# 제 3 장　동적 프로그래밍(계속)

- 문자열 편집
- 0/1 배낭
- 레포트 #4

# 3. 문자열 편집

- 두 문자열 $X = x_1\ x_2\ \ldots\ x_n$ 과 $Y = y_1\ y_2\ \ldots\ y_m$ 이 주어져 있다. 여기서 $x_i$ 와 $y_j$ 가 알파벳으로 알려진 심볼들의 유한 집합이다.
- X 에 대해 일련의 편집 연산을 이용하여 X를 Y로 변환시키려고 한다.
- 가능한 편집 연산은 삽입(insert), 삭제(delete), 교체(change)이고 각각의 연산과 관련된 비용이 주어진다.
- X를 Y로 변환시키는데 드는 비용은 사용된 연산들의 비용 합이다.
- 이 문제는 X를 Y로 변환시키는 편집 연산의 최소-비용 순서를 결정하는 것이다.
- 예제

  $X = x_1\ x_2\ x_3\ x_4\ x_5 = a\ a\ b\ a\ b$

  $Y = y_1\ y_2\ y_3\ y_4 \quad = b\ a\ b\ b$

  삽입과 삭제의 비용은 1, 다른 심볼로 교체하는 비용은 2

경우 1: 각각의 $x_i$ 를 삭제하고 각각의 $y_i$ 를 삽입하는 것 → 비용은 9

경우 2: $x_1$ 과 $x_2$를 삭제하고 문자열 X 의 끝에 $y_4$를 삽입하는 것 → 비용은 3

# 순환 관계식

- 최적성 원칙:

  문자열 편집 문제에 대한 해가 구해질 때 이 해는 일련의 편집연산에 대한 결정으로 구성된다. 이 해의 한 부분은 그 부분에 해당하는 문자열 편집에 대해 최적이어야 한다.

- 순환 관계식:

  cost(i,j) = $x_1\ x_2 \ldots x_i$ 를 $y_1\ y_2 \ldots y_j$ $(0 \le i \le n,\ 0 \le j \le m)$로 변환하는 편집 순서열의 최소비용

  cost(n,m)은 최적 편집 순서열의 비용이 된다.

- i = j = 0 : cost(i,j) = 0

  i = 0, j > 0 : $y_j$ 를 삽입한다.

  cost(0,j) = cost(0, j−1) + $\mathbf{I}(y_j)$ // $\mathbf{I}(\ )$ 는 삽입 비용

  i > 0, j = 0 : $x_i$ 를 삭제한다.

  cost(i,0) = cost(i−1, 0) + $D(x_i)$ // D( ) 는 삭제 비용

# 순환 관계식(계속)

- $i > 0$, $j > 0$ : 다음의 3가지 경우 중 비용이 최소가 되는 것을 선택

  (1) $x_1\ x_2 \ldots x_{i-1}$ 를 $y_1\ y_2 \ldots y_j$ 로 변환하고 $x_i$ 를 삭제

  $\quad cost(i,j) = cost(i-1, j) + D(x_i)$

  (2) $x_1\ x_2 \ldots x_{i-1}$ 를 $y_1\ y_2 \ldots y_{j-1}$ 로 변환하고 $x_i$ 를 $y_j$ 로 교체

  $\quad cost(i,j) = cost(i-1, j-1) + C(x_i,y_j)$ // C( ) 는 교체 비용

  // $x_i = y_j$ 이면 $C(x_i) = 0$

  (3) $x_1\ x_2 \ldots x_i$ 를 $y_1\ y_2 \ldots y_{j-1}$ 로 변환하고 $y_j$ 를 삽입

  $\quad cost(i,j) = cost(i, j-1) + \mathbf{I}(y_j)$

  즉,

  $$cost(i,j) = \min \{cost(i-1, j) + D(x_i),\ cost(i-1, j-1) + C(x_i,y_j),\ cost(i, j-1) + \mathbf{I}(y_j)\}$$

# 순환 관계식(계속)

- 모든 가능한 i 와 j 의 값에 대하여 ($0 \le i \le n$, $0 \le j \le m$) cost(i, j)를 계산하여야 한다.
- 이 값들은 모두 $(n+1)(m+1)$ 개 존재하며, 2차원 배열 M에 저장한다.
- 예제: $X = x_1\ x_2\ x_3\ x_4\ x_5$ = a a b a b

  $Y = y_1\ y_2\ y_3\ y_4$ = b a b b

| i \ j | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 |
| 1 | 1 | 2 | 1 | 2 | 3 |
| 2 | 2 | 3 | 2 | 3 | 4 |
| 3 | 3 | 2 | 3 | 2 | 3 |
| 4 | 4 | 3 | 2 | 3 | 4 |
| 5 | 5 | 4 | 3 | 2 | 3 |

*이 표는 왼쪽에서 오른쪽으로 위쪽에서 아래쪽으로 채워진다.

