알고리즘과 자료구조 워크북

교과목명 : 알고리즘과 자료구조
차시명: 6차시 탐욕 알고리즘

◈ 담당교수: 신일훈 **(서울과학기술대학교)**

◉ 세부목차
- 탐욕 알고리즘의 개념
- 탐욕 알고리즘의 적용1: 거스름돈을 구하는 알고리즘
- 탐욕 알고리즘의 적용2: 배낭 문제를 해결하는 알고리즘

학습에 앞서

■ 학습개요

탐욕(greedy) 알고리즘은 최적의 해를 구하기 위한 방법 중 하나이다. Brute-force 기법이 모든 가능한 해를 전부 탐색하여 이 중 최적을 선택하는데 반해, 탐욕 기법은 현재 주어진 조건 또는 현재의 상황에서 가장 최적으로 생각되는 해를 찾는다. 따라서 시간복잡도는 낮으나 정확한 최적해를 구하지는 못할 수 있다. 본 강의에서는 이러한 탐욕 알고리즘의 개념을 이해하고 이를 활용하여 거스름돈 구하는 문제와 knapsack 문제를 해결하는 알고리즘을 설계한다.

■ 학습목표

1	탐욕 알고리즘의 개념을 이해한다.
2	brute-force 대비 탐욕 알고리즘의 장단점을 이해한다.
3	탐욕 알고리즘을 적용하여 거스름돈을 구하는 알고리즘을 설계할 수 있다.
4	탐욕 알고리즘을 적용하여 knapsack 문제를 해결하는 알고리즘을 설계할 수 있다.

■ 주요용어

용어	해설
최적해	특정 문제를 해결하는 해 중 가장 최적인 해
탐욕 알고리즘	최적해를 찾기 위한 방법 중 하나로서, 현재의 조건에서 가장 최적으로 생각되는 결정을 함으로써 해를 구하는 방법
클래스	파이썬과 같은 객체 지향 언어에서 지원하는 기능으로 프로그래머가 정의한 새로운 자료형을 의미한다. 멤버변수와 메써드로 구성된다.
객체	파이썬과 같은 객체 지향 언어에서 클래스 타입의 변수를 의미한다.

학습하기

1. 탐욕(greedy) 알고리즘 개념
- 최적해를 찾기 위한 방법(brute-force, greedy, …) 중 하나임
- brute-force는 모든 경우를 탐색하고 이 중 최적의 해를 찾음.
  - 모든 경우의 수가 너무 많은 경우, 모든 경우를 탐색하는 시간이 오래 걸릴 수 있음
- 탐욕 알고리즘은 모든 경우를 고려하는 것이 아니라, 그 순간에 최적이라고 생각되는 것을 선택함.
- 가령, 장기를 둘 때 몇 수 앞까지 보는 것이 아니라, 현 상태에서 최적이라고 판단되는 결정을 함.
- 만약, 여러 도시를 최소 거리로 방문하기 위한 방문 순서를 결정하는 문제에서는 현재 위치에서 가장 가까운 도시를 다음 방문지로 결정하는 방식임.
- 결과적으로 global optimum이 아닌 local optimum (최적해의 근사값)을 도출할 가능성이 큼.
- 장점은 brute-force보다 빠른 시간 안에 솔루션을 도출할 수 있다는 것.

2. 탐욕 알고리즘의 적용1: 거스름돈 구하기

가. 문제
- 거스름돈으로 V원을 돌려줘야 할 때 동전 개수를 최소로 하는 솔루션을 구하시오.
- 단, 동전의 종류는 500, 100, 10, 1원이며, 모든 동전은 무한히 사용할 수 있다고 가정한다.
- 가령 580원을 거슬러야 한다면, 500원 1개, 10원 8개를 사용하는 것이 총 동전의 수가 9개로 가장 적다.

나. 아이디어
- brute-force 전략을 사용하다면?
  - 580원을 만드는 모든 경우에 대해 동전이 몇 개가 필요한지를 비교하여 그 중 동전의 개수를 가장 적게 하는 조합을 선택.
  - 코딩도 쉽지 않고 프로그램의 실행 시간도 오래 걸린다 (모든 경우를 비교해야 하므로).
- 탐욕 알고리즘을 사용하다면?
  - 아직 사용하지 않은 동전들 중, 액면가가 가장 높은 동전을 선택하여 해당 동전을 최대로 사용한다. 다시 말해, 액면가*N이 잔액보다 작은 N을 구한다. (N이 해당 동전의 개수가 됨).
  - 잔액을 업데이트한다.
  - 이 작업을 잔액이 0이 될 때까지 반복한다.
  - 탐욕 알고리즘을 사용하면 최적의 해를 구하지 못할 수 있지만, 대체적으로 최적에 가까운 답을 구할 수 있으며 모든 경우를 고려하지 않으므로 알고리즘의 시간 복잡도가 상대적으로 낮다.

