

컴퓨터 네트워크

-11주차(BGP Routing Table 에서 IP주소 탐색)-

학번: 201404377

이름: 진승언

과제 목표(도출해야 할 결과)

BGP Routing Table Data를 읽어 IP주소를 탐색한다. 먼저 제공된 BGP Routing Table Data를 읽어 전처리를 한다. 그 후 전처리된 파일을 읽어 Hash와 Trie 두 가지 알고리즘을 사용하여 input IP 주소 탐색 방법을 구현한다. 마지막으로 두 가지 알고리즘의 성능평가를 위해 그래프로 시각화하여 보여준다.

##시각화를 위해 hash와 trie 모두 하나의 파일에서 구현했습니다.

코드 설명과 과제 해결 방법

```
49 if __name__ == "__main__":
50     # 전처리 시작
51     print("전처리 시작")
52     txt_result = [] # 전처리 결과
53     pickle_result = [{}]
```

before_network = ['', ' ', 0, 0, 0] # 입력데이터가 오름차순으로 되었으나 바로 이전꺼와 비교해줌

```
54 if not os.path.isfile("preProcessingData.pickle"):
55     f = open('oix-full-snapshot-2019-11-28-0600', 'r')
56     print("실행중... 시간이 약 1분정도 소요됩니다")
57     # 필요없는줄 제거
58     for i in range(0, 5):
59         f.readline()
```

먼저 전처리할 파일을 읽어온다.

```
60 line = f.readline()
61 while line:
62     oneline = line.split()
63     network = oneline[1]
64     next_hop = oneline[2]
65     weight = int(oneline[3])
66     loc_prf = int(oneline[4])
67     path = len(oneline) - 7
68
69     # 같은 아이피일 경우 비교 후 하나만 남김 (# Big Weight > High Local Pref > Short AS_PATH 탐색 전처리 기준)
70     if before_network[0] == network:
71         if weight > before_network[2]:
72             before_network = [network, next_hop, weight, loc_prf, path]
73         elif weight == before_network[2] and loc_prf > before_network[3]:
74             before_network = [network, next_hop, weight, loc_prf, path]
75         elif weight == before_network[2] and loc_prf == before_network[3] and path < before_network[4]:
76             before_network = [network, next_hop, weight, loc_prf, path]
77
78     # 다른 아이피
79     else:
80         before_network = [network, next_hop, weight, loc_prf, path]
81
82     # 마지막라인이거나 해당 네트워크에서 가장 우선순위 높은 것으로 판명났을 경우 결과에 추가 해줌
83     nextline = f.readline()
84     if nextline == '':
85         txt_result.append([network, before_network[1]])
86         pickle_result.append(before_network)
87         break
88
89     splitNextline = nextline.split()
90     if splitNextline[1] != network:
91         txt_result.append([network, before_network[1]])
92         pickle_result.append(before_network)
93
94     # cus while line
95     line = nextline
```

파싱을 하고 같은 네트워크 아이피일 경우 **Big Weight > High Local Pref > Short AS_PATH** 기준으로 하나의 아이피만 남기고 리스트에 저장한다.

```

101 # 텍스트파일로 저장 (hash와 trie에 사용하기 좋게 고침)
102 with open('preProcessingData.txt', 'w') as f:
103     for i in txt_result:
104         tmpResult = i[0] + ' ' + i[1] + '\n'
105         f.write(tmpResult)
106
107 # pickle로 저장
108 with open('preProcessingData.pickle', 'wb') as p_file:
109     pickle.dump(pickle_result, p_file)

