11주차 1차시 동적 계획 알고리즘의 이해 I

【학습목표】

1. 동적 계획 알고리즘을 이해할 수 있다.
2. 모든 쌍 최단 경로 알고리즘을 이해할 수 있다.

지난 강의 정리

1. 궤(통) 채우기 알고리즘

궤의 용량이 모두 1이고 각 물체의 크기 xi가 다음과 같이 주어졌을 때, 최초법에 의해 물체를 궤에 집어넣을 때 필요한 궤의 최소 개수는?

(xi)=(0.4, 0.7, 0.2, 0.6, 0.5, 0.4)

서로 다른 크기의 물체들과 동일한 크기의 궤들이 있을 때 최소 개수의 궤를 써서 모든 물체를 궤에 집어넣는 문제가 궤 채우기 문제이다. 이 문제는 NP-완전문제로서 결정론적 다항식 시간 알고리즘을 작성하는 일이 대단히 힘들기 때문에 실용적인 측면에서 근사 알고리즘을 이용하게 된다. 이 문제를 풀기 위한 방법은 최초법, 최선법, 감소순 최초법, 감소순 최선법이 있다. 주어진 문제에 대해 최초법은 (0.4,0.2,0,4)(0.7),(0.6),(0.5)의 4개의 궤가 필요하다.

2. 작업 스케줄링 알고리즘

- 작업 스케줄링 (Job Scheduling) 문제는 n개의 작업, 각 작업의 수행 시간 ti, i = 1, 2, 3, …, n, 그리고 m개의 동일한 기계가 주어질 때, 모든 작업이 가장 빨리 종료되도록 작업을 기계에 배정하는 문제이다.
- 단, 한 작업은 배정된 기계에서 연속적으로 수행되어야 한다.
- 또한 기계는 1번에 하나의 작업만을 수행한다.

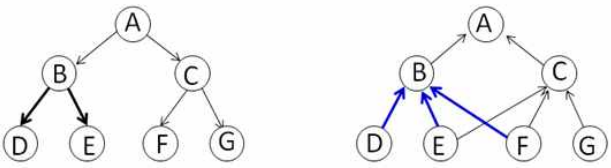
- 작업을 어느 기계에 배정하여야 모든 작업이 가장 빨리 종료될까?
- 이에 대한 간단한 답은 그리디 방법으로 작업을 배정하는 것이다.
- 즉, 현재까지 배정된 작업에 대해서 가장 빨리 끝나는 기계에 새 작업을 배정하는 것이다.

학습내용1 : 동적 계획알고리즘 이란

1. 동적 계획 (Dynamic Programming) 알고리즘

동적 계획 (Dynamic Programming) 알고리즘은 그리디 알고리즘과 같이 최적화 문제를 해결하는 알고리즘이다. 동적 계획 알고리즘은 먼저 입력 크기가 작은 부분 문제들을 모두 해결한 후에 그 해들을 이용하여 보다 큰 크기의 부분 문제들을 해결하여, 최종적으로 원래 주어진 입력의 문제를 해결하는 알고리즘이다.

- 분할 정복 알고리즘과 동적 계획 알고리즘의 전형적인 부분문제들 사이의 관계



분할 정복 알고리즘　　동적 계획 알고리즘

- 분할 정복 알고리즘의 부분문제들 사이의 관계: A는 B와 C로 분할되고, B는 D와 E로 분할되는데, D와 E의 해를 취합하여 B의 해를 구한다. 단, D, E, F, G는 각각 더 이상 분할할 수 없는 (또는 가장 작은 크기의) 부분문제들이다.
- 마찬가지로 F와 G의 해를 취합하여 C의 해를 구하고, 마지막으로 B와 C의 해를 취합하여 A의 해를 구한다.

- 동적 계획 알고리즘은 먼저 최소 단위의 부분 문제 D, E, F, G의 해를 각각 구한다. 그 다음에
- D, E, F의 해를 이용하여 B의 해를 구한다.
- E, F, G의 해를 이용하여 C의 해를 구한다.
- B와 C의 해를 구하는데 E와 F의 해 모두를 이용한다.
- 분할 정복 알고리즘은 부분문제의 해를 중복 사용하지 않는다.
- 동적 계획 알고리즘에는 부분문제들 사이에 의존적 관계가 존재한다.

→ 예를 들면, D, E, F의 해가 B를 해결하는데 사용되는 관계가 있다.

- 이러한 관계는 문제 또는 입력에 따라 다르고, 대부분의 경우 뚜렷이 보이지 않아서 '함축적인 순서' (implicit order)라고 한다.

