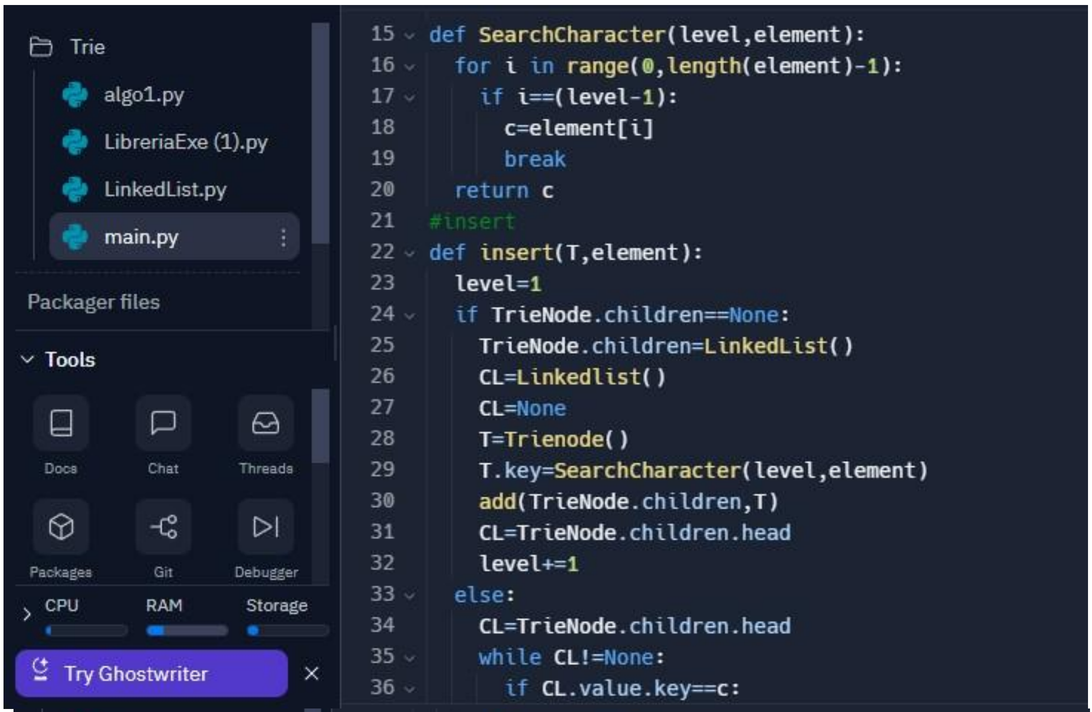
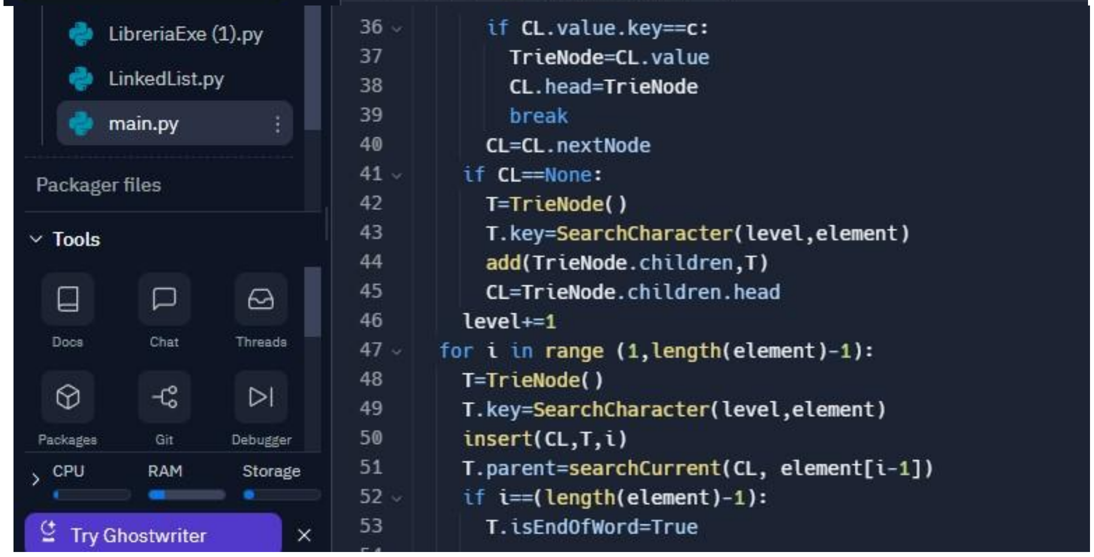
Algoritmos II

