**说 明 书 摘 要**

本发明公开了一种动态存储键值对的布鲁姆过滤器树结构及键值对存储方法，布鲁姆过滤器树结构包括完全d叉树；其特征在于，每一棵完全d叉树的每一个节点都是一个布鲁姆过滤器；每一棵完全d叉树的每个叶子节点表示一个值value；每一个节点的存储单元大小是该节点的父节点的存储单元大小的一半，根节点包括个不相同的哈希函数，即根节点包括d个哈希组，每组中包含k个哈希函数。本发明在数据库交互查询、高速网络中资源定位、计算机网络监控等产生大量数据、需要进行键值对查询的应用领域，可以大大减少集合查询的时间，降低资源消耗，处理动态到达的数据，适应网络环境。

摘要附图为图1

**权 利 要 求 书**

1. 一种动态存储键值对的布鲁姆过滤器树结构，包括完全d叉树；其特征在于，每一棵完全d叉树的每一个节点都是一个布鲁姆过滤器；每一棵完全d叉树的每个叶子节点表示一个值value；每一个节点的存储单元大小是该节点的父节点的存储单元大小的一半，根节点包括个不相同的哈希函数，即根节点包括d个哈希组，每组中包含k个哈希函数。
2. 一种利用权利要求1所述的布鲁姆过滤器树结构进行键值对存储的方法，其特征在于，包括如下操作：

插入操作：当需要插入一个键值对(key,value)时，首先查看value是否已经插入到布鲁姆过滤器中，如果未插入，则需要进行后续的增加新value操作；如果value已经存在于布鲁姆过滤器树中，则直接进行键值对插入，根据value查找该value对应的叶节点，得到此叶节点位置编码，确定一条从根节点到此叶节点的唯一路径，计算根节点的两组哈希函数，即对key使用k个哈希函数，计算，；其中表示选取哈希函数的组号，；将计算的哈希值保存在数组A中，将计算的哈希值保存在数组B中；接着，根据叶节点的第一位编码，在根节点选取A、B数组中的一组值进行插入，即对A或B数组进行右移零位操作，然后根据叶节点的第一位编码，得到下一个需要进行插入操作的布鲁姆过滤器，再根据叶节点的第二位编码，选取A、B数组中的一组值进行右移一位操作，得到插入位置，进行插入；继续进行上述操作，对每一层布鲁姆过滤器插入key，直到插到value对应的叶节点为止；

查询操作：需要查询一个键值key对应的value时，首先，计算根节点的两组哈希函数，将它们的值分别保存在A、B两组数组中；在根节点分别对A、B两数组值对应的位置单元进行查询操作，查到第一位编码值；根据得到的编码值，转到下一个节点继续查询，此时，对A、B两数组进行右移操作，在对应的位置单元进行查询，得到一个编码值；继续进行上述操作，直到查找到叶节点为止，最后得到一个完整的编码值，根据这个叶节点编码进行解码操作得到对应的value；

增加新value操作：要添加一个新的value到布鲁姆过滤器中，分以下两种情况：如果原来的布鲁姆过滤器是一棵未满二叉树，当需要添加新的value时，直接在布鲁姆过滤器的末尾添加新的叶节点，这个叶节点表示新添加的value；如果原来的布鲁姆过滤器是一棵满二叉树，则在根节点的上方添加一个新布鲁姆过滤器，这个新添加的布鲁姆过滤器的大小是原布鲁姆过滤器树的根节点的2倍，此时，这个新的布鲁姆过滤器就变成了新的布鲁姆过滤器树的根节点，原来的布鲁姆过滤器就变成了这个新根节点的左子树，再创建一棵比原布鲁姆过滤器树高少一层的满二叉树作为新的根节点的右子树，把原根节点上置1的位置全部左移一位，插入到新的根节点中，此时新的根节点的两组哈希函数比原来的两组哈希函数多一层，即在基函数中多选取一行，继续在布鲁姆过滤器的末尾添加新的叶节点。

1. 根据权利要求2所述的方法，其特征在于，所述插入操作中，在根节点选取待插入的数组的方法为：如果编码值为0，则选取A数组，如果编码值为1，则选取B数组。
2. 根据权利要求2所述的方法，其特征在于，所述插入操作中，根据叶节点的第一位编码，得到下一个需要进行插入操作的布鲁姆过滤器的方法为：如果编码值为0，则操作左节点，如果编码值为1，则操作右节点。

