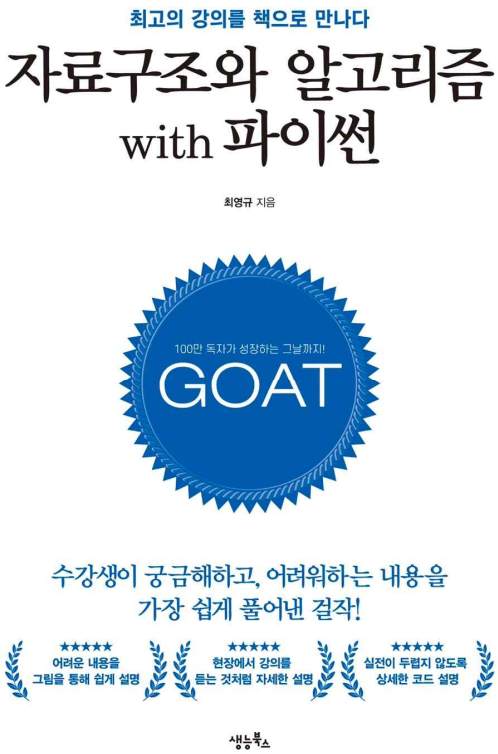
자료구조와 알고리즘 with 파이썬

연습문제 해답

2023. 07.



1장, 해답

1.1 push(A)→push(B)→pop()→push(C)→pop()→push(D)→pop()→pop()

1.2

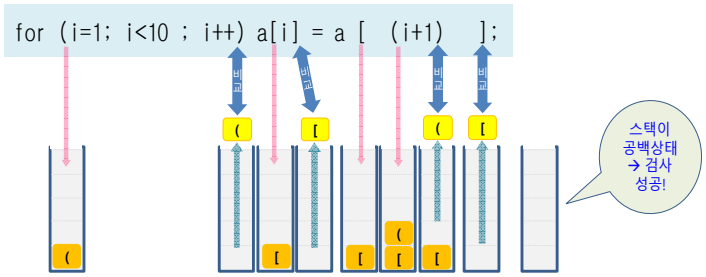
```
def clear( self ) :
   self.top = -1
```

1.3

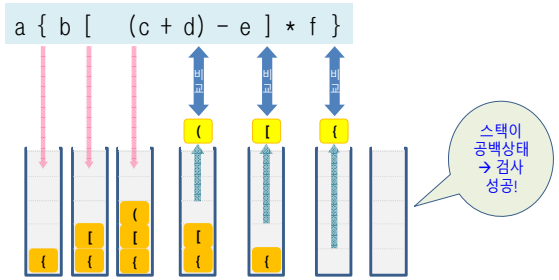
```
def display( self ) :
    for i in range(0, self.top+1) :
        print(self.array[i], end=' ')
    print()
```

1.4

(1)



(2)



1.5 18, 12, 9, 0

1.6

```
def printReverse(msg, len) :
    if len > 0 :
        print(msg[len-1])
        printReverse(msg, len-1)
```


2장, 해답

2.1

front=7, rear=2: 3개

front=2, rear=7: 5개

2.2

```
    def clear( self ) :     # 공백 상태로 초기화
        self.front = 0
        self.rear = 0
```

2.3

9, 12, 15, 18

2.4

```
enqueue(q, x):
  S1이 공백 상태가 될 때까지 S1의 모든 요소를 꺼내서(pop) S2에 삽입(push)한다.
  x를 S1에 삽입(push)한다.
  S2의 모든 요소를 꺼내 다시 S1에 넣는다.

dequeue(q):
  S1이 공백이면 언더플로 에러.
  S1이 공백이 아니면 상단 요소를 꺼내 반환한다.
```

2.5

```
# 큐를 이용한 피보나치 수 구하기
from CircularQueue import CircularQueue
Q = CircularQueue()
Q.enqueue(0)
Q.enqueue(1)
print("F(0) = 0")
print("F(1) = 1")

for i in range(2, 20) :
    val = Q.dequeue() + Q.peek()
```

```
Q.enqueue(val)
print("F(%d) ="%i, val);
```


3장, 해답

3.1

lst가 파이썬 리스트라 하자. 예를 들어, lst = [1, 2, 3].

