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.

$(\underbrace{A_1 \times A_2}_{10 \times 5}) \times A_3$ $A_1 \times (\underbrace{A_2 \times A_3}_{100 \times 50})$

연쇄 행렬곱셈(Matrix-chain Multiplication)

- 또 다른 예,
  - ✓ $A_1 \times A_2 \times A_3 \times A_4$
  - ✓ A_1: (100×2), A_2: (2×50), A_3: (50×3), A_4: (3×10)
  - ✓ $(((A_1 \times A_2) \times A_3) \times A_4)$:
    - ✓ $100 \times 2 \times 50 + 100 \times 50 \times 3 + 100 \times 3 \times 10 = 14500$
  - ✓ $(A_1 \times (A_2 \times (A_3 \times A_4)))$
    - ✓ $50 \times 3 \times 10 + 2 \times 50 \times 10 + 100 \times 2 \times 10 = 3600$
  - ✓ $((A_1 \times A_2) \times (A_3 \times A_4))$
    - ✓ $100 \times 2 \times 50 + 50 \times 3 \times 10 + 100 \times 50 \times 10 = 61500$
  - ✓ $((A_1 \times (A_2 \times A_3)) \times A_4)$
    - ✓ $2 \times 50 \times 3 + 100 \times 2 \times 3 + 100 \times 3 \times 10 = 3900$
  - ✓ $(A_1 \times ((A_2 \times A_3) \times A_4))$
    - ✓ $2 \times 50 \times 3 + 2 \times 3 \times 10 + 100 \times 2 \times 10 = 2360$

A_1A_2 : 100·2·50 (100×50)
$\times A_3$: 100×50·3 (100·3)
$\times A_4$: 100·3·10 (100·10)
+
14500

연쇄 행렬 곱셈 계산 순서에 대한 고찰

- $A_1 \times A_2 \times \ldots \times A_n$ 의 최소 곱셈 수를 구하기 위해
  - ✓ $A_1 \times (A_2 \times \ldots \times A_n)$
  - ✓ $(A_1 \times A_2) \times (A_3 \times \ldots \times A_n)$
  - ✓ …
  - ✓ $(A_1 \times A_2 \times \ldots \times A_{n-2}) \times (A_{n-1} \times A_n)$
  - ✓ $(A_1 \times A_2 \times \ldots \times A_{n-1}) \times A_n$
- 위에서 살펴 본 $(n\text{-}1)$ 가지의 방법 중 최소 선택

연쇄 행렬곱셈 무작정 알고리즘(억지 기법)

- 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, $t_n \geq t_{n-1} + t_{n-1} = 2t_{n-1}$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_k $d_{k-1} \times d_k$

  - ✓ $A_1 \times A_2 \times A_3 \times A_4$
  - ✓ A_1: (100×2), A_2: (2×50), A_3: (50×3), A_4: (3×10)
  - ✓ d_0 d_1 d_2 d_3 d_4
  - ✓ *100* *2* *50* *3* *10*

연쇄 행렬곱셈 동적계획식 설계전략

- A_1의 행의 수는 d_0라고 하면, 다음과 같이 재귀 관계식을 구축할 수 있다. $\mathbf{1} \le \boldsymbol{i} \le \boldsymbol{j} \le \boldsymbol{n}$ 일 때

$A_i \sim A_j$ $\quad i < j$

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

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

$$\text{if } i < j$$

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

짧은 안내

- 17일 수업 시간에 Pop Quiz 다시 실시
- 범위는 지난번 공지때와 동일함(DP 포함하지 않음)

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
d_0 d_1	d_2	d_3	d_4	d_5	d_6

M[i][j]	1	2	3	4	5	6
1	0	$\times d_2$) 30				
2		5 1 3 0	$\times d_3$) 24			
3			2 3 4 0	$\times d_4$) 72		
4				3 4 6 0	$\times d_5$) 168	
5					4 5 6 0	$\times d_6$) 336
6						6 7 8 0

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
d_0 d_1	d_2	d_3	d_4	d_5	d_6

M[i][j]	1	2	3	4	5	6
1	0	30	**64/1**			
2		0	24			
3			0	72		
4				0	168	
5					0	336
6						0

$A_1A_2A_3$ $A_2A_3A_4$ $A_3A_4A_5$ $A_4A_5A_6$

- M[1][3]
  - ✓ $A_1 \times (A_2 \times A_3)$
    - ✓ M[1][1] + M[2][3] + $d_0 \times d_1 \times d_3$
    - ✓ = 0 + 24 + 5×2×4 = 64
  - ✓ $(A_1 \times A_2) \times A_3$
    - ✓ M[1][2] + M[3][3] + $d_0 \times d_2 \times d_3$
    - ✓ = 30 + 0 + 5×3×4 = 90
  - ✓ 위의 두 결과 중 min 선택

$2\cdot 3\cdot 4$ $d_1 \times d_3$ $(2\ 4)$ $2\cdot 3\cdot 4$ $3\cdot 2\cdot 4$ $=64$

$(5\ 2)$ $5\cdot 2\cdot 3$ $+\ 5\cdot 3\cdot 4$ 90

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

M[i][j]	1	2	3	4	5	6
1	0	30	64/1			
2		0	24	**72/3**		
3			0	72		
4				0	168	
5					0	336
6						0

그룹을 나누는 index

- M[2][4]
  - ✓ $A_2 \times (A_3 \times A_4)$
    - ✓ M[2][2] + M[3][4] + $d_1 \times d_2 \times d_4$
    - ✓ = 0 + 72 + 2×3×6 = 108
  - ✓ $(A_2 \times A_3) \times A_4$
    - ✓ M[2][3] + M[4][4] + $d_1 \times d_3 \times d_4$
    - ✓ = 24 + 0 + 2×4×6 = 72
  - ✓ 위의 두 결과 중 min 선택

재귀관계식의 적용 예

- 보기:

$$A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5 \quad A_6$$
$$5\times2\quad 2\times3 \quad 3\times4 \quad 4\times6 \quad 6\times7 \quad 7\times8$$

M[i][j]	1	2	3	4	5	6
1	0	30	64/1			
2		0	24	72/3		
3			0	72	198/4	
4				0	168	392/5
5					0	336
6						0

- 유사하게 M[3][5], M[4][6] 구할 수 있다.

