2018년도 가을학기 SCSC 알고리즘 실습 5주차

문현수, munhyunsu@cs-cnu.org

Thursday October 04, 2018

1 개요

컴퓨터공학에서 '큰 문제를 작은 문제로 나누어 해결'하는 방법은 문제를 해결하는 패러다임의 큰 축을 담당한다. 이 때 큰 문제를 '작은 문제'로 나누는 방향으로 기법이 나뉘어지기도 한다. 지난 실습에서 다루었었던 재귀는 큰 문제를 분할해가며 작은 문제로 가는 것으로 볼 수 있다. 반대로 이번 실습에서 다를 **동적 프로그래밍**은 작은 문제를 합쳐 큰 문제로 나아가는 것이다. 생각의 방향이 사람과 일치하기에 이해는 쉽지만 작은 문제와 그 해답이 큰 문제로 향하는지 아는 것은 어렵다. 동적 프로그래밍, 즉 DP는 재귀와 마찬가지로 알고리즘이라기 보다는 문제 해결 '방법'에 가깝다.

동적 프로그래밍(Dynamic Programming)은 작은 문제를 해결하고 그 해답을 이용하여 더 큰 문제를 해결하는 방법이다. Optimal structure라고 불리는 작은 문제에 대한 해답이 합쳐져서 조금 더 큰 문제에 대한 Optimal structure를 계산해낸다. 이러한 방법의 연속으로 큰 문제를 해결할 수 있다.

동적 프로그래밍은 설계 및 구현이 어렵지만 그 성능이 좋다고 알려져있다. 이번 실습은 동적 프로그래밍의 성능이 다른 기법(재귀)과 얼마나 차이나는지 비교하는 것까지 포함한다. `matplotlib`를 활용하여 그래프를 그려본다.

2 거스름돈 최소 동전 계산 프로그램 with DP

우리는 이번 실습에서 지난 실습과 마찬가지로 거스름돈을 주기위하여 몇 개의 동전을 주어야하는지 계산하는 프로그램을 만들어낸다. 하지만 동시에 '최소 동전 수'뿐 아니라 '동전 종류'도 계산이 되어야 한다. 계산 결과는 '최소 동전 수'와 이에 대한 '동전 종류'여야 한다. 예를 들어 10, 50, 100, 500원 동전이 있는 우리나라에서 2400원을 거슬러주려면 500원 4개, 100원 4개를 주어야 할 것이다. 만약 100원 24개를 주면 액수는 맞지만 최소 동전 수에는 맞지 않는다.

계산의 편의성과 문제 복잡도를 위하여 동전은 1, 5, 10, 21, 25원이 있다고 하자. 이 조건에서 최소 동전 수를 찾는 프로그램은 아래와 같이 표현할 수 있다. 지난

실습 때 사용한 재귀와 달리 DP를 사용하여 문제를 해결하고 있다. 동적 프로그래밍은 원리는 간단하지만 연산 복잡도가 상당히 높기 때문에 코드만으로 이해하기 쉽지 않다. 종이에 리스트를 그려두고 코드를 따라가보며 이해해야 한다.

Listing 1: make_change() with DP

```
1  def make_change(coin_value_list, coin_change, min_coins, coins_used):
2      global counter
3      for cents in range(0, coin_change+1):
4          coin_count = cents
5          new_coin = 1
6          coins = list()
7          for coin in  coin_value_list :
8              if coin <= cents:
9                  coins.append(coin)
10         for coin in coins:
11             counter = counter + 1
12             if min_coins[cents - coin] + 1 < coin_count:
13                 coin_count = min_coins[cents - coin] + 1
14                 new_coin = coin
15         min_coins[cents] = coin_count
16         coins_used[cents] = new_coin
17     return min_coins[coin_change]
18
19
20 def print_coins(coins_used, coin_change):
21     coin = coin_change
22     while coin > 0:
23         this_coin = coins_used[coin]
24         print(this_coin)
25         coin = coin - this_coin
26
27
28 counter = 0
29
30
31 def main():
32     """
33      잔돈     거스르기
34     """
35      coin_value_list = [1, 5, 10, 21, 25]  # 동전의   종류
36     coin_change = input('Change: ') #  거스름돈
```

```
37      coin_change = int(coin_change) #    인트형으로      바꾸자 !
38      coins_used = [0] * (coin_change + 1)
39      coin_count = [0] * (coin_change + 1)
40
41      print('Making change for', coin_change, 'requires')
42      print(make_change(coin_value_list, coin_change, coin_count, coins_used), 'coins')
43      print('They are:')
44      print_coins(coins_used, coin_change)
45      print('The used list is as follows:')
46      print(coins_used)
47      print(counter)
48
49
50  if __name__ == '__main__':
51      main()
```