- delete(pos): lst.pop(pos)
- getEntry(pos): lst[pos]
- isEmpty( ): len(lst)==0
- size( ): len(lst)

3.2

```
    def find(self, e) :
        node = self.head
        pos = 0            # head의 인덱스를 0이라 가정
        while node is not None:
            if node.data == e :
                return pos
            node = node.link
        return -1
```

3.3

```
class Node:
    def __init__ (self, elem, link=None):
        self.data = elem
        self.link = link
class LinkedStack :
    def __init__( self ):
        self.top = None

    def isEmpty( self ):
        return self.top == None

    def push( self, item ):
        self.top = Node(item, self.top)
```

```
    def peek( self ):
        if not self.isEmpty():
            return self.top.data

    def pop( self ):
        if not self.isEmpty():
            data = self.top.data
            self.top = self.top.link
            return data
```

3.4

```
class Node:
    def __init__ (self, elem, link=None):
        self.data = elem
        self.link = link
class LinkedQueue:
    def __init__( self ):
        self.front = None
        self.rear = None

    def isEmpty( self ): return self.front == None
    def isFull( self ): return False

    def enqueue( self, item ):
        node = Node(item, None)
        if self.isEmpty() :
            self.front = self.rear = node
        else :
            self.rear.link = node
            self.rear = node

    def dequeue( self ):
        if not self.isEmpty():
            data = self.front.data
            if self.front == self.rear :
```



```
                self.front = self.rear = None
            else:
                self.front = self.front.link
            return data

    def peek( self ):
        if not self.isEmpty():
            return self.front.data
```

3.5

후단 삭제 연산을 위해서는 rear가 가리키는 노드를 삭제하고, rear를 이 노드 바로 앞 노드(선행 노드)로 이동해야 한다. 그런데, 단순 연결 구조에서 어떤 노드는 후속 노드의 링크는 있지만 선행 노드의 링크가 없다. 따라서 rear의 선행 노드를 찾기 위해 시작노드부터 모든 노드를 검사해야 한다. 따라서 단순 연결 구조로 덱을 구현하면 후단 삭제가 비효율적으로 처리된다.

3.6

```
class Node:
    def __init__(self, data, prev=None, next=None):
        self.data = data
        self.prev = prev
        self.next = next

class CirDblLinkedList:
    def __init__( self ):
        self.head = None

    def isEmpty( self ): return self.head == None
    def isFull( self ): return False

    def getNode(self, pos) :
        if pos < 0 : return None
        node = self.head;
        while pos > 0 and node != None :
            node = node.next
```

```
        pos -= 1
    return node

def getEntry(self, pos) :
    node = self.getNode(pos)
    if node == None : return None
    else : return node.data

def insert(self, pos, elem) :
    if self.isEmpty() :    # 공백이면 무조건 맨 앞에 추가
        self.head = Node(elem)
        self.head.prev = self.head.next = self.head

    else :
        p = self.getNode(pos)
        node = Node(elem, p.prev, p)
        node.prev.next = node
        node.next.prev = node
        if pos == 0 :
            self.head = node

def delete(self, pos) :
    if self.isEmpty():
        return None

    p = self.getNode(pos)
    if p == None :
        return None

    data = p.data   # 반환할 자료

    if self.head.next == self.head :    # 요소가 하나뿐인 경우
        self.head = None
    else:
        p.prev.next = p.next
```



```
        p.next.prev = p.prev
        if p == self.head :
            self.head = p.next
    return data
```

4장. 해답

4.1

```
class N_Node:
    def __init__ (self, elem):
        self.data = elem
        self.link = []

    def addChild(self, node) :
        self.link.append(node)
```

