

알고리즘분석 실습 자료

9주차

Photo by [Piotr Guzik](#) on [Unsplash](#)



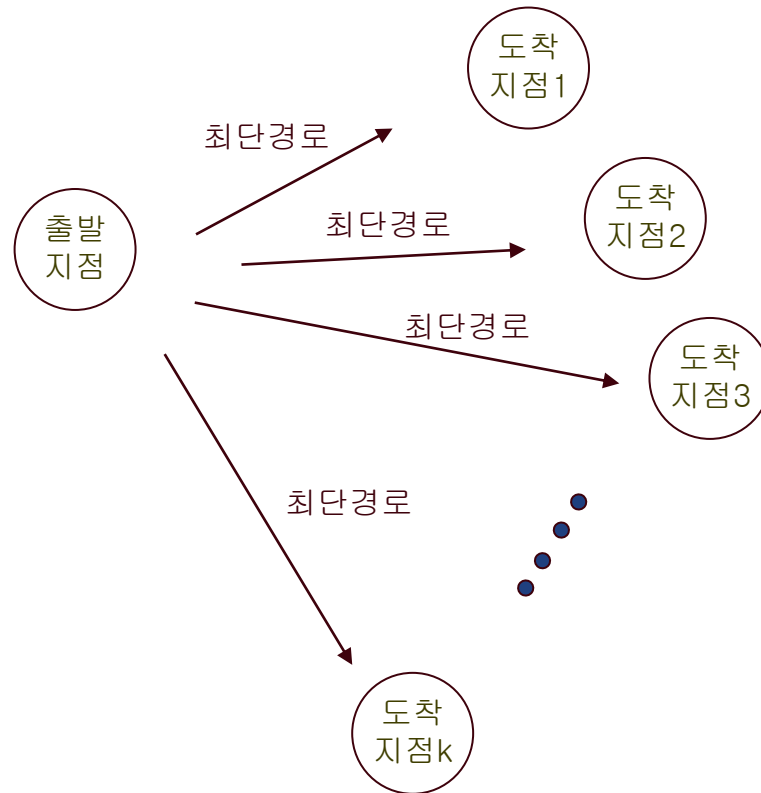
4장 탐욕적인 접근방법 (Greedy Algorithm)

실습프로그램

- ✓ Dijkstra의 알고리즘
- ✓ Huffman code 1
- ✓ Huffman code 2



단일출발점 최단경로문제(single source shortest path problem) Dijkstra의 알고리즘(1959)

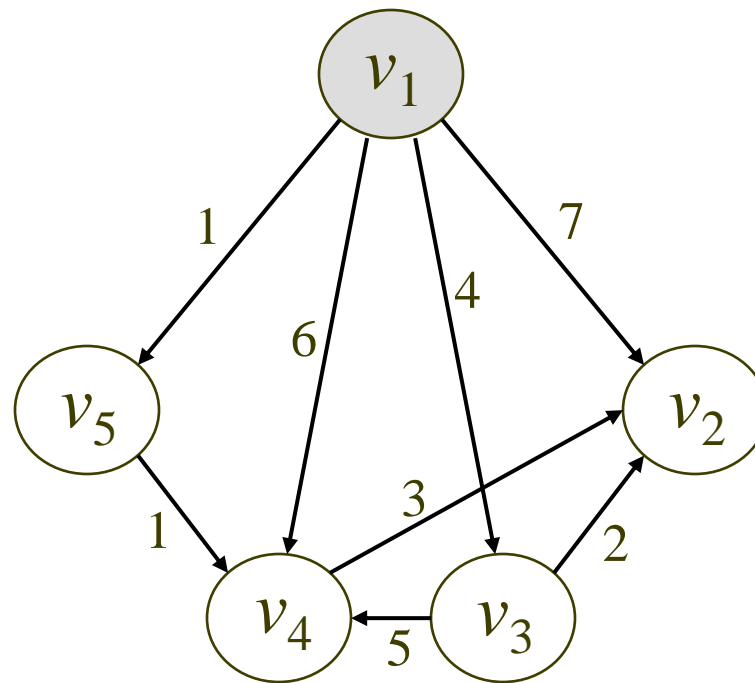


단일출발점 최단경로문제(single source shortest path problem) Dijkstra의 알고리즘(1959)

