2018년도 가을학기 SCSC 알고리즘 실습 3주차

문현수, munhyunsu@cs-cnu.org

Thursday September 20, 2018

1 개요

자료구조를 학습한 후 가장 많이 사용되는 데이터 타입은 'Stack'과 'Queue'라 하여도 과언이 아니다. 이 두가지 데이터 타입은 하드웨어부터 OS, 네트워크, 각종 응용 프로그램에서까지 활용되고 있다. 우리는 이번 실습에서 Queue를 활용한 응용 프로그램을 개발하며 Queue에 대한 지식을 되살리고 활용법을 익힌다. Queue는 손님이나 작업 대기열을 표현하기 매우 적합한 형식이다. 따라서 이것을 활용하는 실습과 과제를 수행한다.

우선 실습으로서 사무실에 설치된 'Printer'에 대한 시뮬레이터를 만들어본다. 무작위로 프린트 Job이 생성될 때 평균적으로 얼마나 기다리는지 알아내는 것을 목적으로 한다. 이를 위하여 2가지 클래스를 구현하고 시뮬레이터를 제작한다.

그 후 과제로서 푸드트럭 시뮬레이터를 제작해본다. 백OO의 푸드트럭이 예능 프로그램으로 인기를 얻으며 트럭에서 음식을 파는 푸드트럭이 각지에 생겨나고 있다. 이에 우리는 푸드트럭을 통하여 흑자를 낼 수 있는지 시뮬레이팅하는 프로그램을 제작한다. 푸드트럭 시뮬레이터를 통하여 대략적인 수익을 계산할 수 있다.

우리는 실습 2를 통하여 unittest를 이용한 코드 방법과 python 3 코드에 대하여 익숙해졌다. 실습 3에서는 제공되는 코드없이 오직 unittest code만을 읽고 그 설계, 목적을 이해하는 훈련을 한다. 이 과정을 통하여 우리는 코드 읽는 실력과 설계 능력을 기르게 된다. 또한 본격적인 알고리즘 학습에 앞서 python 3 코드에 더 익숙해질 것이다.

2 Printer simulator

Queue 활용 예제로 사무실에 설치된 'Printer simulator'를 구현한다. Printer class와 Task class를 구현하고 이를 활용하여 simulator를 작성한다. Printer는 한번에 하나의 출력만 할 수 있기 때문에 출력이 이루어지는 동안에 다른 Task는 대기하여야 한다. 수행 결과로 Task가 평균 얼마나 대기하였는지 출력하고, 지정된 시간동안 마치지 못 한 Task가 몇 개인지도 출력한다.

2.1 Task Class

Task class는 Printer가 수행해야할 업무 정보를 담고 있다. 출력해야할 Page 수와 Task가 생성된 시간을 가지고 있다. 모든 Page가 출력되면 Task가 생성된 시간과 현재 시간을 비교하여 수행에 걸린 시간을 계산할 수 있을 것이다.

Listing 1: Task Class - task.py

```
1  import random
2
3
4  class Task(object):
5      def __init__(self, timestamp):
6          self.timestamp = timestamp
7          self.pages = random.randint(1, 21)
8
9      def get_stamp(self):
10         return self.timestamp
11
12     def get_pages(self):
13         return self.pages
14
15     def wait_time(self, current_time):
16         return current_time - self.timestamp
```

Task class는 3개의 멤버 함수(메소드)를 가지고 있으며 2개의 멤버 변수(필드)를 생성한다. 멤버 변수 중 pages는 1–21사이 무작위 정수로 초기화된다. 멤버 함수 중 `wait_time()`을 통해 Task가 얼마나 오래 기다렸는지 계산할 수 있다.

2.2 Printer Class

Printer class는 주어진 Task를 수행한다. Task를 수행할 때 분당 처리할 수 있는 양에 따라서 시간이 필요하다. 이 기능을 위하여 시간을 보내는 방법과 새로운 업무를 시작하는 방법을 구현해야 한다.

Listing 2: Printer Class - printer.py

```
1  class Printer(object):
2      def __init__(self, page_per_minute):
3          self.page_rate = page_per_minute
4          self.current_task = None
5          self.time_remaining = 0
```

```
6
7       def tick( self ):
8           if  self . current_task  is  not None:
9               self . time_remaining = self.time_remaining - 1
10              if  self . time_remaining <= 0:
11                  self . current_task  = None
12
13      def busy(self ):
14          if  self . current_task  is  not None:
15              return True
16          else:
17              return False
18
19      def start_next( self ,  new_task):
20          self . current_task  = new_task
21          self . time_remaining = new_task.get_pages() * 60//self.page_rate
```

Printer class는 3개의 멤버 함수(메소드)와 3개의 멤버 변수(필드)를 가지고 있다. 이중 `tick()`은 수행하는 업무에 시간을 투자하는 것으로 이해할 수 있다. 예를 들어 수행해야할 업무가 100이라는 시간이 필요할 때 `start_next()`를 통해 Printer는 업무를 시작하게 된다. 이후 `tick()`을 통하여 100이라는 시간이 1씩 줄어들게 되며 0이 되면 업무가 끝나는 것으로 본다. Simulator는 이 2가지 클래스를 이용하여 구현할 수 있다.

