SAMSUNG 삼성SDS

SW 아카데미 알고리즘 특강

Contents

오늘의 원리

- 동적 계획법(Dynamic Programming)은 주어진 문제를 해결하기 위해 여러 개의 **하위 문제로 나누어 푼 다음 결합**하여 답을 찾는 방법이다.
- 문제 해결을 위해서는 다양한 방법으로 하위 문제로 나누어 보고 주어진 시간 복잡도 내에서 수행이 가능한 **최적의 점화식을 찾는 것이 핵심**이다.
- 동적 계획법은 그 자체 만으로 다양한 난이도의 문제를 해결할 수 있는 솔루션을 제시하기도 하지만, 다익스트라, TSP 등과 같은 **다양한 알고리즘들의 답을 구하는 과정에서도 빈번하게 활용**되고 있다.

학습목표

- 주어진 문제를 부분문제로 나누고 **점화식과 초기값을 구할 수 있다.**
- Memoization을 활용한 재귀방식과 결과값의 **순서를 정한 루프방식으로 동적계획법을 구현**할 수 있다.

사전 문제

1. 도둑이 N 가지의 보석이 무한히 있는 가게에 들어와 W kg을 넣을 수 있는 배낭에 보석을 넣어 훔쳐갈 때, 훔쳐가는 보석들의 가치의 합의 최대는 얼마이며, 어떤 보석을 얼마나 가져가야 하는가?
   - Case1 : N≤10, W≤100 인 경우
   - Case2 : N≤100, W≤10,000 인 경우
   - Case3 : 각 N개의 보석이 1개씩만 있는 경우

사전 문제

1. 탐욕법으로 접근하는 경우 주어지는 각 보석의 무게 대비 가치를 구하여 가장 효율이 좋은 보석을 선택해 볼 수 있으나, 효율이 좋은 큰 무게의 보석으로 배낭을 채우고 남는 공간이 애매하여 더 넣지 못하는 경우 보다 조금 효율이 좋지 않은 보석을 섞어 가져가서 배낭의 빈 공간을 채우는 것이 더 좋을 수 있다.
=> 보석을 쪼개서 가져갈 수 있다면 효율이 가장 좋은 보석부터 채우는 것이 최적해이지만 이 문제에서는 보석을 쪼갤 수 있는 조건이 없다.

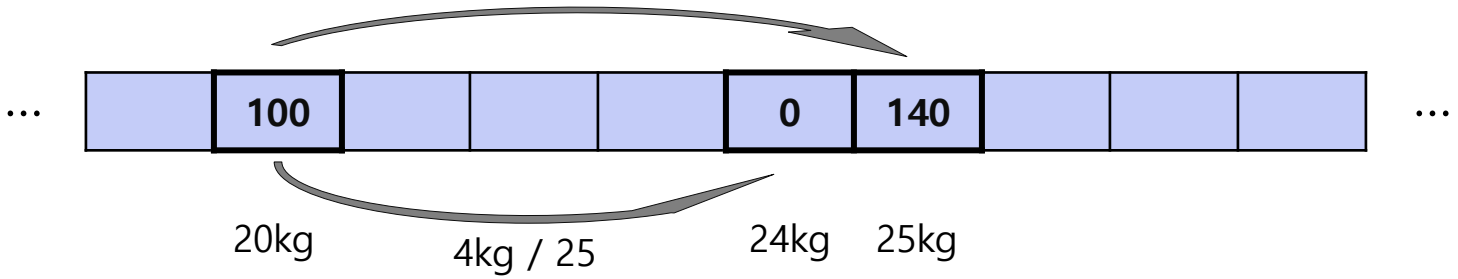
2. 탐욕법으로 접근하지 못하는 경우 조합을 생각해 볼 수 있다.
모든 보석 N가지에 대하여 각각 1.2....K개를 가져가는 경우별로 조합을 만들어본다.
=> 조합하는 경우 지수형태의 시간복잡도를 가지기 때문에 Case2를 만족하지 못한다.

3. 가져갈 수 있는 각 보석의 수가 무한하지 않고 1개씩만 있다면 보석을 선택할 때 알아야 할 조건들이 더 많아진다.

빠른 시간에 문제를 해결할 수 있는 방법을 고민해보자.

사전 문제

이 문제를 **백트래킹으로 접근하는 경우 모든 조합을 검토하게 되지만 중복되는 계산이 많다**는 것을 생각해 볼 수 있다. 아래와 같이 배낭에 20kg 채웠을 때 가져갈 수 있는 보석들의 최대 가치 합이 100이라고 하면, 여기서 5kg에 가치가 30인 보석을 넣는 경우와 4kg에 가치가 25인 보석을 넣는 경우 등 보석의 가지 수 만큼의 경우의 수가 발생하게 된다. **20kg을 만드는 조합이 무수히 많았지만 우리는 최종 결과 최대값 100를 취하여 이 값에 대해서만 새로운 값을 계산**해 냄으로 써 중복된 작업을 최소화 할 수 있다.