**说 明 书**

动态存储键值对的布鲁姆过滤器树结构及键值对存储方法

技术领域

本发明涉及计算机网络领域和计算机系统存储领域，特别是高性能、高吞吐量的交互查询的应用领域，具体是一种可扩展的布鲁姆树对于键值对的存储结构及方法。

背景技术

近年来，随着计算机的飞速发展，数据库，网络和其他应用中的集合规模呈几何增长。存储和查询键值对（key，value）是计算机系统中常见的任务，这就需要设计对应的键值对存储数据结构，支持快速的键值对查询。键值对操作经常出现在网络和存储系统中，如key-value数据库MongoDB，CouchDB。每个放入到键值对存储系统中的唯一键key都对应一个value，例如(3，5)是一个键key为3，值value为5的键值对，将(3，5)存储到键值对存储系统后，可以通过查询键(key)为3，得到值(value)为5。

设计高效的键值对存储及查询结构带来了巨大的挑战。在一个2层交换机中，一个MAC地址关联一个唯一的端口。当要转发一帧时，搜索引擎会查询这一帧要转发的目的地址的MAC表，因此，将一个MAC地址映射到一个端口的问题就转换成了一个键值对查询问题，这个时候，MAC地址就被看成了key，而要查询的端口号就变成了value。由于MAC地址是持续添加到列表中的，因此，元素的大小是未知的。如果采用cell的结构来存储这些键值对，需要消耗大量的空间，而且在查找对应key的value时会消耗大量的时间；如果采用静态的布鲁姆过滤器结构存储键值对，只能处理静态数据，这在实际应用中很不现实。因此，在高速的计算机网络中，如何高效存储这些信息，快速查询对应的键值对成为挑战。

布鲁姆过滤器(Bloom Filter)是一种空间节俭、查询高效的数据结构，它可以满足现如今生活中高效资源交互需求及查找需求，能够有效的表示数据集合。布鲁姆过滤器自1970年由B.Bloom提出以来，就被广泛应用于各种各样的计算机系统之中，用来表示庞大的数据集合，提高查询效率。布鲁姆过滤器结构实质是将集合中元素通过k个哈希函数映射到位向量中。布鲁姆过滤器在达到其高效表示集合的同时，进行元素查询时却存在一定的假阳性（某元素不属于集合而误判为属于集合中）误判率，而不存在假阴性（某元素属于集合而误判为不属于集合中）误判，在查询和存储上有较高的效率。

但是传统的布鲁姆过滤器仅仅只能支持元素是否存在于集合的从属查询。如果元素为key，那么只能支持key是否存在于集合的从属查询，而无法支持(key, value)操作。因为布鲁姆过滤器无法直接存储value，因此并不能使用传统的布鲁姆过滤器对键值对进行操作。要使得布鲁姆过滤器支持键值对的基本操作，必须要改进传统的布鲁姆过滤器，设计新的布鲁姆过滤器结构。

发明内容

本发明所要解决的技术问题是，针对现有技术不足，提供一种动态存储键值对的布鲁姆过滤器树结构及键值对存储方法。

为解决上述技术问题，本发明所采用的技术方案是：一种动态存储键值对的布鲁姆过滤器树结构，包括完全d叉树；每一棵完全d叉树的每一个节点都是一个布鲁姆过滤器；每一棵完全d叉树的每个叶子节点表示一个值value；每一个节点的存储单元大小是该节点的父节点的存储单元大小的一半，根节点包括个不相同的哈希函数，即根节点包括d个哈希组，每组中包含k个哈希函数。

相应地，本发明还提供了一种布鲁姆过滤器树结构进行键值对存储的方法，其包括：

插入操作：当需要插入一个键值对(key,value)时，首先查看value是否已经插入到布鲁姆过滤器中，如果未插入，则需要进行后续的增加新value操作；如果value已经存在于布鲁姆过滤器树中，则直接进行键值对插入，根据value查找该value对应的叶节点，得到此叶节点位置编码，确定一条从根节点到此叶节点的唯一路径，计算根节点的两组哈希函数，即对key使用k个哈希函数，计算，；其中表示选取哈希函数的组号，；将计算的哈希值保存在数组A中，将计算的哈希值保存在数组B中；接着，根据叶节点的第一位编码，在根节点选取A、B数组中的一组值进行插入，即对A或B数组进行右移零位操作，然后根据叶节点的第一位编码，得到下一个需要进行插入操作的布鲁姆过滤器，再根据叶节点的第二位编码，选取A、B数组中的一组值进行右移一位操作，得到插入位置，进行插入；继续进行上述操作，对每一层布鲁姆过滤器插入key，直到插到value对应的叶节点为止；