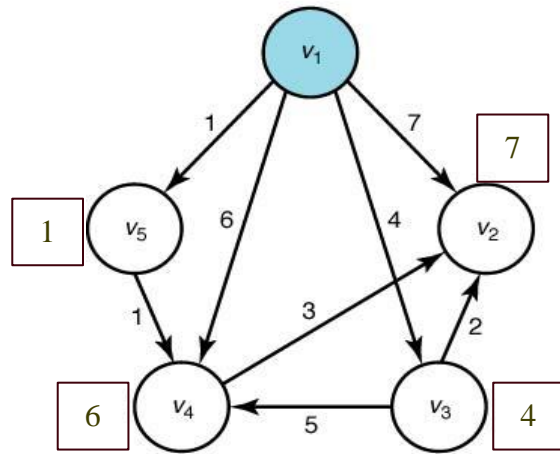
- 가중치가 있는 방향성 그래프에서 한 특정 정점에서 다른 모든 정점으로 가는 최단경로 구하는 문제.
- 알고리즘:
 1. $F := \phi$;
 2. $Y := \{v_1\}$;
 3. 최종해답을 얻지 못하는 동안 다음 절차를 계속 반복
 - (a) 선정 절차/적정성 점검: $V - Y$ 에 속한 정점 중에서, v_1 에서 Y 에 속한 정점 만을 거쳐서 최단경로가 되는 정점 v 를 선정
 - (b) 그 정점 v 를 Y 에 추가.
 - (c) v 에서 F 로 이어지는 최단경로 상의 이음선을 F 에 추가.
 - (d) 해답 점검: $Y = V$ 가 되면, $T = (V, F)$ 가 최단경로를 나타내는 그래프



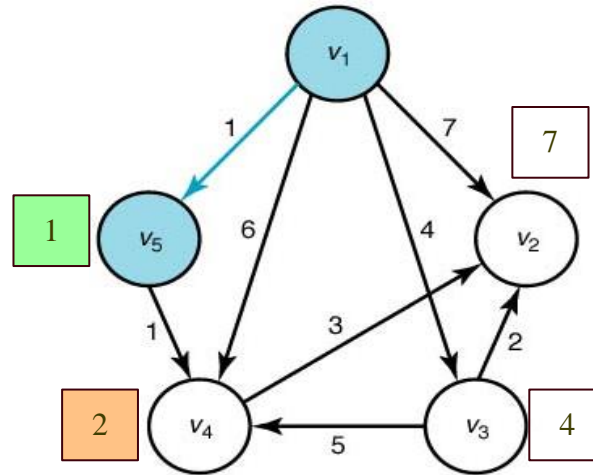
예



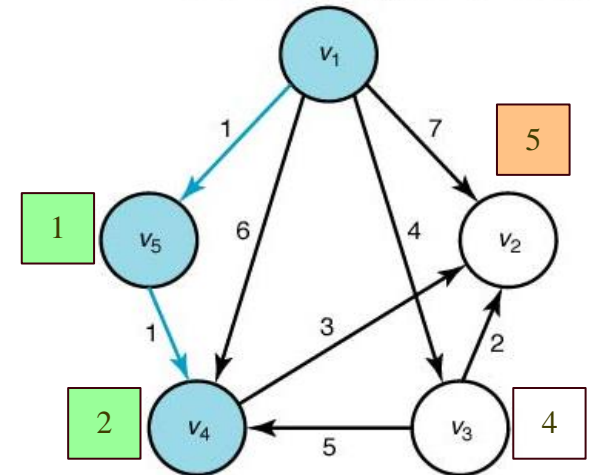
Compute shortest paths from v_1 .



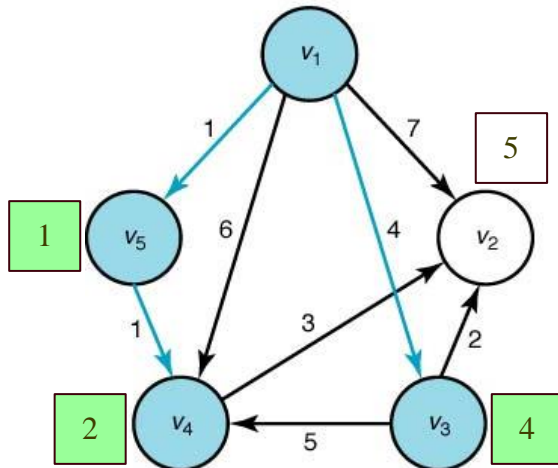
1. Vertex v_5 is selected because it is nearest to v_1 .



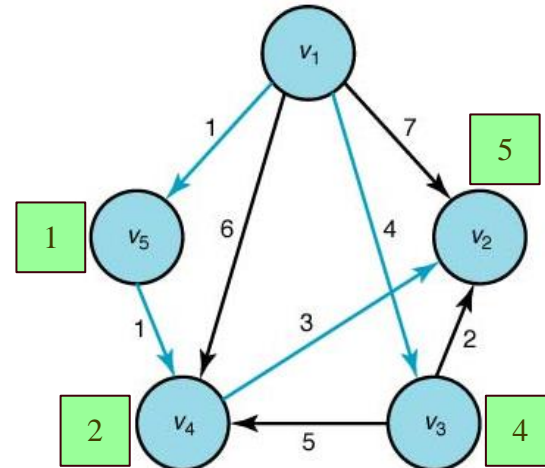
2. Vertex v_4 is selected because it has the shortest path from v_1 using only vertices in $\{v_5\}$ as intermediates.



