2018년도 가을학기 SCSC 알고리즘 실습 4주차

문현수, munhyunsu@cs-cnu.org

Thursday September 27, 2018

1 개요

다양한 알고리즘은 같은 문제를 해결하더라도 시간 복잡도 혹은 공간 복잡도를 낮추어 성능을 올리는데에 목적이 있다. 이러한 알고리즘의 세계에서 중심이 되는 철학은 '큰 문제는 작은 문제로 나눌 것'이다. 직관적으로 해결되거나 어느정도 감당할 수 있을 정도의 수준으로 문제를 나누고 작은 해답을 합쳐 큰 문제에 대한 해답을 구해낸다. 앞으로 우리가 배울 다양한 알고리즘은 이 철학에서 크게 벗어나지 않을 것이다.

재귀는 함수 혹은 클래스 등이 자기 자신을 다시 호출하는 것을 말한다. 대게 콜스택이 지나치게 증가하여 좋은 프로그래밍 기법이라고 이야기되지 않지만 가독성이나 구현의 편의성을 위하여 성능을 포기할 때가 있다. 특히 피보나치 수열이나 하노이의 탑을 구현하는 문제는 많은 프로그래밍 교재에서 재귀를 연습할 때 다루어진다.

재귀적 기법으로 프로그램을 구현할 때에는 '종료 조건'과 '종료 조건으로 가는 식'이 필요하다. 이러한 특징은 위에서 이야기한 '큰 문제를 작은 문제로 나누는 철학'과 일치한다. 큰 문제를 '종료 조건으로 가는 식'이라는 작은 문제로 나누어 최종적으로 '종료 조건'이라는 가장 작은 문제로 분할하는 것이다. 이번 실습에서는 피보나치 수열보다 종료 조건이 좀 더 복잡한 '거스름돈 계산' 프로그램을 만들어 본다. 더불어 실행 시간과 함수 호출 횟수를 세아려 얼마나 오래 그리고 얼마나 많이 호출되었는지 측정한다.

2 거스름돈 최소 동전 계산 프로그램

우리는 이번 실습에서 거스름돈을 주기위하여 몇 개의 동전을 주어야하는지 계산하는 프로그램을 만들어낸다. 이 때 계산 결과는 '최소 동전 수'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.

2.1 Recursion without Memory(Caching)

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. 재귀를 사용하여 문제를 해결하고 있다. 피보나치 수열에 비교하여 그 복잡도가 상당히 높기 때문에 눈으로 결과를 쫓기보다는 손으로 그려가며 계산해보는 것이 좋다.

Listing 1: make_change() without memory

```
1  import time
2  import random
3
4
5  counter = 0
6
7
8  def make_change(coin_value_list, coin_change):
9      global counter
10     counter = counter + 1
11     min_coins = coin_change
12
13     if coin_change in coin_value_list :
14         return 1
15     else:
16         coins = list()
17         for coin in coin_value_list :
18             if coin <= coin_change:
19                 coins.append(coin)
20         for coin in coins:
21             num_coins = 1 + make_change(coin_value_list, coin_change - coin)
22             if num_coins < min_coins:
23                 min_coins = num_coins
24     return min_coins
25
26
27 def main():
28      coin_value_list  = [1, 5, 10, 21, 25]
29     random.shuffle( coin_value_list )
30     coin_change = input('Change:␣')
31     coin_change = int(coin_change)
32
33     start = time.time()
```

```
34      print(make_change(coin_value_list, coin_change))
35      end = time.time()
36      print(counter)
37      return_time = end-start
38      print('Time: {0:7.3f}'.format(return_time))
39
40
41  if __name__ == '__main__':
42      main()
```

이 알고리즘에서 계산해야하는 목표 잔액을 동전을 하나씩 빼가며 '작은 문제'로 만들고 있다. '작은 문제'가 동전의 가치와 일치할 정도로 작게 쪼개지면 그 때부터 계산을 시작한다. 우리는 '최소 동전'이라는 조건에 만족하는지 확인하기 위하여 63원을 반환하도록 입력해보았다. 63은 '큰 동전부터 내어주기' 알고리즘으로 해결하면 25원 2개, 10원 1개, 1원 3개로 총 6개가 결과로 나온다. 하지만 정확한 답은 21원 3개로 총 3개의 동전을 거슬러주면 된다.

