REPORT



**알고리즘**

**Algorithm**

Assignment 4

황호영 교수님

컴퓨터정보공학부

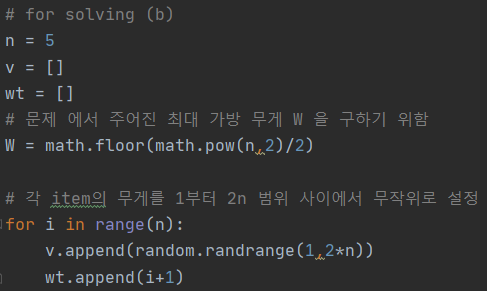
2017202037 오민혁

2020/12/14 (월)

이번 과제는 0-1 knapsack problem을 solving 하기 위한 프로그램을 bottom-up dynamic-programming algorithm 으로 작성하고 이 프로그램을 이용해 optimal solution을 찾고 item 들의 개수를 다양하게 설정하여 그에 따른 optimal solution을 구하기 위한 algorithm의 실행 시간을 측정하고 알아보는 것이다.

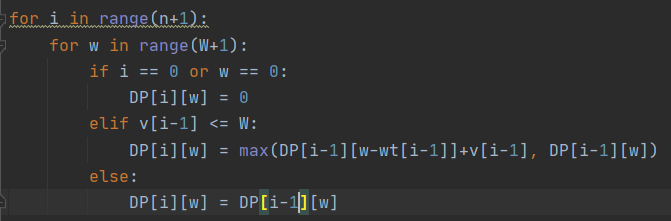
1. **1. 문제 (b)를 해결하기 위한 코드 설명**

문제 (b)는 item의 개수가 5개이고, i = 1, 2, 3, 4, 5 일 때 각 item의 가치인 를 1부터 2n (즉, 여기서 n은 5임으로 2n은 10이다.) 까지 난수로 설정했을 때, 그리고 각 item의 무게는 i 와 같다. ( i가 1일 때 무게 즉, ,) 이 내용을 코드로 표현하면 아래와 같다.



Optimal solution은 bottom-up dynamic-programming algorithm으로 구하는데, 이를 위해서는 점화식을 구해야 한다.

점화식은 다음과 같다.



점화식을 코드로 표현하면 이렇다. 우선 i나 w가 0일 때에는 무조건 2차원 리스트의 해당 값에는 0이 들어가야 한다.

란 i개의 보석이 들어있는 배낭의 무게 한도가 W 일 때, optimal solution 즉 최고의 이익 가치이다.

이 식을 설명하면, 만약 번 째 item의 무게 가 최대 무게인 보다 크면 i 번째 아이템을 추가하지 않은 경우인, 전 단계 즉, i – 1 번째 보석들이 들어있는 무게 한도가 W일 때의 값을 선택하고, 번 째 item의 무게 가 최대 무게인 보다 크지 않으면 위에서 설명한 전 단계 값과 i 번째 item을 추가했을 때의 이익 값 중 더 큰 것을 선택하게 된다.

**2. 문제 (d)를 해결하기 위한 코드 설명**

우선 (b)를 풀기 위해, 설정 해 놓았던 n=5 를 바꾸어야 했다. (d)에서는 다양한 n이 필요하기 때문이다. 넓은 범위와 충분한 개수의 n을 테스트 해야 하기 때문에 n 값을 저장할 list를 선언하였다.



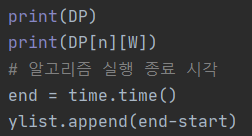
또한 문제 (b)를 해결하기 위한 코드는 n을 5로 설정하였을 때, 한 번 수행하는 코드이기 때문에 문제 (b)를 해결하기 위한 코드를 nlist의 요소 개수 만큼 수행하기 위해 반복문을 추가하였다.



코드 수행 시간의 시작 지점은 Dynamic programming 을 위한 2차원 리스트 생성부터 설정하였다.



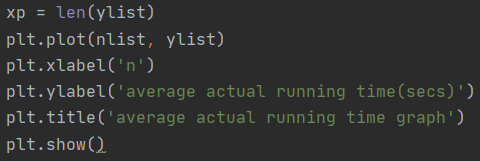
그 다음 optimal solution을 출력하는 부분까지를 코드 수행 시간의 마지막으로 설정하였다.

