**프로그래밍 실습 #8**

201501489 최영진

1. 패턴 매칭 프로그램을 파이썬으로 구현하고, 다음과 같은 정규식에 대해 정

확히 동작하는지 확인해 보라. 패턴을 입력한 다음 패턴을 식별하는 과정에

서 데크가 변화하는 과정을 출력하라. 인식되는 패턴 외에도 인식되지 않는

패턴에 대해서도 확인해 보라.

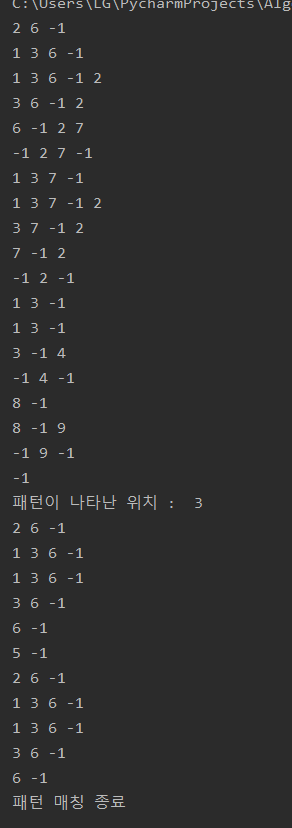
(1) (A\*B + AC)D

* 코드

scan = -1  
  
# (A\*B+AC)D  
  
ch = [' ', 'A', ' ', 'B', ' ', ' ', 'A', 'C', 'D', ' ']  
next1 = [5, 2, 3, 4, 8, 6, 7, 8, 9, 0]  
next2 = [5, 2, 1, 4, 8, 2, 7, 8, 9, 0]  
'''  
# (A + B)\*C  
  
ch = [' ', ' ', 'A', 'B', ' ', 'C', ' ']  
next1 = [4, 2, 4, 4, 5, 6, 0]  
next2 = [4, 3, 4, 4, 1, 6, 0]  
  
# (AB\* + A\*D)E  
  
ch = [' ',' ','A',' ','A', ' ', 'B', 'D', ' ', 'E', ' ']  
next1 = [1, 2, 5, 7, 3, 6, 5, 8, 9, 10, 0]  
next2 = [1, 3, 5, 4, 3, 8, 5, 8, 9, 10, 0]  
  
  
# (A+B)\*(C+D)\*E  
  
ch = [' ', ' ', 'A', 'B', ' ', ' ', 'C', 'D', ' ', 'E', ' ']  
next1 = [4, 2, 4, 4, 1, 6, 8, 8, 5, 10, 0]  
next2 = [4, 3, 4, 4, 8, 7, 8, 8, 9, 10, 0]  
'''  
class Deque:  
 def \_\_init\_\_(self, size):  
 self.deque = []  
 self.first = int(size/2)  
 self.last = int(size/2)  
 for i in range(size):  
 self.deque.append(0)  
  
 def insertFirst(self, v):  
 self.deque[self.first] = v  
 self.first -= 1  
  
 def insertLast (self, v):  
 self.last += 1  
 self.deque[self.last] = v  
  
 def deleteFirst(self):  
 self.deque[self.first] = 0  
 self.first += 1  
 return self.deque[self.first]  
  
 def isEmpty(self):  
 if self.first == self.last:  
 return True  
 else:  
 return False  
  
 def checkDq(self):  
 if self.deque[self.first] == 0:  
 if self.last - self.first < 2 and self.deque[self.last] == scan:  
 return False  
 elif not self.isEmpty():  
 return True  
 else:  
 return False  
 else:  
 return False  
  
 def prDq(self, size):  
 for i in range(size):  
 if self.deque[i] != 0:  
 print(self.deque[i], end=' ')  
 print()  
  
def match(t):  
 dq = Deque(100)  
 j = 0  
 N = len(t)-1  
 state = next1[0]  
 dq.insertLast(scan)  
 while state:  
 if state == scan:  
 j += 1  
 if dq.isEmpty():  
 dq.insertFirst(next1[0])  
 dq.insertLast(scan)  
 elif ch[state] == t[j]:  
 dq.insertLast(next1[state])  
 elif ch[state] == ' ':  
 n1 = next1[state]  
 n2 = next2[state]  
 dq.insertFirst(n1)  
 if n1 != n2:  
 dq.insertFirst(n2)  
 if dq.isEmpty() : return j  
 if j > N : return 0  
 dq.prDq(100) # 왜 여기냐면   
 state = dq.deleteFirst()  
 if dq.checkDq():  
 state = dq.deleteFirst()  
 return j-1  
  
text = 'AABD' + '\0'  
previous = 0  
i = 0  
N = len(text)-1  
while True:  
 pos = match(text[i:])  
 if pos <= 0:  
 break  
 pos += previous  
 i = pos  
 if i <= N:  
 print('패턴이 나타난 위치 : ', pos)  
 else:  
 break  
 previous = i  
print('패턴 매칭 종료')