사전.

빈칸일때 채우기 문제

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

M[i][j]	1	2	3	4	5	6
1	0	30	64/1	**132/1**		
2		0	24	72/3		
3			0	72	198/4	
4				0	168	392/5
5					0	336
6						0

- M[1][4]
  - ✓ $A_1 \times (A_2 \sim A_4)$
    - ✓ M[1][1] + M[2][4] + $d_0 \times d_1 \times d_4$
    - ✓ = 0 + 72 + 5×2×6 = 132
  - ✓ $(A_1 \times A_2) \times (A_3 \times A_4)$
    - ✓ M[1][2] + M[3][4] + $d_0 \times d_2 \times d_4$
    - ✓ = 64 + 72 + 5×3×6 = 226
  - ✓ $(A_1 \sim A_3) \times A_4$
    - ✓ M[1][3] + M[4][4] + $d_0 \times d_3 \times d_4$
    - ✓ = 64 + 0 + 5×4×6 =184
  - ✓ 위의 세 결과 중 min 선택

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

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

- M[2][5]
  - ✓ $A_2 \times (A_3 \sim A_5)$
  - ✓ $(A_2 \times A_3) \times (A_4 \times A_5)$
  - ✓ $(A_2 \sim A_4) \times A_5$
- M[3][6]
  - ✓ $A_3 \times (A_4 \sim A_6)$
  - ✓ $(A_3 \times A_4) \times (A_5 \times A_6)$
  - ✓ $(A_3 \sim A_5) \times A_6$

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

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

- M[1][5]
  - ✓ $A_1 \times (A_2 \sim A_5)$
  - ✓ $(A_1 \times A_2) \times (A_3 \sim A_5)$
  - ✓ $(A_1 \sim A_3) \times (A_4 \times A_5)$
  - ✓ $(A_1 \sim A_4) \times A_5$
- M[2][6]
  - ✓ $A_2 \times (A_3 \sim A_6)$
  - ✓ $(A_2 \times A_3) \times (A_4 \sim A_6)$
  - ✓ $(A_2 \sim A_4) \times (A_5 \times A_6)$
  - ✓ $(A_2 \sim A_5) \times A_6$

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

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

- M[1][6] : Goal
  - ✓ $A_1 \times (A_2 \sim A_6)$
  - ✓ $(A_1 \times A_2) \times (A_3 \sim A_6)$
  - ✓ $(A_1 \sim A_3) \times (A_4 \sim A_6)$
  - ✓ $(A_1 \sim A_4) \times (A_5 \times A_6)$
  - ✓ $(A_1 \sim A_5) \times A_6$
- 곱셈 순서는?
  - ✓ $A_1\times(A_2 \sim A_6)$
  - ✓ $A_1\times((A_2 \sim A_5)\times A_6)$
  - ✓ $A_1\times(((A_2 \sim A_4)\times A_5)\times A_6)$
- $A_1((((A_2 A_3) A_4) A_5) A_6)$

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.

MCM DP 설계전략 다시 보기

- A_1의 행의 수는 d_0라고 하면, 다음과 같이 재귀 관계식을 구축할 수 있다. $\mathbf{1} \le \boldsymbol{i} \le \boldsymbol{j} \le \boldsymbol{n}$ 일 때

$A_i \quad A_{i+1} \quad \cdots \quad A_{j-1} \quad A_j$

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

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

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

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.

최소곱셈(Minimum Multiplication) 알고리즘

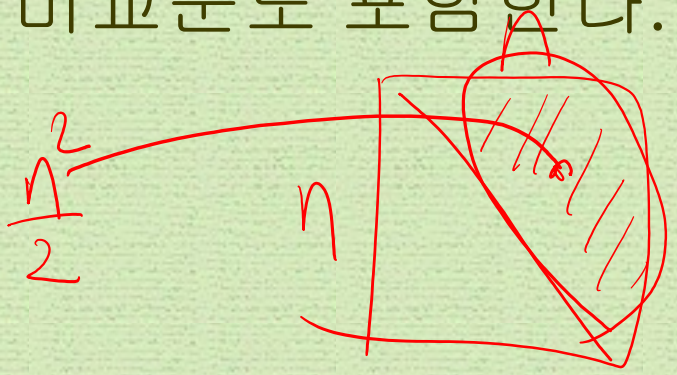
- 알고리즘:

```
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];
}
```

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

$$\sum_{diagonal=1}^{n-1} [(n - diagonal) \times diagonal] = \frac{n(n-1)(n+1)}{6} \in \Theta(n^3)$$

