동적계획

알고리즘해석
강의 슬라이드 #3
도경구

동적계획

- 분할정복식 알고리즘 설계법은 하향식 해결법으로서, 나누어진 부분들 사이에 서로 상관관계가 없는 문제를 해결하는데 적합하다.
- 피보나찌 알고리즘의 경우에는 나누어진 부분들이 서로 연관이 있다. 즉, 분할정복식 방법을 적용하여 알고리즘을 설계하게 되면 같은 항을 한 번 이상 계산하는 결과를 초래하게 되므로 효율적이지가 않다. 따라서 이 경우에는 분할정복식 방법은 적합하지 않다.
- 동적계획법(Dynamic programming)은 상향식 해결법(bottom-up approach)을 사용하여 알고리즘을 설계하는 방법이다. 이 방법은 분할정복식 방법과 마찬가지로 문제를 나눈 후에 나누어진 부분들을 먼저 푼다. 그러나 이미 풀어서 답을 알고 있는 부분의 결과가 다시 필요한 경우에는 반복하여 계산하는 대신에 이미 계산된 결과를 그냥 사용한다.

이항계수 구하기

- 이항계수 구하는 공식

$$\begin{bmatrix} n \\ k \end{bmatrix} = \frac{n!}{k!(n-k)!} \text{ for } 0 \le k \le n$$

- 계산량이 많은 $n!$이나 $k!$을 계산하지 않고 이항계수(binomial coefficient)를 구하기 위해서 통상 다음식을 사용한다.

$$\begin{bmatrix} n \\ k \end{bmatrix} = \begin{cases} \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} + \begin{bmatrix} n-1 \\ k \end{bmatrix} & \text{if } 0 < k < n \\ 1 & \text{if } k = 0 \text{ or } k = n \end{cases}$$

2001 -03 -19 알고리즘 강의 슬라이드 3 3

알고리즘: 분할정복식 접근방법

- 문제: 이항계수를 계산한다.
- 입력: 음수가 아닌 정수 n과 k, 여기서 $k \le n$
- 출력: bin, $\begin{bmatrix} n \\ k \end{bmatrix}$
- 알고리즘:

```
int bin(int n, int k) {
   if (k == 0 || n == k)
     return 1;
   else
     return bin(n-1,k-1) + bin(n-1,k)
}
```

2001 -03 -19 알고리즘 강의 슬라이드 3 4

알고리즘: 분할정복식 접근방법

- 시간복잡도 분석:
  - ✓ 분할정복 알고리즘은 작성하기는 간단하지만, 효율적이지 않다.
  - ✓ 이유? : 알고리즘을 재귀호출(recursive call)할 때 같은 계산을 반복해서 수행하기 때문이다.
  - ✓ 예를 들면, `bin(n-1,k-1)`과 `bin(n-1,k)`는 둘 다 `bin(n-2,k-1)`의 결과가 필요한데, 따로 중복 계산됨
  - ✓ $\begin{bmatrix} n \\ k \end{bmatrix}$을 구하기 위해서 이 알고리즘이 계산하는 항(term)의 개수는 $2\begin{bmatrix} n \\ k \end{bmatrix} - 1$ 이다. (증명을 해보자.)

