

当前位置: 首页>> 数据结构与算法>> 阅读全文

11-04 数据结构之Trie树
10 Category: 数据结构与算法 View: 2,042 阅 Author: Dong

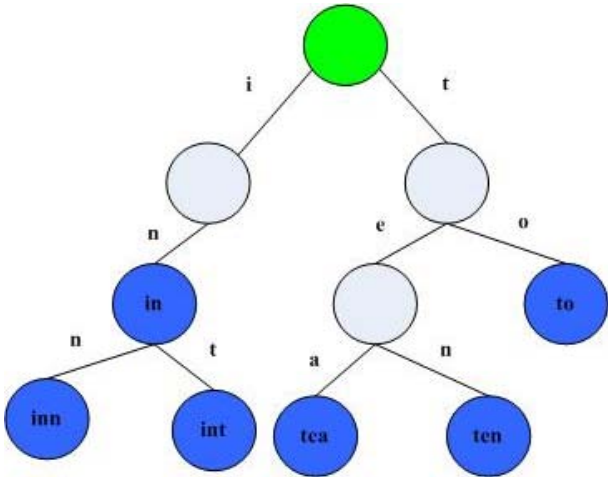
作者: Dong | 可以转载, 但必须以超链接形式标明文章原始出处和作者信息及版权声明
网址: <http://dongxicheng.org/structure/trietree/>

1、概述

Trie树，又称字典树，单词查找树或者前缀树，是一种用于快速检索的多叉树结构，如英文字母的字典树是一个26叉树，数字的字典树是一个10叉树。

Trie一词来自retrieve，发音为/tri:/ “tree”，也有人读为/traɪ/ “try”。

Trie树可以利用字符串的公共前缀来节约存储空间。如下图所示，该trie树用10个节点保存了6个字符串tea, ten, to, in, inn, int:



在该trie树中，字符串in, inn和int的公共前缀是“in”，因此可以只存储一份“in”以节省空间。当然，如果系统中存在大量字符串且这些字符串基本没有公共前缀，则相应的trie树将非常消耗内存，这也是trie树的一个缺点。

Trie树的基本性质可以归纳为：

- (1) 根节点不包含字符，除根节点意外每个节点只包含一个字符。
- (2) 从根节点到某一个节点，路径上经过的字符连接起来，为该节点对应的字符串。
- (3) 每个节点的所有子节点包含的字符串不相同。

2、Trie树的基本实现

字母树的插入（Insert）、删除（Delete）和查找（Find）都非常简单，用一个一重循环即可，即第i次循环找到前i个字母所对应的子树，然后进行相应的操作。实现这棵字母树，我们用最常见的数组保存（静态开辟内存）即可，当然也可以开动态的指针类型（动态开辟内存）。至于结点对儿子的指向，一般有三种方法：

- 1、对每个结点开一个字母集大小的数组，对应的下标是儿子所表示的字母，内容则是这个儿子对应在大数组上的位置，即标号；
- 2、对每个结点挂一个链表，按一定顺序记录每个儿子是谁；
- 3、使用左儿子右兄弟表示法记录这棵树。

三种方法，各有特点。第一种易实现，但实际的空间要求较大；第二种，较易实现，空间要求相对较小，但比较费时；第三种，空间要求最小，但相对费时且不易写。

下面给出动态开辟内存的实现：

```
1 #define MAX_NUM 26
2 enum NODE_TYPE{ // "COMPLETED" means a string is generated so far.
3     COMPLETED,
```

订阅



日志分类

- 下一代MapReduce(YARN/MRv2) (7)
- Hadoop-MapReduce (32)
- NoSQL数据库 (9)
- 大规模数据处理 (1)
- 网络编程 (1)
- Linux操作系统 (1)
- 搜索引擎 (8)
- 数据挖掘 (2)
- 集群管理 (1)
- C/C++语言 (16)
- 脚本语言 (2)
- 数据结构与算法 (19)
- 智力题 (2)
- 未分类 (1)

