据。

[0072] 关于对vector所在的比特位置(从第0个数起第v个)的子的叶节点的访问,运算部11计算(计数)从vector的第0个到第v个的比特位置的0的个数。即,从vector 的右端开始计算(v+1)比特中的0的个数。在设该个数为bc(bit count)时,运算部11通过在leaf array中访问将bc与base0相加后的值减1而得的值(bc+base0-1) 的Index,能够取得该叶节点的数据。

[0073] 图5示出了利用上述的方法访问子的内部节点(Index:2498)及叶节点(Index:3127～3129)。

[0074] 通常CPU具备快速计算比特的个数的popcnt这种功能,在本实施方式中,能够有效运用该功能快速进行搜索。另外,使用popcnt是例子,也可以不使用popcnt。

[0075] 图5示出了块长度是2比特即vector是4比特的例子,但这是例子,块长度/vector

也可以是其它长度。图6示出块长度是6比特即vector是64(2^6)比特的例子。如已经说明的那样，在图6中示出内部节点具有vector、base0/base1，根据比特计数及 base0/base1，能够访问子的内部节点/叶节点。

[0076] 在本实施方式中，内部节点可以具有比特向量和2个base，与在每个分支具有指针的方式相比，能够大幅削减各节点的数据量，其结果是，能够削减检索对象数据的数据量。

[0077] 参照图7说明运算部11执行的检索处理的处理步骤。作为该处理的前提，向运算部11输入密钥数据，并且在存储部12存储具有上述构造的检索对象数据(node array/leaf array)。另外，图7示出了通过到达叶节点，检索处理结束的例子。

[0078] 运算部11从node array中的第一个内部节点取得vector(步骤101)，从密钥数据取得第一个块(步骤102)。

[0079] 运算部11读取与块相对应的vector的位置的比特，判定该比特是否是1(步骤103)。在该比特是1的情况下，如前面所述计算比特计数bc，访问在(bc+base1-1) 的Index中存储的内部节点，取得该内部节点的vector(步骤104)。

[0080] 运算部11从密钥数据取得下一个块(步骤105)，再次执行步骤103的判定。

[0081] 在步骤103的判定的结果是与块相对应的vector的位置的比特是0的情况下(步骤103:否)，进入步骤106。在步骤106，运算部11按照前面所述计算比特计数bc，访问在(bc+base0-1)的Index中存储的叶节点，取得该叶节点的值。

[0082] 另外，例如在转发表中存在成为叶的前缀长度集中于特定的范围(例如:/11～/24) 的倾向，因而能够省略最初的节点的搜索，直接到达目标入口(entry)。将此称为直接指向(direct pointing)。参照图8A、B说明例子。

[0083] 图8A是不进行直接指向的例子，在该例子中，在各6比特的块中进行搜索。图 8B示出了最先在12比特的块中进行搜索的直接指向的例子。在图8B的情况下，当在该块中不能到达叶节点的情况下，以后与实施例1的上述的方法一样，例如在各6 比特的块中进行搜索。直接指向也能够适用于其它实施例。

[0084] (实施例2)

[0085] 下面，作为实施例2，说明对于实施例1所说明的方式能够压缩叶数据的方式。例如，在将实施例1的方式应用于转发表的检索时，考虑到将会多发具有重复的值(下一跳)的叶节点。实施例2是以实施例1的方式为基础，压缩叶节点并进行保存。下面，主要说明与实施例1不同的部分。

[0086] 图9是实施例2的内部节点的图。如图9所示，在实施例2中，除在实施例1 中说明的vector、base0、base1以外，还追加了leafvec。leafvec是与vector的比特长度相同的比特长度。

[0087] 另外，关于leaf array中成为各内部节点的子的叶节点(即，各段的叶节点)，具有相同值的连续的叶节点仅保存开始连续的第一个叶节点。在图9的例子中，关于 Index是3127、3128、3129的叶节点，值全部相同是7，在这种情况下，Index仅保存3127的叶节点。这样压缩的结果是，在具有多个叶节点的情况下，也不包括具有相同值的多个叶节点，而成为彼此不同的值。

[0088] leafvec的要素是0或1的比特，从右端起，对与压缩前的叶节点开始连续的位置对应的比特设定1。例如，在图9的例子中是从第一个叶节点开始连续的，因而对与该第一个叶

节点对应的第一个(第0个)比特设定1。另外,在连续结束、另一个值的叶节点开始的情况下(叶节点变化的情况下),在该位置设定1。叶节点变化的情况包括第一个叶节点。此处的“连续”包括1个的情况。即,在叶节点的数据全部不同的情况下,在与叶节点对应的leafvec的比特位置全部设定1。leafvec的使用方法如下所述。

[0089] 运算部11在检测出与块对应的vector的比特(从第0个数起第v个比特)是0 时,可知子是叶节点。运算部11计算leafvec中从右端的第0个数起到第v个的比特 (v+1个比特)中的1的比特的个数。在设该个数为bc时,与vector的情况相同,运算部11访问(bc+base0-1)的Index的叶节点。

[0090] 图10示出实施例2中的内部节点和叶节点的数据例。在图10的例子中示出,运算部11根据块检测从(a)所示的内部节点中的vector的第0个数起的第1个比特是 1,访问与其对应的(c)的内部节点。另外,例如在(a)的内部节点中,在块对应于从第0个数起的第2个(0)的情况下,运算部11计算截止到leafvec中第2个的3 比特中的1的个数,使用base0访问与该个数对应的叶节点(L(0))。

[0091] 叶节点的压缩也可以利用如上所述使用leafvec以外的方法实现。下面,将有关叶节点的压缩的另一种方法作为实施例3进行说明。但是,实施例3的方法实质上与使用leafvec的方法相同。

[0092] (实施例3)

[0093] 图11是示出实施例3中的内部节点的图。如图11所示,在实施例3中,除在实施例1中说明的vector、base0、base1以外,还追加了mask。mask是与vector的比特长度相同的比特长度。

[0094] 另外,关于leaf array中成为各内部节点的子的叶节点(即,各段的叶节点),具有相同值的连续的叶节点仅保存开始连续的第一个叶节点。在图11的例子中,关于 Index是3127、3128、3129的叶节点,值全部相同是7,在这种情况下,Index仅保存3127的叶节点。这样压缩的结果是,在具有多个叶节点的情况下,也不包括具有相同值的连续的多个叶节点。

[0095] mask的要素是0或1的比特,从右端起,对与压缩前的叶节点开始连续的位置对应的比特设定0,在从该开始位置起相同值连续的叶节点的位置设定1(掩码)。另外,在连续结束、另一个值的叶节点开始的情况下(叶节点变化的情况下),在该位置设定0。叶节点变化的情况包括第一个叶节点。

[0096] 另外,与内部节点相对应的位置既可以设定1,也可以设定0,在该例子中是设为0。在图11的例子中,3个叶节点连续,因而在与第一个叶节点相对应的比特位置设定0,在与以后的叶节点相对应的比特位置设定掩码即1。mask的使用方法如下所述。

[0097] 运算部11在检测出与块对应的vector的比特(从第0个数起第v个比特)是0 时,可知子是叶节点。在实施例3中,运算部11进行vector与mask的OR运算,计算从进行OR运算后的vector中右端的第0个数起到第v个的比特(v+1个比特)中的0的比特的个数。在设该个数为bc时,运算部11访问(bc+base0-1)的Index的叶节点。

[0098] 图12示出实施例3中的内部节点和叶节点的数据例。在图12的例子中示出,运算部11根据块检测从(a)所示的内部节点中的vector的第0个数起的第1个比特是 1,访问与其对应的(c)的内部节点。另外,例如在(a)的内部节点中,在块对应于从第0个数起的第2个(0)的情况下,运算部11计算截止到mask后面的vector 中第2个的3比特中的0的个数,使用

base0访问对应该个数的叶节点(L(0))。

[0099] mask也能够适用于内部节点的压缩。参照图13说明将mask适用于内部节点的压缩的例子。图13与图6一样示出了块长度是6比特、即vector是64(2^6)比特时的例子。在该例子中,除在实施例1中说明的vector、base0、base1以外,还追加了 mask。mask是与vector的比特长度相同的比特长度。