다. 알고리즘(파이썬으로 표현)

```
입력에 대한 가정: 동전 종류는 액면가 기준으로 내림차순으로 정렬되어 전달됨
예> coins = [500, 100, 10, 1]

def min_coins_greedy(coins, change) :
        chosen = []   # 액면가 별로 선택된 동전 개수를 저장하는 리스트
        for coin in coins :
                count = change // coin  # 몫을 구함
                chosen.append(count)
                change -= count * coin
        return chosen
```

- coins는 동전의 리스트로 액면가 기준으로 내림차순으로 정렬되어 전달된다.
- change는 거스름돈 총액을 의미한다.

- min_coins_greedy() 함수의 chosen 리스트는 거스름돈 change를 만들기 위해 동전의 종류 별로 필요한 동전 개수를 저장하며 반환하다.
- //는 나눗셈의 몫을 구하는 연산자이다.
- 액면가가 높은 동전부터 남은 거스름돈을 만들기 위해 필요한 동전의 개수를 // 연산자를 활용하여 구하고 그 결과를 chosen 리스트에 추가한다. 이 때 남은 거스름돈은 그만큼 감소한다.
- 모든 동전 종류에 대해 이를 반복한다.

라. 알고리즘 최악 시간복잡도

- for 문이 동전 종류만큼 반복된다. 따라서 동전의 종류가 N개라면, 최악 시간복잡도는 O(N)이다.

마. 파이썬 구현 및 실행1

```
coins = [500, 100, 10, 1]
changes = 1534

print("잔돈: ", changes)
print("동전 종류", coins)
print("동전 개수", min_coins_greedy(coins, changes))
```

- 파이썬으로 구현된 테스트 코드에서는 거스름돈이 1534원일 때 동전의 종류 별로 필요한 개수를 출력한다.
- 다음은 이 파이썬 코드를 스파이더에서 실행한 결과 화면이다.
- 출력 결과를 보면 500원 3개, 10원 3개, 1원 4개가 필요함을 알 수 있다.

```
In [98]: runfile('D:/data/재정/주식소스/stock/
result/untitled3.py', wdir='D:/data/재정/주식소스/
stock/result')
잔돈:  1534
동전 종류 [500, 100, 10, 1]
동전 개수 [3, 0, 3, 4]
```

바. 파이썬 구현 및 실행2

```
coins = [60, 50, 5]
changes = 250

print("잔돈: ", changes)
print("동전 종류", coins)
print("동전 개수", min_coins_greedy(coins, changes))
```

- 동일한 알고리즘에 대해 동전의 종류를 60, 50, 5원으로 수정하고 거스름돈은 250원으로 수정하였다.
- 이 경우 최적은 50원 동전을 5개 사용하는 것이다.
- 하지만 탐욕 알고리즘은 이 답을 찾지 못한다.
- 다음은 위 코드를 실행한 결과 화면이다.
- 60원 동전 4개, 5원 동전 2개를 사용한 것을 솔루션으로 출력한다. 즉 총 6개의 동전을 사용했다.
- 이처럼 탐욕 알고리즘은 최적의 솔루션(global optimum)을 도출하지 못할 수 있다.

```
In [105]: runfile('D:/data/재정/주식소스/stock/
result/untitled3.py', wdir='D:/data/재정/주식소스/
stock/result')
잔돈:  250
동전 종류 [60, 50, 5]
동전 개수 [4, 0, 2]
```

3. 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.

- 탐욕 알고리즘1
    - 무게 상관 없이, 가치가 가장 높은 물건부터 넣자 .
    - 단, 해당 물건을 넣을 때 배낭의 최대 무게를 초과하면 이 물건은 넣지 않는다.

- 탐욕 알고리즘2
    - 단위 무게 당 가치가 가장 높은 물건부터 넣자 .
    - 단, 해당 물건을 넣을 때 배낭의 최대 무게를 초과하면 이 물건은 넣지 않는다.

다. 탐욕 알고리즘1 (의사코드로 표현)

```
1. item 클래스 정의 및 객체 생성 : (name, value, weight)
      - ('A', 10, 5)
      - ('B', 30, 8),
      - ...
2. items 리스트에 생성된 item 객체들을 추가
      - items =  [('A', 10, 5), ('B', 30, 8), ('C', 20, 3), ('D', 14, 7), ('E', 23, 9)]
3. items 리스트를 item의 value를 기준으로 내림차순 정렬
      - sorted_items = [('B', 30, 8), ('E', 23, 9), ('C', 20, 3), ('D', 14, 7), ('A', 10, 5)]
4. 가치를 최대로 하는 조합 찾기
      cur_weight = 0
      cur_value = 0
      chosen_items = []
      for item in sorted_items :
            if (cur_weight  + item. weight <= max_weight) :
                  cur_value += item.value
                  cur_weight += item.weight
                  chosen_items.append(item)
      return (chosen_items, cur_value, cur_weight)
```

