알고리즘및실습

제11장 동적 프로그래밍 2부

1. 염기 서열 문제

주어진 두 개의 유전자 염기 서열이 얼마나 유사한지 검사할 때 사용하는 다음 문제를 살펴본다.

유전자 염기 서열 유사성 문제

- 입력. A, C, G, T 4개 문자로 구성된 두 개의 문자열 X와 Y, 불일치 패널티 점수 $\alpha_{xy} \geq 0$, 공백 패널티 점수 $\alpha_{\text{gap}} \geq 0$
- 출력. 전체 패널티 점수가 최소화되는 배치

이 문제에서 문자열 X의 길이는 m, Y의 길이는 n이라 가정한다. 이 문제는 두 문자열에 공백 문자를 추가하여 두 문자열의 길이를 같도록 만드는 문제이다. 이 문제를 동적 프로그래밍으로 해결하기 위해 점화식을 찾아보자. 최적해의 구조로부터 점화식을 찾아보자. 두 문자열에 공백 문자를 추가하여 같은 길이로 만들었을 때 두 문자열의 마지막 문자를 생각하여 보자. 다음 3가지 경우가 존재한다.

- 경우 1. x_m과 y_n을 정렬한 경우
- 경우 2. x_m과 공백 문자를 정렬한 경우
- 경우 3. 공백 문자와 y_n을 정렬한 경우

참고로 공백 문자와 공백 문자를 정렬하는 것은 무의미하다. 패널티 점수만 증가하기 때문에 최소화되는 배치를 찾는데 이와 같이 정렬하는 것은 아무런 의미가 없다.

최적해는 위 3가지 경우 중 하나이다. 또 그 최적해에서 마지막 정렬된 두 문자를 제외한 배치는 최적이다. 그 배치의 패널티보다 더 최적인 배치가 있다면 모순이 되기 때문이다. 따라서 P_{ij}를 X의 i번째 문자까지의 문자열과 Y의 j번째 문자까지의 문자열을 배치하였을 때 최소 패널티 점수라 하면 이 값은 다음과 같다.

$$P_{ij} = \min(\alpha_{x_i y_j} + P_{i-1,j-1}, \alpha_{\text{gap}} + P_{i-1,j}, \alpha_{\text{gap}} + P_{i,j-1})$$

i의 범위는 1부터 m이고, j의 범위는 1부터 n이다. 예를 들어 P_{33}은 x_3과 y_3을 일치시킨 경우만을 나타내는 것이 아니라 앞서 제시한 3가지 경우 중 가장 최소 패널티가 되는 점수를 나타낸다. 이 식에서 P_{i0}은 $i \times \alpha_{\text{gap}}$이고, $P_{0j} = j \times \alpha_{\text{gap}}$이다. 예를 들어 P_{05}는 Y의 다섯번째까지 공백문자와 일치시켰다는 것을 의미한다. 이 점화식을 바탕으로 테뷸레이션을 이용한 알고리즘은 알고리즘 11.1과 같다. 이 알고리즘의 시간 복잡도는 $O(mn)$이다.

알고리즘 11.1 염기 서열 배치 알고리즘

```
1: function SEQUENCEALIGNMENT(X, Y, misP, gapP)
2:     table := [[0] × (m + 1)] × (n + 1)                    ▷ 0색인 가정
3:     for i := 1 to m do
4:         table[i][0] := i × gapP
5:     for j := 1 to n do
6:         table[0][j] := j × gapP
7:     for i := 1 to m do
8:         for j := 1 to n do
9:             p := 0 if X[i] = Y[j] else misP
10:            table[i][x] := MIN(p + table[i − 1][j − 1], gapP + table[i − 1][j], gapP + table[i][j − 1])
11:    return table[m][n]
```

2. 최적의 이진 검색 트리

이진 검색 트리(BST, Binary Search Tree)는 검색할 키 값을 노드에 유지하며, 어떤 노드가 주어졌을 때 이 노드에 유지하고 있는 키 값은 그것의 왼쪽 부분 트리에 있는 모든 노드에 유지하고 있는 키 값보다 크며, 그것의 오른쪽 부분 트리에 있는 모든 노드에 유지하고 있는 키 값보다 작은 이진 트리이다. 이진 검색 트리는 키 값을 검색하는 것이 목적이므로 중복된 키 값을 허용하지 않는다. 이전 검색 트리는 키 값을 삽입하는 순서에 의해 트리 모습이 결정된다. 따라서 어떤 값들이 주어지면 이 값들을 이용하여 만들 수 있는 유효한 이진 트리는 여러 개가 존재한다.