코드를 읽어보자! DP를 이용한 동전 문제는 재귀와는 반대로 '작은 문제'부터 계산해나가며 '큰 문제(목표)'로 가고 있다. 3번 라인의 `for cents in range(0, coin_change+1):`은 작은 문제부터 시작하여 목표까지 가는 반복문이다. 알고리즘을 공부하는 학생들은 이처럼 각 라인이 '어떤 역할'을 하는지 분석하며 공부해야한다. 지난 실습 때 공부한 재귀와 DP 모두 코드를 훑어 봐서는 이해하기 어려우므로 라인별로 어떤 의미가 담겨있는지 이해하는 것이 필수다. 이 코드를 실행하여 63원을 입력하면 아래와 같은 결과가 나온다.

Listing 2: make_change() with DP result

```
Change: 63
Making change for 63 requires
3 coins
They are:
21
21
21
The used list is as follows:
[1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 10, 1, 1, 1, 1, 5, 1, 1, 1, 1, 10, 21, 1, 1,
    1, 25, 1, 1, 1, 1, 5, 10, 1, 1, 1, 10, 1, 1, 1, 1, 5, 10, 21, 1, 1, 10,
    21, 1, 1, 1, 25, 1, 10, 1, 1, 5, 10, 1, 1, 1, 10, 1, 10, 21]
258
```

코드를 따라가며 손으로 적어둔 리스트와 결과 리스트가 일치하는지 확인해봐야한다. 함수를 통해 변경된 리스트를 이용하여 `print_coins()`함수는 '동전 종류'를 출력한다. coins_used에는 index에 해당하는 잔돈을 거슬러주기 위하여 어떤

동전이 '추가되었나' 저장되어있다. 추가된 동전을 back-tracking하여 동전들을 출력한다.

3 Draw Line Chart using Matplotlib.pyplot

python 3에서는 `time.time()`을 이용하여 시간을 측정할 수 있다. 우리가 구현한 같은 기능을 하는 재귀, 메모리 재귀, DP 동전 거슬러주기 프로그램은 시간 복잡도, 공간 복잡도에서 성능차이가 존재하므로 비교해볼만 할 것이다. 성능 비교를 하기 전 그래프를 그릴 수 있는 도구를 설치하고 간단한 예제를 수행해보자. 이후 과제로 각 함수 성능을 비교할 것이다.

3.1 Install Matplotlib

`matplotlib`는 python 3에서 가장 다루기 쉬운 그래프 작성 도구다. Jupyter, Web app 등에서 활용되며 python 3에 기본적으로 설치되지 않으므로 직접 설치해주어야 한다. PyCharm을 사용하는 우리는 venv 환경에서 python 3를 사용하고 있기 때문에 각 프로젝트별로 `matplotlib`를 설치해주어야 한다. 프로젝트를 연 후 File-Settings-Project-Project Interpreter를 선택하면 현재 프로젝트에 설치되어있는 각종 도구가 표시된다. +마크를 눌러 'matplotlib'을 검색한 후 Install package를 눌러 설치하자. 그러면 Figure 1과 같이 설치된 도구에 matplotlib이 표시된다.

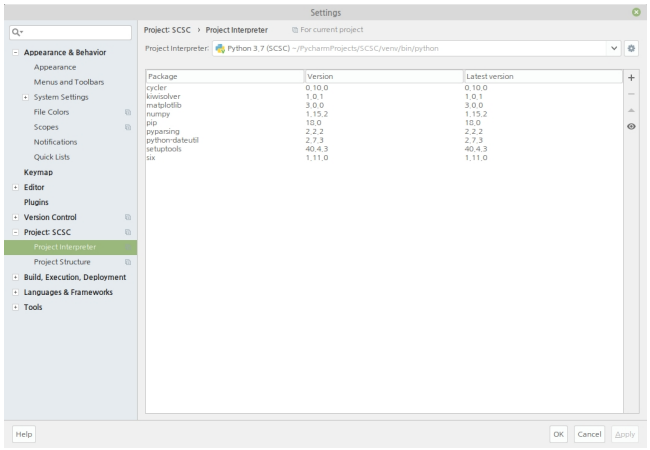

Figure 1: matplotlib이 설치된 후의 Project Interpreter 내용

3.2 Draw Line Chart

`matplotlib`이 설치되었으므로 간단한 라인 차트 예제를 수행해보자. 우리는 라인 차트를 통해 x축에 따라서 변화하는 y값을 쉽게 살펴볼 수 있다. 만약 차트에 여러 개의 선이 그어져있다면 선 사이의 비교도 할 수 있을 것이다. 성능 비교가 우리의

최종 목표이므로 시간 복잡도에 따른 성능 비교 차트를 그려보자.

