**北 京 邮 电 大 学**

**计算机应用编程实践**

**实验报告**



|  |  |
| --- | --- |
| 题目： | 树结构字符串检索 |
| 姓名： | 田昌昊 |
| 学号： | 2019110680 |
| 班级： | 2019111307 |

计算机学院

2019年10月

目录

[一、实验任务与实验内容 3](#_Toc22143908)

[1. 实验任务： 3](#_Toc22143909)

[2. 挑战： 3](#_Toc22143910)

[3. 程序要求 3](#_Toc22143911)

[二、实验过程 3](#_Toc22143912)

[1. 主要的数据结构和流程 3](#_Toc22143913)

[1.1 版本一：二叉搜索树(Binary Search Tree) 3](#_Toc22143914)

[1.2 版本二：B+树(B Plus Tree) 5](#_Toc22143915)

[1.3 版本三radix4\_search 6](#_Toc22143916)

[1.4 版本四patricia\_search 7](#_Toc22143917)

[2. 实验过程 8](#_Toc22143918)

[三、遇到的问题 8](#_Toc22143919)

[1. 在radix4\_search中，分裂结点bug 8](#_Toc22143920)

[四、结果指标 9](#_Toc22143921)

[五、结论和总结 10](#_Toc22143922)

# 一、实验任务与实验内容

## 1. 实验任务：

1. 给定127万个中文字符串作为模式库
2. 待查找的98万个中文串
3. 从模式库中查找98万个串看是否能命中

## 2. 挑战：

模式串的规模较大，需要构造一个高效数据结构来处理

## 3. 程序要求

1. 分别实现四个版本，程序分别命名为：
   1. btree\_search（2叉树）：二叉平衡查找树
   2. bplus\_search（自定义）：m阶B+树
   3. radix4\_search（4叉树）：4叉01二进制数
   4. patricia\_search（2叉树）：2叉二进制数
2. 输入数据
   1. 字典串pattern.txt(127w个)
   2. 待匹配的98w字符串：words.txt
3. 实验结果result.txt
   1. 在模式串中的输出yes，不在就输出no
      1. Keyword1 yes
      2. Keyword2 no
      3. 最后一行输出四个数字，用空格分割：
         * 树结点个数
         * 树结构占用内存量（KB单位）
         * 字符比较次数（K）
         * words总个数
         * 成功检索到的word总个数

# 二、实验过程

## 1. 主要的数据结构和流程

### 1.1 版本一：二叉搜索树(Binary Search Tree)

主要数据结构如下：

typedef struct ValueList {

char\* value;

struct ValueList\* next;

}\*ValueListPtr;

typedef struct Node{

ValueListPtr value\_list;

unsigned int value\_hash;

enum BALANCE\_STATUS balance\_status; // 平衡因子，用来旋转结点，保证树的平衡；等于右子树高度-左子树高度

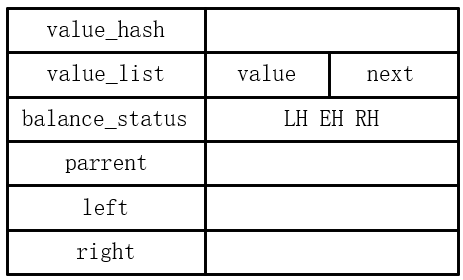
struct Node\* parent;

struct Node\* left;

struct Node\* right;

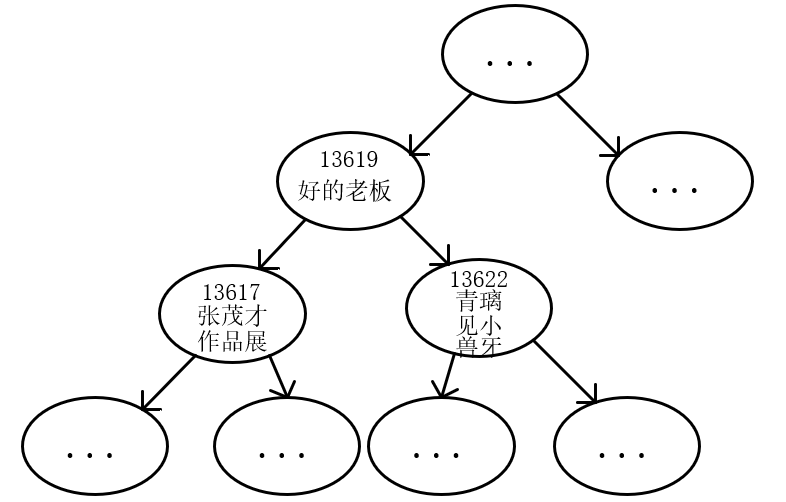
}\* NodePtr;