● **주어진 문제를 여러 개의 소문제로 분할하여 각 소문제의 해결안을 바탕으로 주어진 문제를 해결하는 기법**
  - 각 소문제는 원래 주어진 문제와 동일한 문제이지만 입력의 크기가 작음
  - 소문제를 반복 분할하면 결국 입력의 크기가 아주 작은 단순한 문제가 되어 쉽게 해결 가능
  - 소문제의 해를 표 형식으로 저장해 놓고 이를 이용하여 입력 크기가 보다 큰 원래의 문제를 점진적으로 해결

● **최소치/최대치를 구하는 최적화 문제에 적용**

● **분할 정복 방법과 유사**
  - 분할 정복
    - 분할되는 소문제가 서로 독립적
    - 소문제를 순환적으로 풀어 결과를 합침
  - 동적 프로그래밍
    - 소문제가 독립적이지 않음
    - 분할된 소문제 간에 중복 부분이 존재

* 피보나치수열의 예

● $f_n = f_{n-1} + f_{n-2}, \ n \geq 2$

● $f_{10} = f_9 + f_8$

● $f_9 = f_8 + f_7$

* 최적성의 원리

◆ 주어진 문제에 대한 최적해가 소문제에 대한 최적해로 구성

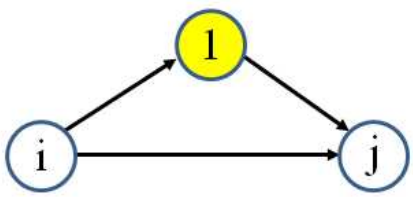
- 욕심쟁이 방법
  - 국부적인 최적해들이 전체적인 최적해를 구성
  - 소문제에 대한 하나의 최적해만을 고려
- 동적 프로그래밍
  - 소문제에 대한 여러 최적해로부터 다음 크기의 소문제에 대한 최적해가 결정

학습내용2 : 모든 쌍 최단 경로 알고리즘

1. 모든 쌍 최단 경로 (All Pairs Shortest Paths)

모든 쌍 최단 경로 (All Pairs Shortest Paths) 문제는 각 쌍의 점 사이의 최단 경로를 찾는 문제이다.
• 이 문제를 해결하려면, 각 점을 시작점으로 정하여 다익스트라(Dijkstra)의 최단 경로 알고리즘을 수행하면 된다.
• 이때의 시간복잡도는 배열을 사용하면 $(n-1)xO(n^2) = O(n^3)$이다. 단, n은 점의 수이다.
• Warshall은 그래프에서 모든 쌍의 경로 존재 여부 (transitive closure)를 찾아내는 동적 계획 알고리즘을 제안했고, Floyd는 이를 변형하여 모든 쌍 최단 경로를 찾는 알고리즘을 고안하였다.
• 따라서 모든 쌍 최단 경로를 찾는 동적 계획 알고리즘을 플로이드-워샬 알고리즘이라 한다. (간략히 플로이드 알고리즘이라고 하자.)

• 동적 계획 알고리즘으로 모든 쌍 최단 경로 문제를 해결하려면 먼저 부분문제 들을 찾아야 한다.
• 이를 위해 일단 그래프의 점의 수가 적을 때를 생각해보자.
• 그래프에 3개의 점이 있는 경우, 점 i에서 점 j까지의 최단 경로를 찾으려면 2가지 경로, 즉, 점 i에서 점 j로 직접 가는 경로와 점 1을 경유하는 경로 중에서 짧은 것을 선택하면 된다.

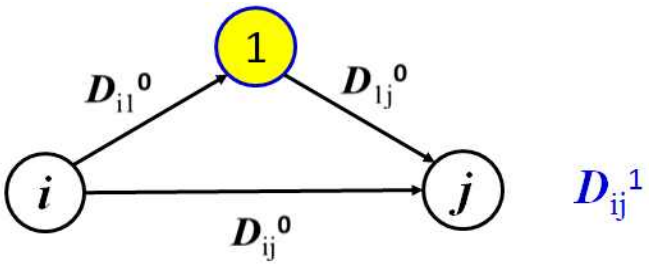


• 또 하나의 중요한 아이디어는 경유 가능한 점들을
- 점 1로부터 시작하여, 점 1과 2, 그 다음엔 점 1, 2, 3으로 하나씩 추가하여, 마지막에는 점 1~n까지의 모든 점을 경유 가능한 점들로 고려하면서, 모든 쌍의 최단 경로의 거리를 계산한다.