```

전처리한 ip리스트를 pickle과 txt파일로 저장한다. (txt 파일은 추후 탐색에서 읽어오기 위해 저장했다.)

```

124 f = open('preProcessingData.txt')
125 while True:
126     line = f.readline()
127     line = line[0:len(line) - 1]
128     if not line:
129         break
130     # bit로 변환
131     tmpId = ['', '', '']
132     networkIdMask = line.split()[0]
133     networkId = networkIdMask.split('/')[0]
134     networkSplitId = networkId.split('.')
135     networkMask = networkIdMask.split('/')[1]
136     for i in range(0, len(networkSplitId)):
137         networkSplitId[i] = str("{0:b}".format(int(networkSplitId[i])).zfill(8))
138         tmpId[0] += networkSplitId[i]
139     tmpId[0] = tmpId[0][:int(networkMask)] # bit IP
140     tmpId[1] = networkId # IP
141     tmpId[2] = line.split()[1] # next hop
142     bitIp = tmpId
143
144     value = [bitIp[0], bitIp[2]]
145     index = hash(bitIp[0]) % lineCount
146     # 해시테이블에 데이터 추가
147     if not hashIpTable[index]:
148         hashIpTable[index] = [value]
149     else:
150         hashIpTable[index].append(value)
151
152 # ip 입력파일 생성
153 txtFile = open("100000.txt", "r")
154 while True:
155     line = txtFile.readline()
156     if not line:
157         break
158     inputIpList.append(line)

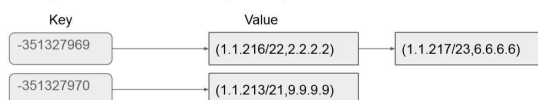
```

전처리한 파일을 불러오고 해시테이블에 저장한다. 그리고 탐색할(입력한) 텍스트파일도 불러와 변수에 저장한다. 100000개의 input값이 들은 파일로 실행했다.

저장하는 방식은 밑의 참고사진과 같다.

Hash Table 구성 과정

- Network: 1.1.216.0/22 -> 00000001.00000001.110110, Next Hop: 2.2.2.2
- hashing Network ID -> -351327969
- Network: 1.1.227.0/23 -> 00000001.00000001.1101100, Next Hop: 6.6.6.6
- hashing Network ID -> -351327969(Collision!)



```

160 print("Hash 탐색 시작")
161 # 해시탐색 시작
162 hashStartTime = time.time()
163 for ipList in inputIpList:
164     mostSameHop = None
165
166     bitIp = '' # a.a.a.a -> 32 bit
167     splittedIP = ipList.split('.')
168
169     for i in range(0, len(splittedIP)):
170         splittedIP[i] = str("{0:b}".format(int(splittedIP[i])).zfill(8))
171         bitIp += splittedIP[i]
172     # 8~32 비트별로 모두 탐색
173     for i in range(0, 33):
174         indexNum = hash(bitIp[0:i]) % lineCount
175         isSearch = hashIpTable[indexNum]
176         if isSearch: # 리스트 순회
177             for kv in isSearch: # [] in value's list
178                 if kv[0] == bitIp[0:i]:
179                     mostSameHop = kv[1]
180                     break
181     tmp = mostSameHop
182     hashingResult.append(tmp)
183     # 시간리스트 추가
184     hashEndTime = time.time()
185     hashResultTime = hashEndTime - hashStartTime
186     hashTimeList.append(hashResultTime)
187
188     hashEndTime = time.time()
189     hashResultTime = hashEndTime - hashStartTime
190     # HASH 탐색 결과값 텍스트파일로 저장
191     with open('HashSearchResult.txt', 'w') as f:
192         for i in hashingResult:
193             tmpResult = str(i) + '\n'
194             f.write(tmpResult)
195
196 # -----

```

Hash탐색을 시작한다. 탐색과정은 위 사진과 같이 하면 된다. 탐색할 ip리스트를 하나씩 불러와 앞서 해시테이블에 저장한 거와 비교해서 찾은 ip를 결과리스트에 추가해주면 된다.

단, Longest Prefix Matching 방법에 의하여 IP 주소 8부터 31까지 Masking 수행하고 Subnet Mask한 결과를 Hash화 하여 Hash 키에대한 모든 Value를 찾아온 후 그 중 가장 긴 Subnet Mask를 가진 Network 주소의 Next Hop으로 결정하면 된다.

Longest Prefix Matching 방법은 다음 사진과 같다.

결과값은 텍스트 파일로 저장하였다.

Longest prefix Matching

- 라우터가 패킷을 포워딩할때 Next Hop을 결정하기 위해 패킷의 목적지 주소와 라우팅 테이블의 각 엔트리에 있는 prefix와 비교 수행
- subnet mask 중 가장 긴 mask를 선택하여 경로를 설정하는 것



```

19 class Trie(object):
20     def __init__(self):
21         self.head = Node(None)
22
23     def insert(self, string, data):
24         currentNode = self.head
25         for char in string:
26             if char not in currentNode.children:
27                 currentNode.children[char] = Node(char)
28             currentNode = currentNode.children[char]
29
30             if not currentNode.data:
31                 currentNode.data = data
32
33     def search(self, string):
34         currentNode = self.head
35         rdata = None
36         for char in string:
37             if char in currentNode.children:
38                 currentNode = currentNode.children[char]
39                 if currentNode.data:
40                     rdata = currentNode.data
41             else:
42                 break
43
44         if currentNode.data != None:
45             return rdata
46         else:
47             return rdata

