ALGORITHM
알고리즘

Lecture 11

동적 프로그래밍

컴퓨터과학과 | 이관용 교수

한국방송통신대학교

학습목차

01. 기본 개념

동적 프로그래밍?

문제의 크기가 작은 소문제에 대한 해를 테이블에 저장해 놓고 이를 이용해서 크기가 보다 큰 문제의 해를 점진적으로 만들어가는 상향식 접근 방법

- 각 소문제는 원래 주어진 문제와 동일한 문제 → 입력 크기만 작아진 문제
- 소문제들은 서로 독립일 필요는 없음
- 동적 "프로그래밍" → "동적 계획법"
  - ✓ 컴퓨터 프로그램과는 무관, 해를 구축하는 데 테이블을 이용한다는 의미
- 최솟값/최댓값을 구하는 최적화 문제에 주로 사용
  - ✓ "최적성의 원리 principle of optimality"를 반드시 만족하는 문제에만 적용 가능

동적 프로그래밍

최적성의 원리?

- "주어진 문제에 대한 최적해는 주어진 문제의 소문제에 대한 최적해로 구성된다."

동적 프로그래밍 방법의 처리 과정

(0) 최적성의 원리가 주어진 문제에 성립하는 지 확인

(1) 주어진 문제에 대해서 최적해를 제공하는 점화식 도출

(2) 가장 작은 소문제부터 점화식의 해를 구한 뒤 이를 테이블에 저장

(3) 테이블에 저장된 소문제의 해를 이용하여
점차적으로 입력의 크기가 큰 상위 문제의 해를 구해 나감

피보나치 수열

$$f(n) = \begin{cases} f(n-1) + f(n-2) & n \geq 2 \\ 1 & n = 1 \\ 0 & n = 0 \end{cases}$$

f(0)	f(1)	f(2)	f(3)	f(4)	f(5)	f(6)	
0	1	1	2	3	5	8	…

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

```
Fibo(n)
{
  f[0]=0;  f[1]=1;
  for (i=2; i<=n; i++)
    f[i] = f[i-1] + f[i-2];
  return (f[n]);
}
```

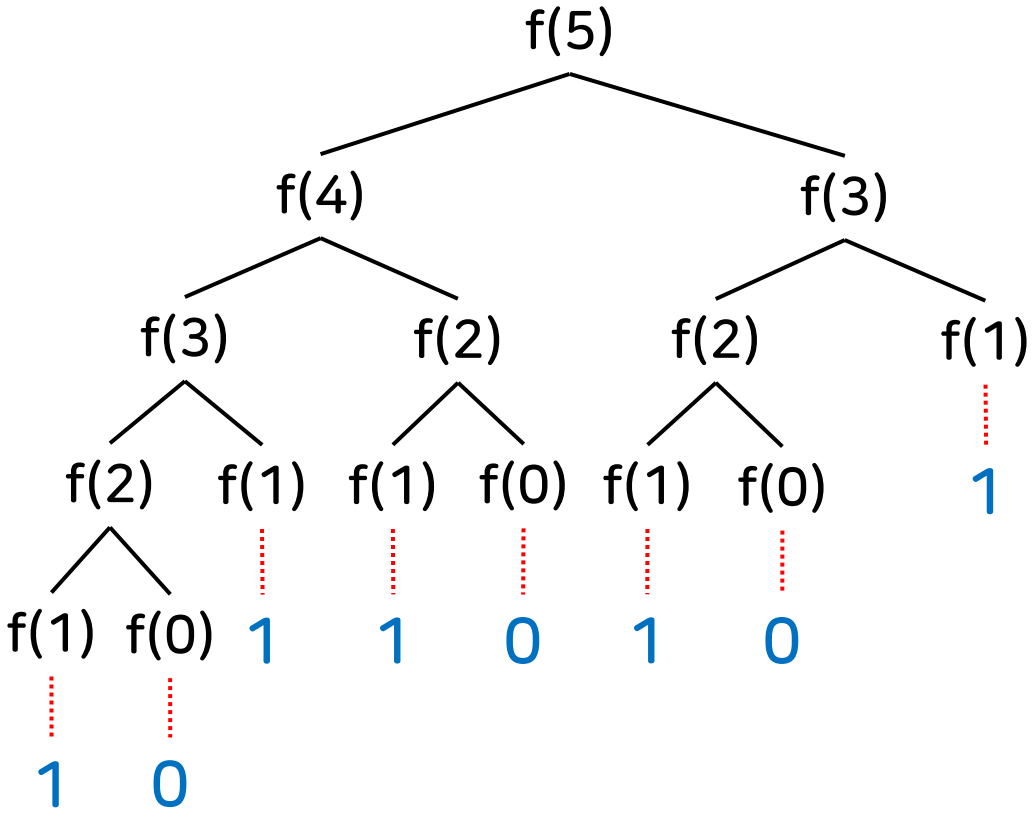
O(n)

