实验 9 · 高效 IP 路由查找实验

吴嘉皓
2015K8009915007

一、实验内容

1. 实现最基本的前缀树查找；
2. 实现多 bit 前缀树及优化：
   叶推、 压缩指针（提交时还未实现）、 压缩向量（提交时还未实现）；
3. 测试上述实现的正确性；

二、实验流程

（一）代码目录

```
2015K8009915007_吴嘉皓_09.tar.gz
└── 09-lookup
    ├── forwarding-table.txt
    ├── gen_test_txt.py
    ├── include
    │   ├── multibit_trietree.h
    │   ├── types.h
    │   ├── unarybit_trietree.h
    │   └── utils.h
    ├── multibit_trietree_test.c
    ├── run.sh
    ├── unary_trietree_test.c
    └── 实验 9-高效 ip 路由查找-实验报告.pdf
```

（二）实验流程

1. 在 09-lookup 目录下输入如下命令：

```
chmod +x run.sh
./run.sh
```

脚本 run.sh 中会编译并单比特树和多比特树的测试代码：

(multibit_trietree_test.c、unarybit_trietree_test.c)

结果会输出二者搜索结果的总占用的时间、总查询条目数以及平均查询时间。

三、实验结果

```
Search Cost:
Unary-bit TrieTree:
Totol record num:       697882
Total time  cost:       633301.000 (μs)
Search      speed:      0.907 (μs) per record.
Multi-bit TrieTree:
Totol record num:       697882
Total time  cost:       459545.000 (μs)
Search      speed:      0.658 (μs) per record.
```

图 1unarybit 和 multibit 测试结果

四、结果分析

（一）结果分析

由图 1 可知，multibit trietree 的查询时间要比 unarybit trietree 的查询时间要短，前者的时间是后者的 72%左右；因为，前者在查询时间上的的性能要比后者好。

（二）代码实现分析

⇨ 叶推代码分析

伪代码如下：

```
LeafPush(TrieTree T, TrieNode N) begin
   Choose the right node to push as Np;
    if(all childPtr of T is null) return ;
    elseif(all childPtr of T is full) begin
        for all childPtr of T begin
        LeafPush(childPtr, Np);
        end
    end
    else begin
        for all childPtr of T which is null
            LeafPush(childPtr, Np);
    end
end
```

实现代码如下：

```
/// Leaf Push
void LeafPush(TrieTree *T, TrieNode *pushed_node){
    // if current Ptr is null,
    // that means it reaches the tail of the tree
    if(!(*T)) return ;
    // Choose the right node to be pushed
    TrieNode tmp, * pushing_node = NULL;
    if((*T)->node_type == INTERNAL)
        pushing_node = pushed_node;
    else if((*T)->node_type == LEAF){
        if(!all_childs_is_null(*T)){
            pushing_node = *T;  // current 'LEAF' node is going to be pushed
            *T = init_new_node(); // generate a new 'INTERNAL' node
            // modify the corresponding variables of new 'INTERNAL' node
            (*T)->info.prefix_len = pushing_node->info.prefix_len;
            for(int i = 0; i < CHILD_NUM; i++){
                (*T)->childs[i] = pushing_node->childs[i];
                pushing_node->childs[i] = NULL;
            }
            memcpy(&tmp, pushing_node, sizeof(tmp));
            free(pushing_node);
            pushing_node = &tmp;
        }
    }
    else{
        printf("ERROR: The current node type does not exits.\n");
        exit(-1);
    }
    // Start pushing
    if(all_childs_is_null(*T))
        return ;
    else if(all_childs_is_full(*T)){
        for(int i = 0; i < CHILD_NUM; i++)
            LeafPush(&((*T)->childs[i]), pushing_node);
    }
    else{
        for(int i = 0; i < CHILD_NUM; i++)
            if((*T)->childs[i])
                LeafPush(&((*T)->childs[i]), pushing_node);
            else{
                TrieNode * pushing_node_bk = init_new_node();
                memcpy(pushing_node_bk, pushing_node, sizeof(TrieNode));
                (*T)->childs[i] = pushing_node_bk;
            }
    }
}
```

其他部分代码实现很常规，就不分析了。
叶推的代码想了很久才实现。

（三）实验感想

在此次实验中，实现代码还是很麻烦的，但是我在实验中，还是花了很大一部分时间来写测试代码。最终还是用 forwarding-table.txt 来建树，并且用它来进行查询的操作。但问题是，我并没有一个很好的标准来比较完备地检验我的算法是否正确。希望之后老师可以给一个标准输入，以及一个比对结果，从而能够准确地检验代码的准确性。