# 알고리즘

| 과제번호 | 11주차 |
| --- | --- |
| 날 짜 | 2018.11.22 |
| 학 번 | 201302395 |
| 이 름 | 류경빈 |

| 과제 1 | Dynamic Programming 0-1 Knapsack problem 해결 프로그램 |
|---|---|

```java
public int OPT(int index, int weight) {
    if (index == 0) {
        return 0;
    } else if (items.get(index).getWeight() > weight){ // bagSize의 값 보다 index의 weight가 더 클 경우
        return OPT( index: index - 1, weight);
    } else {
        return Math.max(OPT( index: index - 1, weight), items.get(index).getValue()
                + OPT( index: index - 1,  weight: weight - items.get(index).getWeight()));
    }
}
```

- bag 배열의 값들과 비교과정을 재귀를 진행하면서 optimize하는 함수이다.

W + 1 →

n + 1 ↓

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ϕ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| {1} | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| {1, 2} | 0 | 1 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| {1, 2, 3} | 0 | 1 | 6 | 7 | 7 | 18 | 19 | 24 | 25 | 25 | 25 | 25 |
| {1, 2, 3, 4} | 0 | 1 | 6 | 7 | 7 | 18 | 22 | 24 | 28 | 29 | 29 | 40 |
| {1, 2, 3, 4, 5} | 0 | 1 | 6 | 7 | 7 | 18 | 22 | 28 | 29 | 34 | 34 | 40 |

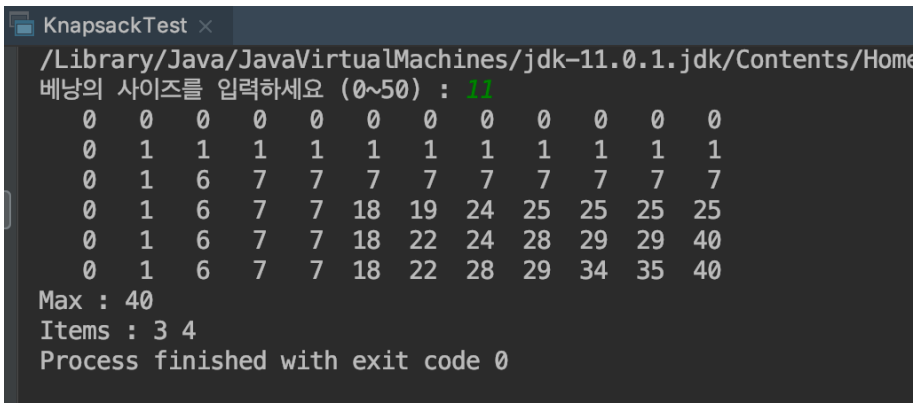
| Item | Value | Weight |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 6 | 2 |
| 3 | 18 | 5 |
| 4 | 22 | 6 |
| 5 | 28 | 7 |

- index의 값이 0 보다 작은 경우는 모두 0의 값을 가진다 = Item이 아무것 도 없는 공집합일 경우
- Items의 입력받은 index 무게 weight와 입력 받은 weight와 비교한다. 비교하는 이유는 해당 row의 가장 큰 index를 가지는 Item의 weight가 가방의 크기에 들어갈 수 있는 순간까지의 비교이다. 그래서 그 전까지는 기존의 index-1의 값을 가진다.
- 위의 조건을 모두 만족하지 않을 때 기존 index-1번째에 저장되어 있는 값과 비교를 진행해 큰 값일 때의 값을 배열에 저장한다.

```java
public void printMaxValueAndPickedItems() {
    ArrayList<Integer> pickedItems = new ArrayList<>();
    System.out.println("Max : " + bag[itemSize-1][bagSize]);
    System.out.print("Items : ");
    int i = itemSize - 1;
    int j = bagSize;
    while (i > 0) {
        if (j < items.get(i).getWeight()) {
            i--;
            continue;
        } else if (bag[i][j] - bag[i - 1][j - items.get(i).getWeight()] == items.get(i).getValue()) {
            pickedItems.add(i);
            j = j - items.get(i).getWeight();
            i--;
            if (j == 0) {
                break;
            }
        } else {
            i--;
        }
    }
    for (i = pickedItems.size() - 1; i >= 0; i--) {
        System.out.print(pickedItems.get(i) + " ");
    }
}
```

- 가치 총합이 높은 것은 맨 마지막 배열에 저장 된 값이며, 이를 구성하는 것을 출력해준다.

## 결과 화면


KnapsackTest
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home
베낭의 사이즈를 입력하세요 (0~50) : 11
0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1
0 1 6 7 7 7 7 7 7 7 7 7
0 1 6 7 7 18 19 24 25 25 25 25
0 1 6 7 7 18 22 24 28 29 29 40
0 1 6 7 7 18 22 28 29 34 35 40
Max : 40
Items : 3 4
Process finished with exit code 0