Listing 3: Draw line chart using matplotlib - draw_graph_practice.py

```
1  import time
2  import matplotlib.pyplot as plt
3
4
5  def func_logn(num_loop):
6      while num_loop > 0:
7          num_loop = num_loop // 2
8
9
10 def func_n(num_loop):
11     for index in range(0, num_loop):
12         pass
13
14
15 def func_n2(num_loop):
16     for index in range(0, num_loop):
17         for index2 in range(0, num_loop):
18             pass
19
20
21 def main():
22     x_data = range(0, 101, 1)
23     y_data1 = list()
24     y_data2 = list()
25     y_data3 = list()
26
27     for x in x_data:
28         start_time = time.time()
29         func_logn(x)
30         end_time = time.time()
31         execution_time = end_time - start_time
32         y_data1.append(execution_time)
33
34         start_time = time.time()
35         func_n(x)
36         end_time = time.time()
37         execution_time = end_time - start_time
38         y_data2.append(execution_time)
```

```
39
40              start_time = time.time()
41              func_n2(x)
42              end_time = time.time()
43              execution_time = end_time - start_time
44              y_data3.append(execution_time)
45
46          plt.plot(x_data, y_data1, 'r.-', label='Log_N')
47          plt.plot(x_data, y_data2, 'g.-', label='N')
48          plt.plot(x_data, y_data3, 'b.-', label='N**2')
49          plt.legend(bbox_to_anchor=(0.05, 0.95), loc=2, borderaxespad=0.0)
50          plt.title('Time_Complexity')
51          plt.ylabel('Execution_time_(sec)')
52          plt.xlabel('N')
53          plt.savefig('time_complexity.png', bbox_inches='tight')
54          # plt.show()
55          plt.close()
56
57
58  if __name__ == '__main__':
59      main()
```

우선 측정할 함수인 `func_logn()`, `func_n()`, `func_n2()`를 정의한다. 각 함수는 시간 복잡도에 맞게 N에 대한 반복문이 정의되어 있다. 메인 함수에서는 N을 입력해주며 각 함수 실행 전, 후에 시간을 측정한다. 이것을 `y_data` 리스트에 추가(`append()`)함으로 그래프 그릴 준비를 마친다. 그래프는 `plt.plot()`을 통해 matplotlib에 데이터를 전달해준다. 각종 범례 및 제목, 축 라벨링을 한 후 `plt.show()` 혹은 `plt.savefig()`을 불러주면 그래프를 볼 수 있다. 이 예제를 수행하면 Fig. 2와 같이 time_complexity.png라는 이름으로 그래프가 저장된다.

4 과제

지난 실습에서 우리는 거스름돈에 대한 '최소 동전 갯수'를 출력해보았다. 그리고 이번 실습에서 DP를 활용한 '최소 동전 갯수 및 종류'를 출력해보았다. 이번 과제는 '동전 종류'를 출력하는 `print_coins()`를 제거하고 지난 과제에서 연습한 `known_result`를 적용하는 것이다. `coins_used`를 사용하지 않고 `known_result`에 동전의 종류를 직접 저장하라. 과제를 위한 `main()`은 아래와 같으며 그 출력 결과도 함께 보인다.

Listing 4: change_money_dp_list.py skeleton code

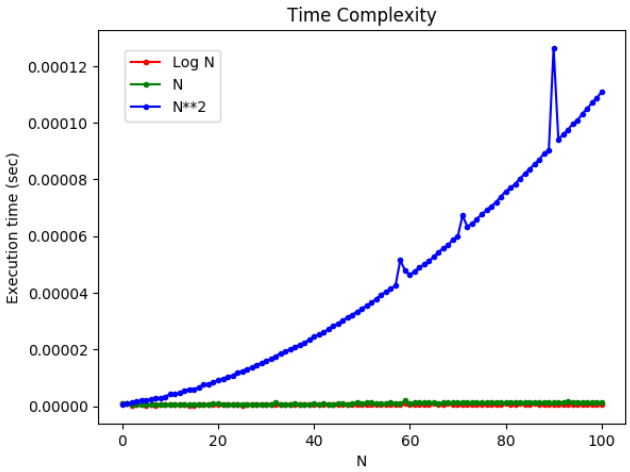

Figure 2: time_complexity.png

```
1  def make_change(coin_value_list, coin_change, known_result):
2      pass
3
4
5  def main():
6       coin_value_list  = [1, 5, 10, 21, 25]
7      coin_change = input('Change: ')
8      coin_change = int(coin_change)
9      known_result = [[]] * (coin_change+1)
10
11     print('Making change for', coin_change, 'requires')
12     print(make_change(coin_value_list, coin_change, known_result), 'coins')
13     print('They are:')
14     print(known_result[coin_change])
15     print('The used list is as follows:')
16     print(known_result)
17
18
19 if __name__ == '__main__':
20     main()
```