[0100] 另外,关于各段的内部节点,具有相同值的连续的内部节点仅保存开始连续的第一个内部节点。在图13的例子中,如(a)所示,具有相同的子树的内部节点有3个。在这种情况下,如(b)所示,仅保存3个中的第一个内部节点。这样压缩的结果是,在具有多个内部节点的情况下,也不包括具有相同值的连续的多个内部节点。

[0101] mask的要素是0或1的比特,从右端起,对与压缩前的内部节点开始连续的位置对应的比特设定1,在从该开始位置起相同值连续的内部节点的位置设定0(掩码)。另外,在连续结束、其它的值的内部节点开始的情况下(内部节点变化的情况下),在该位置设定1。

[0102] 在图13的例子中,3个内部节点连续,因而在与第一个内部节点相对应的比特位置设定1,在与以后的内部节点相对应的比特位置设定掩码即0。即,如图13(b) 所示,与vector的第一个的1对应的mask的比特是1,与下一个的1以及再下一个的1对应的mask的比特是0。mask的使用方法如下所述。

[0103] 运算部11在检测出与块对应的vector的比特(从第0个数起第v个比特)是1 时,可知子是内部节点。运算部11进行vector与mask的AND运算,计算从进行 AND运算后的vector中右端的第0个数起到第v个的比特(v+1个比特)中的1的比特的个数。在设该个数为bc时,运算部11访问(bc+base1-1)的Index的内部节点。

[0104] (实施例4)

[0105] 下面说明实施例4。实施例4是比实施例2、3进一步压缩叶节点的方式。图14 示出实施例4的内部数据的构造。如图14所示,实施例4的内部数据除已经说明的 vector、leafvec、base0、base1以外,如“A”所示还追加了leaf mask和masked leaf。在存储部12中存储有node array和leaf array。

[0106] leaf mask是由与vector相同比特长度的0/1的比特构成的数据。masked leaf是某一叶节点的数据。下面,说明在使用leaf mask和masked leaf时的运算部11的动作。

[0107] 参照图15的流程图,说明实施例4的检索装置10的运算部11的动作例。图15 是用于特别说明与实施例1、2不同的处理的部分的图。

[0108] 在步骤201,运算部11检测当前的块的vector中的相应比特(从第0个数起第v 个比特)是0,由此检测出转移到叶节点。

[0109] 在步骤202,运算部11判定在leaf mask中从第0个数起第v个比特是否是1。在其是1的情况下(步骤202:是),取得masked leaf的值作为叶数据的值(步骤203)。

[0110] 在步骤202,在第v个比特不是1的情况下(步骤202:否),运算部11与实施例2一样计算leafvec的从第0个到第v个的1的个数(bc),访问(bc+base0-1)的 Index的叶节点并取得值(步骤204)。

[0111] 下面,参照图16说明实施例4的有关leaf mask的数据的生成方法。以下说明的数据的生成既可以由检索装置10进行,也可以由其它装置(计算机)进行。下面将进行数据的生成的装置称为生成装置。生成装置是检索装置10或者其它装置。在生成装置是其它装置

的情况下，在数据生成后，将生成数据存储在检索装置10的存储部12中。

[0112] 首先，生成装置计算没有压缩的leaf array。由此，例如如果是四叉树，对于父的每个内部节点，生成例如在图5的L所示的Index连续的叶节点的数据。

[0113] 另外，如果是六十四叉树，按照父的每个内部节点，leaf array的项目数最大达到64。另外，例如在十六叉树的例子中，在设叶信息是A、B、C这三种时，叶信息如图16的(a)所示，例如以ABAA BBBA BCBB CCCC的方式在leaf array内排列。

[0114] 然后，生成装置选择被掩蔽的叶信息。在该例中，将B掩蔽并省略。通常，将连续的断片出现的次数最多的进行掩蔽比较有效，因而生成装置决定将连续的断片出现的次数最多的B进行掩蔽。另外，“连续的断片”包括如ABA中的B那样是1个的情况。将被掩蔽的叶的信息B存储在masked leaf中。

[0115] 然后，生成装置将被掩蔽的叶信息出现的时隙保存在leaf mask中。被掩蔽的叶信息出现的时隙相当于与vector中的该叶对应的比特位置。例如，在vector是0010 的情况下，在将与设左端为第1个开始数起第2个的比特0对应的叶掩蔽的情况下，在leaf mask中保存0100。

[0116] 另外，生成装置将在leaf array中被掩蔽的叶信息的时隙设为与紧前面的值相同的值。由此，根据图16的(a)所示的叶信息，如图16的(b)所示得到“leaf mask:01001110 1011 0000”，并得到“leaf array:AAAA AAAA ACCC CCCC”。另外，该例是高位优先(big endian)，因而从左端数起。在图16的(b)中，下划线部分是被掩蔽的部分，成为与计数方向上的紧前面的值(左侧的值)相同的值。

[0117] 然后，与没有叶掩蔽的情况相同，将连续的部分压缩。由此，如图16的(c)所示成为“leafvec:1000 0000 0100 0000”，成为“leaf array:AC”。

[0118] 上述处理的结果是，如图16的(d)所示，能够得到“leaf mask:0100 1110 1011 0000”、“masked leaf:B”、“leaf vector:1000 0000 0100 0000”、“leaf array:AC”。

[0119] 另外，作为参考，没有叶掩蔽而进行压缩时的leaf array成为“ABABABCBC”，可知根据实施例4得到较高的压缩效果。

[0120] 在实施例4中，追加了1个掩码(例如64bit)和1个叶，但能够省略不连续的几个叶，实现leaf array的进一步压缩。这对于下面的情况特别有效：leaf array被某个下一跳值多次断开(成为条纹状态)的情况、或1个叶的尺寸(leaf array的1入口的尺寸)较大为16字节等情况。

[0121] 另外，实施例2、3、4示出了将叶节点压缩的例子，但对于具有相同数据的内部节点，也能够进行与叶节点的情况一样的压缩。并且，也可以进行叶节点的压缩和内部节点的压缩双方。

[0122] (实施方式的效果)

[0123] 如以上说明的那样，在本实施方式中能够大幅削减树的数据量，因而能够在例如通用CPU的高速闪存(例如L1、L2、L3高速闪存)中存储检索对象数据而实施检索处理，实现快速的检索处理。特别是在例如检索对象数据是路径表的情况下，能够解决在路径表内的路径数成为大规模时性能下降的问题。另外，由于处理高速化，因而也能够解决在地址长度变长时性能下降的问题。

[0124] 另外，在树的各等级中，用比特单位表现部分树的有无，因而存储器效率良好。特

别是在例如使用六十四叉树的情况下，表现每64比特部分树(子排列)的有无，因而具有在64-bit CPU中的处理效率良好的特点。

[0125] 另外，在vector等中，对是1的比特进行计数，对排列中相应的子跳过1步骤，因而能够实现快速处理，存储器效率也良好。并且，由于计数是1的比特，因而能够使用popcnt CPU Instruction，实现快速处理。另外，由于以通用的多叉树(Multiway trie)为基础，因而扩展性/灵活性提高，不限于路径表检索，能够适用于各种检索。

[0126] 另外，通过进行在实施例2～4中说明的叶信息的压缩，能够减小检索对象数据的量，实现更进一步的快速化。

[0127] 作为一例，通过使用本实施方式的技术，能够利用具有popcnt CPU Instruction的通用CPU，对保存50万的IPv4全根(full root)路径的路径表，实现单核约为 240Mlps、4核约为910Mlps。不利用TCAM，在通用CPU中即可发挥与TCAM相同乃至数倍的性能。

[0128] (实施方式的汇总)

[0129] 如以上说明的那样，根据本实施方式提供检索装置，具有：存储单元，其存储检索对象数据；以及运算单元，其根据密钥数据进行对所述检索对象数据的检索处理，其特征在于，在所述存储单元中存储的所述检索对象数据是具有内部节点排列和叶节点排列的多叉树结构的数据，所述检索对象数据中的各内部节点包括用比特表示转移目的地是内部节点还是叶节点的比特向量，所述运算单元反复执行如下处理，一直到转移目的地成为叶节点为止，所述处理是：从密钥数据取得规定比特长度的块，根据所访问的内部节点的所述比特向量中与该块的值对应的比特，判定从该内部节点起的转移目的地是内部节点还是叶节点并访问转移目的地的节点。

