제5장 동적 계획 알고리즘 (3/3)

과 목 명 정 보 처 리 알 고 리 즘
담당교수 김 성 훈
경북대학교 과학기술대학 소프트웨어학과

이 장에서 배울 내용

1. 동적 계획 알고리즘의 기본개념
2. 모든 쌍 최단 경로(All Pairs Shortest Path)
3. 연속 행렬 곱셈(Chained Matrix Multiplication)
4. 동전 거스름돈 문제(Coin Change)
5. 편집 거리 문제(Edit Distance Problem)
6. 배낭문제(Knapsack Problem)

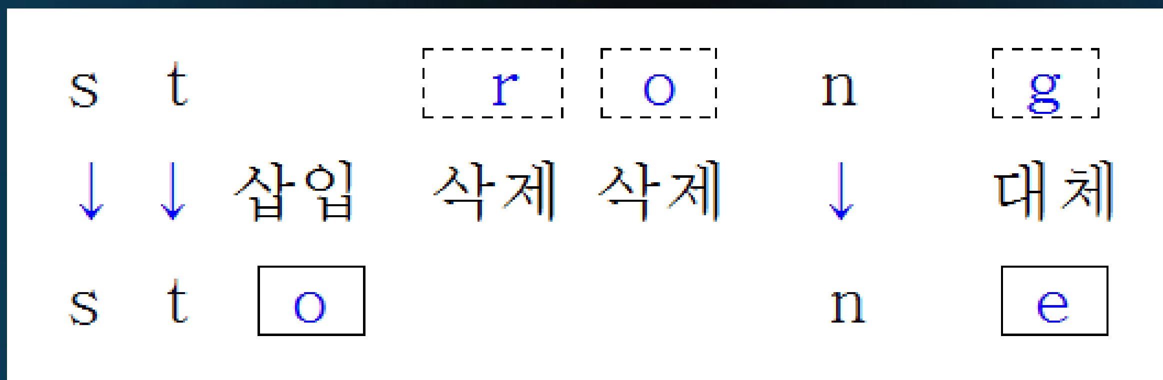
5.3 편집 거리 문제

- 문서 편집기를 사용하는 중에 하나의 스트링(문자열) S를 수정하여 다른 스트링 T로 변환시키고자 할 때,
  삽입(insert), 삭제(delete), 대체(substitute)연산이 사용된다.

- S를 T로 변환시키는데 필요한 최소의 편집 연산 횟수를 편집 거리 (Edit Distance)라고 한다.

편집 거리 문제의 이해

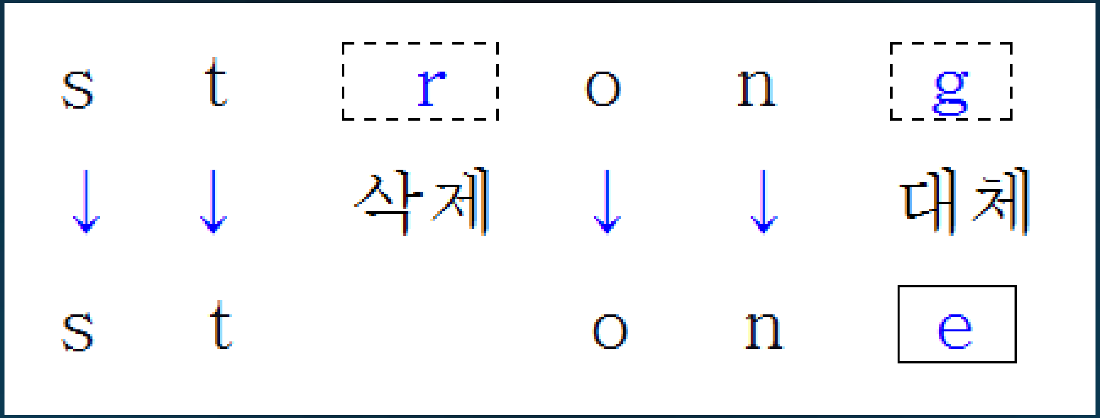
- 예를 들어, ‘strong’을 ‘stone’으로 편집하여 보자.



