**Problem Set #3 (Algorithms)**

**Department** : 컴퓨터정보공학부

**Student ID**: 2018202074

**Student Name**: 김상우

1. **Write your program that includes your comments**

2018202074\_김상우.c 파일내에 주석 처리를 하였다. main함수에선 함수 배열로 입력을, knapSack01 함수에서는 입력을 바탕으로 bottom-up Dynamic Programming방식으로 3가지 경우(더 이상 무게 혹은 아이템 수가 0이라 추가할 수 없는 경우, 한계 무게를 초과하여 더 이상 아이템을 추가할 수 없는 경우, 아이템을 더 추가할 수 있는 경우)에 대해 주석을 달아두었다.

1. **When W = 5, v1 =12, v2 = 20, v3 = 24, v4 = 30, and wi = I for 1,2,3,4 show the value (in dollars) of an optimal solution to the 0-1 knapsack problem for 4 items by executing your program.**

텍스트이(가) 표시된 사진

자동 생성된 설명

문제의 조건을 맞춰 다음과 같이 input을 넣어주었다.

실행 결과 위와 같이 44라는 결과가 출력되는 모습이다 .

위의 input의 경우 조건에 맞는 최대 value는 두번째와 3번재 아이템을 가져가는 경우로 20+24인 44dollar라는 결과를 정확히 출력해내는 모습이다.

1. **Explain your program and your execution.**

텍스트이(가) 표시된 사진

자동 생성된 설명

우선 변수 w와 i에 각각 도둑이 들 수 있는 무게 한계와 아이템의 수를 입력받을 수 있도록 하였다. 이후 val 배열과 wt 배열을 선언하여 각 item의 dollar와 무게를 사용자가 넣을 수 있게 해주었다. 해당 값들을 knapSack01함수에 넣고 return하는 값을 출력하고 main function을 return 0로 종료해주는 모습이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

정답을 구하기 위한 함수는 위와 같다. 만약 item의 수가 0 혹은 들 수 있는 무게가 0인 경우에는 운반 혹은 추가가 불가능하기에 0(dollar)을 return한다.

만약 남은 아이템 개수나 남은 운반 가능 무게가 0보다 크지만, 즉 더 아이템을 추가할 수 있는 다른 조건들은 맞지만 운반 가능한 무게를 이미 넘어버린 경우는 해당 경우에서 추가해본 item을 뺀 경우에 대해 해당 함수를 실행한 결과를 return해준다.

만약 남은 아이템 개수나 남은 운반 가능 무게가 0보다 크고, 운반 가능한 무게를 아직 넘지 않은 경우에는 아이템을 추가한 경우와 추가하지 않은 경우를 비교해서 더 많은 dollar가치가 있는 경우를 return한다.

위 함수는 입력 내에서 가장 높은 dollar를 갖는 경우를 return하며 이를 위해 함수에 해당하는 아이템을 추가했을 때의 가장 높은 dollar를 갖는 경우를 이전까지의 값과 비교하여 얻는 bottom-up dynamic programming 방식으로 execute됨을 코드에서 확인할 수 있다. 가장 아래에 있는 bottom level에서는 i값이 0이 들어가 0을 return하므로 바로 위 level에서는 추가된 value값이 그대로 return된다는 특징 또한 가지고 있다.

위 코드는 i와 w 값을 각각 한 축으로 하는 (i\*W)2차원배열에 값을 채워 최댓값을 찾는 것과 같으며 i=n이므로 O(nW) time인 조건 또한 만족한다.

Reference

Lec6\_part2.pdf