KNU KYUNGPOOK NATIONAL UNIVERSITY

Lecture #17: Dynamic programming

School of Computer Science and Engineering
Kyungpook National University (KNU)

Woo-Jeoung Nam

Coin Change Problem

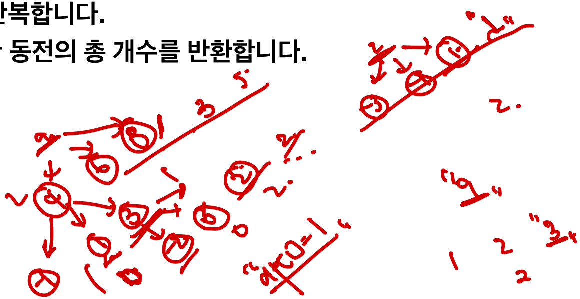
- 주어진 금액을 거스름돈으로 줄 때, 사용하는 동전의 최소 개수를 찾는 최적화 문제
  - ➢ 그리디, 동적 프로그래밍 등..

Coin Change Problem

- 그리디 알고리즘(Greedy Algorithm):
- 그리디 알고리즘을 사용하여 동전 거스름돈 문제를 해결하는 경우

- 각 단계에서 가장 가치가 큰 동전을 선택하여 거스름돈을 만든다

- a. 동전을 가치에 따라 내림차순으로 정렬합니다.
- b. 가장 가치가 큰 동전부터 사용하여 거스름돈 금액을 채우되, 금액을 초과하지 않는 선에서 최대한 많이 사용합니다.
- c. 모든 동전에 대해 위 과정을 반복합니다.
- d. 모든 동전을 검토한 후 사용한 동전의 총 개수를 반환합니다.



Coin Change Problem

- 동적 프로그래밍(Dynamic Programming):
- 동적 프로그래밍을 사용하여 동전 거스름돈 문제를 해결하는 경우
- 각 금액에 대해 필요한 최소 동전 개수를 구하는 부분 문제로 분할

- a. 금액을 저장할 수 있는 1차원 배열을 생성
  - ➢ 배열의 인덱스는 금액을, 값은 해당 금액을 만들기 위해 필요한 최소 동전 개수를 나타냄
- b. 각 동전에 대해 다음 작업을 수행합니다:
  - ➢ i. 현재 동전을 사용하여 가능한 모든 금액에 대해 최소 동전 개수를 업데이트합니다.
  - ➢ ii. 배열의 값을 현재 동전을 사용한 경우와 사용하지 않은 경우 중 최소값으로 업데이트합니다.
  - ➢ c. 배열의 마지막 요소는 주어진 금액에 대한 최소 동전 개수를 나타냅니다. 이 값을 반환합니다.

dp[j] += dp[j-i]

Coin Change Problem

- 문제 정의: $OPT(v)$ = v원을 거슬ㄹ
- 목표: $OPT(V)$ 구하기
- 선택지 탐색: $OPT(v)$를 구하기 위하
- (
- (위해 필
- 성

$$OPT(v) = \begin{cases} \infty & \text{if } v < 0 \\ 0 & \text{if } v = 0 \\ \min\limits_{1 \le i \le n} \{\, 1 + OPT(v - d_i) \,\} & \text{if } v > 0 \end{cases}$$

D.P

$v == 0$

$1 + opt(v - d_i)$

$1 \le i$

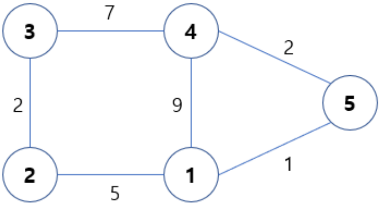
$1 + opt(v - d_i)$

$opt(v) = \min\limits_{1 \le i \le n}$

Floyd-Warshall 알고리즘

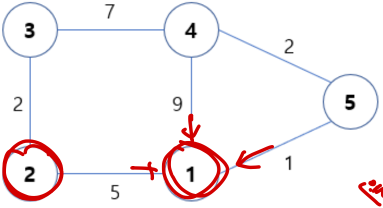
- 그래프에서 모든 노드 간의 최단 경로를 찾는 다이내믹 프로그래밍
- 가중치가 있는 그래프에서 사용할 수 있으며, 음수 가중치를 갖는 간선도 처리

- 응용분야:
- 네트워크 체계 하에서는 거쳐가는 경로(간선)가 많을 수록 전송 딜레이가 높아진다.
- 물류 허브를 거칠 때 마다, 운송 상품의 손실 및 분실 가능성이 높아진다.
- 따라서, 최단 거리만 중요한 것이 아니라, 최대 '경유' 횟수에 제한을 두는 경우도 고려한다.



