가. 초등부

(1) 딱지놀이(card)

A와 B가 딱지치기를 하는데, 각 라운드별로 승부의 결과를 구하는 문제이다. 딱지의 강력함에 대한 규칙은 별, 동그라미, 네모, 세모 순서로 개수를 비교하여 더 많은 쪽이 강력한 것이다. 따라서 각 라운드마다 4가지 모양의 개수를 카운트하고, 별 모양부터 비교하여 더 많은 쪽을 출력한다. 만약 같다면, 동그라미, 네모, 세로 순서로 개수를 비교한다. 각 모양의 개수가 모두 같다면, 비긴다는 의미인 'D'를 출력하면 된다.

(2) 방 배정하기(room)

수용인원이 다른 세 종류의 방이 주어졌을 때, 주어진 학생 수(N)와 정확히 일치하도록 숙소 배정이 가능한지 판단하는 문제이다.

부문문제1은 주어진 예제가 입력으로 들어온다. 주어진 입력 예제와 일치하는 출력 예제를 출력하면 된다.

부분문제2는 첫 번째 방의 수용 인원이 1명이다. 주어진 N 만큼 첫 번째 방을 배정하면 되므로 어떤 입력이 들어와도 숙소를 배정할 수 있다.

부분문제3은 두 번째, 세 번째 방의 수용 인원이 첫 번째 방의 수용 인원의 배수이다. 이 경우, 첫 번째 방만 가지고 모든 학생을 수용할 수 있는지 판단해도 충분하다. N이 첫 번째 방의 수용 인원의 배수이면 숙소 배정이 가능하며, 그렇지 않다면 불가능하다.

부분문제4는 원래의 제약 조건 외에 아무 제약조건이 없다. 입력에 주어지는 어떠한 방도 수용 인원이 한 명이상이기 때문에 각 방은 최대 N개까지 가능하다. 이에 따라 각 방마다 0..N개라고 가정해보고 수용인원 합이 N과 일치하는지 확인하면 된다. 이 과정은 간단한 삼중 반복문을 통해 구현할 수 있으며, 총 시간복잡도는 $O(N^3)$이 된다.

(3) 서울에서 경산까지(travel)

서울에서 경산사이에 N-1개의 도시가 일렬로 있고, 도시 사이에 N개의 구간이 있다. 각 구간을 도보 혹은 자전거로 이동 가능하며, 이 때 두 방법의 소요 시간과 모금액이 다를 수 있다. 이 때, 제한시간 내에 서울에서 경산까지 이동하면서, 모금할 수 있는 금액을 최대로 하는 문제이다.

이 문제는 동적계획법으로 풀이가 가능하다. 다음과 같이 $D(i,j)$을 정의해보자.

$D(i,j)$: i번 도시에 j분에 도착했을 때, 모금액의 최댓값

아래는 다른 변수에 대한 정의이다.

- $T_{0,i}$: 구간i을 도보로 이동할 때, 걸리는 시간
- $C_{0,i}$: 구간i을 도보로 이동할 때, 모금액
- $T_{1,i}$: 구간i을 자전거로 이동할 때, 걸리는 시간
- $C_{1,i}$: 구간i을 자전거로 이동할 때, 모금액

$D(i,j)$의 값을 구하기 위해, 그 이전 상황을 살펴보도록 하자.

- 도보를 이용해서 이동한 경우(단, $j \geq T_{0,i}$): $D(i-1, j-T_{0,i}) + C_{0,i}$
- 자전거를 이용해서 이동한 경우(단, $g \geq T_{1,i}$): $D(i-1, j-T_{1,i}) + C_{1,i}$

위의 두 경우가 $D(i,j)$을 생성하는 바로 직전의 상태이다. 두 상태 중 더 큰 값(모금액)이 $D(i,j)$의 값이 된다.

초기에 서울(0번 도시)에서 출발하기 때문에 $D(0,0) = 0$이고, 그 외 시간에는 0의 위치에 있지 않기 때문에$D(0,j) = -\infty (j > 0)$으로 설정하면 된다. D배열의 전체 크기만큼의 시간이 소요되기 때문에 $O(NK)$의 시간복잡도와 공간복잡도를 갖는다.