查询操作：需要查询一个键值key对应的value时，首先，计算根节点的两组哈希函数，将它们的值分别保存在A、B两组数组中；在根节点分别对A、B两数组值对应的位置单元进行查询操作，查到第一位编码值；根据得到的编码值，转到下一个节点继续查询，此时，对A、B两数组进行右移操作，在对应的位置单元进行查询，得到一个编码值；继续进行上述操作，直到查找到叶节点为止，最后得到一个完整的编码值，根据这个叶节点编码进行解码操作得到对应的value；

增加新value操作：要添加一个新的value到布鲁姆过滤器中，分以下两种情况：如果原来的布鲁姆过滤器是一棵未满二叉树，当需要添加新的value时，直接在布鲁姆过滤器的末尾添加新的叶节点，这个叶节点表示新添加的value；如果原来的布鲁姆过滤器是一棵满二叉树，则在根节点的上方添加一个新布鲁姆过滤器，这个新添加的布鲁姆过滤器的大小是原布鲁姆过滤器树的根节点的2倍，此时，这个新的布鲁姆过滤器就变成了新的布鲁姆过滤器树的根节点，原来的布鲁姆过滤器就变成了这个新根节点的左子树，再创建一棵比原布鲁姆过滤器树高少一层的满二叉树作为新的根节点的右子树，把原根节点上置1的位置全部左移一位，插入到新的根节点中，此时新的根节点的两组哈希函数比原来的两组哈希函数多一层，即在基函数中多选取一行，继续在布鲁姆过滤器的末尾添加新的叶节点。

所述插入操作中，在根节点选取待插入的数组的方法为：如果编码值为0，则选取A数组，如果编码值为1，则选取B数组。

所述插入操作中，根据叶节点的第一位编码，得到下一个需要进行插入操作的布鲁姆过滤器的方法为：如果编码值为0，则操作左节点，如果编码值为1，则操作右节点。

与现有技术相比，本发明所具有的有益效果为：本发明在数据库交互查询、高速网络中资源定位、计算机网络监控等产生大量数据、需要进行键值对查询的应用领域，可以大大减少集合查询的时间，降低资源消耗，处理动态到达的数据，适应网络环境。

附图说明

图1是布鲁姆树的结构示意图，它将布鲁姆过滤器与二叉树的结构相结合，每一个节点都是一个布鲁姆过滤器。根节点有两组H3哈希函数，每组有k个哈希函数。一个叶节点就表示一个value，根据叶节点的编码可以得到一条由根节点到叶节点的唯一路径。根据叶节点编码，如果编码值为0，则对第一组哈希函数进行移位操作；如果编码值为1，则对第二组哈希函数进行移位操作。图1中就是得来的，就是得来的，而是得来的，是得来的。

图2是在未满二叉树中增加新的value操作示意图。当一个未存在于叶节点的value需要插入时，如果二叉树不是满二叉树，则可以直接在树的末尾添加叶节点，表示插入的value。如图中编码为11的叶节点就是新添加的value。

图3是在满二叉树中增加新的value操作示意图。当一个未存在于叶节点的value需要插入时，如果二叉树是满二叉树，则不可以在原树上添加叶节点，此时需要往上扩展一层，如图3中新添加的level 3。原二叉树就变成了level 3的左子树，再构建一棵比原二叉树少一层的满二叉树作为level 3的右子树。此时，level 3也存在两组H3哈希函数，其中是由得到，将原二叉树根节点中所有置向左移一位，插入到level 3中；而则是从原来的基矩阵中比多选取一行即可。此时构建了一棵新的未满二叉树，可以直接在树的末尾添加叶节点，表示插入的value。如图中编码为100的叶节点就是新添加的value。

图4是实施环境数据详细信息，包括数据来源，数据时间，包大小，本实施例与最近几年研究成果进行了对比，其中包括：Bloom Tree是2014年INFOCOM中的论文“Bloom tree: A search tree based on bloom filters for multiple-set membership testing” 提出的一种将树和布鲁姆过滤器结构结合起来进行键值对查询的结构，该算法在每一个布鲁姆过滤器节点上都使用了d组哈希函数，从根节点开始计算，直到找到value对应的叶节点为止，该算法只能对静态数据进行处理；COMB是2012年IEEE/ACM Transactions on Networking中的论文“Fast dynamic multiple-set membership testing using combinatorial bloom filters” 提出的一种基于布鲁姆过滤器的键值对查询算法，该算法将value值进行编码后，对已知的value组数选取合适的布鲁姆过滤器数目和需要插值的布鲁姆过滤器数目，查询时则需要对所有布鲁姆过滤器进行查询，得到相应的编码。

