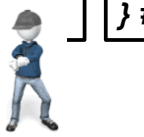
동적 프로그래밍 Part 2

한국기술교육대학교 컴퓨터공학부 김상진

```
while(!morning){
    alcohol++
    dance++
} #party
```

```
while(!sleep){
    think++
    solve++
} #cse-mode
```



교육목표

- 염기 서열 문제
- 최적 이진 검색 트리
- Floyd 최단 경로 문제
- 외판원 문제

Sequence Alignment (1/7)

- Computational Genomics
- 입력. {A, C, G, T}로 구성된 2개의 문자열 X와 Y,
  불일치 패널티 점수 $\alpha_{xy} \geq 0$, gap 패널티 점수 $\alpha_{gap} \geq 0$
  - len(X) = m, len(Y) = n
- 출력. 전체 패널티 점수가 최소화되는 배치
  - 참고. Needleman-Worsch score
- 예) AGGGCT, AGGCA

gap penalty

miss match penalty

A	G	G	G	C	T
A	G	G		C	A

Sequence Alignment (2/7)

- 이 문제는 두 문자열에 gap을 추가하여 두 문자열의 길이를 같도록 만드는 문제임
- 유전자 염기서열의 길이가 500일 때 전수조사해야 하는 수는?
  - 길이가 n인 염기서열과 그보다 작은 길이의 염기서열을 비교하는 경우
    g개의 gap이 필요하면 약 $\binom{n}{g} \approx n^g$정도 정렬하는 경우의 수가 존재함

A	G	C	C	T	A
	A	C	G	C	T

A	G	C	C	T	A
A		C	G	C	T

  - 다음과 같은 정렬은 미포함

A	G	C		C	T	A
A		C	G	C	T	

A		G	C	C	T	A
A	C	G	C		T	

Sequence Alignment (3/7)

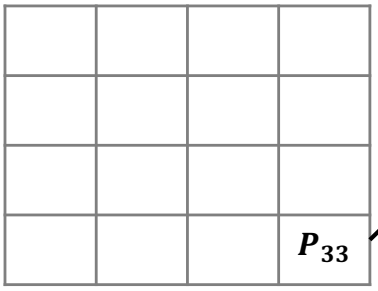
- 소문제를 찾자???
- 최적해의 구조
  - X에 gap 추가한 X^*, Y에 gap 추가한 Y^*: len(X^*) = len(Y^*)
    - 참고. 불필요한 gap의 추가는 무의미함 (gap과 gap 정렬)
- 마지막 위치에 대한 가능한 경우의 수: 3
  - 경우 1. x_m, y_n 정렬
  - 경우 2. x_m과 gap 정렬
  - 경우 3. gap과 y_n 정렬
  - 참고. gap과 gap의 정렬은 무의미함
- $X' = X - x_m$, $Y' = Y - y_n$

Sequence Alignment (4/7)

- 최적해의 특성을 살펴보면…
- 경우 1. 최적해에서 x_m, y_n이 정렬된 경우
  - X'과 Y'의 배치는 최적
  - 증명) 모순에 의한 증명
    - X'과 Y'의 배치의 패널티가 P라 하자.
    - X'과 Y'의 배치가 최적이 아니면 $P^* < P$인 배치가 존재
    - 이 경우 P^*해에 마지막 두 문자를 정렬한 배치를 추가하면 기존 X^*와 Y^*의 배치가 최적이라는 것에 모순
- 경우 2. 최적해에서 x_m과 gap 정렬
  - X'과 Y의 배치는 최적
- 경우 3. 최적해에서 gap과 y_n 정렬
  - X와 Y'의 배치는 최적

Sequence Alignment (5/7)

- 점화식
  - $P_{ij} = min(\alpha_{x_i y_j} + P_{i-1,j-1}, \alpha_{\text{gap}} + P_{i-1,j}, \alpha_{\text{gap}} + P_{i,j-1})$
    - $i = 1, \dots, m, j = 1, \dots, n$
  - base case
    - P_{i0}: $i \times \alpha_{\text{gap}}$
    - P_{0j}: $j \times \alpha_{\text{gap}}$
    - 예) P_{05}: Y 문자열의 다섯 번째까지 gap과 일치시켰다는 것을 의미



$P_{33} = min(\alpha_{x_3 y_3} + P_{22}, \alpha_{\text{gap}} + P_{23}, \alpha_{\text{gap}} + P_{32})$
P_{33}은 $X[3]$과 $Y[3]$ 을 일치시킨 경우만을 나타내는 것이 아님
$X[3]$과 $Y[3]$과 일치시킨 경우
$X[3]$과 공백을 일치시킨 경우
$Y[3]$과 공백을 일치시킨 경우
중 최소 패널티를 나타냄

Sequence Alignment (6/7)

- 알고리즘:

SequenceAlignment(X, Y)

$A \coloneqq [[0] \times (m+1)] \times (n+1)$ // 0 색인
for $i \coloneqq 0$ *to* m *do*
  $A[i][0] \coloneqq i \times \alpha_{gap}$
for $j \coloneqq 0$ *to* n *do*
  $A[0][j] \coloneqq j \times \alpha_{gap}$
for $i \coloneqq 1$ *to* m *do*
  for $j \coloneqq 1$ *to* n *do*
    $A[i][x] \coloneqq min(A[i-1][j-1] + \alpha_{x_i y_j}, A[i-1][j] + \alpha_{\text{gap}}, A[i][j-1] + \alpha_{\text{gap}})$
return $A[m][n]$