(4) 줄 서기(line)

부분문제 1은 $N!$개의 모든 순열에 대해 정보가 일치하는지에 대한 검사를 통해 시간복잡도 $O(N! \times N^2)$ 에 풀 수 있다.

부분문제 2는 $i < j$, $j < k$이면 $i < k$ 라는 사실을 통해 가능한 모든 i, j, k 쌍에 대해 검사하여 $O(N^3)$에 해결할 수 있다.

부분문제 3의 경우, N^2개의 간선들을 모두 복원한 후 각 자리에 있는 숫자에 대해 그 수보다 큰 수의 개수를 세어 주면 $O(N^2)$ 에 해결할 수 있다.

부분문제 4의 풀이는 다음과 같다. M이 0인 기본 상태에서는 오른쪽에 있는 모든 수가 왼쪽에 있는 수보다 크기 때문에, 각 자리에 대해 자신보다 큰 수의 개수는 $N-1$, $N-2$, $\cdots$, 0이 된다. 그 후 입력으로 들어오는 X, Y 쌍에 대해 X보다 큰 수의 개수를 1 감소, Y보다 큰 수의 개수를 1 증가시켜준다. 모든 M개의 쌍에 대해 이를 시행하고 나면 각 수보다 큰 수가 몇 개인지 알 수 있고, 이를 통해 원래 수가 무엇인지 복원할 수 있고 $O(N+M)$ 에 해결할 수 있다.

입력이 성립하지 않는 경우에는 복원한 수 배열에 중복된 수가 존재하게 된다. 이 때는 -1을 출력해 주면 된다.

나. 중등부

(1) 방 배정하기(room)

수용인원이 다른 세 종류의 방이 주어졌을 때, 주어진 학생 수(N)와 정확히 일치하도록 숙소 배정이 가능한지 판단하는 문제이다.

부문문제1은 주어진 예제가 입력으로 들어온다. 주어진 입력 예제와 일치하는 출력 예제를 출력하면 된다.

부분문제2는 첫 번째 방의 수용 인원이 1명이다. 주어진 N 만큼 첫 번째 방을 배정하면 되므로 어떤 입력이 들어와도 숙소를 배정할 수 있다.

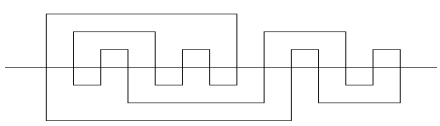
부분문제3은 두 번째, 세 번째 방의 수용 인원이 첫 번째 방의 수용 인원의 배수이다. 이 경우, 첫 번째 방만 가지고 모든 학생을 수용할 수 있는지 판단해도 충분하다. N이 첫 번째 방의 수용 인원의 배수이면 숙소 배정이 가능하며, 그렇지 않다면 불가능하다.

부분문제4는 원래의 제약 조건 외에 아무 제약조건이 없다. 입력에 주어지는 어떠한 방도 수용 인원이 한 명이상이기 때문에 각 방은 최대 N개까지 가능하다. 이에 따라 각 방마다 0..N개라고 가정해보고 수용인원 합이 N과 일치하는지 확인하면 된다. 이 과정은 간단한 삼중 반복문을 통해 구현할 수 있으며, 총 시간복잡도는 $O(N^3)$이 된다.

(2) 곡선(cut)

먼저 입력에서 주어진 직교다각형에서 어떤 봉우리가 생기는지를 알아내어야 한다. $y<0$인 점에서 $y>0$인 점으로 이동한 뒤, $y>0$인 점에서 $y<0$인 점으로 오게 되면 한 봉우리가 생기게 됨을 알 수 있다. 이 성질을 이용하면, $y<0$인 점을 하나 고른 뒤, 그 점으로부터 N개의 점으로 이동하면서 어떤 봉우리가 생기는 지 모두 알 수 있다.