이진 검색 트리에서 검색 비용은 트리 높이에 의해 결정된다. 따라서 최악의 경우 비용을 줄이기 위해서는 트리의 높이가 작을수록 효과적이다. 하지만 삽입과 삭제가 빈번하게 일어나는 경우 트리의 높이를 항상 최적으로 유지하기 힘들기 때문에 트리의 균형을 항상 자동으로 유지해 주는 AVL, red-black 트리와 같은 균형 이진 검색 트리를 보통 사용한다. 그런데 검색키마다 검색하는 빈도가 다르고, 삽입과 삭제가 빈번하게 일어나는 환경이 아니면 검색 빈도가 높은 키 값이 루트에 가까울수록 검색 비용을 줄일 수 있다.

최적 이진 검색 트리 문제는 검색 키와 각 검색 키의 검색 빈도가 주어졌을 때 평균 검색 비용을 최소화하는 최적의 이진 검색 트리를 만드는 문제이다.

최적 이진 검색 트리 문제

- 입력. 정렬된 n개의 검색 키와 각 키의 검색 빈도 p_i
- 출력. 평균 검색 비용이 최소화되는 이진 검색 트리

여기서 k_i의 검색 비용 s_i는 k_i의 깊이가 $d(k_i)$일 때 $s_i = d(k_i) + 1$이며, 평균 검색 비용은 $C(T) = \sum_{i=1}^{n} p_i \times s_i$이다. 이 문제에서 실제 k_i의 값은 중요하지 않다. 이 값들이 정수 1부터 n이라고 가정하여도 문제의 성격이 변하지 않는다. 또 $\sum_{i=1}^{n} p_i = 1$이 아니어도 된다. 물론 $\sum_{i=1}^{n} p_i \leq 1$은 성립해야 한다. 또 이 문제에서는 트리에 없는 키의 검색은 고려하지 않는다.

이 문제는 탐욕적 기법을 이용하여 해결할 수 없다. 가장 빈도가 높은 키가 루트와 가까워야 하고, 가장 빈도가 낮은 키는 루트에서 멀어야 하지만 가장 빈도가 높은 것이 항상 루트가 되어야 하는 것은 아니다. 가장 빈도가 높은 것 중에 하나가 루트가 되겠지만 어느 것이 실제 루트가 될지는 단정할 수 없다. 이 때문에 모든 키 값이 루트가 될 수 있다고 가정하고 모든 경우를 전수 조사하는 형태로 문제를 해결할 수 있다. k_r이 최적 BST의 루트라고 가정하면 k_1부터 k_{r-1}로 구성된 루트의 왼쪽 부분 트리와 k_{r+1}부터 k_n으로 구성된 오른쪽 부분 트리는 이들 값으로 구성된 최적 BST이다. 이것이 실제 참일까?

정리 11.1. 오름차순으로 정렬된 n개의 검색 키와 각 키의 검색 빈도가 주어졌을 때, k_r이 루트가 되는 BST T가 평균 검색 비용이 최소화되는 최적 BST라 하자. 그러면 k_1부터 k_{r-1}로 구성된 루트의 왼쪽 부분 트리 T_L과 k_{r+1}부터 k_n으로 구성된 오른쪽 부분 트리 T_R은 해당 검색 키로 구성된 최적 BST이다.

증명 모순을 이용하여 증명한다. T_L이 최적 BST가 아니라 가정하자. 그러면 $C(T_L^*) < C(T_L)$인 또 다른 최적 BST가 T_L^*이 존재해야 한다. 그런데 다음이 성립한다.

$$\begin{aligned} C(T) &= \sum_{i=1}^{n} p_i s_i = p_r + \sum_{i=1}^{r-1} p_i s_i + \sum_{i=r+1}^{n} p_i s_i = \sum_{i=1}^{n} p_i + \sum_{i=1}^{r-1} p_i (s_i - 1) + \sum_{i=r+1}^{n} p_i (s_i - 1) \\ &= \sum_{i=1}^{n} p_i + C(T_L) + C(T_R) = 1 + C(T_L) + C(T_R) \end{aligned}$$