2.3 Simulator Function

시뮬레이션 프로그램 제작을 위한 클래스가 모두 준비되었다. 이제 Simulator를 구현할 차례다. Simulator에서는 '언제' 새로운 Task가 생성될지 무작위로 계산하여야 한다. 새로운 Task를 생성한 후 Queue에 넣어 Printer에게 할당할 준비를 한다. Printer가 아무런 업무를 진행하고 있지 않을 때 Simulator는 Queue에서 Task를 하나 꺼내 업무를 진행하도록 지시한다.

Listing 3: Printer Simulator Functions - printer_simulator.py

```
1   import random
2
3   from printer import Printer
4   from task import Task
5
6
7   def simulation(num_seconds, page_per_minute):
8       lab_printer  = Printer(page_per_minute)
```

```
 9       print_queue = list()   # Queue
10       waiting_times = list()
11
12       for current_second in range(0, num_seconds):
13           if new_print_task():
14               task = Task(current_second)
15               print_queue.append(task)  # enqueue
16
17           if (not lab_printer.busy()) and (len(print_queue) != 0):
18               next_task = print_queue.pop(0)  # dequeue
19               waiting_times.append(next_task.wait_time(current_second))
20               lab_printer.start_next(next_task)
21
22           lab_printer.tick()
23
24       average_wait = sum(waiting_times) / len(waiting_times)
25       print('Average Wait {0:6.2f} secs {1:3d} tasks remaining.'.format(
             average_wait, len(print_queue)))
26
27
28   def new_print_task():
29       num = random.randint(1, 181)
30       if num == 180:
31           return True
32       else:
33           return False
34
35
36   def main():
37       for index in range(0, 10):
38           simulation(3600, 5)
39
40
41   if __name__ == '__main__':
42       main()
```

Printer simulator에서 새로운 Task를 생성하는 `new_print_task()`와 이를 Queue에 넣는 문장에 집중하여야 한다. 이처럼 Queue는 우리가 실생활에서 '대기열'로 부르는 것에 대입할 수 있다. First In First Out이라는 Queue를 설명하는 문장을 기억하자.

위의 Simulator까지 구현하여 실행하면 아래와 같은 결과가 나온다.

Listing 4: Printer Simulator Result

```
Average Wait  76.25 secs    0 tasks remaining.
Average Wait 118.12 secs    0 tasks remaining.
Average Wait  85.72 secs    5 tasks remaining.
Average Wait 450.14 secs    4 tasks remaining.
Average Wait  62.20 secs    1 tasks remaining.
Average Wait 306.71 secs    2 tasks remaining.
Average Wait 301.80 secs    0 tasks remaining.
Average Wait 112.69 secs    2 tasks remaining.
Average Wait  81.25 secs    0 tasks remaining.
Average Wait 220.00 secs    0 tasks remaining.
```

3 Food Truck Simulator

우리는 Printer 실습을 통하여 Queue 활용 방법을 익혔다. 이제 배운것을 더 확장하는 연습을 하자. 이번 과제를 통하여 익힐 것은 아래와 같다.

- unittest를 통하여 설계도를 이해
- Queue 활용 방법 숙지
- One producer - Multi consumer problem 문제 맛보기

위의 3가지 목표중에서 과제를 통해 얻기 바라는 가장 중요한 능력은 설계도를 이해하는 것이다. 우리는 실습 1에서부터 unittest를 기반으로 코드하고 있다. unittest를 읽는 것은 '다른 사람이 작성한 코드'를 읽는 것이기도 하며 프로그램 전체 구조(설계)를 이해하는 것이기도 하다. 현대 프로그램은 한 명의 프로그래머가 작성하지 않고 여러 프로그래머가 협업을 통하여 만들어내므로 다른 사람의 코드를 읽는 훈련이 필요하다. Queue 활용 방법도 실습과 과제를 통하여 익히자.

3.1 Order Class

Order class에는 2가지 멤버 변수가 있음을 알 수 있다(`__ini__`). 또한 3가지 멤버 함수가 있으며 각각 Getter의 역할을 하거나 시간을 계산하는 역할을 한다. Printer 예제에서의 Task와 유사하다.