[0130] 也可以构成为，所述检索对象数据中的各内部节点包括表示转移目的地的1个内部节点的存储位置的第1基础信息、和表示转移目的地的1个叶节点的存储位置的第 2基础信息，所述运算单元在根据所述比特向量的比特的值判定出的转移目的地是内部节点的情况下，使用所述第1基础信息访问该转移目的地的内部节点，在转移目的地是叶节点的情况下，使用所述第2基础信息访问该转移目的地的叶节点。

[0131] 也可以是，对于所述检索对象数据中的各内部节点，成为转移目的地的内部节点在所述内部节点排列中按照存储位置连续的方式进行存储，成为转移目的地的叶节点在所述叶节点排列中按照存储位置连续的方式进行存储，所述运算单元在根据所述比特向量的比特的值判定出的转移目的地是内部节点的情况下，使用所述第1基础信息和所述比特向量中表示内部节点的比特的数量来访问该转移目的地的内部节点，在转移目的地是叶节点的情况下，使用所述第2基础信息和所述比特向量中表示叶节点的比特的数量来访问该转移目的地的叶节点。

[0132] 也可以是，对于所述检索对象数据中的各内部节点，成为转移目的地的叶节点在所述叶节点排列中按照存储位置连续的方式进行存储，具有相同值的叶节点被压缩，多个叶节点不包括具有相同值的多个叶节点，所述检索对象数据中的各内部节点包括具有表示压缩前的叶节点的值变化的存储位置的比特的叶向量，所述运算单元在根据所述比特向量的比特的值判定出的转移目的地是叶节点的情况下，使用所述第2基础信息和所述叶向量中表示所述存储位置的比特的数量来访问该转移目的地的叶节点。

[0133] 也可以是，所述运算单元先调查所述比特向量和所述叶向量中的所述比特向量，

根据该比特向量的比特的值使用所述叶向量。

[0134] 也可以是，对于所述检索对象数据中的各内部节点，成为转移目的地的叶节点在所述叶节点排列中按照存储位置连续的方式进行存储，具有相同值的叶节点被压缩，多个叶节点不包括具有相同值的多个叶节点，所述检索对象数据中的各内部节点包括具有表示压缩前的叶节点的值变化的存储位置的比特的掩码向量，所述运算单元在根据所述比特向量的比特的值判定出的转移目的地是叶节点的情况下，使用所述第2 基础信息和用所述掩码向量掩蔽后的所述比特向量中的表示叶节点的比特的数量访问该转移目的地的叶节点。

[0135] 也可以是，对于所述检索对象数据中的各内部节点，成为转移目的地的内部节点在所述内部节点排列中按照存储位置连续的方式进行存储，具有相同值的内部节点被压缩，多个内部节点不包括具有相同值的多个内部节点，所述检索对象数据中的各内部节点包括具有表示压缩前的内部节点的值变化的存储位置的比特的掩码向量，所述运算单元在根据所述比特向量的比特的值判定出的转移目的地是内部节点的情况下，使用所述第1基础信息和用所述掩码向量掩蔽后的所述比特向量中的表示内部节点的比特的数量访问该转移目的地的内部节点。

[0136] 也可以是，对于所述检索对象数据中的各内部节点，成为转移目的地的叶节点中的规定值被掩蔽，该被掩蔽的值被变更为另一个值，然后将具有相同值的叶节点压缩，由此多个叶节点不包括具有相同值的多个叶节点，并在所述叶节点排列中使存储位置连续进行存储，所述检索对象数据中的各内部节点包括所述被掩蔽的规定值、具有表示具有该规定值的叶向量的压缩前的位置的比特的叶掩码、以及由表示压缩前的叶节点的值变化的存储位置的比特构成的叶向量，所述运算单元在根据所述比特向量的比特的值判定出的转移目的地是叶节点的情况下，判定是否在所述比特向量中与该比特的位置相同的位置对所述叶掩码设定了比特，在设定了比特的情况下，取得所述规定值作为该转移目的地的叶节点的值，在未设定比特的情况下，使用所述第2基础信息和所述叶向量中表示所述存储位置的比特的数量来访问该转移目的地的叶节点。

[0137] 也可以是，所述运算单元使用由该运算单元构成的CPU的popcnt命令计算所述比特的个数。

[0138] 另外，也可以是，所述运算单元和所述存储单元是在64比特CPU中构成的。并且，也可以是，所述块是6比特长度，所述比特向量是64比特长度。

[0139] 另外，也可以是，所述运算单元和所述存储单元是在64比特CPU中构成的，所述块是6比特长度，所述比特向量是64比特长度，所述运算单元使用所述64比特 CPU的popcnt命令计算所述比特的个数，使用来自基础信息的基于该比特的数量的偏置进行对所述转移目的地的节点的访问。

[0140] 另外，也可以是，所述运算单元从所述密钥数据中取得比所述规定比特长度长的比特长度的块，使用该块进行搜索，由此直接到达叶节点。

[0141] 另外，根据本实施方式也能够提供使计算机作为所述检索装置中的各单元发挥作用的程序。并且，根据本实施方式也能够提供存储了所述检索对象数据的计算机可读的记录介质。

[0142] 另外，上述的“存储单元”也可以置换为存储部、存储电路、存储器件中任意一方。并且，上述的“运算单元”也可以置换为运算部、运算电路、运算器件中任意一方。

[0143] 另外，本实施方式的检索方法也可以构成为根据密钥数据进行对检索对象数据的检索处理的检索方法，所述检索对象数据是具有内部节点排列和叶节点排列的多叉树结构的数据，所述检索对象数据中的各内部节点包括用比特表示转移目的地是内部节点还是叶节点的比特向量，所述检索方法具有如下步骤：反复执行如下处理，一直到转移目的地成为叶节点为止，所述处理是：从密钥数据取得规定比特长度的块，根据所访问的内部节点的所述比特向量中与该块的值对应的比特，判定从该内部节点起的转移目的地是内部节点还是叶节点并访问转移目的地的节点。

[0144] 本发明不限于上述的实施方式，能够在权利要求书的范围内进行各种变更及应用。

[0145] 本专利申请以在2015年3月11日提出的日本专利申请第2015－048657号为基础并对其主张优先权，并且将第2015－048657号日本专利申请的全部内容引用于此。

