EL 7373 High Performance Switches & Routers

Lab 1 Report

Group: **G16**

Yu chen yc1595

Yuqin Wang yw1724

Peixuan Ding pd1178

Section 1

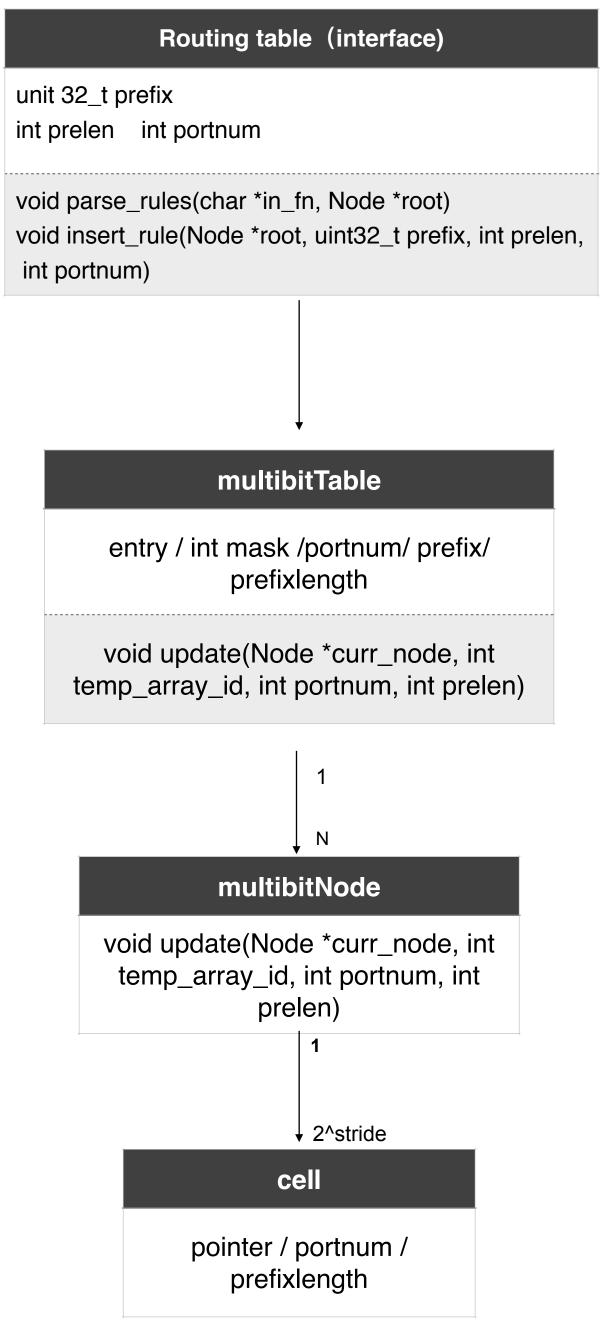
1.1 Algorithms

Multi Bit Trie with Leaf Pushing

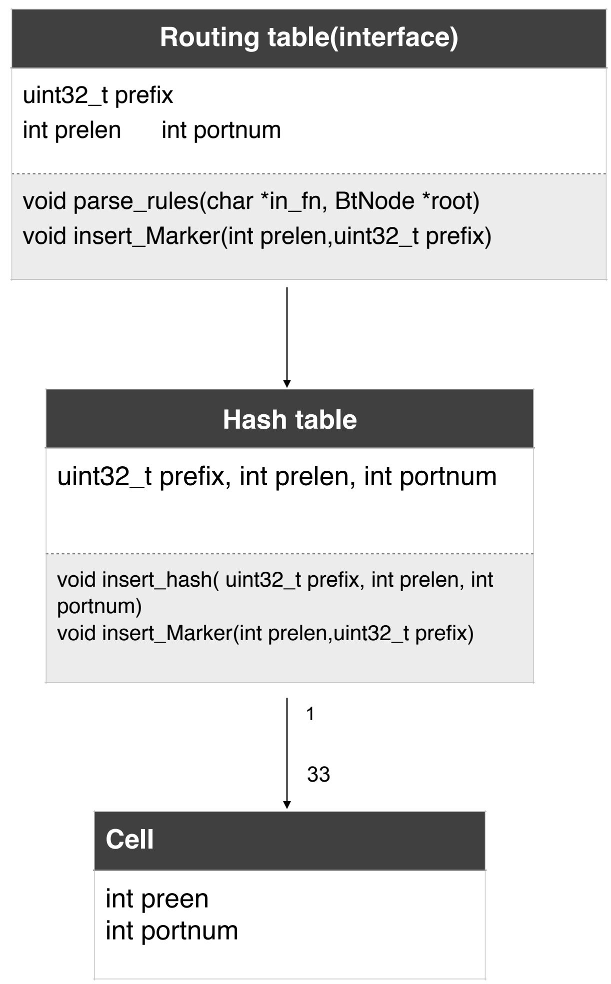
Binary Search on Prefix Lengths

1.2 data structure

Multi Bit Trie with Leaf Pushing

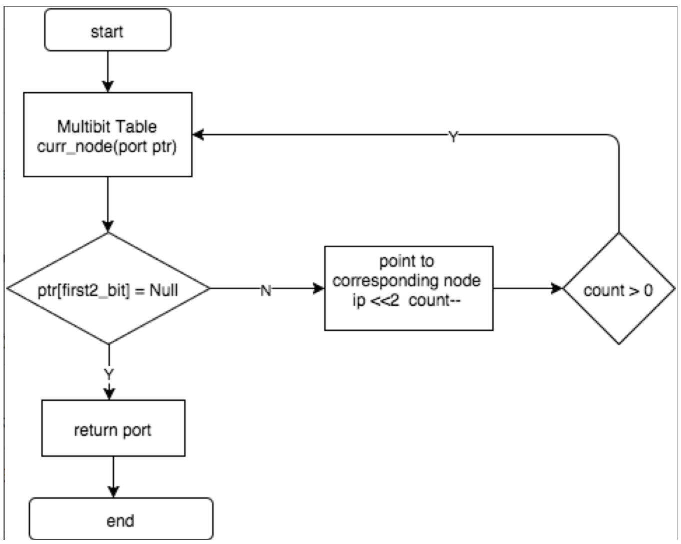


Binary Search on Prefix Lengths

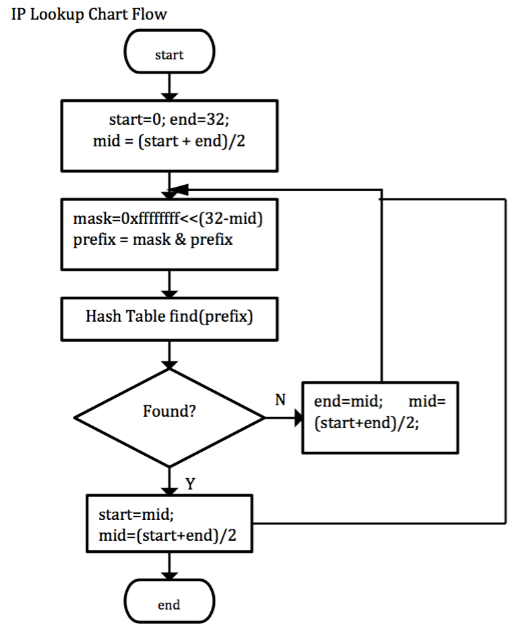


1.3 Ip Lookup

Multi Bit Trie with Leaf Pushing

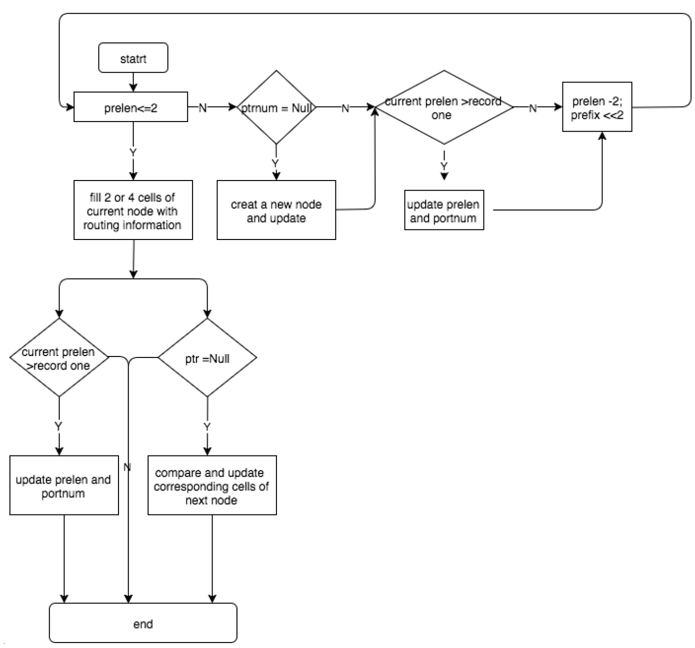


Binary Search on Prefix Lengths



1.4 Insert rule

Multi Bit Trie with Leaf Pushing

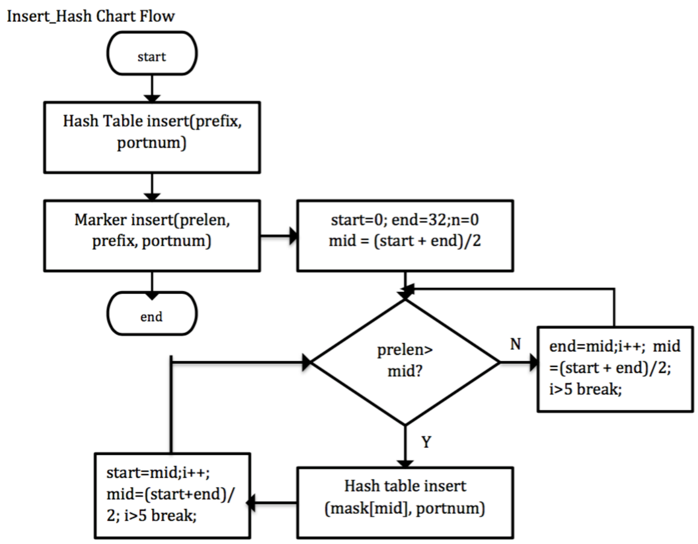


1.We use every entry from Routing Table to generate Mitlbit nodes which contain pointer, portnum and prefixlength(use it as a judge to update nodes) .

2 According to longest prefix match, we should update the prefixlength when the current prefixlength is longer than former one.

3 To implement leaf pushing, when current length is on longer than stride ,we should compare the portnum with those in next node and update them.

Binary Search on Prefix Lengths



1.According to Routing Table, generate hashtable using unordered_map