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 << ")";
  }
}
```

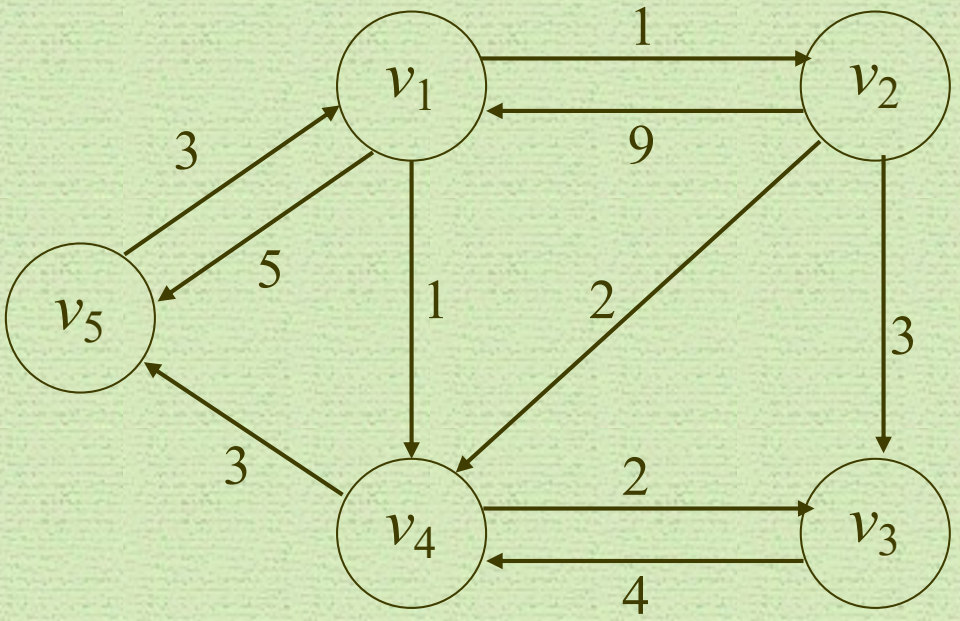
최적의 해를 주는 순서의 출력

- `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?

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 두 번 지나지 않음
- ✓ 순환(cycle) – 한 정점에서 다시 그 정점으로 돌아오는 경로
- ✓ 순환 그래프(cyclic graph) vs 비순환 그래프(acyclic graph)
- ✓ 길이(length)

가중치 포함 방향 그래프의 예



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-2개이고, 그 중에 하나를 선택하면, 그 다음에 도착할 수 있는 정점의 가지 수는 n-3개 이고, 이렇게 계속하여 계산해 보면, 총 경로의 개수는 $(n-2)(n-3)\ldots 1 = (n-2)!$이 된다.
  - ✓ 이 경로의 개수 만 보아도 지수보다 훨씬 크므로, 이 알고리즘은 절대적으로 비효율적이다!

동적계획식 설계전략 - 자료구조

- 그래프의 인접행렬(adjacent matrix)식 표현: W

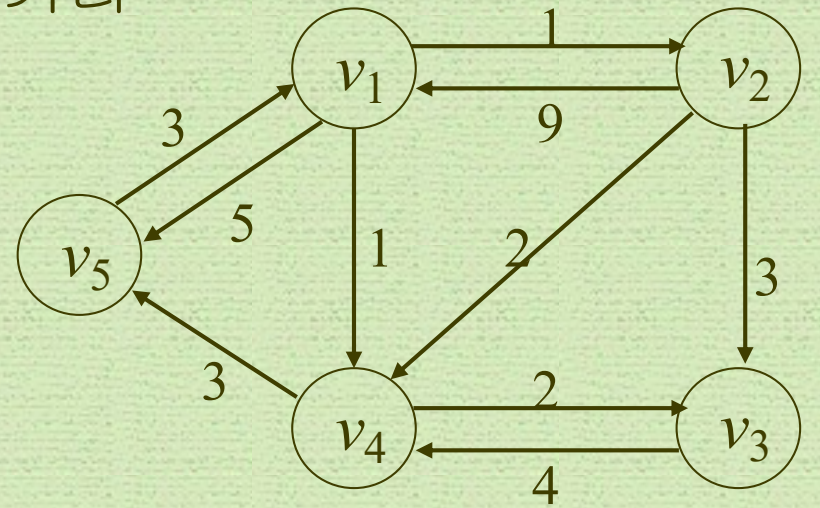
$$W[i][j] = \begin{cases} \text{이음선의 가중치} & v_i\text{에서 } v_j\text{로의 이음선이 있다면} \\ \infty & v_i\text{에서 } v_j\text{로의 이음선이 없다면} \\ 0 & i = j \text{ 이면} \end{cases}$$

- 그래프에서 최단경로의 길이의 표현: $0 \le k \le n$인, $D^{(k)}$
  $D^{(k)}[i][j] = \{v_1, v_2, \ldots, v_k\}$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



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 설계전략 - 자료구조

- 보기:
  - ✓ W : 슬라이드 p.13에 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

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

경우 1: $\{v_1, v_2, \ldots, v_k\}$의 정점들 만을 통해서 v_i에서 v_j로 가는 최단경로가 v_k를 거치지 않는 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 v_k를 거치는 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

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

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.

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

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];
        }
}
```

Floyd의 알고리즘 II

- 앞의 예를 가지고 D와 P를 구해 보시오.

$D^{(0)}$	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

$D^{(1)}$	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	∞	4	0

$P^{(0)}$	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

$P^{(1)}$	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	1
3	0	0	0	0	0
4	0	0	0	0	0
5	0	1	0	1	0

Floyd의 알고리즘 II

- 앞의 예를 가지고 D와 P를 구해 보시오.

$D^{(1)}$	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	∞	4	0

$D^{(2)}$	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$P^{(1)}$	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	1
3	0	0	0	0	0
4	0	0	0	0	0
5	0	1	0	1	0

$P^{(2)}$	1	2	3	4	5
1	0	0	2	0	0
2	0	0	0	0	1
3	0	0	0	0	0
4	0	0	0	0	0
5	0	1	2	1	0

Floyd의 알고리즘 II

- 앞의 예를 가지고 D와 P를 구해 보시오.