$\alpha_{x_i y_j} = x_i == y_j ? 0 : penalty$

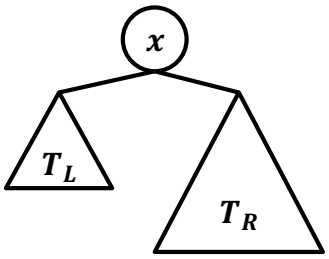
- 시간 복잡도: $O(mn)$

Sequence Alignment (7/7)

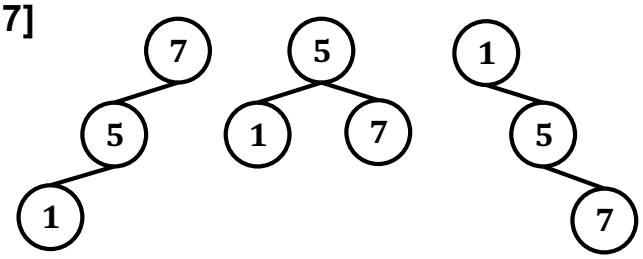
- 해결책은 $A[m][n]$에서 거꾸로
  - $P_{ij} = min(\alpha_{x_i y_j} + P_{i-1,j-1}, \alpha_{\text{gap}} + P_{i-1,j}, \alpha_{\text{gap}} + P_{i,j-1})$
  - $A[i][j]$:
    - $\alpha_{x_i y_j} + A[i-1][j-1], \alpha_{\text{gap}} + A[i-1][j], \alpha_{\text{gap}} + A[i][j-1]$
      3개의 값을 구해 셋 중 가장 작은 값에 따라 이동하면서
      답을 재구성
      - 경우 1. $A[i-1][j-1]$로 이동
      - 경우 2. $A[i-1][j]$로 이동
      - 경우 3. $A[i][j-1]$로 이동
    - $i = 0$ 또는 $j = 0$이 되면 남은 것을 gap과 일치시킴

이진 검색 트리

- 이진 검색 트리
  - $\forall key \in T_L < x < \forall key \in T_R$



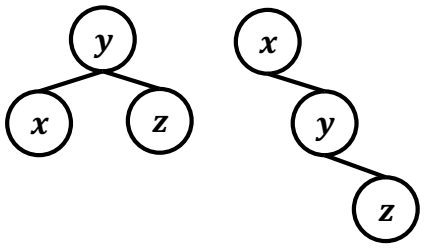
- 주어진 값들이 있을 때, 이 값들을
  이용하여 만들 수 있는 유효한
  이진 트리는 여러 개 존재함
  - 예) [1, 5, 7]



- 이진 검색 트리는 균형 트리일 때 가장 효과적???
  - 균형 트리이면 검색 시간 복잡도는 $O(\log n)$
  - 검색키마다 검색하는 빈도가 다른 경우에는?

검색 빈도의 비균일성

- 예) $x < y < z$를 만족하는 3개의 키로 구성된 이진 검색 트리가 있을 때, x의 검색 빈도는 80%이고, y와 z의 검색 빈도는 각각 10%일 경우, 다음 두 이진 검색 트리에서 평균 검색 비용은?
  - 비용: 이진 검색 트리에서 방문한 노드 수



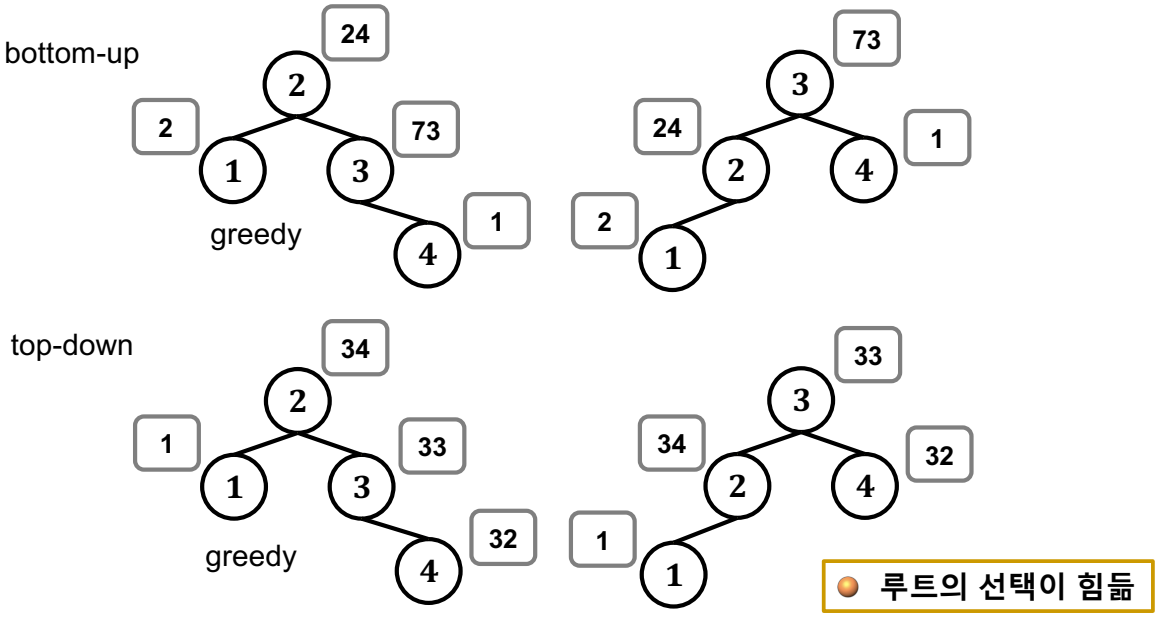
  - $1.9 = 2\times0.8 + 1\times0.1 + 2\times0.1$
  - $1.3 = 1\times0.8 + 2\times0.1 + 3\times0.1$

최적 이진 검색 트리 (1/9)

