Insert

Search

Starts With

TRIE

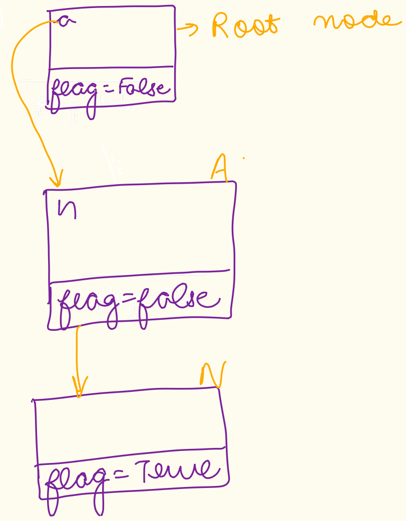
array = [ ]

① search

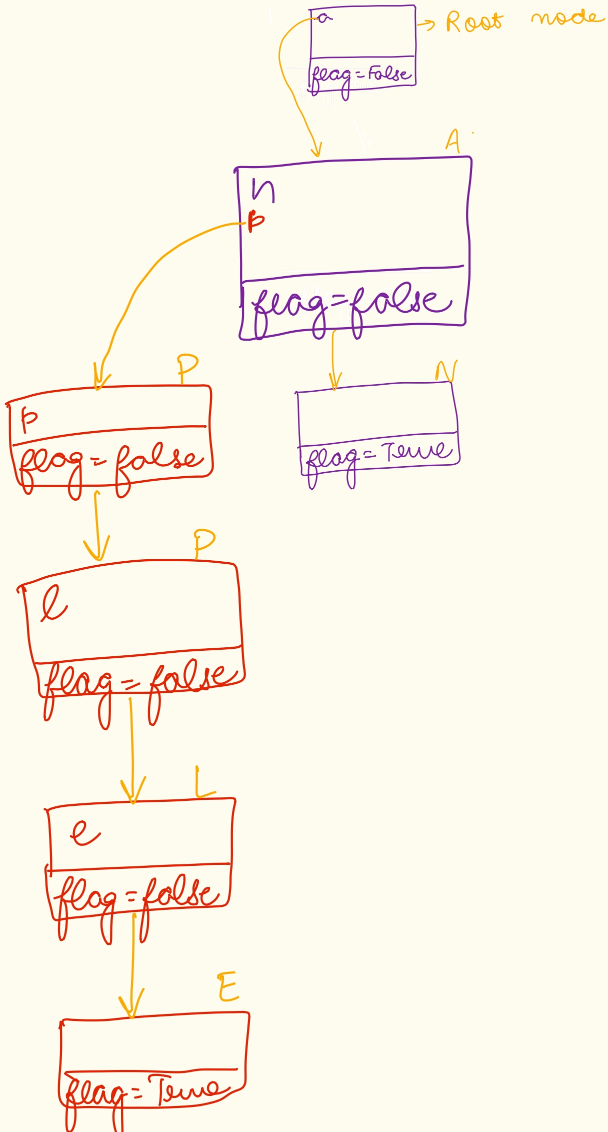
② prefix search

words = [an, apple, bcd, cayaan, ane]