3. Vertex v_3 is selected because it has the shortest path from v_1 using only vertices in $\{v_4, v_5\}$ as intermediates.

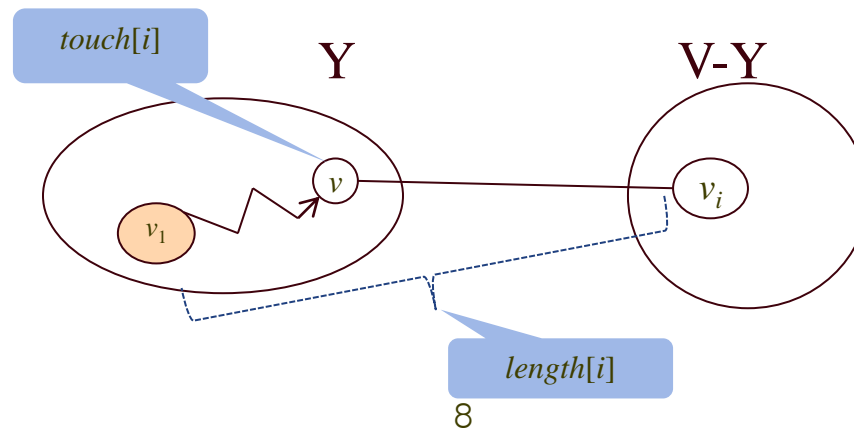


4. The shortest path from v_1 to v_2 is $[v_1, v_5, v_4, v_2]$.



Dijkstra의 알고리즘

- 추가적으로 $\text{touch}[1..n]$ 과 $\text{length}[1..n]$ 배열 유지
 - ✓ $\text{touch}[i] = Y$ 에 속한 정점들만 중간에 거치도록 하여 v_1 에서 $v_i \in V - Y$ 로 가는 현재 최단경로상의 마지막 이음선을 $\langle v, v_i \rangle$ 라고 할 때, Y 에 속한 정점 v
 - ✓ $\text{length}[i] = Y$ 에 속한 정점들만 중간에 거치도록 하여 v_1 에서 v_i 로 가는 현재 최단경로의 길이



```

void dijkstra(int n, const number W[][], set_of_edges& F) {
    index i, vnear;
    edge e;
    index touch[2..n];
    number length[2..n];
    F =  $\phi$ ;
    for(i=2; i <= n; i++) {
        touch[i] = 1;
        length[i] = W[1][i];
    }
    repeat(n-1 times) {
        min =  $\infty$ ;
        for(i=2; i <= n; i++)
            if (0 <= length[i] < min) {
                min = length[i];
                vnear = i;
            }
        e = (touch[vnear], vnear): 이음선;
        e를 F에 추가;
        for(i=2; i <= n; i++)
            if (length[vnear] + W[vnear][i] < length[i]) {
                length[i] = length[vnear] + W[vnear][i];
                touch[i] = vnear;
            }
        length[vnear] = -1;
    }
}

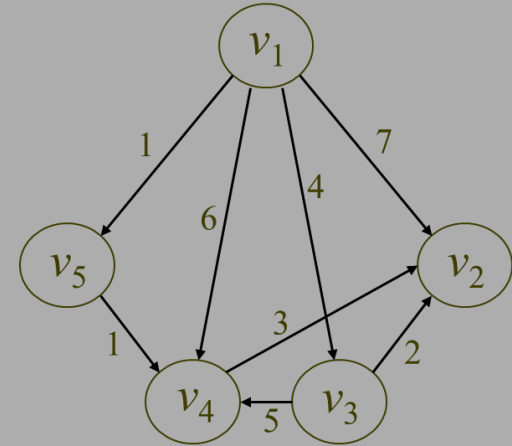
```



[실습프로그램] Dijkstra 알고리즘

```
inf=1000
w=[[0,7,4,6,1],[inf,0,inf,inf,inf],
   [inf,2,0,5,inf],[inf,3,inf,0,inf],[inf,inf,inf,1,0]]
n=5
f=set()
touch=n*[0]
length=n*[0]

for i in range(1,n):
    length[i]=w[0][i]
```



Dijkstra algorithm 구현

```
print(f)
```

```
>>>
{(3, 1), (0, 2), (4, 3), (0, 4)}
>>>
```



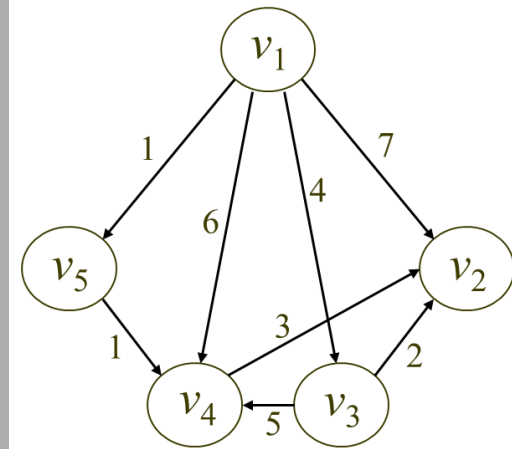
[실습프로그램]