node结点结构如下：



上图中的value\_hash表示对字符串求hash值，结点之间的大小比较通过hash值进行，相同hash值的字符串会同时存于value\_list中；balance\_status用来记录当前结点作为根结点的树的平衡情况。parrent、left、right分别指向父结点、左孩子和右孩子。

平衡二叉树的结构如下：



在每个树结点中，都存有该结点的链表中的pattern对应的hash值，并由链表存有相同hash值的pattern字符串，在插入和比较时，优先比较hash值，当hash值相同时，进入链表进行pattern的字符串比较。其中，hash函数选取BKDRHash函数

本程序先读取patterns文件，获取数据，存于Node结构的树结点中，其中对内存的使用动态分配并记录。然后进行检索工作：遍历words数据文件，读取单词，并从patterns平衡二叉树中二分查找是否存在与之相等的字符串，如果检索到相同的字符串，则返回并将检索成功结果写入文件；如果检索结束仍然未找到相同字符串，则返回并将检索失败结果写入文件。最后将统计信息写入文件最后一行。

### 1.2 版本二：B+树(B Plus Tree)

主要数据结构如下：

typedef struct ValueList {

char\* value;

struct ValueList\* next;

}\*ValueListPtr;

typedef struct Leaf {

ValueListPtr value\_list; // 作为叶子结点的拉链表

unsigned int value\_hash;

struct Node\* parent;

struct Leaf\* last;

struct Leaf\* next;

}\*LeafPtr;

typedef struct Node {

int child\_num; // 记录当前结点的孩子数

bool is\_leaf; // 如果为True则表示孩子在leafs变量中，否则孩子在childs中

unsigned int value\_hash[M]; // 存储的hash值

struct Node\* childs[M]; // 指向非叶子结点

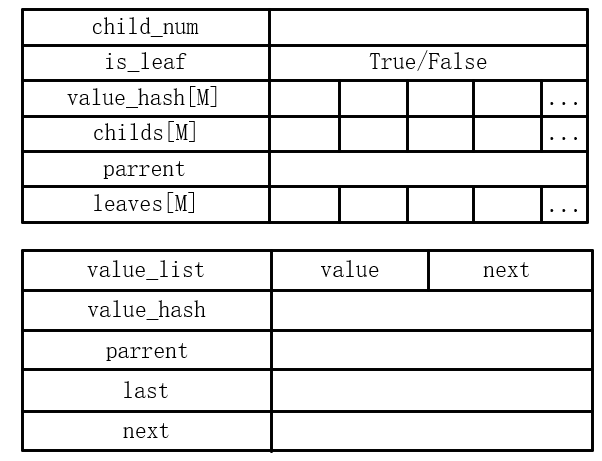
struct Node\* parrent; // 指向父结点

LeafPtr leaves[M]; // 指向叶子结点

struct Node\* next;

}\*NodePtr;