Listing 5: Order Class Testcode - order_test.py

```
1 import unittest
2
```

```
3  from order import Order
4
5
6  class OrderTest(unittest.TestCase):
7      def setUp(self):
8          self.timestamp = 48
9          self.var_order = Order(self.timestamp)
10
11     def tearDown(self):
12         del self.timestamp
13         del self.var_order
14
15     def test___init__(self):
16         self.assertEqual(self.timestamp, self.var_order.order_time)
17         self.assertGreaterEqual(self.var_order.qty, 1)
18         self.assertLessEqual(self.var_order.qty, 5)
19
20     def test_get_order_time(self):
21         self.assertEqual(self.timestamp, self.var_order.get_order_time())
22
23     def test_get_qty(self):
24         self.assertGreaterEqual(self.var_order.get_qty(), 1)
25         self.assertLessEqual(self.var_order.get_qty(), 5)
26
27     def test_wait_time(self):
28         current_time = 60
29         self.assertEqual(12, self.var_order.wait_time(current_time))
30
31
32 if __name__ == '__main__':
33     unittest.main(verbosity=2)
```

3.2 Food Truck Class

Food Truck class에는 Order보다 훨씬 많은 멤버 변수와 함수가 있다. 하지만 Printer 예제에서의 Printer class와 매우 유사하므로 쉽게 완성할 수 있다.

Listing 6: Food Truck Class Testcode - food_truck_test.py

```
1  import unittest
2
```

```
3  from order import Order
4  from food_truck import FoodTruck
5
6
7  class FoodTruckTest(unittest.TestCase):
8      def test__init__(self):
9          price = 5000
10         making_time = 60
11         var_food_truck = FoodTruck(price, making_time)
12         self.assertEqual(price, var_food_truck.price)
13         self.assertEqual(making_time, var_food_truck.making_time)
14         self.assertEqual(None, var_food_truck.current_order)
15         self.assertEqual(0, var_food_truck.time_remaining)
16
17     def test_tick(self):
18         price = 5000
19         making_time = 60
20         var_truck = FoodTruck(price, making_time)
21         var_truck.current_order = Order(10)
22         var_truck.time_remaining = 10
23         for index in range(9, -1, -1):
24             var_truck.tick()
25             self.assertEqual(index, var_truck.time_remaining)
26         self.assertEqual(None, var_truck.current_order)
27
28     def test_is_busy(self):
29         price = 4000
30         making_time = 60
31         var_truck = FoodTruck(price, making_time)
32         var_truck.current_order = Order(10)
33         var_truck.time_remaining = 10
34         self.assertEqual(True, var_truck.is_busy())
35         var_truck.current_order = None
36         var_truck.time_remaining = 0
37         self.assertEqual(False, var_truck.is_busy())
38
39     def test_start_next_food(self):
40         price = 3000
41         making_time = 60
42         var_truck = FoodTruck(price, making_time)
43         next_order = Order(10)
```

```
44            var_truck.start_next_food(next_order)
45            self.assertEqual(next_order, var_truck.current_order)
46            self.assertEqual(making_time*next_order.get_qty(), var_truck.
                  time_remaining)
47
48
49  if __name__ == '__main__':
50      unittest.main(verbosity=2)
```

3.3 Food Zone Class

Food Zone class는 우리가 test code를 통해 이해해야하는 주요 목표다. 이것은 Food Truck 객체들을 멤버 변수로 가지고 있으며 새로운 손님이 왔는지 판별한다.

Listing 7: Food Zone Class Testcode - food_zone_test.py

```
1   import unittest
2
3   from food_truck import FoodTruck
4   from food_zone import FoodZone
5
6
7   class FoodZoneTest(unittest.TestCase):
8       def test__init__(self):
9           order_probability = 300
10          var_food_zone = FoodZone(order_probability)
11          self.assertEqual(set(), var_food_zone.food_trucks)
12          self.assertEqual(list(), var_food_zone.order_queue)
13          self.assertEqual(list(), var_food_zone.waiting_times)
14          self.assertEqual(list(), var_food_zone.income)
15          self.assertEqual(order_probability, var_food_zone.order_probability)
16
17      def test_add_truck(self):
18          order_probability = 300
19          var_food_zone = FoodZone(order_probability)
20          price = 3000
21          making_time = 60
22          var_truck = FoodTruck(price, making_time)
23          var_food_zone.add_truck(var_truck)
24          self.assertIn(var_truck, var_food_zone.food_trucks)
25
```

```
26      def test_is_new_order(self):
27          order_probability = 300
28          var_food_zone = FoodZone(order_probability)
29          self.assertEqual(True, var_food_zone.is_new_order(order_probability)
                )
30          self.assertEqual(False, var_food_zone.is_new_order(order_probability
                - 1))
31
32
33  if __name__ == '__main__':
34      unittest.main(verbosity=2)
```

3.4 Food Truck Simulator

3가지 class를 모두 구현한 후 메인 함수에서 simulator를 완성시켜보자. 이 중 `start_simulation()` 함수는 Queue를 활용하는 함수이므로 직접 구현해보자.