图5（a）～图5（f）是固定大小的布鲁姆过滤器树bit，在不同哈希函数个数下，使用三种不同算法SBFT，Bloom Tree和COMB，处理1024组数据，在对数据的平均处理时间上进行比较。图5（a）为使用数据集MAWI 1进行仿真实验得到的实验结果示意图，图5（b）为使用数据集MAWI 2进行仿真实验得到的实验结果示意图，图5（c）为使用数据集MAWI 3进行仿真实验得到的实验结果示意图，图5（d）为使用数据集ClarkNet-HTTP进行仿真实验得到的实验结果示意图，图5（e）为使用数据集UMass进行仿真实验得到的实验结果示意图，图5（f）为使用数据集TKN进行仿真实验得到的实验结果示意图。图5（a）～图5（f）中所有数据结果显示，针对每种数据集，我们所提出的算法SBFT平均处理数据耗时最少。与Bloom Tree相比，我们的算法只需要在根节点选取d组哈希函数计算，然后进行移位操作，而Bloom Tree则需要每个节点都选取d组哈希函数计算，耗时较大；与COMB相比也类似，COMB对每一个布鲁姆过滤器都是需要选取多个哈希函数，耗时大，但COMB需要操作的布鲁姆过滤器数目比Bloom Tree要少，因此，耗时会比Bloom Tree小。

图6是使用三种算法处理静态数据时内存消耗状态示意图。数据结果显示，三种算法消耗的内存是一样的，也就是说，我们的算法在处理数据耗时最少的情况下也不会占用多余的内存。

图7（a）～图7（f）是数据包不断增加时，三种算法处理数据包耗时情况示意图。每组使用k=3个哈希函数。由于Bloom Tree和COMB两种算法是在已知数据量情况下进行操作的，因此当数据包还未插入时，就需要把结构搭建好。而我们的算法则可以对未知数据量进行操作，随着数据包的到来处理数据，这是动态处理数据的过程。图7（a）为使用数据集MAWI 1进行仿真实验得到的实验结果示意图，图7（b）为使用数据集MAWI 2进行仿真实验得到的实验结果示意图，图7（c）为使用数据集MAWI 3进行仿真实验得到的实验结果示意图，图7（d）为使用数据集ClarkNet-HTTP进行仿真实验得到的实验结果示意图，图7（e）为使用数据集UMass进行仿真实验得到的实验结果示意图，图7（f）为使用数据集TKN进行仿真实验得到的实验结果示意图。图7中所有数据结果显示，针对每种数据集，我们的算法处理数据包是一个动态的过程，并且处理数据耗时比Bloom Tree和COMB都要少，而Bloom Tree和COMB处理数据包则是一个静态过程，并且耗时较多。

图8（a）～图8（f）是数据包不断增加时，三种算法处理数据包消耗内存情况示意图。每组使用k=3个哈希函数。图8（a）为使用数据集MAWI 1进行仿真实验得到的实验结果示意图，图8（b）为使用数据集MAWI 2进行仿真实验得到的实验结果示意图，图8（c）为使用数据集MAWI 3进行仿真实验得到的实验结果示意图，图8（d）为使用数据集ClarkNet-HTTP进行仿真实验得到的实验结果示意图，图8（e）为使用数据集UMass进行仿真实验得到的实验结果示意图，图8（f）为使用数据集TKN进行仿真实验得到的实验结果示意图。图8（a）～图8（f）中所有数据结果显示，针对每种数据集，我们的算法处理数据包是一个动态的过程，我们的算法是边处理数据边搭建结构的过程，而Bloom Tree和COMB处理数据包则是一个静态过程，在处理数据之前就已经把结构搭建好了，这也说明它们的结构只能处理静态数据。

具体实施方式

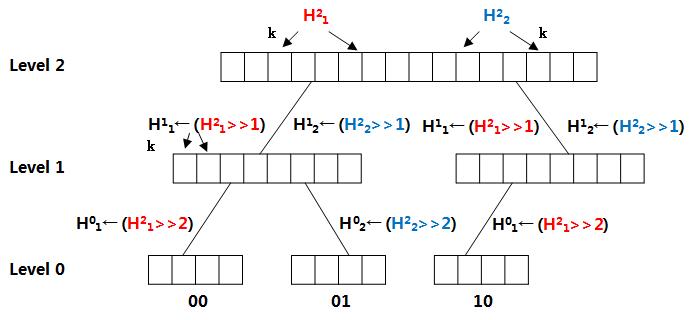
本实施中处理静态数据时，选取的内存大小bit，根节点占用了95325bit，选取了18行8列的H3哈希函数基矩阵，在根节点上提取了基矩阵的前16行，每组选取k=3个哈希函数，处理组数g=1024组的数据，此时树高h=10。图1是静态布鲁姆树结构示意图，可以由此图分析键值对插入，查询过程。