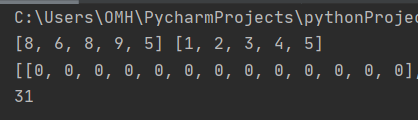


코드 수행 시간의 끝 지점을 구한 다음, 코드 수행 마지막 시각에서 시작 시각을 뺀 값을 ylist에 저장해 주었다. 이는 n이 변함에 따라 달라지는 코드 수행 시간을 기록하기 위해서이다.

이렇게 만든 list들을 통해 pyplot 모듈을 이용하여 그래프를 그릴 수 있었다.

코드는 다음과 같다.



1. 5개의 item을 가지고 bottom-up dynamic-programming algorithm을 통해 0-1 knapsack problem에 대한 optimal solution을 구한 결과는 다음과 같다.

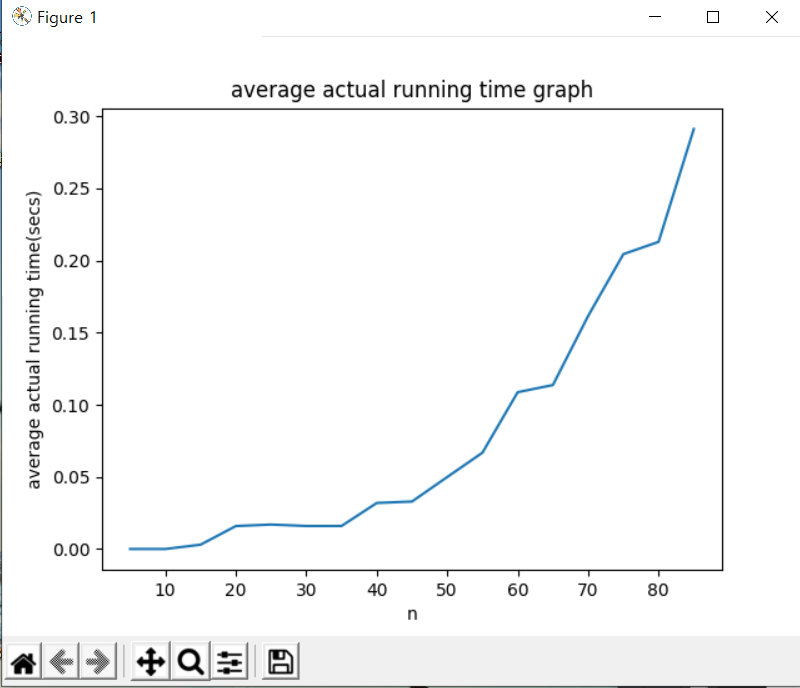
이는 1부터 2n 범위에서 무작위로 5개의 item에 대한 각각의 value를 설정해주었는데, i가 1, 2, 3, 4, 5 일 때, 이고, 가방의 최대 무게가 일 때, optimal solution은 31이다.

1. 2차원 리스트를 표로 표현하여 thief가 고른 item들을 탐색해보았다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i w | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| 2 | 0 | 14 | 8 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 3 | 0 | 22 | 22 | 14 | 22 | 16 | 22 | 22 | 22 | 22 | 22 | 22 | 22 |
| 4 | 0 | 31 | 31 | 31 | 22 | 31 | 31 | 23 | 31 | 25 | 31 | 31 | 31 |
| 5 | 0 | 31 | 36 | 36 | 36 | 31 | 36 | 36 | 36 | 27 | 36 | 36 | 31 |

표를 통해 확인해 본 결과, 1, 2, 3, 4번째 item 총 4개의 item 을 훔쳤음을 알 수 있다.

1. nlist 를 [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85] 로 설정하였을 때 실제 코드 실행시간 그래프이다.



대체로 n이 증가함에 따라 코드 수행시간도 증가하고 있음을 알 수 있다.

하지만 특정 n의 지점에서 그래프가 급격하게 상승 하는 것이 특징으로 관찰된다.

**<Computing environment>**

|  |  |
| --- | --- |
| CPU | Intel® Core™ i7-8750H CPU @ 2.20GHz 2.20 GHz |
| RAM | 16.0GB |
| OS | Windows 10 HOME 64bit |

**[참고문헌]**

1. <https://blog.naver.com/PostView.nhn?blogId=hahaha_oh&logNo=221997321049&parentCategoryNo=&categoryNo=17&viewDate=&isShowPopularPosts=true&from=search> 파이썬 소수점 올림 내림
2. <https://wikidocs.net/79> 파이썬 난수 생성
3. <https://gsmesie692.tistory.com/113> Dynamic Programming: 배낭 채우기 문제 (Knapsack Problem)