Trabajo Practico arboles Nºarios-Trie

1) a) Insert:

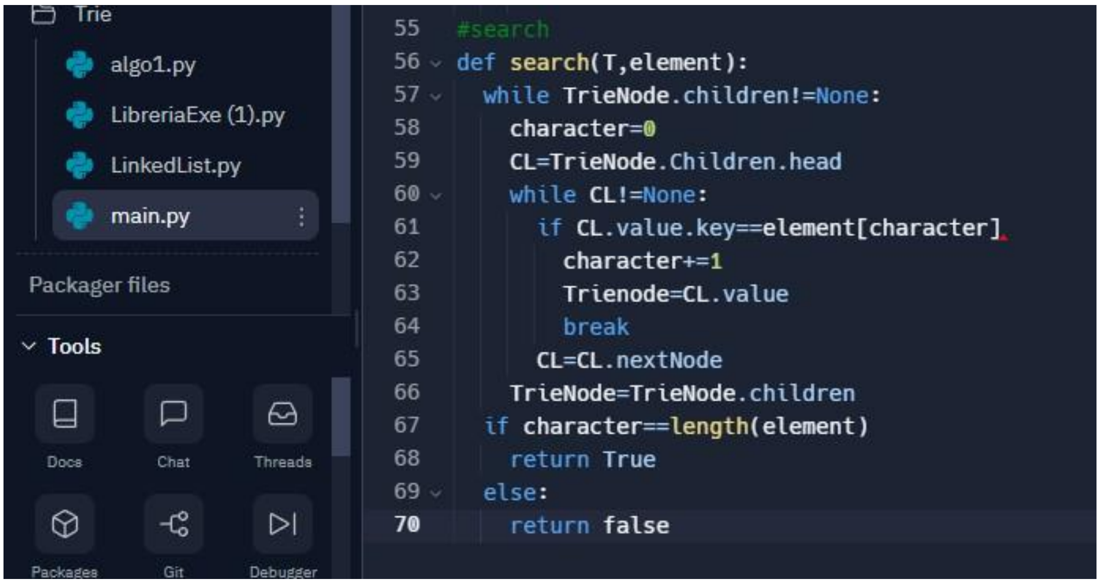


```python
def SearchCharacter(level,element):
  for i in range(0,length(element)-1):
    if i==(level-1):
      c=element[i]
      break
  return c
#insert
def insert(T,element):
  level=1
  if TrieNode.children==None:
    TrieNode.children=LinkedList()
    CL=Linkedlist()
    CL=None
    T=Trienode()
    T.key=SearchCharacter(level,element)
    add(TrieNode.children,T)
    CL=TrieNode.children.head
    level+=1
  else:
    CL=TrieNode.children.head
    while CL!=None:
      if CL.value.key==c:
```