피보나치 수열

분할정복 방법을 적용하면

```
f(n)
{
  if (n<=0) return (0);
  if (n==1) return (1);
  else return( f(n-1) + f(n-2) );
}
```

피보나치 수열 f(5) 계산의 구조도



소문제가 독립이 아니므로 중복된 계산이 필요

→ 매우 비효율적

02. 행렬의 연쇄적 곱셈

행렬의 연쇄적 곱셈

n개의 행렬($M_1, M_2, \cdots, M_n$)을 연쇄적으로 곱하는 경우

- 결합법칙 성립
  - ✓ $M_1 \cdot M_2 \cdot M_3 \rightarrow ((M_1 \cdot M_2) \cdot M_3) = (M_1 \cdot (M_2 \cdot M_3))$
  - ✓ 여러 가지 다른 곱셈 순서가 존재 → 곱셈의 횟수가 달라짐

행렬의 연쇄적 곱셈 문제?

- n개의 행렬을 연쇄적으로 곱할 때
  최소의 기본 곱셈 횟수를 갖는 행렬의 곱셈 순서를 구하는 문제

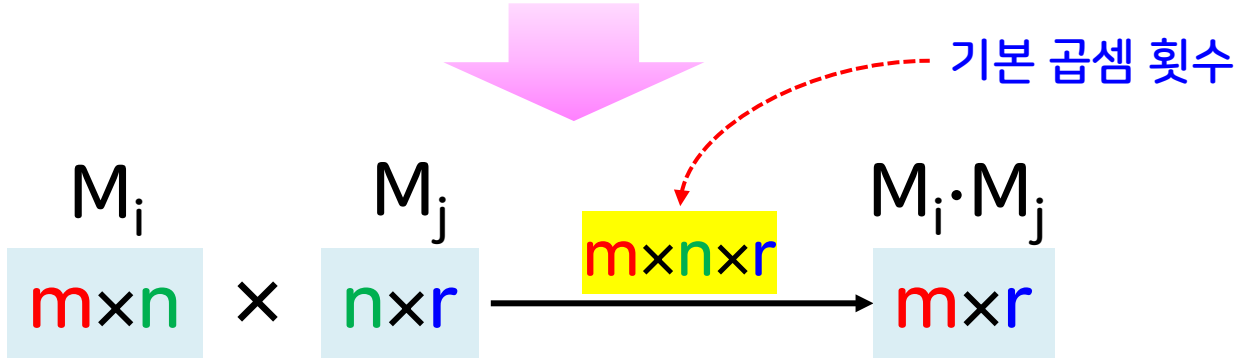
행렬의 연쇄적 곱셈

기본 곱셈

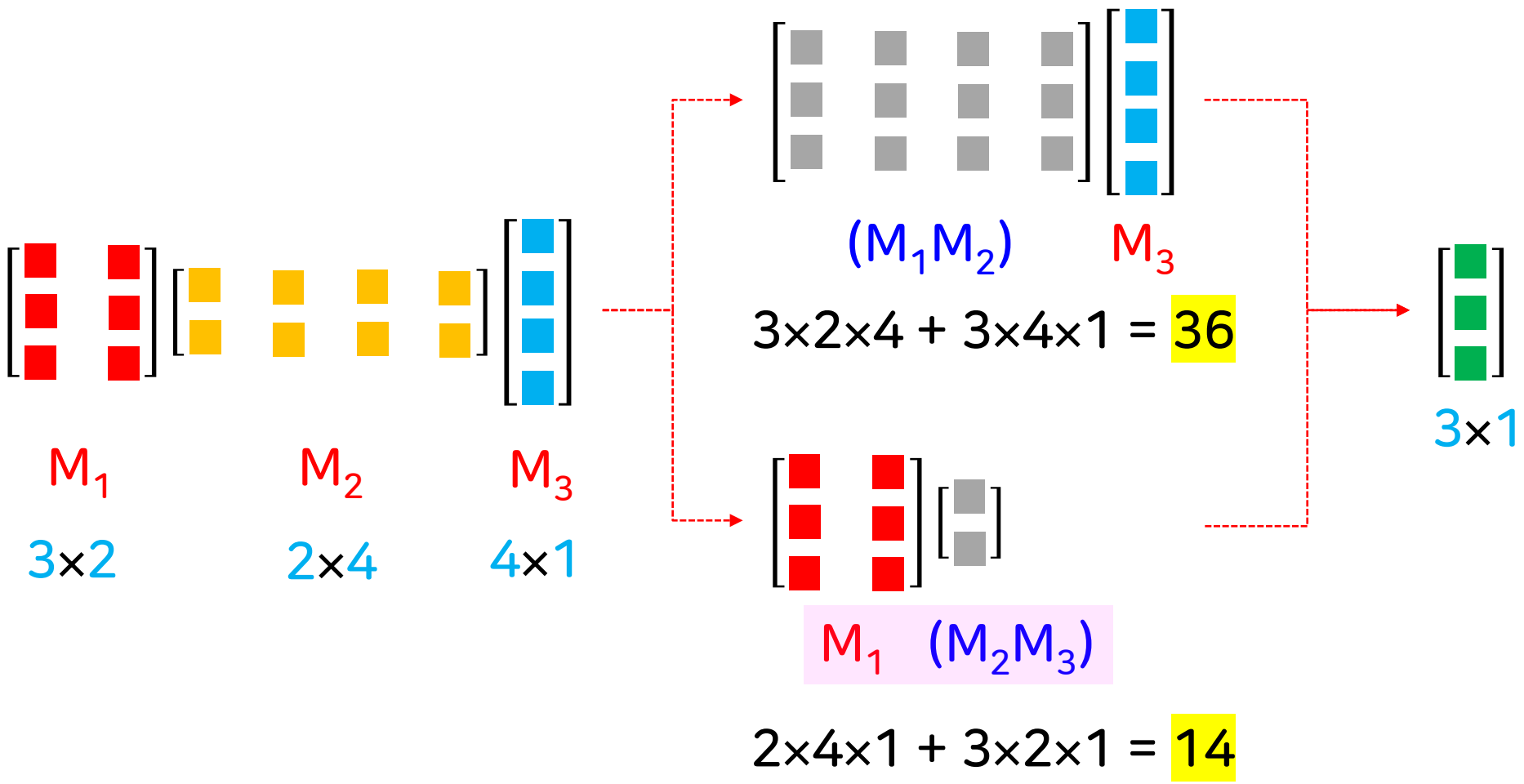
- 행렬 원소끼리의 곱셈