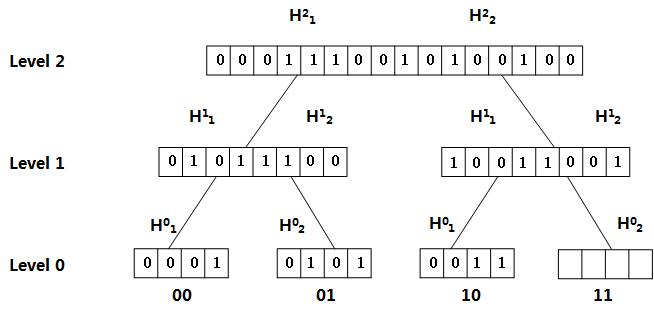
插入操作Insert(key,value)过程：根据value查找该value对应的叶节点，得到此叶节点位置编码，就可以确定一条从根节点到此叶节点的唯一路径。计算根节点的两组哈希函数，对key使用k个哈希函数（表示选取哈希函数的组号）计算，将它们的值分别保存在两个数组A、B中。接着，根据叶节点的第一位编码，在根节点选取A、B数组中的一组值进行插入（如果编码值为0，则选取A数组，如果编码值为1，则选取B数组），即相当于对A、B数组进行了右移零位操作（A>>0或B>>0）。然后根据叶节点的第一位编码，得到下一个需要进行插入操作的布鲁姆过滤器（如果编码值为0，则操作左节点，如果编码值为1，则操作右节点）。再根据叶节点的第二位编码，选取A、B数组中的一组值进行右移一位操作（A>>1或B>>1），得到插入位置。继续进行类似的操作，对每一层布鲁姆过滤器插入key，直到插到value对应的叶节点为止。如图1，假设插入的键值对中value对应的编码为01，计算根节点的两组哈希函数，将它们的值分别保存在两个数组A、B中；由于编码第一位为0，则选择数组A在根节点右移0位A>>0插入根节点中，下一步到达左节点；由于编码第二位为1，则选择数组B在此节点右移一位B>>1插入节点中，下一步到达右节点；最后直接对数组B右移两位B>>2插入节点，完成插入操作。

查询Query(key)过程：首先，计算根节点的两组哈希函数，其中表示选取哈希函数的组号，将计算的哈希值保存在数组A中，将计算的哈希值保存在数组B中。在根节点分别对A、B两数组值对应的位置单元进行查询操作，如果对应位置的值都为1，即为满足，查到第一位编码值（如果A数组满足，则编码为0，如果B数组满足，则编码为1，如果都不满足，则该key值不存在，结束查询）。根据得到的编码值，转到下一个节点继续查询，此时，需要对A、B两数组进行右移操作（A>>1，B>>1），在对应的位置单元进行查询，得到一个编码值。继续进行类似操作，直到查找到叶节点为止。最后得到一个完整的编码值，根据这个叶节点编码就可以得到对应的value。由图1举例说明：假设查询key，首先计算根节点的两组哈希函数，将它们的值分别保存在A、B两组数组中；将A、B两数组的值分别在根节点中查询，A数组中的值在此节点中对应位置单元都为1，则记录下第一位编码为0，下一个操作节点是左节点；将A、B两数组右移一位A>>1，B>>1，分别在此节点中进行查询，B数组中的值在此节点中对应位置单元都为1，则记录下第二位编码为1，下一个操作节点是右节点；最后一步，直接对B数组右移两位B>>2进行查询，即最后的验证，这一步不用记录编码值，验证通过则查询到的编码为01，在表中找到对应的value即可，完成查询。