$(a,0)$와 $(b,0)$을 지나는 봉우리($a<b$)를 a에서 괄호가 열리고('('), b에서 괄호가 닫힌(‘)’)다고 생각해보자. 그러면 주어진 입력들을 봉우리로 분리한 뒤, 괄호 문자열로 바꿀 수 있다. 예를 들어, 입력(1)을 괄호문자열로 바꾸면 “(()())”로 표현 할 수 있고, 입력(2)를 괄호문자열로 바꾸면 “()”로 표현 할 수 있다. 아래 예시는 14개의 괄호로 표현이 가능하다.



→ ((( )) ( )) (( )) ( )

N개의 점들을 괄호문자열로 바꾸게 되면, 최대 $N/2$길이의 괄호문자열이 생긴다는

것을 알 수 있고, 같은 이유로 최대 $N/4$ 괄호쌍이 생긴다. 그러면 이제 괄호 문자열에서

가장 바깥쪽에 있는 괄호 쌍의 개수와, 가장 안쪽에 있는 괄호 쌍의 개수를 구하는 문제로 바뀐다.

각 부분문제에 대한 해결법은 다음과 같다.

부분문제 1은 $N \leqq 1000$, $|X|,|Y| \leqq 1000$ 이다. 각 괄호쌍에 대해서 '('의 위치를 a, ')'의 위치를 b라고 하면 $[a,b]$에 depth를 모두 1 증가시켜 주자. 그 이후에는 가장 바깥쪽에 있는 괄호쌍은 depth가 1인 괄호쌍의 개수가 되고, 가장 안쪽에 있는 괄호는, $[a,b]$가 모두 같은 depth를 가지고 있는 괄호쌍의 개수를 세면된다. 이 경우 시간복잡도는 $O(NL)$이 된다.

부분문제 2는 $N \leqq 10000$ 이다. 그러면 최대 2500개의 괄호 쌍이 생긴다. 어떤 두 괄호쌍에 대해서, 포함관계에 있는지 없는지 판단할 수 있다. 이를 이용하면, 아무도 자신을 포함하지 않는 괄호 쌍의 개수가 가장 바깥쪽에 있는 괄호 쌍이 되고, 다른 어떤 괄호 쌍을 포함하지 않는 괄호 쌍의 개수가 가장 안쪽에 있는 괄호 쌍이 된다. 이 경우 시간복잡도는 $O(N^2)$가 된다. 다른 풀이로는 부분문제 1을 해결하는 방법에서, 괄호들의 위치를 번호를 다시 매겨서 L을 줄인 뒤, $O(N^2)$로 해결 할 수도 있다.

부분문제 3은 별다른 제한이 없다. 먼저 괄호문자열들을 위치로 정렬한다. 앞에서부터 차례대로 보면서 '('가 들어오면 level을 1만큼 올리고, ')'가 들어오면 level을 1만큼 감소시킨다. '('가 들어오는 순간에 level이 1이 되면 가장 바깥쪽에 있는 괄호라는 것을 의미하고. '()'가 연속하게 있으면 가장 안쪽 괄호인 것을 알 수 있다. 이를 이용하여 답을 출력할 수 있다. 이 경우 시간복잡도는 $O(N\log N)$가 된다. 다른 풀이로는 부분문제 1을 해결하는 방법에서, 괄호들의 위치를 번호를 다시 매긴 뒤, $[a,b]$에 depth를 1 증가시키는 것을 Index-tree나 BIT를 이용하여 해결하여도 $O(N\log N)$으로 해결 할 수 있다.

(3) 줄 서기(line)

부분문제 1은 $N!$개의 모든 순열에 대해 정보가 일치하는지에 대한 검사를 통해 시간복잡도 $O(N! \times N^2)$ 에 풀 수 있다.

부분문제 2는 $i<j$, $j<k$이면 $i<k$ 라는 사실을 통해 가능한 모든 i,j,k 쌍에 대해 검사하여 $O(N^3)$에 해결할 수 있다.

부분문제 3의 경우, N^2개의 간선들을 모두 복원한 후 각 자리에 있는 숫자에 대해 그 수보다 큰 수의 개수를 세어 주면 $O(N^2)$ 에 해결할 수 있다.