- 위의 편집에서는 ‘s’와 ‘t’는 그대로 사용하고, ‘o’를 삽입하고, ‘r’과 ‘o’를 삭제한다.
- 그 다음엔 ‘n’을 그대로 사용하고, 마지막으로 ‘g’를 ‘e’로 ‘대체’시키어, 총 4회의 편집 연산이 수행되었다.

편집 거리 문제의 이해 (2)

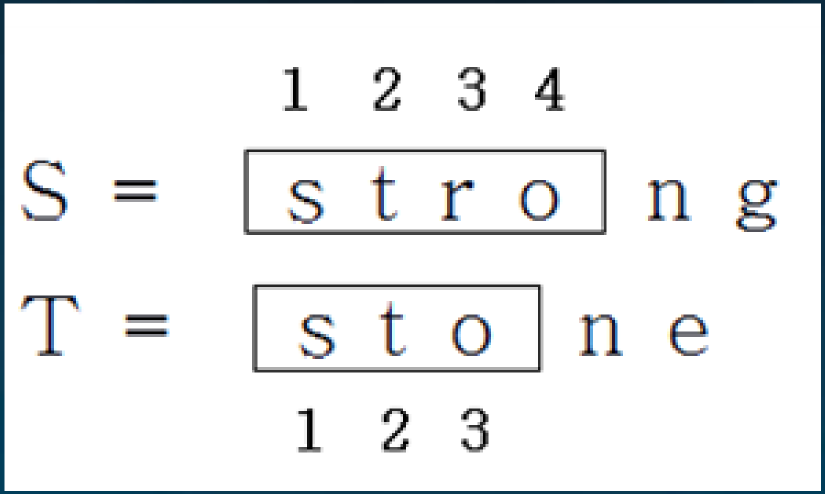
- 반면에 아래의 편집에서는 ‘s’와 ‘t’는 그대로 사용한 후, ‘r’을 삭제하고, ‘o’와 ‘n’을 그대로 사용한 후, ‘g’를 ‘e’로 대체시키어, 총 2회의 편집 연산만이 수행되었고, 이는 최소 편집 횟수이다.



- 이처럼 S를 T로 바꾸는데 어떤 연산을 어느 문자에 수행하는가에 따라서 편집 연산 횟수가 달라진다.

부분문제의 정의

- 편집 거리 문제를 동적 계획 알고리즘으로 해결하려면 부분 문제들을 어떻게 표현해야 할까?
  - ‘strong’을 ‘stone’으로 편집하려는데, 만일 각 접두부 (prefix)에 대해서, 예를 들어, ‘stro’를 ‘sto’로 편집할 때의 편집 거리를 미리 알고 있으면, 각 스트링의 나머지 부분에 대해서, 즉, ‘ng’를 ‘ne’로의 편집에 대해서 편집 거리를 찾음으로써, 주어진 입력에 대한 편집 거리를 구할 수 있다.



부분문제의 정의 (2)

- 부분문제를 정의하기 위해서 스트링 S와 T의 길이가 각각 m과 n이라 하고, S와 T의 각 문자를 다음과 같이 s_i와 t_j라고 하자. 단, i = 1, 2, ⋯, m 이고, j = 1, 2, ⋯, n이다.

$$S = s_1\ s_2\ s_3\ s_4 \cdots s_m$$

$$T = t_1\ t_2\ t_3\ t_4 \cdots t_n$$

- 부분문제의 정의: E[i,j]는
  S의 접두부의 i개 문자를 T의 접두부 j개 문자로 변환시키는데 필요한 최소 편집 연산 횟수, 즉, 편집 거리이다.

부분문제 예시

- 예를 들어, 'strong'을 'stone'에 대해서, 'stro'를 'sto'로 바꾸기 위한 편집 거리를 찾는 문제는 E[4,3]이 되고,
점진적으로 E[6,5]를 해결하면 문제의 해를 찾게 된다.

- 다음 예제에 대해 처음 몇 개의 부분 문제의 편집 거리를 계산하여 보자.