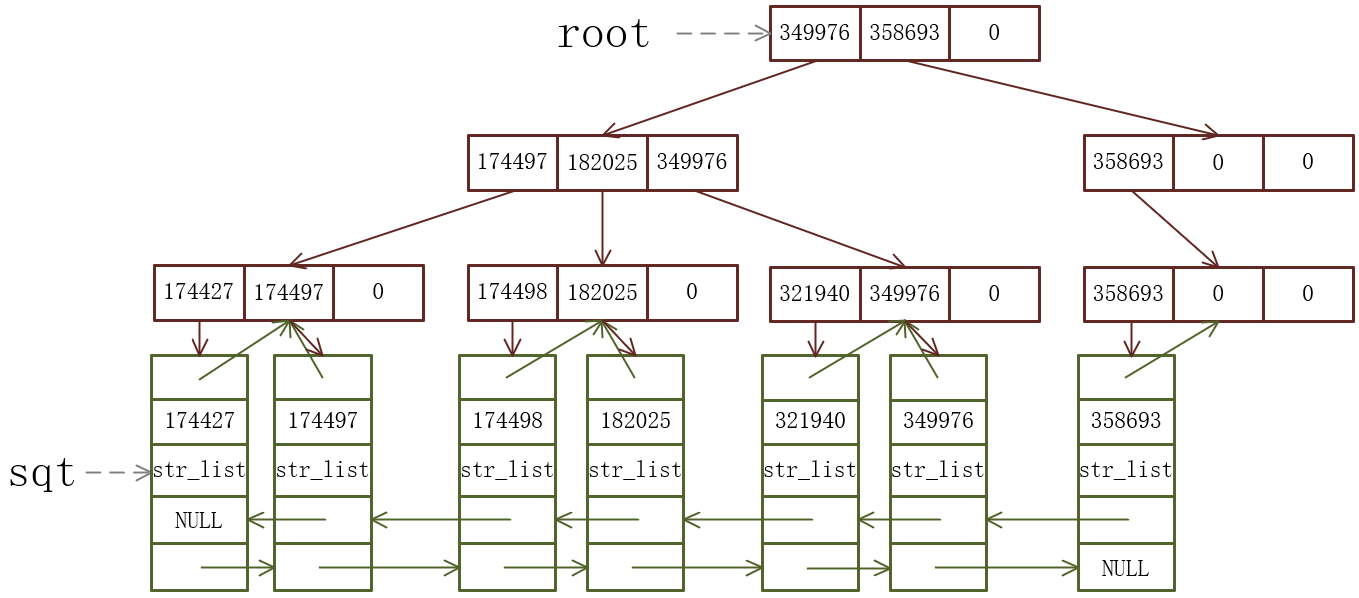
Node结点结构和Leaf结点结构如下：



上图中的child\_num记录该结点中的关键词数目，其≤M；is\_leaf记录该结点是否为叶子结点，如果是，则孩子存于leaves[M]数组中，否则存于childs[M]数组中；value\_hash[M]记录关键词，该关键词为对字符串求hash值；childs[M]和parrent指向Node结点，leaves[M]指向Leaf结点。

Leaf结点为真正的叶子，存有字符串链表和对应的hash值，相同hash值的字符串位于同一个Leaf结点中，parrent指向Node结点，last指向上一个Leaf结点，next指向下一个Leaf结点，从而形成一个顺序拉链表。

B+树的结构如下（示例假设M=3）：



每个树结点中都存有pattern的hash值，作为关键字，当索引到叶子结点之后，叶子结点指向真正的叶子，在真正的叶子中存有数据，其中hash相同的字符串在同一片叶子的链表结构中，所有真正的叶子自己又构成一个链表结构，来支持顺序查找。

本程序先读取patterns文件，获取数据，并构建为M阶B+树，并对内存的使用动态分配并记录。然后进行检索工作：遍历words数据文件，读取单词，并从patternsB+树中通过root查询，是否存在与之相等的字符串，如果检索到相同的字符串，则返回并将检索成功结果写入文件；如果检索结束仍然未找到相同字符串，则返回并将检索失败结果写入文件。最后将统计信息写入文件最后一行。

### 1.3 版本三radix4\_search

主要数据结构如下：

typedef struct ValueList {

char\* value;

struct ValueList\* next;

}\*ValueListPtr;

typedef struct Node {

ValueListPtr value\_list;