4.2

```
class CS_Node:
    def __init__ (self, elem, child=None, sibling=None):
        self.data = elem
        self.child = child
        self.sibling = sibling
```

4.3

```
def count_leaf(n) :
    if n is None :
       return 0
    elif n.left is None and n.right is None :
       return 1
    else :
       return count_leaf(n.left) + count_leaf(n.right)
```

4.4

영어 대문자와 숫자를 포함하는 모스 코드 테이블과 인코딩 함수.
다른 함수들은 본문에서와 동일함.

```
table =[('A', '.-'),   ('B', '-...'), ('C', '-.-.'), ('D', '-..'),
       ('E', '.'),     ('F', '..-.'), ('G', '--.'),  ('H', '....'),
       ('I', '..'),    ('J', '.---'), ('K', '-.-'),  ('L', '.-..'),
       ('M', '--'),    ('N', '-.'),   ('O', '---'),  ('P', '.--.'),
       ('Q', '--.-'), ('R', '.-.'),   ('S', '...'),  ('T', '-'),
```



```
       ('U', '..-'),  ('V', '...-'), ('W', '.--'),  ('X', '-..-'),
       ('Y', '-.--'), ('Z', '--..'),
       ('0', '-----'), ('1', '.----'), ('2', '..---'), ('3', '...--'),
       ('4', '....-'),  ('5', '.....'), ('6', '-....'), ('7', '--...'),
       ('8', '---..'),  ('9', '----.') ]

# 코드 4.10: 모스코드 인코딩 함수 수정
def encode(ch):
    idx = ord(ch.upper())-ord('A')
    if 0 <= idx < 26 :  # 영어 대문자 A~Z
       return table[idx][1]

    idx = ord(ch)-ord('0')
    if 0 <= idx < 10 :  # 숫자 0~9
       return table[26+idx][1]
```

4.5

```
# 연산자의 우선순위 계산 함수
def precedence (op):
    if  (op=='(' or op==')') : return 0;
    elif (op=='+' or op=='-') : return 1;
    elif (op=='*' or op=='/') : return 2;
    else : return -1

# 중위 표기 수식의 후위식 변환
def Infix2Postfix( expr ):
    s = ArrayStack(100)
    output = []

    for term in expr :
       if term in '(' :
          s.push('(')

       elif term in ')' :
          while not s.isEmpty() :
```

```
            op = s.pop()
            if op=='(' :
                break;
            else :
                output.append(op)

    elif term in "+-*/" :
        while not s.isEmpty() :
            op = s.peek()
            if( precedence(term) <= precedence(op)):
                output.append(op)
                s.pop()
            else: break
        s.push(term)

    else :                  # 피연산자
        output.append(term)

while not s.isEmpty() :
    output.append(s.pop())

return output
```


5장, 해답

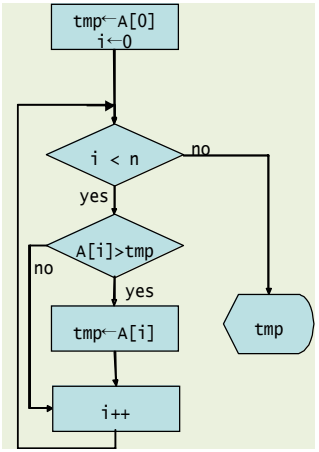
5.1

자연어 표현

find_max(A)

1. 배열 A의 첫 번째 요소를 변수 tmp에 복사한다.
2. 배열 A의 다음 요소들을 차례대로 tmp와 비교하여, 더 크면 그 값을 tmp로 복사한다.
3. 배열 A의 모든 요소를 비교했으면 tmp를 반환한다.