• 부분문제 정의: 단, 입력 그래프의 점을 각각 1, 2, 3, …, n이라 하자.
D_{ij}^{k} = 점 {1, 2, …, k}만을 경유 가능한 점들로 고려하여, 점 i로부터 점 j까지의 모든 경로 중에서 가장 짧은 경로의 거리

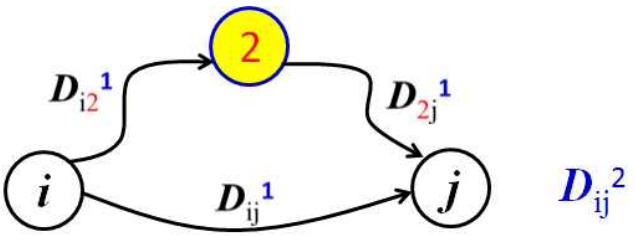
• 여기서 주의할 것은 점 1에서 점 k까지의 모든 점들을 반드시 경유하는 경로를 의미하는 것이 아니다.
• 심지어는 D_{ij}^{k}는 이 점들을 하나도 경유하지 않으면서 점 i에서 점 j에 도달하는 경로, 즉 선분 (i,j)가 최단 경로가 될 수도 있다.
• 여기서 k≠i, k≠j이고, k=0인 경우, 점 0은 그래프에 없으므로 어떤 점도 경유하지 않는다는 것을 의미한다.
따라서D_{ij}^{0}은 입력으로 주어지는 선분 (i,j)의 가중치이다.

• D_{ij}^{1}은 i에서 점 1을 경유하여 j로 가는 경로와 i에서 j로 직접 가는 경로, 선분(i,j), 중에서 짧은 거리이다.
• 따라서 모든 쌍 i와 j에 대하여 D_{ij}^{1}을 계산하는 것이 가장 작은 부분 문제들이다. 단 i≠1, j≠1 이다.

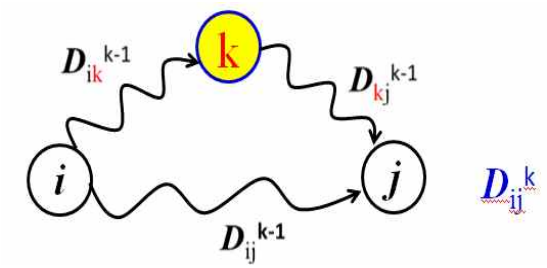


• 그 다음엔 i에서 점 2를 경유하여 j로 가는 경로의 거리와 D_{ij}^{1} 중에서 짧은 거리를 D_{ij}^{2}로 정한다. 단, 점 2를 경유하는 경로의 거리는 D_{i2}^{1} + D_{2j}^{1} 이다.

• 모든 쌍 i와 j에 대하여 D_{ij}^{2}를 계산하는 것이 그 다음으로 큰 부분 문제들이다. 단, i≠2, j≠2이다.



• 점 i에서 점 k를 경유하여 j로 가는 경로의 거리와 D_{ij}^{k-1} 중에서 짧은 것을 로 정한다. 단, 점 k를 경유 하는 경로의 거리는 D_{ik}^{k-1} + D_{kj}^{k-1} 이고, i≠k, j≠k이다.



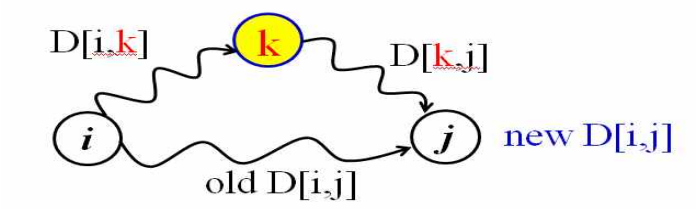
• 이런 방식으로 k가 1에서 n이 될 때까지 D_{ij}^{k}를 계산해서, D_{ij}^{n}, 즉, 모든 점을 경유 가능한 점들로 고려된 모든 쌍 i와 j의 최단 경로의 거리를 찾는 방식이 플로이드의 모든 쌍 최단 경로 알고리즘이다.