- 각 노드 별 최단 거리를 저장하는 방법
- save_length 배열에 저장

```
inf=1000
w=[[0,7,4,6,1],[inf,0,inf,inf,inf],
   [inf,2,0,5,inf], [inf,3,inf,0,inf], [inf,inf,inf,1,0]]
n=5
f=set()
touch=n*[0]
length=n*[0]
save_length=n*[0]
```

구현

```
print(save_length)
```



```
{(3, 1), (0, 2), (4, 3), (0, 4)}
[0, 5, 4, 2, 1]
>>>
```



[실습프로그램]

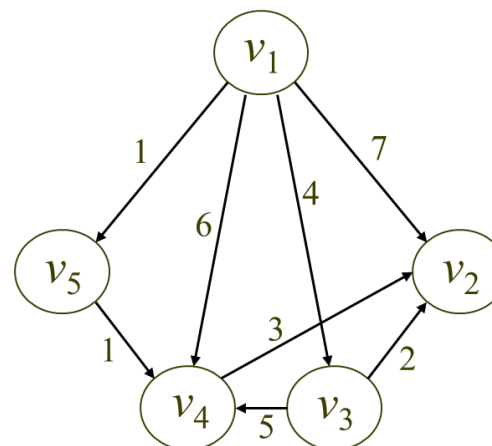
- length[]를 update 하는 최대 횟수를 확인해 본다.
- 최대 횟수(NoC)는 "총 아크 수 - 출발점에 연결된 아크 수"

```
inf=1000
w=[[0,7,4,6,1],[inf,0,inf,inf,inf],
   [inf,2,0,5,inf],[inf,3,inf,0,inf],[inf,inf,inf,1,0]]
n=5
f=set()
touch=n*[0]
length=n*[0]
NoC=0
```

```
for i in range(1,n):
    length[i]=w[0][i]
```

Dijkstra algorithm 구현

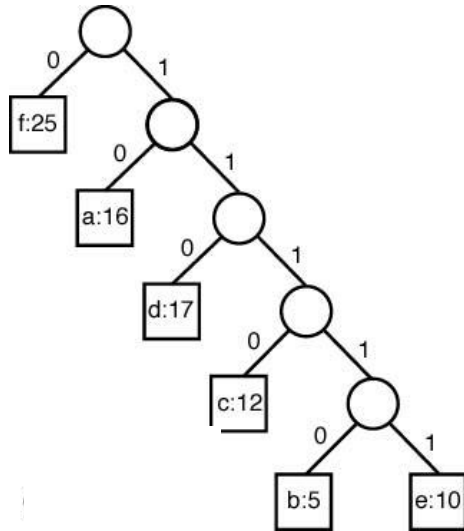
```
print(f)
print(NoC)
```



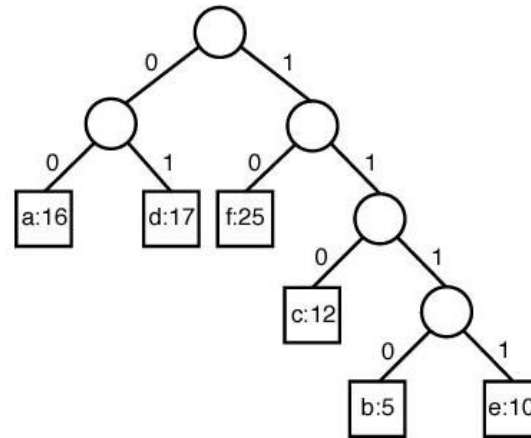
```
0 4
4 3
0 2
3 1
{(3, 1), (0, 2), (4, 3), (0, 4)}
4
>>>
```



(A)



(B)

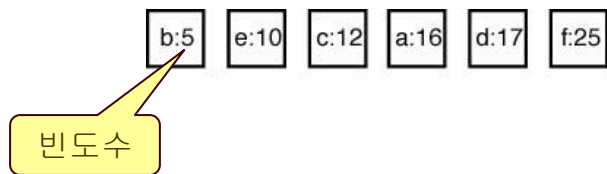


prefix 코드의 예. (B)는 Huffman code

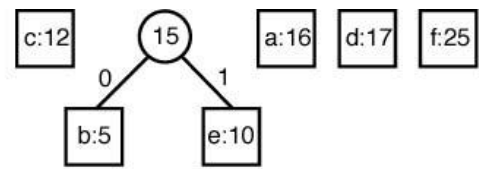
● Huffman 코드 구축 방법

- (1) 빈도수를 데이터로 갖는 n 개의 노드를 생성
- (2) 빈도수의 합이 최소가 되는 노드를 merge 시켜 이진트리로 구축
- (3) 모든 노드가 하나의 이진트리가 될 때까지 단계(2)를 반복