```

Trie는 먼저 트리구조를 만들어 주고 초기화, 삽입, 탐색하는 기능을 구현해줬다.

```

206 print("Trie 탐색 시작")
207 f = open('preProcessingData.txt')
208 t = Trie()
209 while True:
210     tmpId = ''
211     line = f.readline()
212     # 줄개행 자름
213     line = line[0:len(line) - 1]
214
215     # 마지막 줄이면 종료
216     if not line:
217         break
218     networkIdMask = line.split(' ')[0]
219     hop = line.split(' ')[1]
220     networkId = networkIdMask.split('/')[0]
221     networkSplitId = networkId.split('.')
222     networkMask = networkIdMask.split('/')[1]
223
224     if networkMask != '0':
225         for i in range(0, len(networkSplitId)):
226             networkSplitId[i] = str("{0:b}".format(int(networkSplitId[i])).zfill(8))
227             tmpId += networkSplitId[i]
228             t.insert(tmpId[0:int(networkMask)], hop)
229
230 # Trie 탐색 시작
231 trieStartTime = time.time()
232 treeResultList = []
233 for ip in inputIpList:
234     tmp = ip.split('.')
235     tmp2 = ''
236     for i in range(0, len(tmp)):
237         tmp2 += "{0:b}".format(int(tmp[i])).zfill(8)
238     treeResultList.append(t.search(tmp2))
239     # 시간 리스트 추가
240     trieEndTime = time.time()
241     trieResultTime = trieEndTime - trieStartTime
242     trieTimeList.append(trieResultTime)
243     trieEndTime = time.time()
244     trieResultTime = trieEndTime - trieStartTime
245 print("Trie 탐색 끝")

```

앞서 Hash에서 했던 것처럼 ip 리스트들을 Trie에 넣어주고 찾을 ip를 search()를 통해 탐색하였다. 그리고 Trie또한 결과를 텍스트 파일로 출력하였다. Trie 노드는 비트화한 ip를 차례대로 추가해주고 nextHop을 최종 노드에 저장해주면 된다. 탐색은 탐색할 ip를 비트로 변환하여 순차적으로 대입하고 찾아 맞는 일치한 비트까지의 노드를 탐색하여 nextHop을 알아낸다.

```
246
247     # Hash, Trie 탐색 걸린시간 그래프 시각화
248     plt.plot(hashTimeList)
249     plt.plot(trieTimeList)
250     plt.xlabel("X")
251     plt.ylabel("TIME(second)")
252     plt.title("Hash, Trie Compare")
253     plt.legend(["Hash", "Trie"])
254     plt.show()
255
```

Hash와 Trie 탐색을 할 때 하나탐색할때마다의 시간을 time 리스트에 넣어주었다. 그것을 시각화한 코드이다.