Floyd-Warshall 알고리즘

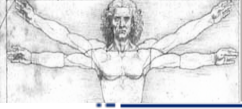
- 초기 그래프를 인접행렬로 나타내면 다음과 같습니다. INF는 해당 노드에서 특정 노드까지 가는 길이 없다는 의미



출발노드 도착노드

1을 거쳐가

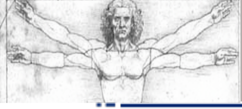
	1	2	3	4	5
1	0	5	INF	9	1
2	**5**	**0**	**2**	2→1→4 **INF**	2→1→5 **INF**
3	**INF**	**2**	**0**	**7**	**INF**
4	**9**	**INF**	**7**	**0**	**2**
5	**1**	**INF**	**INF**	**2**	**0**



Floyd-Warshall 알고리즘

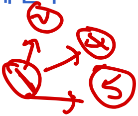
- 핵심 아이디어: 최단 경로 $\{u \rightarrow w_1 \rightarrow$
이에 몇 몇개의 노드가 존재할 수 있
- 만약 이러한 **intermediate node(ve**
연결되어 있음.
- 우리가 허용가능한 **intermediate n**

	1	2	3	4	5
1	0	5	INF	9	1
2	**5**	**0**	**2**	**INF**	**INF**
3	**INF**	**2**	**0**	**7**	**INF**
4	**9**	**INF**	**7**	**0**	**2**
5	**1**	**INF**	**INF**	**2**	**0**



Floyd-Warshall 알고리즘

- **ROUND 1 : 1번 노드를 새로운 중간 노드로 설정**
  - ➢ 1번부터 5번 노드까지 존재하므로 알고리즘은 총 5라운드의 과정을 거침
    - 즉, 모든 노드들을 중간 노드로 선정하는 과정을 각 라운드마다 거친다
  - ➢ 2번에서 4번으로 가는 길은 원래 없었으나, 1번 노드를 중간 노드로 선정된 경우 2-1-4(길이 5+9=14) 로 갈 수 있게 된다



1번노드를 중간 노드로

	1	2	3	4	5
1	0	5	INF	9	1
2	5	0	2	14	6
3	INF	2	0	7	INF
4	9	14	7	0	2
5	1	6	INF	2	0

Floyd-Warshall 알고리즘

- ROUND 2 : 2번 노드를 새로운 중간 노드로 설정
  - ➢ 1번-3번 노드를 잇는 경로, 3번-5번 노드를 잇는 경로가 새로 생기게 된다

- 이런 과정으로 1~5번 노드를 중간 노드로 선정하는 라운드까지 모두 거치면 행렬에는 모든 노드 간 최단 거리가 들어가게 된다

- https://www.acmicpc.net/problem/11404

	1	2	3	4	5
1	0	5	7	9	1
2	5	0	2	14	6
3	7	2	0	7	8
4	9	14	7	0	2
5	1	6	8	2	0

Floyd-Warshall 알고리즘

- ROUND 2 : 2번 노드를 새로운 중간 노드로 설정
  - ➢ 1번-3번 노드를 잇는 경로, 3번-5번 노드를 잇는 경로가 새로 생기게 된다

- 이런 과정으로 1~5번 노드를 중간 노드로 선정하는 라운드까지 모두 거치면 행렬에는 모든 노드 간 최단 거리가 들어가게 된다

- https://www.acmicpc.net/problem/11404

```
for i = 1 to n
    for j = 1 to n
        dist(i, j, 0) = ∞
for all (i, j) ∈ E
    dist(i, j, 0) = ℓ(i, j)
```

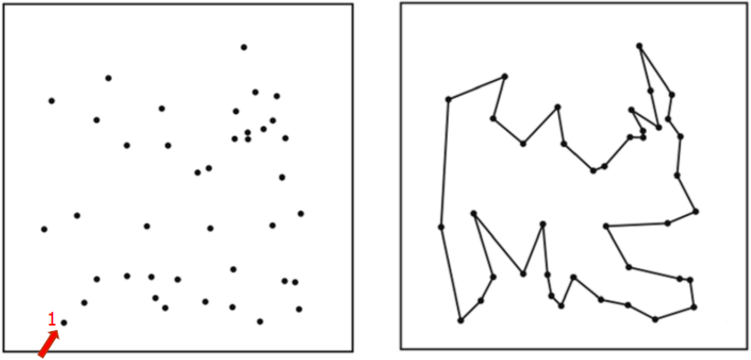
initialization

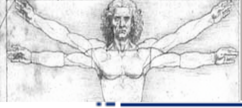
```
for k = 1 to n          → 거쳐가는 노드로 설정
  for i = 1 to n
    for j = 1 to n      → 모든 정점을 확인.
      dist(i, j, k) =
```

$$\boxed{\min\{\mathtt{dist}(i,k,k-1)+\mathtt{dist}(k,j,k-1),\ \mathtt{dist}(i,j,k-1)\}}$$

