10주차 2차시 근사알고리즘 I

【학습목표】

1. 배낭문제를 이해할 수 있다.
2. 여행자문제를 이해할 수 있다.

학습내용1 : 배낭문제

* 근사알고리즘
- NP-완전 문제들은 실 세계의 광범위한 영역에 활용되지만, 이 문제들을 다항식 시간에 해결할 수 있는 알고리즘이 아직 발견되지 않았다.
- 또한 아직까지 그 누구도 이 문제들을 다항식 시간에 해결할 수 없다고 증명하지 못했다.
- 대부분의 학자들은 이 문제들을 해결할 다항식 시간 알고리즘이 존재하지 않을 것이라고 추측하고 있다.

- 이러한 NP-완전 문제들을 어떤 방식으로든지 해결하려면 다음의 3가지 중에서 1가지는 포기해야 한다.
    1) 다항식 시간에 해를 찾는 것
    2) 모든 입력에 대해 해를 찾는 것
    3) 최적해를 찾는것

- NP-완전 문제를 해결하기 위해 3 번째 것을 포기한다. 즉, 최적해에 아주근사한(가까운) 해를 찾아주는 근사 알고리즘(Approximation algorithm)을 살펴본다.

- 근사 알고리즘은 근사해를 찾는 대신에 다항식시간의 복잡도를 가진다.
- 근사 알고리즘은 근사해가 얼마나 최적해에 근사한지(즉, 최적해에 얼마나가까운지)를 나타내는 근사비율(Approximation Ratio)을 알고리즘과 함께 제시하여야 한다.
- 근사 비율은 근사해의 값과 최적해의 값의 비율로서, 1.0에 가까울수록 정확도가 높은 알고리즘이다.
- 근사 비율을 계산하려면 최적해를 알아야 하는 모순이 생긴다.
- 따라서 최적해를 대신할 수 있는 '간접적인' 최적해를 찾고, 이를 최적해로 삼아서 근사 비율을 계산한다.

1. 배낭 (Knapsack) 문제

배낭 (Knapsack) 문제는 n개의 물건과 각 물건 i의 무게 w_i와 가치 v_i가 주어지고, 배낭의 용량은 C일 때, 배낭에 담을 수 있는 물건의 최대 가치를 찾는 문제이다. 단, 배낭에 담은 물건의 무게의 합이 C를 초과하지 말아야 하고, 각 물건은 1개씩만 있다.

- 먼저 배낭 문제의 부분 문제를 찾아내기 위해 문제의 주어진 조건을 살펴보면 물건, 물건의 무게, 물건의 가치, 배낭의 용량, 모두 4가지의 요소가 있다.
- 이중에서 물건과 물건의 무게는 부분 문제를 정의하는데 반드시 필요하다.
- 왜냐하면 배낭이 비어 있는 상태에서 시작하여 물건을 하나씩 배낭에 담는 것과 안 담는 것을 현재 배낭에 들어 있는 물건의 가치의 합에 근거하여 결정해야 하기 때문이다.
- 또한 물건을 배낭에 담으려고 할 경우에 배낭 용량의 초과 여부를 검사해야 한다.
- 따라서 배낭 문제의 부분문제를 아래와 같이 정의할 수 있다.
- K[i,w] = 물건 1~i까지만 고려하고, (임시) 배낭의 용량이 w일 때의 최대 가치
  단, i = 1, 2, …, n이고, w = 1, 2, 3, …, C이다.
- 그러므로 문제의 최적해는 K[n,C]이다.
- 여기서 주의하여 볼 것은 배낭의 용량이 C이지만, 배낭의 용량을 1부터 C까지 1씩 증가시킨다는 것이다.
- 이 때문에 C의 값이 매우 크면, 알고리즘의 수행시간 너무 길어지게 된다.
  - 따라서 다음의 알고리즘은 제한적인 입력에 대해서만 효용성을 가진다.