위 그림에서와 같이 20kg에서 4kg에 가치 25짜리 보석을 넣어서 24kg에 값이 없으면 125로 갱신이 될 것이고, 5kg에 가치 30짜리 보석을 넣어서 25kg에 130의 가치를 만들었으나 이미 140라는 더 큰 값이 있으면 갱신하지 않는다.

이렇게 수행하는 경우 O(WN)의 시간복잡도를 가지게 된다.

사전 문제

이 문제에서 최대의 가치뿐만 아니라 어떤 보석을 몇 개 넣어야 하는 지를 알기 위해서는 추가적인 정보를 저장해야 한다.

설계한 부분문제는 D[i] 를 채우는 것이며, D[i]는 무게 i kg의 가방을 채웠을 때 얻을 수 있는 가장 높은 가치의 합을 저장해 놓는 것이다. 하지만 이렇게 완성된 정보로는 어떤 보석들을 사용하였는지는 알 수가 없다. 하지만 D[i]를 구하며 i kg을 만들 때 가장 마지막에 어떤 보석을 넣었는지 알 수 있으므로, 그 정보를 저장하여 답을 구할 수 있다.

번호	무게	가치
1	2	11
2	3	16
3	4	26

	1	2	3	4	5	6	7	8	9	10
D[i]	0	11	16	26	27	37	42	52	53	63
P[i]	0	1	2	3	1	1	2	3	1	1

위와 같은 3개의 보석으로 10 kg짜리 배낭을 채울 때 얻을 수 있는 최대 가치 63은 D[i] 배열을 이용하여 구할 수 있고, 그 때 사용되는 보석은 P[i] 배열을 이용하여 구할 수 있으며 위의 경우 (2kg/11), (4kg/26), (4kg/26) 3개의 보석을 넣으면 최대 가치를 만들 수 있다.

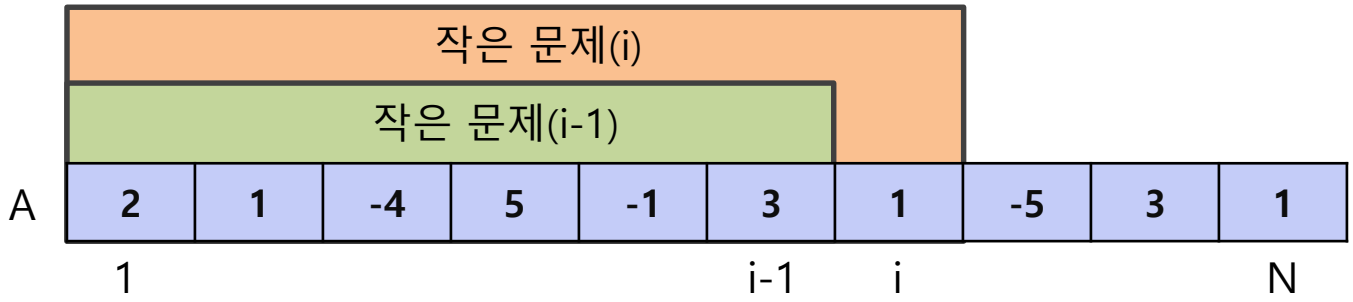
부분문제와 점화식

주어진 수열에서 연속된 구간의 합이 최대가 되는 값을 찾는다고 가정해 보자.

- **문제 나누기**

아래 그림과 같이 N 개의 전체 수열에서 1~i 구간만 취하여 **i번째 숫자를 포함하는 연속된 구간의 최대 합을 부분문제로 정의**할 수 있다.

그러면 이전 작은 문제(i-1)에서 i 번째 수열값을 더한 값 또는 이전 작은 문제(i-1)가 음수인 경우 i 번째의 값만 취한 결과가 현재 작은 문제(i)의 답이 된다는 걸 알 수 있다.



- **점화식: 작은 문제의 결과에서 큰 문제의 결과를 계산하기 위한 관계식**

부분문제로 나눈 결과를 구현하기 위하여 수식으로 나타내면 아래와 같다.

$$D_i = max(A_i + D_{i-1}, A_i)$$

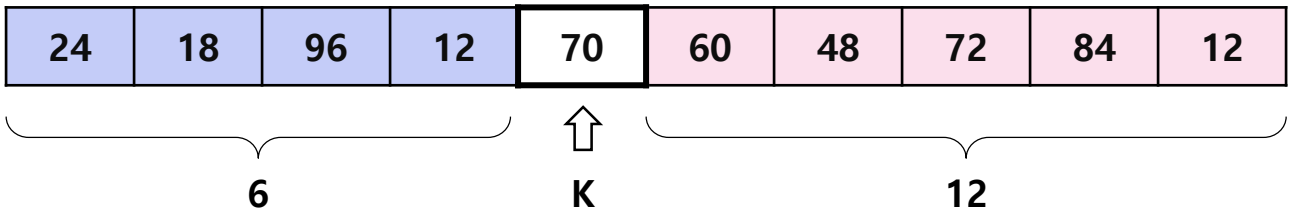
점화식 연습

1. 길이 N인 수열이 주어질 때 하나를 제외 한 나머지 숫자들의 최대공약수가 최대가 되는 값을 구하는 점화식을 설계하시오.