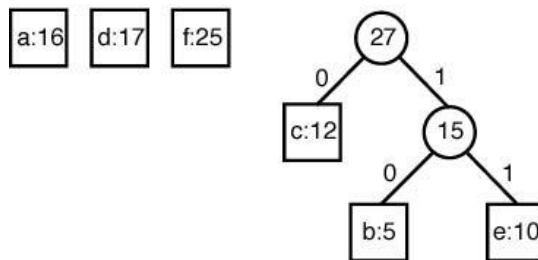




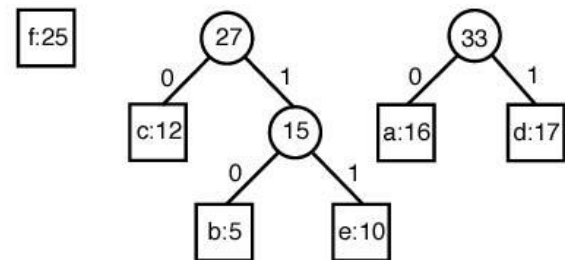
(0)



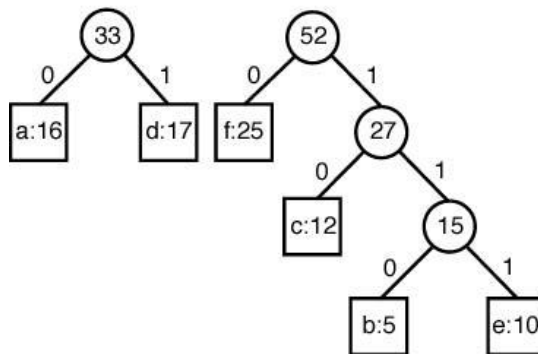
(1)



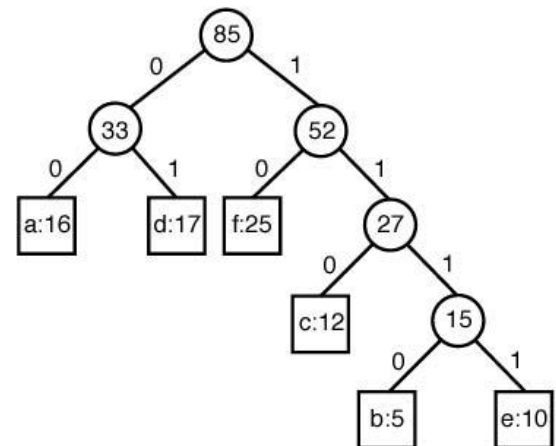
(2)



(3)



(4)



(5)

Huffman 코드 구축 단계



● Huffman code 생성 알고리즘

- 우선순위 큐(Priority Queue) 사용 - Heap
- 자료 구조

```
struct nodetype {  
    char symbol;  
    int frequency;  
    nodetype* left;  
    nodetype* right;  
};
```

● 초기

- ✓ 우선순위 큐 PQ에서 nodetype 레코드를 가리키는 포인터 n 개를 생성.
- ✓ PQ의 각 포인터 p 에 대해

$p \rightarrow \text{symbol} = \text{문자}$

$p \rightarrow \text{frequency} = \text{문자의 빈도수}$

$p \rightarrow \text{left} = p \rightarrow \text{right} = \text{NULL}$

- 빈도수가 작을수록 우선순위가 높다

```
for (i=1; i <= n-1; i++) {  
    remove(PQ, p);  
    remove(PQ, q);  
    r = new nodetype;  
    r->left = p;  
    r->right = q;  
    r->frequency = p->frequency + q->frequency;  
    insert(PQ, r);  
}  
remove(PQ, r);  
return r;
```

- `remove(PQ, r)` : 우선순위큐에서 최대 우선순위 데이터 `r`을 제거
- $\Theta(n \lg n)$

defaultdict: 값이 주어지지 않았을 경우의 값을 부여

```
a=defaultdict(int)
print(a)

a["1stitem"]="BBB"
print(a["1stitem"])
print(a["2nditem"])

for x,y in a.items():
    print(x,y)

a=[[1,2]]
b=[[c,d]]
print(a+b)
```

```
defaultdict(<class 'int'>, {})
```



```
BBB
0
```



```
1stitem BBB
2nditem 0
```



```
[[1, 2], [3, 5]]
```

sorted & sort

- sorted: 원래의 순서는 바뀌지 않음
- sort: 원래의 순서가 바뀜

```
a=[ ['rr',5],['a',3],['a',1],['ccc',8]]  
print(sorted(a, key=lambda p: (p[0],p[1])))  
print(a)
```


```
[['a', 1], ['a', 3], ['ccc', 8], ['rr', 5]]  
[['rr', 5], ['a', 3], ['a', 1], ['ccc', 8]]
```

```
a=[ ['rr',5],['a',3],['a',1],['ccc',8]]  
print(sorted(a, key=lambda p: (p[1],p[0])))
```

```
[['a', 1], ['a', 3], ['rr', 5], ['ccc', 8]]
```

```
a=[ ['rr',5],['a',3],['a',1],['ccc',8]]  
a.sort(key=lambda p: (p[0],p[1]))  
print(a)
```

```
[['a', 1], ['a', 3], ['ccc', 8], ['rr', 5]]
```