	1	2	3	4	5	6
S	s	t	r	o	n	g
T	s	t	o	n	e	

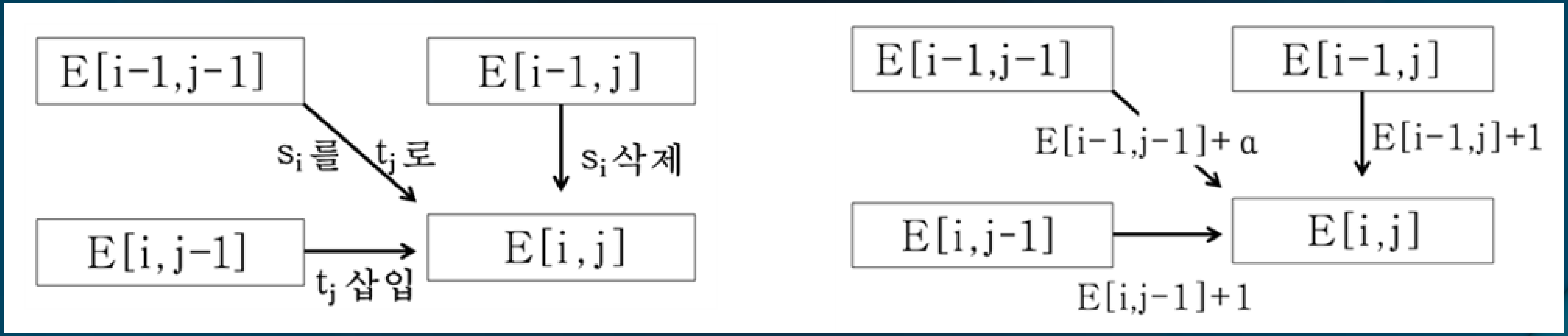
부분문제 예시 (2)

E	T	ε	s	t	o
S	i \ j	0	1	2	3
ε	0	0	1	2	3
s	1	1	0	1	2
t	2	2	1	0	1
r	3	3	2	1	1
o	4	4	3	2	1

부분문제의 해 계산식

- 따라서 E[4,3]의 편집 거리는 위의 3가지 부분 문제들의 해, 즉, E[4,2], E[3,3], E[3,2]의 편집 거리를 알면 된다. 그런데 E[4,2]=2, E[3,3]=1, E[3,2]=1이므로, (2+1), (1+1), 1 중에서 최소값인 1이 E[4,3]의 편집 거리가 된다.

- 일반적으로 E[i-1,j], E[i,j-1], E[i-1,j-1]의 해가 미리 계산되어 있으면 E[i,j]를 계산할 수 있다. 그러므로 편집 거리 문제의 부분 문제간의 함축적인 순서는 다음과 같다.



해의 계산식과 배열 초기화

- 따라서 위의 3가지 경우 중에서 가장 작은 값을 E[i,j]의 해로서 선택한다. 즉,

$$E[i,j] = \min\{E[i,j-1]+1,\ E[i-1,j]+1,\ E[i-1,j-1]+\alpha\}$$

단, $\alpha=1$ if $s_i \neq t_j$, else $\alpha=0$

- 위의 식을 위해서 E[0,0], E[1,0], E[2,0], …, E[m,0]과 E[0,1], E[0,2], …, E[0,n]이 아래와 같이 초기화한다.

	T	ε	t_1	t_2	t_3	..	t_n
S		0	1	2	3	..	n
ε	0	0	1	2	3	..	n
s_1	1	1					
s_2	2	2					
s_3	3	3					
.	.	.					
.	.	.					
s_m	m	m					

EditDistance 알고리즘

EditDistance(S, T)
입력: 스트링 S, T, 단, S와 T의 길이는 각각 m과 n이다.
출력: S를 T로 변환하는 편집 거리, E[m,n]

1. for i=0 to m E[i,0]=i   // 0번 열의 초기화
2. for j=0 to n E[0,j]=j   // 0번 행의 초기화
3. for i=1 to m
4.   for j=1 to n
5.   E[i,j] = min{E[i,j-1]+1, E[i-1,j]+1, E[i-1,j-1]+α}
6. return E[m,n]