Listing 8: Food Truck Simulator - food_main.py

```
1   from order import Order
2   from food_truck import FoodTruck
3   from food_zone import FoodZone
4
5
6   def start_simulation(zone, test_second):
7       """Need Implement
8       """
9       pass
10
11
12  def print_food_zone_result(zone, open_seconds):
13      average_wait = sum(zone.waiting_times) / len(zone.waiting_times)
14      remain_order = len(zone.order_queue)
15      total_income = sum(zone.income)
16      setting_cost = len(zone.food_trucks) * 40000000
17      recipe_cost = int(sum(zone.income)*0.3)
18      staff_cost = (open_seconds//(60*60)) * 8350 * len(zone.food_trucks) * 1
19      profit = total_income - setting_cost - recipe_cost - staff_cost
20      for truck in zone.food_trucks:
21          print('- Food_price: {0:,d}, making_time: {1:d}'.format(truck.price
                , truck.making_time))
```

```
22      print('Average wait: {0:10.2f} secs'.format(average_wait))
23      print('Order remain: {0:10d}'.format(remain_order))
24      print('Total income: {0:10,d} Won'.format(total_income))
25      print('Setting cost : {0:10,d} Won'.format(setting_cost))
26      print('Recipe cost: {0:10,d} Won'.format(recipe_cost))
27      print('Staff cost : {0:10,d} Won'.format(staff_cost))
28      print('Profit : {0:10,d} Won'.format(profit))
29
30
31  def main():
32      order_probability = 300
33      open_seconds = 60*60*6*20*12
34      food_list = [(2000, 120)]
35      food_zone1 = FoodZone(order_probability)
36
37      for price, making_time in food_list:
38          food_zone1.add_truck(FoodTruck(price, making_time))
39
40      start_simulation(food_zone1, open_seconds)
41      print('----# Food zone 결과(영업 시간 : {0:.1f}, 손님 확률 :
            대략 {1} 초당 1 명 ) #----'.format(
42              open_seconds/(60*60), order_probability))
43      print_food_zone_result(food_zone1, open_seconds)
44
45
46  if __name__ == '__main__':
47      main()
```

과제를 완료한 후 실행시키면 다음과 같은 출력이 나온다. 2,000원짜리 음식을 팔 때 개당 120초가 걸리면 300초마다 손님이 1명씩 온다고 가정했을 때 지나치게 대기 시간이 길어지게 된다(공급이 수요를 못 따라간다.). 이를 어떻게 수정해야하는지 실험해보자. 시뮬레이터는 미리 실험해보고 우리가 어떻게 행동해야하는지 알 수 있도록 한다.

Listing 9: Food Truck Simulator Result

```
----# Food zone 결과(영업 시간 : 1440.0, 손님 확률 : 대략 300
    초당 1 명 ) #----
- Food price: 2,000, making time: 120
Average wait:  444837.50 secs
Order remain:        2767
Total income: 86,382,000 Won
```

```
Setting cost: 40,000,000 Won
Recipe cost: 25,914,600 Won
Staff cost: 12,024,000 Won
Profit: 8,443,400 Won
```

예를 들어 2000원짜리 음식을 만드는 시간을 120초에서 60초로 단축시키면 손님들이 거의 기다리지 않는 것을 알 수 있다. Queue를 활용해 시뮬레이터를 구현하여 사용하면 이와 같은 것을 예측할 수 있다.

Listing 10: Food Truck Simulator Better Result

```
----# Food zone 결과(영업 시간 : 1440.0, 손님 확률 : 대략 300
    초당 1명) #----
- Food price: 2,000, making time: 60
Average wait: 164.91 secs
Order remain: 0
Total income: 103,168,000 Won
Setting cost: 40,000,000 Won
Recipe cost: 30,950,400 Won
Staff cost: 12,024,000 Won
Profit: 20,193,600 Won
```

4 과제

실습에서 우리는 Printer simulator를 구현하였고 과제로 Food truck simulator를 구현하였다. 이번 과제는 Food truck simulator를 완성하는 것이다.

4.1 과제 목표

- order.py 구현
- food_truck.py 구현
- food_zone.py 구현
- fodd_main.py 내부 start_simulation() 구현
- 문제 해결법 찾기(선택): 6개월 만에 흑자로 돌아서는 방법은? 등등

4.2 제출 관련

- 마감 날짜: 2018. 09. 26. 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_03.zip(파일명 준수!)

4.3 조교 연락처

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