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \\ 9 & 10 \end{bmatrix} \times \begin{bmatrix} 11 & 13 & 15 \\ 12 & 14 & 16 \end{bmatrix} = \begin{bmatrix} 35 & 41 & 47 \\ 81 & 95 & 109 \\ 127 & 149 & 171 \\ 173 & 203 & 233 \\ 219 & 257 & 295 \end{bmatrix}$$

5×2 $\quad$ 2×3 $\quad$ 5×3



행렬의 연쇄적 곱셈



최적성의 원리

n개의 행렬을 곱하는 최적의 순서는 n개 행렬의 연쇄적 곱셈 중 일부분에서의 최적의 곱셈 순서를 포함

7개 행렬을 곱하는 최적의 순서 $(M_1M_2)((((M_3M_4)M_5)M_6)M_7)$

- M_3, M_4, M_5를 곱하는 최적의 순서 $((M_3M_4)M_5)$

↓

부분 문제들의 최적해로 n개 행렬을 곱하는 최적의 순서를 구할 수 있음

↓

최적성의 원리 성립

↓

동적 프로그래밍 방법으로 해결 가능

연쇄 행렬 곱셈 문제의 점화식

n개의 행렬 M_i ($d_{i-1} \times d_i$ 차원) ($1 \le i \le n$)

$$M_1 \times M_2 \times M_3 \times \cdots \times M_{n-1} \times M_n$$

$d_0 \times d_1$ $d_1 \times d_2$ $d_2 \times d_3$ $d_{n-2} \times d_{n-1}$ $d_{n-1} \times d_n$