2. 최장증가수열(Longest increasing subsequence)을 구하는 점화식을 설계하시오.
   최장증가수열이란 길이 N인 수열에서 앞에부터 뒤로 숫자를 증가하는 순서대로 선택해 나갈 때, 최대 길이를 갖는 수열이다. 또한 최장증가수열을 이루는 수열을 구하기 위한 방법도 설계하시오.

최대공약수를 최대로 접근법

최대 공약수 역시 곱셈과 마찬가지로 교환법칙과 결합법칙이 성립하기 때문에 주어진 수열에서 하나를 뺀 나머지 숫자들에 대해 아래 그림과 같이 **빠지는 숫자를 기준으로 좌·우측 구간에 대한 계산 결과값을 활용**하여 구할 수 있다.



※ 최대공약수는 유클리드호제법을 이용하여 계산한다.

유클리드호제법 : 2개의 자연수(또는 정식) a, b에 대해서 a를 b로 나눈 나머지를 r이라 하면(단, a>b), a와 b의 최대공약수는 b와 r의 최대공약수와 같다.

연습문제

길이 N인 수열이 주어질 때 하나를 제외 한 나머지 숫자들의 최대공약수가 최대가 되는 값을 구하는 점화식을 설계하시오.

⇒ A라는 수열이 주어질 때 좌측영역의 최대공약수(이하 gcd)를 D1, 우측영역의 gcd를 D2라고 정의하면 이 영역들의 gcd를 계산하는 점화식은 아래와 같다.

$$D1_i = \gcd(D1_{i-1}, A_i)$$

$$D2_i = \gcd(D2_{i+1}, A_i)$$

최종적으로 구하고자 하는 답은 **N번째 위치를 제외한 좌우영역 D1과 D2 값을 조합한 값들의 최대값**과 같다.

$$\boldsymbol{max(\{D2_2, gcd(D1_1, D2_3), \cdots, gcd(D1_{N-2}, D2_N), D1_{N-1}\})}$$

※ 위와 같은 점화식을 도출하기 위해서는 아래 식을 증명할 수 있어야 한다.

$$\gcd(\{a_1, a_1, \cdots, a_{i-1}, a_i\}) = \gcd(\gcd(\{a_1, a_1, \cdots, a_{i-1}\}), a_i)$$

$$\gcd(\{a_1, a_1, \cdots, a_k, \cdots, a_i\}) = \gcd\big(\gcd(\{a_1, a_1, \cdots, a_k\}), \gcd(\{a_{k+1}, \cdots, a_i\})\big) \ (k < i)$$

최장증가수열 접근법

아래와 같은 수열이 주어졌을 때 빨간 체크 부분이 길이 4인 최장 증가수열이 되며 답은 유일하지 않을 수 있다.

수열

12	5	14	3	19	4	6	12	5	8

12	5	14	3	19	4	6	12	5	8

수열이 주어질 때 동적계획법의 가장 일반적인 순서는 처음부터 N까지 순서대로 수행하는 것이다.
이 문제를 부분문제로 나누기 위하여 i번째 위치에서 답 즉 **D_i에 어떤 값을 저장해야 하는지를 정의하고 이를 D_k (1 ≤ k ≤ i-1) 값들을 사용하여 계산 가능한지를 확인**해야 한다.
만약, 계산이 불가능 하다면 동적계획법의 순서 또는 저장해야하는 값을 바꿔가며 생각해 본다.

수열

12	5	14	3	19	4	6	12	5	8

i

만약 이 문제에서 D_i 값을 단순히 1 ~ i 까지 수열의 최장증가길이로 정의하면 부분문제를 통해 전체문제의 답을 계산하기가 힘들다.

연습문제

2. 최장증가수열(Longest increasing subsequence)을 구하는 점화식을 설계하시오.
   최장증가수열이란 길이 N인 수열에서 앞에부터 뒤로 숫자를 증가하는 순서대로 선택해 나갈 때, 최대 길이를 갖는 수열이다. 또한 최장증가수열을 이루는 수열을 구하기 위한 방법도 설계하시오.