$D^{(2)}$	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(3)}$	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$P^{(2)}$	1	2	3	4	5
1	0	0	2	0	0
2	0	0	0	0	1
3	0	0	0	0	0
4	0	0	0	0	0
5	0	1	2	1	0

$P^{(3)}$	1	2	3	4	5
1	0	0	2	0	0
2	0	0	0	0	1
3	0	0	0	0	0
4	0	0	0	0	0
5	0	1	2	1	0

Floyd의 알고리즘 II

- 앞의 예를 가지고 D와 P를 구해 보시오.

$D^{(3)}$	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(4)}$	1	2	3	4	5
1	0	1	3	1	4
2	9	0	3	2	5
3	∞	∞	0	4	7
4	∞	∞	2	0	3
5	3	4	6	4	0

$P^{(3)}$	1	2	3	4	5
1	0	0	2	0	0
2	0	0	0	0	1
3	0	0	0	0	0
4	0	0	0	0	0
5	0	1	2	1	0

$P^{(4)}$	1	2	3	4	5
1	0	0	4	0	4
2	0	0	0	0	4
3	0	0	0	0	4
4	0	0	0	0	0
5	0	1	4	1	0

Floyd의 알고리즘 II

- 앞의 예를 가지고 D와 P를 구해 보시오.

$D^{(4)}$	1	2	3	4	5
1	0	1	3	1	4
2	9	0	3	2	5
3	∞	∞	0	4	7
4	∞	∞	2	0	3
5	3	4	6	4	0

$D^{(5)}$	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

$P^{(4)}$	1	2	3	4	5
1	0	0	4	0	0
2	0	0	0	0	4
3	0	0	0	0	4
4	0	0	0	0	0
5	0	1	4	1	0

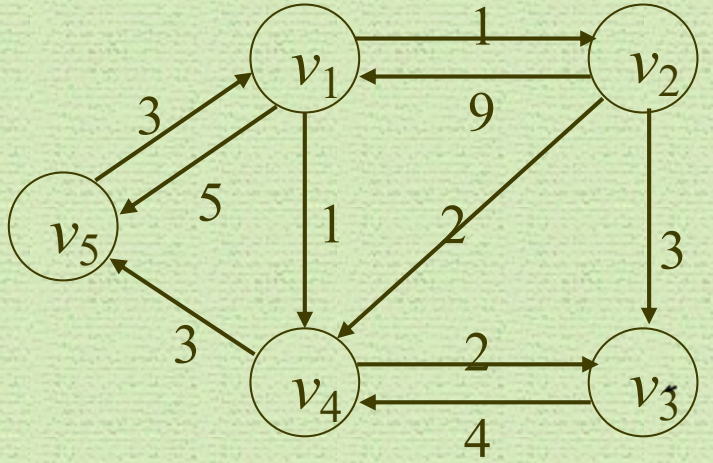
$P^{(5)}$	1	2	3	4	5
1	0	0	4	0	4
2	5	0	0	0	4
3	5	5	0	0	4
4	5	5	0	0	0
5	0	1	4	1	0