- 
- 첫 번째 원소를 기준으로 정렬
 - 첫 번째 원소가 동일할 경우, 두 번째 원소를 기준으로 정렬

```
a=[ (3,2), (9,5), (3,6) ]  
print(sorted(a, key=lambda p: p[0]-p[1]))
```

```
[(3, 6), (3, 2), (9, 5)]
```

Huffman code 1

[실습프로그램] Huffman code

```
import heapq
from collections import defaultdict

def encode(frequency):
    heap = [[weight, [symbol, '']] for symbol, weight in frequency.items()]
    print(heap)
    heapq.heapify(heap)
    while len(heap) > 1:
        lo = heapq.heappop(heap)
        hi = heapq.heappop(heap)
        for pair in lo[1:]:
            pair[1] = '0' + pair[1]
        for pair in hi[1:]:
            pair[1] = '1' + pair[1]

        heapq.heappush(heap, [lo[0] + hi[0]] + lo[1:] + hi[1:])
    return sorted(heapq.heappop(heap)[1:], key=lambda p: (len(p[-1]), p))

data = "The frog at the bottom of the well drifts off into the great ocean"
frequency = defaultdict(int)

for symbol in data:
    frequency[symbol] += 1

huff = encode(frequency)
print ("Symbol".ljust(10) + "Weight".ljust(10) + "Huffman Code")
for p in huff:
    print (p[0].ljust(10) + str(frequency[p[0]]).ljust(10) + p[1])
```

Huffman code 1 output

```
>>>
[[1, ['T', '']], [4, ['h', '']], [7, ['e', '']], [13, [' ', '']], [5,
['f', '']], [3, ['r', '']], [7, ['o', '']], [2, ['g', '']], [3, ['a',
'']], [9, ['t', '']], [1, ['b', '']], [1, ['m', '']], [1, ['w', '']],
[2, ['l', '']], [1, ['d', '']], [2, ['i', '']], [1, ['s', '']], [2,
['n', '']], [1, ['c', '']]]
```

Symbol	Weight	Huffman Code
	13	111
e	7	001
o	7	010
t	9	110
a	3	0000
f	5	1011
h	4	1000
r	3	0001
g	2	01111
i	2	10010
l	2	10011
n	2	10101
w	1	01100
T	1	011010
b	1	011011
c	1	011100
d	1	011101
m	1	101000
s	1	101001

```
>>>
```

[작은 문제 예] 1/2

[[freq, [symbol,code]], [freq, [symbol,code]], ...]

```
import heapq
from collections import defaultdict
```

```
def encode(frequency):
```

```
    heap = [[weight, [symbol, '']] for symbol, weight in frequency.items()]
```

```
    print(heap)
```

```
    heapq.heapify(heap)
```

```
    while len(heap) > 1:
```

```
        left = heapq.heappop(heap)
```

```
        print("LLL",left,left[1:])
```

```
        right = heapq.heappop(heap)
```

```
        print("HHH",right)
```

```
        for pair in left[1:]:
```

```
            print("PPP",pair, pair[1])
```

```
            pair[1] = '0' + pair[1]
```

```
        for pair in right[1:]:
```

```
            pair[1] = '1' + pair[1]
```

```
        print("OOO",[left[0] + right[0]])
```

```
        print("RRR", left[1:] + right[1:])
```

```
        print("SSS",[left[0] + right[0] + left[1:] + right[1:]])
```

```
        heapq.heappush(heap, [left[0] + right[0] + left[1:] + right[1:]])
```

```
#     print("WWW",heapq.heappop(heap) [1] [1])
```

```
#     print("WWW",heapq.heappop(heap) [1:])
```

```
#     print("WWW",heapq.heappop(heap) )
```

```
    return sorted(heapq.heappop(heap) [1:], key=lambda p: (len(p[1]),p))
```

[[1, ['f', '']], [1, ['u', '']], [2, ['L', '']]]

LLL [1, ['f', '']] [['f', '']]

HHH [1, ['u', '']]

PPP ['f', '']

OOO [2]

RRR [['f', '0'], ['u', '1']]

SSS [2, ['f', '0'], ['u', '1']]

WWW 00

WWW [['f', '00'], ['u', '01'], ['L', '1']]

WWW [4, ['f', '00'], ['u', '01'], ['L', '1']]