⇒ A라는 수열에서 임의의 지점 i를 선택하고, 처음부터 i까지의 부분수열을 만들었을 때 **i를 포함하는 최장증가수열의 길이를 부분문제로 정의**한다.
즉, i 이전 지점 중 i지점의 값보다 작은 지점 중 최대값을 취하게 되면 동적계획법으로 풀이가 가능하며 점화식은 아래와 같다.

$$D_i = \max(\{D_k\}) + 1 \qquad (k \in \{1, 2, \cdots, i-2, i-1\},\ A_i > A_k)$$

연습문제

⇒ 최장 증가 수열에 어떤 원소들이 들어갈 수 있는지는 동적계획법의 구조를 잘 파악하고 추가적인 정보를 구성하여 구할 수 있다. 앞에서 정의한 부분문제는 i를 포함하는 최장증가수열의 길이로 정의 되어 있으므로 i보다 앞서 있는 위치 중 가장 큰 부분문제의 값을 선택하는 방법으로 구현되어 있다. 그러므로 i앞에는 가장 큰 부부분문제의 최적해가 k번째 수일 때, i앞에 k번째 수가 있어야 하는 정보를 구성하면, 최장 길이를 구성하는 수열의 요소를 뒤에서 부터 찾을 수 있다.

idx	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A[]	12	5	14	3	19	4	6	12	5	8
D[]	1	1	2	1	3	2	3	4	3	4
P[]	-1	-1	1	-1	3	4	6	7	6	9(7)

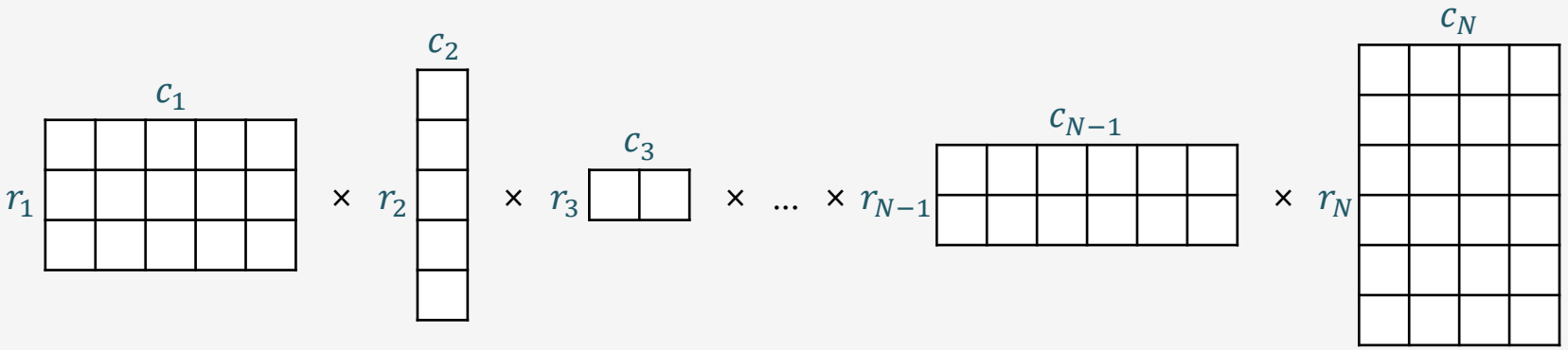
* P[i] : i번째 수 앞에 있어야 하는 수의 index 번호

※ 최장증가수열은 동적계획법의 경우 $O(N^2)$의 시간복잡도를 가지지만 바이너리서치 또는 인덱스트리를 사용하여 O(NlogN)에도 문제를 해결할 수 있다. 또한 최장증가수열의 요소를 찾는 것도 위와 같은 원리로 추가적인 정보를 구성하여 해결 할 수 있다.

행렬곱 접근법 (사선 DP (구간 DP))

1. N개의 행렬이 주어지고, 순서대로 곱셈을 한다고 할 때, 곱셈 연산의 최소값을 구하는 점화식을 설계하시오.

$$\text{연산 횟수 } (A \times B) = r_A \times c_A \times c_B$$
$$= r_A \times r_B \times c_B \quad * c_A = r_B$$