Floyd의 알고리즘 II

D	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

P	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

$W[i][j]$	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



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
```

<u>결과</u>: v1 v4.

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

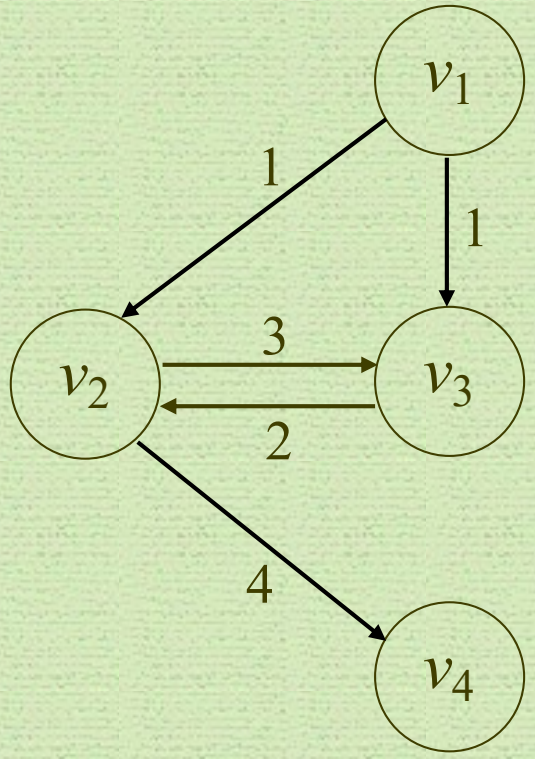
동적계획법에 의한 설계 절차

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

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.

Find a Longest Common Subsequence

B

			G	V	C	E	K	S	T
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	1	1	1	1
D	2	0	1	1	1	1	1	1	1
V	3	0	1	2	2	2	2	2	2
E	4	0	1	2	2	3	3	3	3
G	5	0	1	2	2	3	3	3	3
T	6	0	1	2	2	3	3	3	4
A	7	0	1	2	2	3	3	3	4

A

$$c[i,j]=\begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } a[i]=b[j] \\ \max\{c[i-1,j],c[i,j-1]\} & \text{if } i,j>0 \text{ and } a[i]\neq b[j] \end{cases}$$

TEUG

GVET

길이 4

Find a Longest Common Subsequence

B

A

			G	V	C	E	K	S	T	A	G	T
		0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	1	1	1	1	1	1	1
D	2	0	1	1	1	1	1	1	1	1	1	1
V	3	0	1	2	2	2	2	2	2	2	2	2
E	4	0	1	2	2	3	3	3	3	3	3	3
G	5	0	1	2	2	3	3	3	3	3	4	4
T	6	0	1	2	2	3	3	3	4	4	4	5
A	7	0	1	2	2	3	3	3	4	5	5	5

ATEVG

GVETA · 5

How many different LCSes?

How to find such LCSes?

Find a Longest Common Subsequence

B

A

			G	V	C	E	K	S	T	A	G	T
		0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	1	1	1	1	1	1	1
D	2	0	1	1	1	1	1	1	1	1	1	1
V	3	0	1	2	2	2	2	2	2	2	2	2
E	4	0	1	2	2	3	3	3	3	3	3	3