전처리 파일

```
preProcessingData.pickle x
preProcessingData.txt x

94.156.252.18q@K K K@e]q@X
1.0.0.0/24q@X 87.121.64.4q@K K K@e]q@X
1.0.4.0/22q X 203.189.128.233q
K K K@e]q@X
1.0.4.0/24q@X 203.189.128.233q
K K K@e]q@X
1.0.5.0/24q@X 203.189.128.233q@K K K@e]q@X
1.0.6.0/24q@X 203.189.128.233q@K K K@e]q@X
1.0.7.0/24q@X 203.189.128.233q@K K K@e]q@X 1.0.16.0/24q@X
64.71.137.241q@K K K@e]q@X 1.0.64.0/18q@X
64.71.137.241q@K K K@e]q@X 1.0.128.0/17q@X 203.189.128.233q@K K K@e]q X 1.0.128.0/18q!X 203.189.128.233q"K K K@e]
(X 1.0.128.0/24q"X 87.121.64.4q(K K K@e]q)(X 1.0.129.0/24q*X 203.189.128.233q+K K K@e]q,(X 1.0.130.0/24q-X 87.1
.4qIK K K@e]q2(X 1.0.132.0/22q3X 87.121.64.4q4K K K@e]q5(X 1.0.136.0/24q6X 87.121.64.4q7K K K@e]q8(X 1.0.137.0/24
.121.64.4q=K K K@e]q(X 1.0.142.0/24q?X 87.121.64.4q@K K K@e]qA(X 1.0.143.0/24qBX 87.121.64.4qCK K K@e]qD(X 1.0.14
.0/19qHX 203.189.128.233qIK K K@e]qJ(X 1.0.160.0/22qKX 87.121.64.4qLK K K@e]qM(X 1.0.164.0/24qNX 203.189.128.233q
.3qRK K K@e]qS(X 1.0.166.0/24qTX 203.189.128.233qUK K K@e]qV(X 1.0.168.0/22qWX 87.121.64.4qXK K K@e]qY(X 1.0.172.
.0/20qX 87.121.64.4q*K K K@e]q(X 1.0.192.0/18q X 203.189.128.233qaK K K@e]qb(X 1.0.192.0/19qcX 203.189.128.233q
.4qgK K K@e]qh(X 1.0.200.0/24qiX 87.121.64.4qjK K K@e]qk(X 1.0.201.0/24qlX 87.121.64.4qmK K K@e]qn(X 1.0.204.0/22
.121.64.4qsK K K@e]qt(X 1.0.210.0/23quX 87.121.64.4qvK K K@e]qw(X 1.0.212.0/23qxX 87.121.64.4qyK K K@e]qz(X 1.0.21
.0/24q~X 87.121.64.4q@K K K@e]q(X 1.0.224.0/19qX 203.189.128.233q*K K K@e]q(X 1.0.240.0/21qX 87.121.64.4q
K K K@e]q(X
1.1.1.0/24qX 87.121.64.4qK K K@e]q(X
1.1.6.0/24qX
80.91.255.137qX K K K@e]q(X
1.1.8.0/24qX 87.121.64.4qK K K@e]q(X 1.1.20.0/24qX 195.22.216.188qX K K K@e]q(X 1.1.64.0/19qX
64.71.137.241qX K K K@e]q(X 1.1.103.0/24qX
64.71.137.241qX K K K@e]q(X 1.1.104.0/24qX
64.71.137.241qX K K K@e]q(X 1.1.105.0/24qX
64.71.137.241qX K K K@e]q(X 1.1.106.0/24qX
64.71.137.241qX K K K@e]q(X 1.1.107.0/24qX
64.71.137.241qX K K K@e]q(X 1.1.108.0/24qX
64.71.137.241aX K K K@e]aX 1.1.109.0/24aX

preProcessingData.pickle x preProcessingData.txt x

31.18.110.0/24 94.156.252.18
31.18.111.0/24 94.156.252.18
31.18.112.0/24 94.156.252.18
31.18.113.0/24 94.156.252.18
31.18.114.0/24 94.156.252.18
31.18.115.0/24 94.156.252.18
31.18.116.0/24 94.156.252.18
31.18.117.0/24 94.156.252.18
31.18.118.0/24 194.153.0.253
31.18.120.0/24 94.156.252.18
31.18.121.0/24 94.156.252.18
31.18.122.0/24 94.156.252.18
31.18.123.0/24 94.156.252.18
31.18.124.0/24 94.156.252.18
31.18.125.0/24 94.156.252.18
31.18.126.0/24 94.156.252.18
31.18.127.0/24 94.156.252.18
31.18.128.0/17 64.71.137.241
31.18.128.0/24 94.156.252.18
31.18.129.0/24 94.156.252.18
31.18.130.0/24 94.156.252.18
31.18.131.0/24 94.156.252.18
31.18.132.0/24 94.156.252.18
31.18.133.0/24 94.156.252.18
31.18.134.0/24 94.156.252.18
31.18.135.0/24 94.156.252.18
31.18.136.0/24 94.156.252.18
31.18.137.0/24 94.156.252.18
31.18.138.0/24 94.156.252.18
31.18.139.0/24 94.156.252.18
31.18.140.0/24 94.156.252.18
31.18.141.0/24 94.156.252.18
31.18.142.0/24 94.156.252.18
31.18.143.0/24 94.156.252.18
31.18.144.0/24 94.156.252.18
31.18.145.0/24 94.156.252.18
31.18.146.0/24 94.156.252.18
31.18.147.0/24 94.156.252.18
31.18.148.0/24 94.156.252.18
31.18.149.0/24 94.156.252.18
31.18.150.0/24 94.156.252.18
31.18.151.0/24 94.156.252.18
31.18.152.0/24 94.156.252.18
31.18.153.0/24 94.156.252.18
31.18.154.0/24 94.156.252.18
31.18.155.0/24 94.156.252.18
31.18.156.0/24 94.156.252.18
31.18.157.0/24 94.156.252.18
31.18.158.0/24 94.156.252.18
31.18.159.0/24 94.156.252.18
31.18.160.0/24 94.156.252.18
31.18.161.0/24 94.156.252.18
31.18.162.0/24 94.156.252.18
31.18.163.0/24 94.156.252.18
31.18.164.0/24 94.156.252.18
31.18.165.0/24 94.156.252.18
31.18.166.0/24 94.156.252.18
31.18.167.0/24 94.156.252.18
31.18.168.0/24 94.156.252.18
31.18.169.0/24 94.156.252.18
31.18.170.0/24 94.156.252.18
31.18.171.0/24 94.156.252.18
31.18.172.0/24 94.156.252.18
31.18.173.0/24 94.156.252.18
31.18.174.0/24 94.156.252.18
31.18.175.0/24 94.156.252.18
31.18.176.0/24 94.156.252.18
31.18.177.0/24 94.156.252.18
31.18.178.0/24 94.156.252.18
31.18.179.0/24 94.156.252.18
31.18.180.0/24 94.156.252.18
31.18.181.0/24 94.156.252.18
31.18.182.0/24 94.156.252.18
31.18.183.0/24 94.156.252.18
31.18.184.0/24 94.156.252.18
31.18.185.0/24 94.156.252.18
31.18.186.0/24 94.156.252.18
31.18.187.0/24 94.156.252.18
```

