# 11주차 2차시 동적 계획 알고리즘의 이해 II

**【학습목표】**

1. 편집거리 문제 알고리즘을 이해할 수 있다.
2. 배낭 문제 알고리즘을 이해할 수 있다.

**지난 강의 정리**

- **주어진 문제를 여러 개의 소문제로 분할하여 각 소문제의 해결안을 바탕으로 주어진 문제를 해결하는 기법**
  - 각 소문제는 원래 주어진 문제와 동일한 문제이지만 입력의 크기가 작음
  - 소문제를 반복 분할하면 결국 입력의 크기가 아주 작은 단순한 문제가 되어 쉽게 해결 가능
  - 소문제의 해를 표 형식으로 저장해 놓고 이를 이용하여 입력 크기가 보다 큰 원래의 문제를 점진적으로 해결
- **최소치/최대치를 구하는 최적화 문제에 적용**

- **분할 정복 방법과 유사**
  - 분할 정복
    - 분할되는 소문제가 서로 독립적
    - 소문제를 순환적으로 풀어 결과를 합침
  - 동적 프로그래밍
    - 소문제가 독립적이지 않음
    - 분할된 소문제 간에 중복 부분이 존재

- $f_n = f_{n-1} + f_{n-2},\ n \geq 2$
- $f_{10} = f_9 + f_8$
- $f_9 = f_8 + f_7$

* 최적성의 원리

◆ **주어진 문제에 대한 최적해가 소문제에 대한 최적해로 구성**

- 욕심쟁이 방법
  - 국부적인 최적해들이 전체적인 최적해를 구성
  - 소문제에 대한 하나의 최적해만을 고려
- 동적 프로그래밍
  - 소문제에 대한 여러 최적해로부터 다음 크기의 소문제에 대한 최적해가 결정

* 모든 쌍 최단 경로 알고리즘

모든 쌍 최단 경로 (All Pairs Shortest Paths) 문제는 각 쌍의 점 사이의 최단 경로를 찾는 문제이다.

* 모든 쌍 최단 경로알고리즘

AllPairsShortest

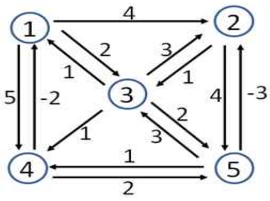
입력: 2차원 배열 D, 단, D[i,j]=선분 (i,j)의 가중치, 만일 선분 (i,j)이 존재하지 않으면 D[i,j]=∞, 모든 i에 대하여 D[i,i]=0이다.

출력: 모든 쌍 최단 경로의 거리를 저장한 2-d 배열 D

```
1. for k = 1 to n
2.     for i = 1 to n (단, i≠k)
3.         for j = 1 to n (단, j≠k, j≠i)
4.             D[i,j] = min{D[i,k]+D[k,j], D[i,j]}
```

KG ITBANK 원격평생교육원
교육부인가학점은행제원격교육훈련기관

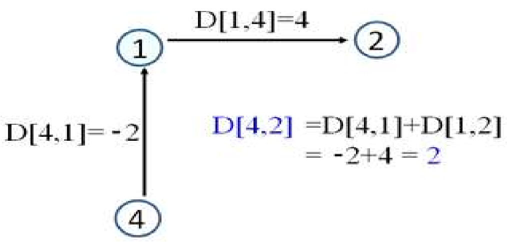
## AllPairsShortest 알고리즘 수행 과정




| D | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 4 | 2 | 5 | ∞ |
| 2 | ∞ | 0 | 1 | ∞ | 4 |
| 3 | 1 | 3 | 0 | 1 | 2 |
| 4 | -2 | ∞ | ∞ | 0 | 2 |
| 5 | ∞ | -3 | 3 | 1 | 0 |

- 배열 D의 원소들이 k가 1부터 5까지 증가함에 따라서 갱신되는 것을 살펴보자.

k=1일 때:
- D[2,3] = min{D[2,3], D[2,1]+D[1,3]} = min{1, ∞+2} = 1
- D[2,4] = min{D[2,4], D[2,1]+D[1,4]} = min{∞, ∞+5} = ∞
- D[2,5] = min{D[2,5], D[2,1]+D[1,5]} = min{4, ∞+∞} = 4
- D[3,2] = min{D[3,2], D[3,1]+D[1,2]} = min{3, 1+4} = 3
- D[3,4] = min{D[3,4], D[3,1]+D[1,4]} = min{1, 1+5} = 1
- D[3,5] = min{D[3,5], D[3,1]+D[1,5]} = min{2, 1+∞} = 2
- D[4,2] = min{D[4,2], D[4,1]+D[1,2]} = min{∞, -2+4} = 2 //갱신됨.




## 학습내용1 : 편집거리 문제 알고리즘

1. 편집거리 문제 알고리즘
- 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.

- 예를 들어, ‘strong’을 ‘stone’으로 편집하여 보자.

| | | | | | | |
|---|---|---|---|---|---|---|
| s | t | | r | o | n | g |
| ↓ | ↓ | 삽입 | 삭제 | 삭제 | ↓ | 대체 |
| s | t | o | | | n | e |

- 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.