标签云

热门文章

- 📖 开源日志系统比较 - 8,062 阅
- 📖 Thrift使用指南 - 7,084 阅
- 📖 Hadoop Streaming 编程 - 6,656 阅
- 📖 Thrift框架介绍 - 6,391 阅
- 📖 Hadoop-0.20.2公平调度器算法解析 -
- 📖 非阻塞connect编写方法介绍 - 5,158 阅
- 📖 HDFS小文件问题及解决方案 - 4,669 阅
- 📖 数据结构与算法汇总 - 4,228 阅
- 📖 使用Thrift RPC编写程序 - 4,057 阅
- 📖 如何在Hadoop上编写MapReduce程序 -

最新日志

- 📖 统一资源管理与调度平台（系统）介绍
- 📖 Hadoop MapReduce容错性分析
- 📖 Hadoop中Speculative Task调度策略
- Hadoop Kerberos安全机制介绍

```
4      UNCOMPLETED
5  };
6  struct Node {
7      enum NODE_TYPE type;
8      char ch;
9      struct Node* child[MAX_NUM]; //26-tree->a, b ,c, .....z
10 };
11
12 struct Node* ROOT; //tree root
13
14 struct Node* createNewNode(char ch){
15     // create a new node
16     struct Node *new_node = (struct Node*)malloc(sizeof(struct Node));
17     new_node->ch = ch;
18     new_node->type == UNCOMPLETED;
19     int i;
20     for(i = 0; i < MAX_NUM; i++){
21         new_node->child[i] = NULL;
22     }
23     return new_node;
24 }
25
26 void initialization() {
27     //intiaization: creat an empty tree, with only a ROOT
28     ROOT = createNewNode(' ');
29 }
30
31 int charToIndex(char ch) { //a "char" maps to an index<br>
32     return ch - 'a';
33 }
34
35 int find(const char chars[], int len) {
36     struct Node* ptr = ROOT;
37     int i = 0;
38     while(i < len) {
39         if(ptr->child[charToIndex(chars[i])] == NULL) {
40             break;
41         }
42         ptr = ptr->child[charToIndex(chars[i])];
43         i++;
44     }
45     return (i == len) && (ptr->type == COMPLETED);
46 }
47
48 void insert(const char chars[], int len) {
49     struct Node* ptr = ROOT;
50     int i;
51     for(i = 0; i < len; i++) {
52         if(ptr->child[charToIndex(chars[i])] == NULL) {
53             ptr->child[charToIndex(chars[i])] = createNewNode(chars[i]);
54         }
55         ptr = ptr->child[charToIndex(chars[i])];
56     }
57     ptr->type = COMPLETED;
```

3、Trie树的高级实现

可以采用双数组（Double-Array）实现。利用双数组可以大大减小内存使用量，具体实现细节见参考资料（5）

（6）。

4、Trie树的应用

Trie是一种非常简单高效的数据结构，但有大量的应用实例。

（1）字符串检索

事先将已知的一些字符串（字典）的有关信息保存到trie树里，查找另外一些未知字符串是否出现过或者出现频率。

举例：

@ 给出N 个单词组成的熟词表，以及一篇全用小写英文书写的文章，请你按最早出现的顺序写出所有不在熟词表中的生词。

@ 给出一个词典，其中的单词为不良单词。单词均为小写字母。再给出一段文本，文本的每一行也由小写字母构成。判断文本中是否含有任何不良单词。例如，若rob是不良单词，那么文本problem含有不良单词。

（2）字符串最长公共前缀

Trie树利用多个字符串的公共前缀来节省存储空间，反之，当我们把大量字符串存储到一棵trie树上时，我们可以快速得到某些字符串的公共前缀。

举例：

@ 给出N 个小写英文字母串，以及Q 个询问，即询问某两个串的最长公共前缀的长度是多少？

解决方案：首先对所有的串建立其对应的字母树。此时发现，对于两个串的最长公共前缀的长度即它们所在结点的公共祖先个数，于是，问题就转化为了离线（Offline）的最近公共祖先（Least Common Ancestor，简称LCA）问题。

