알고리즘과 자료구조 워크북

교과목명 : 알고리즘과 자료구조
차시명: 3차시 Brute-force 전략2

◈ 담당교수: 신일훈 **(서울과학기술대학교)**

◉ 세부목차
- brute-force 전략의 적용1: 특정 문자의 개수를 구하는 알고리즘
- brute-force 전략의 적용2: 선택정렬을 수행하는 알고리즘
- brute-force 전략의 적용3: 최근접 거리를 구하는 알고리즘
- brute-force 전략의 적용3: 배낭 문제의 해답을 구하는 알고리즘

학습에 앞서

■ 학습개요

Brute-force 전략은 문제의 해답을 찾기 위해 가능한 솔루션을 전부 시도하는 단순한 방법이다. 본 강에서는 문자열에서 특정 문자의 개수를 세는 문제, 파이썬 리스트를 선택 정렬을 사용하여 오름차순으로 정렬하는 문제, N개의 2차원 평면 상에서의 점들의 좌표가 주어졌을 때 최근접 거리를 찾는 문제, N개의 아이템을 최대 무게가 제한되어 있는 배낭에 넣는 최적 조합을 찾는 문제를 brute-force 전략을 활용하여 해결하는 것을 배운다.

■ 학습목표

1	brute-force 전략의 개념을 이해한다.
2	문자의 개수를 구하는 알고리즘을 설계하고 표현할 수 있다.
3	선택 정렬을 수행하는 알고리즘을 설계하고 표현할 수 있다.
4	최근접 거리를 구하는 알고리즘을 설계하고 표현할 수 있다.
5	배낭 문제를 해결하는 알고리즘을 설계하고 표현할 수 있다.

■ 주요용어

용어	해설
brute-force	어떤 문제의 해답을 구하기 위해 가능한 모든 경우를 모두 시도하는 방법
문자열	여러 개의 문자로 구성된다. 파이썬에서는 “” 또는 ‘’로 표시한다.
선택정렬	대상 숫자들 중에서 최대값 또는 최소값을 반복해서 찾아가며 정렬을 수행하는 알고리즘
거리	$\sqrt{(x1-x2)^2+(y1-y2)^2}$
Knapsack(배낭) 문제	배낭에 넣을 수 있는 최대 무게가 제한되어 있고, N개의 물건이 있을 때, 배

	낭에 넣을 수 있는 물건들의 가치의 총합을 최대로 하는 물건 조합을 찾는 최적화 문제

학습하기

1. brute-force 전략의 적용: 특정 문자의 개수 구하기

가. 문제

- 문자열에서 알파벳 a 의 개수를 출력하시오.

나. 아이디어

- 결과를 저장할 변수 count를 0으로 초기화한다.
- 문자열의 각 문자가 a인지를 검사하고 a이면 count를 1 증가한다.
- 이러한 작업을 문자열의 모든 문제에 대해 반복한다.

다. 알고리즘(의사코드로 표현)

```
count = 0
for character in string :
        if (character == 'a') :
                count += 1
print(count)
```

- sring은 대상 문자열을 저장하는 변수를 의미한다.
- for 문은 string의 길이 만큼 반복되며, 각 반복 시마다 string의 각각의 문자가 변수 character에 저장된다.

라. 알고리즘 최악 시간복잡도

- for 문은 string 문자열의 길이만큼 반복되므로 for문의 반복 회수가 입력 데이터의 크기인 string에 비례한다.
- 따라서 알고리즘의 최악 시간복잡도는 O(N)이다.
- 이 경우에는 최선, 평균 시간복잡도도 모두 O(N)이다.