Traveling Salesman Problem

- 도시 한 곳에서 출발, 나머지 모든 도시들을 한 번씩만 방문한 뒤 다시 출발한 도시로 도착
- 각 도시 i 및 도시 j 간의 거리는 d_ij로 주어지며, 배열 D에 모든 도시들 간의 거리가 주어진다.
- 목표: 모든 도시를 순회하여 출발 지점으로 돌아오는 경로 중 가장 짧은 경로를 구하기
- 대표적인 NP-Complete (이면서 NP-Hard인) 문제



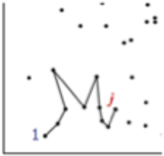


Traveling Salesman Problem

- 1. 완전 탐색(**Brute Force**): 가능한 식입니다. 시간 복잡도는 **O(n!)**, 실용
- 2. 동적 프로그래밍(**Dynamic Prog** 제를 해결하는 방식.
  - **Held-Karp** 알고리즘은 **TSP**에 동적 $O(n^2 * 2^n)$입니다. 소규모의 **TSP**에 율적입니다.
- 3. 근사 알고리즘(**Approximation A**

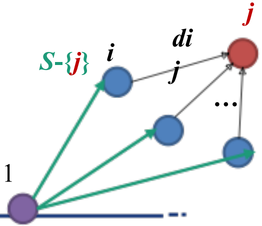
Traveling Salesman Problem

- 1번 도시에서 출발하여 몇 개의 도시를 거친 후,
  현재 j번째 도시에 있다고 가정해보자.
- 이 상태에서 추가적으로 도시 방문을 하고자 할 때,
  우리에게 어떤 정보가 필요할까?
- 당연히 이미 방문했던 도시들을 재방문 할 필요는 없다. (문제 조건)
- 그 다음 방문할 도시로 제일 적합한 도시는? (어떤 기준으로?)
- 부분문제 정의:
  - ➢ 도시들의 부분 집합 S ⊆ {1, 2, …, n}가 도시 1, j를 포함한다고 할 때,
  - ➢ C(S, j)를 부분 집합 S에 있는 도시들을 '한 번만' 방문했을 경우의 최단 경로라고 하자.
  - ➢ C(S, j) = 도시 1에서 부터, 도시 j까지 방문했을 경우의 최단 순회 경로
  - ➢ 점화식:



min

$$C(S, j) = \min_{i \in S : i \neq j} C(S - \{j\}, i) + d_{ij}$$



Traveling Salesman Problem

- 부분 문제들은 |S|에 의해서 정렬 됨
- 최대 n * 2n개의 부분 문제들이 존재할 수 있음

$C(\{1\},1) = 0$
for $s = 2$ to n
  for all subsets $S \subseteq \{1,2,\ldots,n\}$ of size s and containing 1
    $C(S,1) = \infty$
    for all $j \in S, j \neq 1$
      $C(S,j) = \min\{C(S-\{j\},i) + d_{ij} : i \in S, i \neq j\}$
return $\min_j C(\{1,\ldots,n\},j) + d_{j1}$

기타 동적프로그래밍 알고리즘

- 1, 2, 3 더하기 4: BOJ 15989 (https://www.acmicpc.net/problem/15989)
- 점프: BOJ 1890 (https://www.acmicpc.net/problem/1890)
- 평범한 배낭: BOJ 12865 (https://www.acmicpc.net/problem/1890)
- 출근 기록: BOJ 14238 (https://www.acmicpc.net/problem/14238)
- 뮤탈리스크: BOJ 12869 (https://www.acmicpc.net/problem/12869)
- 사회망 서비스(SNS): BOJ 2533 (https://www.acmicpc.net/problem/2533)
- 합리적인 이동경로: BOJ 2176 (https://www.acmicpc.net/problem/2176)
- 괄호: BOJ 10422 (https://www.acmicpc.net/problem/10422)