cost(5,4) = min{cost(4,4)+1,
cost(4,3) + 0,
cost(5,3) + 1} = 3

$X = x_1\, x_2\, x_3\, x_4\, x_5 = a\ a\ b\ a\ b$

$Y = y_1\, y_2\, y_3\, y_4 \quad = b\ a\ b\ b$

| $i \downarrow$ \ $j \rightarrow$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 |
| 1 | 1 | 2 | 1 | 2 | 3 |
| 2 | 2 | 3 | 2 | 3 | 4 |
| 3 | 3 | 2 | 3 | 2 | 3 |
| 4 | 4 | 3 | 2 | 3 | 4 |
| 5 | 5 | 4 | 3 | 2 | 3 |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | aabab | baabab | baaabab | babaabab | babbaabab |
| 1 | abab | babab | | | |
| 2 | bab | | babab | | |
| 3 | ab | bab | | babab | |
| 4 | b | | bab | babb | |
| 5 | – | | | bab | babb |

- 편집 순서열은 cost(i, j) 가 결정될 때 min 값에 해당하는 연산을 edit(i,j)에 저장하여, edit(n,m) 에서부터 역추적하여 결정할 수 있다.
- 최소 비용 편집 순서는 여러 개 존재할 수 있다(모두 비용이 같음).
- 앞의 예에서

  (1) $x_1$을 삭제하고, $x_2$를 삭제하고, 그리고 $y_4$를 삽입하는 것이다.

  또는

  (2) $x_1$을 $y_1$으로 교체하고, $x_4$를 삭제하는 것이다.
- 시간복잡도: O(nm)

# 4. 0/1 배낭

- forward 접근:

  $x_i$ 의 결정은 $x_{i+1}$, $x_{i+2}$, …, $x_n$ 의 최적 결정에 따라 결정된다.
- $g_1(m)$ 을 KNAP(1, n, m) 에 대해 최적 해의 값이라 하자.

  $g_1(m) = \max \{ g_2(m), g_2(m-w_1)+p_1 \}$

  일반화시키면, (단, $g_{n+1}(y) = 0$ 이고 $g_i(y) = -\infty$ , $y < 0$ )

  $g_i(y) = \max \{ g_{i+1}(y), g_{i+1}(y-w_i)+p_i \}$
- 위의 식을 이용하여 다음과 같은 일반적인 순환 관계식을 새로 정의한다.

  f(i, y) = 물건 i, i+1, …, n 과 용량 y 에 대한 최적 해의 값(최대 이익)

$$f(n,y)=\begin{cases} p_n, & y \ge w_n \\ 0, & 0 \le y < w_n \end{cases}$$

$$f(i,y)=\begin{cases} \max\{ f(i+1,y), f(i+1,y-w_i)+p_i\}, & y \ge w_i \\ f(i+1,y), & 0 \le y < w_i \end{cases}$$

# 순환관계식의 순환호출 함수 구현(lec9-1)

```
private static int f(int i, int theCapacity)
   {
      if (i == numberOfObjects)
         return (theCapacity < weight[numberOfObjects])
                  ? 0 : profit[numberOfObjects];
      if (theCapacity < weight[i])
         return f(i + 1, theCapacity);
      return Math.max(f(i + 1, theCapacity),
                     f(i + 1, theCapacity - weight[i]) + profit[i]);
   }
```

- 순환 호출 함수의 호출 :

```
f(1, knapsackCapacity);
```

- 앞의 f(i, y)를 이용하여 f(n, *), f(n−1, *), ..., f(2, *), f(1,m)을 구한다(여기서 *는 용량의 범위를 나타낸다).
- 예: n = 3, c = 116, $p_i$ = (20, 18, 15), $w_i$ = (100, 14, 10)

$$f(3,y)=\begin{cases}0, & 0\le y<10\\ 15, & 10\le y\end{cases}$$

$$f(2,y)=\begin{cases}0, & 0\le y<10\\ 15, & 10\le y<14\\ 18, & 14\le y<24\\ 33, & 24\le y\end{cases}$$

f(1,116) = max {f(2,116), f(2,116−$w_1$)+$p_1$}
= max {f(2,116), f(2,16)+20}
= max {33, 38}
= 38

- $x_i$ 의 결정: $f(i, c) = f(i+1, c)$ 이면 $x_i = 0$
  $f(i, c) \neq f(i+1, c)$ 이면 $x_i = 1$
- 앞의 예에서,
  $f(1,116) = 38 \neq 33 = f(2, 116)$ : $x_1 = 1$
  따라서 $38 - p_1 = 38 - 20 = 18$
  $116 - w_1 = 116 - 100 = 16$
  $f(2, 16) = 18$
  $f(2,16) = 18 \neq 15 = f(3, 16)$ : $x_2 = 1$
  따라서 $18 - p_2 = 18 - 18 = 0$
  $16 - w_2 = 16 - 14 = 2$
  $f(3, 2) = 0$
  이것은 $x_3 = 0$ 을 의미한다.
- 이 예제의 $x_i$ 는 (1, 1, 0) 이다.