Listing 2: make_change() without memory result: 63

```
1  Change: 63
2  3
3  33160648
4  Time:  17.499
```

결과를 살펴보면 63원을 계산하기 위하여 무려 33,160,648번의 함수 호출이 있었다. 또한 17.499초나 걸리고 있다. 왜 이럴까? 손으로 직접 따져보면 금방 알 수 있으니 해보자. 이유는 '같은 연산'을 여러번 반복하기 때문이다. 예제로 입력한 63을 계산하기 위하여 같은 연산을 최대 5,083,416번 반복하고 있다(궁금하면 코드를 추가하여 계산해보자.).

2.2 Recursion with Memory(Caching)

재귀의 단점으로 지목되는 호출 스택은 피할 수 없으니 무시하더라도 '같은 연산'을 여러번 반복하는 것은 비효율적이다. 컴퓨터가 같은 작업을 하는데에 특화되어 있어도 연산 시간이 오래걸리면 사용자에게도 불편하다. 따라서 같은 연산을 가능한 안 하는 방법이 필요하다. 이럴 때 사용하는 방법이 '메모리 혹은 캐싱'이다. 예를 들어 6원을 반환하기 위하여 5원 1개, 1원 1개를 계산하였다고 하자. 이것을 '저장' 해두었다가 나중에 6원을 계산하려고 할 때에 재계산없이 동전 2개가 필요하다고 저장된 값을 '불러올 수'있다. 이러한 방법을 '메모리 혹은 캐싱'이라고 부른다.

위의 코드를 메모리를 적용하여 다시 적어보면 아래와 같다. `known_result`라고 이름진 변수를 매 함수 호출 인자로 전달해주어 계산했었던 결과를 '저장'한다. 그 결과 같은 연산을 하더라도 월등히 빨라진다.

Listing 3: make_change() with memory

```
1  import time
2  import random
3
4
5  counter = 0
6
7
8  def make_change(coin_value_list, coin_change, known_result):
9      global counter
10     counter = counter + 1
11     min_coins = coin_change
12
13     if coin_change in coin_value_list :
14         known_result[coin_change] = 1
15         return 1
16     elif known_result[coin_change] > 0:
17         return known_result[coin_change]
18     else:
19         coins = list()
20         for coin in coin_value_list :
21             if coin <= coin_change:
22                 coins.append(coin)
23         for coin in coins:
24             num_coins = 1 + make_change(coin_value_list, coin_change - coin, known_result)
25             if num_coins < min_coins:
26                 min_coins = num_coins
27                 known_result[coin_change] = min_coins
28     return min_coins
29
30
31 def main():
32      coin_value_list = [1, 5, 10, 21, 25]
33     random.shuffle( coin_value_list )
34     coin_change = input('Change: ')
35     coin_change = int(coin_change)
36
37     start = time.time()
38     print(make_change(coin_value_list, coin_change, [0] * (coin_change+1)))
39     end = time.time()
```

```
40      print(counter)
41      return_time = end-start
42      print('Time: {0:7.3f}'.format(return_time))
43
44
45  if __name__ == '__main__':
46      main()
```

함수 호출은 262번으로 줄었으며, 0.001초만에 결과를 찾아내었다. 호출 스택 문제는 여전히 남아있겠지만 같은 연산 문제는 해결되었다.

Listing 4: make_change() with memory result: 63

```
1  Change: 63
2  3
3  262
4  Time:   0.001
```

3 거스름돈 동전 계산하기

실습으로 작성해본 프로그램은 최소 '동전 갯수'를 찾아낸다. 이 프로그램이 자판기나 POS같은 곳에 들어간다고 하면 동전 갯수만으로는 결과가 부족할 것이다. 손님에게 정확한 액수의 최소 동전을 반환해주어야 한다. 따라서 이번 과제는 최소 동전 갯수뿐 아니라 어떤 동전을 반환해주어야하는지도 계산해본다.

3.1 거스름돈 동전 계산 프로그램 without Memory

우선 실습에서 연습했던 대로 메모리 기능이 없는 프로그램먼저 접근해보도록 하자. 이 프로그램의 결과를 테스트하는 코드는 아래와 같다.