2. 모든 쌍 최단 경로알고리즘

AllPairsShortest
입력: 2차원 배열 D, 단, D[i,j]=선분 (i,j)의 가중치, 만일 선분 (i,j)이 존재하지 않으면 D[i,j]=∞, 모든 i에 대하여 D[i,i]=0이다.
출력: 모든 쌍 최단 경로의 거리를 저장한 2-d 배열 D

```
1. for k = 1 to n
2.       for i = 1 to n (단, i≠k)
3.             for j = 1 to n (단, j≠k, j≠i)
4.                   D[i,j] = min{D[i,k]+D[k,j], D[i,j]}
```

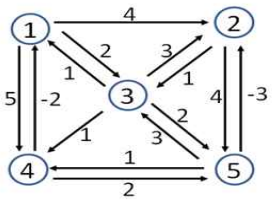
Line 1의 for-루프는 k가 1에서 n까지 변하는데, 이는 경유 가능한 점을 1부터 n까지 확장시키기 위한 것이다.
Line 2~3: 점들의 각 쌍, 즉, 1-1, 1-2, 1-3, …, 1-n, 2-1, 2-2, …, 2-n, …, n-1, n-2, …, n-n을 하나씩 고려하기 위한 루프이다. 단, i-i, 라든가 i=k 또는 j=k의 경우에는 수행하지 않는다.
Line 4: 각 점의 쌍 i-j에 대해 i에서 j까지의 거리가 k를 포함하여 경유하는 경로의 거리, 즉, D[i,k]+D[k,j]와 점 {1, 2, …, (k-1) }만을 경유 가능한 점들로 고려하여 계산된 최단 경로의 거리 D[i,j]가 짧은지를 결정하여 D[i,j]를 갱신한다.



- 모든 쌍 최단 경로 문제의 부분문제간의 함축적 순서는 line 4에 표현되어있다.
- 즉, 새로운 D[i,j]를 계산하기 위해서 미리 계산되어 있어야 할 부분문제들은 D[i,k]와 D[k,j]이다.

- AllPairsShortest 알고리즘의 입력 그래프에는 사이클 상의 선분들의 가중치 합이 음수가 되는 사이클은 없어야 한다.
- 이러한 사이클을 음수 사이클 (negative cycle)이라 하는데, 최단 경로를 찾는데 음수 사이클이 있으면, 이 사이클을 반복하여 돌아 나올 때마다 경로의 거리가 감소되기 때문이다.

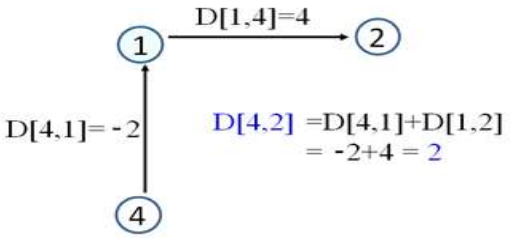
AllPairsShortest 알고리즘 수행 과정



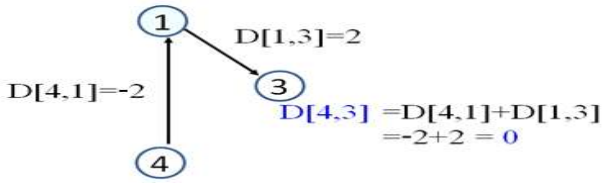
D	1	2	3	4	5
1	0	4	2	5	∞
2	∞	0	1	∞	4
3	1	3	0	1	2
4	-2	∞	∞	0	2
5	∞	-3	3	1	0

- 배열 D의 원소들이 k가 1부터 5까지 증가함에 따라서 갱신되는 것을 살펴보자.

k=1일 때:
- D[2,3] = min{D[2,3], D[2,1]+D[1,3]} = min{1, ∞+2} = 1
- D[2,4] = min{D[2,4], D[2,1]+D[1,4]} = min{∞, ∞+5} = ∞
- D[2,5] = min{D[2,5], D[2,1]+D[1,5]} = min{4, ∞+∞} = 4
- D[3,2] = min{D[3,2], D[3,1]+D[1,2]} = min{3, 1+4} = 3
- D[3,4] = min{D[3,4], D[3,1]+D[1,4]} = min{1, 1+5} = 1
- D[3,5] = min{D[3,5], D[3,1]+D[1,5]} = min{2, 1+∞} = 2
- D[4,2] = min{D[4,2], D[4,1]+D[1,2]} = min{∞, -2+4} = 2 //갱신됨.



D[4,3] = min{D[4,3], D[4,1]+D[1,3]} = min{∞, -2+2} = 0//갱신됨.



- D[4,5] = min{D[4,5], D[4,1]+D[1,5]} = min{2, -2+∞} = 2
- D[5,2] = min{D[5,2], D[5,1]+D[1,2]} = min{-3, ∞+4} = -3
- D[5,3] = min{D[5,3], D[5,1]+D[1,3]} = min{3, ∞+2} = 3
- D[5,4] = min{D[5,4], D[5,1]+D[1,4]} = min{1, ∞+5} = 1

- k=1일 때 D[4,2], D[4,3]이 각각 2, 0으로 갱신된다.
  다른 원소들은 변하지 않았다.