T_L를 T_L^*로 바꾼 T^*를 생각하여 보자. 그러면 위와 동일한 방법을 통해 $C(T^*) = 1 + C(T_L^*) + C(T_R)$를 얻을 수 있다. $C(T_L^*) < C(T_L)$이기 때문에 $C(T^*) < C(T)$가 성립한다. 이것이 $C(T)$가 최적 BST라는 것에 모순이 되므로 T_L, T_R은 모두 최적 BST이어야 한다. □

위 증명을 통해 소문제를 해결하고, 이 소문제를 결합하여 더 큰 문제를 해결할 수 있을 것 같다. n개 키가 모두 루트가 될 수 있다. 이때 k_i가 루트가 되면 그것의 왼쪽 부분 트리 T_L은 k_1부터 k_{i-1}로 구성된 부분 트리이고, 그것의 오른쪽 부분 트리 T_R은 k_{i+1}부터 n로 구성된 부분 트리가 된다. 따라서 k_1부터 k_{n-1}까지 구성된 최적 BST $n-1$개, k_2부터 k_n까지 구성된 최적 BST $n-1$개, 총 $2n-2$개를 찾으면 문제를 해결할 수 있다고 착각할 수 있다. 하지만 우리가 실제 해결해야 하는 소문제는 모든 $i \le j$에 대해 k_i부터 k_j로 구성된 최적 BST이다. 이것은 k_i가 루트인 최적 BST의 T_L의 루트가 될 수 있는 것은 또 다시 k_1부터 k_{i-1}까지 모두 가능하기 때문이다.

이제 점화식을 정리하여 보자. C_{ij}가 $1 \le i \le j \le n$에 대해 k_i부터 k_j로 구성된 최적 BST의 평균 검색 비용이라 하면 그것의 점화식은 다음과 같다.

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

$i > j$이면 C_{ij}는 존재하지 않는 경우에 해당하며 $C_{ij} = 0$으로 설정하여 처리하면 된다.

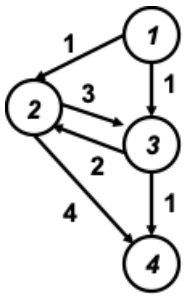
이 문제는 구간의 크기가 1인 것부터 차례로 해결하여 최종적으로 크기가 n인 것을 해결해야 한다. 예를 들어 4개의 키로 구성된 최적 BST 문제를 해결하기 위해서는 먼저 $C_{11}, C_{22}, C_{33}, C_{44}$를 해결하고, 이를 이용하여 크기가 2인 C_{12}, C_{23}, C_{34}를 해결한다. 그다음 크기가 3인 C_{13}, C_{24}를 해결하고, 최종적으로 C_{14}를 해결하여 최종답을 구하게 된다.

알고리즘 11.2 최적 BST 테뷸레이션 알고리즘

```
1:  function OPTIMALBST(p[])
2:      table := [[0] × (n + 2)] × (n + 2)          ▷ 0색인 가정
3:      for s := 0 to n − 1 do                      ▷ 구간의 크기
4:          for i := 1 to n − s do
5:              sumP := 0
6:              min := ∞
7:              for k := i to i + s do
8:                  sumP += p[k]
9:                  min := MIN(min, table[i][k − 1] + table[k + 1][i + s])
10:             table[i][i + s] := sumP + min
11:     return table[1][n]
```

이 알고리즘은 3개의 중첩 반복문으로 구성되어 있으므로 시간 복잡도를 대략적으로 계산하면 $O(n^3)$이다. 하지만 테뷸레이션을 위해 사용하는 테이블을 생각하면 실제 시간 복잡도는 $O(n^2)$임을 알 수 있다.

3. 최적화 문제



<그림 11.1> 최장 경로 문제

최적화 문제(optimization problem)란 가능한 모든 해로부터 가장 최적의 해를 찾는 문제를 말하며, **최적 원칙**(principle of optimality)을 만족하는 최적화 문제는 동적 프로그래밍으로 해결할 수 있다. 최적 원칙을 만족하는 문제이면 문제의 사례에 대한 최적해가 그 사례의 부분 사례의 최적해를 항상 포함하고 있다. 예를 들어 이전 절에서 살펴본 최적 BST 문제는 최적 원칙을 만족하는 최적화 문제이다. 또 그래프에서 최단 경로 문제도 최적 원칙을 만족하는 최적화 문제이다. 하지만 그래프에서 최장 경로 문제는 최적 원칙을 만족하지 못하는 최적화 문제이다. 그림 11.1에 주어진 그래프에서 노드 1에서 4까지의 최장 경로는 (1, 3), (3, 2), (2, 4)이며, 그 길이는 7이다. 하지만 이 경로는 노드 1에서 3까지의 최장 경로를 포함하지 않고 있다. 노드 1에서 3까지의 최장 경로는 (1, 2), (2, 3)이다.