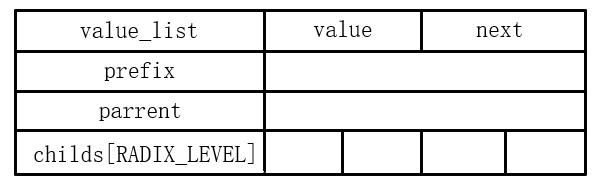
unsigned int prefix;

struct Node\* parent;

struct Node\* childs[RADIX\_LEVEL]; // RADIX\_LEVEL为4

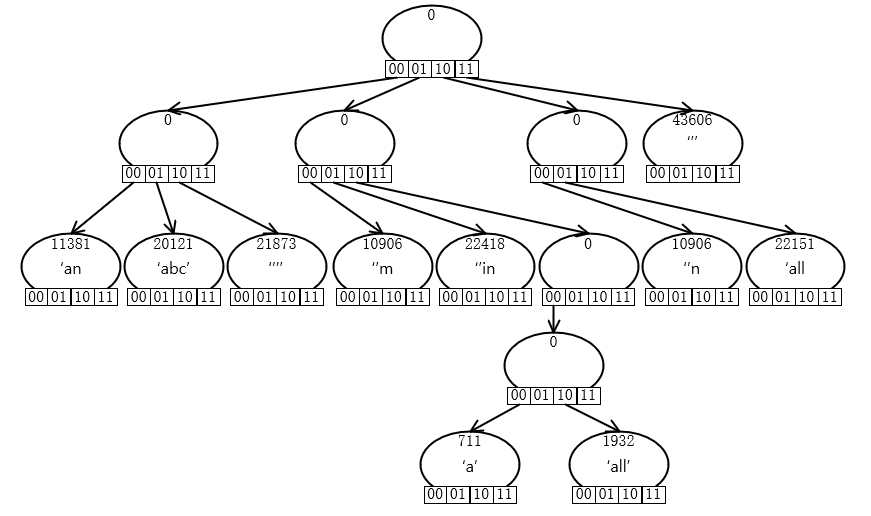
}\*NodePtr;

Node结点结构和Leaf结点结构如下：



上图的value\_list存此结点下满足prefix前缀的模式串，prefix为前缀，parrent指向父结点node，childs为该结点的四个孩子，默认为NULL，在需要放置模式串时才生成子node结点并连接父子关系。

四阶基数树的结构如下（此为插入10个结点后的树结构）：



上述结点中每个结点的第一行数据是该结点的前缀，第二行为value\_list，当其为NULL，表示该结点下无数据（如果模式串在某一层的prefix为0时，将模式串存于该层结点）；

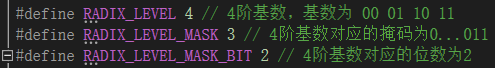
在存储过程中采用的是压缩后的树结构，仅当出现相同关键字前缀的模式串时，才分裂结点，生成前缀结点，并将数据存于子结点中。

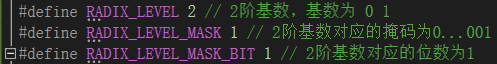
本程序先读取patterns文件，获取数据，存于Node结构的基数树结点中，其中对内存的使用动态分配并记录。然后进行检索工作：遍历words数据文件，读取单词，并从patterns基数树中四分查找是否存在与之相等的字符串，如果检索到相同的字符串，则返回并将检索成功结果写入文件；如果检索结束仍然未找到相同字符串，则返回并将检索失败结果写入文件。最后将统计信息写入文件最后一行。

### 1.4 版本四patricia\_search

在patricia\_search中，只是将radix4\_search中的3个设定更改：基数阶数4改为2，基数掩码3改为1，移位操作位数2改为1，其余设定均相同。