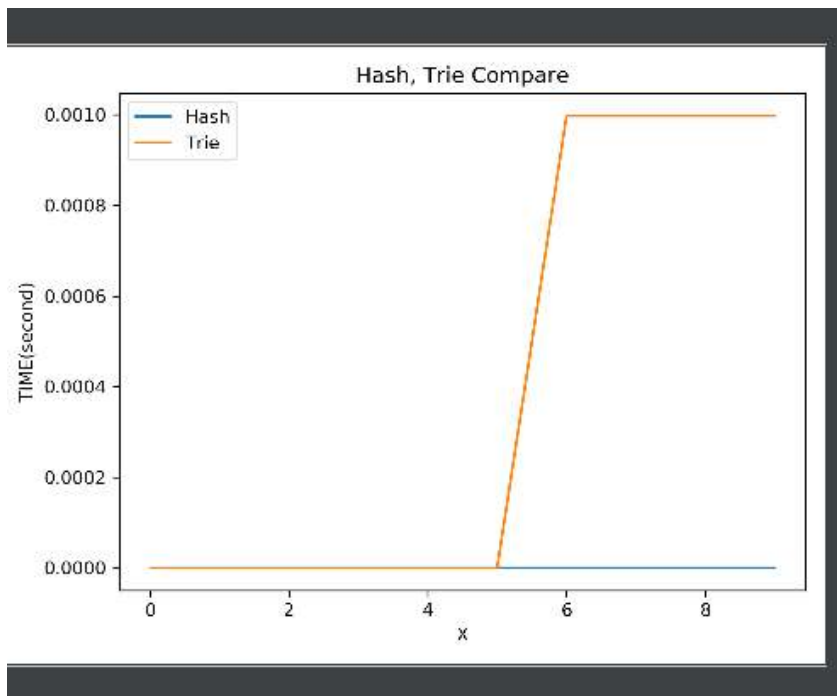
Hash탐색 결과

preProcessingData.pickle × preProcessingData.txt × HashSearchResult.txt ×	
1	None
2	195.22.216.188
3	None
4	64.71.137.241
5	94.156.252.18
6	195.22.216.188
7	None
8	87.121.64.4
9	87.121.64.4
10	None
11	None
12	195.22.216.188
13	None
14	87.121.64.4
15	None
16	None
17	None
18	194.153.0.253
19	12.0.1.63
20	None
21	137.39.3.55
22	12.0.1.63
23	None
24	87.121.64.4
25	129.250.1.71
26	195.22.216.188
27	154.11.12.212
28	None
29	None
30	None
31	203.189.128.233
32	144.228.241.130
33	None
34	195.22.216.188
35	None
36	None
37	203.189.128.233
38	None
39	195.22.216.188
40	None
41	None
42	195.22.216.188
43	154.11.12.212
44	None