G	5	0	1	2	2	3	3	3	3	3	4	4
T	6	0	1	2	2	3	3	3	4	4	4	5
A	7	0	1	2	2	3	3	3	4	5	5	5

TGEVG

How many different LCSes?

How to find such LCSes?

GVEGT → 5

Find a Longest Common Subsequence

B

			G	V	C	E	K	S	T	A	G	T
		0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	1	1	1	1	1	1	1
D	2	0	1	1	1	1	1	1	1	1	1	1
V	3	0	1	2	2	2	2	2	2	2	2	2
E	4	0	1	2	2	3	3	3	3	3	3	3
G	5	0	1	2	2	3	3	3	3	3	4	4
T	6	0	1	2	2	3	3	3	4	4	4	5
A	7	0	1	2	2	3	3	3	4	5	5	5

A

How many different LCSes?

How to find such LCSes?

키가 클수록 머리가 좋다?

- 원숭이 키와 IQ의 관계: 우측 표 참조
- 위의 주장이 틀렸다는 것을 보이기 위해 키가 커지면서 머리가 나빠지는 가장 긴 순서를 찾아 보자. (즉, 찾은 순서에서 키는 반드시 증가해야 하며, IQ는 반드시 감소해야 함)
- How?

이름	키	IQ
A	84	13
B	80	21
C	67	20
D	69	40
E	70	30
F	75	20
G	98	14
H	72	12
I	75	29
J	72	19

이름				D	E	I	B	F	C	J	G	A	H
	키			69	70	75	80	75	67	72	98	84	72
		IQ		40	30	29	21	20	20	19	14	13	12
			0	0	0	0	0	0	0	0	0	0	0
C	67	20	0	0	0	0	0	0	1	1	1	1	1
D	69	40	0	1	1	1	1	1	1	1	1	1	1
E	70	30	0	1	2	2	2	2	2	2	2	2	2
H	72	12	0	1	2	2	2	2	2	2	2	2	3
J	72	19	0	1	2	2	2	2	2	3	3	3	3
F	75	20	0	1	2	2	2	3	3	3	3	3	3
I	75	29	0	1	2	3	3	3	3	3	3	3	3
B	80	21	0	1	2	3	4	4	4	4	4	4	4
A	84	13	0	1	2	3	4	4	4	4	4	5	5
G	98	14	0	1	2	3	4	4	4	4	5	5	5

How to find an LCS?

DEIBG G

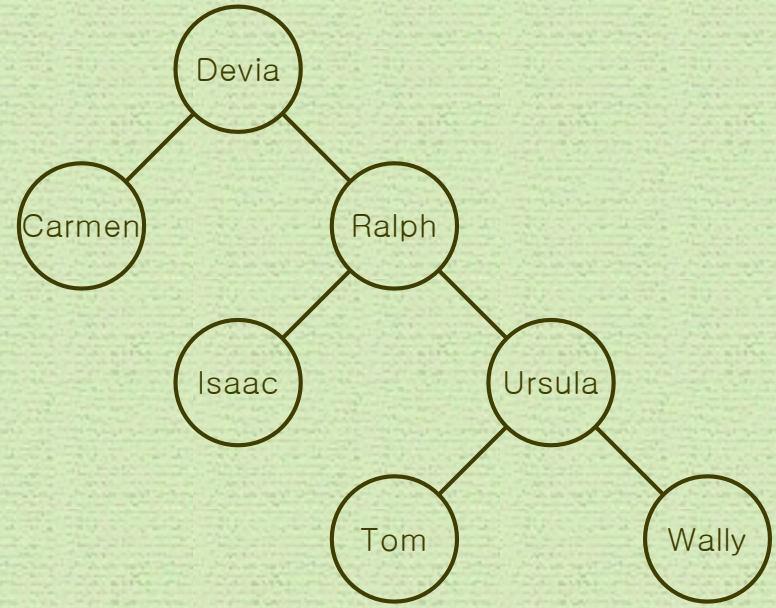
DEIBA G

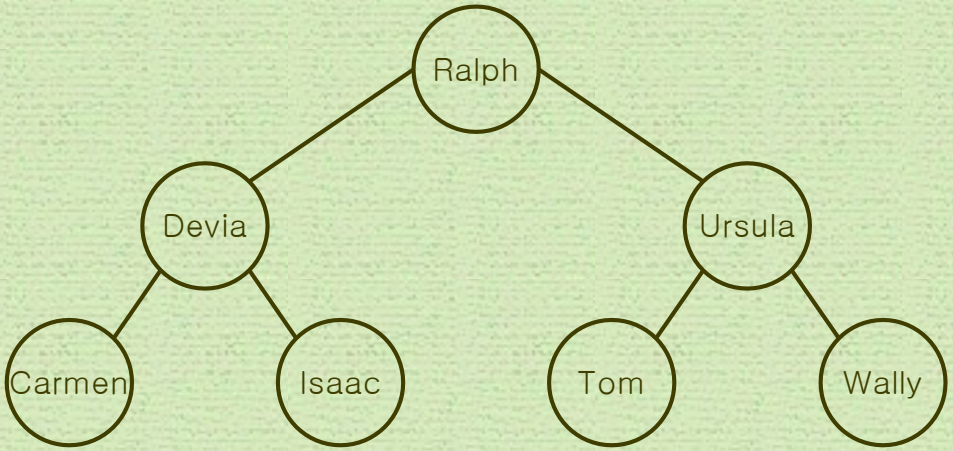
Optimal Binary Search Tree (최적 이진 검색 트리)

- 정의
  - 이진검색트리는 순서가능집합(ordered set: 순서를 매길 수 있는 요소로 구성된 집합)에 속한 item(key 라고도 함)으로 구성된 이진 트리인데, 다음 조건을 만족한다.

1. 각 마디는 하나의 키만 가지고 있다.
2. 주어진 마디의 왼쪽 부분트리(subtree)에 있는 키는 그 마디의 키보다 작거나 같다.