Listing 5: change_money_dp_list.py result

```
Change: 63
Making change for 63 requires
```

```
3 coins
They are:
[21, 21, 21]
The used list is as follows:
[[], [1], [1, 1], [1, 1, 1], [1, 1, 1, 1], [5], [1, 5], [1, 1, 5], [1, 1, 1,
    5], [1, 1, 1, 1, 5], [10], [1, 10], [1, 1, 10], [1, 1, 1, 10], [1, 1,
    1, 1, 10], [5, 10], [1, 5, 10], [1, 1, 5, 10], [1, 1, 1, 5, 10], [1, 1,
    1, 1, 5, 10], [10, 10], [21], [1, 21], [1, 1, 21], [1, 1, 1, 21], [25],
    [1, 25], [1, 1, 25], [1, 1, 1, 25], [1, 1, 1, 1, 25], [5, 25], [10, 21],
    [1, 10, 21], [1, 1, 10, 21], [1, 1, 1, 10, 21], [10, 25], [1, 10, 25],
    [1, 1, 10, 25], [1, 1, 1, 10, 25], [1, 1, 1, 1, 10, 25], [5, 10, 25],
    [10, 10, 21], [21, 21], [1, 21, 21], [1, 1, 21, 21], [10, 10, 25], [21,
    25], [1, 21, 25], [1, 1, 21, 25], [1, 1, 1, 21, 25], [25, 25], [1, 25,
    25], [10, 21, 21], [1, 10, 21, 21], [1, 1, 10, 21, 21], [5, 25, 25],
    [10, 21, 25], [1, 10, 21, 25], [1, 1, 10, 21, 25], [1, 1, 1, 10, 21,
    25], [10, 25, 25], [1, 10, 25, 25], [10, 10, 21, 21], [21, 21, 21]]
```

동적 프로그래밍을 이용한 구현이 잘 되었는지 확인할 수 있는 테스트는 다음과 같다.

Listing 6: change_money_dp_list_test.py

```
1  import unittest
2  import random
3
4  from change_money_dp_list import make_change
5
6
7  class TestMakechange(unittest.TestCase):
8      def test_makechange(self):
9          coin_value_list  = [1, 5, 10, 21, 25]
10         for change in range(0, 101, 1):
11             random.shuffle( coin_value_list )
12             known_result = list()
13             for index in range(0, change + 1):
14                 known_result.append([])
15             result  = make_change(coin_value_list, change, known_result)
16             print(sum(known_result[change]) == change, known_result[
                   change], change)
17             self.assertEqual(sum(known_result[change]), change)
18
19     def test_63(self):
```

```
20              coin_value_list  = [1, 5, 10, 21, 25]
21              random.shuffle( coin_value_list )
22              known_result = list()
23              for index in range(0, 63 + 1):
24                  known_result.append([])
25              make_change(coin_value_list, 63, known_result)
26              self.assertEqual([21, 21, 21], known_result[63])
27
28          def test_12(self):
29              coin_value_list  = [1, 2, 3, 5, 7]
30              random.shuffle( coin_value_list )
31              known_result = list()
32              for index in range(0, 12 + 1):
33                  known_result.append([])
34              make_change(coin_value_list, 12, known_result)
35              known_result[12].sort()
36              self.assertEqual([5, 7], known_result[12])
37
38
39      if __name__ == '__main__':
40          unittest.main(verbosity=2)
```

4.1 과제 목표

- 10원 이상 예제 손으로 따라가서 그려보기(워드 가능)
- change_money_dp_list.py 구현
- 재귀, 메모리 재귀, DP 성능 비교
- (선택) 시간 복잡도, 공간 복잡도 관점으로 성능 비교 그래프 분석

4.2 제출 관련

- 마감 날짜: 2018. 10. 10. 23:59:59
- 딜레이: 1일당 10% 감점(처음 2일까지는 -2)
- 제출 방법: 과목 사이버캠퍼스
- 제출 형식: 과제 리포트 PDF(HWP, DOC 받지 않음!), 소스코드(구현한 .py 만 추가할 것)를 압축한 .zip 파일
- 리포트에 포함해야하는 내용: 목표, 목표를 위해 알아야하는 것, 해결 방법, 결과, (선택)느낀점 or전달할 말

- 제출 파일 제목: AL_201550320_문현수_05.zip(파일명 준수!)

4.3 조교 연락처

- 문현수
- munhyunsu@cs-cnu.org
- 공학5호관 633호 데이터네트워크연구실
- 이메일, 연구실 방문