흐름도 표현



유사코드 표현

find_max(A)
tmp ← *A*[0];
for *i*←1 to size(A) do
    if *tmp* < *A*[*i*] then
        tmp ← *A*[*i*]
return tmp

파이썬 표현

```
def find_max( A ):
    tmp = A[0]
    for item in A :
        if item > tmp :
            tmp = item
    return tmp
```

5.2

$O(1), O(\log n), O(n), O(nlogn), O(n^2), O(n^3), O(2^n), O(n!)$

5.3

$O(\log_2 n)$

5.4

decode_simple()은 테이블의 모든 요소를 순서대로 비교하므로 $O(n)$임.
decode()는 결정 트리의 루트 노드에서 아래로 탐색하는데, 최악의 경우에도 트리의 높이 이상은 탐색하지 않음. 따라서 트리의 높이에 비례하는 시간이 걸림. 결정 트리가 완전 이진 트리의 형태를 갖는다고 가정하면 시간 복잡도는 $O(\log_2 n)$임.

6장, 해답

6.1

[6, 3, 9, 1, 2, 7]
[1, 3, 9, 6, 2, 7]
[1, 2, 9, 6, 3, 7]
[1, 2, 3, 6, 9, 7]
[1, 2, 3, 6, 9, 7]
[1, 2, 3, 6, 7, 9]

6.2

[6, 3, 9, 1, 2, 7]
[1, 6, 3, 9, 2, 7]
[1, 2, 6, 3, 9, 7]
[1, 2, 3, 6, 9, 7]
[1, 2, 3, 6, 9, 7]
[1, 2, 3, 6, 7, 9]

6.3

[6, 3, 9, 1, 2, 7]
[6, 3, 2, 1, 9, 7] 피벗 6, 조건에 맞지 않은 2와 9 교환
[1, 3, 2, 6, 9, 7] 한번 스캔 완료. 피벗을 제 위치로 이동(1과 교환)
[1, 3, 2, 6, 7, 9] [1, 3, 2] -> 피벗이 가장 왼쪽. [9, 7] -> 피벗 9가 제 위치로
[1, 2, 3, 6, 7, 9] [3, 2] -> [2, 3] 정렬 완료

6.4 (2)

6.5

[10, 123, 56, 636, 992, 119, 234, 76, 82, 345, 567, 432]
step 1 [10, 992, 82, 432, 123, 234, 345, 56, 636, 76, 567, 119]
step 2 [10, 119, 123, 432, 234, 636, 345, 56, 567, 76, 82, 992]
step 3 [10, 56, 76, 82, 119, 123, 234, 345, 432, 567, 636, 992]

7장. 해답

7.1

```
def search(self, elem) :
    node = self.head
    while node != None :
        if node.data == elem :
            return node
        node = node.link
    return None
```

7.2

```
def search_m2f(self, elem) :
    if self.isEmpty() : return None
    if self.head.data == elem :
        return self.head

    before = self.head
    while before.link != None :
        if before.link.data == elem :
            n = before.link
            before.link = n.link
            n.link = self.head
            self.head = n
            return n
        before = before.link
    return None
```

7.3 32 탐색

[3, 5, 8, 15, 16, 19, 22, 25, 27, 31, 32, 36, 39, 40, 43, 45]
[~~3, 5, 8, 15, 16, 19, 22, 25,~~ 27, 31, 32, 36, 39, 40, 43, 45]
[~~3, 5, 8, 15, 16, 19, 22, 25,~~ 27, 31, 32, ~~36, 39, 40, 43, 45~~]
[~~3, 5, 8, 15, 16, 19, 22, 25, 27, 31~~, 32, ~~36, 39, 40, 43, 45~~]
[~~3, 5, 8, 15, 16, 19, 22, 25, 27, 31~~, 32, ~~36, 39, 40, 43, 45~~]

25 → 36 → 31 → 32 탐색 성공

7.4

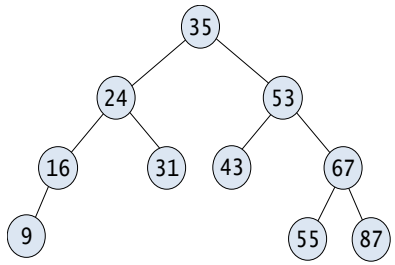
좋은 자료의 예 [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
나쁜 자료의 예 [10, 11, 12, 13, 14, 15, 16, 17, 18, 100]