2001 -03 -19 알고리즘 강의 슬라이드 3 5

- **증명**: (n에 대한 수학적귀납법으로 증명)
- 귀납출발점: 항의 개수 n이 1일 때 $2\begin{bmatrix} n \\ k \end{bmatrix} - 1 = 2 \times 1 - 1 = 1$이 됨을 보이면 된다. $\begin{bmatrix} 1 \\ k \end{bmatrix}$는 $k = 0$이나 1 일 때 1이므로 항의 개수는 항상 1이다.
- 귀납가정: $\begin{bmatrix} n \\ k \end{bmatrix}$을 계산하기 위한 항의 개수는 $2\begin{bmatrix} n \\ k \end{bmatrix} - 1$ 이라고 가정한다.
- 귀납절차: $\begin{bmatrix} n+1 \\ k \end{bmatrix}$을 계산하기 위한 항의 개수가 $2\begin{bmatrix} n+1 \\ k \end{bmatrix} - 1$임을 보이면 된다. 알고리즘에 의해서 $\begin{bmatrix} n+1 \\ k \end{bmatrix} = \begin{bmatrix} n \\ k-1 \end{bmatrix} + \begin{bmatrix} n \\ k \end{bmatrix}$ 이므로, $\begin{bmatrix} n+1 \\ k \end{bmatrix}$를 계산하기 위한 항의 총 개수는 $\begin{bmatrix} n \\ k-1 \end{bmatrix}$을 계산하기 위한 총 개수와 $\begin{bmatrix} n \\ k \end{bmatrix}$를 계산하기 위한 항의 총 개수에다가 이 둘을 더하기 위한 항 1을 더한 수가 된다. 그런데 $\begin{bmatrix} n \\ k-1 \end{bmatrix}$을 계산하기 위한 항의 개수는 가정에 의해서 $2\begin{bmatrix} n \\ k-1 \end{bmatrix} - 1$이고, $\begin{bmatrix} n \\ k \end{bmatrix}$를 계산하기 위한 항의 개수는 가정에 의해서 $2\begin{bmatrix} n \\ k \end{bmatrix} - 1$이다. 따라서 항의 총 개수는

$$
\begin{aligned}
& 2\begin{bmatrix} n \\ k-1 \end{bmatrix} - 1 + 2\begin{bmatrix} n \\ k \end{bmatrix} - 1 + 1 \\
&= 2\left(\frac{n!}{(k-1)!\,(n-k+1)!} + \frac{n!}{k!\,(n-k)!}\right) - 1 \\
&= 2\left(\frac{n!\,(k+n+1-k)}{k!\,(n+1-k)!}\right) - 1 \\
&= 2\left(\frac{n!\,(n+1)}{k!\,(n+1-k)!}\right) - 1 \\
&= 2\left(\frac{(n+1)!}{k!\,(n+1-k)!}\right) - 1 \\
&= 2\begin{bmatrix} n+1 \\ k \end{bmatrix} - 1
\end{aligned}
$$

2001 -03 -19 알고리즘 강의 슬라이드 3 6

동적계획식 알고리즘 설계전략

1. 재귀 관계식(recursive property)을 정립:
   2차원 배열 B를 만들고, 각 $B[i][j]$에는 $\binom{i}{j}$ 값을 저장하도록 하면, 그 값은 다음과 같은 관계식으로 계산할 수 있다.

$$B[i][j] = \begin{cases} B[i-1][j-1] + B[i-1][j] & \text{if } 0 < j < i \\ 1 & \text{if } j = 0 \text{ or } j = i \end{cases}$$

동적계획식 알고리즘 설계전략

2. $\binom{n}{k}$를 구하기 위해서는 다음과 같이 $B[0][0]$부터 시작하여 위에서 아래로 재귀 관계식을 적용하여 배열을 채워 나가면 된다. 결국 값은 $B[n][k]$에 저장된다.

	0	1	2	3	4	j	k
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
i							
n							

$$\begin{array}{cc} B[i-1, j-1] & B[i-1, j] \\ \downarrow & \downarrow \\ \longrightarrow & B[i, j] \end{array}$$

동적계획 알고리즘

- 문제: 이항계수를 계산한다.
- 입력: 음수가 아닌 정수 n과 k, 여기서 $k \le n$
- 출력: bin, $\begin{bmatrix} n \\ k \end{bmatrix}$
- 알고리즘:

```
int bin2(int n, int k) {
  index i, j;
  int B[0..n][0..k];
  for(i=0; i <= n; i++)
    for(j=0; j <= minimum(i,k); j++)
      if (j==0 || j == i)
        B[i][j] = 1;
      else B[i][j] = B[i-1][j-1] + B[i-1][j];
  return B[n][k];
}
```