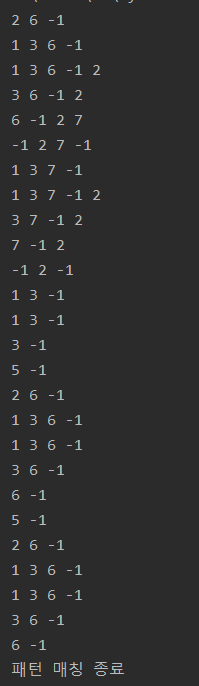
=====================================================

* 인식되는 경우(AABD)



=====================================================

* 인식되지 않는 경우(AACD)



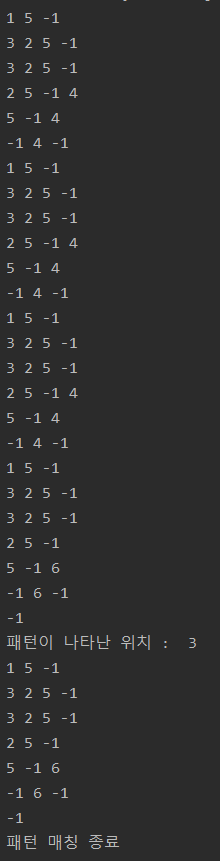
(2) (A + B)\*C

* 코드

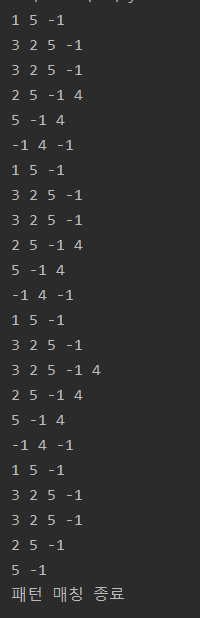
# (A + B)\*C  
  
ch = [' ', ' ', 'A', 'B', ' ', 'C', ' ']  
next1 = [4, 2, 4, 4, 5, 6, 0]  
next2 = [4, 3, 4, 4, 1, 6, 0]

=====================================================

* 인식되는 경우(AAAC)



* 인식되지 않는 경우(AAB)



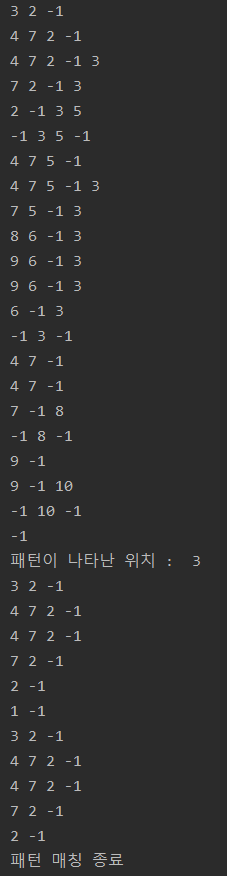
(3) (AB\* + A\*D)E

* 코드

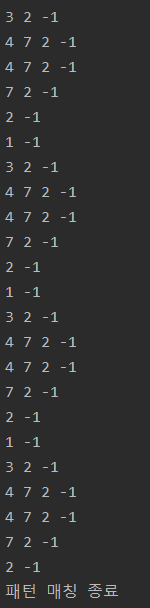
# (AB\* + A\*D)E  
  
ch = [' ',' ','A',' ','A', ' ', 'B', 'D', ' ', 'E', ' ']  
next1 = [1, 2, 5, 7, 3, 6, 5, 8, 9, 10, 0]  
next2 = [1, 3, 5, 4, 3, 8, 5, 8, 9, 10, 0]

=====================================================

* 인식되는 경우(AADE)



* 인식되지 않는 경우(BBE)