각 코드의 길이순, 그 자체로 정렬

heap 의 최종 1개의 데이터 모양

21

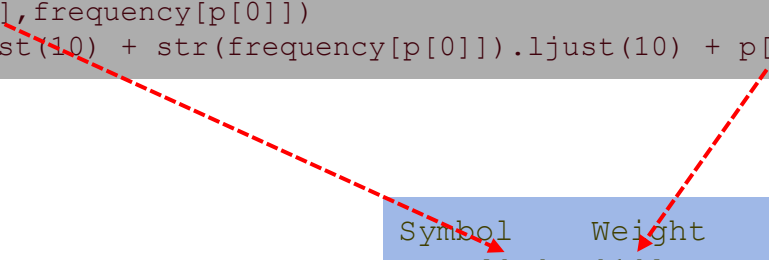
p

[작은 문제 예] 2/2

```
data="fuLL"
frequency = defaultdict(int)

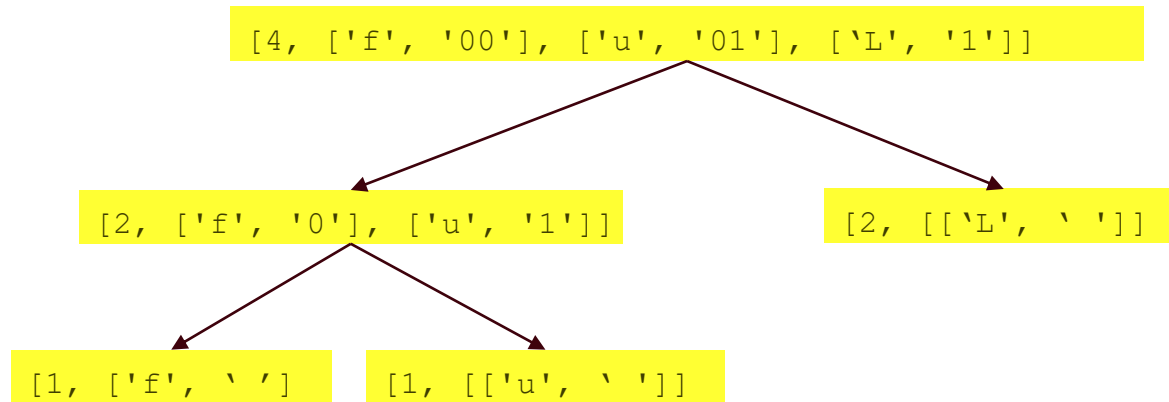
for symbol in data:
    frequency[symbol] += 1

huff = encode(frequency)
print ("Symbol".ljust(10) + "Weight".ljust(10) + "Huffman Code")
for p in huff:
    print("AAA",p)
    print("KKK",p[0],frequency[p[0]])
    print (p[0].ljust(10) + str(frequency[p[0]]).ljust(10) + p[1])
```



Symbol	Weight	Huffman Code
AAA ['L', '1']		
KKK L 2		
l	2	1
AAA ['f', '00']		
KKK f 1		
f	1	00
AAA ['u', '01']		
KKK u 1		
u	1	01

생성 트리 모습



```

[[1, ['f', '']], [1, ['u', '']], [2, ['L', '']]
LLL [1, ['f', '']] [['f', '']]
HHH [1, ['u', '']]
PPP ['f', '']
OOO [2]
RRR [['f', '0'], ['u', '1']]
SSS [2, ['f', '0'], ['u', '1']]
LLL [2, ['f', '0'], ['u', '1']] [['f', '0'], ['u', '1']]
HHH [2, ['L', '']]
PPP ['f', '0'] 0
PPP ['u', '1'] 1
OOO [4]
RRR [['f', '00'], ['u', '01'], ['L', '1']]
SSS [4, ['f', '00'], ['u', '01'], ['L', '1']]
Symbol      Weight      Huffman Code
L           2           1
f           1           00
u           1           01
>>>
  
```

- `del` : 리스트의 한 원소를 삭제

```
a=[1,2,3,4]
del a[1]
print(a)
```

```
[1, 3, 4]
>>>
```

- 리스트의 원소가 `tuple`일 때 정렬 기준을 제시

```
a=[(8, 'D'), (1, 'B'), (2, 'A'), (4, 'M')]
a.sort(key=lambda x :x[0])
print(a)
a.sort(key=lambda x :x[1])
print(a)
```

```
[(1, 'B'), (2, 'A'), (4, 'M'), (8, 'D')]
[(2, 'A'), (1, 'B'), (8, 'D'), (4, 'M')]
>>>
```

```
c=3
d=5
print((lambda c,d: c*d)(c,d))
```

```
15
>>>
```

- 리스트의 `append` 와 `extend` 차이

```
a=["A","B","C"]
a.append("D")
print(a)
```

```
['A', 'B', 'C', 'D']
```

```
a=["A","B","C"]
a.append(["A"])
print(a)
```

```
['A', 'B', 'C', ['D']]
```

```
a=["A","B","C"]
a.extend(["D"])
print(a)
```

```
['A', 'B', 'C', 'D']
```

- `split`: 문자열을 기준에 따라 나눈다. 결과는 `list` 에 저장.

```
a="this #is 10:30"  
print(a.split())  
print(a.split(" ",1))  
print(a.split(" ",2))  
print(a.split("#",1))
```

```
['this', '#is', '10:30']  
['this', '#is 10:30']  
['this', '#is', '10:30']  
['this ', 'is 10:30']
```

자르는 기준

자르는 횟수, 1회 자르면 2개의 부분이 된다.