동적계획 알고리즘의 분석

- 단위연산: for-j 루프 안의 문장
- 입력의 크기: n, k

$i=0$일 때 j-루프 수행 횟수	: 1
$i=1$일 때 j-루프 수행 횟수	: 2
$i=2$일 때 j-루프 수행 횟수	: 3
…………………	
$i=k$-1일 때 j-루프 수행 횟수	: k
$i=k$일 때 j-루프 수행 횟수	: $k+1$
$i=k$+1일 때 j-루프 수행 횟수	: $k+1$
…………………	
$i=n$일 때 j-루프 수행 횟수	: $k+1$

따라서 총 수행횟수는:

$$1+2+3+\cdots+k+\overbrace{(k+1)+\cdots+(k+1)}^{n-k+1\text{ times}}=\frac{k(k+1)}{2}+(n-k+1)(k+1)$$

$$=\frac{(2n-k+2)(k+1)}{2}\in\Theta(nk)$$

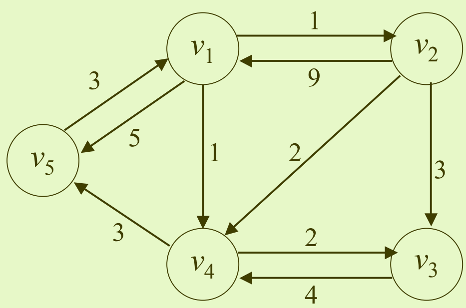
도경구역, 알고리즘, 사이텍미디어, 1999

그래프 용어

- 정점(vertex, node), 이음선(edge, arc)
- 방향 그래프(directed graph)
- 가중치(weight), 가중치 포함 그래프(weighted graph)
- 경로(path), 단순경로(simple path) – 같은 정점을 두번 지나지 않음
- 순환(cycle) – 한 정점에서 다시 그 정점으로 돌아오는 경로
- 순환 그래프(cyclic graph) vs 비순환 그래프(acyclic graph)
- 길이(length)

2001 -03 -19 알고리즘 강의 슬라이드 3 11

가중치 포함 방향 그래프의 예



2001 -03 -19 알고리즘 강의 슬라이드 3 12

최단경로 찾기(Shortest Path)

- 보기 : 한 도시에서 다른 도시로 직항로가 없는 경우 가장 빨리 갈 수 있는 항로를 찾는 문제
- 문제: 가중치 포함, 방향성 그래프에서 최단경로 찾기
- 최적화문제(optimization problem)
  - ✓ 주어진 문제에 대하여 하나 이상의 많은 해답이 존재할 때, 이 가운데에서 가장 최적인 해답(optimal solution)을 찾아야 하는 문제를 최적화문제(optimization problem)라고 한다.
- 최단경로 찾기 문제는 최적화문제에 속한다.

최단경로찾기: 무작정 알고리즘

- 무작정 알고리즘(brute-force algorithm)
  한 정점에서 다른 정점으로의 모든 경로의 길이를 구한 뒤, 그들 중에서 최소길이를 찾는다.
- 분석:
  - ✓ 그래프가 n개의 정점을 가지고 있고, 모든 정점들 사이에 이음선이 존재한다고 가정하자.
  - ✓ 그러면 한 정점 v_i에서 어떤 정점 v_j로 가는 경로들을 다 모아 보면, 그 경로들 중에서 나머지 모든 정점을 한번씩 은 꼭 거쳐서 가는 경로들도 포함되어 있는데, 그 경로들의 수 만 우선 계산해 보자.
  - ✓ v_i에서 출발하여 처음에 도착할 수 있는 정점의 가지 수는 n-2개 이고, 그 중에 하나를 선택하면, 그 다음에 도착할 수 있는 정점의 가지 수는 n-3개 이고, 이렇게 계속하여 계산해 보면, 총 경로의 개수는 $(n-2)(n-3)\dots1 = (n-2)!$이 된다.
  - ✓ 이 경로의 개수 만 보아도 지수보다 훨씬 크므로, 이 알고리즘은 절대적으로 비효율적이다!