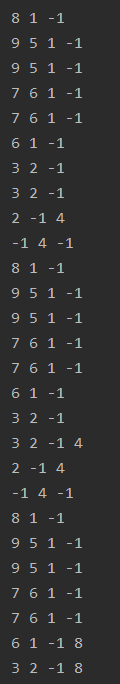
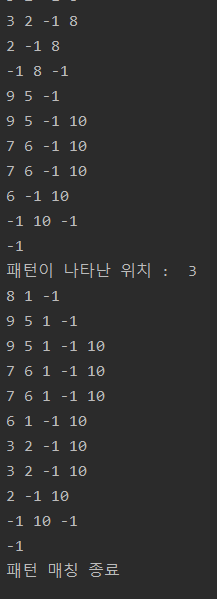
(4) (A+B)\*(C+D)\*E

* 코드

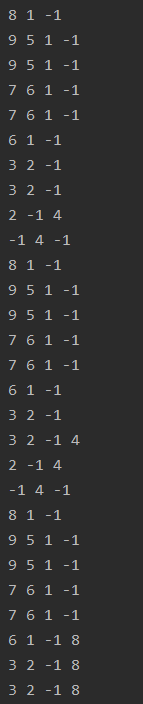
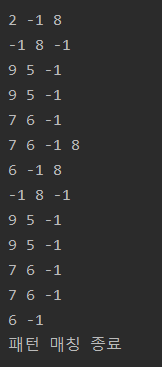
# (A+B)\*(C+D)\*E  
  
ch = [' ', ' ', 'A', 'B', ' ', ' ', 'C', 'D', ' ', 'E', ' ']  
next1 = [4, 2, 4, 4, 1, 6, 8, 8, 5, 10, 0]  
next2 = [4, 3, 4, 4, 8, 7, 8, 8, 9, 10, 0]

=====================================================

* 인식되는 경우(ABCE)

* 인식되지 않는 경우(ABCD)

2. 문자열 “A SIMPLE STRING TO BE ENCODED USING A MINIMAL

NUMBER OF BITS”에 대해 물음에 답하라.

(1) 문자들의 빈도수를 구하는 프로그램을 작성하고, count[k]를 출력하라.

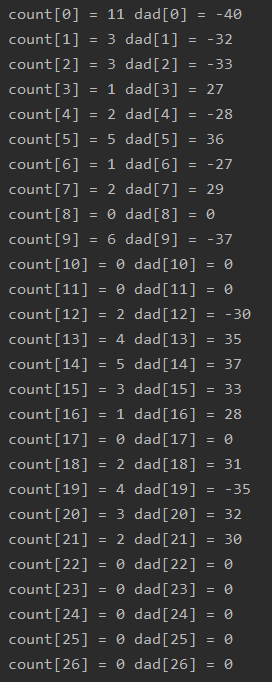
value = [' ', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'I', 'L', 'M', 'N', 'O', 'P', 'R', 'S', 'T', 'U']  
ten = "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"  
v = list(ten)  
N = len(v)  
key = len(value)  
  
def count(v,value, N, key):  
 count = list(range(len(value))) # or []  
 for i in range(key):  
 su = 0  
 for j in range(N):  
 if value[i] == v[j]:  
 su += 1  
 count[i] = su  
 print(value)  
 print(count)  
  
count(v,value,N,key)



(2) 허프만 트리를 구성하기 위해 count[k]와 dad[k]를 구하는 프로그램을

작성하고, count[k]와 dad[k]를 출력하라.

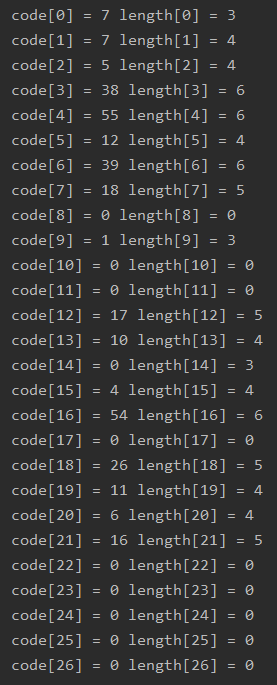
class PQ:  
 def \_\_init\_\_(self):  
 self.heap = [0] \* 100  
 self.info = [0] \* 100  
 self.n = 0  
  