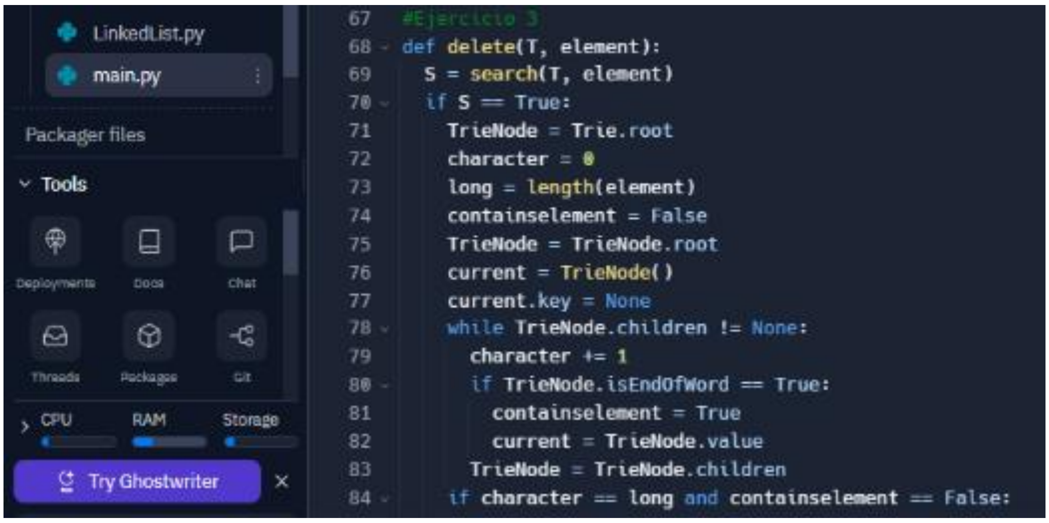
```python
      if CL.value.key==c:
        TrieNode=CL.value
        CL.head=TrieNode
        break
      CL=CL.nextNode
    if CL==None:
      T=TrieNode()
      T.key=SearchCharacter(level,element)
      add(TrieNode.children,T)
      CL=TrieNode.children.head
    level+=1
  for i in range (1,length(element)-1):
    T=TrieNode()
    T.key=SearchCharacter(level,element)
    insert(CL,T,i)
    T.parent=searchCurrent(CL, element[i-1])
    if i==(length(element)-1):
      T.isEndOfWord=True
```

B) Search:

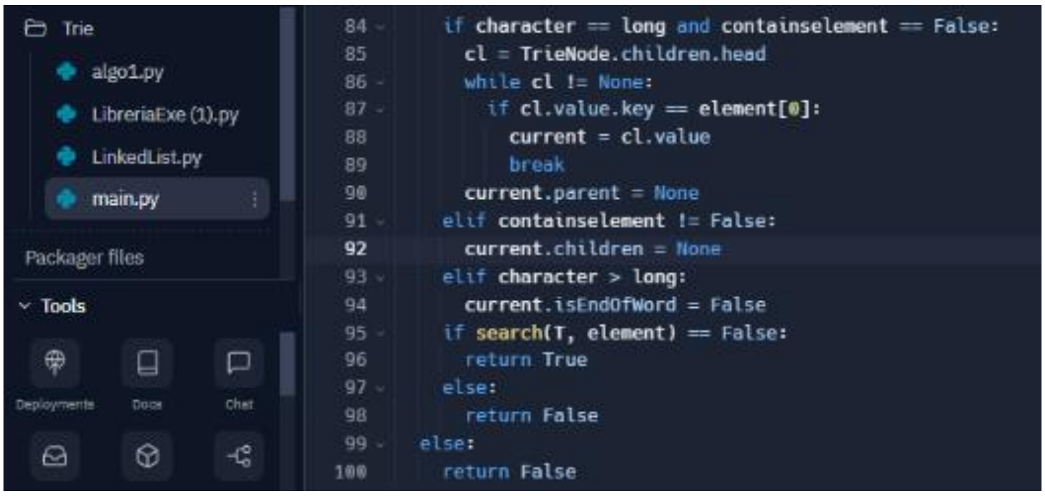


```
#search
def search(T,element):
  while TrieNode.children!=None:
    character=0
    CL=TrieNode.Children.head
    while CL!=None:
      if CL.value.key==element[character]
        character+=1
        Trienode=CL.value
        break
      CL=CL.nextNode
    TrieNode=TrieNode.children
  if character==length(element)
    return True
  else:
    return false
```

3)

