以下的報告是根據助教提供的任務描述而寫的。引述助教為
“
Hw2範例.docx檔是該作業的繳交格式範例
要做的事情有：
1.用自己的話敘述Binary Trie / Fourbit Trie比較
2.使用我們提供的程式碼去跑binary_trie.c 4_bit_trie.c
並分析自己實作的結果
3.用自己的話敘述leaf pushing algorithm
4.使用我們提供的程式碼去跑leaf_push.c
(執行的部分可以參考範例檔案的指令)
”
請注意：第二部提到的 ‘自己的實作’ 我解讀為我們需要實作binary trie和 fourbit trie. 我的 github repository為https://github.com/felixweiss1999/trie_performance。我希望這個報告滿足您的要求！

1. **Binary Trie / Fourbit Trie**比較

Binary Trie / Fourbit Trie都在最長前綴匹配中是兩種常用的資料結構。

Binary Trie：
1. 結構：二元搜尋樹，每個節點最多有兩個子節點：左子節點和右子節點。
2. 內存佔用：每個節點需要存儲一個二進制位，因此在內存方面佔用較少。

Fourbit Trie：
1. 結構：是一種特殊的前綴樹，每個節點代表一個四位二進制數（0-15），並且可以具有多個子節點。
2. 內存佔用：每個節點需要存儲四個二進制位，因此在內存方面佔用相對較多。

最長前綴匹配是通過遍歷樹結構來查尋與輸入IP地址的最長前綴匹配項。選擇哪種資料結構取決於具體的應用需求和性能要求。

2. 跑**binary_trie.c 4_bit_trie.c**並分析自己實作的結果
跑binary_trie.c的輸出：

```
# of prefix = 997952
# of prefix is len < 16 = 4409
# of prefix is len = 16 = 14910
max_segment_size =1852, index = 48771
empty_segment =38053 , non_empty_segment =27482
number of layer0: 32
number of sum: 0
```

跑4_bit_trie.c的輸出:

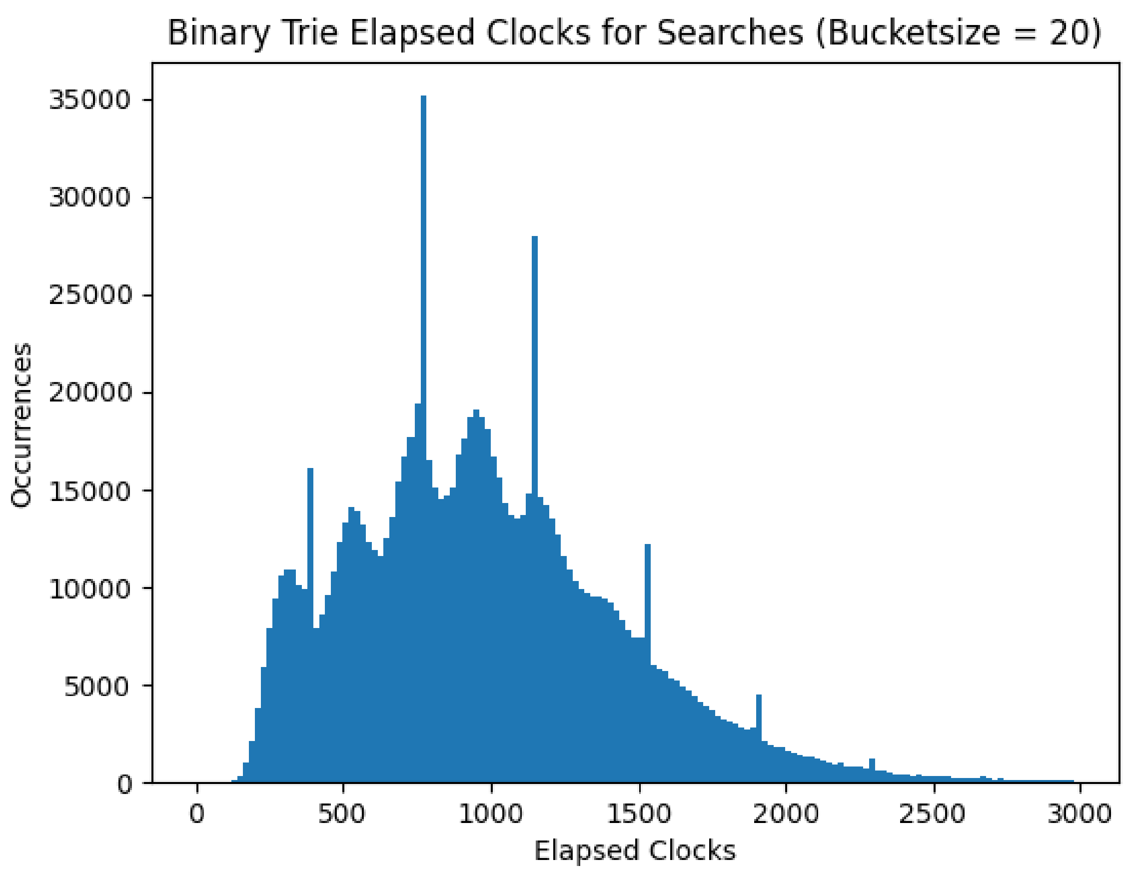
```
Avg. build: 424
number of nodes: 1096379
Total memory requirement: 69594 KB
Avg. Search: 31
(MaxClock, MinClock) = (  873,     1)
907157
45828
27897
13126
3079
757
89
17
2
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
memory access time table:
1 times = 0
2 times = 0
3 times = 1
4 times = 189
5 times = 11873
6 times = 89968
7 times = 861276
8 times = 34645
9 times = 0
10 times = 0
11 times = 0
12 times = 0
13 times = 0
14 times = 0
15 times = 0
16 times = 0
17 times = 0
18 times = 0
19 times = 0
20 times = 0
21 times = 0
22 times = 0
23 times = 0
24 times = 0
25 times = 0
26 times = 0
27 times = 0
28 times = 0
29 times = 0
30 times = 0
31 times = 0
avg. memory access time:6.920197
Avg. insert time:441
number of nodes after insert: 1192284
```