7.5

```
def search_bst_iter(n, key) :
    while n != None :
        if key == n.key:
            return n
        elif key < n.key:
            n = n.left
        else:
            n = n.right
    return None
```

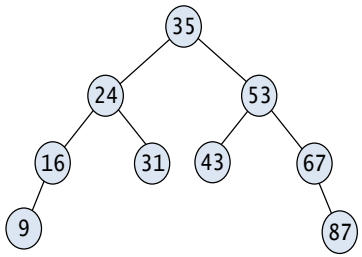
7.6

(a)

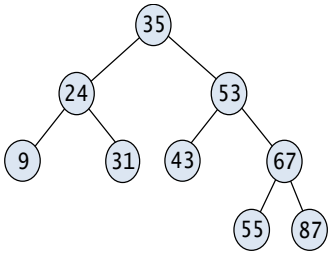


(b) 35 → 24 → 16 → 9

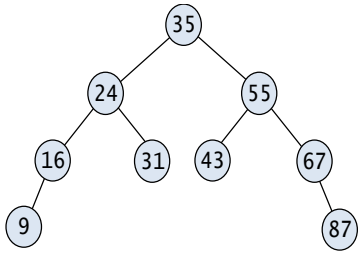
(c)



(d)



(e)



(f) 노드가 10개이므로 최소 높이 4 이상의 트리가 만들어짐. 현재의 입력 순서도 최소 높이 4인 트리를 만들어 줌.

35, 24, 16, 31, 53, 67, 43, 87, 55, 9

(g) 크기순으로 삽입. 예를 들어,

9, 16, 24, 31, 35, 43, 53, 55, 67, 87

8장, 해답

8.1

```
vertex =  [ 'A','B','C','D','E','F','G','H']
adjMat = [[  0,  1,  1,  0,  0,  0,  0,  0 ],
          [  1,  0,  0,  1,  0,  0,  0,  0 ],
          [  1,  0,  0,  1,  1,  0,  0,  0 ],
          [  0,  1,  1,  0,  0,  1,  0,  0 ],
          [  0,  0,  1,  0,  0,  0,  1,  1 ],
          [  0,  0,  0,  1,  0,  0,  0,  0 ],
          [  0,  0,  0,  0,  1,  0,  0,  1 ],
          [  0,  0,  0,  0,  1,  0,  1,  0 ]]
```

8.2

```
vertex =  [ 'A','B','C','D','E','F','G','H']
adjList = [[ 1, 2 ],     # 'A'의 인접정점 인덱스
           [ 0, 3 ],     # 'B'의 인접정점 인덱스
           [ 0, 3, 4 ],  # 'C'
           [ 1, 2, 5 ],  # 'D'
           [ 2, 6, 7 ],  # 'E'
           [ 3 ],        # 'F'
           [ 4, 7 ],     # 'G'
           [ 4, 6 ] ]    # 'H'
```

8.3

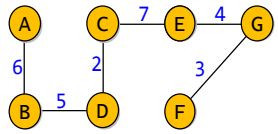
ABDCEGHF

8.4

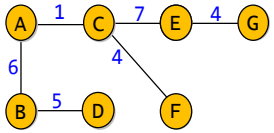
ABCDEFGH

8.5

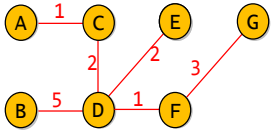
깊이우선탐색: 가중치 합 = 27



너비우선탐색: 가중치 합 = 27



8.6



9장, 해답

9.1

잔돈 액수 = 68
동전 종류 = [25, 10, 5, 1]
동전 개수 = [2, 1, 1, 3]
최소화할 수 있다.