- 먼저 하나의 아이템을 나타내는 item 클래스를 정의한다. 아이템의 이름, 가치, 무게 등이 클래스의 멤버 변수가 된다.
- 입력으로 주어진 아이템들의 정보를 이용해 각 item 객체를 생성하고 이들을 리스트에 저장한다.
- 아이템이 저장된 리스트를 아이템의 가치를 기준으로 내림차순으로 정렬한다.
- 이후, 가치가 가장 높은 아이템부터 아이템을 추가할 때 배낭의 최대 무게를 초과하지 않는 한 해당 아이템을 배낭에 넣기로 선택한다. 즉 아이템의 무게와 가치를 총 무게와 총 가치에 더해 준다.
- 이를 모든 아이템에 대해 차례로 반복하면 chosen_items에는 선택된 아이템들이 저장되어 있고, cur_value에는 총 가치가 cur_weight에는 총 무게가 저장된다.
- 최종적으로 이들을 반환한다.

라. 최악 시간복잡도

- 1단계 객체를 생성하는 시간복잡도는 아이템의 개수만큼 실행되므로 O(N)이다.
- 2단계 객체를 리스트에 저장하는 시간복잡도도 아이템의 개수만큼 실행되므로 O(N)이다.
- 3단계 객체들을 정렬하는 시간복잡도는 정렬 알고리즘에 따라 O(NlogN) 또는 $O(N^2)$이다.
- 4단계 아이템들을 선택하는 시간복잡도는 아이템의 개수만큼 실행되므로 O(N)이다.
- 따라서 전체 알고리즘의 시간복잡도는 3단계 정렬에 의해 좌우되며, 어떤 정렬 알고리즘을 사용했느냐에 따라 O(NlogN) 또는 $O(N^2)$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
```

- item 클래스는 하나의 물건을 나타낸다. 멤버 변수로 name, value, weight를 가진다.

2) 2단계 - Knapsack 클래스 정의

```
class Knapsack(object):
        def __init__(self, names, values, weights, max_weight): # 아이템들을 생성함
                self.items = []
                self.max_weight = max_weight
                for i in range(len(names)) :
                        item = Item(names[i], values[i], weights[i])
                        self.items.append(item)

        def findBestCaseGreedy1(self):
                self.items.sort(reverse=True, key=lambda x:x.value)
                 value = 0
                 weight = 0
                 chosen_items = []
                 for item in self.items :
                         if (weight + item.weight <= self.max_weight) :
                                 chosen_items.append(item.name)
                                 weight += item.weight
                                 value += item.value

                  return (chosen_items, value, weight)
```

- 생성자(__init__())에서는 N개의 Item 객체를 생성하여, itesms 리스트에 저장한다.
- findBestCaseGreedy1()는 탐욕 알고리즘1을 사용하여 가치를 최대로 하는 물건의 조합을 구하는 함수이다.
- 먼저 아이템들을 가치를 기준으로 하여 내림차순으로 정렬한다. (sort())
- 멤버 변수 chosen_items는 가치를 최대로 하는 물건들의 조합을 저장한다.
- 각 아이템에 대해 아이템을 더해도 총 무게가 배낭의 허용 무게를 초과하지 않으면 해당 아이템을 선택한다. 즉 chosen_items에 해당 아이템의 이름을 추가하고 weight, value에 아이템의 무게와 가치를 각각 더한다.
- 최종적으로 선택된 아이템들과 가치, 무게를 반환한다.

3) 3단계 - Knapsack 테스트 코드

```
names = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
values = [10, 30, 20, 14, 23, 11, 15, 18]
weights = [5, 8, 3, 7, 9, 2, 6, 1]
max_weight = 20
knapsack = Knapsack(names, values, weights, max_weight)
(chosen_items, value, weight) = knapsack.findBestCaseGreedy1()
print(chosen_items, value, weight)
```

- 다음은 이 파이썬 코드를 스파이더에서 실행한 결과 화면이다.
- 배낭의 최대 무게가 20일 때, 가치를 최대로 하는 조합은 B, C, E를 선택한 경우이며 그때의 가치의 총합은 73이다.