$C(i, j)$ $1 \le i \le j \le n$ → $C(1, n) = ?$

$M_i \times M_{i+1} \times M_{i+2} \times \cdots \times M_{j-1} \times M_j$ 를 수행하는 데 필요한 기본 곱셈의 최소 횟수

$C(i, i) = 0$

$M_i \times \cdots \times M_k$ $M_{k+1} \times \cdots \times M_j$ 결합 비용

$$C(i, j) = \min_{i \le k < j} \{ C(i, k) + C(k+1, j) + d_{i-1} d_k d_j \} \quad (i<j)$$

값이 가장 작은 분리 위치 k를 찾아야 함

$P(i, j) = k$

$(M_i \times \cdots \times M_k)(M_{k+1} \times \cdots \times M_j)$

행렬의 연쇄적 곱셈 알고리즘

입력: 행렬의 개수 n, 행렬의 크기 d[0..n] (i번째 행렬의 크기 d[i-1]×d[i])
출력: C[1][n] : n개의 행렬을 곱하는 데 필요한 기본 곱셈 횟수의 최소값
P[1..n][1..n] : 최적의 곱셈 순서를 구할 수 있는 배열

```
MinMatMult (n, d[ ])
{
   int  i, j, k, s;
   int  C[n+1][n+1];
   for (i=1; i<=n; i++) C[i][i] = 0;
   for (s=1; s<=n-1; s++)
      for (i=1; i<=n-s; i++) {
         j = i+s;
         C[i][j] = min i≤k<j (C[i][k]+C[k+1][j]+d[i-1]d[k]d[j]);
         P[i][j] = 최소값이 되는 k의 값;
      }
   return (C[1][n], P[1..n][1..n]);
}
```

행렬의 연쇄적 곱셈_예_1

$M_1 \times M_2 \times M_3$

4×2 2×3 3×1 --------→ d[0]=4, d[1]=2, d[2]=3, d[3]=1

C[i][j]

i \ j	1	2	3
1			C[1][3]
2			
3			

s=j-i=0 (i=j)

C[i][i] = 0, 1≤i≤3

C[i][j]

i \ j	1	2	3
1	0		
2		0	
3			0

행렬의 연쇄적 곱셈_예_1

s=j-i=1 (d[0]=4, d[1]=2, d[2]=3, d[3]=1)

$i=1 \rightarrow C[1][2] = \min_{1 \le k < 2} \{ C[1][k] + C[k+1][2] + d[0]d[k]d[2] \}$

$= C[1][1] + C[2][2] + (4 \times 2 \times 3) = 24$

P[1][2] = 1

$i=2 \rightarrow C[2][3] = \min_{2 \le k < 3} \{ C[2][k] + C[k+1][3] + d[1]d[k]d[3] \}$

$= C[2][2] + C[3][3] + (2 \times 3 \times 1) = 6$

P[2][3] = 2

C[i][j]

i \ j	1	2	3
1	0	24	
2		0	6
3			0

P[i][j]

i \ j	1	2	3
1		1	
2			2
3			

행렬의 연쇄적 곱셈_예_1

s=j-i=2 (d[0]=4, d[1]=2, d[2]=3, d[3]=1)

$$i=1 \rightarrow C[1][3] = \min_{1 \le k < 3} \{ C[1][k] + C[k+1][3] + d[0]d[k]d[3] \}$$

$$= \min \{ C[1][1] + C[2][3] + d[0]d[1]d[3], \; C[1][2] + C[3][3] + d[0]d[2]d[3] \}$$

$$= \min \{ 0+6+(4\times2\times1), \; 24+0+(4\times3\times1) \}$$

$$= \min \{ 14, 36 \} = 14$$

P[1][3] = 1

C[i][j]

i \ j	1	2	3
1	0	24	14
2		0	6
3			0

P[i][j]

i \ j	1	2	3
1		1	1
2			2
3			

$M_1 | M_2 M_3 \rightarrow M_1 (M_2 M_3)$

행렬의 연쇄적 곱셈_예_2

M_1 ×	M_2 ×	M_3 ×	M_4 ×	M_5
5×4	4×2	2×3	3×1	1×6

d[0]=5, d[1]=4, d[2]=2, d[3]=3, d[4]=1, d[5]=6

i \ j	1	2	3	4	5
1					C[1][5]
2					
3					
4					
5					

C[i][j]

i \ j	1	2	3	4	5
1					
2					
3					
4					
5					

P[i][j]