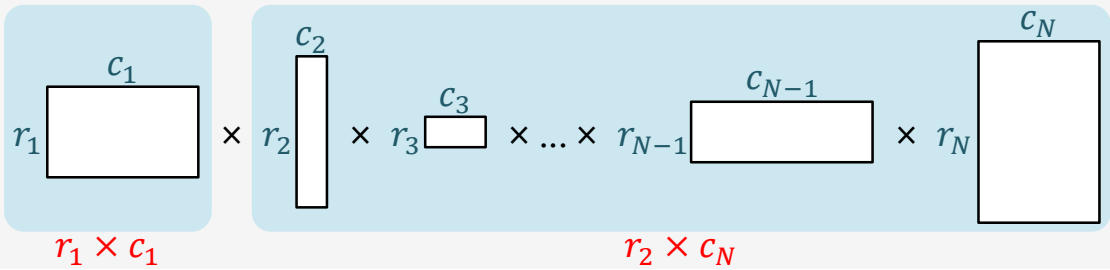
r_i : i번째 행렬의 행의 크기
c_i : i번째 행렬의 열의 크기

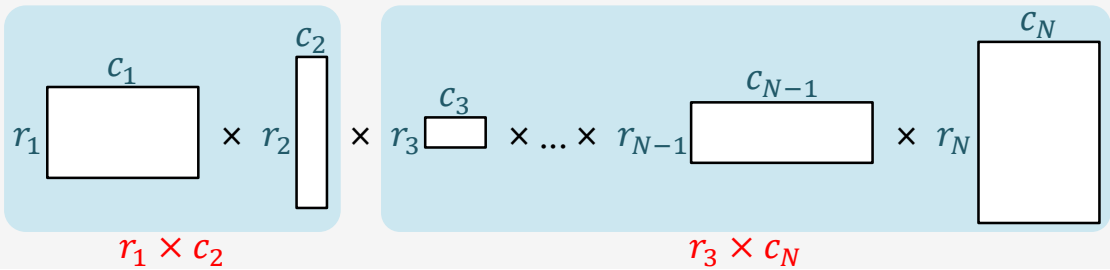
행렬곱 접근법 (사선 DP (구간 DP))

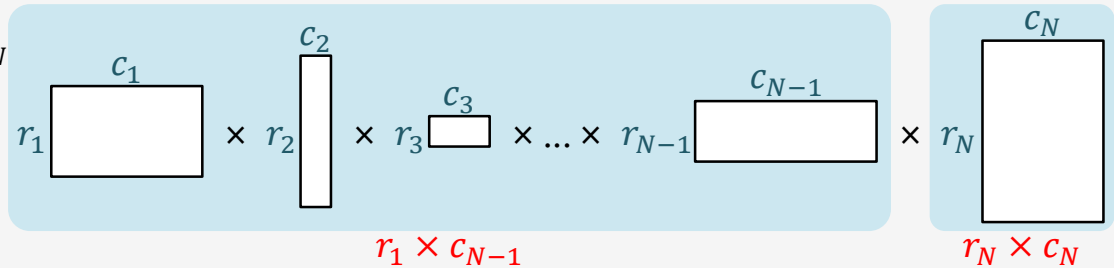
어떤 순서로 연산을 하든 크기가 $r_1 \times c_N$인 행렬이 된다는 성질을 이용해 문제를 해결할 수 있다.

D_{se} : s번째 행렬부터 e번째 행렬까지 곱하는데 필요한 최소 연산 횟수 $(s \le e)$

$$D_{1N} = min \begin{cases} D_{11} + D_{2N} + r_1 \times c_1 \times c_N \\ D_{12} + D_{3N} + r_1 \times c_2 \times c_N \\ \vdots \\ D_{1(N-1)} + D_{NN} + r_1 \times c_{N-1} \times c_N \end{cases}$$







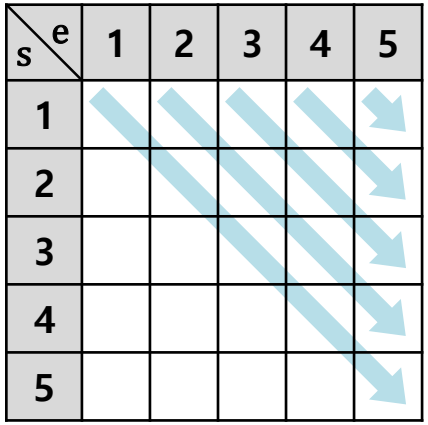
행렬곱 접근법 (사선 DP (구간 DP))

$$D_{se} = \min\left(D_{sk} + D_{(k+1)e} + r_s \times c_k \times c_e\right) \quad \therefore s \le k < e$$

위 점화식은 구간의 크기가 작은 것부터 채워줘야 한다.

$$D_{j(j+i-1)} = \min\left(D_{jk} + D_{(k+1)(j+i-1)} + r_j \times c_k \times c_{j+i-1}\right) \quad \therefore j \le k < j + i - 1$$

i : 구간의 크기
j : 구간의 시작 위치



외판원 순회

외판원 순회 문제는 영어로 Traveling Salesman problem (TSP) 라고 불리는 문제로 computer science 분야에서 가장 중요하게 취급되는 문제 중 하나이다. 여러 가지 변종 문제가 있으나, 여기서는 가장 일반적인 형태의 문제를 살펴보자.

1번부터 N번까지 번호가 매겨져 있는 도시들이 있고, 도시들 사이에는 길이 있다. (길이 없을 수도 있다) 이제 한 외판원이 어느 한 도시에서 출발해 N개의 도시를 모두 거쳐 다시 원래의 도시로 돌아오는 순회 여행 경로를 계획하려고 한다. 단, 한 번 갔던 도시로는 다시 갈 수 없다. (맨 마지막에 여행을 출발했던 도시로 돌아오는 것은 예외) 이런 여행 경로는 여러 가지가 있을 수 있는데, 가장 적은 비용을 들이는 여행 계획을 세우고자 한다.