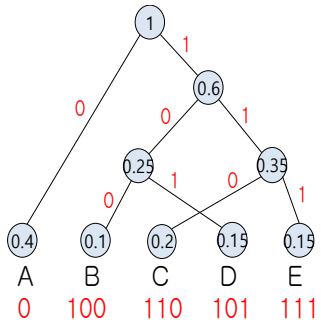
9.2

잔돈 액수 = 30
동전 종류 = [25, 10, 1]
동전 개수 = [1, 0, 5]
최소화할 수 없다.

9.3

A=(60만원, 10kg), B=(100만원, 20kg), C=(120만원, 30kg)
배낭의 용량: 50kg
무게당 가격 높은 순: A(6만원/kg) -> B(5만원/kg) -> C(4만원/kg)
부분적인배낭(50): A전부(10kg) + B전부(20kg) + C일부(20kg)
--> 60만원 + 100만원 + 120만원*2/3 = 240만원

9.4



10장, 해답

10.1

```
def quick_select_iter(a, left, right, k) :
    while left <= right :
        pivotIndex = partition(a, left, right)
        if pivotIndex == k - 1 :
            return a[pivotIndex]
        elif pivotIndex > k - 1 :
            right = pivotIndex - 1
        else :
            left = pivotIndex + 1
    return -1
```

10.2

정렬된 리스트에서 가장 작거나 가장 큰 수를 찾는 경우
불균일 분할이 계속됨

10.3

7 4 9 6 3 8 7 5
4 7 6 9 3 8 5 7
4 6 7 9 3 5 7 8
3 4 5 6 7 7 8 9

10.4

(1) i<j인 모든 (A[i], A[j]) 쌍에 대해 검사 → 복잡도 $O(n^2)$

```
def count_inversion(arr):
    n = len(arr)
    inv_count = 0
    for i in range(n):
        for j in range(i + 1, n):
            if (arr[i] > arr[j]):
                inv_count += 1
    return inv_count
```

(2)


```
def merge_sort(arr, tmp, left, right):
    inv_count = 0
    if left < right:
        mid = (left + right)//2
        inv_count += merge_sort(arr, tmp, left, mid)
        inv_count += merge_sort(arr, tmp, mid + 1, right)
        inv_count += merge(arr, tmp, left, mid, right)
    return inv_count

def merge(arr, tmp, left, mid, right):
    i = left        # 왼쪽 리스트 인덱스 시작
    j = mid + 1     # 오른쪽 리스트 인덱스 시작
    k = left        # tmp 인덱스 시작
    inv_count = 0   # 역전 개수

    while i <= mid and j <= right:
        if arr[i] <= arr[j]:
            tmp[k] = arr[i]
            i += 1
        else:
            tmp[k] = arr[j]
            inv_count += (mid-i + 1)
            j += 1
        k += 1

    if i <= mid :
        tmp[k:k+(mid-i+1)] = arr[i:i+(mid-i+1)]
    if j <= right :
        tmp[k:k+(right-j+1)] = arr[j:j+(right-j+1)]
    arr[left:right+1] = tmp[left:right+1]
    return inv_count

def count_inversion_dc(arr):
    n = len(arr)
    tmp = [0]*n
    return merge_sort(arr, tmp, 0, n-1)
```

10.5

동전을 세 그룹으로 나눈다. L L L L M M M M R R R R

L그룹과 M그룹을 비교해서

동일하면

M M M 과 R R R

동일하면

M R4

R 그룹이 무거우면(가벼우면)

R1 R2 비교

같으면 R3가 무거움(가벼움)

다르면 무거운 쪽이 불량

L쪽이 무거우면(가벼우면)

L1 L2 L3 M1과 L4 R R R 비교

균형 M1 M2 M3에 가벼운 것이 있음

왼쪽 L1 L2 L3에 무거운 것이 있음

오른쪽 M1이 가볍거나 L4가 무거움

11장. 해답

11.1

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2]
[0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 4, 4]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 4, 4]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 4, 4]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 4, 4]
```

11.2

