알고리즘 실습

181004 - Dynamic Programming

오늘의 목표

- Dynamic Programming 맛보기
  - 거스름돈 반환
- Recursion 다시보기
  - 거스름돈 반환 예제

Feedback

지난 과제: Queue 응용 프로그램

- 제출: 33 / 41 (80.49%)
  - 손으로 그리기 누락 8명!(24%)
  - 지난 실습 자료를 보면…
- 질문: 200개: 신기록!

과제) 기능이 개선된 재귀(with/without M.)

1. change_money_list.py 구현
   - change_money_list_test.py 로 테스트
2. change_money_saved_list.py 구현
   - change_money_saved_list_test.py 로 테스트

4.1 과제 목표

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

Feedback

- 어떤 질문이 이렇게나 많았는가?
  - 프로그램의 전체는 이해되지만 Line by line은 이해가 안 됨
  - 왜 이 code ‘line’이 필요한지 이해 부족
- counter를 왜 지역변수로 만드는가?
  - https://stackoverflow.com/questions/45291651/is-it-better-to-use-local-or-global-variables
- 복습, 요점 정리 시간

Feedback

- 테스트코드도 제출코드에 포함:
  구현한.py + 제공된_test.py
- 쉬는 시간에 대하여…
- 메일/연구실 방문

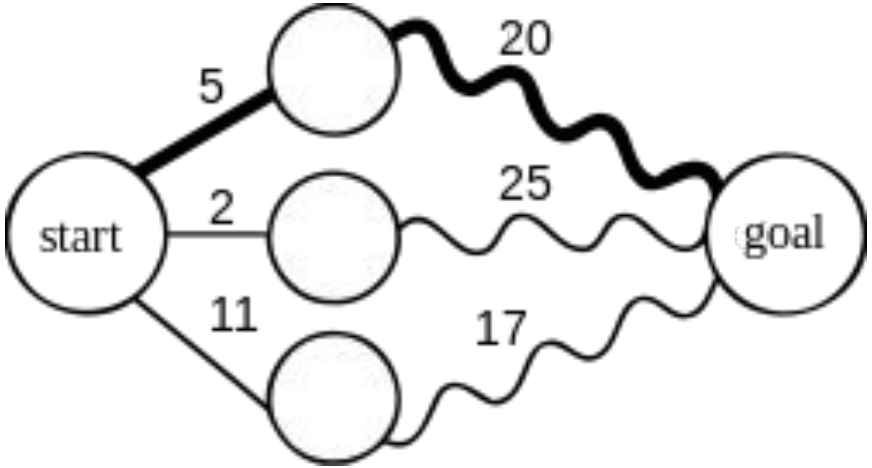
(또 간단한) 실습: 잔돈 거슬러주기 Ver. 2

잔돈 거슬러주기: 목표

- 잔돈을 최소한의 동전 수로 되돌려주자!
- 동적 프로그래밍 이용

Dynamic Programming

- Optimal substructure라고도 불림
- '작은 문제'에 대한 '최적의 솔루션'을 쌓아나가 큰 문제를 해결



잔돈 거슬러주기 DP 알고리즘

- 가장 작은 문제부터 '목표'까지 계산
- 이전에 계산해둔 '결과'를 재활용

잔돈 거슬러주기 코드 DP (이론 교재)

- 코딩해보자!
- Google python coding style:
  - https://github.com/google/styleguide/blob/gh-pages/pyguide.md
  - 지난 시간에 요즘 스타일로 변환한 예제를 참고하면 스스로 할 수 있음

```
def dpMakeChange(coinValueList,change,minCoins,coinsUsed):
    global counter
    for cents in range(change+1):
        coinCount = cents
        newCoin = 1
        for j in [c for c in coinValueList if c <= cents]:
            counter = counter + 1
            if minCoins[cents-j] + 1 < coinCount:
                coinCount = minCoins[cents-j]+1
                newCoin = j
        minCoins[cents] = coinCount
        #print("minCoins for ", cents, "is", coinCount)
        coinsUsed[cents] = newCoin
        #print("coinUsed for ", cents, "is", newCoin)
    return minCoins[change]

def printCoins(coinsUsed,change):
    coin = change
    while coin > 0:
        thisCoin = coinsUsed[coin]
        print(thisCoin)
        coin = coin - thisCoin
```

잔돈 거슬러주기 코드 DP (이론 교재)

- 작성한 코드를 호출

```
def main():
    amnt = 63
    clist = [1,5,10,21,25]
#    clist = [1,5,10,25]

    coinsUsed = [0]*(amnt+1)
    minCoins = [0]*(amnt+1)

    print("Making change for",amnt,"requires",
          dpMakeChange(clist,amnt,minCoins,coinsUsed),"coins")
    print(printCoins(coinsUsed,amnt))
    print("The used list is as follows:")
    print(coinsUsed)
    print(counter)

main()
```

```
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
```