4. 최단 경로 찾기 문제

최단 경로 찾기 문제

- 입력. 가중치 방향 그래프 G, 출발 노드 s, 목적 노드 d
- 출력. 노드 s에서 d까지의 최단 경로

그래프에서 한 노드에서 다른 노드까지 경로는 유일하지 않다. 완전 그래프이면 한 노드에서 다른 모든 노드를 한번 지나 목적 노드로 가는 경로만 총 $(n-2)!$개 존재한다. 따라서 가능한 모든 경로를 찾은 후 그 중에서 최단 경로를 찾는 것은 경우의 수가 너무 많기 때문에 가능하지 않다.

모든 가중치가 양수이면 다익스트라 알고리즘을 이용하여 찾을 수 있다. 실제 많은 응용에서 필요한 최단 경로 찾기 문제는 여기에 해당한다. 하지만 음수 가중치가 존재하면 다익스트라 알고리즘을 이용하여 찾을 수 없다. 음수 가중치가 있는 그래프에서 최단 경로는 Bellman-Ford 알고리즘이나 Floyd-Warshall 알고리즘을 이용할 수 있으며, 이 경우에는 음의 주기가 없어야 한다. Bellman-Ford 알고리즘은 SSSP 알고리즘이며, Floyd-Warshall은 APSP 알고리즘이다. 둘다 동적 프로그래밍을 이용하는 알고리즘이다.

음수 가중치 최단 경로 찾기 문제

- 입력. 가중치에 대한 제한이 없은 가중치 방향 그래프 G, 출발 노드 s
- 출력. 음의 주기가 없으면 노드 s에서 다른 모든 노드까지의 최단 경로, 음의 주기가 있으면 그 사실을 알림

4.1 Bellman-Ford 알고리즘

Bellman-Ford 알고리즘은 경로의 간선 수가 1인 것부터 최대 $n-1$인 것 까지만 고려한다. 경로의 간선 수가 $n-1$보다 큰 경로는 길이가 더 짧고 간선 수도 $n-1$ 이하인 경로로 바꿀 수 있다. 경로의 간선 수가 $n-1$보다 크다는 것은 한 노드를 두 번 이상 방문한다는 것을 의미한다. 한 노드를 두 번 이상 방문한다는 것은 경로에 주기가 포함되어 있다는 것이다. 이 주기는 음의 주기는 아니다. 이 알고리즘은 음의 주기가 있으면 그것을 보고하여 종료하기 때문에 최단 경로를 찾았다는 것은 그래프에 음의 주기가 없다는 것을 의미한다. 따라서 음이 아닌 이 주기를 제거하면 더 짧은 길이의 경로를 확보할 수 있다. Bellman-Ford 알고리즘은 Bellman(1958)과 Ford(1956)가 각각 별도 발견한 알고리즘이다.

최단 경로 문제는 최적 원칙을 만족하는 문제이다. P가 s에서 v까지 가능 최단 경로이고 마지막 간선이 (w,v)이면 이 경로의 부분 경로인 s에서 w까지의 경로 P'에는 어떤 최적 원칙이 적용되는지 생각하여 보자. (w,v)의 가중치가 음수일 수 있으므로 $len(P') < len(P)$은 성립하지 않는다. 하지만 P'의 간선 수는 P보다 하나 적다. P가 s에서 v까지 가는 최단 경로이므로 P는 최대 i개 간선을 이용하는 s에서 v로 가는 최단 경로이다. 참고로 이 문제에서 이용할 수 있는 최대 간선 수는 $n-1$이다. P의 실제 간선 수가 i보다 적다고 하자. 그러면 P는 최대 $i-1$개 간선을 이용하는 s에서 v로 가는 최단 경로이다. 반대로 P가 정확하게 i개 간선으로 구성되어 있다면 P'은 간선 $i-1$개로 구성된 s에서 w로 가는 최단 경로이다. 이와 같은 P와 P'의 특성은 모순을 이용하여 증명할 수 있다.

이 최적의 원칙을 이용하여 점화식을 정의하여 보자. $L_{i,v}$가 최대 i개 간선로 구성된 s에서 v로 가는 최단 경로이면 $L_{i,v}$는 다음과 같이 정의할 수 있다.

