Skip List

Date: 2019-06-01

Chapter 1: Introduction

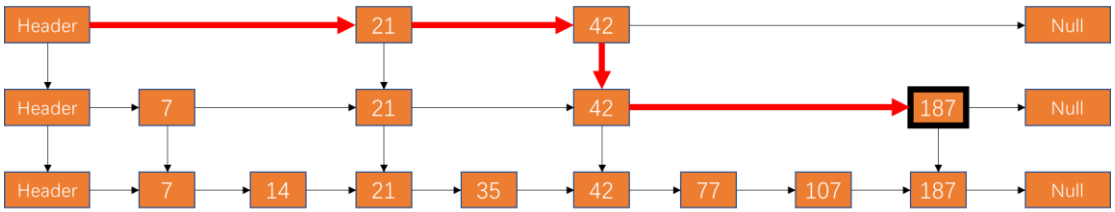
1.1 题目描述

跳跃表(Skip list)是一种数据结构，其插入和查找操作的时间复杂度为 O(logN)。题目要求我们构建跳跃表结构，实现插入、删除、查找操作，并证明这些操作的时间复杂度为 O(logN)。

1.2 跳跃表简介

跳跃表的基本结构为：在有序链表的基础上，增加一些前进到该结点之后的某个结点的链接，因此查找时可以快速地跳过部分结点，而不必将所有结点全部遍历。

下图以一种直观的表示形式展示了跳跃表中查找元素 187 的过程。如果是普通的有序链表，需要访问 8 个值才能得到结果，而跳跃表只需访问 3 个值，大大减少了查找操作的开销。



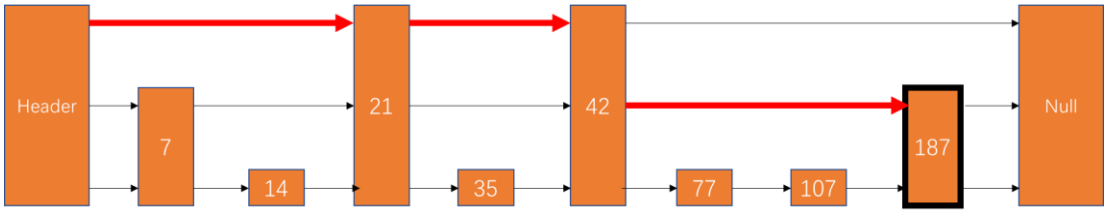
为使得每次查找的跳跃更为有效，应设计某种算法，使得跳跃表较高层的结点个数比较低层少很多。因此，倘若每一层的结点个数都大约是上一层的某个整数倍，整个表的结点就类似一棵树的结构，高度为 log 级别。因此，跳跃表可以加速有序链表的查找操作。

跳跃表中运用了随机化的思想，从而维持两层之间结点个数的倍数关系。算法实现详见下文。

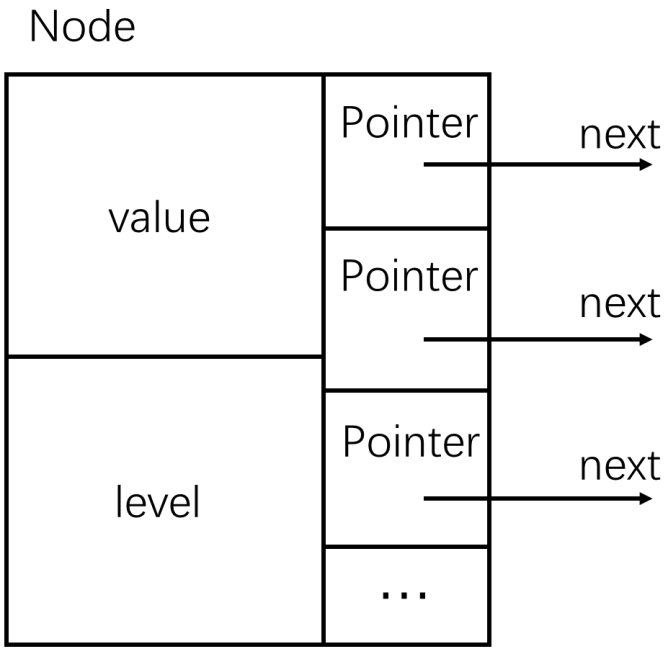
Chapter 2: Data Structure / Algorithm Specification

2.1 数据结构图示

为避免重复储存结点中的值，我们存储了每个结点的高度，并存储了一个指针数组，用于记录每一层的 next 指针。因此，上面的例子用我们的数据结构来实现的图示如下：



因此，每个结点中需要存放该结点的数据值 value，该结点的高度 level，以及每一层对应的 next 指针。指针的数量与高度 level 相同。Node 结构的示意图如下，其中的省略号代表若干 next 指针。



2.2 数据结构定义

首先，根据上述图示，给出结点 Node 的结构体定义：

```
typedef struct _Node Node;
typedef struct _Pointer
{
    Node * next;
} Pointer;
struct _Node
{
    ElementType value;
    int level;
    Pointer ** pointer_list;
};
```