D	1	2	3	4	5
1	0	4	2	5	∞
2	∞	0	1	∞	4
3	1	3	0	1	2
4	-2	**2**	**0**	0	2
5	∞	-3	3	1	0

- k=2일 때:
  - D[1,5]가 1 → 2 → 5의 거리인 8로 갱신된다.
  - D[5,3]이 5 → 2 → 3의 거리인 -2로 갱신된다.

D	1	2	3	4	5
1	0	4	2	5	**8**
2	∞	0	1	∞	4
3	1	3	0	1	2
4	-2	2	0	0	2
5	∞	-3	**-2**	1	0

- k=3일 때 총 7개의 원소가 갱신된다.

D	1	2	3	4	5
1	0	4	2	**3**	**4**
2	**2**	0	1	**2**	**3**
3	1	3	0	1	2
4	-2	2	0	0	2
5	**-1**	-3	-2	**-1**	0

D[1,4]=1→3→4 3으로 갱신된다.
D[1,5]=1→3→5 4로 갱신된다.
D[2,1]=2→3→1 2로 갱신된다.
D[2,4]=2→3→4 2로 갱신된다.
D[2,5]=2→3→5 3로 갱신된다.
D[5,1]=5→2→3→1 -1로 갱신된다.
D[5,4]=5→2→3→4 -1로 갱신된다.

- k=4일 때 총 3개의 원소가 갱신된다.

D	1	2	3	4	5
1	0	4	2	3	4
2	**0**	0	1	2	3
3	**-1**	3	0	1	2
4	-2	2	0	0	2
5	**-3**	-3	-2	-1	0

D[2,1]=2→3→4→1 0로 갱신된다.
D[3,1]=3→4→1 -1로 갱신된다.
D[5,1]=5→2→3→4→1 -3로 갱신된다.

- k=5일 때 총 3개의 원소가 갱신되고, 이것이 주어진 입력에 대한 최종해이다.

D	1	2	3	4	5
1	0	**1**	2	3	4
2	0	0	1	2	3
3	-1	**-1**	0	1	2
4	-2	**-1**	0	0	2
5	-3	-3	-2	-1	0

D[1,2]=1→3→5→2 1로 갱신된다.
D[3,2]=3→5→2 -1로 갱신된다.
D[4,2]=4→5→2 -1로 갱신된다.

【학습정리】

1. 동적 계획알고리즘 이란
- 동적 계획 (Dynamic Programming) 알고리즘은 그리디 알고리즘과 같이 최적화 문제를 해결하는 알고리즘이다. 동적 계획 알고리즘은 먼저 입력 크기가 작은 부분 문제들을 모두 해결한 후에 그 해들을 이용하여 보다 큰 크기의 부분 문제들을 해결하여, 최종적으로 원래 주어진 입력의 문제를 해결하는 알고리즘이다.

- 동적 계획 알고리즘에는 부분문제들 사이에 의존적 관계가 존재한다.
→ 예를 들면, D, E, F의 해가 B를 해결하는데 사용되는 관계가 있다.

- 이러한 관계는 문제 또는 입력에 따라 다르고, 대부분의 경우 뚜렷이 보이지 않아서 '함축적인 순서' (implicit order)라고 한다.

2. 모든 쌍 최단 경로 알고리즘
모든 쌍 최단 경로 (All Pairs Shortest Paths) 문제는 각 쌍의 점 사이의 최단 경로를 찾는 문제이다.
- 이 문제를 해결하려면, 각 점을 시작점으로 정하여 다익스트라(Dijkstra)의 최단 경로 알고리즘을 수행하면 된다.
- 이때의 시간복잡도는 배열을 사용하면 $(n-1)xO(n^2) = O(n^3)$이다. 단, n은 점의 수이다.
- Warshall은 그래프에서 모든 쌍의 경로 존재 여부 (transitive closure)를 찾아내는 동적 계획 알고리즘을 제안했고, Floyd는 이를 변형하여 모든 쌍 최단 경로를 찾는 알고리즘을 고안하였다.
- 따라서 모든 쌍 최단 경로를 찾는 동적 계획 알고리즘을 플로이드-워샬 알고리즘이라 한다. (간략히 플로이드 알고리즘이라고 하자.)