- 입력. 정렬된 검색키 $k_1, k_2, \ldots, k_n$과 각 키의 검색 빈도 $p_1, p_2, \ldots, p_n$
- 출력. 평균 검색 비용이 최소화되는 이진 검색 트리
  - 검색 비용 $s(i)$: $s(i) = \text{depth}(k_i) + 1$
  - 평균 검색 비용: $C(T) = \sum_{i=1}^{n} p_i \cdot s(i)$
- 균형 트리이면 평균 검색 비용은 $O(\log n)$임
- 문제 특징
  - k_i의 실제 값은 중요하지 않음. 그냥 키 값을 1, 2, …, n으로 가정하여도 문제의 성격이 변하지 않음
  - $\sum_{i=1}^{n} p_i = 1$이 아니어도 됨 ($\sum_{i=1}^{n} p_i \le 1$)
  - 트리에 없는 키의 검색은 고려하지 않음

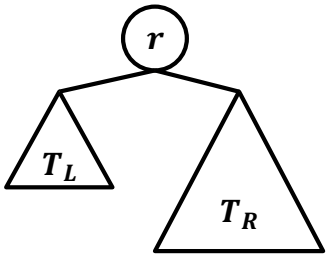
탐욕적 기법이 가능하지 않을까?

- 동기. 가장 빈도가 높은 키들은 루트와 가까워야 함
  - 가장 빈도가 낮은 키들은 루트에서 멀리



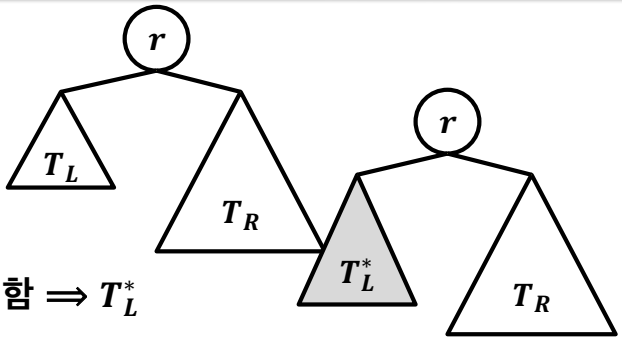
루트의 선택

- top-down 방식에서는 루트의 선택은 매우 힘듦
- What if we knew the root?
  - 동적 프로그래밍을 이용하여 모든 경우를 다 고려???
- 검색키 $1, 2, \ldots, n$으로 구성된 최적 BST의 루트 노드의 키 값이 r이고, 이것의 왼쪽, 오른쪽 부분트리가 각각 T_L과 T_R이라 하자.
  - 이 경우 T_L과 T_R은 각각 $\{1, \ldots, r-1\}$과 $\{r+1, \ldots, n\}$으로 구성하는 값들의 최적의 BST이어야 함
  - 이것이 실제 참일까?
    - 증명해야 함



최적 이진 검색 트리 (4/9)

- 최적해 구조를 증명하자
  - T가 최적해이면 그것의 부분 트리도 최적해임
  - 모순 증명
    - T_L이 최적이 아니라고 가정하자.
      - 그러면 최적인 BST가 존재해야 함 $\Longrightarrow T_L^*$
      - $C(T_L^*) < C(T_L)$
    - $C(T^*) < C(T)$이면 T가 최적해라는 것이 모순
    - $C(T) = \sum_{i=1}^{n} p_i\, s(i) = p_r + \sum_{i=1}^{r-1} p_i\, s(i) + \sum_{i=r+1}^{n} p_i\, s(i)$
      $= \sum_{i=1}^{n} p_i + \sum_{i=1}^{r-1} p_i\, (s(i) \text{ in } T_L) + \sum_{i=r+1}^{n} p_i\, (s(i) \text{ in } T_R)$
      $= \sum_{i=1}^{n} p_i + C(T_L) + C(T_R) = 1 + C(T_L) + C(T_R)$
    - $C(T^*) = 1 + C(T_L^*) + C(T_R)$
    - 따라서 $C(T_L^*) < C(T_L)$이므로 $C(T^*) < C(T)$이므로 모순

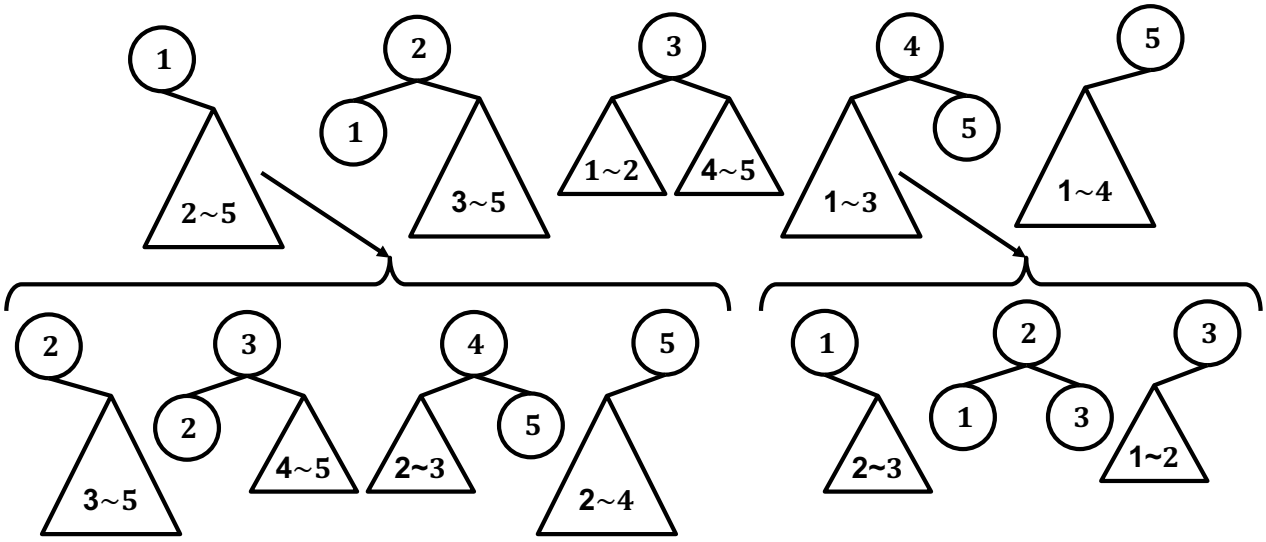


최적 이진 검색 트리 (5/9)