```
LCS =  TART
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2]
[0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 4, 4]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 4, 4]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 4, 4]
[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 4, 4]
```

11.3

```
[0,  0,  0,  0,  0,  0,  0]
[0,  0,  0, 26, 26, 26, 26]
[0,  0, 20, 26, 26, 46, 46]
[0, 14, 20, 34, 40, 46, 60]
[0, 14, 20, 34, 40, 54, 60]
[0, 14, 20, 34, 40, 54, 64]
```

11.4

```
def knapSack_mem(W, wt, val, n):
    if A[n][W] == None :
        if n == 0 or W == 0 :
            A[n][W] = 0
        elif (wt[n-1] > W):
            A[n][W] = knapSack_mem(W, wt, val, n-1)
        else:
            valWithout = knapSack_mem(W, wt, val, n-1)
            valWith = val[n-1] + knapSack_mem(W-wt[n-1], wt, val, n-1)
            A[n][W] = max(valWith, valWithout)
    return A[n][W]
```

11.5

```
def edit_distance_mem(S, T, m, n, mem):
    if m == 0: return n       # S가 공백이면, T의 모든 문자를 S에 삽입
    if n == 0: return m       # T가 공백이면, S의 모든 문자들을 삭제

    if mem[m-1][n-1] == None :    # 아직 해결되지 않은 문제이면
        if S[m-1]== T[n-1]:       # S와 T의 마지막 문자가 같으면,
            mem[m-1][n-1] = edit_distance_mem(S, T, m-1, n-1, mem)

        else:  # 그렇지 않으면, 세 연산을 모두 적용하
            mem[m-1][n-1] = 1 + W
               min( edit_distance_mem(S, T, m, n-1, mem),      # 삽입
                    edit_distance_mem(S, T, m-1, n, mem),      # 삭제
                    edit_distance_mem(S, T, m-1, n-1, mem))    # 대체
        # print("mem[%d][%d] = "%(m-1,n-1), mem[m-1][n-1])  # 저장 순서

    return mem[m-1][n-1]    # 해를 반환
```


12장. 해답

12.1

(1) 선형 조사법

	12	44	13	88	23	94	11	39	20	16	5
[0]		44	44	44	44	44	44	44	44	44	44
[1]	12	12	12	12	12	12	12	12	12	12	12
[2]			13	13	13	13	13	13	13	13	13
[3]				88	88	88	88	88	88	88	88
[4]					23	23	23	23	23	23	23
[5]							11	11	11	11	11
[6]						94	94	94	94	94	94
[7]								39	39	39	39
[8]										16	16
[9]									20	20	20
[10]											5
h(k)	1	0	2	0	1	6	0	6	9	5	5

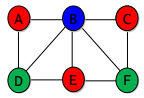
(2) 이차 조사법: j*j 사용: h'(k) = (k+j*j) % 11

	12	44	13	88	23	94	11	39	20	16	5
[0]		44	44	44	44	44	44	44	44	44	44
[1]	12	12	12	12	12	12	12	12	12	12	12
[2]			13	13	13	13	13	13	13	13	13
[3]										16	16
[4]				88	88	88	88	88	88	88	88
[5]					23	23	23	23	23	23	23
[6]						94	94	94	94	94	94
[7]								39	39	39	39
[8]											5
[9]							11	11	11	11	11
[10]									20	20	20
h(k)	1	0	2	0	1	6	0	6	9	5	5
이차 조사				+1 +4	+1 +4		+1 +4 +9	+1	+1	+1 +4 +9	+1 +4 +9 +16 +25

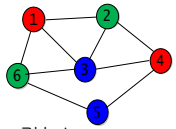
(3) 이중 해시법: step = 7 - (k mod 7) 사용

	12	44	13	88	23	94	11	39	20	16	5
[0]		44	44	44	44	44	44	44	44	44	44
[1]	12	12	12	12	12	12	12	12	12	12	12
[2]			13	13	13	13	13	13	13	13	13
[3]				88	88	88	88	88	88	88	88
[4]								39	39	39	39
[5]									20	20	20
[6]					23	23	23	23	23	23	23
[7]											5
[8]										16	16
[9]							11	11	11	11	11
[10]						94	94	94	94	94	94
h(k)	1	0	2	0	1	6	0	6	9	5	5
step				3	5	4	3	3	1	5	2
이중 해시				+3	+5	+4	+3 +6 +9	+3 +6 +9	+1 ... +7	+5 +10 ... +25	+2

12.2



가능!



가능!

12.3