알고리즘 수행 과정

- 다음은 EditDistance 알고리즘이 ‘strong'을 ’stone'으로 바꾸는데 필요한 편집 거리를 계산한 결과인 배열 E이다.

j

i

E	T	ε	s	t	o	n	e
S		0	1	2	3	4	5
ε	0	0	1	2	3	4	5
s	1	1	**0**	1	2	3	4
t	2	2	1	**0**	1	2	3
r	3	3	2	**1**	1	2	3
o	4	4	3	2	**1**	2	3
n	5	5	4	3	2	**1**	2
g	6	6	5	4	3	2	**2**

알고리즘 수행 과정 (2~8)

- 배열에서 파란색 음영으로 표시된 원소가 계산되는 과정을 각각 상세히 살펴보자.

T		ε	s	t	o	n	e
S		0	1	2	3	4	5
ε	0	0	1	2	3	4	5
s	1	1					
t	2	2					
r	3	3					
o	4	4					
n	5	5					
g	6	6					

알고리즘 수행 과정 (9)

- E[5,4] = min{E[5,3]+1, E[4,4]+1, E[4,3]+α} = min{(2+1), (1+1), (1+0)} = 1. 즉, 현재 T의 처음 3문자 'sto'가 만들어져 있는 상태에서 S의 5번째 문자와 T의 4번째 문자가 같으므로 아무런 연산이 필요 없이 'ston'이 만들어지는 것이다.

	T	ε	s	t	o	n
S	i \ j	0	1	2	3	4
ε	0	0	1	2	3	4
s	1	1	0	1	2	3
t	2	2	1	0	1	2
r	3	3	2	1	1	2
o	4	4	3	2	1	2
n	5	5	4	3	2	1

알고리즘 수행 과정 (10)

- $E[6,5] = \min\{E[6,4]+1, E[5,5]+1, \underline{E[5,4]+\alpha}\} = \min\{(2+1), (2+1), \underline{(1+1)}\} = 2.$

	T	ε	s	t	o	n	e
S	i \ j	0	1	2	3	4	5
ε	0	0	1	2	3	4	5
s	1	1	0	1	2	3	4
t	2	2	1	0	1	2	3
r	3	3	2	1	1	2	3
o	4	4	3	2	1	2	3
n	5	5	4	3	2	1	2
g	6	6	5	4	3	2	2

시간복잡도

- EditDistance 알고리즘의 시간복잡도는 O(mn)이다.
  단, m과 n은 두 스트링의 각각의 길이이다.

- 그 이유는 총 부분 문제의 수가 배열 E의 원소 수인 mxn이고,
  각 부분 문제 (원소)를 계산하기 위해서 주위의 3개의 부분 문제들의 해 (원소)를 참조한 후 최소값을 찾는 것이므로 O(1) 시간이 걸리기 때문이다.

응 용

- 2개의 스트링 사이의 편집 거리가 작으면, 이 스트링들이 서로 유사하다고 판단할 수 있으므로, 생물 정보 공학 (Bioinformatics) 및 의학 분야에서 두 개의 유전자가 얼마나 유사한가를 측정하는데 활용된다.
  - 예를 들어, 환자의 유전자 속에서 암 유전자와 유사한 유전자를 찾아내어 환자의 암을 미리 진단하는 연구와 암세포에만 있는 특징을 분석하여 항암제를 개발하는 연구에 활용되며, 좋은 형질을 가진 유전자들 탐색 등의 연구에도 활용된다.
- 그 외에도 철자 오류 검색 (Spell Checker), 광학 문자 인식 (Optical Character Recognition)에서의 보정 시스템 (Correction System), 자연어 번역 (Natural Language Translation) 소프트웨어 등에도 활용된다.

5.4 배낭 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.

문제 분석

- 배낭 문제는 제한적인 입력에 대해서 동적 계획 알고리즘으로 해결할 수 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.

부분문제 정의

- 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.