2.In order to obtain correct results. we add some markers when we can't find IP on specific length after searching the hash Table.

section 2 Complexity analysis

2.1 Multi Bit Trie with Leaf Pushing

Assuming N is the number of entry that the table stores, W is the Maximum length of the prefix,K is the stride.

Memory Usage Complexity:in the worst case, length of each entry is W. So there are W/K nodes. There are N entrys and each node is 2^K. it is O(2^K * W *N/K).Since leaf pushing can reduce momery of each node to half .so the final memory complexity is O(2^K * W *N/(2*K)).

Memory Access Complexity: in the worst case . we need to compare W/K times for ip lookup. In this code. It shoud be 32/2 = 16(we set stride equals to 2).

2.2 Binary Search on Prefix Lengths

Memory Usage Complexity: Because only logW markers would be probed instead of W markers, the storage complexity is O(N logW)

Memory Access Complexity: Taking no account of the hash collision, it should be O(logW) in the worst case, which is 5.

2.3 compare with binary trie

We assume W is the maximum prefix length. N is the number of entry that the table stores.

	storage complexity	**lookup complexity**	**update**
Binary Trie	O(NW)	O(W)	O(W)
Multi Bit Trie with Leaf Pushing	O(2^k *W*N/k)	O(W/k)	O(W/k+2^k)
Binary Search on Prefix Lengths	O(N*logW)	O(logW)	O(logW)