```
def is_safe(grid, x, y, n):
    for i in range(9):      # 가로나 세로줄에 n이 있으면 안됨.
        if grid[y][i] == n or grid[i][x] == n:
            return False
    for i in range(3):      # 3x3 subgrid에도 n이 있으면 안됨.
        for j in range(3):
            if grid[(y - y%3) + i][(x - x%3) + j] == n:
                return False
    return True

def is_completed(grid):
    for row in grid:
        for cell in row:
            if cell == 0:
                return False
    return True

def find_empty_location(grid):
    for y in range(9):
        for x in range(9):
            if grid[y][x] == 0:
                return x, y

def solveSudoku(grid):
    if is_completed(grid):              # 종료조건 검사
        return True

    x, y = find_empty_location(grid)    # 숫자가 채워지지 않은 위치

    for digit in range(1, 10):          # dight=1~9 에 대해
        if is_safe(grid, x, y, digit):  # 겹치지 않으면
            grid[y][x] = digit          # digit를 해당 위치에 놓고
            if solveSudoku(grid):       # 나머지 그리그를 풀어서
                return True             # 성공하면 종료
                                        # 실패하면
            grid[y][x] = 0              # 그 자리를 0으로 만들고 backtracking

    return False

# 수도쿠 그리드를 출력
def print_grid(grid):
    print('-'*18)
    for row in grid:
        for cell in row:
            print(cell, end=" ")
        print()
    print()


#################################################################
import copy

# assigning initial values to the grid
```

```
grid1 = [
    [3, 0, 6, 5, 0, 8, 4, 0, 0],
    [5, 2, 0, 0, 0, 0, 0, 0, 0],
    [0, 8, 7, 0, 0, 0, 0, 3, 1],
    [0, 0, 3, 0, 1, 0, 0, 8, 0],
    [9, 0, 0, 8, 6, 3, 0, 0, 5],
    [0, 5, 0, 0, 9, 0, 6, 0, 0],
    [1, 3, 0, 0, 0, 0, 2, 5, 0],
    [0, 0, 0, 0, 0, 0, 0, 7, 4],
    [0, 0, 5, 2, 0, 6, 3, 0, 0],
]
# a grid with no solution
grid2 = [
    [5, 0, 6, 5, 0, 8, 4, 0, 3],
    [5, 2, 0, 0, 0, 0, 0, 0, 2],
    [1, 8, 7, 0, 0, 0, 0, 3, 1],
    [0, 0, 3, 0, 1, 0, 0, 8, 0],
    [9, 0, 0, 8, 6, 3, 0, 0, 5],
    [0, 5, 0, 0, 9, 0, 6, 0, 0],
    [1, 3, 0, 0, 0, 0, 2, 5, 0],
    [0, 0, 0, 0, 0, 0, 0, 7, 4],
    [0, 0, 5, 2, 0, 6, 3, 0, 0],
]
print_grid(grid1)
if solveSudoku(grid1):
    print("스도쿠 정답:")
    print_grid(grid1)
else:
    print("답이 없는 스도쿠 문제입니다.")

print_grid(grid2)
if solveSudoku(grid2):
    print("스도쿠 정답:")
    print_grid(grid2)
else:
    print("답이 없는 스도쿠 문제입니다.")
```