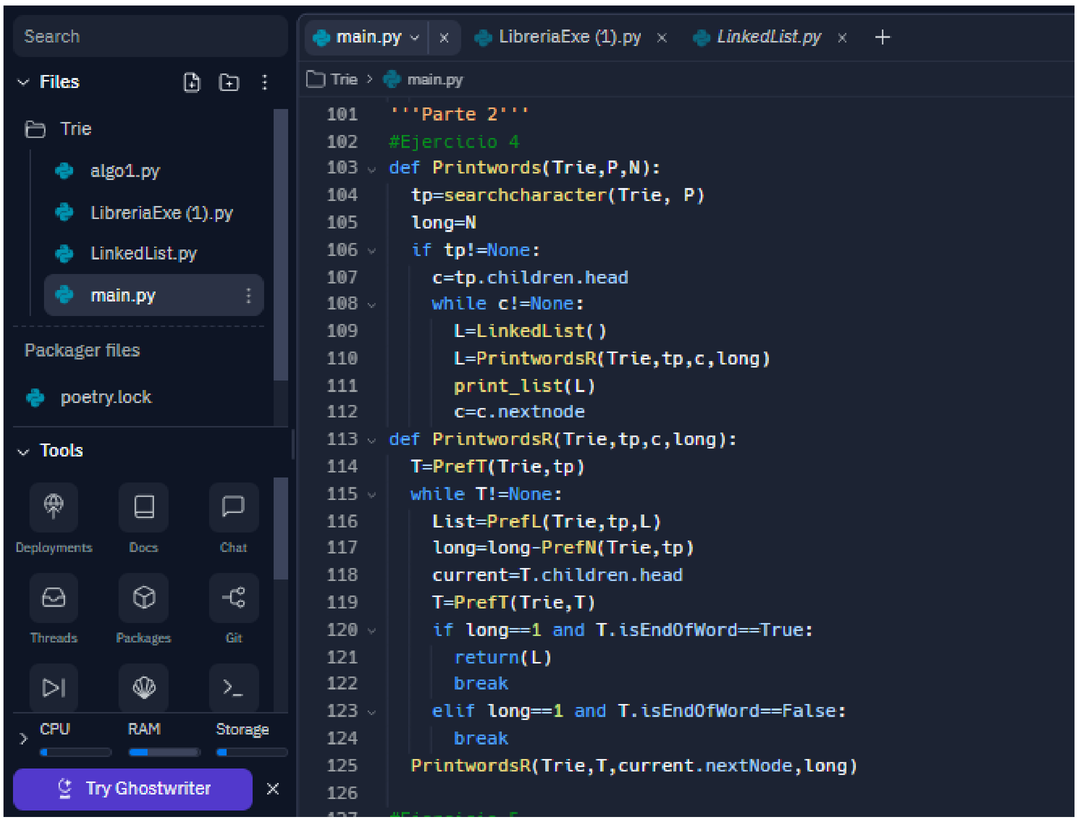


```
#Ejercicio 3
def delete(T, element):
  S = search(T, element)
  if S == True:
    TrieNode = Trie.root
    character = 0
    long = length(element)
    containselement = False
    TrieNode = TrieNode.root
    current = TrieNode()
    current.key = None
    while TrieNode.children != None:
      character += 1
      if TrieNode.isEndOfWord == True:
        containselement = True
        current = TrieNode.value
      TrieNode = TrieNode.children
    if character == long and containselement == False:
```