如下图即为radix4\_search和patricia\_search的设定。





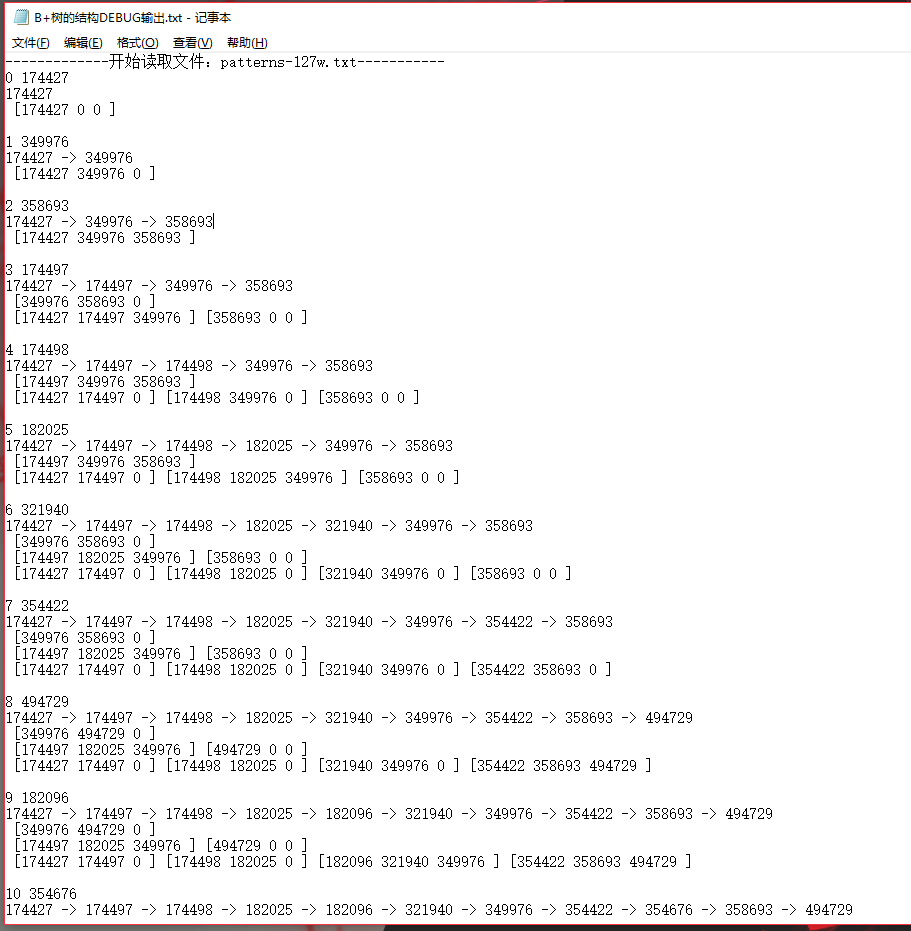
即本程序所实现的基数树操作，只需要更改上述三个对应的设定，即可实现任意阶基数树的操作。由于所取hash范围为0~50w，所有字符串的关键字均在第19位内，即数的深度不超过19层（4阶基数树不超过10层）。设定参数时需要考量实用性，否则将导致查询效率低下。

由于过程中采用压缩方式，当出现相同前缀才继续分裂结点，故实际内存占用量会比完全基数树小很多。

## 2. 实验过程

在实验时，借助第一次作业的实验过程继承文件的读取操作。同时继承上次作业的generate\_hash.c文件用来提供hash函数和bupt.c文件用来提供公用操作。

之后陆续编写了btree\_search.c、bplus\_search.c进行运行测试。在bplus\_search编写过程中耗时较长，主要在思考树结构如何建立以及中文字符如何存储，同时B+树的实现细节需要琢磨并细心，很容易出现思考不周的bug，最后通过输出树结构和顺序链表结构来一步步查错和调整，终于完成（血泪史啊！！！）。如下图即为对应输出（放大看）。



上图输出第一行为编号和字符串的hash关键字，第二行为顺序表输出，第三行为树结构输出，每层树占据一行。

完成后，继续编写radix4\_search.c，耗时也比较长，主要为思考如何实现普适的基数树，并着手实现，实现完成后出现了一个释放空间阶段的bug，这个bug的修复时间消耗比较长。详情见章节三的问题1。

之后由于radix4\_search的实现是普适结构，故patricia\_search只需要更改radix4\_search中的设定即可实现。过程顺利。

# 三、遇到的问题

## 1. 在radix4\_search中，分裂结点bug

在插入操作中

函数逻辑如下：

rt\_insert(data, prefix, node)

{

if (前缀和本结点前缀相同) {

数据写入本结点

return;

}

if (前缀 == 0) {

if (本结点无数据) {

数据写入本结点

return;

}

else{

本结点数据下移到子结点

数据写入本结点

return; ........这里忘了return，

..........................导致写入本结点的数据又被写入了子结点（进入下面代码中的if），

..........................两个地方的指针(本结点和某个子结点)指向相同的数据，

..........................导致在释放空间的时候，试图释放已经释放掉的空间，就炸了。

}

}

// 到此说明前缀和本结点不同且不为0

数据进行递归插入操作(rt\_insert(data, new\_prefix, child\_node))

if(本结点存在数据){

本结点数据进行子结点插入递归操作(rt\_insert(node\_data, new\_node\_prefix, child\_node)