- 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.

| | | | | | |
|---|---|---|---|---|---|
| s | t | r | o | n | g |
| ↓ | ↓ | 삭제 | ↓ | ↓ | 대체 |
| s | t | | o | n | e |

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

- 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.

| | | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | | |
| S = | s | t | r | o | n | g |
| T = | s | t | o | n | e | |
| | 1 | 2 | 3 | | | |

• 부분문제를 정의하기 위해서 스트링 S와 T의 길이가 각각 m과 n이라 하고, S와 T의 각 문자를 다음과 같이 si와 tj라고 하자. 단, i = 1, 2, …, m이고, j = 1, 2, …, n이다.

S = s1 s2 s3 s4 … sm
T = t1 t2 t3 t4 … tn

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

예제

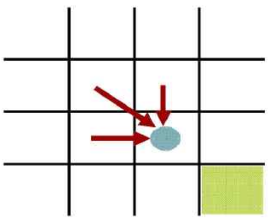
$$X = x_1 x_2 x_3 x_4 x_5 = bbabb$$
$$Y = y_1 y_2 y_3 y_4 = abaa$$
$$\delta_D = \delta_I = 1, \delta_C = 2$$

1. 전체 $x_i$ 삭제 후 전체 $y_i$ 삽입
→ 편집 비용 9
2. $x_1$, $x_4$를 a로 변경, $x_5$ 삭제
→ 편집 비용 5
(최소 비용 → 편집거리)

$$X = x_1 x_2 \cdots x_i,\ Y = y_1 y_2 \cdots y_j$$

$$D(i,j) = \min[D(i-1,j) + \delta_D,\ D(i,j-1) + \delta_I,\ D(i-1,j-1) + 0/\delta_C]$$

$$0/\delta_C = \begin{cases} 0, & x_i = y_i \\ \delta_c, & x_i \neq y_i \end{cases}$$



D(1,1): b를 a로 교환(2)
D(1,2): b를 ab로 a삽입(1)
D(1,3): b를 aba로 a 두 번 삽입(2)
D(1,4): b를 abaa로 a 3번 삽입(3)
D(2,1): bb를 a로 b를 a로 교환 하고 b삭제(3)
D(2,2): bb를 ab로 b를 a로 교환(2)
D(2,3): bb를 aba로 b를 a로 교환 a삽입(3)
D(2,4): bb를 abaa로(4)
.

$$X = abaa,\ Y = bbabb \qquad \delta_D = \delta_I = 1, \delta_C = 2$$

$$D(i,j) = \min[D(i-1,j) + \delta_D,\ D(i,j-1) + \delta_I,\ D(i-1,j-1) + 0/\delta_C]$$

| | | a | b | a | a |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 |
| b | 1 | | | | |
| b | 2 | | | | |
| a | 3 | | | | |
| b | 4 | | | | |
| b | 5 | | | | |

.

.

D(5,4) bbabb를 abaa(5)

| | | a | b | a | a |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 |
| b | 1 | 2 | 1 | 2 | 3 |
| b | 2 | 3 | 2 | 3 | 4 |
| a | 3 | 2 | 3 | 2 | 3 |
| b | 4 | 3 | 2 | 3 | 4 |
| b | 5 | 4 | 3 | 4 | 5 |

최종 D(i,j) 테이블

예제 2)

strong→stone

| 비교문자 | | s | t | o | n | e |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| s | 1 | 0 | 1 | 2 | 3 | 4 |
| t | 2 | 1 | 0 | 1 | 2 | 3 |
| r | 3 | 2 | 1 | 2 | 3 | 4 |
| o | 4 | 3 | 2 | 1 | 2 | 3 |
| n | 5 | 4 | 3 | 2 | 1 | 2 |
| g | 6 | 5 | 4 | 3 | 2 | 3 |

**2. 알고리즘**

```
int EditDist(n, X[], m, Y[], ins, del, chg)
/* 입력 : 문자 배열 X[1..n], Y[1..m],
          삽입 비용 ins, 삭제 비용 del,
          변경 비용 chg
   출력 : 편집 거리     */
{ int D[n+1][m+1];              /* D 테이블 */
  int i, j ;
  D[0][0] = 0;
  for (i=1; i<n+1; i++)       /* 첫 열의 초기화 */
     D[i][0] = D[i-1][0] + del;
  for (j=1; j<m+1; j++)      /* 첫 행의 초기화 */
     D[0][j] = D[0][j-1] + ins;
```

```
  for (i=1; i<n+1; i++)                       O(nm)
     for (j=1; j<m+1; j++) {
        c = (X[i] == Y[j]) ? 0 : chg;
        D[i][j] = min( D[i-1][j]+del,
                       D[i][j-1]+ins,
                       D[i-1][j-1]+c);
     }
  return D[n][m];
}
```

## 학습내용2 : 배낭 문제 알고리즘

**1. 문제의 개념적 정의**

어떤 사람이 보석상에서 마음대로 배낭에 물건을 가져가기로 했다.
보석의 총 무게가 용량 W를 초과하면 배낭이 망가진다.
이 사람은 각 보석의 (무게, 값어치)을 알고 있다.
이 사람은 총 무게가 W를 초과하지 않으면서 보석들의 총 값어치가 최대가 되도록 보석을 배낭에 담고자 한다.