```
In [107]: runfile('D:/data/재정/주식소스/stock/result/
untitled3.py', wdir='D:/data/재정/주식소스/stock/result')
['B', 'E', 'C'] 73 20
```

바. 탐욕 알고리즘2 (의사코드로 표현)

```
1. item 클래스 정의 및 객체 생성 : (name, value, weight)
2. items 리스트에 생성된 item 객체들을 추가
3. items 리스트를 item의 value/weight를 기준으로 내림차순 정렬
4. 가치를 최대로 하는 조합 찾기
        cur_weight = 0
        cur_value = 0
        chosen_items = []
        for item in sorted_items :
                if (cur_weight  + item. weight <= max_weight) :
                        cur_value += item.value
                        cur_weight += item.weight
                        chosen_items.append(item)
        return (chosen_items, cur_value, cur_weight)
```

- 탐욕 알고리즘2는 탐욕 알고리즘1과 거의 동일하다.
- 다른 부분은 3단계 아이템을 정렬하는 부분이다.
- 여기서는 단위무게당 가치를 기준으로 정렬을 수행한다.
- 나머지 부분은 알고리즘1과 동일하다.

사. 최악 시간복잡도

- 탐욕 알고리즘1과 동일한 시간복잡도를 갖는다.

아. 파이썬 구현 및 실행

1) 2단계 - Knapsack 클래스 정의

```
class Knapsack(object):
        def __init__(self, names, values, weights, max_weight): # 아이템들을 생성함
                self.items = []
                self.max_weight = max_weight
                for i in range(len(names)) :
                        item = Item(names[i], values[i], weights[i])
                        self.items.append(item)

        def findBestCaseGreedy2(self):
                self.items.sort(reverse=True, key=lambda x:(x.value/x.weight))
                value = 0
                weight = 0
                chosen_items = []
                for item in self.items :
                        if (weight + item.weight <= self.max_weight) :
                                chosen_items.append(item.name)
                                weight += item.weight
                                value += item.value

                return (chosen_items, value, weight)
```

- 탐욕 알고리즘2의 구현은 탐욕 알고리즘1과 거의 동일하다.
- 차이나는 부분은 findBestCaseGreedy2() 함수에서 정렬을 수행하는 부분이다.

- 즉 정렬 기준으로 단위무게 당 가치를 사용한다.
- 그 외 코드는 모두 동일하다.

2) 3단계 - Knapsack 테스트 코드

```
names = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
values = [10, 30, 20, 14, 23, 11, 15, 18]
weights = [5, 8, 3, 7, 9, 2, 6, 1]
max_weight = 20
knapsack = Knapsack(names, values, weights, max_weight)
(chosen_items, value, weight) = knapsack.findBestCaseGreedy1()
print(chosen_items, value, weight)
(chosen_items, value, weight) = knapsack.findBestCaseGreedy2()
print(chosen_items, value, weight)
```

- 다음은 이 파이썬 코드를 스파이더에서 실행한 결과 화면이다.
- 배낭의 최대 무게가 20일 때, 탐욕 알고리즘2가 찾아낸 최적의 조합은 B, C, F, G, H를 선택한 경우이며 그때의 가치의 총합은 94이다.

```
In [108]: runfile('D:/data/재정/주식소스/stock/result/
untitled3.py', wdir='D:/data/재정/주식소스/stock/result')
['B', 'E', 'C'] 73 20
['H', 'C', 'F', 'B', 'G'] 94 20
```

연습문제

1. 탐욕 알고리즘의 개념을 설명하시오.

정답 : 탐욕 알고리즘은 최적해를 찾기 위한 방법 중 하나로서, 현재의 조건에서 가장 최적으로 생각되는 결정을 함으로써 해를 구하는 방법

해설 : 정답 참조

2. brute-force 기법 대비 탐욕 알고리즘의 장점과 단점을 설명하시오.

정답 : 탐욕 알고리즘은 가능한 모든 경우를 검사하지 않고, 최적으로 생각되는 일부 경우만을 탐색하므로 시간복잡도가 낮은 장점이 있다. 하지만 경우에 따라 가장 최적인 해를 구하지 못할 수 있는 단점이 있다.

해설 : 정답 참조

3. Knapsack 문제를 해결하는 탐욕 알고리즘을 제시하시오.

정답 : 무게당 가치가 큰 아이템부터 순서대로 knapsack에 추가한다. 이 때 아이템을 추가한 무게가 최대 무게를 초과하면 해당 아이템은 knapsack에 추가하지 않는다.

해설 : 정답 참조

정리하기

1. 탐욕 알고리즘은 최적해를 찾기 위한 방법 중 하나로서, 현재의 조건에서 가장 최적으로 생각되는 결정을 함으로써 해를 구하는 방법이다.
2. 탐욕 알고리즘은 brute-force에 비해 낮은 시간복잡도를 갖는 장점이 있지만, 최적해를 구하지 못할 수 있다.
3. 거스름돈을 구하는 알고리즘을 탐욕 알고리즘을 활용하여 설계할 수 있다.
4. Knapsack을 해결하는 알고리즘을 탐욕 알고리즘을 활용하여 설계할 수 있다.

참고자료

- 파이썬 알고리즘, 최영규, 2021, 생능출판

다음 차시 예고

- 몬테카를로 시뮬레이션을 활용한 문제들에 대해 학습한다.