부분문제 4의 풀이는 다음과 같다. M이 0인 기본 상태에서는 오른쪽에 있는 모든 수가 왼쪽에 있는 수보다 크기 때문에, 각 자리에 대해 자신보다 큰 수의 개수는 $N-1$, $N-2$, $\cdots$, 0이 된다. 그 후 입력으로 들어오는 X, Y 쌍에 대해 X보다 큰 수의 개수를 1 감소, Y보다 큰 수의 개수를 1 증가시켜준다. 모든 M개의 쌍에 대해 이를 시행하고 나면 각 수보다 큰 수가 몇 개인지 알 수 있고, 이를 통해 원래 수가 무엇인지 복원할 수 있고 $O(N+M)$ 에 해결할 수 있다.

입력이 성립하지 않는 경우에는 복원한 수 배열에 중복된 수가 존재하게 된다. 이 때는 -1을 출력해 주면 된다.

(4) 고양이(cat)

부분문제 1의 경우 N제한이 5000이므로, 모든 정점을 지워 보고 남은 그래프에 사이클이 존재하는지 확인하는 것으로 해결할 수 있다. 사이클이 존재하는지 확인하는 것은 Disjoint set 또는 DFS, BFS 등을 이용해 O(N+M)의 시간복잡도에 해결할 수 있으므로, 전체 알고리즘의 시간복잡도는 O(N*(N+M))이 된다. 이 시간복잡도는 부분문제 1을 해결하는 데 충분하다.

부분문제 2의 경우 N = M이므로, 그래프 전체에 사이클 하나만이 존재하는 형태이다. 이 경우, 사이클에 속하는 임의의 정점 하나를 지우면 남은 그래프에 사이클이 없게 할 수 있으며, 그 외의 정점을 지우면 사이클을 제거할 수 없다. 따라서 사이클을 하나 찾은 뒤, 그 정점의 번호를 모두 합치면 정답을 찾을 수 있다. 사이클 하나를 찾는 것은 BFS 등을 사용해 O(N+M)의 시간복잡도로 해결할 수 있다.

부분문제 3의 경우 이미 모든 정점을 이루는 사이클이 존재하는 형태이다. 이제, 전체 사이클을 제외한 간선이 존재하는 경우와 존재하지 않는 정점으로 경우를 나눌 수 있다. 전체 사이클을 제외한 간선이 존재하지 않는 경우, 부분문제 2에 해당하므로 해결할 수 있다. 전체 사이클을 제외한 간선 a-b가 존재하는 경우, 정점 a 혹은 b를 제거하지 않는 경우 항상 사이클이 하나 이상 존재함을 확인할 수 있다. 따라서, 정답이 될 수 있는 정점을 두 개 이하로 줄일 수 있다. 정답이 될수 있는 정점들을 모두 지워보고, 사이클이 존재하는지 확인하면 O(N+M)의 시간복잡도로 부분문제 3을 해결할 수 있다.

부분문제 4의 경우 앞의 부분문제들의 관찰을 종합해 해결할 수 있다. 사이클이 하나만 존재하는 경우는 부분문제 2의 경우에 속하므로 해결할 수 있다. 서로 다른 사이클이 두 개 이상 존재하는 경우는 부분문제 3과 비슷하지만, 한 쪽의 사이클이 모든 정점을 포함하지 않으므로 약간 경우를 세밀하게 나누어야 한다. 두 사이클의 정점이 하나도 겹치지 않는 경우 하나의 정점으로 두 사이클을 모두 지울 수 없으므로

정답은 0이 된다. 두 사이클이 a -- c1 -- c2 -- ... -- b 간선들을 공유하는 경우, 부분문제 3과 비슷하게 정답이 될 수 있는 후보는 a와 b 정점뿐임을 확인할 수 있다. 따라서, 이 정점들을 모두 지워보고 사이클이 존재하는지 확인할 수 있다. 총 시간복잡도는 O(N+M)이 된다.