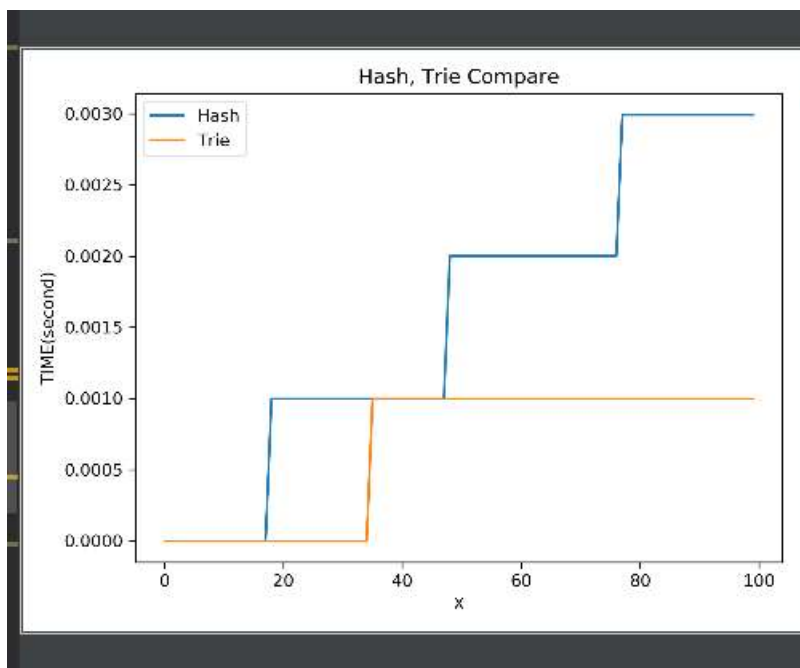
Trie탐색 결과(Hash와 일치함을 볼 수 있다.)

```
preProcessingData.pickle × preProcessingData.txt × HashSearchResult.txt × TrieSearchResult.txt ×
1 None
2 195.22.216.188
3 None
4 64.71.137.241
5 94.156.252.18
6 195.22.216.188
7 None
8 87.121.64.4
9 87.121.64.4
10 None
11 None
12 195.22.216.188
13 None
14 87.121.64.4
15 None
16 None
17 None
18 194.153.0.253
19 12.0.1.63
20 None
21 137.39.3.55
22 12.0.1.63
23 None
24 87.121.64.4
25 129.250.1.71
26 195.22.216.188
27 154.11.12.212
28 None
29 None
30 None
31 203.189.128.233
32 144.228.241.130
33 None
34 195.22.216.188
35 None
36 None
37 203.189.128.233
38 None
39 195.22.216.188
40 None
41 None
42 195.22.216.188
43 154.11.12.212
44 None
45 87.121.64.4
```

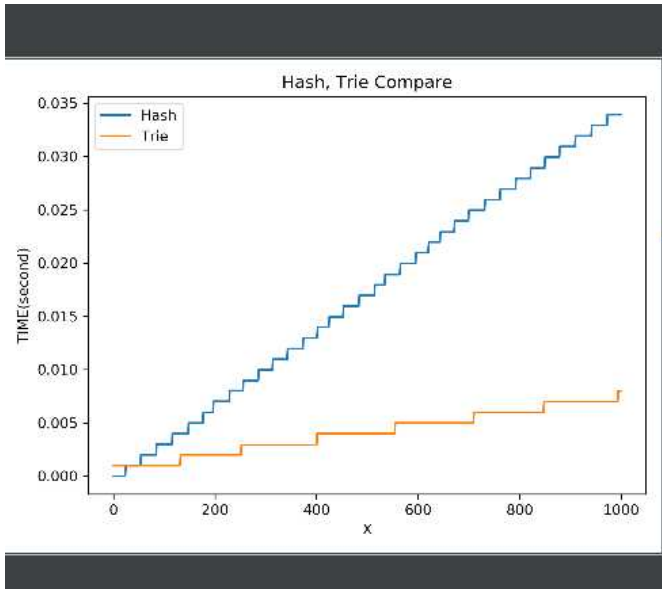
두 알고리즘 시각화 비교
(10개)