행렬의 연쇄적 곱셈_예_2

s=j-i=0

C[i][i] = 0, 1≤i≤5

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

C[i][j]

행렬의 연쇄적 곱셈_예_2

s=j-i=1 d[0]=5, d[1]=4, d[2]=2, d[3]=3, d[4]=1, d[5]=6

$C[1][2] = \min_{1 \le k < 2} \{ C[1][k] + C[k+1][2] + d[0]d[k]d[2] \}$

$= C[1][1] + C[2][2] + 5 \times 4 \times 2 = 0 + 0 + 40 = 40$ P[1][2] = 1

$C[2][3] = \min_{2 \le k < 3} \{ C[2][k] + C[k+1][3] + d[1]d[k]d[3] \}$

$= C[2][2] + C[3][3] + 4 \times 2 \times 3 = 0 + 0 + 24 = 24$ P[2][3] = 2

$C[3][4] = 0 + 0 + 2 \times 3 \times 1 = 0 + 0 + 6 = 6$ P[3][4] = 3

$C[4][5] = 0 + 0 + 3 \times 1 \times 6 = 0 + 0 + 18 = 18$ P[4][5] = 4

C[i][j]

i \ j	1	2	3	4	5
1	0	40			
2		0	24		
3			0	6	
4				0	18
5					0

P[i][j]

i \ j	1	2	3	4	5
1		1			
2			2		
3				3	
4					4
5					

행렬의 연쇄적 곱셈_예_2

s=j-i=2 d[0]=5, d[1]=4, d[2]=2, d[3]=3, d[4]=1, d[5]=6

$C[1][3] = \min_{1 \le k < 3} \{ C[1][k] + C[k+1][3] + d[0]d[k]d[3] \}$

$= \min\{ C[1][1]+C[2][3]+d[0]d[1]d[3], C[1][2]+C[3][3]+d[0]d[2]d[3] \}$

$= \min\{ 0+24+5\times4\times3, 40+0+5\times2\times3 \} = 70$

P[1][3] = 2

$C[2][4] = \min_{2 \le k < 4} \{ C[2][k] + C[k+1][4] + d[1]d[k]d[4] \}$

$= \min\{ C[2][2]+C[3][4]+d[1]d[2]d[4], C[2][3]+C[4][4]+d[1]d[3]d[4] \}$

$= \min\{ 0+6+4\times2\times1, 24+0+4\times3\times1 \} = 14$

P[2][4] = 2

$C[3][5] = \min_{3 \le k < 5} \{ C[2][k] + C[k+1][4] + d[1]d[k]d[4] \}$

$= \min\{ C[2][2]+C[3][4]+d[1]d[2]d[4], C[2][3]+C[4][4]+d[1]d[3]d[4] \}$

$= \min\{ 0+18+2\times3\times6, 6+0+2\times1\times6 \} = 18$

P[3][5] = 4

행렬의 연쇄적 곱셈_예_2

s=j-i=2

i \ j	1	2	3	4	5
1	0	40	70		
2		0	24	14	
3			0	6	18
4				0	16
5					0

C[i][j]

i \ j	1	2	3	4	5
1		1	2		
2			2	2	
3				3	4
4					4
5					

P[i][j]

행렬의 연쇄적 곱셈_예_2

s=j-i=3 d[0]=5, d[1]=4, d[2]=2, d[3]=3, d[4]=1, d[5]=6

$C[1][4] = \min_{1 \le k < 4} \{ C[1][k] + C[k+1][4] + d[0]d[k]d[4] \}$

$= \min\{ C[1][1] + C[2][4] + d[0]d[1]d[4],$

$C[1][2] + C[3][4] + d[0]d[2]d[4]$

$C[1][3] + C[4][4] + d[0]d[3]d[4] \}$

$= \min\{ 0+14+5\times4\times1, 40+6+5\times2\times1, 70+0+5\times3\times1 \}$

$= 34$

P[1][4] = 1

$C[2][5] = \min_{2 \le k < 5} \{ C[2][k] + C[k+1][5] + d[1]d[k]d[5] \}$

$= \min\{ C[2][2] + C[3][5] + d[1]d[2]d[5],$

$C[2][3] + C[4][5] + d[1]d[3]d[5]$

$C[2][4] + C[5][5] + d[1]d[4]d[5] \}$

$= \min\{ 0+18+4\times2\times6, 24+18+4\times3\times6, 14+0+4\times1\times6 \}$

$= 38$

P[2][5] = 4

행렬의 연쇄적 곱셈_예_2

s=j-i=3

i \ j	1	2	3	4	5
1	0	40	70	34	
2		0	24	14	38
3			0	6	18
4				0	16
5					0

C[i][j]