- `isinstance`: `class type` 을 확인한다..

```
a=3  
print(isinstance(a, int))  
print(isinstance(a, str))
```

```
True  
False
```

Huffman code 2 (1/4)

```
class Queue(object):

    def __init__(self):
        self._q = []

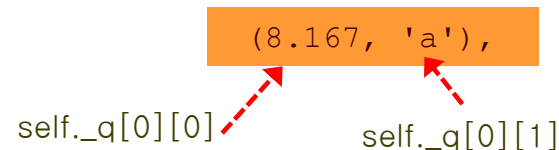
    def put(self, x):
        self._q.append(x)
        self._q.sort(key=lambda x: x[0])

    def get(self):
        x = self._q[0]
        del self._q[0]
        return x

    def qsize(self):
        return len(self._q)

class HuffmanNode(object):

    def __init__(self, left, right):
        self.left = left
        self.right = right
```



Huffman code 2 (2/4)

```
freq = [  
    (8.167, 'a'), (1.492, 'b'), (2.782, 'c'), (4.253, 'd'),  
    (12.702, 'e'), (2.228, 'f'), (2.015, 'g'), (6.094, 'h'),  
    (6.966, 'i'), (0.153, 'j'), (0.747, 'k'), (4.025, 'l'),  
    (2.406, 'm'), (6.749, 'n'), (7.507, 'o'), (1.929, 'p'),  
    (0.095, 'q'), (5.987, 'r'), (6.327, 's'), (9.056, 't'),  
    (2.758, 'u'), (1.037, 'v'), (2.365, 'w'), (0.150, 'x'),  
    (1.974, 'y'), (0.074, 'z') ]  
  
def create_tree(frequencies):  
    p = Queue()  
    for value in frequencies:      # 1. Create a leaf node for each symbol  
        p.put(value)              #    and add it to the priority queue  
    while p.qsize() > 1:          # 2. While there is more than one node  
        l, r = p.get(), p.get()   # 2a. remove two highest nodes  
        node = HuffmanNode(l, r) # 2b. create internal node with children  
        p.put((l[0]+r[0], node)) # 2c. add new node to queue  
    return p.get()                # 3. tree is complete - return root node  
  
node = create_tree(freq)
```


Huffman code 2 (3/4)

```
def side_by_side(a, b, w):
    a = a.split("\n") #결과를 list에 저장
    b = b.split("\n")
    n1 = len(a)
    n2 = len(b)
    if n1 < n2:
        a.extend([" " * len(a[0])] * (n2 - n1))
    else:
        b.extend([" " * len(b[0])] * (n1 - n2))
    r = [" " * len(a[0]) + " ^ " + " " * len(b[0])]
    r += ["/" + "-" * (len(a[0]) - 1) + "%7.3f" % w + "-" * (len(b[0]) - 1) + "\\"]
    for l1, l2 in zip(a, b):
        r.append(l1 + " " + l2)
    return "\n".join(r)

def print_tree(node):
    w, n = node
    if isinstance(n, str):
        return "%s = %.3f" % (n, w)
    else:
        l = print_tree(n.left)
        r = print_tree(n.right)
        return side_by_side(l, r, w)
```

Huffman code 2 (4/4)

```
print(print_tree(node))

def walk_tree(node, prefix="", code={}):
    w, n = node
    if isinstance(n, str):
        code[n] = prefix
    else:
        walk_tree(n.left, prefix + "0")
        walk_tree(n.right, prefix + "1")
    return(code)

code = walk_tree(node)
for i in sorted(freq, reverse=True):
    print (i[1], '{:6.2f}'.format(i[0]), code[i[1]])
```

Huffman code 2 output

윗 부분 생략

e	12.70	100
t	9.06	000
a	8.17	1110
o	7.51	1101
i	6.97	1011
n	6.75	1010
s	6.33	0111
h	6.09	0110
r	5.99	0101
d	4.25	11111
l	4.03	11110
c	2.78	01001
u	2.76	01000
m	2.41	00111
w	2.37	00110
f	2.23	00100
g	2.02	110011
y	1.97	110010
p	1.93	110001
b	1.49	110000
v	1.04	001010
k	0.75	0010111
j	0.15	001011011
x	0.15	001011010
q	0.10	001011001
z	0.07	001011000

>>>

[작은 문제의 output]

```
freq = [  
    (4, 'a'), (3, 'b'), (3, 'c'), (4, 'd'),  
    ]
```

```
/----- 14.000-----\  
      ^                               ^  
/----- 6.000-----\   /----- 8.000-----\  
b = 3.000      c = 3.000   a = 4.000      d = 4.000  
d   4.00 11  
a   4.00 10  
c   3.00 01  
b   3.00 00
```