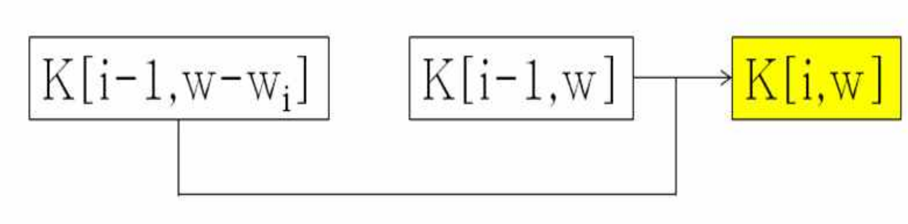
* 알고리즘

```
Knapsack
입력: 배낭의 용량 C, n개의 물건과 각 물건 i의 무게 wi와 가치 vi, 단, i = 1, 2, …, n
출력: K[n,C]
1. for i = 0 to n    K[i,0]=0       // 배낭의 용량이 0일 때
2. for w = 0 to C K[0,w]=0 // 물건 0이란 어떤 물건도 고려하지 않을 때
3. for i = 1 to n {
4.      for w = 1 to C { // w는 배낭의 (임시) 용량
5.                if ( wi > w )   // 물건 i의 무게가 임시 배낭 용량을 초과하면
6.                        K[i,w] = K[i-1,w]
7.                else // 물건 i를 배낭에 담지 않을 경우와 담을 경우 고려
8.                        K[i,w] = max{K[i-1,w], K[i-1,w-wi]+vi}
          }
     }
9. return K[n,C]
```

- Line 1에서는 2차원 배열 K의 0번 열을 0으로 초기화시킨다. 그 의미는 배낭의 (임시)용량이 0일 때, 물건 1~n까지 각각 배낭에 담아보려고 해도 배낭에 담을 수 없으므로 그에 대한 각각의 가치는 0일 수밖에 없다는 뜻이다.
- Line 2에서는 0번 행의 각 원소를 0으로 초기화시킨다. 여기서 물건 0이란 어떤 물건도 배낭에 담으려고 고려하지 않는다는 뜻이다. 따라서 배낭의 용량을 0에서 C까지 각각 증가시켜도 담을 물건이 없으므로 각각 의 최대 가치는 0이다.
- Line 3~8에서는 물건을 1에서 n까지 하나씩 고려하여 배낭의 (임시) 용량을 1에서 C까지 각각 증가시키며, 다음을 수행한다.

- Line 5~6에서는 현재 배낭에 담아보려고 고려하는 물건 i의 무게 wi가 (임시) 배낭 용량 w보다 크면 물건 i를 배낭에 담을 수 없으므로, 물건 i까지 고려했을 때의 최대 가치 K[i,w]는 물건 (i-1)까지 고려했을 때의 최대 가치 K[i-1,w]가 된다.
- Line 7~8에서는 만일 현재 고려하는 물건 i의 무게 wi가 현재 배낭의 용량 w보다 같거나 작으면, 물건 i를 배낭에 담을 수 있다. 그러나 현재 상태에서 물건 i를 추가로 배낭에 담으면 배낭의 무게가 (w+wi)로 늘어난다. 따라서 현재의 배낭 용량인 w를 초과하게 되어, 물건 i를 추가로 담을 수는 없다.

- 그러므로 앞의 그림에서와 같이, 물건 i를 배낭에 담기 위해서는 2가지 경우를 살펴보아야 한다.
- 물건 i를 배낭에 담지 않는 경우, K[i,w] = K[i-1,w]가 된다.
- 물건 i를 배낭에 담는 경우, 현재 무게인 w에서 물건 i의 무게인 wi를 뺀 상태에서 물건을 (i-1)까지 고려했을 때의 최대 가치인 K[i-1,w-wi]와 물건 i의 가치 vi의 합이 K[i,w]가 되는 것이다.
- Line 8에서는 이 2가지 경우 중에서 큰 값이 K[i,w]가 된다.
- 배낭 문제의 부분 문제간의 함축적 순서는 다음과 같다. 즉, 2개의 부분 문제 K[i-1,w-wi]과 K[i-1,w]가 미리 계산되어 있어야만 K[i,w]를 계산할 수 있다.



2. Knapsack 알고리즘 수행과정

배낭의 용량 C=10kg이고, 각 물건의 무게와 가치는 다음과 같다.

물건	1	2	3	4
무게 (kg)	5	4	6	3
가치 (만원)	10	40	30	50

- Line 1~2에서는 아래와 같이 배열의 0번 행과 0번 열의 각 원소를 0으로 초기화한다.