- 최적해 구조
  - 소문제는 몇 개? 아니면 어떤 소문제를 해결해야 하나?
  - 검색키 집합이 $\{1, 2, \ldots, n\}$이면 이 키들이 모두 루트가 될 수 있음
  - 따라서 우리가 구해야 하는 것은
    - 모든 i에 대해 $\{1, \ldots, i-1\}$와 $\{i+1, \ldots, n\}$를 구해야 하는 것으로 생각할 수 있지만…
    - 실제 구해야 하는 것은 모든 $i \le j$에 대해 $\{i, \ldots, j\}$로 구성된 BST의 최적해를 구해야 함
      - Why? 가능한 모든 형태의 트리를 조사해야 함

최적 이진 검색 트리 (6/9)

- 예) [1,2,3,4,5]로 가능한 이진 검색 트리



- 크기가 1인 것: 1, 2, 3, 4, 5
- 크기가 2인 것: 1~2, 2~3, 3~4, 4~5
- 크기가 3인 것: 1~3, 2~4, 3~5
- 크기가 4인 것: 1~4, 2~5
- 크기가 5인 것: 1~5

최적 이진 검색 트리 (7/9)

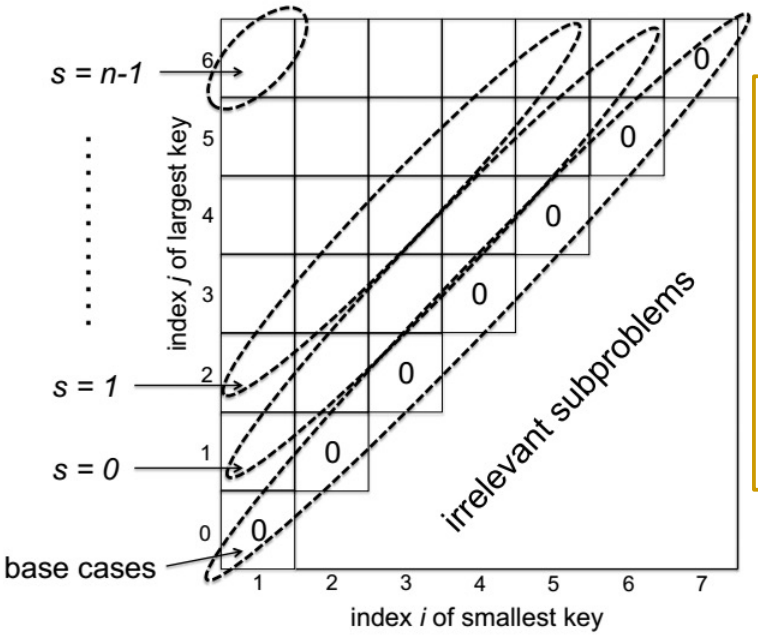
- 점화식
  - C_{ij}: $1 \le i \le j \le n$에 대해 $\{i, i+1, \dots, j-1, j\}$로 구성된 최적 BST의 평균 검색 비용
  - 모든 $1 \le i \le j \le n$에 대해 점화식은 다음과 같음

$$C_{ij} = \min_{r=i,\dots,j} \left\{ \sum_{k=i}^{j} p_k + C_{i,r-1} + C_{r+1,j} \right\}$$

$$= \sum_{k=i}^{j} p_k + \min_{r=i,\dots,j} \{ C_{i,r-1} + C_{r+1,j} \}$$

    - 총 경우의 수: $j - i + 1$ (가능한 루트)
      - 이들에 대해 전수조사 방법으로 최솟값을 구함
  - Edge case: 첫 번째 요소가 루트 또는 마지막 요소가 루트인 경우
    - $i > j$이면 $C_{ij} = 0$

최적 이진 검색 트리 (8/9)



$n = 4$인 경우 계산되는 순서
C[1][1]
C[2][2]
C[3][3]
C[4][4]
C[1][2]
C[2][3]
C[3][4]
C[1][3]
C[2][4]
C[1][4]

최적 이진 검색 트리 (9/9)

- 알고리즘

왜 $n+2$?
왼쪽 부분트리가 없는 경우
오른쪽 부분트리가 없는 경우

optimalBST()

$C := [[0]\times(n+2)] \times(n+2)$
for $s := 0$ *to* $n-1$ *do* // 작은 크기의 문제 먼저 (1 ~ n)
  for $i := 1$ *to* $n-s$ *do*
    $C[i][i+s] := \sum_{k=i}^{i+s} p_k + \min_{r=i,\ldots,i+s}\{C[i][r-1] + C[r+1][i+s]\}$
return $C[1][n]$

- 시간 복잡도
  - 소문제의 수
  - 각 소문제를 해결하는 비용
  - 전체: $O(n^3)$
- 참고. 최적화하면 $O(n^2)$에 할 수 있음