**문제의 정형적 정의**

**○ 입력:**

- $S = \{item_1, item_2, \ldots, item_n\}$
- $w_i$ = $item_i$의 무게
- $p_i$ = $item_i$의 가치
- $W$ = 배낭에 넣을 수 있는 총 무게

**○ 문제 정의**

$\sum_{item_i \in A} w_i \le W$를 만족하면서 $\sum_{item_i \in A} p_i$가 최대가 되도록 $A \subseteq S$가 되는 $A$를 결정하는 문제이다.

**2. 탐욕적 방법 (1)**

**가장 비싼 물건부터 우선적으로 채운다.**

**애석하게도 이 알고리즘은 최적이 아니다!**

**왜 아닌가? 보기:** $W$ = 30kg →

| 품목 | 무게 | 값 |
|---|---|---|
| $item_1$ | 25kg | 10 만원 |
| $item_2$ | 10kg | 9 만원 |
| $item_3$ | 10kg | 9 만원 |

- **탐욕적인 방법**: $item_1 \Rightarrow$ 25kg $\Rightarrow$ 10**만원**
- **최적인 해답**: $item_2$ + $item_3 \Rightarrow$ 20kg $\Rightarrow$ 18**만원**

**3. 탐욕적 방법 (2)**

**가장 가벼운 물건부터 우선적으로 채운다.**

**마찬가지로 이 알고리즘도 최적이 아니다!**

**왜 아닌가? 보기:** $W$ = 30kg →

| 품목 | 무게 | 값 |
|---|---|---|
| $item_1$ | 25kg | 20 만원 |
| $item_2$ | 10kg | 9 만원 |
| $item_3$ | 10kg | 5 만원 |

- **탐욕적인 방법**: $item_2$ + $item_3 \Rightarrow$ 20kg $\Rightarrow$ 14**만원**
- **최적인 해답**: $item_1 \Rightarrow$ 25kg $\Rightarrow$ 20**만원**

**4. 탐욕적 방법 (3)**

**무게 당 가치가 가장 높은 물건부터 우선적으로 채운다.**

**그래도 최적이 아니다!**

**왜 아닌가? 보기:** W = 30kg →

| 품목 | 무게 | 값 | 값어치 |
|---|---|---|---|
| $item_1$ | 5kg | 50 만원 | 10 만원/kg |
| $item_2$ | 10kg | 60 만원 | 6 만원/kg |
| $item_3$ | 20kg | 140 만원 | 7 만원/kg |

- **탐욕적인 방법**: $item_1$ + $item_3 \Rightarrow$ 25kg $\Rightarrow$ 190**만원**
- **최적인 해답**: $item_2$ + $item_3 \Rightarrow$ 30kg $\Rightarrow$ 200**만원**

**더 복잡한 탐욕적 방법을 쓰더라도, 0-1 배낭 채우기 문제는 풀리지 않는다.**

5. 배낭 빈틈없이 채우기 문제

**물건의 일부분을 잘라서 담을 수 있다.**
(보석이 금괴가 아니라 금가루라고 해석하면 된다.)
**탐욕적인 접근방법으로 최적 해를 구하는 알고리즘을 만들 수 있다.**
$item_1 + item_3 + item_2 \times 1/2 \Rightarrow 30kg \Rightarrow 220$**만원**
**최적이다**! → **증명**?

6. 탐욕적인 접근법과 동적계획법 비교

| 탐욕적인 접근방법 | 동적계획법 |
|---|---|
| 최적화 문제를 푸는데 적합 | 최적화 문제를 푸는데 적합 |
| 탐욕적 알고리즘이 존재할 경우 일반적으로 더 효율적 | 때로는 불필요하게 복잡 |
| 알고리즘이 최적인지를 증명해야 함 | 최적화 원칙이 적용되는지를 점검해 보기만 하면 됨 |
| 단일출발점 최단경로 문제: $\Theta(n^2)$ | 단일출발점 최단경로 문제: O(1)xnxC = O(nC) |
| 배낭 빈틈없이 채우기 문제는 풀지만, 0-1 배낭 채우기 문제는 풀지 못함 | 0-1 배낭 채우기 문제를 푼다 |

동적계획법 : Knapsack 알고리즘의 시간복잡도는 O(1)xnxC = O(nC)이다.
배낭 문제에서는 배낭 용량에 제한이 없다. 따라서 배낭 용량이 $c=2^n$이라면 시간 복잡도가 지수 시간이 된다. 그러므로 배낭문제가 다항식 시간에 항상 해결된다고 볼 수 없다.

**【학습정리】**

1. 편집거리 문제 알고리즘
- 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.

2. 배낭 문제 알고리즘
- 배낭 (Knapsack) 문제는 n개의 물건과 각 물건 i의 무게 $w_i$와 가치 $v_i$가 주어지고, 배낭의 용량은 C일 때, 배낭에 담을 수 있는 물건의 최대 가치를 찾는 문제로 탐욕적인 방법과 동적계획 방법이 있다.