C=10

배낭 용량 → w=			0	1	2	3	4	5	6	7	8	9	10
무게	가치	물건	0	0	0	0	0	0	0	0	0	0	0
5	10	1	0										
4	40	2	0										
6	30	3	0										
3	50	4	0										

- Line 3에서는 물건을 하나씩 고려하기 위해서, 물건 번호 i가 1~4까지 변하며, line 4에서는 배낭의 (임시) 용량 w가 1kg씩 증가되어 마지막엔 배낭의 용량인 10kg이 된다.
- i=1일 때 (즉, 물건 1만을 고려한다.)
- w=1 (배낭의 용량이 1kg)일 때, 물건 1을 배낭에 담아보려고 한다. 그러나 w1>w 이므로, (즉, 물건 1의 무게가 5kg이므로, 배낭에 담을 수 없기 때문에) K[1,1] = K[i-1,w] = K[1-1,1] = K[0,1] = 0이다. 즉, K[1,1]=0이다.
- 담을 수 없다. 따라서 각각 K[1,2]=0, K[1,3]=0, K[1,4]=0 이다. 즉, 배낭의 용량을 4kg까지 늘려 봐도 5kg의 물건 1을 배낭에 담을 수 없다.

- w=5 (배낭의 용량이 5kg)일 때, 물건 1을 배낭에 담을 수 있다. 왜냐하면 w1=w이므로, 즉, 물건 1의 무게가 5kg이기 때문이다.

따라서 K[1,5] = max{K[i-1,w], K[i-1,w-wi]+vi}
= max{K[1-1,5], K[1-1,5-5]+10}
= max{K[0,5], K[0,0]+10}
= max{0, 0+10}
= max{0, 10} = 10이다.

- w=6, 7, 8, 9, 10일 때, 각각의 경우가 w=5일 때와 마찬가지로 물건 1을 담을 수 있다. 따라서 각각 K[1,6] = K[1,7] = K[1,8] = K[1,9] = K[1,10] = 10이다.

- 다음은 물건 1에 대해서만 배낭의 용량을 1~C까지 늘려가며 알고리즘을 수행한 결과이다.

C=10

배낭 용량 → w=			0	1	2	3	4	5	6	7	8	9	10
무게	가치	물건	0	0	0	0	0	0	0	0	0	0	0
5	10	i =1	0	**0**	**0**	**0**	**0**	**10**	**10**	**10**	**10**	**10**	**10**
4	40	2	0										
6	30	3	0										
3	50	4	0										

- i=2일 때 (즉, 물건 1에 대한 부분 문제들의 해는 i=1일 때 위에서 이미 구하였고, 이를 이용하여 물건 2를 고려한다.)
- w=1, 2, 3 (배낭의 용량이 각각 1, 2, 3kg)일 때, 물건 2를 배낭에 담아보려고 한다. 그러나 w2>w이므로, 즉, 물건 2의 무게가 4kg이므로, 배낭에 담을 수 없다. 따라서 K[2,1]=0, K[2,2]=0, K[2,3]=0이다.

- w=4 (배낭의 용량이 4kg)일 때, 물건 2를 배낭에 담을 수 있다.

K[2,4] = max{K[i-1,w], K[i-1,w-wi]+vi}
= max{K[2-1,4], K[2-1,4-4]+40}
= max{K[1,4], K[1,0]+40}
= max{0, 0+40}
= max{0, 40} = 40이다.

- w=5 (배낭의 용량이 5kg)일 때, 물건 2의 무게가 4kg이므로, 역시 배낭에 담을 수 있다. 그러나 이 경우에는 물건 1이 배낭에 담았을 때의 가치와 물건 2를 담았을 때의 가치를 비교하여, 더 큰 가치를 얻는 물건을 배낭에 담는다.

K[2,5] = max{K[i-1,w], K[i-1,w-wi]+vi}
= max{K[2-1,5], K[2-1,5-4]+40}
= max{K[1,5], K[1,1]+40}
= max{10, 0+40}
= max{10, 40} = 40이다.

즉, 물건 1을 배낭에서 빼낸 후, 물건 2를 담는다.
그 때의 가치가 40이다.

- w=6, 7, 8일 때, 각각의 경우도 물건 1을 빼내고 물건 2를 배낭에 담는 것이 더 큰 가치를 얻는다. 따라서 각각 K[2,6] = K[2,7] = K[2,8] = 40이 된다.

- w=9 (배낭의 용량이 9kg)일 때, 물건 2를 배낭에 담아보려고 한다. 그런데 w2<w 이므로, 물건 2를 배낭에 담을 수 있다. 따라서

K[2,9] = max{K[i-1,w], K[i-1,w-wi]+vi}
= max{K[2-1,9], K[2-1,9-4]+40}
= max{K[1,9], K[1,5]+40}
= max{10, 10+40}
= max{10, 50} = 50이다.

- 즉, 이때에는 배낭에 물건 1, 2 둘 다를 담을 수 있는 것이고, 그때의 가치가 50이 된다는 의미이다.