而一个跳跃表结构体中不仅要有头指针，还要有一些参数来控制表的行为。因此，给出跳跃表 SkipList 的结构体定义：

```
typedef struct _SkipList
{
    Node * head;
    int prob;
    int MAXLEVEL;
    int level;
    int number;
} SkipList;
```

其中，head 为指向头结点的指针。我们采用了空的头结点(dummy head)作为头指针，避免了 head 指针为 NULL 的情况。

prob 为每层结点数与上一层的比值，即上文中提到的控制每层结点数量的参数。我们的代码中取 prob = 2, 即每层结点的数量大约是上一层的 2 倍。

MAXLEVEL 为跳跃表不能超过的最大高度，level 为当前跳跃表中最高结点的高度，number 为跳跃表中的结点个数（不包含头结点）

2.3 核心函数算法描述

2.3.1 查找

进行查找操作时，如上文的例子中所示，我们从跳跃表的顶层开始，找到不大于查找的值的最大值的结点位置。如果这个值等于查找的值，则查找成功，返回该结点；如果到达了底层仍然没有找到，说明表中没有这个值，返回空结点。伪代码描述如下：

```
for i from list->level-1 to 0:
    while value < nextNode->value:
        nowNode = nextNode;

    if value == nowNode->value:
        return nowNode;
    else if level == 0:
        return NULL;
```

2.3.2 插入

插入新结点时，首先需要根据随机算法获得新结点的高度，之后在结点高度的每一层都把新结点插入进去。

先根据表中总结点数，计算出该结点可以获得的最高高度。由于总结点数应该是高度的指数倍，因此每次取一个随机数，如果能够整除跳跃表中定义的 prob 值，则高度增加，否则将当前值即为结点高度。伪代码描述如下：

```
// calculate max level
MaxLevel = 0;
while number > 0 && MaxLevel < list->MAXLEVEL:
    number /= prob;
    MaxLevel ++;

// randomly get the level of new node
newLevel = 0;
while rand() % prob == 0 && newLevel < MaxLevel:
    newLevel ++;

```

获得高度后，申请内存构建一个新结点插入表中。从表的最高层开始，找到当前层中比插入值小的最大值，记录该位置后往下一层继续查找。直到最底层后，从新结点的最高层开始，把新结点插入当前的层，直到最底层为止。伪代码描述如下：

```
// create new node
newLevel = randomly get the level of the new Node;
newNode = create new node <-- newLevel;

// find the position to insert from top to bottom
for i from list->level-1 to 0:
    while insert value > nextNode->value:
            nowNode = nextNode;
    // if the value is already in the list, quit
    if insert value == nowNode->value:
            quit;
    else
            record the node in this level;

// insert the value into each level
for i from newNode->level-1 to 0:
    newNode->next = recorded Node->next;
    recorded Node->next = newNode;

// update info of the list
update node number and level in the list
```

2.3.3 删除

删除操作时，先需要执行查找操作，找到需要删除的结点。如果能找到，则在该结点的所有层数中删除该结点，返回删除后的表；如果找不到，则返回原表。伪代码描述如下：

```
for i from list->level-1 to 0:
    while value < nextNode->value:
            nowNode = nextNode;

    if value == nowNode->value:
            prevNode->next = nextNode;
            delete nowNode;
```

Chapter 3: Testing Results

3.1 表格数据

为充分体现跳跃表的优越性，我们同时构建了普通的有序链表，并让二者对相同的测试数据集进行操作，并计算出他们的运行时间。得到的测试结果如下：

3.2 图表数据

将上述表格绘制成图像，我们可以得到更加直观的比较结果。绘制出的图像如下图所示：

Chapter 4: Analysis and Comments

4.1 时间复杂度分析

$$\begin{cases} \text{查找：O(logN)} \\ \text{插入：O(logN)} \\ \text{删除：O(logN)} \end{cases}$$

对跳跃表的所有操作的时间复杂度均为 O(logN)，因为这些操作均基于查找操作。

由于构建跳跃表时，随机算法保证了每一层的结点个数都大约是上一层的 prob 倍（实际实现中 prob = 2），因此有：

$$N_{i+1} \approx \text{prob} \times N_i$$

将每一层的结点求和，可以得到总结点数 N 的表达式：

$$N = \sum_{i=0}^{level} N_i \approx \text{C} \times level^{prob}$$

其中 level 为跳跃表的高度，C 为比例系数。

对两边求和可得：level = O(logN)

4.2 空间复杂度分析

由于每个结点都只储存了一个 value 值，因此表中总共有 N 个结点。而每个结点的最高层数是由 MAXLEVEL 确定的，因此占用的空间为 MAXLEVEL * N * sizeof(Node)。

因此，跳跃表的空间复杂度为 O(N)。

Appendix: Source Code (if required)

At least 30% of the lines must be commented. Otherwise the code will NOT be evaluated.

Author List

Programmer: null

Tester: null

Reporter: null

Declaration

We hereby declare that all the work done in this project titled "Skip List" is of our independent effort as a group.