마. 파이썬 구현 및 실행1

```
def count_characters(string) :
        count = 0
        for character in string :
                if (character == 'a') :
                        count += 1
        return count

print(count_characters("There is an apple in the table.")
```

- 위의 파이썬 코드는 의사 코드와 거의 동일하다.
- 다음은 이 파이썬 코드를 스파이더에서 실행한 결과 화면이다.

```
In [20]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled1.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
3
```

2. brute-force 전략의 적용: 선택정렬

가. 문제

- N개의 숫자를 저장한 파이썬 리스트를 내림차순으로 정렬하시오.
- 가령 대상 리스트가 [3, 2, 7, 1, 5] 라면, [7, 5, 3, 2, 1]을 반환한다.

나. 아이디어

- 정렬을 수행하는 알고리즘은 다양한다. 버블정렬, 선택정렬, 병합정렬, 고속정렬 등이 있다.
- 선택정렬은 brute-force 전략을 적용한 알고리즘이다.
- 먼저 정렬된 결과를 저장할 리스트 sorted_list를 비어 있는 상태로 초기화한다.
- 내림차순으로 정렬해야 하므로 대상 리스트에서 가장 큰 값을 찾는다. 그리고 이 값을 대상 리스트에서 제거하고 sorted_list 리스트에 추가(append)한다.
- 따라서 대상 리스트의 크기는 1 감소하며, sorted_list 리스트의 크기는 1 증가한다.
- 이 작업을 대상 리스트가 빌 때까지 반복한다.

다. 선택정렬의 진행 과정

```
1. list = [3, 2, 7, 1, 5],          sorted_list = [ ]
2. list = [3, 2, 1, 5],             sorted_list = [7]
3. list = [3, 2, 1],                sorted_list = [7, 5]
4. list = [2, 1],                   sorted_list = [7, 5, 3]
5. list = [1],                      sorted_list = [7, 5, 3, 2]
6. list = [ ],                      sorted_list = [7, 5, 3, 2, 1]
```

- 1번 단계에서는 최대값 7을 찾고 이를 list에서 제거하고 sorted_list에 추가한다.
- 이 작업을 list가 빌 때까지 반복하면, sorted_list에 정렬된 결과가 저장된다.

라. 알고리즘(의사코드로 표현)

```
result = [ ]
while (list is not empty)  :
        find max_number in list
        remove max_number from list
        append max_number to result
return result
```

- 결과를 저장할 리스트 result를 비어 있는 상태로 초기화한다.
- while 문은 대상 리스트 list에 값이 남아 있는 동안 반복된다.
- while 문 안에서는 대상 리스트 list에서 최대값을 찾고 이를 대상 리스트에서 제거한다. 그리고 이 값을 결과를 저장할 result에 추가한다.
- 결과적으로 while 문은 list의 원소의 개수만큼 반복된다.
- 위 의사코드에서 find max_number in list의 알고리즘도 기술해야 하지만, 이는 2차시의 최대값 찾는 문제와 동일하므로 생략한다.

마. 파이썬 구현 및 실행

```
def selection_sort(list) :
        sorted_list = []
        while (len(list) > 0) :
                max_index = get_max_index(list)
                max = list.pop(max_index)
                sorted_list.append(max)
        return result
```

- len(list)는 list의 원소의 개수를 반환한다.
- 따라서 while 문은 list에 원소가 남아 있는 한 반복되며, while 문 안에서 list의 크기가 1 감소하므로 원 list의 원소의 개수만큼 반복된다.
- get_max_index()는 list에서 최대값의 인덱스(순서)를 반환한다.
- pop(max_index) 함수는 list에서 인덱스가 max_index 인 값을 제거한다.
- append(max) 함수는 max 값을 sorted_list의 뒤에 추가한다.
- 따라서 while 문을 모두 반복하면 list는 빈 리스트가 되고, sorted_list에는 정렬된 결과가 저장된다.