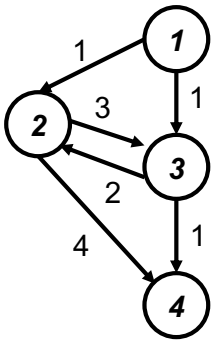
$r = i$: $C[i][i-1] + C[i+1][i+s]$
$r = i+s$: $C[i][i+s-1] + C[i+s+1][i+s]$

$C_{i,i-1} + C_{i+1,j}$
...
$C_{i,j-1} + C_{j+1,j}$

때문에 $(n+2)\times(n+2)$ 배열

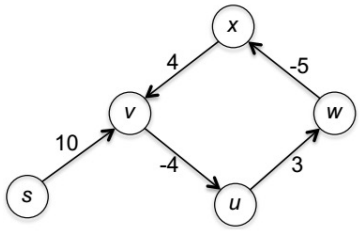
최적화 문제

- 최적화 문제(optimization problem)
  - 가능한 모든 해로부터 가장 최적의 해를 찾는 문제
  - 최적 원칙(principle of optimality)을 만족하는 최적화 문제는 동적 프로그래밍으로 해결할 수 있음
- 최적 원칙: 어떤 문제의 사례에 대한 최적해가 그 사례의 부분 사례의 최적해를 항상 포함하고 있으면 이 원칙을 만족하는 문제임
  - 예) 최단 경로 문제: 최적 원칙 적용 가능 문제
  - 예) 최장 경로 문제: 최적 원칙 적용이 가능하지 않은 문제
    - 노드 1에서 4까지의 최장 경로
      - [1, 3, 2, 4] = 7
    - 노드 1에서 3까지의 최장 경로
      - [1, 2, 3] = 4



최단 경로 찾기 문제 (1/3)

- 최단 경로 찾기 문제
  - 가중치 방향 그래프에서 한 노드에서 다른 노드로 가는 최단 경로 찾기
    - 경로가 여러 개 존재할 수 있음
  - 어떻게 최단 경로를 찾나?
    - 가능한 모든 경로를 찾은 후 그 중에서 최단 경로를 찾음
      - 완전 그래프이면 한 노드에서 다른 모든 노드를 한번 지나 목적 노드로 가는 경로만 총 $(n-2)!$개 존재함
        - 지수 시간보다 더 많은 경우임
- 모든 가중치가 양수이면 다익스트라 알고리즘으로 찾을 수 있음
  - 많은 경우가 여기에 해당함
- 음수 가중치가 존재하면 Bellman-Ford 또는 Floyd-Warshall 알고리즘으로 찾을 수 있음
  - 단, 음의 주기가 없어야 함



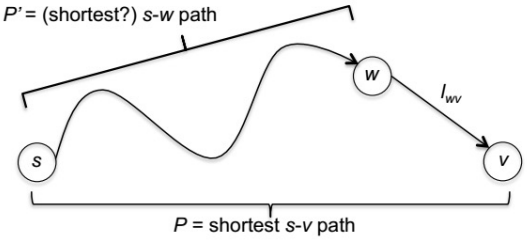
최단 경로 찾기 문제 (2/3)

- 입력. 가중치 방향 그래프 $G = (V, E)$, 출발 노드 $s \in V$
  - 가중치 e에 대한 제한 없음 (음수가 가능하다는 것)
- 출력.
  - 선택 1. 모든 $v \in V$에 대해 s에서 v까지 최단 경로 $dist(s, v)$
  - 선택 2. G는 음의 주기 존재
- 참고.
  - 음의 주기를 허용하면 최단 경로를 구할 수 없음
    (알고리즘이 종료하지 않음)
  - 음의 주기를 포함하지 않는 경로만 고려하는 것은 더 어려운 문제임

최단 경로 찾기 문제 (3/3)

- 어떻게? 경로의 간선 수가 최대 $n - 1$인 것 까지만 고려함
  - 경로의 간선 수가 $n - 1$보다 큰 경로는 길이가 더 짧고 간선 수도 $n - 1$ 이하인 경로로 바꿀 수 있음
    - 경로의 간선 수가 $n - 1$보다 크다는 것은 한 노드를 두 번 이상 방문한다는 것을 의미함
    - 경로에 주기(음의 주기가 아님)가 존재한다는 것임
    - 이 주기를 제거하면 더 짧은 경로를 확보할 수 있음
- Bellman-Ford 알고리즘
  - R. Bellman 1958, L. R. Ford 1956년
  - 음의 주기가 없는 음의 가중치 방향 그래프에서 특정 출발노드부터 다른 모든 노드까지의 최단 경로를 찾아 줌
  - 시간 복잡도: $O(mn)$

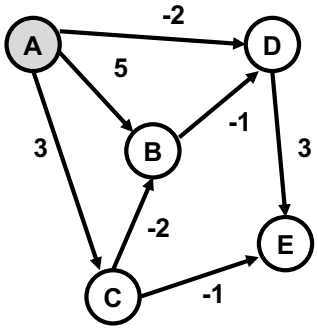
Bellman-Ford 알고리즘 (1/2)



- 최단 경로: 최적 원리 적용
  - P'의 특성
    - P보다 길이가 짧음 (NO)
    - P보다 간선 수가 적음 (YES)
  - 경로를 구성하는 간선의 수를 늘리면서 최적 경로를 찾음
- Optimal Substructure
  - P: 최대 i개 간선을 이용하는 s에서 v로 가는 최단 경로
  - $\#e(P) \leq i-1$: P는 최대 $i-1$개 간선을 이용할 수 있는 문제의 최적해
  - $\#e(P) = i$
    - (w, v): P의 최종 간선, P': s에서 w로 가는 P의 부분 경로
      - P'은 간선 $i-1$개로 구성된 s에서 w로 가는 최단 경로
    - 모순으로 증명 가능
- 점화식: 간선 i개 이하로 구성된 s에서 v로 가는 최단 경로

$$L_{i,v} = \min\left\{L_{i-1,v}, \min_{(w,v)\in E}\left(L_{i-1,w} + e_{w,v}\right)\right\}$$

