**프로그래밍 실습 #9**

2023년 11월 5주차

1. 다음과 같은 HTML 파일에서 "mailto:" 라는 문자열 뒤에 있는 이메일 주소를 가져와서 한 줄에 하나씩 화면에 출력하는 프로그램을 작성하라.

프로그램은 다음과 같은 순서로 작성한다.

(1) 먼저 파이썬 소스 파일이 저장된 폴더와 같은 폴더에 email.html 파일을 저장한다.

(2) email.html 파일의 내용을 변수 text에 저장한다.

(3) 직선적 스트링 탐색 알고리즘을 구현한 bruteForce() 함수를 사용하여 "mailto:" 패턴이 나오는 위치를 탐색한다.

(4) "mailto:" 패턴을 찾으면 "(쌍따옴표)가 나올 때까지 text에 있는 문자를 모아서 화면에 출력한다.

[email.html 파일 내용]

--------------------------------------------------------------------

<html>

<head>

<title>이메일 출력</title>

</head>

<body>

<ul>

<li> <a href="mailto:gdhong@hanmail.net">홍길동</a>

<li> <a href="mailto:gsjang@yahoo.co.kr">장길산</a>

<li> <a href="mailto:yhkim@naver.com">김영희</a>

</ul>

</body>

</html>

--------------------------------------------------------------------

email.html 파일을 여는 방법은 다음과 같다.

--------------------------------------------------------------------

fin = open('email.html', encoding='UTF-8')

text = fin.read()

print(text)

--------------------------------------------------------------------

[출력 예]

--------------------------------------------------------------------

gdhong@hanmail.net

gsjang@yahoo.co.kr

yhkim@naver.com

--------------------------------------------------------------------

2. 다음 패턴 매칭 알고리즘을 ADL로 작성하라.

--------------------------------------------------------------------

match(t[])

dq ← Deque(100);

j ← 0;

N ← t의 길이 – 1;

state ← next1[0];

dq.insertLast(scan);

while (state ≠ 0) {

case {

state = scan:

j ← j + 1;

if (dq.isEmpty()) then dq.insertFirst(next1[0]);

dq.insertLast(scan);

ch[state] = t[j]:

dq.insertLast(next1[state]);

ch[state] = ‘ ’:

n1 ← next1[state];

n2 ← next2[state];

dq.insertFirst(n1);

if (n1 ≠ n2) then dq.insertFirst(n2);

}

if (dq.isEmpty()) then return j;

if (j > N) then return 0

state ← dq.deleteFirst()

if (dq.checkDq()) then state ← dq.deleteFirst();

return j-1;

end match()

--------------------------------------------------------------------

패턴 매칭 알고리즘을 파이썬으로 작성하고, 다음과 같은 정규식에 대해 정확히 동작하는지 확인해 보라. 패턴을 입력한 다음 패턴을 식별하는 과정에서 데크가 변화하는 과정을 출력하라. 인식되는 패턴 외에도 인식되지 않는 패턴에 대해서도 확인해 보라.

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

(2) (A + B)\*C

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

정규식이 (A\*B + AC)D 이고 텍스트가 AABD 일 때 데크의 변화 과정은 다음과 같다. 데크에서 0이 아닌 원소를 출력하는 prDq() 함수를 사용하라.

--------------------------------------------------------------------

2 6 -1

1 3 6 -1

1 3 6 -1 2

3 6 -1 2

6 -1 2 7

-1 2 7 -1

1 3 7 -1

1 3 7 -1 2

3 7 -1 2

7 -1 2

-1 2 -1

1 3 -1

1 3 -1

3 -1 4

-1 4 -1

8 -1

8 -1 9

-1 9 -1

-1

패턴이 나타난 위치 : 3

2 6 -1

1 3 6 -1

1 3 6 -1

3 6 -1

6 -1

5 -1

2 6 -1

1 3 6 -1

1 3 6 -1

3 6 -1

6 -1

패턴 매칭 종료

--------------------------------------------------------------------

--------------------------------------------------------------------

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]

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):

pass

text = 'AABD' + '\0'

#text = 'AABCAAABBACDE' + '\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')

--------------------------------------------------------------------

3. 허프만 트리를 생성해 주는 파이썬 프로그램을 사용하여 문자열 “A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS”에 대한 허프만 코드를 생성하려고 한다.

(1) count[k]와 dad[k]를 출력하여 다음 표와 동일한지 확인해 보라.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 | 12 | 13 | 14 | 15 | 16 | 18 | 19 | 20 | 21 |
| count[k] | 11 | 3 | 3 | 1 | 2 | 5 | 1 | 2 | 6 | 2 | 4 | 5 | 3 | 1 | 2 | 4 | 3 | 2 |
| dad[k] | -40 | -32 | -33 | 27 | -28 | 36 | -27 | 29 | -37 | -30 | 35 | 37 | 33 | 28 | 31 | -35 | 32 | 30 |
| k | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 |  |
| count[k] | 2 | 3 | 4 | 4 | 5 | 6 | 6 | 8 | 8 | 10 | 11 | 12 | 16 | 21 | 33 | 37 | 60 |  |
| dad[k] | -29 | -31 | -34 | 34 | -36 | -38 | 38 | 39 | -39 | 40 | 41 | -41 | 42 | -42 | 43 | -43 | 0 |  |

(2) code[k]와 length[k]를 출력하여 다음 표와 동일한지 확인해 보라.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | k | code[k] | length[k] |  |
|  | 0 | 7 | 3 | 111 |
| A | 1 | 7 | 4 | 0111 |
| B | 2 | 5 | 4 | 0101 |
| C | 3 | 38 | 6 | 100110 |
| D | 4 | 55 | 6 | 110111 |
| E | 5 | 12 | 4 | 1100 |
| F | 6 | 39 | 6 | 100111 |
| G | 7 | 18 | 5 | 10010 |
| I | 9 | 1 | 3 | 001 |
| L | 12 | 17 | 5 | 10001 |
| M | 13 | 10 | 4 | 1010 |
| N | 14 | 0 | 3 | 000 |
| O | 15 | 4 | 4 | 0100 |
| P | 16 | 54 | 6 | 110110 |
| R | 18 | 26 | 5 | 11010 |
| S | 19 | 11 | 4 | 1011 |
| T | 20 | 6 | 4 | 0110 |
| U | 21 | 16 | 5 | 10000 |

(3) findDad(maxIndex, k) 함수를 사용하여 허프만 코드로 인코딩된 스트링을 디코드하는 함수 decode()를 파이썬으로 작성하고, 생성된 허프만 코드가 정확하게 디코드되는지 확인해 보라.

--------------------------------------------------------------------

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 = 27

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

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

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

return huffman\_code

def char(k):

if k == 0: return chr(32)

else: return chr(k+64)

def findDad(max\_i, k):

for i in range(max\_i):

if dad[i] == k:

return i

def decode(h):

pass

text = 'VISION QUESTION ONION CAPTION GRADUATION EDUCATION'

#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()

makeHuffman(text, M)

h = encode(text, M)

print(h)

#d = decode(h)

#print(d)

--------------------------------------------------------------------