- 다음은 get_max_index(list)의 코드이다.

```
def get_max_index(list) :
        max_index = 0
        for index in range(len(list)) :
                if (list[max_index] < list[index]) :
                        max_index = index
        return max_index
```

- max_index는 최대값의 인덱스를 저장한다.
- 2차시의 최대값을 찾는 문제에서 학습한 것처럼 리스트의 첫 번째 값을 최대값으로 가정한다. 즉 max_index를 0으로 초기화한다.
- for 문은 list의 원소의 개수만큼 반복되며 개수가 N이라고 하면 index에는 0 ~ N-1의 값이 차례로 저장된다. 따라서 기존의 최대값과 list의 각 값을 비교하여 list의 값이 크면 이 인덱스를 max_index에 저장한다.
- 따라서 for 문의 반복이 끝나면, 최대값의 인덱스가 max_index에 저장된다.

- 다음은 이 파이썬 코드를 테스트하기 위한 코드이다.

```
list = [3, 5, 7, 2, 9, 11, 2, 3, 8]
sorted_list = selection_sort(list)
print(sorted_list)
```

- 다음은 이 파이썬 코드를 스파이더에서 실행한 결과 화면이다.

```
In [21]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled1.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
[11, 9, 8, 7, 5, 3, 3, 2, 2]
```

바. 알고리즘 최악 시간복잡도

- selection_sort(list) 함수는 list의 원소의 개수만큼 반복된다. 즉, 반복 회수가 입력 데이터의 크기인 N에 비례한다.
- get_max_index(list) 함수의 반복 회수는 list의 원소의 개수와 동일하다. 단, list의 크기가 매번 달라짐

에 유의해야 한다. 즉 selection_sort() 안의 첫 번째 반복 시에는 list의 길이가 N이지만, 매 반복 시마다 1씩 감소하고 최종 반복 시에는 list의 길이가 1이다. 따라서 평균적으로 보면 N/2만큼 반복된다고 할 수 있다.

- 따라서 전체적인 반복 회수는 N*(N/2)에 비례한다.
- 결과적으로 알고리즘의 최악 시간복잡도는 $O(N^2)$이다.
- 이 경우에는 최선, 평균 시간복잡도도 모두 $O(N^2)$이다. (N개의 값들을 모두 비교해야만 최대값을 찾을 수 있기 때문)

3. brute-force 전략의 적용: 최근접 거리

가. 문제

- 2차원 평면 상에 n개의 점이 있다. 가장 인접한 쌍의 거리를 구하라.
- 하나의 점의 좌표는 (x, y)로 주어지며, n 개의 점의 좌표값 리스트가 입력으로 주어진다.

나. 아이디어

- 입력값: points = [(3, 2), (0, 5), (-3, 2)]
- 모든 두 점 간의 거리를 구하고, 이들 중 최소값을 찾는다. ($_nC_2$)

다. 알고리즘(의사코드로 표현)

```
n = len(points)
min = INF

for i in 0 ~ n-2 :
        p1 = points[i]
        for j in i+1 ~ n-1 :
                p2 = points[j]
                dist = cal_dist(p1, p2)
                if (min > dist)
                        min = dist
return min
```

- points는 대상 점들의 좌표쌍들을 저장한 리스트이다.
- 따라서 len(points)는 점들의 개수를 반환한다.
- min은 결과를 저장할 변수이며, 가장 가까운 거리도 0이상일 것으로 0으로 초기화해도 되며 나올 수 있는 가장 큰 값으로 초기화해도 된다. 여기서 INF는 나올 수 있는 가장 큰 값을 의미한다.
- 모든 두 점 간의 거리를 구해야 하므로 ($_nC_2$ 경우의 수), for 문을 중첩하여 사용하되 바깥 for문은 시작점을 의미하고 안쪽 for문은 끝점을 의미한다.
- 만약 점이 $P_0 \sim P_{N-1}$ 이 있다면, 시작점은 $P_0 \sim P_{N-2}$ 가 되므로 바깥쪽 for 문은 인덱스 0 ~ N-2에 대해 반복해야 한다.
- 시작점의 인덱스가 i 라고 한다면, 끝점은 $P_{i+1} \sim P_{N-1}$ 이 되므로 안쪽 for 문은 인덱스 i+1 ~ N-1에 대해 반복해야 한다.
- cal_dist() 함수는 두 점 사이의 거리를 구하는 기능을 수행한다.