[0146] 标号说明

[0147] 10、20检索装置；11运算部；12存储部；13输入部；14输出部；21分组处理部；22目的地决定部。

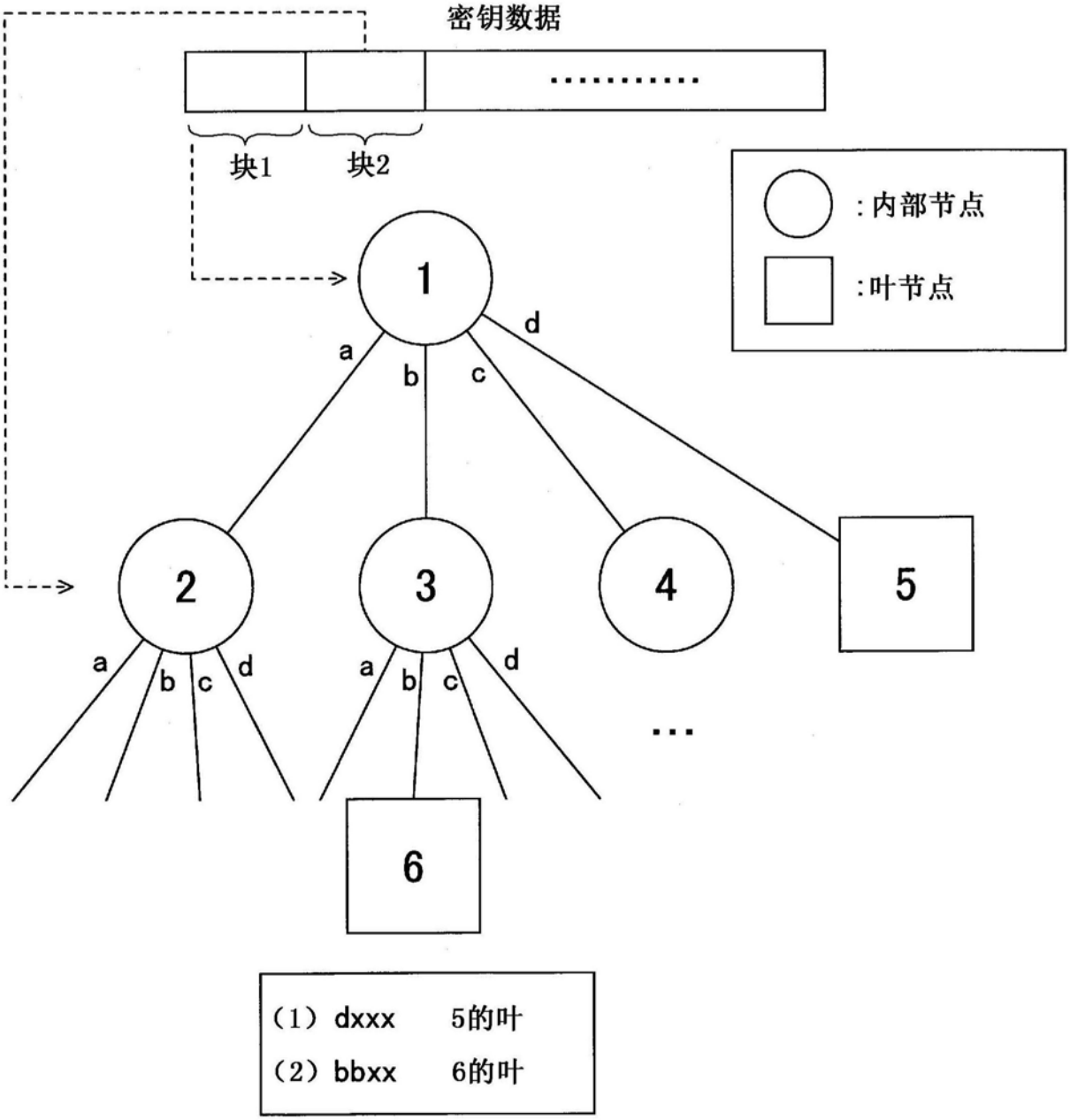


图1

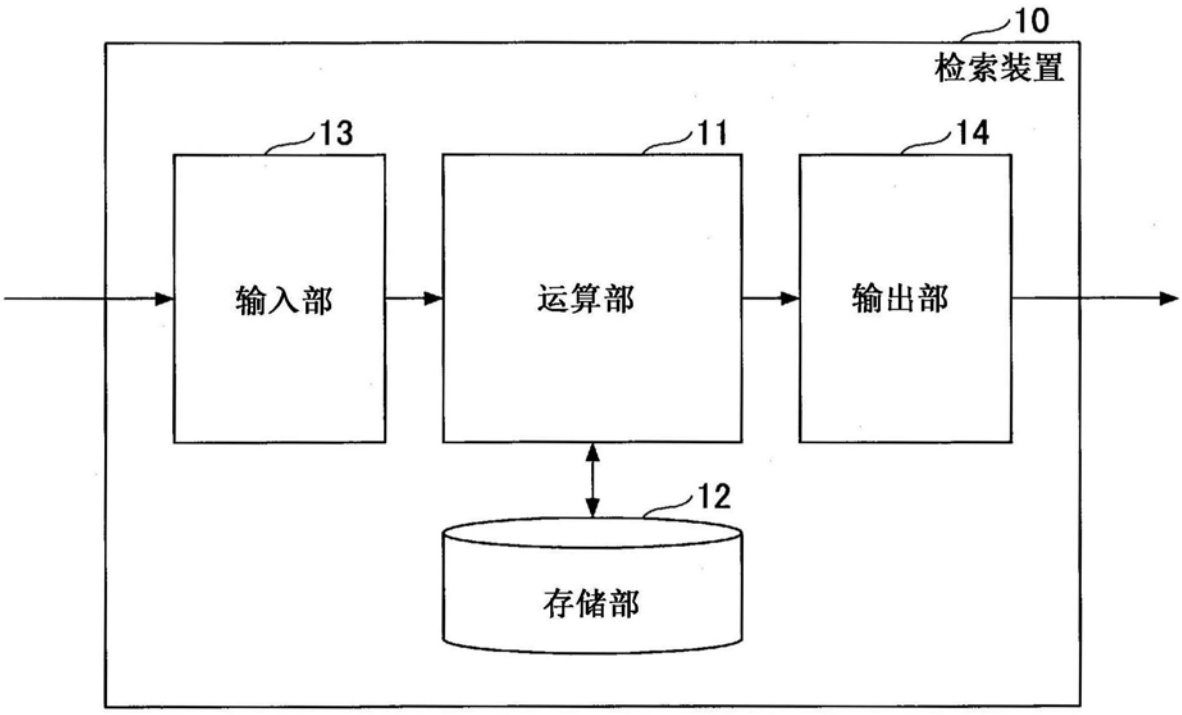

10
检索装置
13
输入部
11
运算部
14
输出部
12
存储部

图2

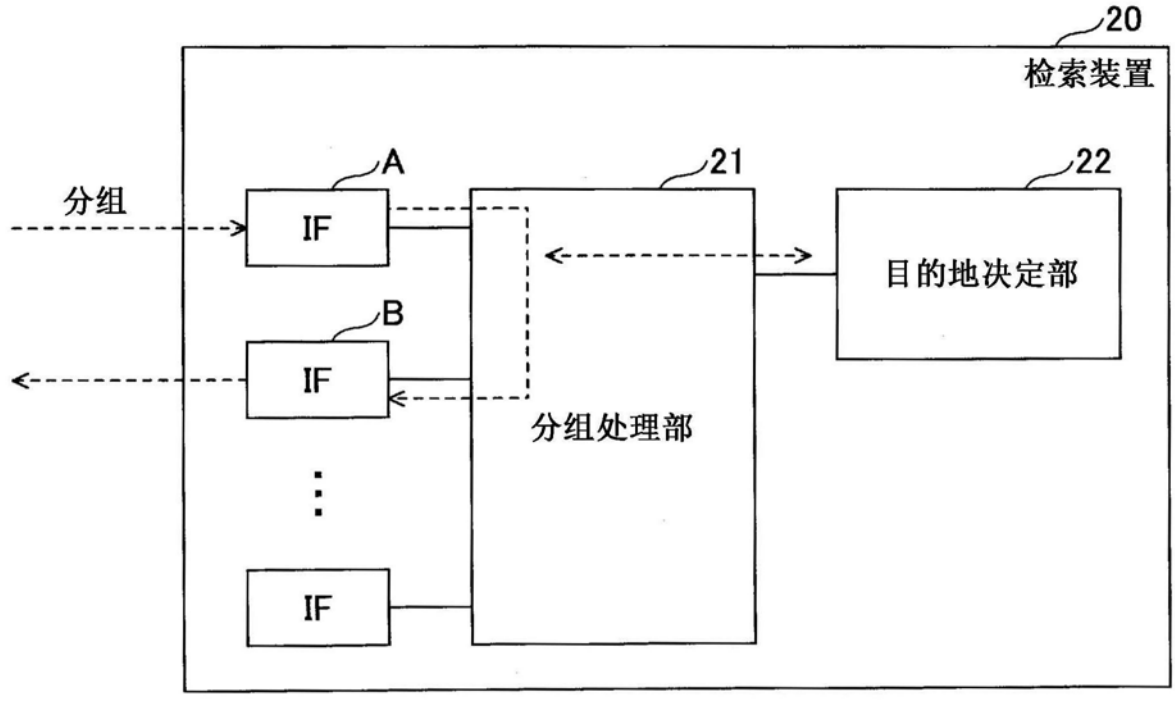

20
检索装置
分组
A
IF
B
IF
IF
21
分组处理部
22
目的地决定部

图3

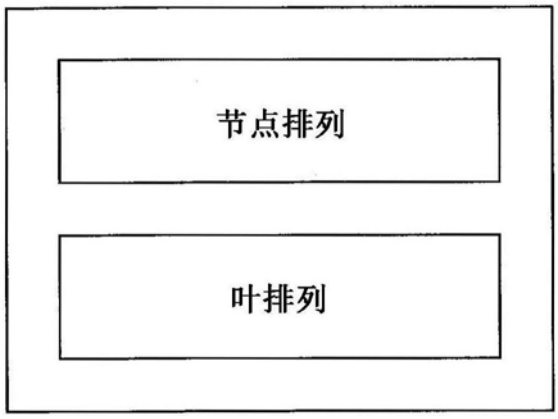

节点排列
叶排列

图4

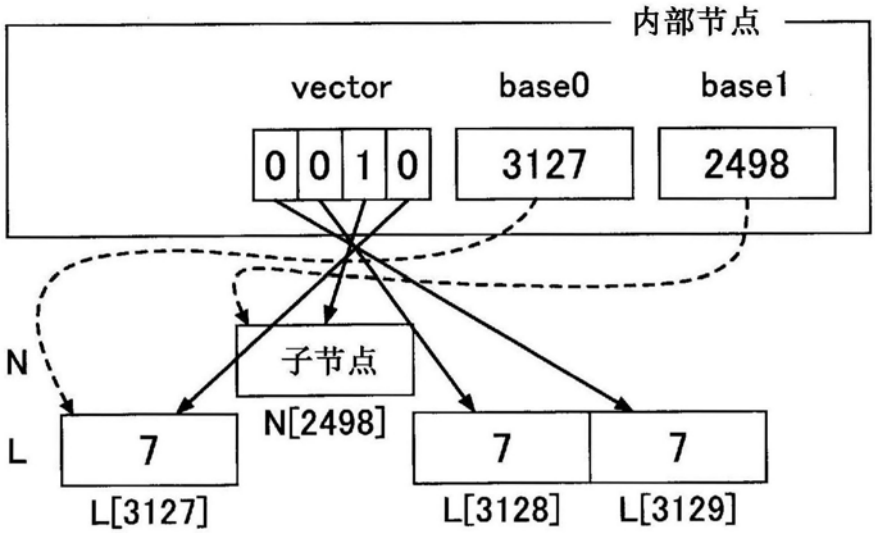