Knapsack 알고리즘

Knapsack(C, n, W, V)
입력: 배낭의 용량 C, n개의 물건과 각 물건 i의 무게 w_i와 가치 v_i, 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 ( w_i > 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-w_i]+v_i}
        }
   }
9. return K[n,C]
```

알고리즘 설명

- 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.

알고리즘 설명 (2)

- Line 5~6에서는 현재 배낭에 담아보려고 고려하는 물건 i의 무게 w_i가 (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 w_i가 현재 배낭의 용량 w보다 같거나 작으면, 물건 i를 배낭에 담을 수 있다. 그러나 현재 상태에서 물건 i를 추가로 배낭에 담으면 배낭의 무게가 $(w+w_i)$로 늘어난다. 따라서 현재의 배낭 용량인 w를 초과하게 되어, 물건 i를 추가로 담을 수는 없다.

알고리즘 설명 (3)

물건 1 ~ (i-1)까지 고려하여 현재
배낭의 용량이 w인 경우의 최대 가치

배낭에서 물건 i를
담을 공간을 마련

K[i-1,w] 0

물건 i K[i,w]

K[i-1,w-w_i] + 물건 i의 가치 v_i 1

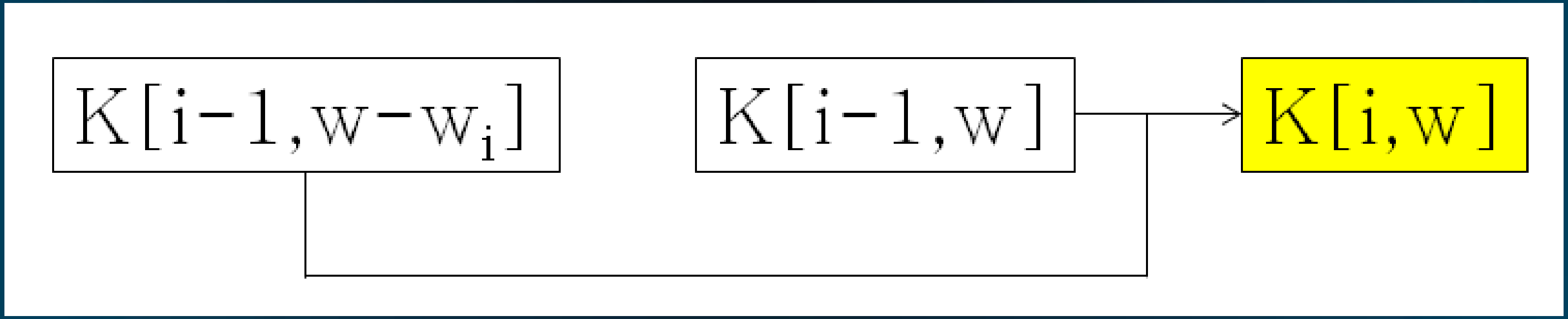
물건 1 ~ (i-1)까지 고려하여 현재 배낭의
용량이 (w-w_i)인 경우의 최대 가치

알고리즘 설명 (4)

- 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 w_i를 뺀 상태에서 물건을 (i-1)까지 고려했을 때의 최대 가치인 K[i-1,w-w_i]와 물건 i의 가치 v_i의 합이 K[i,w]가 되는 것이다.
- Line 8에서는 이 2가지 경우 중에서 큰 값이 K[i,w]가 된다.

알고리즘 설명 (5)

- 배낭 문제의 부분 문제간의 함축적 순서는 다음과 같다. 즉, 2개의 부분 문제 K[i-1,w-w_i]과 K[i-1,w]가 미리 계산되어 있어야만 K[i,w]를 계산할 수 있다.



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

수행과정 (2)

- 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										

수행과정 (3)

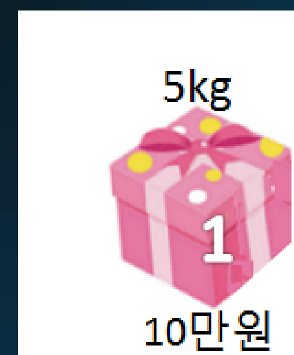
- Line 3에서는 물건을 하나씩 고려하기 위해서, 물건 번호 i가 1~4까지 변하며, line 4에서는 배낭의 (임시) 용량 w가 1kg씩 증가되어 마지막엔 배낭의 용량인 10kg이 된다.

- i=1일 때 (즉, 물건 1만을 고려한다.)

- w=1 (배낭의 용량이 1kg)일 때, 물건 1을 배낭에 담아보려고 한다. 그러나 w_1>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.

수행과정 (4)

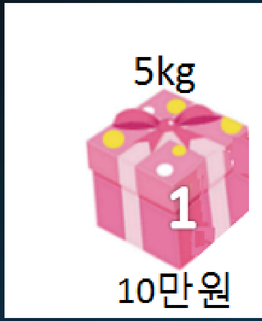
- w=2, 3, 4일 때, 각각 $w_1 > w$ 이므로, 물건 1을 담을 수 없다. 따라서 각각 K[2,1]=0, K[1,3]=0, K[1,4]=0 이다. 즉, 배낭의 용량을 4kg까지 늘려봐도 5kg의 물건 1을 배낭에 담을 수 없다.