i \ j	1	2	3	4	5
1		1	2	1	
2			2	2	4
3				3	4
4					4
5					

P[i][j]

행렬의 연쇄적 곱셈_예_2

s=j-i=4 d[0]=5, d[1]=4, d[2]=2, d[3]=3, d[4]=1, d[5]=6

$$C[1][5] = \min_{1 \le k < 5} \{ C[1][k] + C[k+1][5] + d[0]d[k]d[5] \}$$

$$= \min\{ C[1][1] + C[2][5] + d[0]d[1]d[5],$$

$$C[1][2] + C[3][5] + d[0]d[2]d[5],$$

$$C[1][3] + C[4][5] + d[0]d[3]d[5],$$

$$C[1][4] + C[5][5] + d[0]d[4]d[5] \}$$

$$= \min\{ 0+38+5\times4\times6,\ 40+18+5\times2\times6,$$

$$70+18+5\times3\times6,\ 34+0+5\times1\times6 \}$$

$$= 64$$

P[1][5] = 4

행렬의 연쇄적 곱셈_예_2

s=j-i=4

C[i][j]

i \ j	1	2	3	4	5
1	0	40	70	34	64
2		0	24	14	38
3			0	6	18
4				0	16
5					0

P[i][j]

i \ j	1	2	3	4	5
1		1	2	1	4
2			2	2	4
3				3	4
4					4
5					

최적의 곱셈 순서

P[1][5] = 4 → $(M_1 M_2 M_3 M_4) M_5$

P[1][4] = 1 → $(M_1 (M_2 M_3 M_4)) M_5$

P[2][4] = 2 → $(M_1 (M_2 (M_3 M_4))) M_5$

성능 및 특징

시간 복잡도 → $O(n^3)$

```
...
 for (s=1; s<=n-1; s++)
   for (i=1; i<=n-s; i++) {
     j = i + s;
     C[i][j] = min i≤k<j (…);
...
```

$$O\left(\sum_{s=1}^{n-1}\{\underbrace{(n-s)}_{\text{i 루프}}\times\underbrace{s}_{\text{k 루프}}\}\right)=O\left(\frac{n(n-1)(n+1)}{6}\right)=O(n^3)$$

(j-1)-i+1 ⟶ j=i+s이므로 (i+s-1)-i+1=s번

성능 및 특징

최적의 행렬 곱셈 순서를 구하는 알고리즘 → O(n)

- P[i][j]를 이용할 때마다 나뉠 부분이 하나씩 줄어듦

  → n개 행렬의 곱셈 순서에서는 n-2번의 분할 지점이 필요 → O(n)

03. 최장 공통 부분 수열

최장 공통 부분 수열?

LCS, Longest Common Subsequence

- 두 스트링의 공통된 부분 수열 중 가장 긴 수열을 구하는 것
  - ✓ 부분 수열 → 스트링에서 연속일 필요는 없지만 순서는 유지되는 스트링의 일부분

    (예) KNOU → O, KO, KU, NO, KOU, KNOU, …
- $X = x_1 x_2 \cdots x_n$과 $Y = y_1 y_2 \cdots y_m$ 사이의 공통 부분 수열 $l_1 l_2 \cdots l_s$
  - ✓ $1 \le k \le s$인 모든 l_k가 스트링 X와 Y에 존재
  - ✓ k가 커질수록 X와 Y에 존재하는 위치도 커짐

$X = \mathrm{S}\,\mathrm{N}\,\mathrm{O}\,\mathrm{W}\,\mathrm{Y}$

$Y = \mathrm{S}\,\mathrm{U}\,\mathrm{N}\,\mathrm{N}\,\mathrm{Y}$

LCS? → "SNY"