- i=3과 i=4일 때 알고리즘이 수행을 마친 결과이다.

C

배낭 용량 → w=			0	1	2	3	4	5	6	7	8	9	10
물건	가치	무게	0	0	0	0	0	0	0	0	0	0	0
1	10	5	0	0	0	0	0	10	10	10	10	10	10
2	40	4	0	0	0	0	40	40	40	40	40	50	50
3	30	6	0	0	0	0	40	40	40	40	40	50	70
4	50	3	0	0	0	50	50	50	50	90	90	90	90

- 마지막으로 최적해는 K[4,10]이고, 그 가치는 물건 2와 4의 가치의 합인 90이다.

3. 시간복잡도

- 하나의 부분 문제에 대한 해를 구할 때의 시간복잡도는 line 5에서의 무게를 한 번 비교한 후 line 6에서는 1개의 부분문제의 해를 참조하고, line 8에서는 2개의 해를 참조한 계산하므로 O(1) 시간이 걸린다.
- 그런데 부분 문제의 수는 배열 K의 원소 수인 nxC개이다. 여기서 C는 배낭의 용량이다.
- 따라서 Knapsack 알고리즘의 시간복잡도는 O(1)xnxC = O(nC)이다.

4. 응용

배낭 문제는 다양한 분야에서 의사 결정 과정에 활용된다. 예를 들어, 원자재의 버리는 부분을 최소화시키기 위한 자르기/분할, 금융 포트폴리오와 자산 투자의 선택, 암호 생성 시스템 (Merkle-Hellman Knapsack Cryptosystem) 등에 활용된다.

* 다른 배낭 문제의 예
다음과 같은 조건의 배낭문제에 대해 욕심쟁이 방법으로 해를 구하려고 한다. 두 번째로 배낭에 넣은 물건의 이익은 얼마인가?(단, 물건을 쪼갤 수 있다고 가정, M: 배낭의 용량, n: 물건의 개수, 각 물건(Xi)의 무게 wi와 그 물건의 이익 pi)

M=10, n=4 $(p_1, p_2, p_3, p_4)=(18, 15, 12, 25)$, $(w_1, w_2, w_3, w_4)=(3, 5, 3, 4)$

욕심쟁이 방법으로 해를 구할 때 배낭의 용량 M=10 임으로
첫 번째 X4 인 w4 의 4를 선택 물건의 이익 p4 25가 선택 된다.
두 번째 X1 인 w1 의 3을 선택 물건의 이익 p1 18이 선택 된다.
세 번째 X3 인 w3 의 3을 선택 물건의 이익 p3 12가 선택 된다.

학습내용2 : 여행자문제

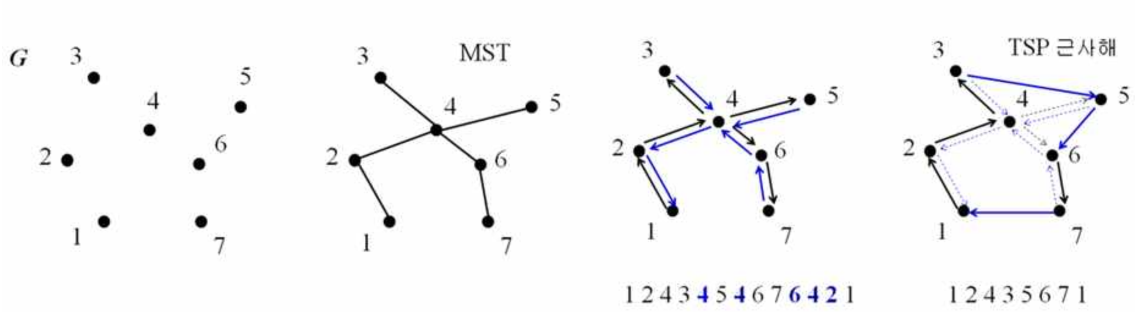
1. 여행자 문제 (Traveling Salesman Problem, TSP)

- 여행자 문제 (Traveling Salesman Problem, TSP)는 여행자가 임의의 한 도시에서 출발하여 다른 모든 도시를 1번씩만 방문하고 다시 출발했던 도시로 돌아오는 여행 경로의 길이를 최소화하는 문제이다.