수행과정 (5)

- $w=5$ (배낭의 용량이 5kg)일 때, 물건 1을 배낭에 담을 수 있다. 왜냐하면 $w_1=w$이므로, 즉, 물건 1의 무게가 5kg이기 때문이다. 따라서

$$K[1,5] = \max\{K[i-1,w],\ K[i-1,w-w_i]+v_i\}$$
$$= \max\{K[1-1,5],\ K[1-1,5-5]+10\}$$
$$= \max\{K[0,5],\ K[0,0]+10\}$$
$$= \max\{0,\ 0+10\}$$
$$= \max\{0,\ 10\} = 10$$ 이다.



수행과정 (6)

- 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.

수행과정 (7~12)

- 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										

수행과정 (13)

- w=10 (배낭의 용량이 10kg)일 때, w_2<w 이므로, w=9일 때와 마찬가지로 K[2,10]=50이고, 물건 1, 2를 배낭에 둘 다 담을 때의 가치인 50을 얻는다는 의미이다.
- 다음은 물건 1과 2에 대해서만 배낭의 용량을 1부터 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	1	0	0	0	0	0	10	10	10	10	10	10
4	40	i = 2	0	**0**	**0**	**0**	**40**	**40**	**40**	**40**	**40**	**50**	**50**
6	30	3	0										
3	50	4	0										

수행과정 (14)

- 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	$\boxed{90}$

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.

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.

요 약

- 편집 거리(Edit Distance) 문제를 위한 동적 계획 알고리즘은,
  - E[i,j]를 3개의 부분 문제 E[i,j-1], E[i-1,j], E[i-1,j-1]만을 참조하여 계산한다. 시간 복잡도는O(mn)이다. 단, m과 n은 두 스트링의 길이이다.
- 배낭(Knapsack) 문제를 위한 동적 계획 알고리즘은,
  - 부분 문제 K[i,w]를 물건 1~i까지만 고려하고, (임시) 배낭의 용량이 w일 때의 최대 가치로 정의하여, i를 1 ~ 물건 수인 n까지, w를 1 ~배낭 용량 C까지 변화시켜가며 해를 찾는다. 시간 복잡도는 O(nC)이다.
- 동적 계획 알고리즘은 부분 문제들 사이의 '관계'를 빠짐없이 고려하여 문제를 해결한다.
  - 동적 계획 알고리즘은 최적 부분구조(optimal substructure) 또는 최적성 원칙(principle of optimality) 특성을 가지고 있다.

실 습

실습1: 편집거리 산출 알고리즘 구현하기

실습2: 배낭문제의 DP알고리즘을 구현하기

숙제:

1. 편집거리 산출 알고리즘에서 편집내용을 출력할 수 있도록 알고리즘을 수정하고 이를 구현하라.(연습문제9)
2. 배낭문제의 DP알고리즘에서 배낭의 물건을 출력할 수 있도록 프로그램을 수정하라.(연습문제13)