而最近公共祖先问题同样是一个经典问题，可以用下面几种方法：

- 1. 利用并查集（Disjoint Set），可以采用经典的Tarjan 算法；
- 2. 求出字母树的欧拉序列（Euler Sequence）后，就可以转为经典的最小值查询（Range Minimum Query，简



Hadoop安全机制介绍

运行Hadoop作业时一处常见错误以及解

调整心跳间隔增加Hadoop小集群吞吐率

Hadoop 升级创建硬链接效率优化

YARN/MRv2 ResourceManager代码分

YARN/MRv2 ResourceManager代码结

最新评论

大雄:谢谢你

三江小渡:new hadoop，看到优化神马的字眼特别开心，所以先来留

Dong:恩，Tearasort中自己实现的采样算法不一定高效，注意这里的

Ekans:Map阶段的排序都是Partition在Merge过程中完成的，

Dong:仿真只是实验环境的结果，真实环境下应采用真正的Hadoop作业。

大雄:请教一下，如何测试自己编写的hadoop调度器的性能好坏，用什么

chegvra:2 “如果一个pool的minShare>weight”，

chegvra:1 上面pool3的图是否画错了，因为demand<R*w

左邻右舍

张晓林的博客	台湾hadoop论坛
starfish	程序员面试题狂想曲
张士军的博客	大陆hadoop论坛
nosqlfan	结构之法_算法之道
Alex的个人Blog	david.org的博客

文章归档

▼2012 (12)

►三月 (5)

►二月 (7)

►2011 (90)

称RMQ)问题了;
(关于并查集, Tarjan算法, RMQ问题, 网上有很多资料。)
(3) 排序
Trie树是一棵多叉树, 只要先序遍历整棵树, 输出相应的字符串便是按字典序排序的结果。
举例:
@ 给你N 个互不相同的仅由一个单词构成的英文名, 让你将它们按字典序从小到大排序输出。
(4) 作为其他数据结构和算法的辅助结构
如后缀树, AC自动机等
5、Trie树复杂度分析
(1) 插入、查找的时间复杂度均为O(N), 其中N为字符串长度。
(2) 空间复杂度是26^n级别的, 非常庞大(可采用双数组实现改善)。
6、总结
Trie树是一种非常重要的数据结构, 它在信息检索, 字符串匹配等领域有广泛的应用, 同时, 它也是很多算法和复杂数据结构的基础, 如后缀树, AC自动机等, 因此, 掌握Trie树这种数据结构, 对于一名IT人员, 显得非常基础且必要!

7、参考资料

(1) wiki: <http://en.wikipedia.org/wiki/Trie>

(2) 博文《字典树的简介及实现》:
<http://hi.baidu.com/luyade1987/blog/item/2667811631106657f2de320a.html>

(3) 论文《浅析字母树在信息学竞赛中的应用》

(4) 论文《Trie图的构建、活用与改进》

(5) 博文《An Implementation of Double-Array Trie》:
<http://linux.thai.net/~thep/datrie/datrie.html>


(6) 论文《An Efficient Implementation of Trie Structures》:
[http://www.google.com.hk/url?](http://www.google.com.hk/url?sa=t&source=web&cd=4&ved=0CDEQFjAD&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.14.8665%26rep%3Drep1%26type=eWFlty_fQ&sig2=xfqSGYHBKqOLXjdONIQNVw)

sa=t&source=web&cd=4&ved=0CDEQFjAD&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.14.8665%26rep%3Drep1%26type=eWFlty_fQ&sig2=xfqSGYHBKqOLXjdONIQNVw


更多关于数据结构和算法的介绍, 请查看: [数据结构与算法汇总](#)

原创文章, 转载请注明: 转载自董的博客
本文链接地址: <http://dongxicheng.org/structure/trietree/>

分享到:  QQ空间  新浪微博  腾讯微博  人人网  开心网  更多

 [trie树](#), [前缀树](#), [单词查找树](#), [字典树](#)

相关日志

 [怎样从10亿查询词找出出现频率最高的10个](#)

发表评论 发起引用

评论 (0) 引用通告 (0)

目前还没有任何评论.

发表评论

昵称 (必填)

电子邮箱 (我们会为您保密) (必填)

网址

订阅评论