内部节点
vector
base0
base1
0 0 1 0
3127
2498
N
子节点
N[2498]
L
7
7
7
L[3127]
L[3128]
L[3129]

图5

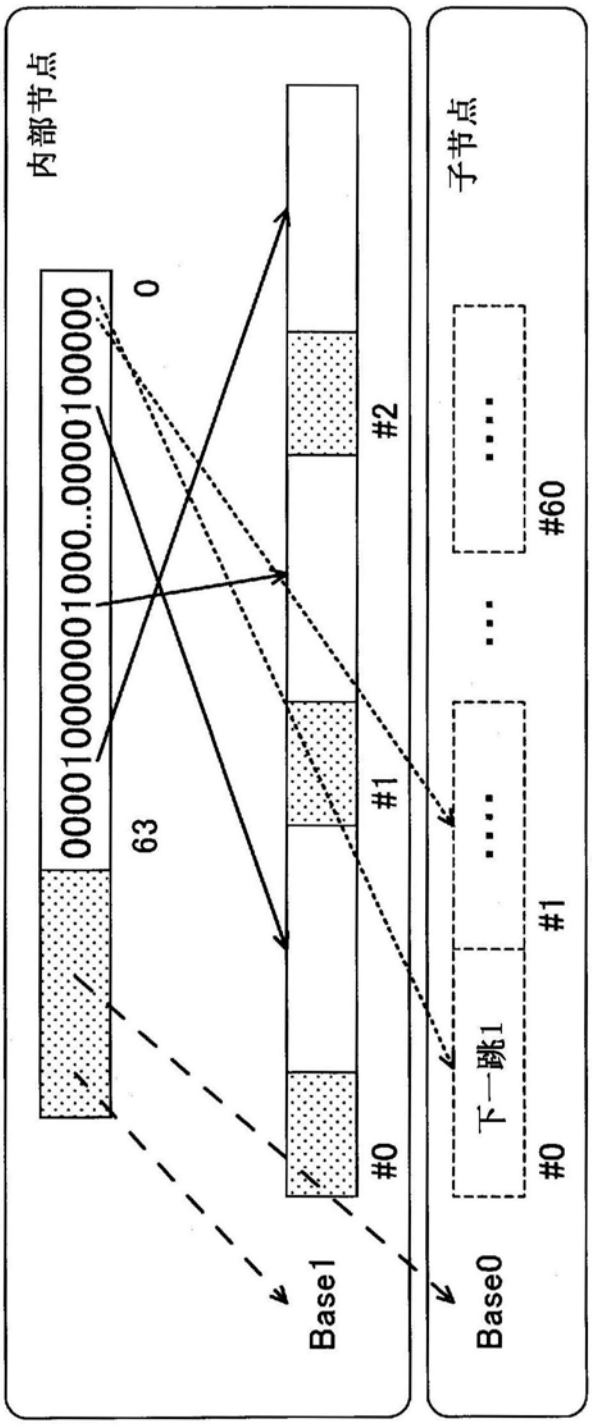

内部节点
子节点
0000100000001000...0000100000
0
63
#2
#1
#0
#60
#1
#0
下一跳1
Base1
Base0

图6

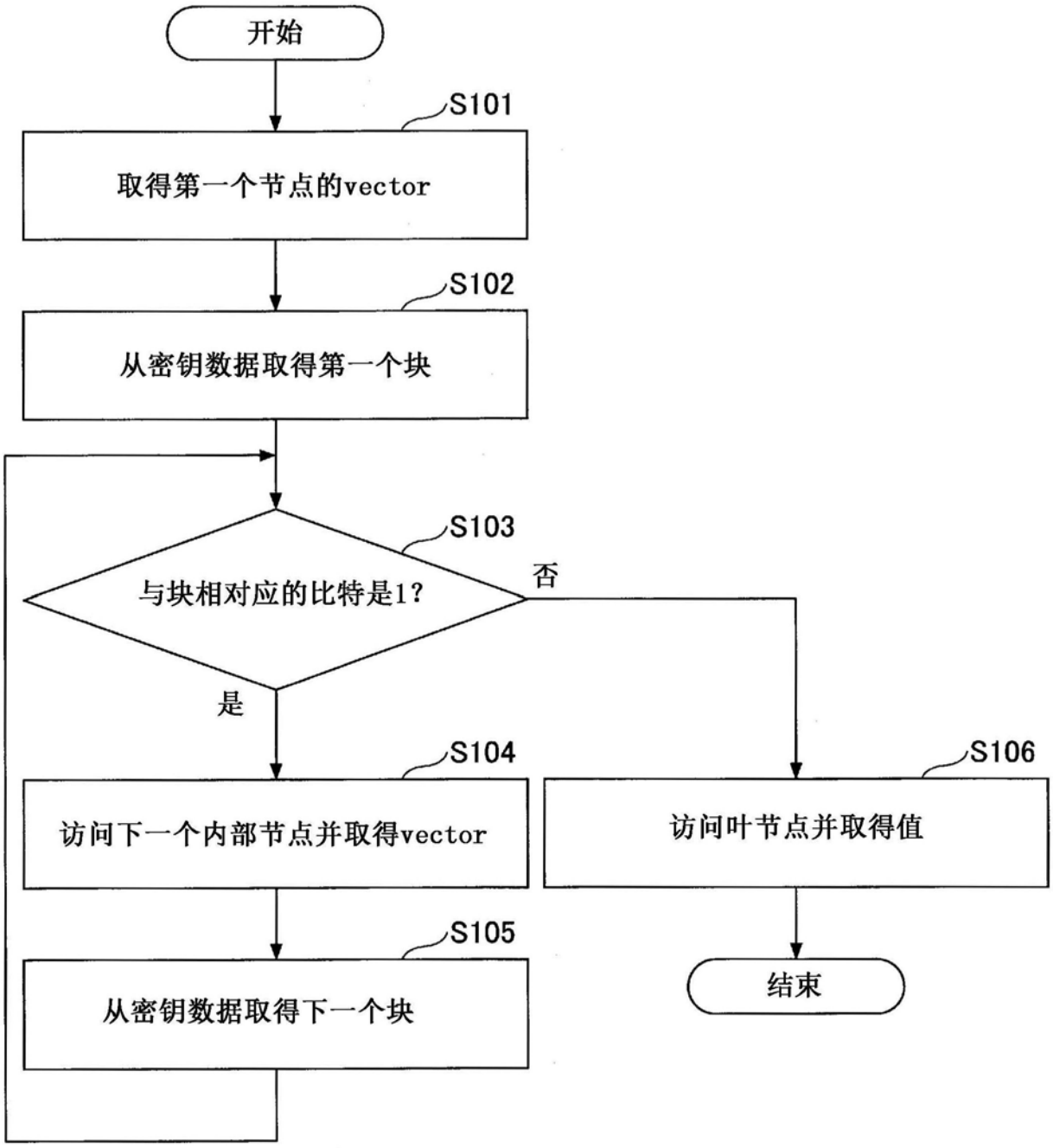

开始
S101
取得第一个节点的vector
S102
从密钥数据取得第一个块
S103
与块相对应的比特是1?
否
是
S104
访问下一个内部节点并取得vector
S105
从密钥数据取得下一个块
S106
访问叶节点并取得值
结束