LCS에서의 최적성의 원리

두 스트링 X와 Y 사이의 LCS는 이들의 어떤 서브스트링 사이의 LCS를 포함

(1) X의 마지막 글자가 Y의 마지막 글자와 같은 경우

- ✓ $x_1 \cdots x_{n-1}$과 $y_1 \cdots y_{m-1}$의 LCS에 마지막 글자를 추가한 경우

(2) X의 마지막 글자가 Y의 마지막 글자와 다르며 LCS에 사용되지 않은 경우

- ✓ $x_1 \cdots x_{n-1}$과 $y_1 \cdots y_m$의 LCS와 동일

(3) X의 마지막 글자가 Y의 마지막 글자와 다르며 LCS에 사용된 경우

(이 경우 X의 마지막 글자는 Y의 마지막 글자 이전의 다른 글자와 일치)

- ✓ $x_1 \cdots x_n$과 $y_1 \cdots y_{m-1}$의 LCS와 동일

LCS의 점화식

$X = x_1 x_2 \cdots x_n$ 과 $Y = y_1 y_2 \cdots y_m$ 간의 LCS 점화식

빈 스트링

$$LCS(i,j) = \begin{cases} \epsilon & i = 0 \text{ 또는 } j = 0 \text{인 경우} \\ LCS(i-1, j-1) \mid x_i & x_i = y_j \text{인 경우} \\ \max\{LCS(i, j-1), LCS(i-1, j)\} & x_i \neq y_j \text{인 경우} \end{cases}$$

$LCS(n,\ m)$ = ?

알고리즘_LCS 길이와 관련 정보

입력: X[1..n], Y[1..m] : 두 스트링
출력: LCS[n][m] : LCS 길이
LCS[0..n][0..m] : LCS를 구할 수 있는 배열

```
LCS (n, X[ ], m, Y[ ])
{
  int LCS[n+1][m+1];                    // LCS 길이 테이블
  int i, j;
  LCS[0][0] = 0;
  for (i=1; i<=n; i++) LCS[i][0] = 0;   // 첫 열의 초기화
  for (j=1; j<=m; j++) LCS[0][j] = 0;   // 첫 행의 초기화
  for (i=1; i<=n; i++)
    for (j=1; j<=m; j++)
      if (X[i] == Y[j]) LCS[i][j] = LCS[i-1][j-1] + 1;
      else LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1]);
  return (LCS[n][m], LCS[0..n][0..m]);
}
```

알고리즘_LCS 출력

입력: X[1..n], Y[1..m] : 두 스트링
LCS[0..n][0..m] : LCS를 구할 수 있는 배열
출력: L : LCS

```
GetLCS (n, X, m, Y, LCS[ ][ ])
{
  int i = n, j = m, idx = LCS[n][m];
  char L[idx+1];          // LCS를 저장할 배열
  while (idx > 0)         // LCS 길이만큼 공통 문자 찾을 때까지 반복
    if (X[i] == Y[j]) {
      L[idx] = X[i];      // 공통 문자를 LCS로 추가
      idx--; i--; j--;
    }
    else if (LCS[i][j-1] > LCS[i-1][j]) j--;   // LCS[i][j]와 일치하는
          else i--;                            // 큰 값 쪽으로 i 또는 j를 1 줄임
  return (L[1..LCS[n][m]]);
}
```

최장 공통 부분 수열_예_1

스트링 X=abc와 Y=bdc의 LCS?

$$LCS(i,j) = \begin{cases} \epsilon & i = 0 \text{ 또는 } j = 0 \text{인 경우} \\ LCS(i-1, j-1) \mid x_i & x_i = y_j \text{인 경우} \\ \max\{LCS(i, j-1), LCS(i-1, j)\} & x_i \neq y_j \text{인 경우} \end{cases}$$

LCS[ ][ ]

		b	d	c
	0	0	0	0
a	0	0	0	0
b	0			
c	0			

X[1]=a ≠ Y[1]=b

LCS[1][1] ← max{ LCS[1][0] , LCS[0][1] }

X[1]=a ≠ Y[2]=d

LCS[1][2] ← max{ LCS[1][1], LCS[0][2] }

X[1]=a ≠ Y[3]=c

LCS[1][3] ← max{ LCS[1][2], LCS[0][3] }

최장 공통 부분 수열_예_1

$$LCS(i,j) = \begin{cases} \epsilon & i = 0 \text{ 또는 } j = 0 \text{인 경우} \\ LCS(i-1, j-1) \mid x_i & x_i = y_j \text{인 경우} \\ \max\{LCS(i, j-1), LCS(i-1, j)\} & x_i \neq y_j \text{인 경우} \end{cases}$$