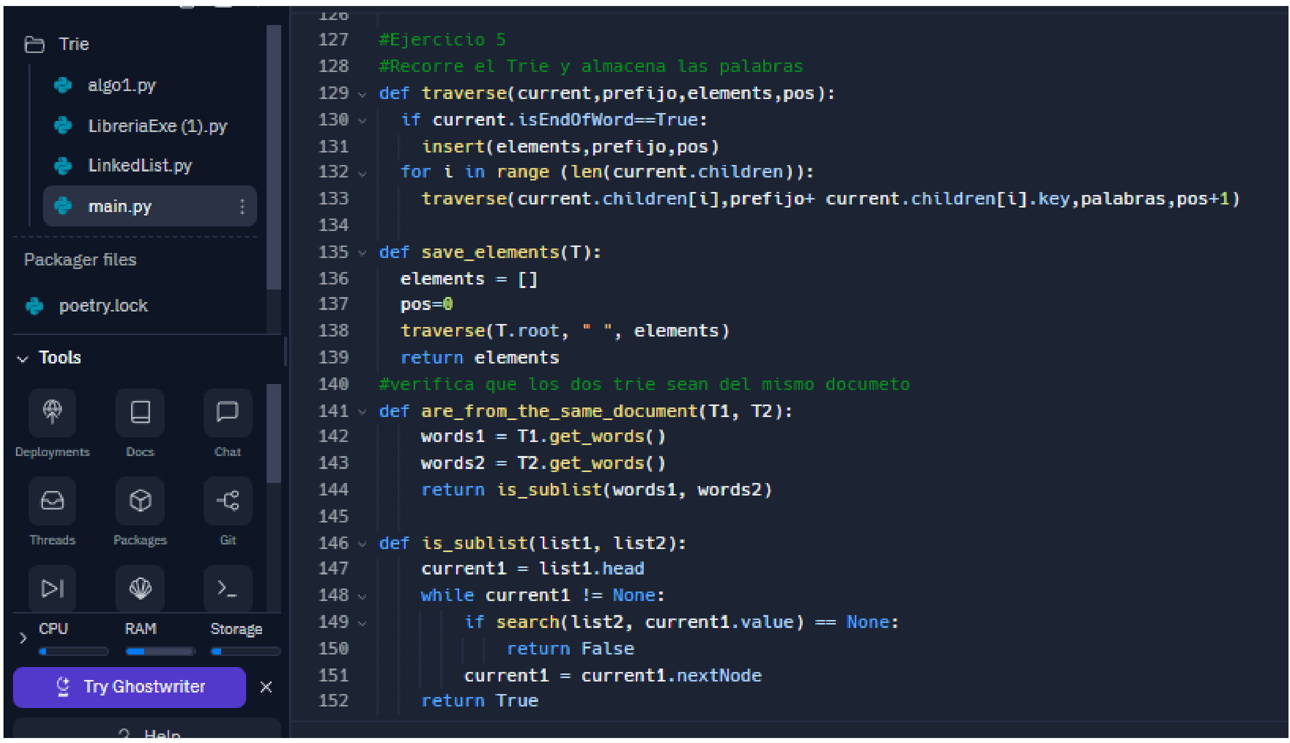
```
    if character == long and containselement == False:
      cl = TrieNode.children.head
      while cl != None:
        if cl.value.key == element[0]:
          current = cl.value
          break
      current.parent = None
    elif containselement != False:
      current.children = None
    elif character > long:
      current.isEndOfWord = False
    if search(T, element) == False:
      return True
    else:
      return False
  else:
    return False
```

4)



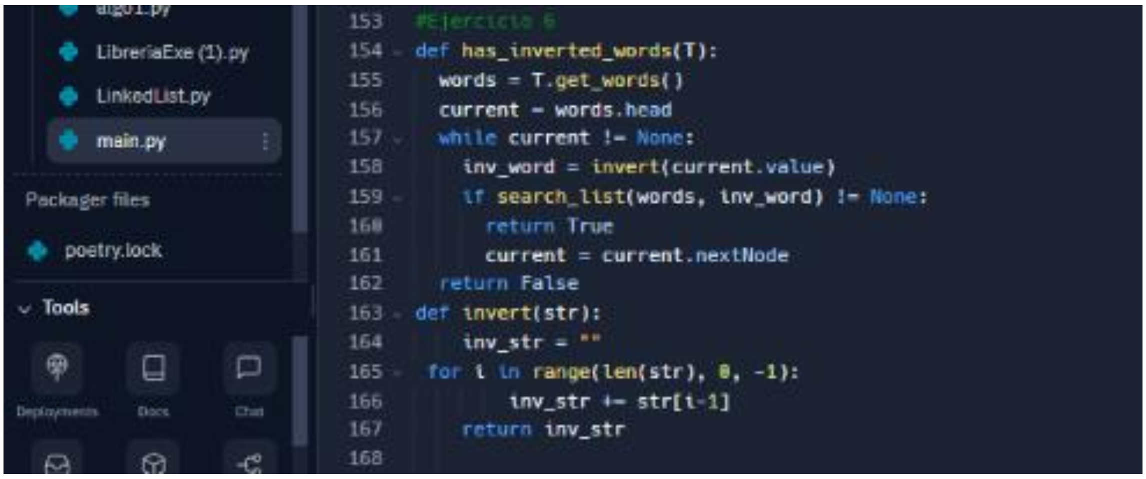
```python
'''Parte 2'''
#Ejercicio 4
def Printwords(Trie,P,N):
  tp=searchcharacter(Trie, P)
  long=N
  if tp!=None:
    c=tp.children.head
    while c!=None:
      L=LinkedList()
      L=PrintwordsR(Trie,tp,c,long)
      print_list(L)
      c=c.nextnode
def PrintwordsR(Trie,tp,c,long):
  T=PrefT(Trie,tp)
  while T!=None:
    List=PrefL(Trie,tp,L)
    long=long-PrefN(Trie,tp)
    current=T.children.head
    T=PrefT(Trie,T)
    if long==1 and T.isEndOfWord==True:
      return(L)
      break
    elif long==1 and T.isEndOfWord==False:
      break
  PrintwordsR(Trie,T,current.nextNode,long)
```

