03강

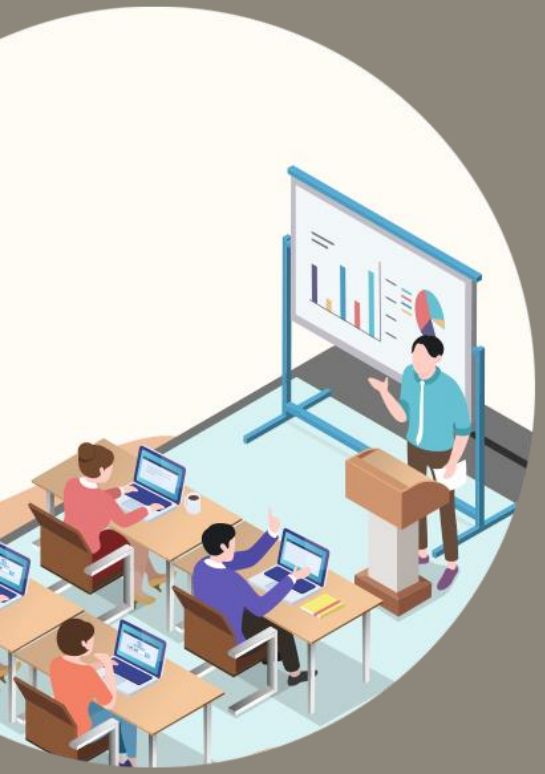
알고리즘과 자료구조

Brute-force 전략2

서울과학기술대학교 신일훈 교수

한국방송통신대학교 프리임칼리지

학습목표



1. 문자의 개수를 구하는 알고리즘을 설계할 수 있다.
2. 선택 정렬을 수행하는 알고리즘을 설계할 수 있다.
3. 최근접 거리를 구하는 알고리즘을 설계할 수 있다.
4. 배낭 문제의 해답을 구하는 알고리즘을 설계할 수 있다.

01

문자의 개수

1. 문자의 개수

- 문자열에서 알파벳 a 의 개수를 출력하시오.

의사코드

```
string = “There is an apple in the table.”
```

1. 문자의 개수

- 문자열 string 에서 알파벳 a 의 개수를 출력하시오.

의사코드

string = “There is an apple in the table.”

count = 0

1. 문자의 개수

- 문자열 string 에서 알파벳 a 의 개수를 출력하시오.

의사코드

```
string = “There is an apple in the table.”

count = 0
for character in string :
    if (character == ‘a’) :
        count += 1
return count
```

1. 문자의 개수

- 문자열 string 에서 알파벳 a 의 개수를 출력하시오.

의사코드

```
string = "There is an apple in the table."
count = 0
for character in string :
    if (character == 'a') :
        count += 1
return count
```

최악 시간복잡도 ?

1. 문자의 개수

- 문자열 string 에서 알파벳 a 의 개수를 출력하시오.

의사코드

```
string = "There is an apple in the table."
count = 0
for character in string :
    if (character == 'a') :
        count += 1
return count
```

최악 시간복잡도 : O(N)

1. 문자의 개수

- 문자열 string 에서 알파벳 a 의 개수를 출력하시오.

파이썬 코드

```
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."))
```

1. 문자의 개수

- 문자열 string 에서 알파벳 a 의 개수를 출력하시오.

파이썬 코드 실행

```
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."))
```

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

02

선택 정렬

2. 선택 정렬

- N개의 숫자를 저장한 파이썬 리스트를 내림차순으로 정렬하시오.

아이디어

list = [3, 2, 7, 1, 5]

2. 선택 정렬

- N개의 숫자를 저장한 파이썬 리스트를 내림차순으로 정렬하시오.

아이디어

list = [3, 2, 7, 1, 5] , sorted_list = [ ]

- list에서 가장 큰 값을 찾는다.
- 찾은 큰 값을 list에서 제거하고, 이를 새로운 sorted_list에 추가한다.
- 이를 list가 빌 때까지 반복한다.

2. 선택 정렬

- N개의 숫자를 저장한 파이썬 리스트를 내림차순으로 정렬하시오.

아이디어

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]

2. 선택 정렬

- N개의 숫자를 저장한 파이썬 리스트를 내림차순으로 정렬하시오.

의사코드

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

2. 선택 정렬

- N개의 숫자를 저장한 파이썬 리스트를 내림차순으로 정렬하시오.

파이썬 코드

```
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  sorted_list
```

2. 선택 정렬

- N개의 숫자를 저장한 파이썬 리스트를 내림차순으로 정렬하시오.

파이썬 코드

```
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
```

2. 선택 정렬

- N개의 숫자를 저장한 파이썬 리스트를 내림차순으로 정렬하시오.

파이썬 코드

```
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  sorted_list
```

최악 시간복잡도 ?

2. 선택 정렬

- N개의 숫자를 저장한 파이썬 리스트를 내림차순으로 정렬하시오.

파이썬 코드

```
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
```

최악 시간복잡도 ?