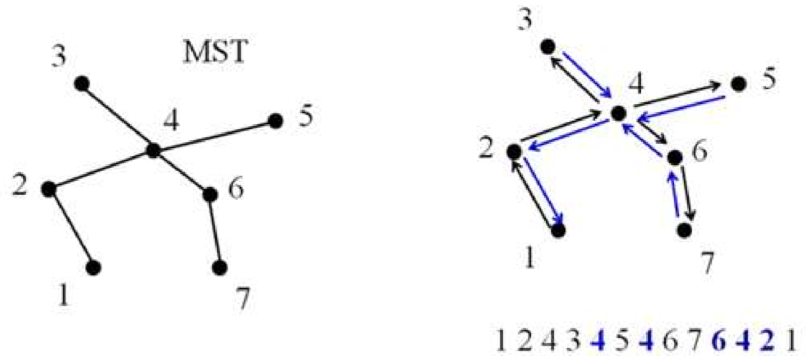
- 여행자 문제는 주어지는 문제의 조건에 따라서 여러 종류가 있다. 여기서 다루는 여행자 문제의 조건은 다음과 같다.
1) 도시 A에서 도시 B로 가는 거리는 도시 B에서 도시 A로 가는 거리와 같다. (대칭성)
2) 도시 A에서 도시 B로 가는 거리는 도시 A에서 다른 도시 C를 경유하여 도시 B로 가는 거리보다 짧다.
(삼각 부등식 특성)

- TSP를 위한 근사 알고리즘을 고안하려면, 먼저 다항식 시간 알고리즘을 가지면서 유사한 특성을 가진 문제를 찾아야 한다.
- TSP와 비슷한 특성을 가진 문제는 최소 신장 트리 (Minimum Spanning Tree, MST) 문제이다.
- MST 는 모든 점을 사이클 없이 연결하는 트리 중에서 트리 선분의 가중치 합이 최소인 트리이다.
- MST 의 모든 점을 연결하는 특성과 최소 가중치의 특성을 TSP에 응용하여, 시작 도시를 제외한 다른 모든 도시를 트리 선분을 따라 1번씩 방문하도록 경로를 찾는 것이다.

- MST를 활용한 근사해 찾는 과정: MST를 활용하여 여행자 문제의 근사해를 찾기 위해 삼각 부등식 원리를 적용한다.



- 먼저 MST를 찾고, 임의의 도시 (그림에서는 도시 1)에서 출발하여 트리의 선분을 따라서 모든 도시를 방문하고 돌아오는 도시의 방문 순서를 구한다.
- [1 2 4 3 4 5 4 6 7 6 4 2 1].



- 마지막으로 이 순서를 따라서 도시를 방문하되 중복 방문하는 도시를 순서에서 제거한다. 단, 도시 순서의 가장 마지막에 있는 출발 도시 1은 중복되어 나타나지만 제거하지 않는다.

- 중복하여 방문하는 도시를 제거하는 과정에 삼각형 부등식 원리가 사용된 것이다

* Approx_MST_TSP
입력: n개의 도시, 각 도시간의 거리
출력: 출발 도시에서 각 도시를 1번씩만 방문하고 출발 도시로 돌아오는 도시 순서

1. 입력에 대하여 MST를 찾는다.
2. MST에서 임의의 도시로부터 출발하여 트리의 선분을 따라서 모든 도시를 방문하고 다시 출발했던 도시로 돌아오는 도시 방문 순서를 찾는다.
3. return 이전 단계에서 찾은 도시 순서에서 중복되어 나타나는 도시를 제거한 도시 순서 (단, 도시 순서의 가장 마지막의 출발 도시는 제거하지 않는다.)

- Line 1에서는 4.2절의 크러스컬 (Kruskal)이나 프림 (Prim) 알고리즘을 사용하여 MST를 찾는다.
- Line 2에서는 하나의 도시에서 어떤 트리 선분을 선택하여 다른 도시를 방문할 때 지켜야 할 순서가 정해져 있지 않으므로, 임의의 순서로 방문해도 괜찮다.
- Line 3에서는 삼각 부등식의 원리를 적용함과 동시에 각 도시를 1번씩만 방문하기 위해서 중복 방문된 도시를 제거한다. 단, 가장 마지막 도시인 출발 도시는 중복되지만 TSP 문제의 출발 도시로 돌아와야 한다는 조건에 따라 제거하지 않는다.

2. 시간복잡도

- Line 1: MST를 찾는 데에는 크러스컬이나 프림 알고리즘의 시간복잡도만큼 시간이 걸리고,
- Line 2에서 트리 선분을 따라서 도시 방문 순서를 찾는 데는 O(n) 시간이 걸린다. 왜냐하면 트리의 선분 수가 (n-1)이기 때문이다.
- Line 3: line 2에서 찾은 도시 방문 순서를 따라가며, 단순히 중복된 도시를 제거하므로 O(n) 시간이 걸린다.
- 시간복잡도는 (크러스컬이나 프림 알고리즘의 시간복잡도)+O(n)+O(n)이므로 크러스컬이나 프림 알고리즘의 시간복잡도와 같다.