Bellman-Ford 알고리즘 (2/2)



	A	B	C	D	E
A	0	5	3	-2	∞
B	∞	0	∞	-1	∞
C	∞	-2	0	∞	-1
D	∞	∞	∞	0	3
E	∞	∞	∞	∞	0

$$L_{i,v} = \min\left\{L_{i-1,v}, \min_{(w,v)\in E}\left(L_{i-1,w} + e_{w,v}\right)\right\}$$

Dist¹

A	B	C	D	E
0	5	3	-2	∞

Dist²

A	B	C	D	E
0	1	3	-2	1

Dist³

A	B	C	D	E
0	1	3	-2	1

Dist²[B]
$A \to A \to B = 5$
$A \to C \to B = 1$

- 존재하지 않는 것도 고려하고 있음
- 진입 간선만 고려함. $O(m)$

- 이처럼 모든 v에 대해 $L_{i,v} = L_{i-1,v}$이면 더 이상 변하지 않으므로 반복을 여기서 중단할 수 있음
- 모든 v에 대해 $L_{n,v} = L_{n-1,v}$가 성립하지 않으면 음의 주기가 존재함

Floyd-Warshall 알고리즘 (1/7)

- Floyd-Warshall 알고리즘
  - 보통 그냥 Floyd 알고리즘이라 함 (1962년)
    - B. Roy 1959년, S. Warshall 1962년
  - 음의 주기가 없는 음의 가중치 방향 그래프에서 모든 쌍 간의 최단 경로를 찾아 줌
  - 시간 복잡도: $O(n^3)$
    - Bellman-Ford 알고리즘을 n번 호출하는 것보다 효율적임. $O(mn^2)$
- Bellman-Form 알고리즘과 차이점
  - 모든 노드 쌍 간의 최단 경로를 구해 줌
  - 간선 수가 소문제의 기준이 아니라 경로를 구성하는 노드가 소문제의 기준

Floyd-Warshall 알고리즘 (2/7)

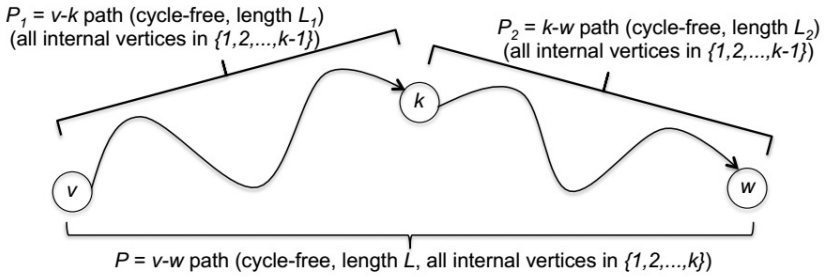
- $D^{(k)}[i][j]$: 노드 1부터 k까지만 중간 노드로 사용하는 i에서 j까지 주기가 없는 최단 경로
  - 총 문제의 수: $O(n^3)$
- Optimal substructure
  - P는 1부터 k까지 노드만 사용하는 i에서 j까지 최단 경로
  - 경우 1. k가 P에 포함되지 않은 경우
    - P는 1부터 $k-1$까지 노드만 사용하는 i에서 j까지 최단 경로와 같음
  - 경우 2. k가 P에 포함된 경우
    - P를 i에서 k까지 최단 경로 P_1과 k에서 j까지 최단 경로 P_2로 나눌 수 있고, 이들은 모두 1부터 $k-1$까지 노드만 사용

1부터 $k-1$까지 노드만 사용하는 i에서 j까지 최단 경로	1부터 $k-1$까지 노드만 사용하는 i에서 k까지 최단 경로	1부터 $k-1$까지 노드만 사용하는 k에서 j까지 최단 경로

Floyd-Warshall 알고리즘 (3/7)

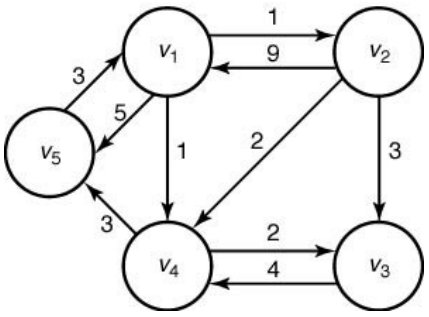
- Bellman-Ford(간선의 수가 기준)와 달리 주기가 없어야 하는 조건이 추가됨
  - P_1과 P_2에 주기가 없지만 그것을 결합하면 주기가 나타날 수 있음
    - 예) 1→2→5, 5 →3→2→4
  - 음의 주기가 있으면 P_1과 P_2가 최적이 아닐 때, 이들의 결합이 최적이 될 수 있음. 음의 주기가 있으면 최적 원칙이 만족하지 않음
- 이 논리(경로를 구성하는 내부 노드 기준)는 왜 Bellman-Ford(특정 노드에서 출발하는 경로만 고려)에 적용하지 못할까?
  - P_2는 문제에서 고려하는 출발노드에서 시작하는 경로가 아님
- 점화식

$$D^{(k)}[i][j] = min(D^{(k-1)}[i][j],\ D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$$



Floyd-Warshall 알고리즘 (4/7)