LCS[ ][ ]

		b	d	c
	0	0	0	0
a	0	0	0	0
b	0	1	1	1
c	0			

+1

X[2]=b = Y[1]=b

LCS[2][1] ← LCS[1][0] + 1

X[2]=b ≠ Y[2]=d

LCS[2][2] ← max{ LCS[2][1], LCS[1][2] }

X[2]=b ≠ Y[3]=c

LCS[2][3] ← max{ LCS[2][2], LCS[1][3] }

최장 공통 부분 수열_예_1

$$LCS(i,j) = \begin{cases} \epsilon & i = 0 \text{ 또는 } j = 0 \text{인 경우} \\ LCS(i-1, j-1) \mid x_i & x_i = y_j \text{인 경우} \\ \max\{LCS(i, j-1), LCS(i-1, j)\} & x_i \neq y_j \text{인 경우} \end{cases}$$

LCS[ ][ ]

		b	d	c
	0	0	0	0
a	0	0	0	0
b	0	1	1	1
c	0	1	1	2

+1

LCS 길이

X[3]=c ≠ Y[1]=b

LCS[3][1] ← max{ LCS[3][0], LCS[2][1] }

X[3]=c ≠ Y[2]=d

LCS[3][2] ← max{ LCS[3][1], LCS[2][2] }

X[3]=c = Y[3]=c

LCS[3][3] ← LCS[2][2] + 1

최장 공통 부분 수열_예_1

LCS[ ][ ]

		b	d	c
	0	0	0	0
a	0	0	0	0
b	0	1	1	1
c	0	1	1	2

L[ ]

b	c

→ abc와 bdc의 LCS → “bc”

최장 공통 부분 수열_예_2

스트링 X=SNOWY와 Y=SUNNY의 LCS?

LCS[ ][ ]

		S	U	N	N	Y
	0	0	0	0	0	0
S	0	1	1	1	1	1
N	0	1	1	2	2	2
O	0	1	1	2	2	2
W	0	1	1	2	2	2
Y	0	1	1	2	2	3

L[ ]

S	N	Y

성능 및 특징

길이 n과 m인 두 스트링의 LCS 길이 → O(nm)

- 각각 길이 n과 m을 사용하는 중첩된 이중 루프

테이블 LCS[][]로부터 LCS를 구하는 알고리즘 → O(n+m)

- idx가 LCS 길이만큼 줄어들 때까지 반복하는 형태의 루프로 구성
  - ✓ 실제로는 i=n, j=m부터 최대 i=0, j=0까지 도달 가능

정리하기

1. 기본 개념

- 상향식 접근 방법, 소문제는 서로 독립일 필요 없음(중첩 가능)
- 최적성의 원리, 적용 단계, 피보나치 수열

2. 행렬의 연쇄적 곱셈

- n개 행렬의 연쇄적 곱셈에서 최소의 기본 곱셈 횟수를 가진 곱셈 순서를 구하는 문제
- $C(i, j) = \min_{i \le k < j} \{ C(i, k) + C(k+1, j) + d_{i-1} d_k d_j \}$ $(i < j)$
- $O(n^3)$, 최적의 곱셈 순서는 $P(i,j)=k$ 정보를 활용해서 구함

3. 최장 공통 부분 수열

- 두 스트링의 공통된 부분 수열 중 가장 긴 수열을 구하는 문제

$$LCS(i,j) = \begin{cases} \epsilon & i = 0 \text{ 또는 } j = 0 \text{인 경우} \\ LCS(i-1, j-1) \mid x_i & x_i = y_j \text{인 경우} \\ \max\{LCS(i, j-1), LCS(i-1, j)\} & x_i \neq y_j \text{인 경우} \end{cases}$$

- $O(nm)$ $(n=|X|, m=|Y|)$

ALGORITHM
알고리즘

다음시간에는

Lecture 12

스트링 알고리즘 (1)

컴퓨터과학과 | 김진욱 교수

한국방송통신대학교