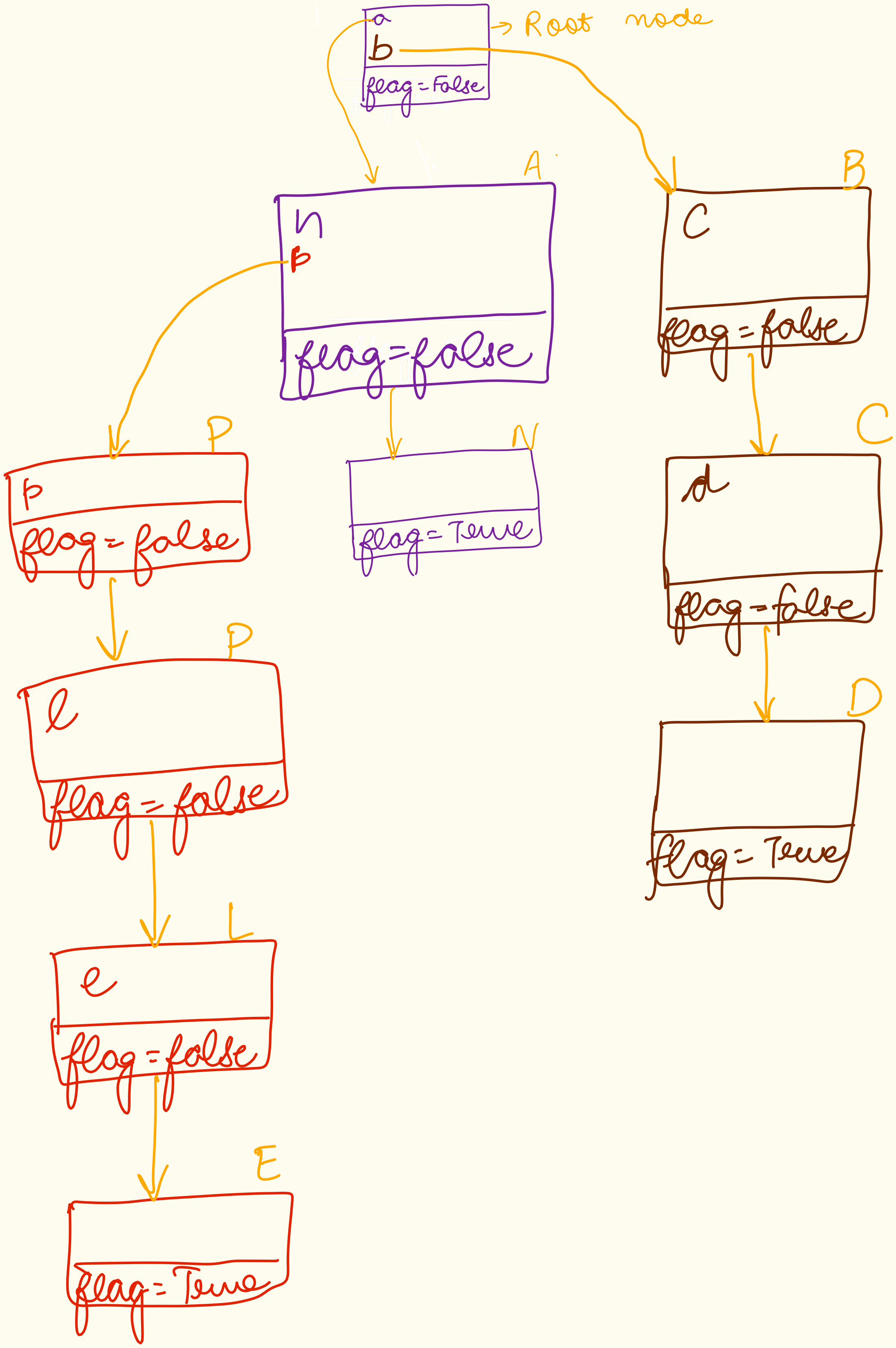
an



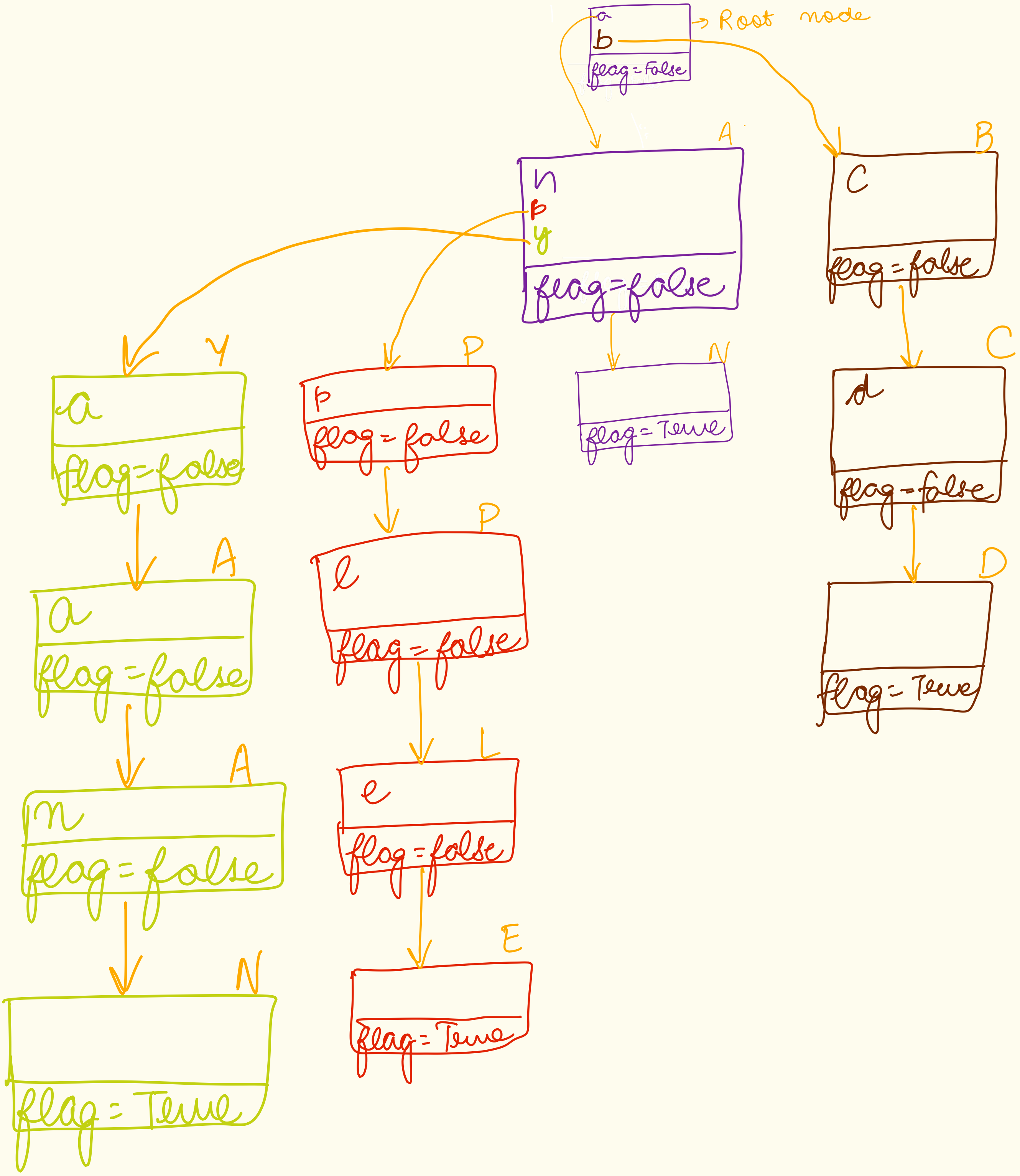
apple



bcd

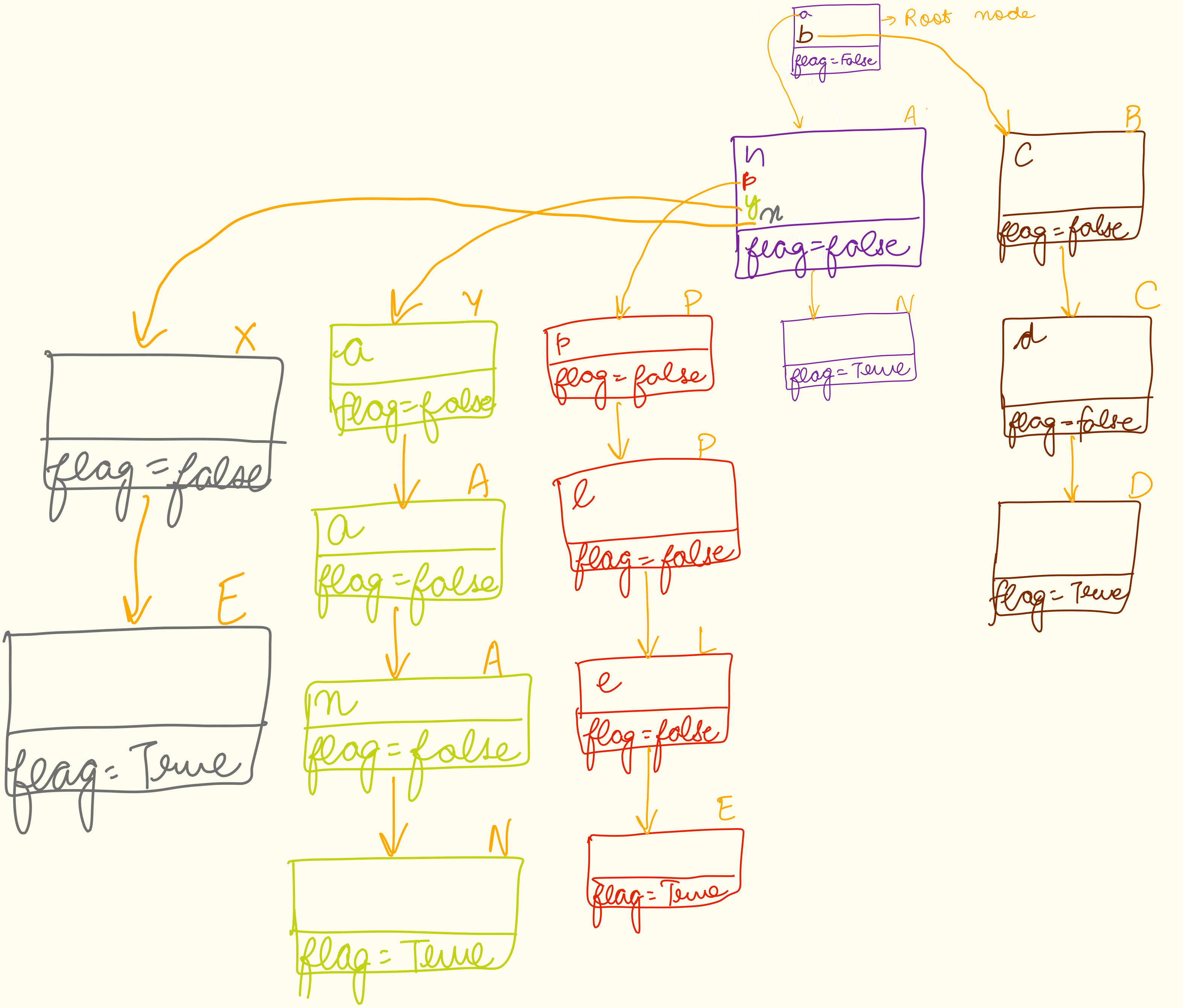

a
b
flag=False
Root node
A
n
p
flag=false
N
flag=True
B
c
flag=false
C
d
flag=false
D
flag=True
P
p
flag=false
P
l
flag=false
L
e
flag=false
E
flag=True

ayaan



a
b
flag=False
Root node
A
B
n
p
y
flag=false
c
flag=false
Y
P
N
C
a
flag=false
p
flag=false
flag=True
d
flag=false
A
P
D
a
flag=false
l
flag=false
flag=True
A
L
n
flag=false
e
flag=false
N
E
flag=True
flag=True

are



Final Trie

```
Trie {
   Trie * links[26];
   bool flag = 0

  Trie* getInsideNode (char ch){
      return links[ch-'a'];
  }
  void putInsideNode (char ch,
       Trie * node) {
   links[ch - 'a'] = node;
  }

  bool containsKey (char ch){
     return links[ch - 'a'] != NULL;
  }

  bool isEnd (){
     return flag;
  }

  void setEnd (){
     flag = true;
  }
};
```

Insert

```
void insert (string word){
   Trie*  node = root;
   int len = word.size();
   for( int i=0; i<len; i++){
      if (!node->containsKey (word[i])){
          node->putInsideNode (word[i],
          new Trie());
    node = node->getInsideNode (word[i]);
   }
  node->setEnd();
}
```

Search

```
bool search (string word]{
   Trie * node = root;
   int len= word.size();
  for(int i=0; i<len; i++){
     if(!node->containskey (word[i])
         return false;
```

```
    node = node -> getInsideNode (word[i]);
  }
  return node -> isEnd ();
}
```

Search (starts With)

```
bool startsWith (string prefix){
  Trie * node = root;
  int len = word. size();
  for(int i=0; i<len; i++){
    if(!node -> containsKey (word[i])
      return false;
    node = node -> getInsideNode (word[i]);
  }
  return true;
}
```