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

알고리즘 11.3 Bellman-Ford 알고리즘

```
1:  function BELLMAN_FORD(G, s)
2:      dist := [∞] × n
3:      dist[s] := 0
4:      for all (s, i) ∈ E do
5:          dist[i] := G[s][i]
6:      for i := 2 to n do
7:          found := false
8:          for all v ∈ V − {s} do
9:              d := ∞
10:             for all (w, v) ∈ E do
11:                 d := MIN(d, dist[w] + G[w][v])
12:             if d < dist[v] then
13:                 dist[v] := d
14:                 found := true
15:         if not found then break
16:         else if i = n then return null                ▷ 음의 주기 검사
17:     return dist
```

이 알고리즘의 3중 **for** 문으로 구성되어 있지만 내부 이중 **for** 문은 그래프에 있는 모든 간선을 한번 처리하는 형태이므로 이것의 비용은 $O(m)$이다. 하지만 진출 간선을 고려하는 형태가 아니라 진입 간선을 고려하는 형태이므로 인접 리스트로 그래프를 표현할 경우 역인접 리스트가 필요하다. 따라서 이 문제는 보통 인접 행렬을 사용하게 되며, 인접 행렬을 사용할 경우 구현의 특성 때문에 내부 이중 **for** 문은 $O(n^2)$이 된다. 따라서 전체 시간 복잡도는 $O(n^3)$이다.

countsum 테뷸레이션 알고리즘의 시간 복잡도와 공간 복잡도는 cansum 테뷸래이션 알고리즘과 차이가 없으며, countsum 메모이제이션 알고리즘과도 차이가 없다. 시간 복잡도는 $O(mn)$이고, 공간 복잡도는 $O(m+n)$이다. 물론

재귀 호출 없이 반복문으로 구성되며, 재귀 호출에 의한 함수 스택 공간이 필요 없으므로 실제 성능과 필요한 공간은 메모이제이션보다 우수하다.

이 알고리즘은 모든 v에 대해 $L_{i,v}$가 $L_{i-1,v}$와 같으면 간선의 길이를 하나 더 늘리더라도 최단 경로가 바뀌지 않으므로 중단할 수 있다. 또 음의 주기가 없으면 최단 경로의 최대 간선 수는 $n-1$이지만 경로의 간선 수가 n인 경우까지 고려해야 한다. 이때 모든 v에 대해 $L_{n,v}$가 $L_{n-1,v}$와 같지 않으면 음의 주기가 존재하는 것을 의미한다.

4.2 Floyd-Warshall 알고리즘

Floyd-Warshall 알고리즘은 실제 1959년에 B. Roy가 고안하였고, 1962년에 Floyd와 S. Warshall이 각각 독립적으로 같은 알고리즘을 찾아냈다. 이 알고리즘은 이전 절에서 살펴본 알고리즘과 달리 모든 노드에서 다른 모든 노드까지의 최단 경로를 찾아준다. 이 알고리즘은 Bellman-Ford를 각 노드를 출발 노드로 지정하여 n번 수행하는 것보다 효과적이다. Bellman과 Ford는 경로의 간선 수를 이용하여 소문제를 정의하였지만 Floyd-Warshall은 경로를 구성하는 노드를 이용하여 소문제를 정의한다.

$D_{i,j}^{(k)}$가 노드 1부터 k까지만 중간 노드로 사용하는 i에서 j까지 주기가 없는 최단 경로라 하자. 그러면 이것이 어떤 최적 원칙을 만족하는지 살펴보자. P가 노드 1부터 k까지만 중간 노드로 사용하는 i에서 j까지 최단 경로일 때, 이 경로에 k가 포함될 수 있고, 포함되지 않을 수 있다. 포함되지 않으면 P는 노드 1부터 $k-1$까지만 중간 노드로 사용하는 i에서 j까지 최단 경로와 같다. 반대로 포함하면 P를 i에서 k까지 1부터 $k-1$까지 노드만 사용하는 최단 경로 P_1과 k에서 j까지 1부터 $k-1$까지 노드만 사용하는 최단 경로 P_2로 나눌 수 있다.