잔돈 거슬러주기 DP: code

- inline code 등을 제거한 버전
- 손으로 리스트를 그리고 채워가며 살펴볼 것!

```python
def make_change(coin_value_list, coin_change, min_coins, coins_used):
    global counter
    for cents in range(0, coin_change+1):
        coin_count = cents
        new_coin = 1
        coins = list()
        for coin in coin_value_list:
            if coin <= cents:
                coins.append(coin)
        for coin in coins:
            counter = counter + 1
            if min_coins[cents - coin] + 1 < coin_count:
                coin_count = min_coins[cents - coin] + 1
                new_coin = coin
        min_coins[cents] = coin_count
        coins_used[cents] = new_coin
    return min_coins[coin_change]
```

```python
def print_coins(coins_used, coin_change):
    coin = coin_change
    while coin > 0:
        this_coin = coins_used[coin]
        print(this_coin)
        coin = coin - this_coin
```

잔돈 거슬러주기 DP: code

- 사용된 '동전' 리스트
- 동전 갯수 리스트
- 인수로 전달

```
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()
```

과제) print_coins() 제거

- 지난 시간에 했던 known_result 기법을 활용
- 재귀, DP 이해가 필수!

```
15  def main():
16      coin_value_list = [1, 5, 10, 21, 25]
17      coin_change = input('Change: ')
18      coin_change = int(coin_change)
19      known_result = [[]] * (coin_change+1)
20
21      print('Making change for', coin_change, 'requires')
22      print(make_change(coin_value_list, coin_change, known_result), 'coins')
23      print('They are:')
24      print(known_result[coin_change])
25      print('The used list is as follows:')
26      print(known_result)
```

```
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]]
```

테스트코드

- 잔돈거스르기 테스트코드
  - 입력 잔돈을 무작위로 섞어서 넣어줌
    (기존 전역변수를 통한 해결 불가)
  - 0~100원까지 테스트
- 재귀 테스트코드와 유사

```
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)
```

테스트코드 결과

- 0부터 혹은 100까지 차례대로 결과가 맞을 경우 True와 함께 결과 출력
- 마지막 Ran 3 test OK 가 뜨면 성공

```
True [21, 25, 25, 25] 96
True [25, 25, 5, 21, 21] 97
True [25, 21, 21, 21, 10] 98
True [10, 5, 21, 21, 21, 21] 99
True [25, 25, 25, 25] 100


Ran 3 tests in 0.039s

OK
```