3. 주어진 마디의 오른쪽 부분트리에 있는 키는 그 마디의 키보다 크거나 같다.

이진 검색 트리의 예

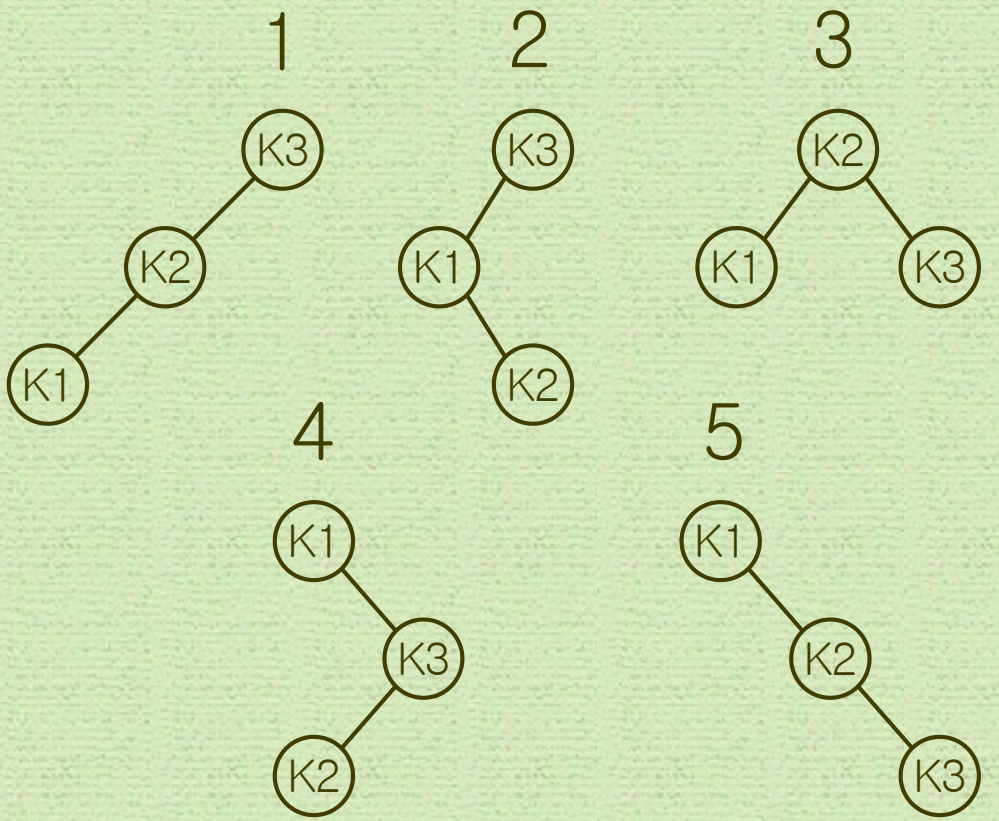

Devia
Carmen
Ralph
Isaac
Ursula
Tom
Wally


Ralph
Devia
Ursula
Carmen
Isaac
Tom
Wally

평균 검색시간

- 키가 3개인 트리
- 각 키의 검색확률: $p_1=0.7$, $p_2=0.2$, $p_3=0.1$
- 평균 검색 시간
  1. $3(0.7)+2(0.2)+1(0.1)=2.6$
  2. $2(0.7)+3(0.2)+1(0.1)=2.1$
  3. $2(0.7)+1(0.2)+2(0.1)=1.8$
  4. $1(0.7)+3(0.2)+2(0.1)=1.5$
  5. $1(0.7)+2(0.2)+3(0.1)=1.4$

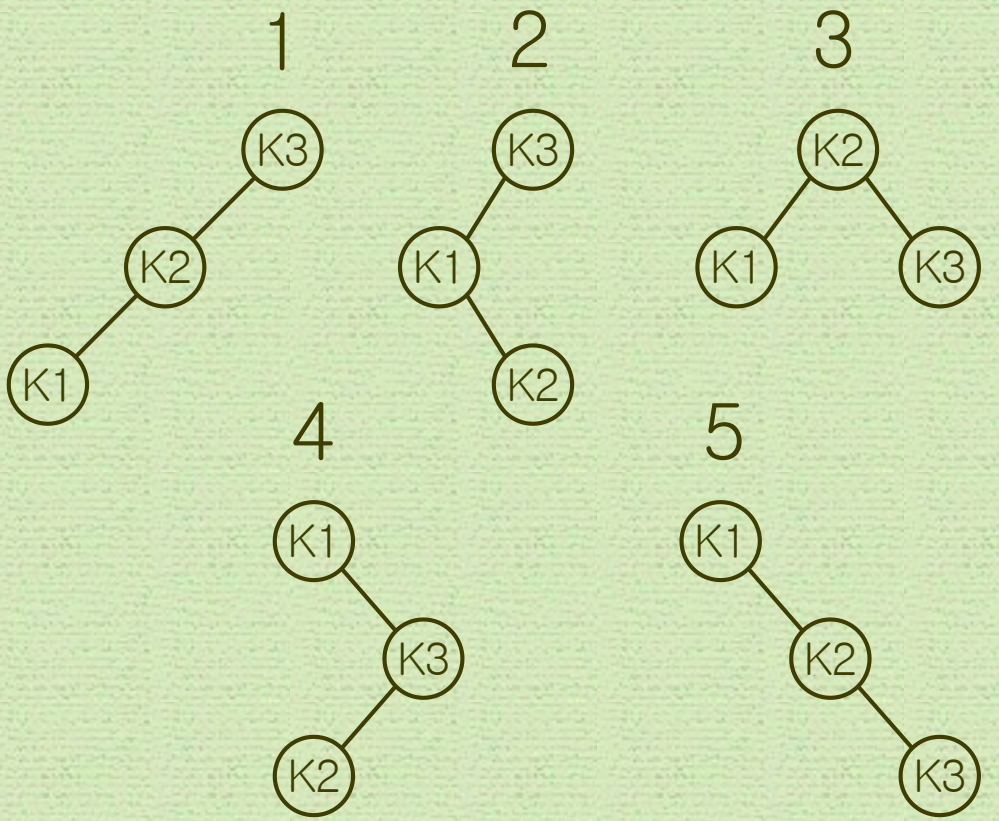
→ 트리 5가 최적



평균 검색시간

- 각 키의 검색확률이 아래와 같이 바뀌면?
- 각 키의 검색확률: $p_1=0.3$, $p_2=0.2$, $p_3=0.5$
- 평균 검색 시간
  1. $3(0.3)+2(0.2)+1(0.5)=1.8$
  2. $2(0.3)+3(0.2)+1(0.5)=1.7$
  3. $2(0.3)+1(0.2)+2(0.5)=1.8$
  4. $1(0.3)+3(0.2)+2(0.5)=1.9$
  5. $1(0.3)+2(0.2)+3(0.5)=2.2$

→ 트리 2가 최적



동적계획법을 사용한 OBST 구성

- key_i에서 key_j까지의 키들은 다음 식을 최소화하는 트리에 정리되어 있다고 가정

$$\sum_{m=i}^{j} c_m p_m$$

여기서 c_m은 그 트리에서 key_m을 찾는데 필요한 비교횟수.

이러한 트리를 그 키들에 대해 최적이라 하고, 최적값은 A[i][j]로 표시함.

- $A[i][i]=p_i$

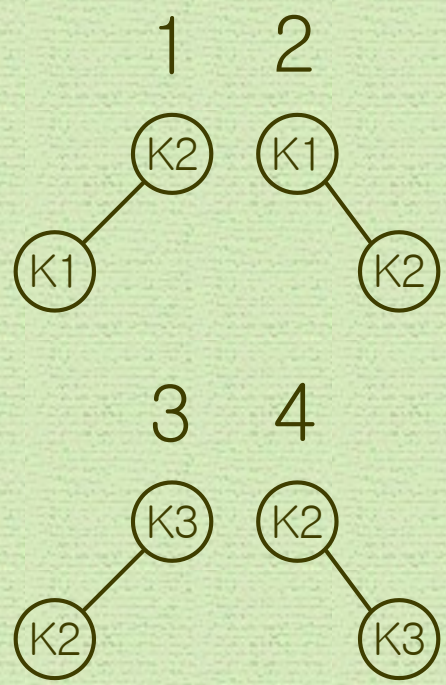
동적계획법을 사용한 OBST 구성

- 키가 3개, 각 키의 검색확률: p_1=0.7, p_2=0.2, p_3=0.1
- A[1][2]를 구하기 위해선 트리 1과 트리 2를 고려해야 함
  1. 2(0.7)+1(0.2)=1.6
  2. 1(0.7)+2(0.2)=1.1

  → 트리 2가 최적이고, A[1][2]=1.1

- A[2][3]를 구하기 위해선 트리 3과 트리 4를 고려해야 함
  3. 2(0.2)+1(0.1)=0.5
  4. 1(0.2)+2(0.1)=0.4

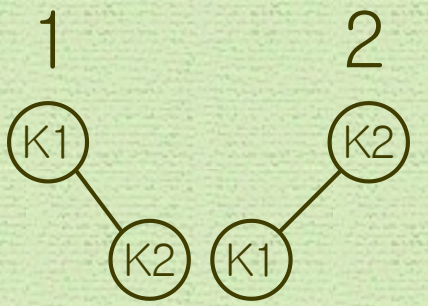
  → 트리 4가 최적이고, A[2][3]=0.4



동적계획법을 사용한 OBST 구성

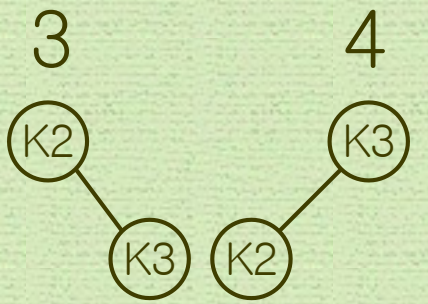
- 키가 3개, 각 키의 검색확률: p_1=0.7, p_2=0.2, p_3=0.1
- A[1][2]를 구하기 위해선 트리 1과 트리 2를 고려해야 함
  1. 1(0.7)+2(0.2)=1.1
  2. 2(0.7)+1(0.2)=1.6