- 가중치 그래프를 인접 행렬로 표현



	A	B	C	D	E
A	0	1	∞	1	5
B	9	0	3	2	∞
C	∞	∞	0	4	∞
D	∞	∞	2	0	3
E	3	∞	∞	∞	0

- $D^{(0)}[2][5] = \infty$
- $D^{(1)}[2][5] = min(\infty, D^{(0)}[2][1] + D^{(0)}[1][5]) = min(\infty, 9+5) = 14$
- $D^{(2)}[2][5] = min(14, D^{(1)}[2][2] + D^{(1)}[2][5]) = min(14, 14) = 14$
- $D^{(3)}[2][5] = min(14,\ D^{(2)}[2][3] + D^{(2)}[3][5])$
- $D^{(4)}[2][5] = min(D^{(3)}[2][5],\ D^{(3)}[2][4] + D^{(3)}[4][5])$
- $D^{(5)}[2][5] = min(D^{(4)}[2][5],\ D^{(4)}[2][5] + D^{(4)}[5][5]) = D^{(4)}[2][5]$

$D^{(0)}$	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

$D^{(2)}$	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

$D^{(4)}$	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

$D^{(4)}[i][j] = min(D^{(3)}[i][j], D^{(3)}[i][4] + D^{(3)}[4][j])$

- 현재 셀 계산을 위해 이전 셀 값 필요 ⟹ 조회 후 수정하기 때문에 문제 없음
- $D^{(k)}$ 계산에서는 $D^{(k-1)}[i][k]$와 $D^{(k-1)}[k][j]$ 필요, 하지만 $D^{(k)}[i][k] = D^{(k-1)}[i][k]$ 이고 $D^{(k)}[k][j] = D^{(k-1)}[k][j]$임 (회색 부분은 변하지 않음)
- 따라서 이전 2차원 배열을 이용하여 그대로 다음 2차원 배열을 계산할 수 있음

$D^{(4)}[i][4] = min(D^{(3)}[i][4], D^{(3)}[i][4] + D^{(3)}[4][4] = 0)$
$D^{(4)}[4][i] = min(D^{(3)}[4][i], D^{(3)}[4][i] + D^{(3)}[i][i] = 0)$

Floyd-Warshall 알고리즘 (6/7)

- 알고리즘

```
floyd(W)
D := W
for k := 1 to n do
    for i := 1 to n do
        for j := 1 to n do
            D[i][j] := min(D[i][j], D[i][k] + D[k][j])
for i := 1 to n do
    if D[i][j] < 0 then return negative cycle
return D
```

  - 시간 복잡도: $O(n^3)$
- 음의 주기?
  - 어떤 i와 j에 대해 $D[i][j] + D[j][i] < 0$이면 음의 주기 존재

Floyd-Warshall 알고리즘 (7/7)

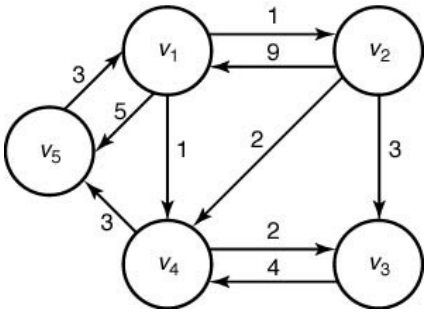
- 경로 찾기

$D^{(5)}$	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

$$D^{(1)}[2][5] = min\left(D^{(0)}[2][5], D^{(0)}[2][1] + D^{(0)}[1][5]\right)$$
$$D^{(2)}[2][5] = min\left(D^{(1)}[2][5], D^{(1)}[2][2] + D^{(1)}[2][5]\right)$$
$$D^{(3)}[2][5] = min\left(D^{(2)}[2][5], D^{(2)}[2][3] + D^{(2)}[3][5]\right)$$
$$D^{(4)}[2][5] = min\left(D^{(3)}[2][5], D^{(3)}[2][4] + D^{(3)}[4][5]\right)$$
$$D^{(5)}[2][5] = min\left(D^{(4)}[2][5], D^{(4)}[2][5] + D^{(4)}[5][5]\right)$$

$$D[2][5] = min\left(D^{(k)}[2][5], D[2][k] + D[k][5]\right)$$

- k를 찾아 재귀적으로 재구성 가능
- 위 테이블을 구축하면서 같은 배열에 k값을 기록하는 방법도 가능

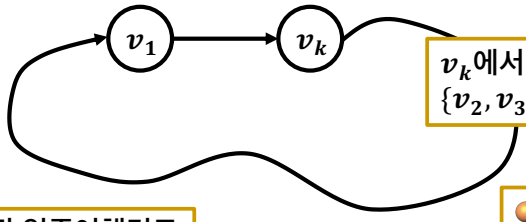


	1	2	3	4	5
1	0	0	4	0	4
2	5	0	0	0	4
3	5	5	0	0	4
4	5	5	0	0	0
5	0	1	4	1	0

외판원 문제 (1/5)

완전 그래프일 때 일주여행경로 수: $(n-1)!/2$

- **외판원 문제**(traveling salesperson problem): 출발노드에서 다른 모든 노드를 한 번씩 방문하고 다시 출발 노드로 돌아오는 최단 경로를 찾는 문제
  - 최적 **해밀톤 경로**(hamiltonian circuit), 최적 일주여행경로(tour, **투어**)
  - 보통 완전 그래프를 가정함. 시작 노드가 중요하지 않음
- 최적화 문제인가?
  - v_1을 출발점으로 하는 최적 투어를 고려하여 보자.
    - v_k가 최적 투어에서 v_1 다음에 오는 첫 번째 노드이면 v_k에서 v_1로 가는 투어의 부분 경로는 다른 노드를 정확하게 한 번씩만 거치며 v_k에서 v_1로 가는 최단 경로임



v_k에서 v_1로 가는 경로 P(최적 투어의 부분경로):
$\{v_2, v_3, \ldots, v_n\}$ 노드를 정확하게 한번씩만 거치는 최단 경로임