图7

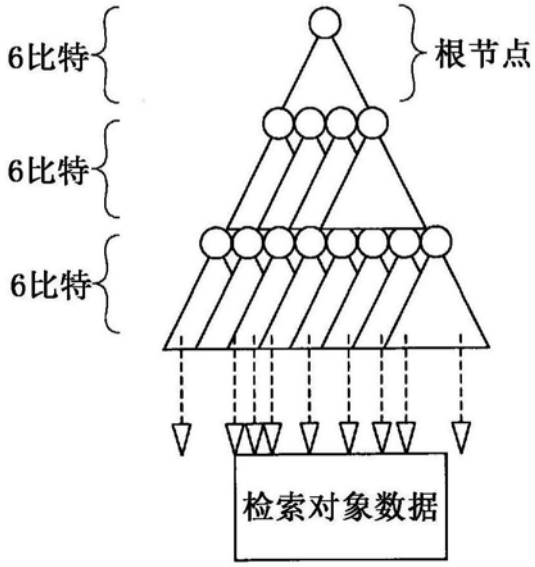


图8A

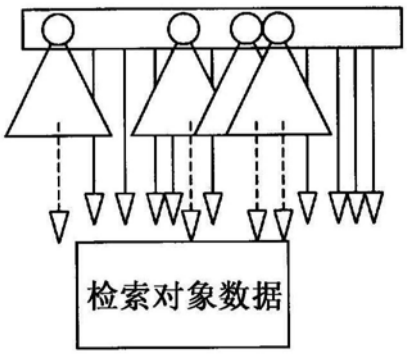


图8B

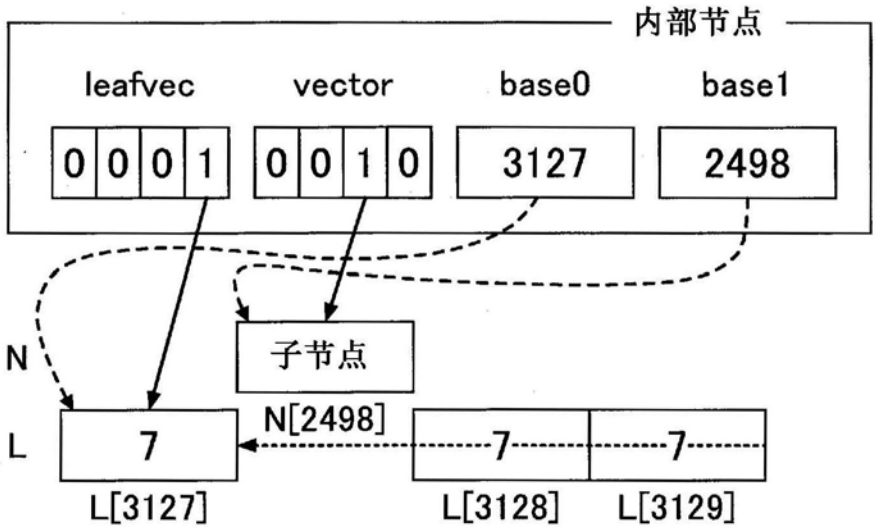

内部节点
leafvec
vector
base0
base1
0 0 0 1
0 0 1 0
3127
2498
子节点
N
L
7
N[2498]
7
7
L[3127]
L[3128]
L[3129]

图9

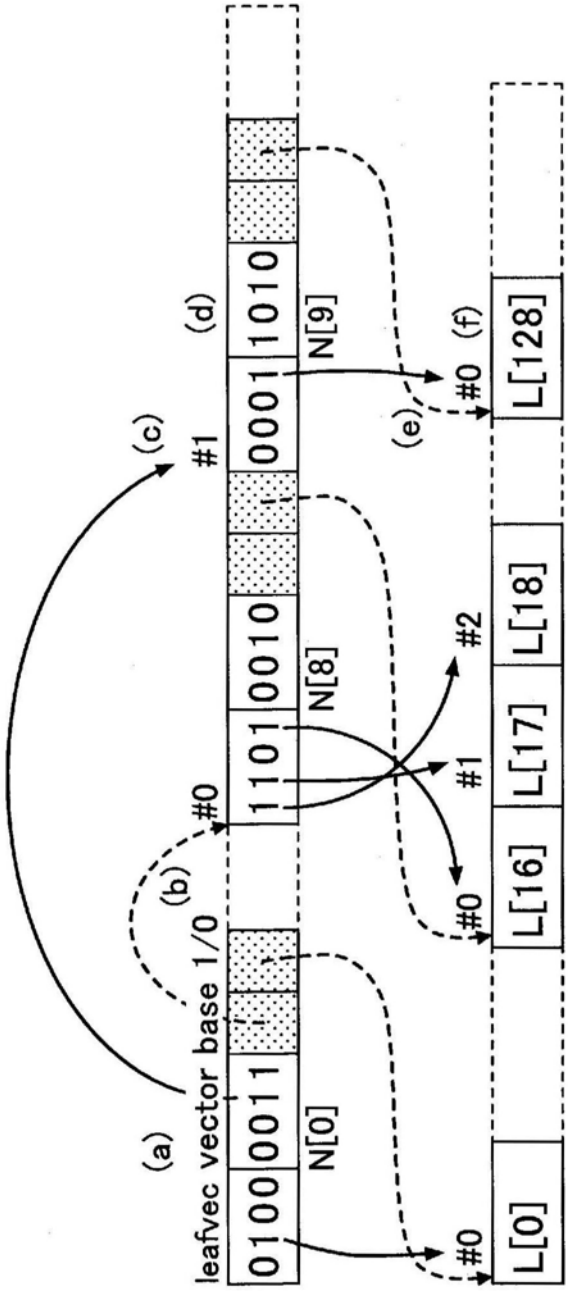

leafvec
vector
base
1/0
0100
0011
1101
0001
1010
0010
N[0]
N[8]
N[9]
#0
#1
#2
#0
#1
#0
#0
(a)
(b)
(c)
(d)
(e)
(f)
L[0]
L[16]
L[17]
L[18]
L[128]

图10

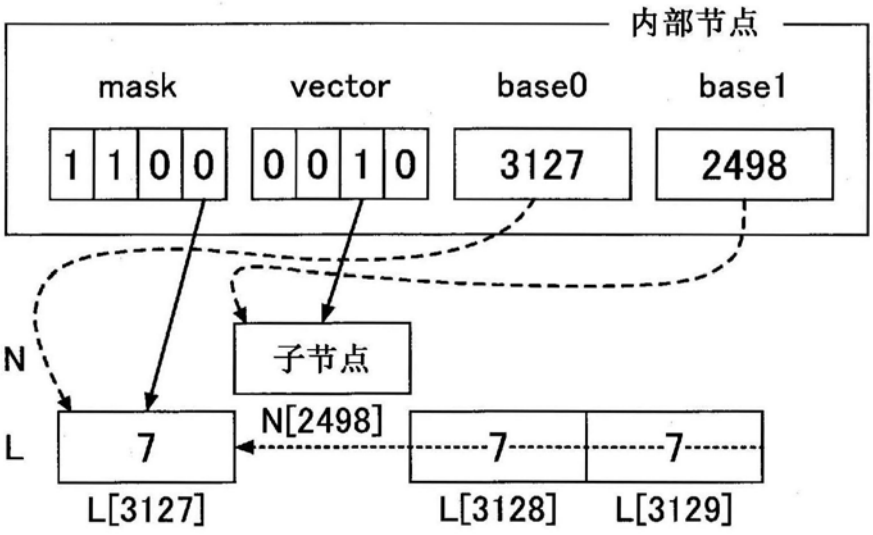

内部节点
mask
vector
base0
base1
1 1 0 0
0 0 1 0
3127
2498
N
子节点
L
7
N[2498]
7
7
L[3127]
L[3128]
L[3129]