라. 알고리즘 최악 시간복잡도

- 바깥 for 문이 N-1번 반복된다.
- 안쪽 for 문은 바깥 for 문의 반복 때마다 반복 회수가 1씩 감소한다. 즉 처음에는 i가 0이므로 N-1번 반복되며 이후에는 N-2번 반복되고 마지막에는 1번 반복된다. 따라서 평균적으로는 대략 N/2번 반복된다고 볼 수 있다.

- 따라서 전체 알고리즘의 반복 회수는 (N-1)*(N/2)에 비례하며 결과적으로 알고리즘의 최악 시간복잡도는 $O(N^2)$이다.
- 이 경우에는 최선, 평균 시간복잡도도 모두 $O(N^2)$이다.

마. 파이썬 구현 및 실행

```
def closest_dist(points) :
        count = len(points)
        min = float("inf")

        for i in range(count - 1) :
                for j in range(i+1, count) :
                        dist = cal_dist(points[i], points[j])
                        if (dist < min) :
                                min = dist
        return min


def cal_dist(p1, p2) :
        x_dist = (p1[0] - p2[0]) ** 2
        y_dist = (p1[1] - p2[1]) ** 2
        dist = (x_dist + y_dist) ** 0.5
        return dist
```

- float("inf")는 가능한 가장 큰 값을 반환한다.
- cal_dist() 함수는 두 점 사이의 거리를 반환한다.
- **연산자는 지수 연산자이다.

- 다음은 이 파이썬 코드를 테스트하기 위한 코드이다.

```
points = [(2,3), (3,5), (8,10), (11,-1)]
print(closest_dist(points))
```

- 다음은 이 파이썬 코드를 스파이더에서 실행한 결과 화면이다.

```
In [22]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled1.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
2.23606797749979
```

4. brute-force 전략의 적용: 배낭(knapsack) 문제

가. 문제

- 배낭에 넣을 수 있는 최대 무게가 W로 제한되어 있다.
- N개의 물건이 있으며, 이들은 각각 가치와 무게가 다를 수 있다.
- 배낭의 최대 무게를 초과하지 않으면서 가치의 총합을 최대로 하는 물건의 조합을 구하시오.

나. 아이디어

- 입력값: names = ['A', 'B', 'C', 'D', 'E'], values = [10, 30, 20, 14, 23], weights = [5, 8, 3, 7, 9], max_weight = 20
    - names 리스트는 물건들의 이름을 저장한다.
    - values 리스트는 물건들의 가치를 저장한다.
    - weights 리스트는 물건들의 무게를 저장한다.
    - max_weight는 배낭이 허용할 수 있는 최대 무게를 의미한다.

- 모든 경우의 수에 대해 가치의 총합과 무게의 총합을 각각 구한다.
    - A
    - A, B
    - A, B, C
    - A, B, C, D
    - A, B, C, D, E
    - ...
- 무게의 총합이 최대 무게 이하인 경우들 중에서 가장 가치가 높은 경우를 선택한다.

- 모든 경우를 어떻게 표현할까?
- 가령 물건이 3개인 경우, 총 경우의 수는 2^3 으로 XXX, XXO, XOX, XOO, OXX, OXO, OOX, OOO의 경우가 있다. (X는 물건을 선택하지 않은 경우이고, O는 물건을 선택한 경우를 의미한다.)
- 물건이 N개인 경우, 총 경우의 수는 2^N 이다.
- 관건은 이를 어떻게 반복문으로 설계할 것인가 이다.