}

return;

}

Bug表现为：读取pattern文件创建树结构过程正常，读取words文件查询操作正常，且结果正确；在最后释放空间destory树结构的时候，出错程序崩掉。

# 四、结果指标

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **版本** | **树结点个数** | **比较次数(K)** | **内存使用（kb）** | **单词数/匹配数** | **单词**  **匹配数** | **运行时间** |
| **btree\_search** | 460030 | 7335/29988 | 33742 | 985996/1270688 | 516816 | 4s |
| **bplus\_search** | 116716/460030 | 7335/31247 | 43343 | 985996/1270688 | 516816 | 5s |
| **radix4\_search** | 488196 | 7373/17378 | 36309 | 985996/1270688 | 516816 | 3s |
| **patricia\_search** | 600687 | 7354/25781 | 34692 | 985996/1270688 | 516816 | 4s |

上述比较次数，前者为不考虑中间结点的比较次数，后者为考虑中间结点的比较次数。

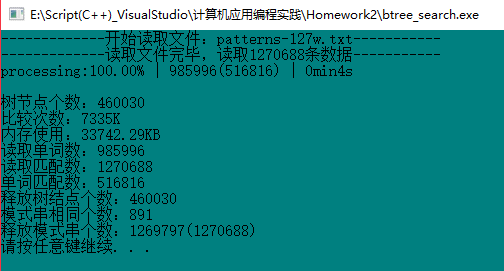
其中btree\_search中对hash范围取0~50w，采用的是BDKRHash函数；

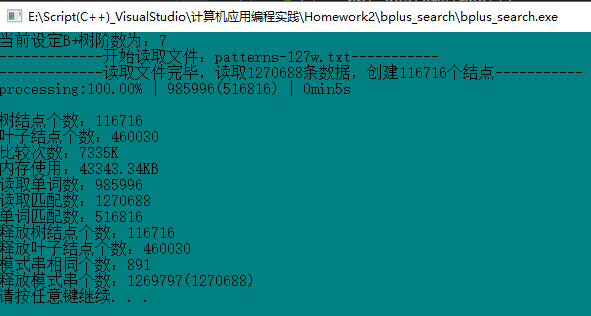
在bplus\_search中树结点个数设定阶数M为7，hash范围取0~50w，采用的是BDKRHash函数，同样得到与btree\_search中相同的460030个叶子结点个数，树节点个数为116716个。

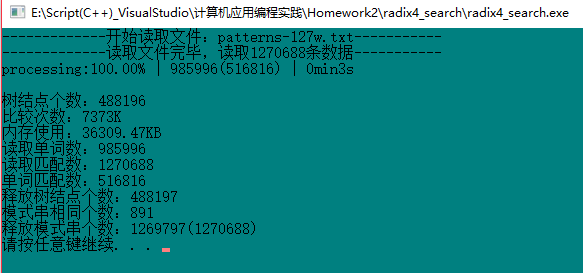
在radix4\_search中，设定hash范围为0~50w，采用的是BDKRHash函数。

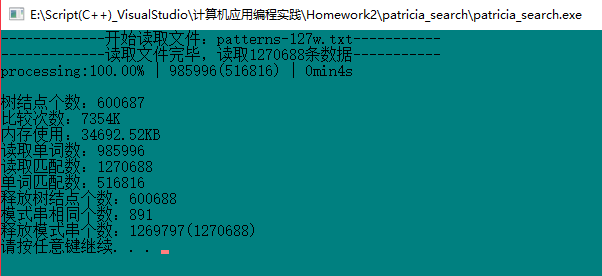
在patricia\_search中，只是将radix4\_search中的3个设定更改：基数阶数4改为2，基数掩码3改为1，移位操作位数2改为1，其余设定均相同。

上述结果运行输出如下：









# 五、结论和总结

通过本次实验，复习并巩固了树结构和对树结构的算法操作。虽然以前已经完成过，不过重新实现一遍，还是有些难度的。实现过程中，我对树的理解加深了不少。