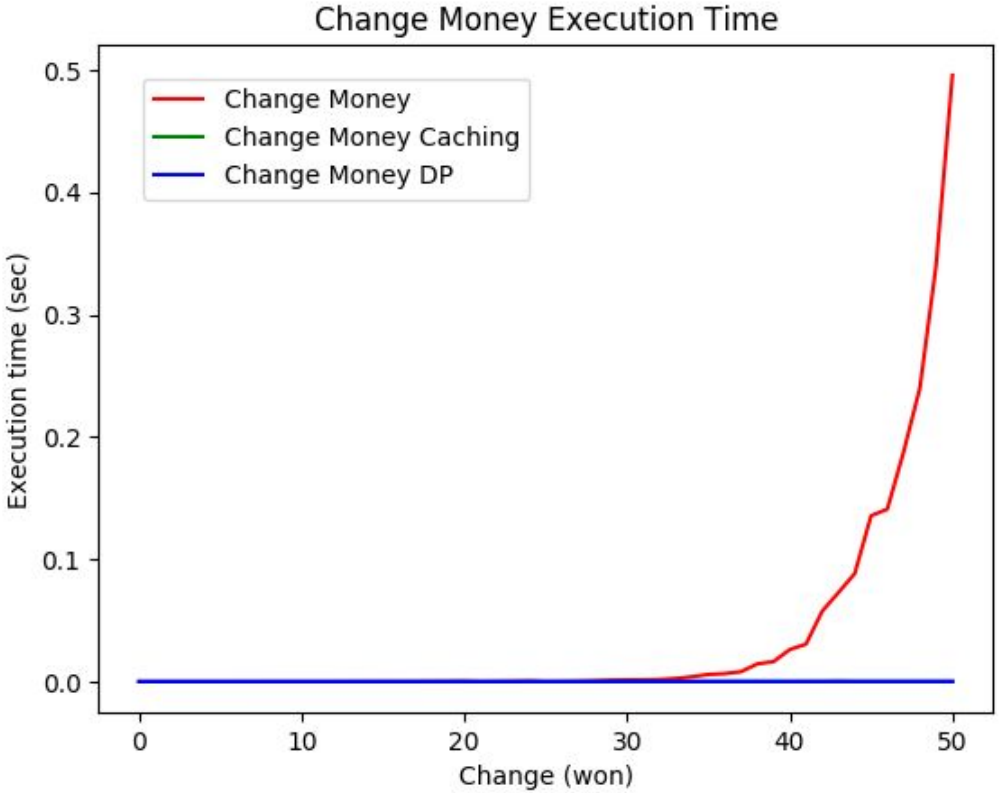
과제) known_result에 결과를 저장하는 DP

1. change_money_dp_list.py 구현
    - change_money_dp_list_test.py 로 테스트

성능 측정

지금까지 구현한 것을 비교

- 재귀가 진짜로 느릴까?
  - DP랑 Caching의 차이는?
- 얼마나 느릴까?
- 비교해보자



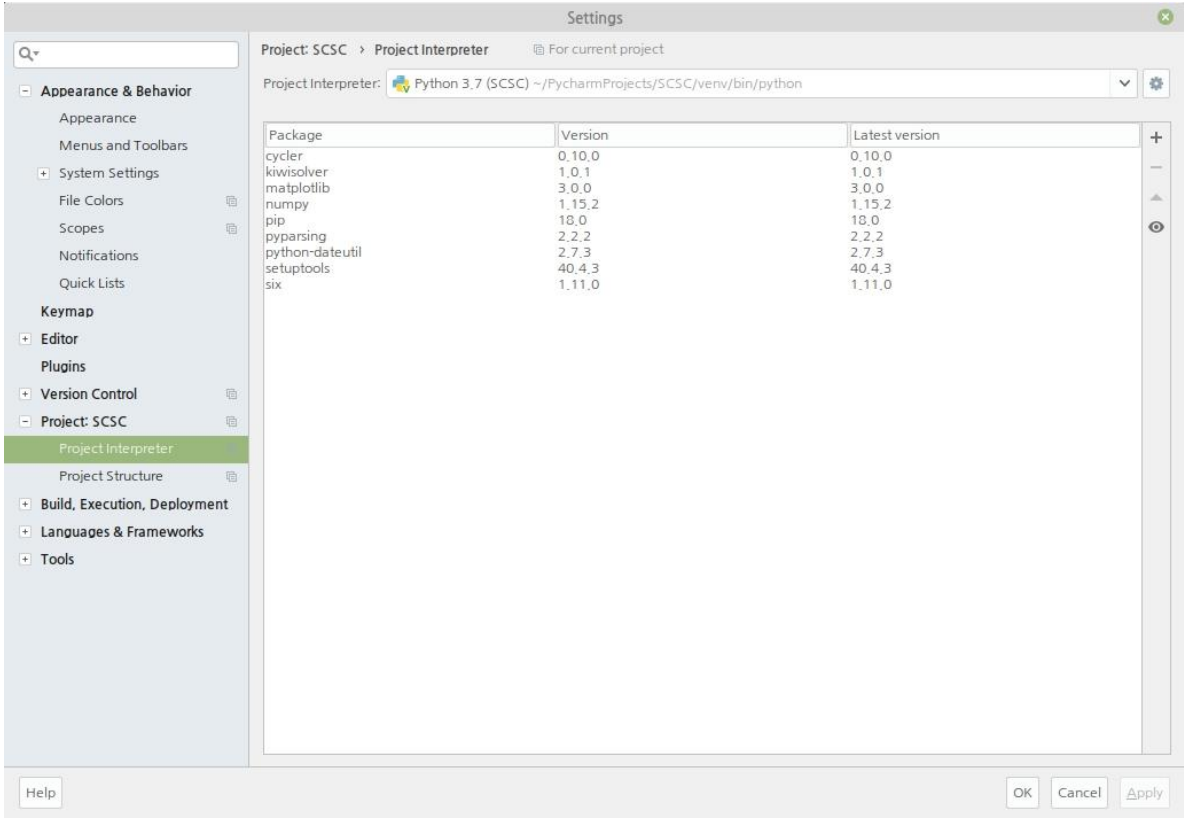
Caching vs DP

- python 3 에서는 Caching이 1000이상은 못 계산함


```
  File "/home/harny/PycharmProjects/SCSC/week4/change_money_saved_list.py", line 24, in make_change
    result = make_change(coin_value_list, coin_change - coin, known_result, counter)
  [Previous line repeated 993 more times]
  File "/home/harny/PycharmProjects/SCSC/week4/change_money_saved_list.py", line 12, in make_change
    if coin_change in coin_value_list:
RecursionError: maximum recursion depth exceeded in comparison
```