Binary Trie(BT) compare each bit once. In the worst case it compares W times, thus the lookup complexity is O (W). For memory storage, in worst case every entry length is W. So the memory usage is O(WN) .

Multi Bit Trie with Leaf Pushing compare k bits once, in the worst case it has to compare [W/k] times, so the lookup complexity is OW/k) .For memory usage, in the worst case each entry is W length, so we get W/k nodes for each entry and total (W/k) * N entry. None of entry shares any node and each node is 2^k, so the memory complexity is O(2^k *W*N/k). Compare to BT, multi bit trie with leaf pushing speed up the lookup but require a large memory spaces.

Binary Search on Prefix Lengths compare each bit once, in the worst cast it has to compare logW times, thus the ip lookup complexity is O(logW). For memory storage, in the worst case each entry is W length, so for total entry is N*logW, and the memory complexity is O(N*logW). Compare to the Binary Trie, we conclude that binary search on prefix lengths is better than BT on time complexity and memory complexity.

Section 3

	Yuqin Wang	Peixuan Ding	Yu Chen
Multi Bit Trie with Leaf Pushing code	√	√	√
Binary Search on Prefix Lengths code	√	√	√
debug	√	√	√
Complexity Analysis	√	√	
Leaf Pushing Function		√	√
add marker function	√		√
Lab report	√	√	√