각 도시간에 이동하는데 드는 비용은 행렬 W[i][j]형태로 주어진다. W[i][j]는 도시 i에서 도시 j로 가기 위한 비용을 나타낸다. 비용은 대칭적이지 않다. 즉, W[i][j] 는 W[j][i]와 다를 수 있다. 모든 도시간의 비용은 양의 정수이다. W[i][i]는 항상 0이다. 경우에 따라서 도시 i에서 도시 j로 갈 수 없는 경우도 있으며 이럴 경우 W[i][j]=0이라고 하자.

N과 비용 행렬이 주어졌을 때, 가장 적은 비용을 들이는 외판원의 순회 여행 경로를 구하는 프로그램을 작성하시오

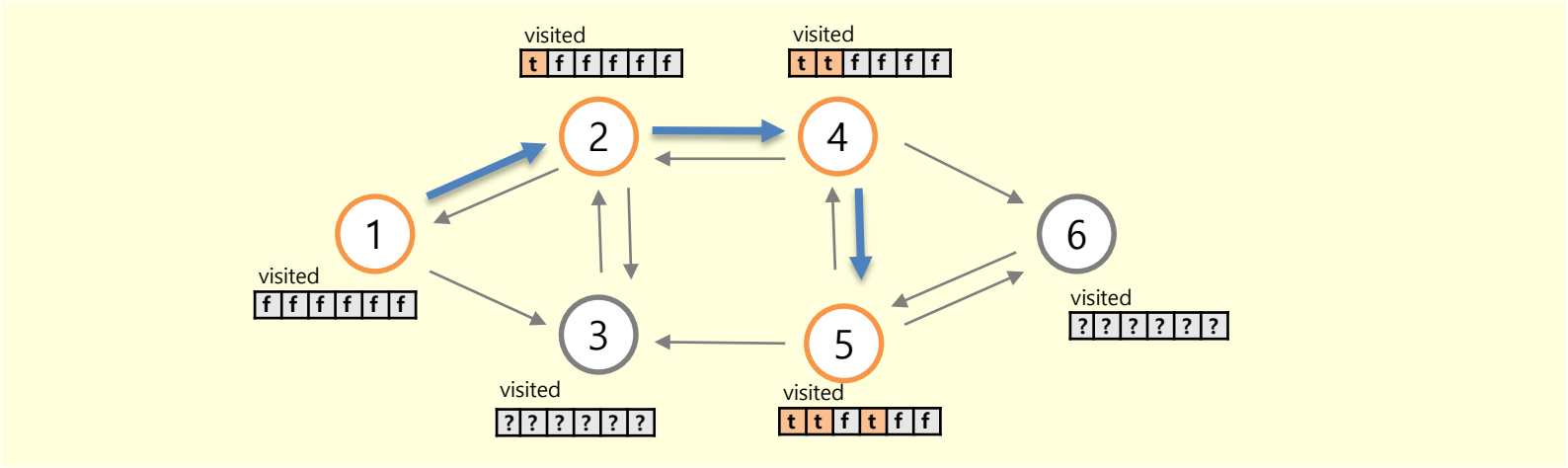
외판원 순회

(1) 완전탐색으로 푸는 방법을 생각해보자. 그리고 이 때의 시간복잡도를 계산해보자.

(2) 점화식을 세워보자.