 def insert(self, v, x):  
 self.n += 1  
 i = self.n  
 while True:  
 if i == 1: break  
 if v >= self.heap[int(i / 2)]: break  
 self.heap[i] = self.heap[int(i / 2)]  
 self.info[i] = self.info[int(i / 2)]  
 i = int(i / 2)  
 self.heap[i] = v  
 self.info[i] = x  
  
 def remove(self):  
 x = self.info[1]  
 temp\_v = self.heap[self.n]  
 temp\_x = self.info[self.n]  
 self.n -= 1  
 i = 1  
 j = 2  
 while j <= self.n:  
 if (j < self.n) and (self.heap[j] > self.heap[j+1]):  
 j += 1  
 if temp\_v <= self.heap[j]: break  
 self.heap[i] = self.heap[j]  
 self.info[i] = self.info[j]  
 i = j  
 j \*= 2  
 self.heap[i] = temp\_v  
 self.info[i] = temp\_x  
 return x  
  
 def isEmpty(self):  
 if self.n == 0: return True  
 else: return False  
  
def index(c):  
 if ord(c) == 32: return 0  
 else: return (ord(c)-64)  
  
def makeHuffman(t,m):  
 for i in range(m):  
 count[index(t[i])] += 1  
 for i in range(27):  
 if count[i]:  
 pq.insert(count[i], i)  
 i += 1  
 while not pq.isEmpty():  
 t1 = pq.remove()  
 t2 = pq.remove()  
 dad[i] = 0  
 dad[t1] = i  
 dad[t2] = -i  
 count[i] = count[t1] + count[t2]  
 if not pq.isEmpty():  
 pq.insert(count[i], i)  
 i += 1



(3) 허프만 코드를 생성하기 위한 code[k]와 length[k]를 구하는 프로그램

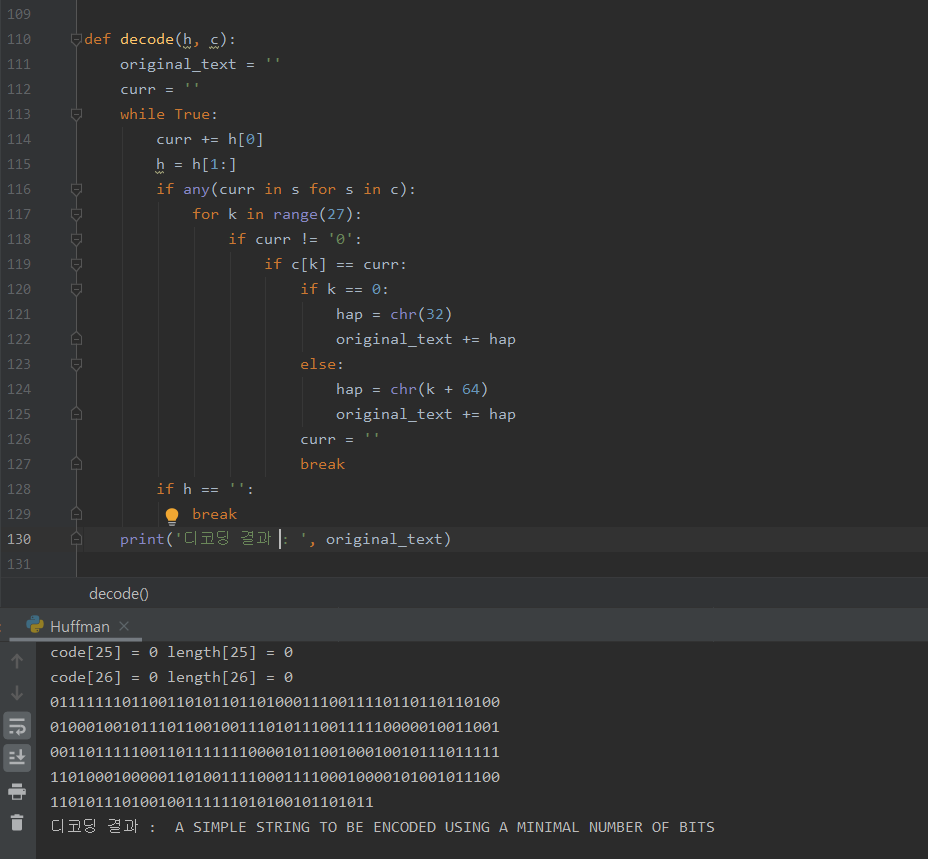
을 작성하고, code[k]와 length[k]를 출력하라.

for k in range(27):  
 i = x = 0  
 j = 1  
 if count[k]:  
 q = dad[k]  
 while q:  
 if q < 0:  
 x += j  
 q = -q  
 q = dad[q]  
 j += j  
 i += 1  
 code[k] = x  
 length[k] = i



3. count[k]와 dad[k]로 이루어진 허프만 트리가 주어질 경우 허프만 코드로

인코딩된 스트링을 복원하는 함수 decode()를 구현하라.