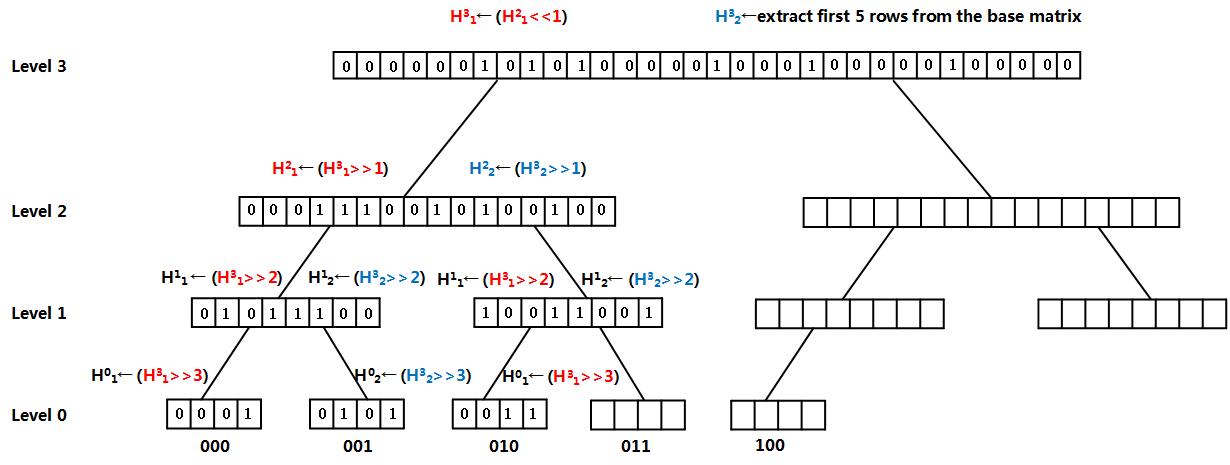
本实施例中处理动态数据时，增加新value操作分两种情况：如果原来的布鲁姆树是一棵未满二叉树，当需要添加新的value时，可以直接在树的末尾添加新的叶节点，这个叶节点就表示新添加的value，如图2编码为11的新的叶子节点；如果原来的布鲁姆树是一棵满二叉树，此时就需要在根节点的上方添加一个新布鲁姆过滤器，这个布鲁姆过滤器的大小是原根节点的2倍，此时，这个新的布鲁姆过滤器就是新的根节点，原来的布鲁姆树就变成了这个根节点的左子树，再创建一棵比原布鲁姆树高少一层的满二叉树作为新的根节点的右子树，把原根节点上置1的位置全部左移一位，插入到新的根节点中，此时根节点的两组哈希函数要比原来的两组哈希函数多一层，即在基函数中多选取一行，这时，就可以继续在树的末尾添加新的叶节点了，如图3中的编码为100的新的叶节点。