Listing 5: make_change without memory test code - change_money_list_test.py

```
1  import unittest
2  import random
3
4  from change_money_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, 51, 1):
11                random.shuffle( coin_value_list )
12                result  = make_change(coin_value_list, change)
13                print(sum(result[2]) == change, result[2], change)
14                self.assertEqual(change, sum(result[2]))
15
16        def test_63(self):
17            coin_value_list  = [1, 5, 10, 21, 25]
18            random.shuffle( coin_value_list )
19            result  = make_change(coin_value_list, 63)[2]
20            self.assertEqual([21, 21, 21], result)
21
22        def test_12(self):
23            coin_value_list  = [1, 2, 3, 5, 7]
24            random.shuffle( coin_value_list )
25            result  = make_change(coin_value_list, 12)[2]
26            result.sort()
27            self.assertEqual([5, 7], result)
28
29
30    if __name__ == '__main__':
31        unittest.main(verbosity=2)
```

모든 경우에 대한 '최소 갯수'와 '금액'을 정확하기 테스트하기 어려워 몇가지 사례(Case)만 테스트하기로 한다. `test_makechange()`는 0원부터 50원까지 '금액'을 테스트한다. 이후 2가지 테스트는 '최소 갯수'를 테스트한다. 스켈레톤 코드는 아래에서 제공한다.

Listing 6: make_change without memory skeleton code - change_money_list.py

```
1   import time
2   import random
3
4
5   def make_change(coin_value_list, coin_change, counter = 0):
6       pass
7
8
9   def main():
10      coin_value_list  = [1, 5, 10, 21, 25]
11      random.shuffle( coin_value_list )
12      coin_change = input('Change: ')
```

```
13      coin_change = int(coin_change)
14
15      start  = time.time()
16      print(make_change(coin_value_list,  coin_change))
17      end = time.time()
18      return_time = end-start
19      print('Time:_{0:7.3f}'.format(return_time))
20
21
22  if  __name__ == '__main__':
23      main()
```

테스트 코드를 통과하는 코드를 완성하여 실행시켜보면 결과가 출력된다.

Listing 7: change_money_list.py result

```
1   거스름돈  : 63
2  (3,  33160648, [21,  21,  21])
3  Time:  26.283
```

3.2 거스름돈 동전 계산 프로그램 with Memory

거스름돈 동전을 계산하기는 했으나 지나치게 오래걸린다. 메모리 기능을 구현하여 수행 속도를 개선해보자.

Listing 8: make_change with memory test code - change_money_saved_list_test.py

```
1   import unittest
2   import random
3
4   from change_money_saved_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(None)
15              result = make_change(coin_value_list, change, known_result)
16              print(sum(result[2]) == change, result[2], change)
17              self.assertEqual(sum(result[2]), 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(None)
25          result = make_change(coin_value_list, 63, known_result)[2]
26          self.assertEqual([21, 21, 21], result)
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(None)
34          result = make_change(coin_value_list, 12, known_result)[2]
35          result.sort()
36          self.assertEqual([5, 7], result)
37
38
39  if __name__ == '__main__':
40      unittest.main(verbosity=2)
```

메모리 기능이 없는 테스트 코드와 마찬가지로 모든 경우에 대한 '최소 갯수'와 '금액'을 정확하기 테스트하기 어려워 몇가지 사례(Case)만 테스트하기로 한다. `test_makechange()`는 0원부터 50원까지 '금액'을 테스트한다. 이후 2가지 테스트는 '최소 갯수'를 테스트한다. 스켈레톤 코드는 아래에서 제공한다.

Listing 9: make_change with memory skeleton code - change_money_saved_list.py

```
1  import time
2  import random
3
4
5  def make_change(coin_value_list, coin_change, known_result, counter=0):
6      pass
7
```

```
8
9  def main():
10      coin_value_list  = [1, 5, 10, 21, 25]
11      random.shuffle( coin_value_list )
12      coin_change = input(' 거스름돈 : ')
13      coin_change = int(coin_change)
14      known_result = list()
15      for index in range(0, coin_change+1):
16          known_result.append(None)
17
18      start = time.time()
19      print(make_change(coin_value_list, coin_change, known_result))
20      end = time.time()
21      return_time = end-start
22      print('Time: {0:7.3f}'.format(return_time))
23
24
25  if __name__ == '__main__':
26      main()
```

테스트를 통과하는 코드를 수행해보면 메모리 기능이 없는 프로그램보다 빠르게 같은 결과가 나온다.

Listing 10: change_money_saved_list.py result

```
1  거스름돈 : 63
2  (3, 244, [21, 21, 21])
3  Time:  0.002
```

4 과제

실습에서 우리는 거스름돈에 대한 '최소 동전 갯수'를 출력해보았다. 이번 과제는 '최소 동전 갯수'를 넘어 '반환 해야 하는 동전 종류와 수'를 출력한다.

4.1 과제 목표

- 10원 이상 예제 손으로 따라가서 그려보기(워드 가능)
- change_money_list.py 구현
- change_money_saved_list.py 구현

4.2 제출 관련

- 마감 날짜: 2018. 10. 03. 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_04.zip(파일명 준수!)

4.3 조교 연락처

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