* 전체 코드

import collections  
  
class PQ:  
 def \_\_init\_\_(self):  
 self.heap = [0] \* 100  
 self.info = [0] \* 100  
 self.n = 0  
  
 def insert(self, v, x):  
 self.n += 1  
 i = self.n  
 while True:  
 if i == 1: break  
 if v >= self.heap[int(i / 2)]: break  
 self.heap[i] = self.heap[int(i / 2)]  
 self.info[i] = self.info[int(i / 2)]  
 i = int(i / 2)  
 self.heap[i] = v  
 self.info[i] = x  
  
 def remove(self):  
 x = self.info[1]  
 temp\_v = self.heap[self.n]  
 temp\_x = self.info[self.n]  
 self.n -= 1  
 i = 1  
 j = 2  
 while j <= self.n:  
 if (j < self.n) and (self.heap[j] > self.heap[j+1]):  
 j += 1  
 if temp\_v <= self.heap[j]: break  
 self.heap[i] = self.heap[j]  
 self.info[i] = self.info[j]  
 i = j  
 j \*= 2  
 self.heap[i] = temp\_v  
 self.info[i] = temp\_x  
 return x  
  
 def isEmpty(self):  
 if self.n == 0: return True  
 else: return False  
  
def index(c):  
 if ord(c) == 32: return 0  
 else: return (ord(c)-64)  
  
def makeHuffman(t,m):  
 code1 = [0] \* 27  
 for i in range(m):  
 count[index(t[i])] += 1  
 for i in range(27):  
 if count[i]:  
 pq.insert(count[i], i)  
 i += 1  
 while not pq.isEmpty():  
 t1 = pq.remove()  
 t2 = pq.remove()  
 dad[i] = 0  
 dad[t1] = i  
 dad[t2] = -i  
 count[i] = count[t1] + count[t2]  
 if not pq.isEmpty():  
 pq.insert(count[i], i)  
 i += 1  
  
 for k in range(27):  
 i = x = 0  
 j = 1  
 curr = ''  
 if count[k]:  
 q = dad[k]  
 while q:  
 if q < 0:  
 x += j  
 q = -q  
 q = dad[q]  
 j += j  
 i += 1  
 code[k] = x  
 length[k] = i  
 code1[k] = bin(int(code[k]))[2:].zfill(length[k])  
 #print('count[%i] = %s' %(k, count[k]),'dad[%i] = %s' %(k, dad[k]))  
 print('code[%i] = %s' %(k, code1[k]),'length[%i] = %s' %(k, length[k]))  
 return code1  
  
def removeAllOccur(l, i):  
 try:  
 while True : l.remove(i)  
 except ValueError:  
 pass  
  
def encode(t,m):  
 huffman\_code = ''  
 for j in range(m):  
 i = length[index(t[j])]  
 while i > 0:  
 huffman\_code += str((code[index(t[j])] >> i - 1) & 1)  
 i -= 1  
 n = len(huffman\_code)  
 cnt = 0  
 for i in range(n):  
 cnt += 1  
 print(huffman\_code[i], end='')  
 if cnt % 50 == 0:  
 print()  
 return huffman\_code  
  
  
def decode(h, c):  
 original\_text = ''  
 curr = ''  
 while True:  
 curr += h[0]  
 h = h[1:]  
 if any(curr in s for s in c):  
 for k in range(27):  
 if curr != '0':  
 if c[k] == curr:  
 if k == 0:  
 hap = chr(32)  
 original\_text += hap  
 else:  
 hap = chr(k + 64)  
 original\_text += hap  
 curr = ''  
 break  
 if h == '':  
 break  
 print('디코딩 결과 : ', original\_text)  
  
  
  
  
  
text = 'A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS'  
count = [0]\*100  
dad = [0]\*100  
length = [0]\*27  
code = [0]\*27  
M = len(text)  
  
pq = PQ()  
c = makeHuffman(text, M)  
h = encode(text, M)  
print()  
decode(h,c)