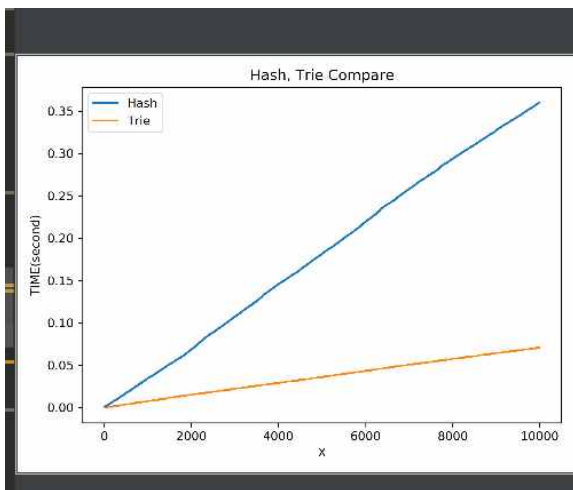
(100개)



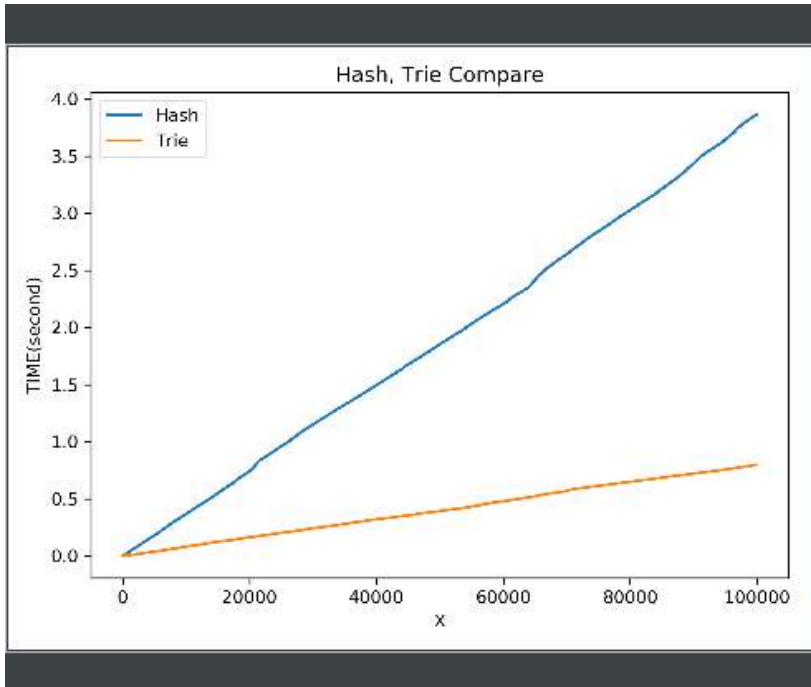
(1000개)



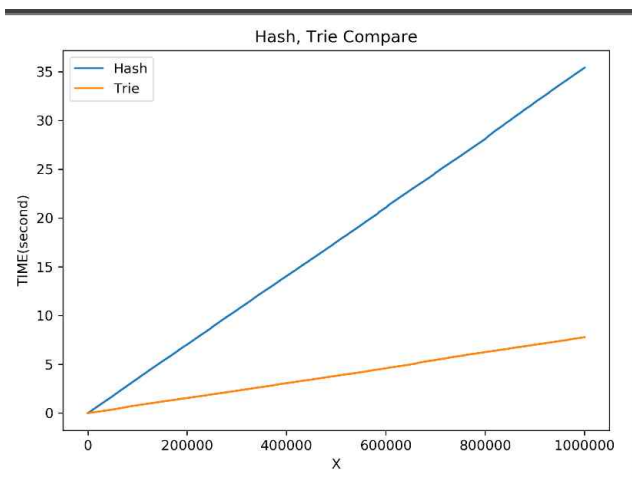
(10000개)



(100000개)



(1000000개)



(결론)

10개일 때 빼고 Trie가 우수함을 알 수 있다.

입력데이터 100000개를 기준으로 실험하였다.

Hash 탐색 시간 복잡도 : $O(1)$

Trie 탐색 시간 복잡도 : $O(m)$, m 은 탐색하려는 문자열 길이