- 물건을 선택하지 않은 경우를 0으로 표현하고, 물건을 선택한 경우를 1로 표현하면
- XXX, XXO, XOX, XOO, OXX, OXO, OOX, OOO는 000, 001, 010, 011, 100, 101, 110, 111이 된다.
- 이를 이진수로 해석하면 => 0 ~ (2^3-1) 이다.
- 즉 for 반복문 등을 0 ~ (2^3-1) 까지 반복하면 된다.
- 이진수의 각 비트가 0이면 비트 순서에 해당하는 물건을 선택하지 않았고 1은 해당 물건을 선택했음을 의미한다.
- 가령 3은 이진수로 011 이므로 첫째 물건을 선택, 둘째 물건을 선택, 셋째 물건을 제외한 경우를 의미한다.

다. 알고리즘(의사코드로 표현)

```
max_case = 0
max_value = 0
total_count = power(2, n)

for case  in (0 ~ total_count-1)  :
        value = 0, weight = 0
        for bitnum in (0 ~ n-1)  :
                if (bitnum bit of case is 1) :
                        value += values[bitnum]
                        weight += weight[bitnum]
        if (value > max_value and weight <= max_weight) :
                max_case = case
                max_value = value
return (max_case, max_value)
```

- power(2, n)은 2^n을 반환한다. 즉 물건이 n개일 때 조합이 가능한 경우의 수를 의미한다.
- max_value는 가치의 합의 최대값을 저장하며, max_case는 그 때의 물건의 조합을 저장한다.
- 바깥 for 문은 모든 경우의 수에 대해 반복된다. value는 선택된 물건의 가치의 합을 저장하고 weight는 선택된 물건들의 무게의 합을 저장한다.
- 안쪽 for 문은 물건의 개수 만큼 반복된다. case의 각 비트가 0인지, 1인지를 확인하여 1이면 해당 물건을 선택한다는 의미이므로 물건의 가치와 무게를 각각 value, weight에 더한다.
- 각 케이스에 대해 가치의 총합(value)이 기존의 max_value보다 크며 무게의 총합(weight)이 배낭의 총 무게를 초과하지 않으면 max_value와 max_case를 각각 업데이트 한다.

라. 알고리즘 최악 시간복잡도

- 바깥 for 문은 2^N번 반복된다.
- 안쪽 for 문은 N번 반복된다.
- 따라서 전체 알고리즘의 반복 회수는 2^N*N에 비례하며 결과적으로 알고리즘의 최악 시간복잡도는 $O(N*2^N)$이다.

마. 파이썬 구현 및 실행

1) 1단계 - Item 클래스 정의

```
class Item(object):
    def __init__(self, name, value, weight):
        self.name = name
        self.value = value
        self.weight = weight

    def __str__(self):
        return (self.name)
```

- item 클래스는 하나의 물건을 나타낸다. 멤버 변수로 name, value, weight를 가진다.
- __str__()는 item 객체를 출력할 경우 name이 출력된다는 것을 의미한다.

2) 2단계 - Knapsack 클래스 정의

```
class Knapsack(object):
    def __init__(self, names, values, weights, max_weight):
        self.items = []
        self.chosen_items = []

        self.max_weight = max_weight
        self.max_value = 0

        for i in range(len(names)) :
            item = Item(names[i], values[i], weights[i])
            self.items.append(item)

    def findOptCase(self):
        self.max_value = 0
        item_count = len(self.items)
        case_count = 2**item_count

        for case in range(1, case_count):
            value = 0
            weight = 0
            chosen_items = []

            for bitnum in range(item_count):
                target_bit_value = (1 << bitnum)
                if (case & target_bit_value != 0):
                    # bitnum에 해당하는 비트 자리가 1
                    item = self.items[bitnum]
                    value += item.value
                    weight += item.weight
                    chosen_items.append(item)

            if (value > self.max_value and weight <= self.max_weight):
                self.chosen_items = list(chosen_items)
                self.max_value = value

        return (self.chosen_items, self.max_value)

    def print(self, policy, chosen_items, value):
        print(f"{policy}: {value}")
        for item in chosen_items:
            print(item)
        print()
```