图11

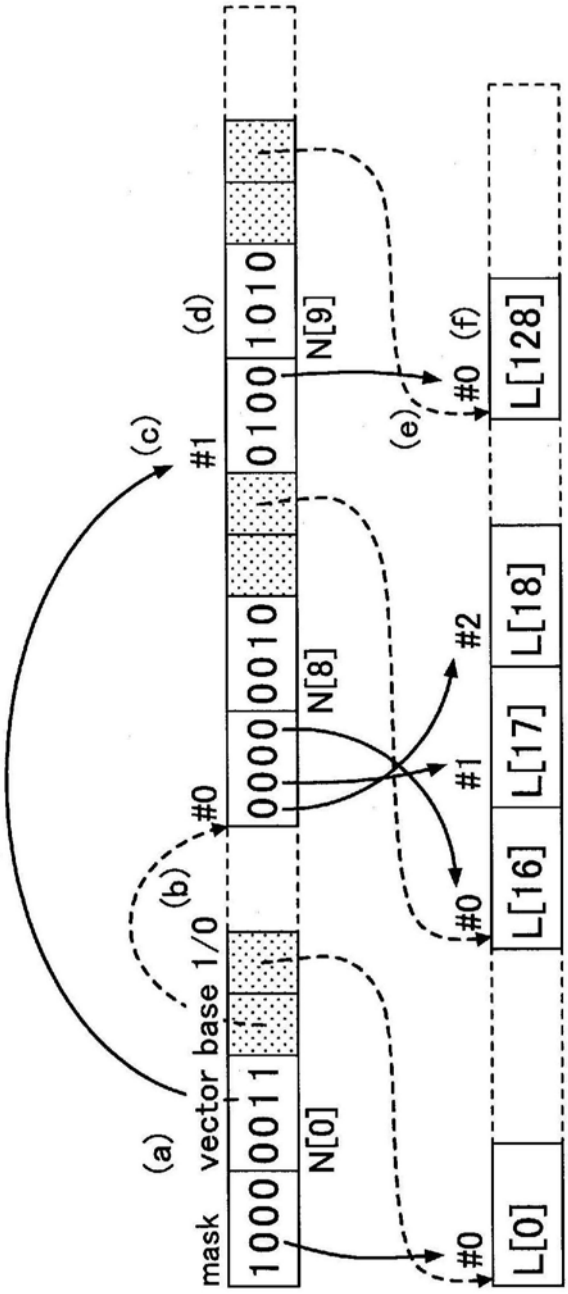

mask
vector base 1/0
1000
0011
0000
0010
0100
1010
N[0]
N[8]
N[9]
(a)
(b)
(c)
(d)
(e)
(f)
#0
#1
#2
L[0]
L[16]
L[17]
L[18]
L[128]

图12

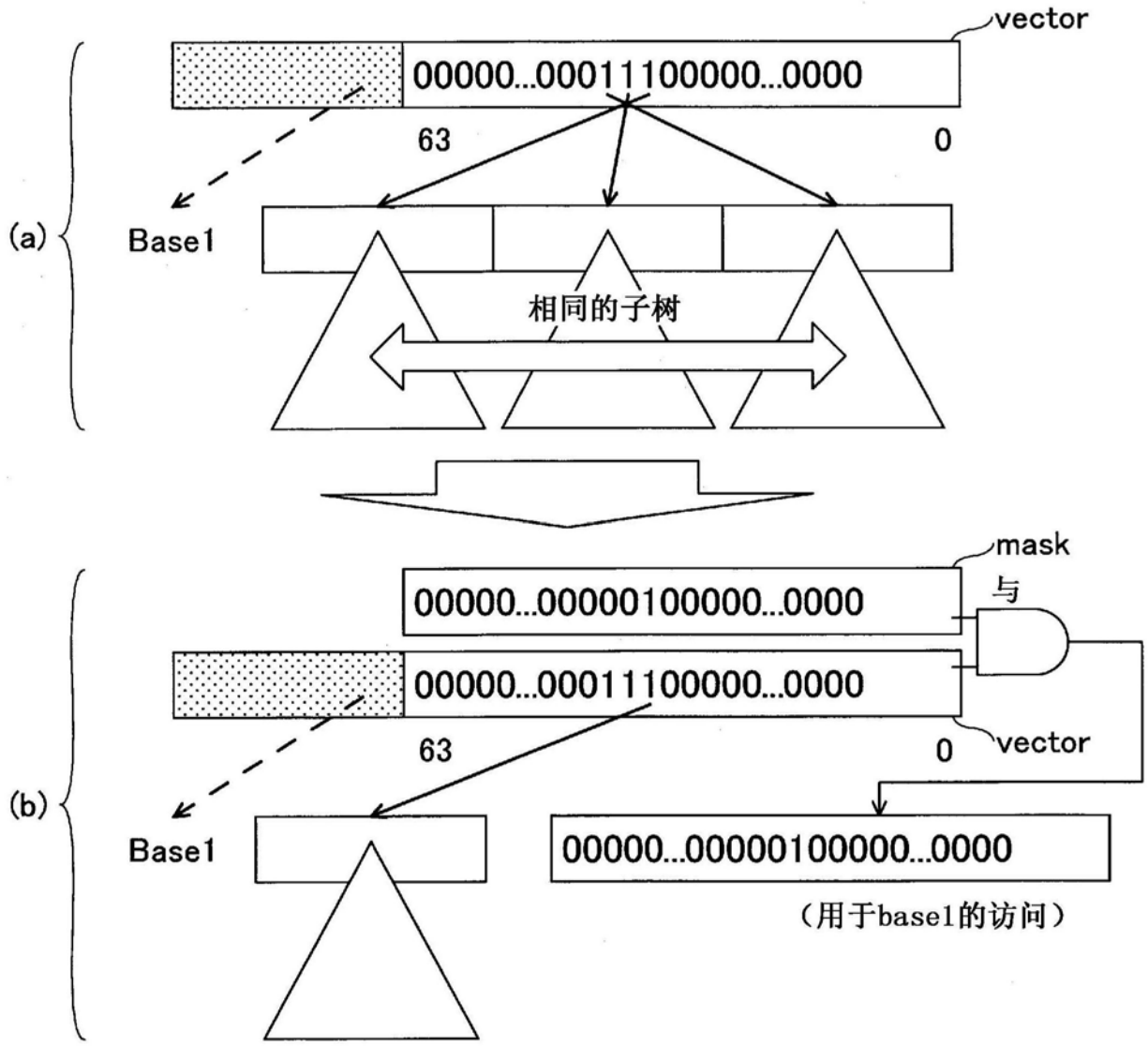

vector
00000...00011100000...0000
63
0
(a)
Base1
相同的子树
mask
与
00000...00000100000...0000
00000...00011100000...0000
63
0
vector
(b)
Base1
00000...00000100000...0000
（用于base1的访问）