동적계획식 설계전략 - 자료구조

- 그래프의 인접행렬(adjacent matrix)식 표현: W

$$W[i][j] = \begin{cases} \text{이음선의 가중치} & v_i\text{에서 } v_j\text{로의 이음선이 있다면} \\ \infty & v_i\text{에서 } v_j\text{로의 이음선이 없다면} \\ 0 & i = j \text{ 이면} \end{cases}$$

- 그래프에서 최단경로의 길이의 표현: $0 \le k \le n$인, $D^{(k)}$
  $D^{(k)}[i][j] = \{v_1, v_2, ..., v_k\}$의 정점들 만을 통해서 v_i에서 v_j로 가는 최단경로의 길이

동적계획식 설계전략 - 자료구조

- 보기:
  - ✓ W : 슬라이드 p.12에 있는 그래프의 인접행렬식 표현
  - ✓ D : 각 정점들 사이의 최단 거리

$W[i][j]$	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

$D[i][j]$	1	2	3	4	5
1	0	1	3	1	4
2	8	0	3	2	5
3	10	11	0	4	7
4	6	7	2	0	3
5	3	4	6	4	0

여기서, $0 \le k \le 5$ 일 때, $D^{(k)}[2][5]$를 구해보자.

- $D^{(0)} = W$이고, $D^{(n)} = D$임은 분명하다. 따라서 D를 구하기 위해서는 $D^{(0)}$를 가지고 $D^{(n)}$을 구할 수 있는 방법을 고안해 내어야 한다.

동적계획식 설계절차

- 1. $D^{(k-1)}$을 가지고 $D^{(k)}$를 계산할 수 있는 재귀 관계식(recursive property)을 정립

$$D^{(k)}[i][j] = minimum(\underbrace{D^{(k-1)}[i][j]}_{경우1}, \underbrace{D^{(k-1)}[i][k] + D^{(k-1)}[k][j]}_{경우2})$$

경우 1: $\{v_1, v_2, \ldots, v_k\}$의 정점들 만을 통해서 v_i에서 v_j로 가는 최단경로가 <u>v_k를 거치지 않는</u> 경우.

보기: $D^{(5)}[1][3] = D^{(4)}[1][3] = 3$

경우 2: $\{v_1, v_2, \ldots, v_k\}$의 정점들 만을 통해서 v_i에서 v_j로 가는 최단경로가 <u>v_k를 거치는</u> 경우.

보기: $D^{(2)}[5][3] = D^{(1)}[5][2] + D^{(1)}[2][3] = 4 + 3 = 7$

보기: $D^{(2)}[5][4]$

- 2. 상향식으로 $k = 1$부터 n까지 다음과 같이 이 과정을 반복하여 해를 구한다.

$$D^{(0)}, D^{(1)}, \ldots\ldots, D^{(n)}$$

Floyd의 알고리즘 I

- 문제: 가중치 포함 그래프의 각 정점에서 다른 모든 정점까지의 최단거리를 계산하라.
- 입력: 가중치 포함, 방향성 그래프 W와 그 그래프에서의 정점의 수 n.
- 출력: 최단거리의 길이가 포함된 배열 D

Floyd의 알고리즘 I

- 알고리즘:

```
void floyd(int n, const number W[][],
           number D[][]) {
  int i, j, k;
  D = W;
  for(k=1; k <= n; k++)
    for(i=1; i <= n; i++)
      for(j=1; j <= n; j++)
        D[i][j] = minimum(D[i][j],
                          D[i][k]+D[k][j]);
}
```

- 모든 경우를 고려한 분석:
  - ✓ 단위연산: for-j 루프안의 지정문
  - ✓ 입력크기: 그래프에서의 정점의 수 n

$$T(n) = n \times n \times n = n^3 \in \Theta(n^3)$$

2001 -03 -19 알고리즘 강의 슬라이드 3 19

Floyd의 알고리즘 II

- 문제: 가중치 포함 그래프의 각 정점에서 다른 모든 정점까지의 최단거리를 계산하고, 각각의 최단경로를 구하라.
- 입력: 가중치 포함 방향성 그래프 W와 그 그래프에서의 정점의 수 n.
- 출력: 최단경로의 길이가 포함된 배열 D, 그리고 다음을 만족하는 배열 P.