- 생성자(__init__())에서는 N개의 Item 객체를 생성하여, itesms 리스트에 저장한다.
- 멤버 변수 chosen_items는 가치를 최대로 하는 물건들의 조합을 저장하며, max_value는 그 때의 가치의 총합을 저장한다.
- find_opt_case() 함수는 의사코드에서 기술된 것과 동일하게 가치의 총합을 최대로 하는 물건의 조합

을 찾아서 self.chosen_items에 저장한다.
- print() 함수는 가치의 총합을 최대로 하는 물건들의 조합을 출력한다.

3) 3단계 - Knapsack 테스트 코드

```
names = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
values = [10, 30, 20, 14, 23, 11, 15, 18, 12]
weights = [5, 8, 3, 7, 9, 2, 6, 2, 3]
max_weight = 20
knapsack = Knapsack(names, values, weights, max_weight)
(chosen_items, max_value) = knapsack.findOptCase()
knapsack.print('brute force', chosen_items, max_value)
```

- 다음은 이 파이썬 코드를 스파이더에서 실행한 결과 화면이다.
- 배낭의 최대 무게가 20일 때, 가치를 최대로 하는 조합은 B, C, F, H, I를 선택한 경우이며 그때의 가치의 총합은 91이다.

```
brute force: 91
B
C
F
H
I
```

연습문제

1. 주어진 파이썬 리스트에서 숫자 3의 개수를 출력하는 알고리즘을 의사코드로 표현하시오.

정답 :

```
count = 0
for num in list :
        if (num == 3) :
                count += 1
print(count)
```

해설 : 결과를 저장할 변수 count를 0으로 초기화하고 for 문을 활용하여 리스트의 각 원소가 3인지를 검사한다. 3이면 count를 증가시킨다.
최종적으로 count를 출력한다.

2. 선택 정렬에 대해서 설명하시오.

정답 : 최소값 또는 최대값을 반복적으로 찾아가며 정렬을 수행하는 정렬 방식이다.
해설 : 선택 정렬은 내림 차순 정렬의 경우, 원본 list에서 최대값을 찾고, 이를 list에서 제거하며, 정렬된 결과를 저장할 리스트에 최대값을 추가한다. 이를 원본 list가 빌 때까지 반복하는 정렬 알고리즘이다.

3. 주어진 파이썬 리스트를 선택정렬을 사용하여 오름차순으로 정렬하는 알고리즘을 의사코드로 표현하시오.

정답 :

```
def sort(list) :
        result = []
        while (list is not empty) :
                min = get_min(list)
                list.remove(min)
                result.append(min)
        return result

def get_min(list) :
        min = list[0]
        for num in list[1] ~ list[N-1] :
                if (min > num) :
                        min = num
        return min
```

해설 : list에서 최소값을 찾고, 이를 list에서 제거하며, 정렬된 결과를 저장할 result에 추가한다. 이를 list가 빌 때까지 반복한다.

정리하기

1. brute-force 전략은 문제의 해답을 찾기 위해 가능한 솔루션을 전부 시도하는 단순한 방법이다.
2. 문자의 개수를 세는 알고리즘을 brute-force 전략을 활용하여 설계할 수 있다.
3. 선택 정렬을 수행하는 알고리즘을 brute-force 전략을 활용하여 설계할 수 있다.
4. 최근접 거리를 구하는 알고리즘을 brute-force 전략을 활용하여 설계할 수 있다.
5. 배낭 문제를 해결하는 알고리즘을 brute-force 전략을 활용하여 설계할 수 있다.

참고자료

- 파이썬 알고리즘, 최영규, 2021, 생능출판

다음 차시 예고

- 축소정복과 분할정복에 대해 학습한다.