5)



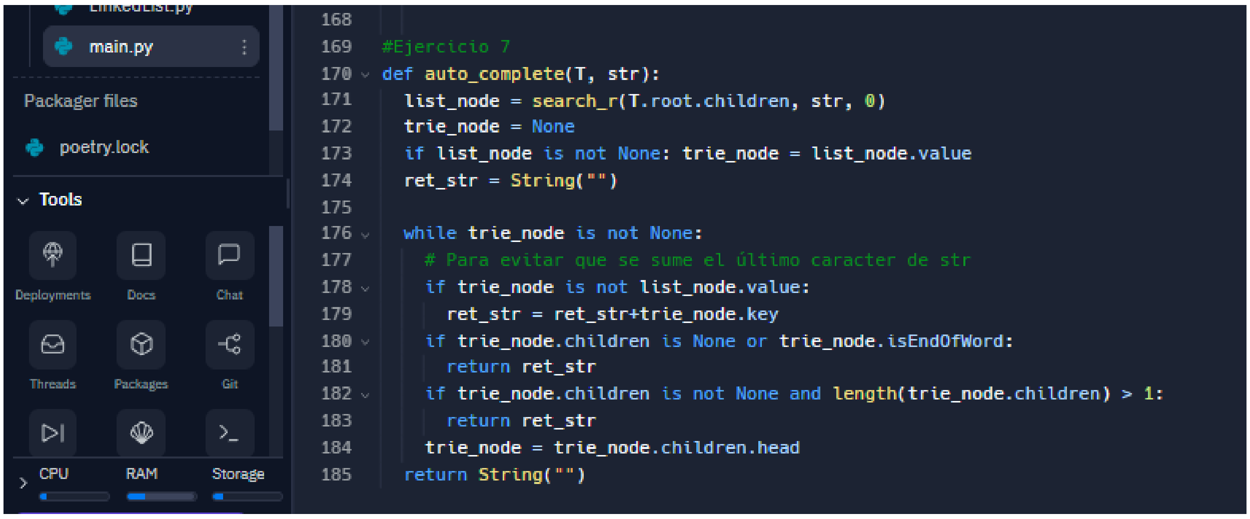
```python
#Ejercicio 5
#Recorre el Trie y almacena las palabras
def traverse(current,prefijo,elements,pos):
  if current.isEndOfWord==True:
    insert(elements,prefijo,pos)
  for i in range (len(current.children)):
    traverse(current.children[i],prefijo+ current.children[i].key,palabras,pos+1)

def save_elements(T):
  elements = []
  pos=0
  traverse(T.root, " ", elements)
  return elements
#verifica que los dos trie sean del mismo documeto
def are_from_the_same_document(T1, T2):
    words1 = T1.get_words()
    words2 = T2.get_words()
    return is_sublist(words1, words2)

def is_sublist(list1, list2):
    current1 = list1.head
    while current1 != None:
        if search(list2, current1.value) == None:
            return False
        current1 = current1.nextNode
    return True
```

6)



```
#Ejercicio 6
def has_inverted_words(T):
  words = T.get_words()
  current = words.head
  while current != None:
    inv_word = invert(current.value)
    if search_list(words, inv_word) != None:
      return True
      current = current.nextNode
  return False
def invert(str):
    inv_str = ""
 for i in range(len(str), 0, -1):
        inv_str += str[i-1]
    return inv_str
```

7)



```
#Ejercicio 7
def auto_complete(T, str):
  list_node = search_r(T.root.children, str, 0)
  trie_node = None
  if list_node is not None: trie_node = list_node.value
  ret_str = String("")

  while trie_node is not None:
    # Para evitar que se sume el último caracter de str
    if trie_node is not list_node.value:
      ret_str = ret_str+trie_node.key
    if trie_node.children is None or trie_node.isEndOfWord:
      return ret_str
    if trie_node.children is not None and length(trie_node.children) > 1:
      return ret_str
    trie_node = trie_node.children.head
  return String("")
```