$$P[i][j] = \begin{cases} v_i\text{에서 } v_j \text{ 까지 가는 최단경로의 중간에 놓여 있는 정점이 최소한} \\ \quad\text{하나는 있는 경우} \rightarrow \text{그 놓여 있는 정점 중에서 가장 큰 인덱스} \\ \text{최단경로의 중간에 놓여 있는 정점이 없는 경우} \rightarrow 0 \end{cases}$$

2001 -03 -19 알고리즘 강의 슬라이드 3 20

Floyd의 알고리즘 II

- 알고리즘:

```
void floyd2(int n, const number W[][],
                 number D[][], index P[][]) {
   index i, j, k;
   for(i=1; i <= n; i++)
     for(j=1; j <= n; j++)
       P[i][j] = 0;
   D = W;
   for(k=1; k<= n; k++)
     for(i=1; i <= n; i++)
       for(j=1; j<=n; j++)
         if (D[i][k] + D[k][j] < D[i][j]) {
           P[i][j] = k;
           D[i][j] = D[i][k] + D[k][j];
         }
}
```

2001 -03 -19 알고리즘 강의 슬라이드 3 21

Floyd의 알고리즘 II

- 앞의 예를 가지고 *D*와 *P*를 구해 보시오.

$P[i][j]$	1	2	3	4	5
1	0	0	4	0	4
2	5	0	0	0	4
3	5	5	0	0	4
4	5	5	0	0	0
5	0	1	4	1	0

2001 -03 -19 알고리즘 강의 슬라이드 3 22

최단경로의 출력

- 문제: 최단경로 상에 놓여 있는 정점을 출력하라.
- 알고리즘:

```
void path(index q,r) {
  if (P[q][r] != 0) {
    path(q,P[q][r]);
    count << " v" << P[q][r];
    path(P[q][r],r);
  }
}
```

- 위의 P를 가지고 path(5,3)을 구해 보시오.

```
path(5,3) = 4
    path(5,4) = 1
        path(5,1) = 0
        v1
        path(1,4) = 0
    v4
    path(4,3) = 0
```

결과: v1 v4.

즉, v_5에서 v_3으로 가는 최단경로는 v_5, v_1, v_4, v_3,이다.

2001 -03 -19 알고리즘 강의 슬라이드 3 23

동적계획법에 의한 설계 절차

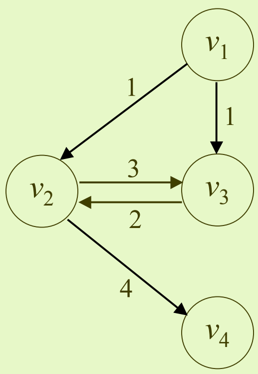
- 문제의 입력에 대해서 최적(optimal)의 해답을 주는 재귀 관계식(recursive property)을 설정
- 상향적으로 최적의 해답을 계산
- 상향적으로 최적의 해답을 구축

2001 -03 -19 알고리즘 강의 슬라이드 3 24

최적의 원칙

- 어떤 문제의 입력에 대한 최적 해가 그 입력을 나누어 쪼갠 여러 부분에 대한 최적 해를 항상 포함하고 있으면, 그 문제는 **최적의 원칙(the principle of optimality)이 적용된다** 라고 한다.
- 보기: 최단경로를 구하는 문제에서, v_k를 v_i에서 v_j로 가는 최적 경로 상의 정점이라고 하면, v_i에서 v_k 로 가는 부분경로와 v_k에서 v_j로 가는 부분경로도 반드시 최적이어야 한다. 이렇게 되면 최적의 원칙을 준수하게 되므로 동적계획법을 사용하여 이 문제를 풀 수 있다.

최적의 원칙이 적용되지 않는 예: 최장경로(Longest Path) 문제



- v_1에서 v_4로의 최장경로는 $[v_1, v_3, v_2, v_4]$가 된다.
- 그러나, 이 경로의 부분 경로인 v_1에서 v_3으로의 최장경로는 $[v_1, v_3]$이 아니고, $[v_1, v_2, v_3]$이다.
- 따라서 최적의 원칙이 적용되지 않는다.
- 주의: 여기서는 단순경로(simple path), 즉 순환(cycle)이 없는 경로만 고려한다.