최적 일주여행경로

- 최단 경로는 최적 원칙을 적용할 수 있음
- 기존 최단 경로와 달리 특정 노드를 반드시 포함해야 함

외판원 문제 (2/5)

- v_1을 출발 노드로 가능한 모든 투어를 고려함
  - 다음 노드는 v_2부터 v_n까지 가능함
    - v_2부터 v_1까지 최단 경로, v_3부터 v_1까지, ..., v_n부터 v_1까지 최단 경로를 구하여 $d[1][j] + D[v_j][V - \{v_1, v_j\}]$ 중 최소가 되는 것이 해
    - $D[v_k][A]$: A에 속한 노드를 한번씩 거치며 v_k에서 v_1로 가는 최단 경로의 길이
  - v_2부터 v_1까지 최단 경로:

$$\min_{j=V-\{v_1,v_2\}} d[2][j] + D[v_j][V - \{v_1, v_2, v_j\}]$$

  - 다음 단계에서는 지나가는 노드 수가 하나씩 줄어듬
- 동적 프로그래밍이므로 하향식이 아니라 상향식. 예) 노드 4개
  - $(v_2 \rightarrow v_1)$, $(v_3 \rightarrow v_1)$, $(v_4 \rightarrow v_1)$
  - $(v_2 \rightarrow v_3 \rightarrow v_1)$, $(v_2 \rightarrow v_4 \rightarrow v_1)$, ..., $(v_4 \rightarrow v_3 \rightarrow v_1)$
  - $(v_2 \rightarrow v_3, v_4 \rightarrow v_1)$, $(v_3 \rightarrow v_2, v_4 \rightarrow v_1)$, $(v_4 \rightarrow v_2, v_3 \rightarrow v_1)$

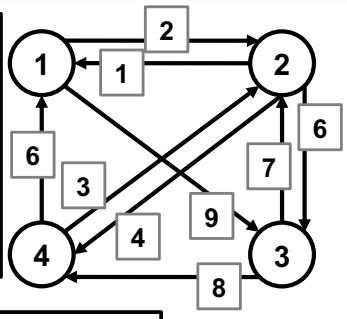
$(v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_1)$, $(v_2 \rightarrow v_4 \rightarrow v_3 \rightarrow v_1)$,

외판원 문제 (3/5)

$D[2][\emptyset] = 1$
$D[3][\emptyset] = \infty$
$D[4][\emptyset] = 6$

$D[2][\{3\}] = \min_{j\in\{3\}}(d[2][3] + D[3][\{3\} - \{3\}]) = \infty$
$D[2][\{4\}] = \min_{j\in\{4\}}(d[2][4] + D[4][\emptyset]) = 10$
$D[3][\{2\}] = 8$
$D[3][\{4\}] = 14$
$D[4][\{2\}] = 4$
$D[4][\{3\}] = \infty$

$\min_{v_j \in A}(d[k][j] + D[v_j][A - \{v_j\}])$



$D[2][\{3,4\}] = \min_{j\in\{3,4\}}(d[2][j] + D[j][\{3,4\} - \{j\}])$
$= \min(d[2][3] + D[3][\{4\}], d[2][4] + D[4][\{3\}]) = \min(6 + 14, 4 + \infty) = 20$
$D[3][\{2,4\}] = \min_{j\in\{2,4\}}(d[3][j] + D[j][\{2,4\} - \{j\}])$
$= \min(d[3][2] + D[2][\{4\}], d[3][4] + D[4][\{2\}]) = \min(7 + 10, 8 + 4) = 12$
$D[4][\{2,3\}] = \min_{j\in\{2,3\}}(d[4][j] + D[j][\{2,3\} - \{j\}])$
$= \min(d[4][2] + D[2][\{3\}], d[4][3] + D[3][\{2\}]) = \min(3 + \infty, \infty + 8) = \infty$

$D[1][\{2,3,4\}] = \min_{j\in\{2,3,4\}}(d[1][j] + D[j][\{2,3,4\} - \{j\}])$
$= \min(d[1][2] + D[2][\{3,4\}], d[1][3] + D[3][\{2,4\}], d[1][4] + D[4][\{2,3\}])$
$= \min(2 + 20, 9 + 12, \infty + \infty) = 21$

외판원 문제 (4/5)

- 알고리즘

```
tsp(G[][])
D := [[0]×n] ×2^(n−1)
for i := 2 to n do
    D[i][∅] := G[i][1]
for k := 1 to n − 2 do
    for S: ∀S ⊂ V − {1} and |S| = k do
        for i such that i ≠ 1 and i ∉ S do
            D[i][S] := min_{j∈S}(G[i][j] + D[j][S − {j}])
            P[i][S] := 최솟값 j
D[1][V − {1}] := min_{j∈V−{1}}(G[1][j] + D[j][V − {1, j}])
P[1][V − {1}] := 최솟값 j
return D[1][V − {1}]
```

- 가능한 부분집합의 수: 2^{n-1}
- 1<<(n-1)

- 전수조사 방법을 사용하여 모든 부분 집합을 만들어도 색인 작업을 어떻게?

- v_1을 출발점으로 하는 최적 일주여행경로 길이는 다음과 같음
  $$\min_{2 \le k \le n} (d[1][k] + D[v_k][V - \{v_1, v_k\}])$$
- base case: $D[v_k][\emptyset] = d[k][1]$

외판원 문제 (5/5)

- 시간복잡도: $O(n^2 2^n)$
  - 전수조사: $\Theta(n!)$
    - $n = 12, 13$이면 전수 조사도 가능
    - 동적 프로그래밍은 $n = 30$ 정도까지도 가능