**说 明 书 附 图**

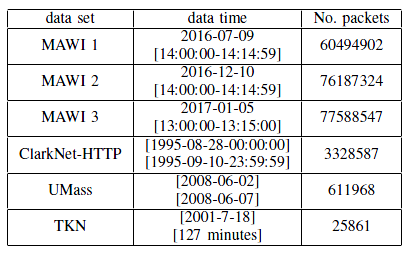
**图1**

****

**图2**

****

**图3**



**图4**

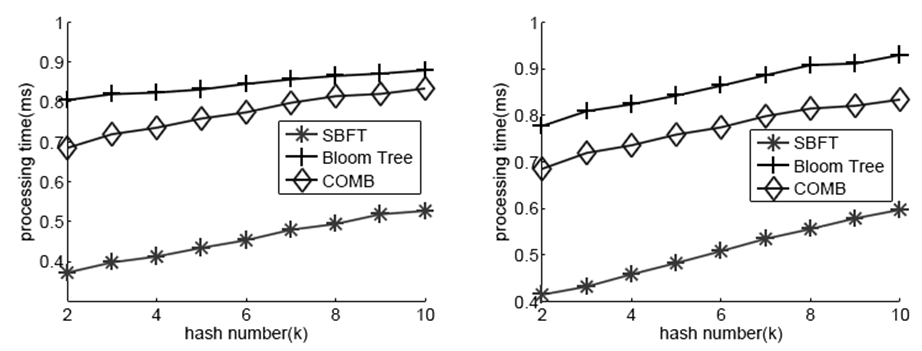


图5（a） 图5（b）

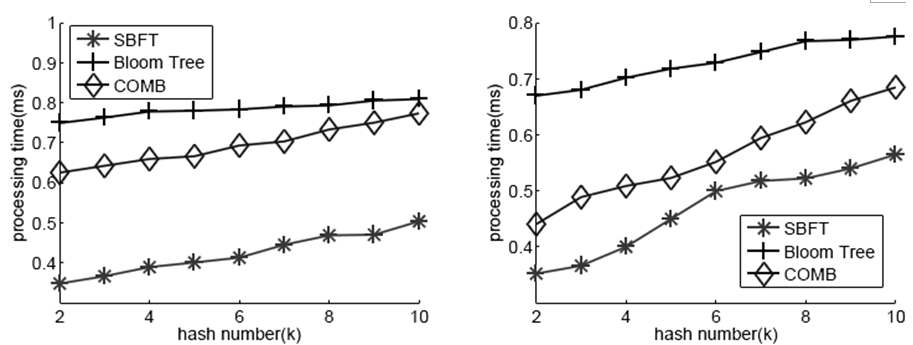


图5（c） 图5（d）

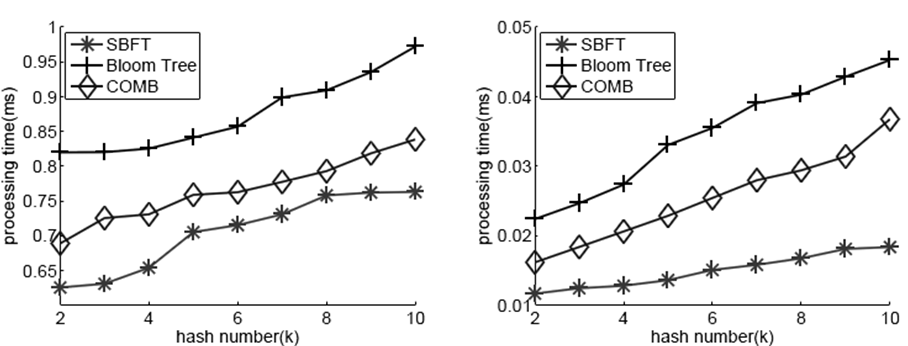
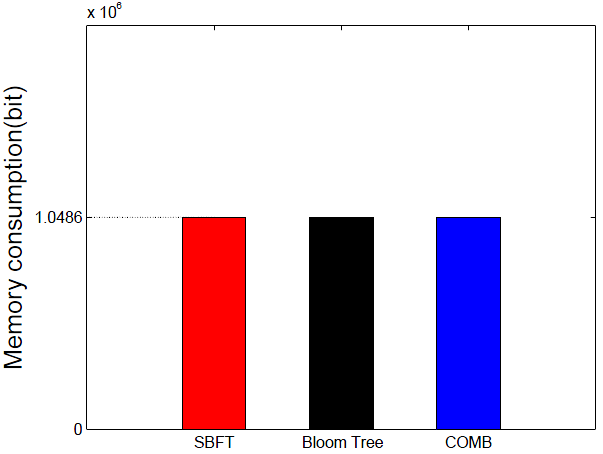