연쇄 행렬곱셈(Matrix-chain Multiplication)

- $i \times j$ 행렬과 $j \times k$행렬을 곱하기 위해서는 일반적으로 $i \times j \times k$번 만큼의 기본적인 곱셈이 필요하다.
- 연쇄적으로 행렬을 곱할 때, 어떤 행렬곱셈을 먼저 수행하느냐에 따라서 필요한 기본적인 곱셈의 횟수가 달라지게 된다.
- 예를 들어서, 다음 연쇄행렬곱셈을 생각해 보자:
  - ✓ $A_1 \times A_2 \times A_3$.
  - ✓ A_1의 크기는 10×100이고,
  - ✓ A_2의 크기는 100×5이고,
  - ✓ A_3의 크기는 5×50라고 하자.
  - ✓ $A_1 \times A_2$를 먼저 계산한다면, 기본적인 곱셈의 총 횟수는 7,500회
  - ✓ $A_2 \times A_3$를 먼저 계산한다면, 기본적인 곱셈의 총 횟수는 75,000회
  - ✓ 따라서, 연쇄적으로 행렬을 곱할 때 기본적인 곱셈의 횟수가 가장 적게 되는 최적의 순서를 결정하는 알고리즘을 개발하는 것이 목표이다.

연쇄 행렬곱셈 무작정 알고리즘

- 알고리즘: 가능한 모든 순서를 모두 고려해 보고, 그 가운데에서 가장 최소를 택한다.
- 시간복잡도 분석: 최소한 지수(exponential-time) 시간
- 증명:
  - n개의 행렬($A_1, A_2, \ldots, A_n$)을 곱할 수 있는 모든 순서의 가지 수를 t_n이라고 하자.
  - 만약 A_1이 마지막으로 곱하는 행렬이라고 하면, 행렬 $A_2, \ldots, A_n$을 곱하는 데는 t_{n-1}개의 가지수가 있을 것이다.
  - A_n이 마지막으로 곱하는 행렬이라고 하면, 행렬 $A_1, \ldots, A_{n-1}$을 곱하는 데는 또한 t_{n-1}개의 가지수가 있을 것이다.
  - 그러면, $t_n \geq t_{n-1} + t_{n-1} = 2\ t_{n-1}$이고 $t_2 = 1$이라는 사실은 쉽게 알 수 있다.
  - 따라서 $t_n \geq 2t_{n-1} \geq 2^2 t_{n-2} \geq \ldots \geq 2^{n-2} t_2 = 2^{n-2} = \Theta(2^n)$

연쇄 행렬곱셈 동적계획식 설계전략

- d_k를 행렬 A_k의 열(column)의 수라고 하자. 그러면 자연히 A_k의 행(row)의 수는 d_{k-1}가 된다.
- A_1의 행의 수는 d_0라고 하자. 그러면, 다음과 같이 재귀 관계식을 구축할 수 있다. $1 \le i \le j \le n$ 일 때

$M[i][j] =$ $i<j$일 때 A_i부터 A_j까지의 행렬을 곱하는 데 필요한 기본적인 곱셈의 최소 횟수

$$= minimum_{i \le k \le j-1}(M[i][k] + M[k+1][j] + d_{i-1}d_k d_j) \quad \text{if } i < j$$

$$M[i][j] = 0$$

2001 -03 -19 알고리즘 강의 슬라이드 3 29

재귀관계식의 적용 예

- 보기:

A_1	A_2	A_3	A_4	A_5	A_6
5×2	2×3	3×4	4×6	6×7	7×8

$$\begin{aligned} M[4][6] &= minimum_{4\le k\le 5}(M[4][4]+M[5][6]+4\times6\times8, M[4][5]+M[6][6]+4\times7\times8) \\ &= minimum(0+6\times7\times8+4\times6\times8, 4\times6\times7+0+4\times7\times8) \\ &= minimum(528,392) = 392 \end{aligned}$$

$M[i][j]$	1	2	3	4	5	6
1	0	30	64	132	226	348
2		0	24	72	156	268
3			0	72	198	366
4				0	168	392
5					0	336
6						0

2001 -03 -19 알고리즘 강의 슬라이드 3 30

최적 순서의 구축

- 최적 순서를 얻기 위해서는 $M[i][j]$를 계산할 때 최소값을 주는 k값을 $P[i][j]$에 기억한다.
- 예: $P[2][5] = 4$인 경우의 최적 순서는 $(A_2 A_3 A_4) A_5$이다. 구축한 P는 다음과 같다.