  → 트리 1이 최적이고, A[1][2]=1.1

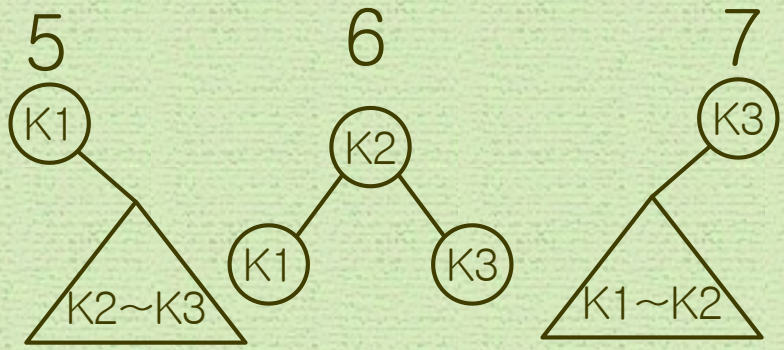


- A[2][3]을 구하기 위해선 트리 3과 트리 4를 고려해야 함
  3. 1(0.2)+2(0.1)=0.4
  4. 2(0.2)+1(0.1)=0.5

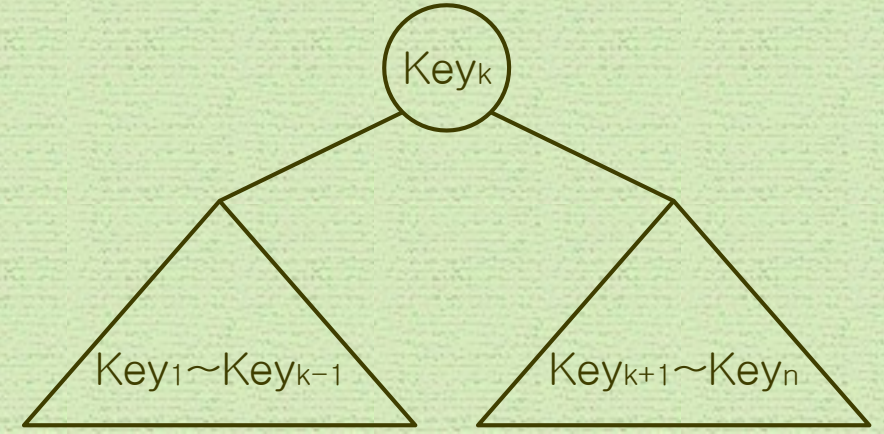
  → 트리 3이 최적이고, A[2][3]=0.4



- A[1][3]: 트리 5~7을 고려해야 함
  5. A[2][3]+0.7+0.2+0.1=1.4
  6. A[1][1]+A[3][3]+ 0.7+0.2+0.1=1.8
  7. A[1][2]+0.7+0.2+0.1=2.1



동적계획법을 사용한 OBST 구성



$$A[1][k-1]+p_1+\ldots+p_{k-1} + p_k + A[k+1][n]+ p_{k+1}+ \ldots +p_n$$

- $A[1][k-1]+A[k+1][n]+\sum_{m=1}^{n} p_m$
- $A[1][n]=\min_{1\le k\le n}(A[1][k-1]+A[k+1][n])+\sum_{m=1}^{n} p_m$

Example

- 키가 4개인 OBST를 구하라.
- Key : Don, Isaac, Ralph, Wally
- 각 키의 검색확률: $p_1=3/8$, $p_2=3/8$, $p_3=1/8$, $p_4=1/8$

Example

- $p_1=3/8$, $p_2=3/8$, $p_3=1/8$, $p_4=1/8$

	0	1	2	3	4
1	0	3/8			
2		0	3/8		
3			0	1/8	
4				0	1/8
5					0

A

	0	1	2	3	4
1	0	1			
2		0	2		
3			0	3	
4				0	4
5					0

R

Example

- $p_1=3/8$, $p_2=3/8$, $p_3=1/8$, $p_4=1/8$

	0	1	2	3	4
1	0	3/8	9/8		
2		0	3/8	5/8	
3			0	1/8	3/8
4				0	1/8
5					0

A

	0	1	2	3	4
1	0	1	1		
2		0	2	2	
3			0	3	3
4				0	4
5					0

R

Example

- $p_1=3/8$, $p_2=3/8$, $p_3=1/8$, $p_4=1/8$

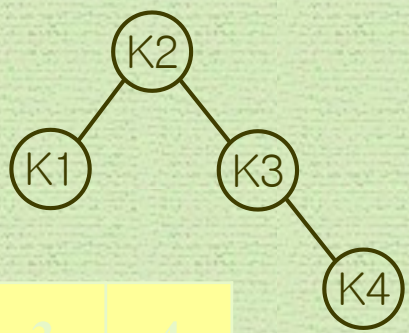
	0	1	2	3	4
1	0	3/8	9/8	11/8	
2		0	3/8	5/8	8/8
3			0	1/8	3/8
4				0	1/8
5					0

A

	0	1	2	3	4
1	0	1	1	2	
2		0	2	2	2
3			0	3	3
4				0	4
5					0

R

Example



- $p_1=3/8$, $p_2=3/8$, $p_3=1/8$, $p_4=1/8$

	0	1	2	3	4
1	0	3/8	9/8	11/8	14/8
2		0	3/8	5/8	8/8
3			0	1/8	3/8
4				0	1/8
5					0

A

	0	1	2	3	4
1	0	1	1	2	2
2		0	2	2	2
3			0	3	3
4				0	4
5					0

R