Floyd-Warshall 알고리즘이 올바르게 동작하기 위해서는 P가 k를 지나는 노드 1부터 k까지만 중간 노드로 사용하는 i에서 j까지 최단 경로일 때, 중간 노드 k를 기준으로 P_1과 P_2로 나눌 수 있으며, 이때 P_1과 P_2는 각각 최적이어야 한다. 이것은 모순을 이용하여 쉽게 증명할 수 있다. 그런데 P가 음의 주기가 있으면 이 최적 원칙이 보장되지 않는다. 그 이유는 P_1과 P_2가 주기가 없더라도 그것의 결합으로 주기가 만들어질 수 있기 때문이다. 예를 들어 P_1이 (1, 2), (2, 5)이고, P_2가 $(5,3),(3,2),(2,4)$이면 각각은 주기가 없지면 결합하면 주기가 만들어진다.

Bellman-Ford 방식에는 Floyd-Warshall이 사용한 논리를 적용할 수 없다. Bellman-Ford는 항상 출발 노드를 기준으로 알고리즘을 수행하고 있으며, 중간 노드를 기준으로 P를 P_1과 P_2로 나누었을 때 P_2는 P_1과 출발 노드가 달라진다.

Floyd-Warshall이 사용하는 경로를 구성하는 노드를 이용한 점화식은 다음과 같다.

$$D_{i,j}^{(k)} = \min\left(D_{i,j}^{(k-1)}, D_{i,k}^{(k-1)} + D_{k,j}^{(k-1)}\right)$$

여기서 $D_{i,j}^{(0)}$은 그래프 G의 인접 행렬이다.

따라서 그래프의 인접 행렬에서 시작하여 이 인접 행렬을 계속 갱신하면서 알고리즘을 진행할 수 있다. 이때 새 2차원 배열의 생성이 필요할 것으로 생각할 수 있지만 새 2차원 배열을 만들 필요 없이 주어진 인접 행렬만 이용하여 알고리즘을 진행할 수 있다.

다음을 계산할 차례라고 생각하여 보자

$$D_{i,j}^{(4)} = \min\left(D_{i,j}^{(3)}, D_{i,4}^{(3)} + D_{4,j}^{(3)}\right)$$

$D_{i,j}^{(4)}$ 값을 구하기 위해서는 이전 값 $D_{i,j}^{(3)}$가 필요하고 노드 4의 진출과 진입을 나타내는 행과 열 정보가 필요하다. 그런데 $D_{i,4}^{(4)}$는 목적 노드가 4이므로 $D_{i,4}^{(3)}$과 같으며, $D_{4,j}^{(4)}$는 출발 노드가 4이므로 $D_{4,j}^{(3)}$와 같다. 즉, $D_{i,j}^{(4)}$를 계산할 때 $D_{i,4}^{(4)}$와 $D_{4,j}^{(4)}$는 이전 값에서 변하지 않는다. 따라서 초기 인접 행렬 공간만 이용하여 알고리즘에서 필요한 모든

계산을 수행할 수 있다. 따라서 공간 복잡도는 $O(n^2)$이다. Floyd-Warshall 알고리즘은 알고리즘 11.4과 같다.

알고리즘 11.4 Floyd-Warshall 알고리즘

```
1: function FLOYD(G)
2:     table := G
3:     for k := 1 to n do
4:         for i := 1 to n do
5:             for j := 1 to n do
6:                 table[i][j] := MIN(table[i][j], table[i][k] + table[k][j])
7:     for i := 1 to n do                                   ▷ 음의 주기 검사
8:         if D[i][i] < 0 then return null
9:     return table
```

알고리즘 11.5 Floyd-Warshall 해 찾기 알고리즘

```
1: function FLOYD(table, v, w)
2:     path := []
3:     FLOYD(table, v, w, path)
4:     return path
5: procedure FLOYD(table, v, w, path)
6:     for k := 1 to n do
7:         if k ≠ v and k ≠ w and table[v][w] = table[v][k] + table[k][w] then
8:             FLOYD(G, v, k, path)
9:             FLOYD(G, k, w, path)
10:    path.PUSHBACK((v, w))
```

5. 외판원 문제

외판원 문제(traveling salesperson problem)는 0-1 배낭 채우기 문제만큼 컴퓨터공학에서는 매우 유명한 문제이다. 이 문제는 출발 노드에서 다른 모든 노드를 한 번씩 방문하고 다시 출발 노드로 돌아오는 최단 경로를 찾는 문제이다. 이 문제는 외판원이 모든 도시를 방문하고 시작 도시로 되돌아오는 가장 짧은 경로를 찾는 문제이며, 보통 이 문제는 모든 노드가 서로 이웃인 완전 그래프를 가정한다. 모든 도시를 한번 방문해야 하기 때문에 출발 노드가 중요하지 않다. 외판원 문제에서 찾는 경로를 다른 말로 해밀톤 경로(hamiltonian circuit) 또는 일주여행경로(tour)라 한다. 앞으로 간결하게 표현하기 위해 투어라는 표현을 사용한다.