$P[i][j]$	1	2	3	4	5	6
1		1	1	1	1	1
2			2	3	4	5
3				3	4	5
4					4	5
5						5

따라서, 최적 분해는 $(A_1((((A_2 A_3) A_4) A_5) A_6))$.

- 최적의 원칙이 이 알고리즘에 적용이 되는가? 생각해 보시오.

2001 -03 -19 알고리즘 강의 슬라이드 3 31

최소곱셈(Minimum Multiplication) 알고리즘

- 문제: n개의 행렬을 곱하는데 필요한 기본적인 곱셈의 횟수의 최소치를 결정하고, 그 최소치를 구하는 순서를 결정하라.
- 입력: 행렬의 수 n, 배열 $d[1..n] \times d[i]$는 i번째 행렬의 규모를 나타낸다.
- 출력: 기본적인 곱셈의 횟수의 최소치를 나타내는 *minmult*; 최적의 순서를 얻을 수 있는 배열 P, 여기서 $P[i][j]$는 행렬 i부터 j까지가 최적의 순서로 갈라지는 기점을 나타낸다.

2001 -03 -19 알고리즘 강의 슬라이드 3 32

최소곱셈(Minimum Multiplication) 알고리즘

- 알고리즘:

```
int minmult(int n, const int d[], index P[][]) {
  index i, j, k, diagonal;
  int M[1..n, 1..n];
  for(i=1; i <= n; i++)
    M[i][j] = 0;
  for(diagonal = 1; diagonal <= n-1; diagonal++)
    for(i=1; i <= n-diagonal; i++) {
      j = i + diagonal;
      M[i][j] = minimum(M[i][k]+M[k+1][j]+
                   d[i-1]*d[k]*d[j]);
                      where i <= k <= j-1
      P[i][j] = 최소치를 주는 k의 값
    }
  return M[1][n];
}
```

2001 -03 -19 알고리즘 강의 슬라이드 3 33

최소곱셈 알고리즘의 모든 경우분석

- 단위연산: 각 k값에 대하여 실행된 명령문(instruction), 여기서 최소값인 지를 알아보는 비교문도 포함한다.
- 입력크기: 곱할 행렬의 수 n
- 분석: $j = i + diagonal$이므로,
  - ✓ k-루프를 수행하는 횟수 =
    $$(j-1)-i+1 = i + diagonal - 1 - i + 1 = diagonal$$
  - ✓ for-i-루프를 수행하는 횟수 = $n - diagonal$
  - ✓ 따라서
    $$\sum_{diagonal=1}^{n-1} [(n - diagonal)\times diagonal] = \frac{n(n-1)(n+1)}{6} \in \Theta(n^3)$$

2001 -03 -19 알고리즘 강의 슬라이드 3 34

최적의 해를 주는 순서의 출력

- 문제: n개의 행렬을 곱하는 최적의 순서를 출력하시오.
- 입력: n과 P.
- 출력: 최적의 순서
- 알고리즘:

```
void order(index i, index j) {
  if (i == j) cout << "A" << i;
  else {
    k = P[i][j];
    cout << "(";
    order(i,k);
    order(k+1,j);
    cout << ")";
  }
}
```

2001 -03 -19 알고리즘 강의 슬라이드 3 35

최적의 해를 주는 순서의 출력

- order(i,j)의 의미: $A_i \times \ldots \times A_j$ 의 계산을 수행하는데 기본적인 곱셈의 수가 가장 적게 드는 순서대로 괄호를 쳐서 출력하시오.
- 분석: $T(n) \in \Theta(n)$. 어떻게?

2001 -03 -19 알고리즘 강의 슬라이드 3 36

다른 알고리즘

- Yao(1982) - $\Theta(n^2)$
- Hu and Shing(1982, 1984) - $\Theta(n \lg n)$