# 정수 값의 무게를 갖는 0/1 배낭 문제

- 예: n = 5, m = 10, p = (6, 3, 5, 4, 6)
  w = (2, 2, 6, 5, 4)
  i=5, 4, 3, 2와 y ≤ m (즉 y ≤ 10)에 대해 f(i, y)를 구하여 표를 만든다.

| i \ y | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 0 | 0 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 4 | 0 | 0 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 10 | 10 |
| 3 | 0 | 0 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 10 | 11 |
| 2 | 0 | 0 | 3 | 3 | 6 | 6 | 9 | 9 | 9 | 10 | 11 |

- 위의 표를 이용하여, f(1,10) = max{ f(2,10), f(2,8)+6} = 15 // 최적 해
- f(1,10) = 15 ≠ f(2,10) → $x_1 = 1$, f(2,8) = 9 ≠ f(3,8) → $x_2 = 1$
  f(3,6) = 6 = f(4,6) → $x_3 = 0$, f(4,6) = 6 = f(5,6) → $x_4 = 0$
  f(5,6) ≠ 0 → $x_5 = 1$

# tuple 방법

- P(i) = (y, f(i, y)) 의 순서 리스트
  y의 오름 차순, f(i, y)의 오름 차순으로 정렬됨
- 앞의 표에서,

| i \ y | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 0 | 0 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 4 | 0 | 0 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 10 | 10 |
| 3 | 0 | 0 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 10 | 11 |
| 2 | 0 | 0 | 3 | 3 | 6 | 6 | 9 | 9 | 9 | 10 | 11 |

i=5; P(5) = [(0,0) (4,6)]
i=4; P(4) = [(0,0) (4,6) (9,10)]
i=3; P(3) = [(0,0) (4,6) (9,10) (10,11)]
i=2; P(2) = [(0,0) (2,3) (4,6) (6,9) (9,10) (10,11)]
$f(1,10) = 15 \neq f(2,10) \rightarrow x_1 = 1$, $f(2,8) = 9 \neq f(3,8) \rightarrow x_2 = 1$
…

# P(i)를 효율적으로 구하기

- P(i)는 $x_i$, $x_{i+1}$,…, $x_n$ 의 0/1 조합에 의해 결정된다.
- 앞의 예: n = 5, m = 10, w = (2, 2, 6, 5, 4)

  p = (6, 3, 5, 4, 6)

P(5) = [(0,0) (4,6)]

P(4) = [(0,0) (4,6)] ∪ [(5,4) (9,10)]

P(5) Q = P(5)의 각 tuple에 $(w_4, p_4)$를 더한 집합

= [(0,0) (4,6) (5,4) (9,10)] ← 순서 리스트로 만든다.

인접한 두 tuple (a, b) (c,d) 에서 만약 $a \le c$ 이고 $b>d$ 이면

(c, d)의 선택은 의미가 없으므로 버린다.

= [(0,0) (4,6) (9,10)]

P(3) = [(0,0) (4,6) (9,10) ~~(6,5)~~ (10,11) ~~(15,15)~~]

= [(0,0) (4,6) (9,10) (10,11)]

P(2) = [(0,0) (4,6) (9,10) (10,11) (2,3) (6,9) ~~(11,13)~~ ~~(12,14)~~]

= [(0,0) (2,3) (4,6) (6,9) (9,10) (10,11)]

# 시간 복잡도

- P(i)는 $x_i, x_{i+1}, \ldots, x_n$ 의 0/1 조합에 의해 결정된다.
  $|P(n)| \le 2$
  $|P(n-1)| \le 2 \cdot 2 = 2^2$
  …
  $|P(i)| \le 2^{n-(i-1)} = 2^{n-i+1}$
- P(i)를 구하기 위해, Q를 구하고 이것을 P(i+1)과 합병해야 한다.
  이 시간은 최대  $2 \cdot |P(i+1)|$ 이므로 $O(|P(i+1)|)$로 나타낼 수 있다.
- 모든 P(i)들을 구하는 시간은
  $\sum_{2 \le i \le n} |P(i+1)| = 2^{n-2} + 2^{n-3} + \ldots + 2 = O(2^n)$

# 레포트 #4

- 두 개의 문자열을 입력 받아 최적의 문자열 편집 순서를 결정하는 프로그램을 작성하라.

  편집 순서가 올바른가를 테스트하기 위해 문자열 X 에 적용하여 최종 결과가 문자열 Y가 되는지 확인하는 함수도 포함하라.
- 입력:

  두 문자열의 길이를 입력: n m

  두 문자열을 입력: $x_1$ $x_2$ … $x_n$

  $y_1$ $y_2$ … $y_m$

  출력:

  c(i,j) 표

  최적의 편집 순서열: 맨 앞 연산부터 차례로…

  입력문자열에 차례로 적용하기: ____ ____ ____ …
- 5개 이상의 문자열에 대해 적용하라.