외판원 문제를 동적 프로그래밍으로 해결할 수 있기 위해서는 최적화 문제이어야 한다. 이 문제가 과연 최적화 문제일까? v_1을 출발 노드로 시작하는 최적 투어를 생각하여 보자. v_k가 최적 투어 상에서 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 경로이다. 이때 주의할 것은 기존에 살펴본 최단 경로 문제는 최단 경로를 구성하는 간선의 수나 노드에 대한 제한이 없었다. 이 문제에서는 모든 노드를 정확하게 한 번 꼭 지나가야 한다.

앞서 살펴본 최적 원칙을 활용하여 이 문제를 해결할 방법을 생각하여 보자. v_2에서 모든 노드를 정확히 한 번 지나가는 v_1까지 최단 경로를 구하고, 그다음 차례 차례 v_i에서 모든 노드를 정확히 한 번 지나가는 v_1까지 최단 경로를 모두 구하면 우리가 찾고자 하는 최적 투어를 구할 수 있다. $D[v_k][A]$가 A에 속한 노드를 한 번씩 거치며 v_k에서 v_1로 가는 최단 경로의 길이이면 최적 투어의 길이는 다음과 같이 정의할 수 있다.

$$D[1][V-\{1\}] = \min_{k=2}^{n} \{G[1][k] + D[k][V-\{k,1\}]\}$$

상향식으로 이 문제를 해결하기 위해 $D[k][A]$를 일반화하면 다음과 같이 정의할 수 있다.

$$D[k][A] = \min_{j \in A} \{G[k][j] + D[k][A - \{j\}]\}$$

$D[k][A]$는 노드 k에서 출발하여 A에 있는 모든 노드를 한 번씩 지나 노드 1로 가는 최단 경로의 길이이며, 이때 $D[k][\emptyset] = G[k][1]$이다.

실제 위 점화식을 이용하여 외판원 문제를 테블레이션으로 해결할 수 있지만 여전히 지수 비용이 필요하다. 위 점화식을 이용한 알고리즘의 시간 복잡도는 $O(n^2 2^n)$이다. 위 D에서 두 번째 색인이 정수가 아니라 집합이기 때문에 구현이 간단하지 않다. 보통 집합을 비트 벡터로 표현하여 구현한다.

퀴즈

1. 동적 프로그래밍을 이용하여 최적화 문제를 해결하기 위해서는 해당 문제가 최적 원칙을 만족해야 한다. 이와 관련된 다음 설명 중 틀린 것은?

   ① 외판원 문제에서 노드 v_1을 출발점으로 하는 최적 투어 P가 있을 때, 이 경로에서 v_1 다음 노드가 v_2이면 P의 부분경로 v_2에서 v_1까지의 경로 P'은 그래프의 모든 노드를 한 번씩 거치며 v_2에서 v_1로 가는 최단 경로이다.

   ② Floyd-Warshall 알고리즘에서 s에서 v까지의 최단 경로가 P이고, 이 경로가 중간에 w 노드를 지나가면 P를 s에서 w까지의 경로 P_1과 w에서 v까지의 경로 P_2로 나눌 수 있다. 이때 P_1은 s에서 w로 가는 w를 이용하지 않는 최단 경로이고, P_2는 w에서 v로 가는 w를 이용하지 않는 최단 경로이다.

   ③ 0-1 배낭 채우기 문제에서 n개의 물건이 주어지고, 가방의 용량이 W일 때, 최적해 S에 n번째 물건이 포함되었다고 하자. n번째 물건의 무게가 w_n이면 S에서 n번째 물건을 제외한 해 S'은 용량이 $W - w_n$인 가방에 n번째 물건을 제외한 $n-1$개의 물건을 이용한 문제의 최적해이다.

   ④ 최장 경로 문제에서 s에서 v까지의 최장 경로가 P일 때, 이 경로가 w를 지나가면 s에서 w까지의 P의 부분경로 P'은 s에서 w까지의 최장 경로이다.

2. $x < y < z$를 만족하는 3개의 키로 구성된 이진 검색 트리가 있다. x, y, z의 검색빈도가 각각 60%, 10%, 30%일 때 최적 이진 검색 트리의 평균 검색 비용은?