실전문제

(1) 완전 탐색하는 방법

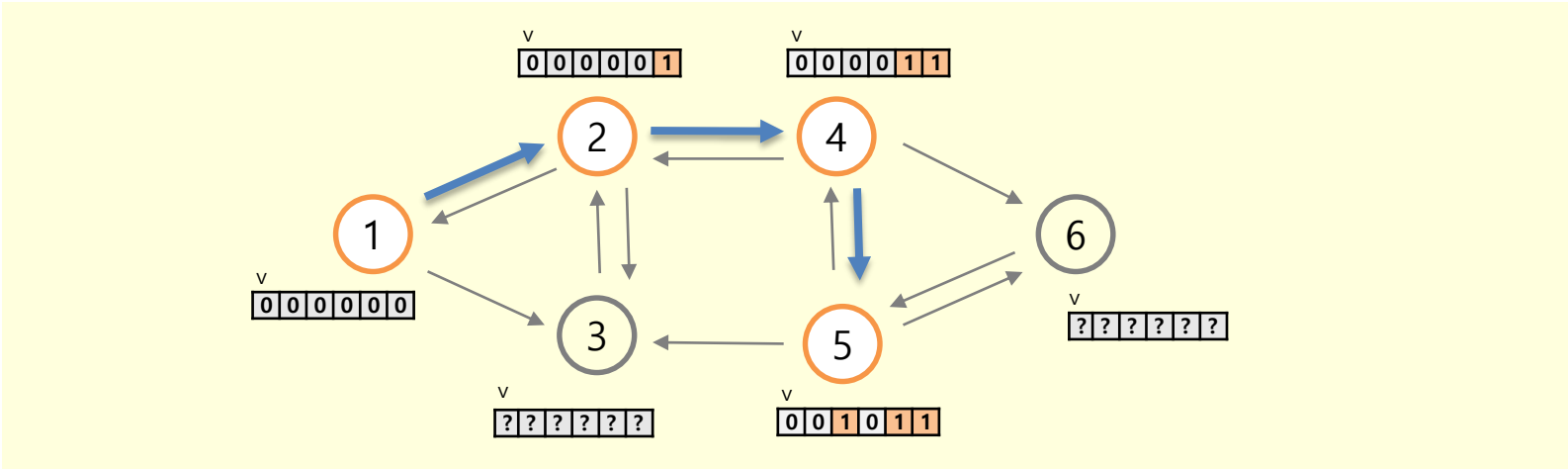


TSP 문제는 모든 도시를 단 한번씩 순회하며 모두 방문해야 한다. 그렇기 때문에 특정 도시까지 방문을 했을 때 그 전에 방문했던 도시의 목록을 가지고 있어야 한다. 특정 도시에서 연결된 도시 중에서 그전에 방문했던 도시를 제외해야 하기 때문이다. 이를 위해서는 false 값으로 초기화된 visited[N] 배열을 사용하여 도시를 방문할 때 마다 해당 도시의 visited값을 true 로 만들어주고, 다음에 방문할 도시로 visited값이 false 인 도시로 탐색하는 방법을 생각할 수 있다.

시간복잡도 : 모든 도시가 서로 연결되어있어 어느 도시로든 방문할 수 있다고 가정할 때, 처음에 방문할 수 있는 도시는 총 N 개, 그 다음 방문할 수 있는 도시는 N-1개, … , 마지막으로 방문할 수 있는 도시는 1개, 그 다음 처음에 방문한 도시로 돌아온다. 이 때 모든 경우의 수는 N! 이 될 것이다.

실전문제

(2) 점화식 세우기



앞에서 도시를 방문할 때마다 들고 다녔던 visited 배열을 비트마스크 기법을 사용하여 숫자로 나타내어 보자. 비트마스크란, 값을 비트단위로 계산하여 숫자로 나타내는 것이다. visited 된 도시 정보를 저장하는 숫자를 v라 할 때 i 번 도시를 방문했을 경우, 2^{i-1} 의 값을 v에 더하는 것만으로도 이미 방문한 도시를 숫자로 나타낼 수 있다. 예를 들어 1번, 2번, 4번 도시를 방문했다는 것을 v로 표현하면 1번 도시방문여부를 2^0으로, 2번 도시 방문여부를 2^1으로, 4번 도시 방문여부를 2^3 으로 계산하게 되어 v = $001011_{(2)}$ = 11 이 된다.

실전문제

(2) 점화식 세우기

nexti : 다음 방문할 도시 숫자
v : 현재 도시를 방문하기 까지 이미 방문한 도시들의 비트마스크 값

위와 같이 정의할 때 아래와 같은 점화식을 만들 수 있다.

D[v][nexti]
= min(D[v 에서 도시 i 를 방문하지 않은 비트마스크 값][i] + W[i][nexti])

이 때의 시간복잡도는 $2^N * N^2$ 이 될 것이다.

폐지 두 번 줍기 문제

M x N 격자로 이루어진 도시가 있다. 이 도시 군데군데에는 폐지가 버려져 있다.

범수는 가장 왼쪽 위 격자 (1, 1)에서 출발하여 가장 오른쪽 아래 격자 (M, N)까지 이동하며 폐지를 줍는데, 최단 경로를 따라가야만 한다. 즉, 인접한 오른쪽 칸 또는 아래쪽 칸으로만 이동할 수 있다.

여기에 더해, (M, N)에서 다시 (1, 1)으로 이동하면서 폐지를 더 줍는다. 역시 최단 경로를 따라가야 한다. 즉, 인접한 왼쪽 칸 또는 위쪽 칸으로만 이동할 수 있다.
폐지를 한번 주우면 없어진다는 것에 유의하자.

이 때, 범수가 수집할 수 있는 폐지의 최대값을 출력하시오.

[입력]
첫 줄에는 테스트 케이스의 수를 나타내는 T($1 \le T \le 20$)가 주어진다.
각 테스트 케이스의 첫 줄에는 N, M($2 \le N, M \le 100$)이 주어진다. 다음 M줄에 N개의 문자가 주어진다.
'*'는 갈 수 있으며 폐지가 있는 곳을 나타낸다.
'.'는 갈 수 있지만 폐지가 없는 곳을 나타낸다.
'#'는 갈 수 없는 곳을 나타낸다.
(1, 1)과 (M, N)은 갈 수 있는 곳임이 보장된다.
또, 이 두 점 사이에 경로가 있음도 보장된다.

[출력]
각 테스트 케이스에 대해, 범수가 주울 수 있는 최대 폐지 수를 한 줄에 하나씩 출력한다.

실전문제 접근법

먼저 좀더 간단한 폐지를 한 번 줍는 경우를 생각해 보자.