图5（e） 图5（f）



**图6**

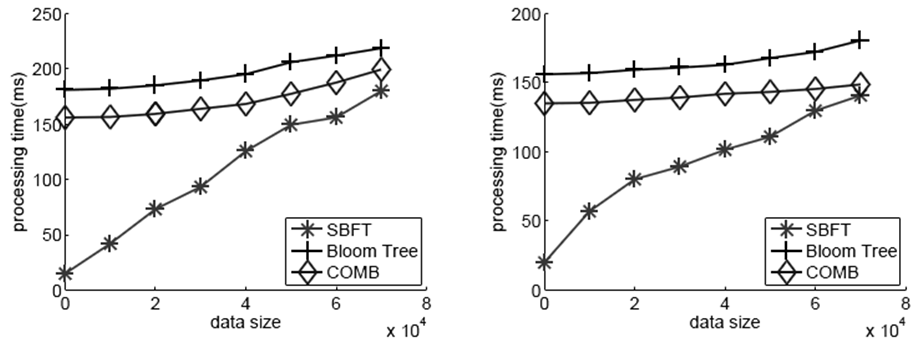


图7（a） 图7（b）

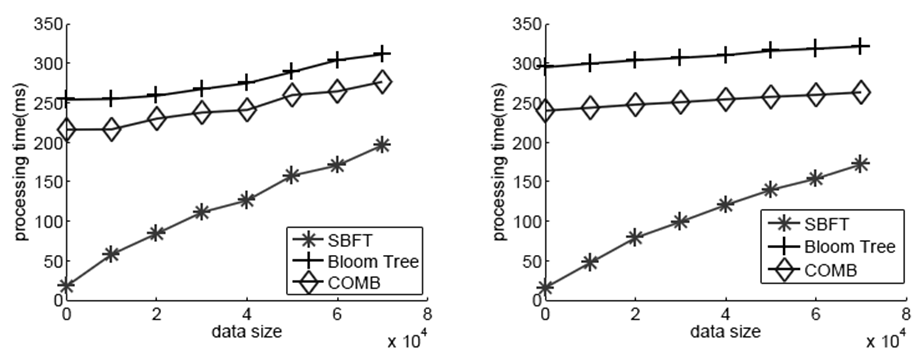


图7（c） 图7（d）

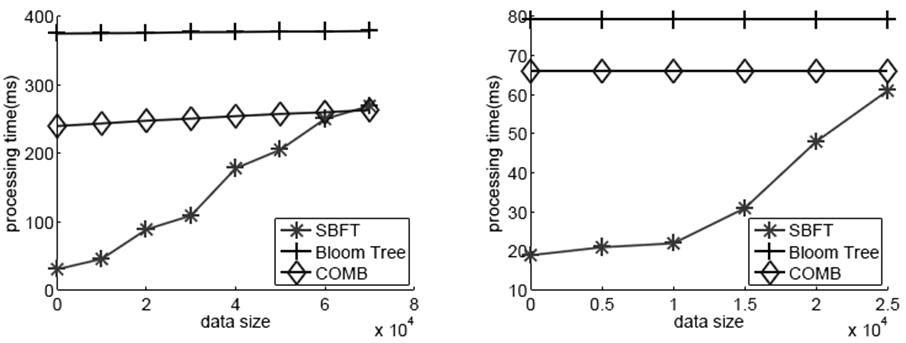


图7（e） 图7（f）

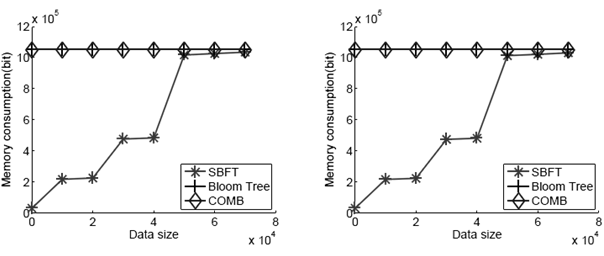


图8（a） 图8（b）

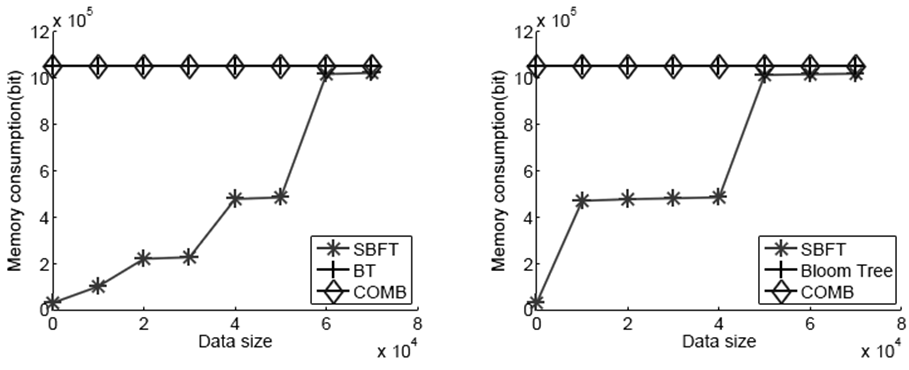


图8（c） 图8（d）

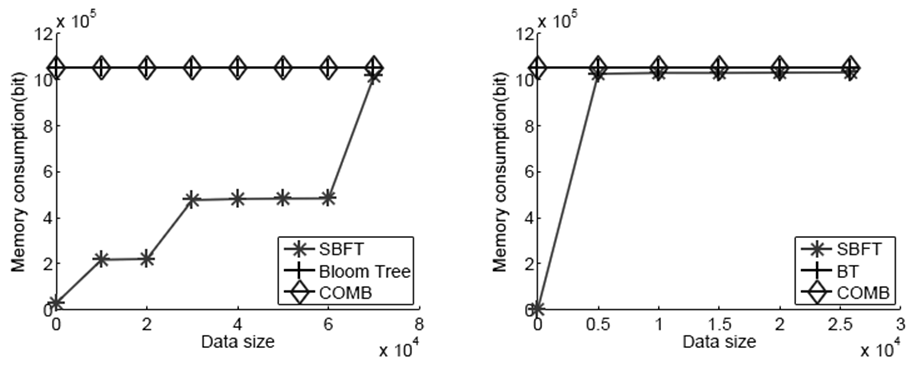


图8（e） 图8（f）