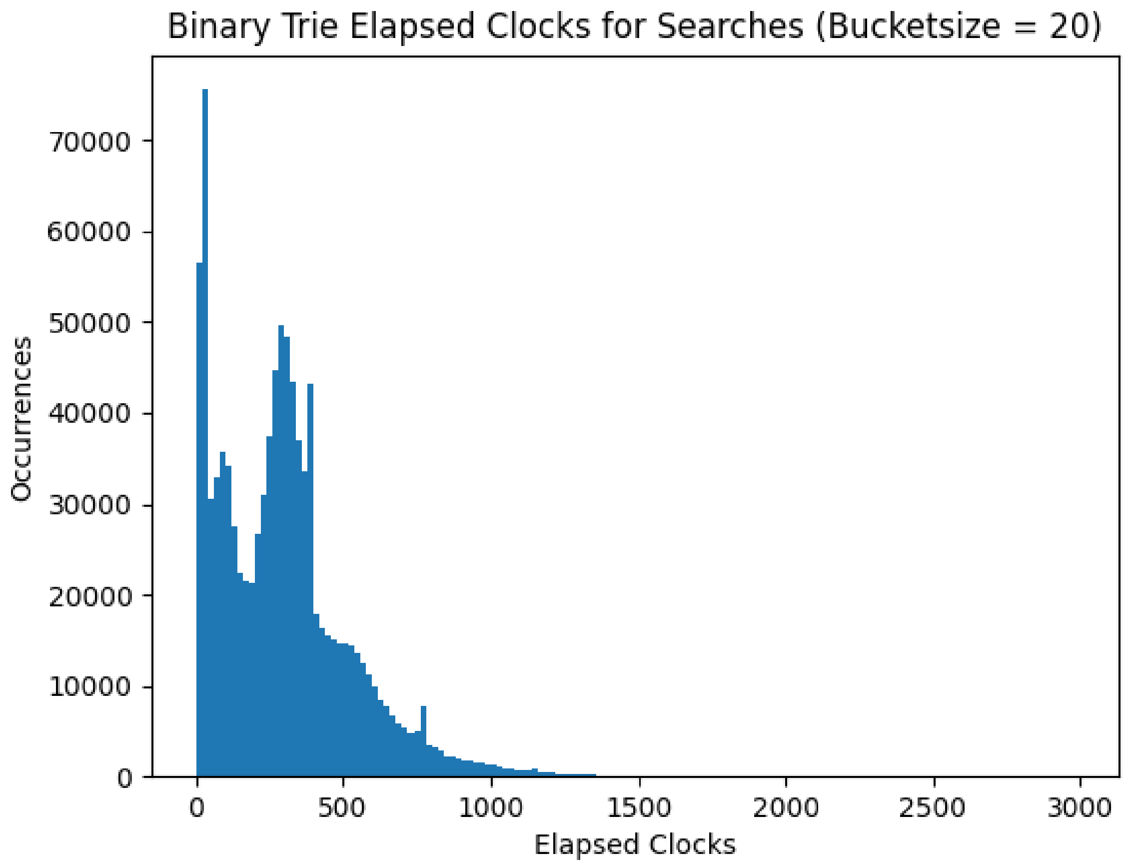
自己的實作結果（Build: ipv4_rrc_all_90build.txt, Insert: ipv4_rrc_all10insert.txt, Search: ipv4_rrc_all_90build.txt）Executed on Windows. MaxClock, MinClock measure search times.

	Fourbit Trie	**Binary Trie**
Average Build Time	541.483055	1091.195962
Average Insert Time	292.669077	1018.485173
Average Search Time	470.270782	1026.658968
Number Of Nodes (before insert)	1096379	2282296
Total memory requirement	(8 * 16 + 4) * 1096379 = 141330 KB	(8+8+4) * 2282296 = 44576 KB
(MaxClock, MinClock)	(90269, 18)	(57380, 57)

我的評論：
正如預期的那樣，與我實作的binary trie相比，我fourbit trie的實作在建構、插入和搜索的方便掌握明顯的優勢。這是可以預料的，因為fourbit trie的深度應該比binary trie小得多，從而減少了緩慢的記憶體查詢次數。它也具有相對少的節點，雖然其個別節點佔用更多空間，導致其佔用的総空間超過對應的binary trie的三倍以上。我發現測量的MaxClock值在不同運行中變化很大，因此應該對這些極端值持保留態度。我不確定是什麼原因導致了這些極端情況的出現。

個別搜尋時間分佈如下：





3. 敘述Leaf pushing algorithm

葉推算法將binary trie中每個內部節點的前綴搬移到其對應的葉節點，確保所有前綴僅存儲在葉節點中。舉個例子，考慮一個擁有三個前綴的樹：10*、110*和1110*。應用葉推算法後，結果的樹將包含四個前綴：110*、1110*、1111*和01*。前綴01*和1111*從前綴10*繼承了路由資訊。儘管前綴數量增加了，葉推算法下的每個前綴仍然是獨立且互不相交的。

4. **跑1level_push.c的輸出：**

```
Before one-level push have prefix nodes: 997952
After one-level push have prefix nodes: 959645
add space: 16206
Avg. Build Time: 1667
Total memory requirement: 53871 KB
There are 3258148 nodes in binary trie
Avg. Search: 1
(MaxClock, MinClock) = (    1,     1)
997952
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
Avg. insert Time:1402
```