폐지를 한 번만 줍는 경우 특정 지점(i,j)에서 최대값은 이전 지점(즉 좌측 아니면 상단)들 중 최대값에서 현재 지점에 폐지가 있는지 여부에 따라 1을 더해줌으로써 계산할 수 있다.

$$D_{ij} = \max(D_{i-1j}, D_{ij-1}) + A_{ij}$$

이제 폐지를 두 번 줍는 경우를 생각해 보자.

폐지를 두 번 줍는 경우 첫 번째 폐지를 줍는 경로가 두 번째 경로에 영향을 주기 때문에 동적계획법이 쉬워 보이지 않으며 오히려 완전탐색의 문제처럼 보이기도 한다. 하지만 동적계획법으로 이 문제를 해결하기 위해 주요 아이디어를 먼저 정리해 보자.

1. (1,1)에서 출발하여 (M,N)에 도착한 뒤 다시 (1,1)로 돌아가는 경로에서 줍는 폐지의 양은 (1,1)에서 두명이 출발하여 (M,N)에 도달하며 줍는 폐지의 양을 계산한 것과 동일하다.
2. 두 명(이하 A와 B)이 출발하여 폐지를 줍게되면 A는 항상 B와 같거나 더 좌측의 열에 있다고 생각해도 된다. A가 B와 교차하여 반대편으로 넘어가서 진행하는 경우와 동일한 결과를 가지는 넘어가지 않고 진행하는 경로가 항상 존재한다.
3. A, B에 대해 A가 (i1,j1) B가 (i2,j2)에 있는 경우를 생각해 보자.
   A가 B보다 먼저(행이 큰 경우) 진행하고 있는 경우에는 점화식이 아래와 같이 될 것이다.

$$D_{i1j1i2j2} = \max(\{D_{i1-1j1i2j2}, D_{i1j1-1i2j2}\}) + A_{i1j1} \ (j1 < j2)$$

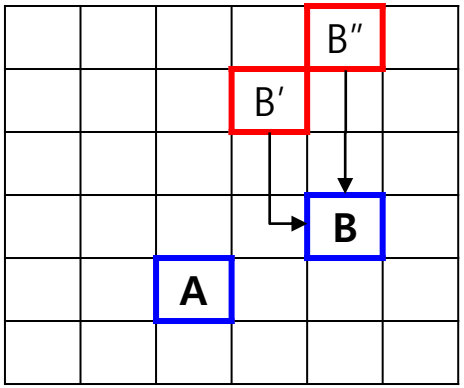
   단, 이 점화식은 N^4의 시간복잡도를 가지기 때문에 주어진 시간에 문제를 해결할 수가 없다.

실전문제

이 문제를 시간 내에 해결하기 위하여 아래와 같이 구한 점화식을 변경하여 최적화해야 한다.

$$D_{i1j1i2j2} = \max(\{D_{i1-1j1i2j2}, D_{i1j1-1i2j2}\}) + A_{i1j1}\ (j1 < j2)$$

위 점화식은 A의 모든 좌표에 대해 각각 B의 모든 좌표를 대응시켜 값을 가지고 있는 형태이다. 즉, 아래 그림과 같이 임의의 A지점 하나에 대해 B′, B″, B를 비롯한 모든 대응되는 B 지점에 대해 최대값을 구하고 있다. 결국 B′이나 B″의 경우 이동하여 B지점을 지나게 되기 때문에(물론 정답 경로에 B가 있는 경우) A지점에 B′이나 B″을 대응시켜 계산을 할 필요가 없다는 것을 알 수 있다.



이 아이디어를 가지고 A와 B의 이동횟수가 (1,1)에서 (M,N)으로 갈 때 M+N-2로 동일하다는 점을 적용하면 점화식을 개선할 수 있다. 이동횟수를 m, A의 행 위치를 a, B의 행 위치를 b라고 하면 점화식은 아래와 같다. (단, a와 b가 같은 경우 A,B가 동일한 위치에 있기 때문에 A는 한번만 더해준다.)

$$D_{mab} = \max(\{D_{(m-1)ab}, D_{(m-1)(a-1)b}, D_{(m-1)a(b-1)}, D_{(m-1)(a-1)(b-1)}\}) + A_{a(m-a+2)} + A_{b(m-b+2)}$$

Summary

- 문제를 여러 방법으로 부분 문제로 나누어 본다.
- 문제를 나누어 보았으면 차원을 결정하고 점화식을 세워 이전 값을 통해 새로운 값을 만들 수 있는지 검증 하고 초기값과 경계값을 정의 한다.
- 필요한 경우 논리적인 오류가 없는지 증명을 통해 검증해야 하지만 반드시 수학적인 공식을 사용 할 필요는 없다.
- 구현 이전에 반드시 점화식의 시간복잡도를 계산하여 문제에서 요구하는 시간을 만족하는 지 확인해 보아야 한다.
- 불필요한 요소가 있는지 검토하고 가능한 경우 점화식의 차원 또는 계산과정을 최적화 한다.