图13

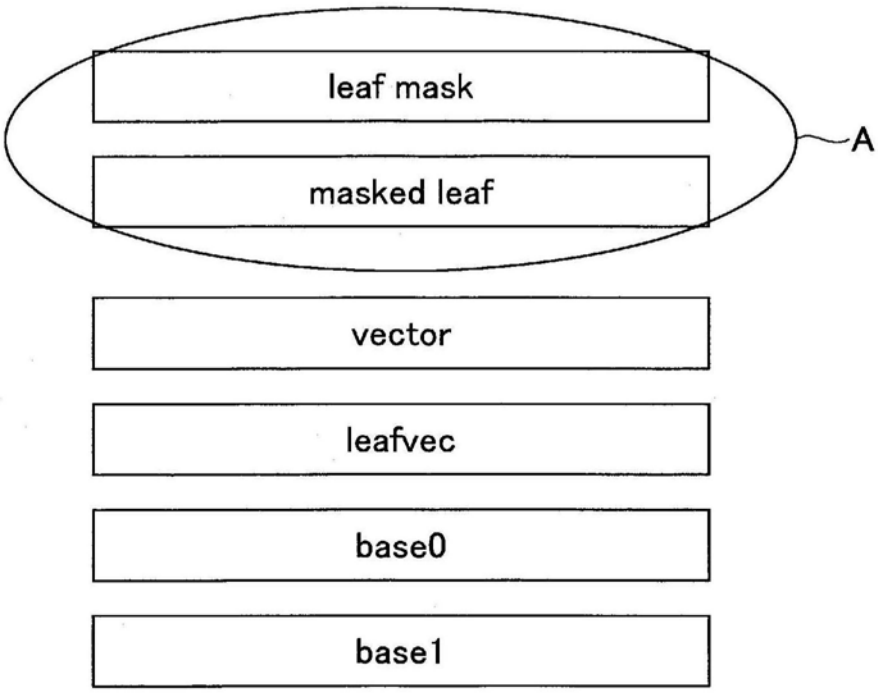


图14

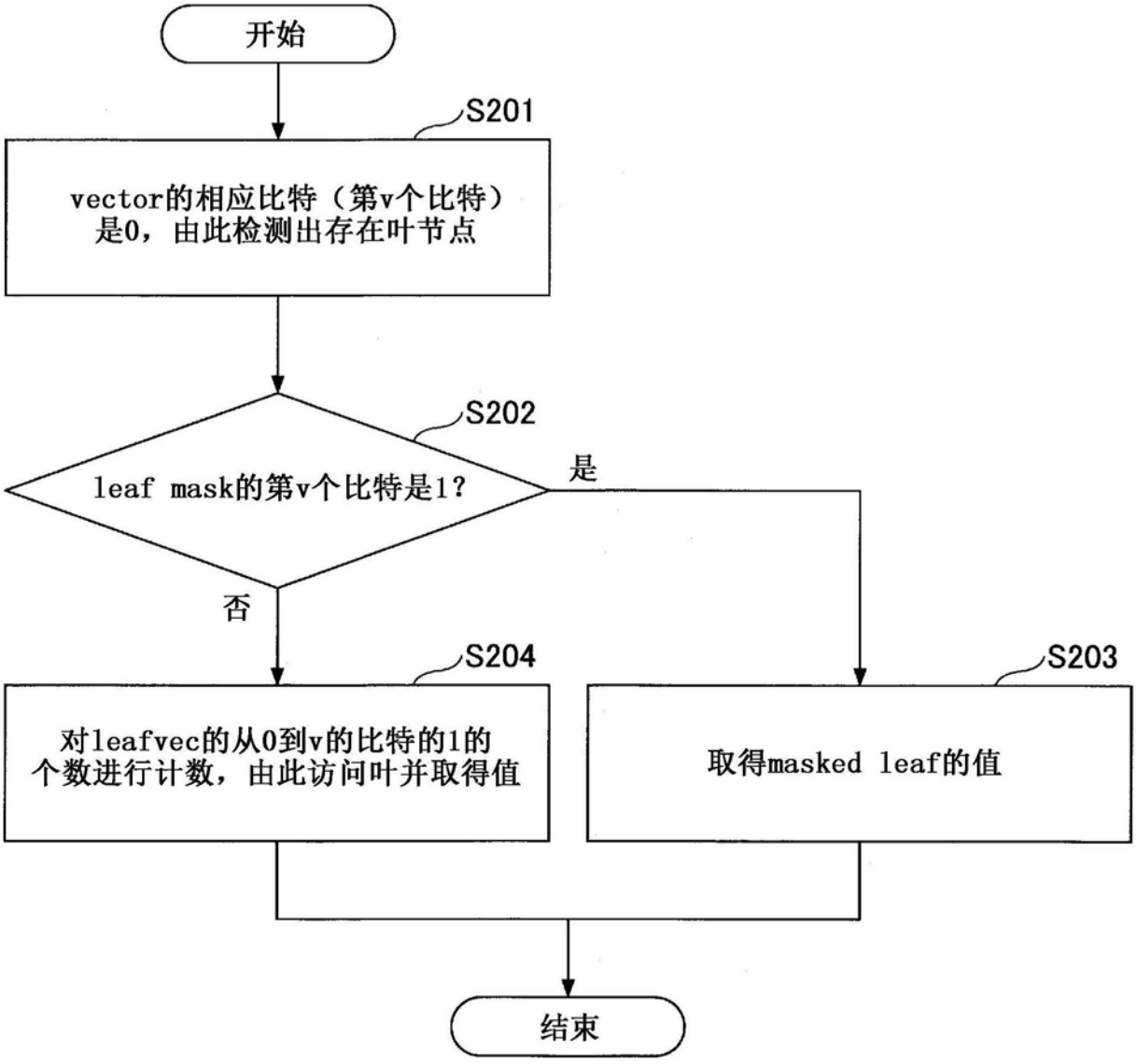

开始
S201
vector的相应比特（第v个比特）是0，由此检测出存在叶节点
S202
leaf mask的第v个比特是1？
是
否
S204
对leafvec的从0到v的比特的1的个数进行计数，由此访问叶并取得值
S203
取得masked leaf的值
结束

图15

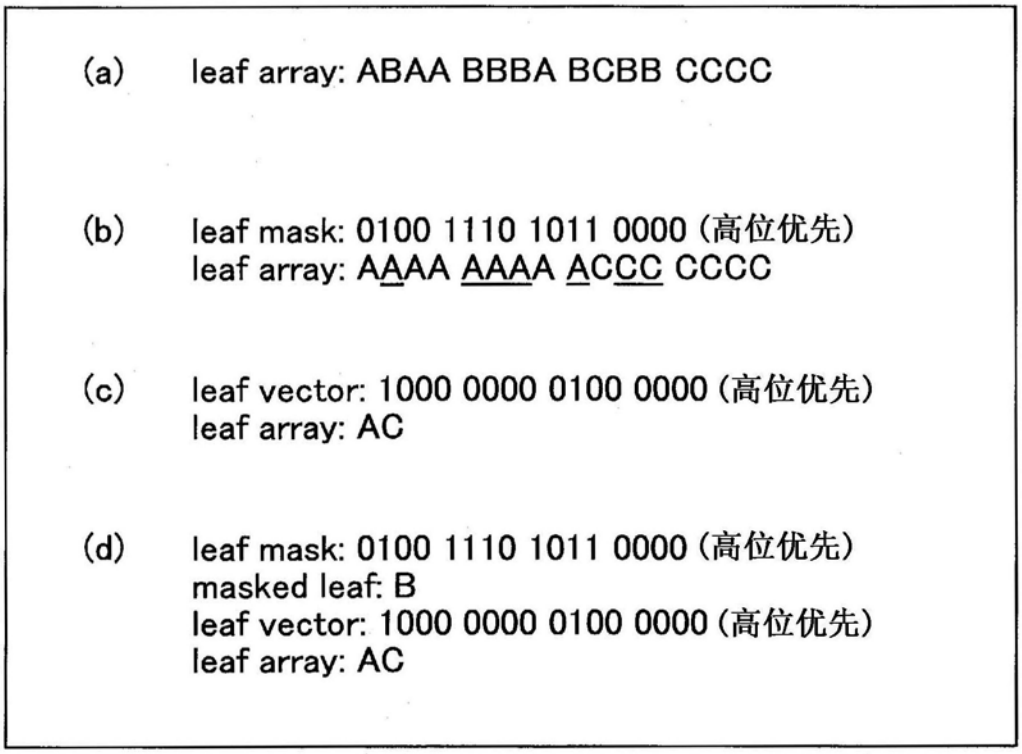

(a) leaf array: ABAA BBBA BCBB CCCC
(b) leaf mask: 0100 1110 1011 0000 (高位优先)
leaf array: AAAA AAAA ACCC CCCC
(c) leaf vector: 1000 0000 0100 0000 (高位优先)
leaf array: AC
(d) leaf mask: 0100 1110 1011 0000 (高位优先)
masked leaf: B
leaf vector: 1000 0000 0100 0000 (高位优先)
leaf array: AC

图16