3. 근사 비율

- 여행자 문제의 최적해를 실질적으로 알 수 없으므로, '간접적인' 최적해인 MST 선분의 가중치의 합(M)을 최적해의 값으로 활용한다.
- 왜냐하면 실제의 최적해의 값이 M보다 항상 크기 때문이다.
- 그런데 Approx_MST_TSP 알고리즘이 계산한 근사해의 값은 2M보다는 크지 않다. 왜냐하면
 - Line 2에서 MST의 선분을 따라서 도시 방문 순서를 찾을 때 사용된 트리 선분을 살펴보면, 각 선분이 2번 사용되었다. 따라서 이 도시 방문 순서에 따른 경로의 총 길이는 2M이다.
 - Line 3에서는 삼각 부등식의 원리를 이용하여 새로운 도시 방문 순서를 만들기 때문에, 이전 도시 방문 순서에 따른 경로의 길이보다 새로운 도시 방문 순서에 따른 경로의 길이가 더 짧다.
- 따라서 이 알고리즘의 근사비율은 2M/M=2보다 크지 않다. 즉, 근사해의 값이 최적해의 값의 2배를 넘지 않는다.

【학습정리】

1.
 - NP-완전 문제들은 실 세계의 광범위한 영역에 활용되지만, 이 문제들을 다항식 시간에 해결할 수 있는 알고리즘이 아직 발견되지 않았다.
- 또한 아직까지 그 누구도 이 문제들을 다항식 시간에 해결할 수 없다고 증명하지 못했다.
- 대부분의 학자들은 이 문제들을 해결할 다항식 시간 알고리즘이 존재하지 않을 것이라고 추측하고 있다.
- 이러한 NP-완전 문제들을 어떤 방식으로든지 해결하려면 다음의 3가지 중에서 1가지는 포기해야 한다.
    1) 다항식 시간에 해를 찾는 것
    2) 모든 입력에 대해 해를 찾는 것
    3) 최적해를 찾는것
- NP-완전 문제를 해결하기 위해 3 번째 것을 포기한다. 즉, 최적해에 아주근사한(가까운) 해를 찾아주는 근사 알고리즘(Approximation algorithm)을 살펴본다.
- 근사 알고리즘은 근사해를 찾는 대신에 다항식시간의 복잡도를 가진다.
- 근사 알고리즘은 근사해가 얼마나 최적해에 근사한지(즉, 최적해에 얼마나가까운지)를 나타내는 근사비율(Approximation Ratio)을 알고리즘과 함께 제시하여야 한다.
- 근사 비율은 근사해의 값과 최적해의 값의 비율로서, 1.0에 가까울수록 정확도가 높은 알고리즘이다.
- 근사 비율을 계산하려면 최적해를 알아야 하는 모순이 생긴다.
- 따라서 최적해를 대신할 수 있는 '간접적인' 최적해를 찾고, 이를 최적해로 삼아서 근사 비율을 계산한다.
 0/1배낭 (Knapsack) 문제, 궤 채우기 문제, TSP (외판원 방문 판매 문제) 등

2. 배낭문제
- 배낭 (Knapsack) 문제는 n개의 물건과 각 물건 i의 무게 w_i와 가치 v_i가 주어지고, 배낭의 용량은 C일 때, 배낭에 담을 수 있는 물건의 최대 가치를 찾는 문제이다. 단, 배낭에 담은 물건의 무게의 합이 C를 초과하지 말아야 하고, 각 물건은 1개씩만 있다.
- 응 용 : 배낭 문제는 다양한 분야에서 의사 결정 과정에 활용된다. 예를 들어, 원자재의 버리는 부분을 최소화시키기 위한 자르기/분할, 금융 포트폴리오와 자산 투자의 선택, 암호 생성 시스템 (Merkle-Hellman Knapsack Cryptosystem) 등에 활용된다.

3. 여행자 문제
- 여행자 문제 (Traveling Salesman Problem, TSP)는 여행자가 임의의 한 도시에서 출발하여 다른 모든 도시를 1번씩만 방문하고 다시 출발했던 도시로 돌아오는 여행 경로의 길이를 최소화하는 문제이다.