2. 선택 정렬

- N개의 숫자를 저장한 파이썬 리스트를 내림차순으로 정렬하시오.

파이썬 코드

```
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  sorted_list
```

최악 시간복잡도 : $O(N^2)$

2. 선택 정렬

- N개의 숫자를 저장한 파이썬 리스트를 내림차순으로 정렬하시오.

파이썬 코드 실행

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

```
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]
```

03

최근접 거리

3. 최근접 거리

- 2차원 평면 상에 n개의 점이 있다. 가장 인접한 쌍의 거리를 구하라.

아이디어

points = [(3, 2), (0, 5), (-3, 2)]

3. 최근접 거리

- 2차원 평면 상에 n개의 점이 있다. 가장 인접한 쌍의 거리를 구하라.

아이디어

points = [(3, 2), (0, 5), (-3, 2)]

- 모든 두 점 간의 거리를 구하고, 이들 중 최소값을 찾는다. ($_nC_2$)
  - P0 - P1
  - P0 - P2
  - P1 - P2

3. 최근접 거리

- 2차원 평면 상에 n개의 점이 있다. 가장 인접한 쌍의 거리를 구하라.

의사코드

```
points = [(3, 2), (0, 5), (-3, 2)]
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
```

3. 최근접 거리

- 2차원 평면 상에 n개의 점이 있다. 가장 인접한 쌍의 거리를 구하라.

의사코드

```
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
```

최악 시간복잡도 ?

3. 최근접 거리

- 2차원 평면 상에 n개의 점이 있다. 가장 인접한 쌍의 거리를 구하라.

의사코드

```
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
```

최악 시간복잡도 : $O(N^2)$

3. 최근접 거리

- 2차원 평면 상에 n개의 점이 있다. 가장 인접한 쌍의 거리를 구하라.

파이썬 코드

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

3. 최근접 거리

- 2차원 평면 상에 n개의 점이 있다. 가장 인접한 쌍의 거리를 구하라.

파이썬 코드

```
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
```

3. 최근접 거리

- 2차원 평면 상에 n개의 점이 있다. 가장 인접한 쌍의 거리를 구하라.

파이썬 코드

```
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
```

3. 최근접 거리

- 2차원 평면 상에 n개의 점이 있다. 가장 인접한 쌍의 거리를 구하라.

파이썬 코드 실행

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

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

04

Knapsack

4. Knapsack

- 배낭에 넣을 수 있는 최대 무게: W, N개의 물건 (value , weight)
  배낭에 넣을 수 있으면서 가치의 총합을 최대로 하는 물건의 조합을 구하시오.

아이디어

names = [‘A', ‘B', ‘C', ‘D', ‘E’], values = [10, 30, 20, 14, 23]

weights = [5, 8, 3, 7, 9], max_weight = 20

4. Knapsack

아이디어

names = [‘A', ‘B', ‘C', ‘D', ‘E’], values = [10, 30, 20, 14, 23]
weights = [5, 8, 3, 7, 9], max_weight = 20

- 모든 경우의 수에 대해 가치의 총합과 무게의 총합을 각각 구하자.
  1. A
  2. A, B
  3. A, B, C
  4. A, B, C, D
  5. A, B, C, D, E
  6. …
- 무게의 총합이 최대 무게 이하인 경우들 중에서 가장 가치가 높은 경우를 선택한다.

4. Knapsack

아이디어

- 모든 경우를 어떻게 표현할까?
- 물건이 3개인 경우, 총 경우의 수
  - 2^3
  - XXX, XXO, XOX, XOO, OXX, OXO, OOX, OOO
- 물건이 N개인 경우, 총 경우의 수
  - 2^N
- 어떻게 반복문으로 설계할까?

4. Knapsack

아이디어

- 어떻게 반복문으로 설계할까?
  - 물건을 선택하지 않은 경우를 0으로 표현하고, 물건을 선택한 경우를 1로 표현하면
    - XXX, XX0, X0X, X00, 0XX, 0X0, 00X, 000
      => 000, 001, 010, 011, 100, 101, 110, 111
    - 이진수로 해석하면 => 0 ~ (2^3-1)
    - for 반복문 등을 0 ~ (2^3-1) 까지 반복하면 됨.
    - 각 비트가 0이면 해당 물건을 선택하지 않았고 1은 해당 물건을 선택했다고 해석하여 무게와 가치를 계산
    - 가령 3은 011 이므로 첫째 물건 선택, 둘째 물건 선택, 셋째 물건 제외한 경우를 의미.

4. Knapsack

의사코드

```
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)
```

4. Knapsack

의사코드

```
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)
```

최악 시간복잡도?

4. Knapsack

의사코드

```
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)
```

최악 시간복잡도 : $O(N*2^N)$

정리하기

- 문자의 개수를 세는 알고리즘
- 선택 정렬을 수행하는 알고리즘
- 최근접 거리를 찾는 알고리즘
- 배낭 문제의 해를 찾는 알고리즘

04강

다음 시간 안내 ▶▶▶

축소 정복, 분할 정복