   ① 2.3 ② 2.0 ③ 1.6 ④ 1.7

3. 간선 가중치에 대한 제한(음수가 가능한)이 없는 방향 그래프에서 최단 경로 찾아주는 Bellman-Ford 알고리즘과 관련된 다음 설명 중 틀린 것은?

   ① Bellman-Ford 알고리즘은 특정 출발노드에서 다른 모든 노드까지 최단 경로 찾아 준다. 이 알고리즘은 경로의 간선 수를 기준으로 최단 경로를 찾는다.

   ② Bellman-Ford에서 P가 s에서 v까지 최단 경로이고, 이 경로에서 v의 바로 전 노드가 w일 때 s에서 w로 가는 P의 부분 경로 P'은 s에서 w까지 최단 경로이다.

   ③ Bellman-Ford 알고리즘은 출발노드에서 간선 수를 늘리면서 각 노드까지의 최단경로를 찾아준다. m개의 간선을 이용하여 찾는 최단 경로와 $m+1$개의 간선을 이용하여 찾은 최단 경로가 같으면 알고리즘을 중단할 수 있다.

   ④ Bellman-Ford 알고리즘은 $n-1$개의 간선을 이용하는 최단 경로까지 계속 더 짧은 경로를 찾은 경우 알고리즘을 종료할 수 없고, n개의 간선을 이용하는 최단 경로까지 찾아야 한다.

4. 간선 가중치에 대한 제한(음수가 가능한)이 없는 방향 그래프에서 최단 경로 찾아주는 Floyd-Warshall 알고리즘과 관련된 다음 설명 중 틀린 것은?

   ① Floyd-Warshall 알고리즘은 경로를 구성하는 노드를 하나씩 늘리면서 최단 경로를 찾는다.

   ② Floyd-Warshall 알고리즘은 알고리즘을 수행하기 위해 인접 행렬로 그래프를 표현한 후에 추가로 같은 크기의 2차원 배열 n개가 반드시 필요하다.

   ③ Floyd-Warshall 알고리즘은 모든 노드 간 최단 경로를 찾아주며, Bellman-Ford 알고리즘을 각 노드마다 수행하는 것보다 성능이 좋다.

   ④ Floyd-Warshall 알고리즘은 최종 2차원 배열에서 $D[i][i]$의 값을 조사하여 음의 주기가 존재하는지 검사한다.

연습문제

1. 동전의 액면가를 나타내는 일련의 정수와 거스름액을 나타내는 정수가 주어진다. 이 거스름을 지급하기 위한 최소 동전 수를 출력하라. 주어진 액면가로 거스름을 제공할 수 없으면 -1을 출력하라. 각 액면가의 동전은 무한히 많다고 가정한다. 예를 들어 액면가를 나타내는 $[1, 2, 5]$가 주어지고, 거스름액이 11이면 3개의 동전을 이용하여 거스름을 지급할 수 있다. 이 문제를 테뷸레이션으로 해결하기 위한 점화식을 제시한 후에 테뷸레이션을 이용하는 알고리즘을 제시하라.

2. 두 개의 문자열이 주어진다. 두 문자열에서 공통된 가장 긴 부분 문자열의 길이를 출력하라. 공통된 문자열이 없으면 0을 출력해야 한다. 여기서 부분 문자열이란 기존 문자열에서 일부 문자를 제거한 문자열을 말하여, 원래 문자의 순서는 유지된 문자열을 말한다. 예를 들어 "ace"는 "abcde"의 부분 문자열이지만 "bac"는 아니다.

3. 길동이와 춘향이가 나눗셈 게임을 하고 있다. 이 게임을 시작하면 하나의 정수 $n(1 \leq n \leq 1,000)$이 주어진다. 길동이와 춘향이는 번갈아 다음을 수행하여 게임을 진행한다.

   - 0보다 크고 n보다 작으며 n을 나눌 수 있는 정수 x를 선택하여 n을 $n - x$로 교체한다.

   이와 같은 x를 선택할 수 없으면 게임에서 지게 된다. 춘향이가 먼저 시작하고, 항상 최적의 수를 선택할 경우 춘향이가 이길 수 있으면 true를 출력하고 이길 수 없으면 false를 출력하라. 테뷸레이션을 이용하여 해결하기 위한 점화식을 제시하고, 이를 바탕으로 테뷸레이션을 이용하는 알고리즘을 제시하라.