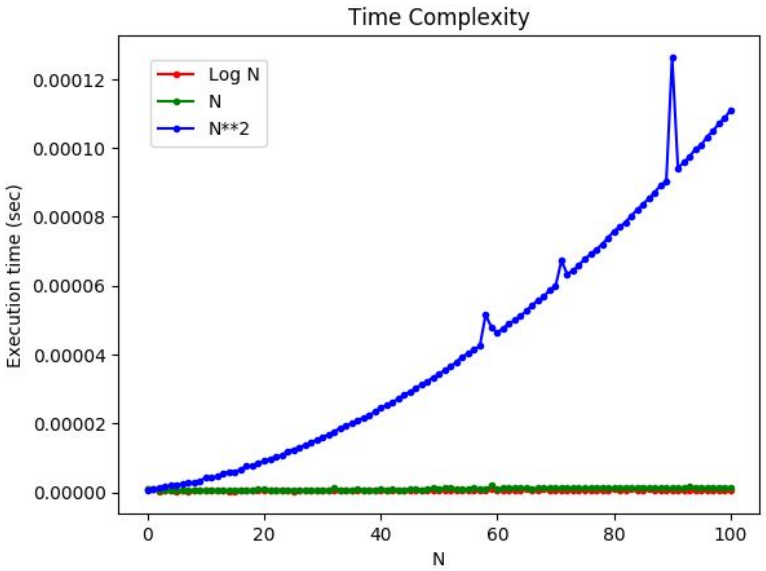
Install matplotlib

- File-Settings-Project-
  - Project Interpreter
- + 눌러서 matplotlib 검색
- 설치



그래프 예제

- O(log n)
- O(n)
- O(n**2)




```python
    # 차트 그리기
    plt.plot(x_data, y_data1, 'r.-', label='Log N')
    plt.plot(x_data, y_data2, 'g.-', label='N')
    plt.plot(x_data, y_data3, 'b.-', label='N**2')
    # 범례 추가
    plt.legend(bbox_to_anchor=(0.05, 0.95), loc=2, borderaxespad=0.0)
    plt.title('Time Complexity')  # 제목
    plt.ylabel('Execution time (sec)')  # Y축
    plt.xlabel('N')  # X축
    # 차트 저장 및 표시
    plt.savefig('time_complexity.png', bbox_inches='tight')  # 차트 저장
    # plt.show() # 차트 표시
    plt.close()  # 차트 닫기
```

과제) 재귀, 캐싱, DP 성능 비교

- 각 Y축 데이터를 계산할 때 호출하는 함수를 변경해주면 됨
- 3개의 함수 모두 import 할것
  - 참조 ex) import change_money_list
  - 호출 ex) change_money_list.change_money()

```
28
29      # Y1
30      start_time = time.time()
31      func_logn(x)
32      end_time = time.time()
33      execution_time = end_time - start_time
        y_data1.append(execution_time)
```

기타 유용한 정보

과제) 2+1개!

1. 손으로 DP 예제 써보기
2. 잔돈 거슬러주기 DP 완성
3. 성능 비교 그래프 그리기
4. 시간, 공간 복잡도 면에서 분석(선택)

실습 숙제 제출

- 숙제 제출 기한: 2018. 10. 10. 23:59:59
  - 실습 전 날
- 파일 제목: AL_학번_이름_05.zip
  - 파일 제목 다를 시 채점 안 합니다.
  - .egg 안 됨!

실습 숙제 제출할 것

- 2가지 파일을 제출
- AL_학번_이름_숙제번호.zip
  - 소스코드
    - Pycharm을 사용했을 경우 Project 디렉터리에 .idea, venv 같은 디렉터리는 제외
    - Jupyter + IPython을 사용했을 경우 ‘File - Download as’ 에서 .py 다운로드 가능
  - AL_학번_이름_숙제번호.pdf
    - 보고서는 무조건 .pdf
    - .hwp, .doc 등 채점 안 함

실습 보고서에 들어가야할 것

- 목표(할 일)
- 과제를 해결하는 방법
  - 알아야 할 것
- 과제를 해결한 방법
  - 주요 소스코드: 굳이 소스코드 전체를 붙일 필요는 없음
- 결과화면
  - 결과화면 설명(해석), 테스트코드 통과
- 보고서는 기본적으로 '내가 숙제를 했음'을 보이는 것
  - 지나치게 대충 작성하면 의심하게 됨

출석부 및 실습 점수가 궁금하다면?

- 출석부 및 실습 채점표
  - 수업 시작 후 30분까지 지각, 이후 결석
  - 실습 딜레이 1일당: -2점
    - 딜레이 2일까지: -2
    - 이후 -1씩 추가
- 튜터의 테스트 결과

질문이 생기면?

- 이름: 문현수
- 전공: 통신및보안
- 과정: 석박사통합과정 8학기
- 연구실: 데이터네트워크연구실(공5633)
- 메일: munhyunsu@cs-cnu.org
- 알고리즘은 함께 해결해가는 과목이므로 과감하게 연락
- 이메일로 처리가 안 되는 급한일: 문자/전화 등