다. 고등부

(1) 물통(bottle)

두 물통의 용량(a, b)과 각 물통마다 목표로 하는 물의 용량(c, d)이 주어졌을 때, 목표를 달성하기 위한 최소 작업 횟수를 구하는 문제이다.

부분문제1은 a = 1이다. 두 물통의 상태를 (i, j)로 나타낼 때, (0, d)를 만드는 과정은 다음 같은 두 방법 중 하나이다.

1. $(0,0) \rightarrow (1,0) \rightarrow (0,1) \rightarrow (1,1) \rightarrow (0,2) \rightarrow \dots \rightarrow (1,x-1) \rightarrow (0,x)$
2. $(0,0)$ → (0, b) → (1, b-1) → (0, b-1) → ... → (1, x) → (0, x)

$x < b$인 경우, $(1,x)$ 또한 위 두 방법과 비슷하게 만들 수 있다. 두 방법 중 최소 횟수를 구하면 되며, 이를 식으로 나타내면 $\min(2d+c, 2b-2d-c+1)$이 된다.

예외적으로 (a, b)를 만들기 위해서는 2회 작업을 진행해야한다.

부분문제2는 b가 a의 배수이다. c와 d가 a의 배수이면 a, b, c, d에 각각 a를 나눈 후 부분문제1과 같이 풀면 된다. a의 배수가 아니면 불가능하다.

부분문제3은 a,b <= 1000 이다. 이 경우에는 다이나믹 프로그래밍 기법을 사용하여 풀 수 있다. 다음과 같이 $D_{i,j}$를 정의한다.

$$D_{i,j} = (i,j)\text{에서 } (c,d)\text{로 가기 위한 최소 조작 횟수}$$

$i = c$, $j = d$인 경우, 이미 목표 상태에 도달했으므로 필요한 조작 횟수는 0이다.

그 이외의 경우는 다음 조작을 고려한다.

- 용량이 a인 물통을 비운다. → $D_{0,j}$ + 1
- 용량이 a인 물통을 채운다. → $D_{a,j}$ + 1
- 용량이 b인 물통을 비운다. → $D_{i,0}$ + 1
- 용량이 b인 물통을 채운다. → $D_{i,b}$ + 1
- 용량이 a인 물통에서 b인 물통으로 물을 옮긴다. → $D_{i-\min(i,b-j),j+\min(i,b-j)}$ + 1
- 용량이 b인 물통에서 a인 물통으로 물을 옮긴다. → $D_{i+\min(a-i,j),j-\min(a-i,j)}$ + 1

위 6가지 경우 중 최솟값이 $D_{i,j}$가 된다.

이를 구현하기 위해서 초기에 $dp(c,d) = 0$, 나머지 $dp(i,j) = \infty$로 초기화 해놓고 $(0,0)$부터 검색을 한다. 이미 검색을 했던 (i,j)는 구해놓은 $dp(i,j)$를 참조하고, 그렇지 않

은 (i,j)에 대해서는 재귀호출을 이용해서 구한다.

부분문제4는 원래의 제약조건 이외에는 아무런 제약조건이 없다. 본 부분문제는 부분문제3에서 $dp(i,j)$를 다르게 정의하여 메모리를 절약하면 풀 수 있다. 항상 두 물통 중 하나는 완전히 채워져 있거나 비워져 있으므로 가능한 두 물통의 상태 수는 $O(n+m)$개이다. 따라서 부분문제3에서 $D_{i,j}$를 저장할 때 $O(n+m)$개의 메모리를 사용하도록 구현하면 본 문제를 풀 수 있다. 총 시간복잡도는 $O(n+m)$이다.

(2) 문명(civilization)

$N \times N$ 그리드에 K개의 격자가 색칠되어 주어지고, 각 단위시간마다 색칠된 격자의 인접한 격자들이 색칠될 때, 얼마의 단위 시간 후에 모든 색칠된 격자들이 하나로 결합되는지를 계산하는 문제이다.

부분문제 1번을 풀기 위한 해답은 다음과 같다.

각 단위 시간이 흐르기 이전에, 임의의 색칠된 격자로부터 DFS 혹은 BFS를 수행하여 모든 색칠된 격자를 방문하는지 확인한다. 만약 모든 색칠된 격자를 방문한다면 이미 격자들이 하나로 결합되어 있는 것이고, 일부를 방문하지 못했다면 단위 시간이 더 흘러야함을 알 수 있다. 1의 단위 시간이 흐른 이후에는 기존에 색칠된 격자들의 인접한 칸들도 색칠된 것으로 처리를 해주면 된다.

최대 N의 단위 시간이 흐를 수 있고, 탐색과 색칠에 $O(N^2)$의 시간이 흐르기 때문에 전체적으로 $O(N^3)$의 시간복잡도에 문제를 해결할 수 있다.

부분문제 2번을 풀기 위한 해답은 다음과 같다.

일정 시간이 흐른 이후에는 모든 격자들이 하나로 결합될 것이고, 그 결합된 상태가 유지될 것이기 때문에 답이 되는 시간을 결정해 볼 수 있다. 결정된 시간 t에 대해서 모든 색칠된 격자들이 하나로 결합되었다면 답은 t 이하인 것이고, 하나로 결합되지 않았다면 답은 t 초과인 것이다. 0에서 N 사이의 값을 가질 수 있는 t에 대해서 이분탐색을 시도할 수 있다.

t가 결정된 후에 모든 색칠된 격자들이 하나로 결합되어 있는지는 부분문제 1을 해결할 때와 비슷한 방법으로 확인할 수 있다. BFS를 수행한다고 생각한다면, 큐에 입력으로 주어진 K개의 문명 발상지 위치를 넣고 인접한 격자를 방문할 때에 거리를 1씩 늘려나가며 최단 거리를 기록한다. 최단 거리가 t이내일 때에만 탐색을 수행하여 탐색하였던 영역이 하나의 영역인지 확인하면 $O(N^2)$에 확인이 가능하다. 따라서 t를 결정하는 데에 $O(\log N)$이므로 전체적으로 $O(N^2 \log N)$이다.

부분문제 3번을 풀기 위한 해답은 다음과 같다.

각 단위 시간마다, 시간이 흐르기 이전과 이후에 대한 작업을 수행한다. 시간이 흐

르기 이전에는 인접한 색칠된 격자들을 서로소 집합 자료구조(Union & Find)를 활용하여 묶어준다. 시간이 흐른 이후에는 인접한 격자들 중 색칠이 되지 않은 격자들에 색칠을 한다. 이후에는 기존에 색칠되었던 격자들은 제외하고 새롭게 색칠한 격자들에 대해서만 고려할 수 있다.

서로소 집합 자료구조를 구현할 때에 경로 압축(Path Compression)을 해주어야 Union 할 때에 $O(N)$이 아니라, 평균적으로 상수 시간 만에 수행이 가능하다. 따라서 이를 구현한다면 전체적으로 $O(N^2)$에 문제를 해결할 수 있다.

(3) 요리 강좌(cook)

요리강좌 문제는 각 N개의 학원에서 M개의 강의가 존재하는데, 한 학원에서 연속된 강의를 S개 이상 수강한 이후에는 다른 학원의 강좌를 들을 수 있고, 한 학원에서 연속된 강의를 E개의 강의를 수강했을 때, 다음 강의는 무조건 다른 학원의 강의를 수강해야하는 문제이다. 그리고 각 학원에서는 경쟁관계가 있어서 불허용 직전학원이 존재한다. 불허용 직전학원이란, 불허용 직전학원에서 바로 이전 강의를 들은 학생은, 자신의 학원에 이번강의를 못 듣게 하는 제도이다. 또한 학원을 옮기는 데에는 T원의 비용이 발생한다.

이 문제는 동적계획법으로 문제를 해결할 수 있다.

$A(i,j)$, $X(i)$, $D(i,j)$, $RS(i,j)$ 배열을 다음과 같이 정의하자.

$A(i,j)$ = i번째 학원에서 j번째 강좌를 수강하는데 드는 비용

$X(i)$ = i번째 학원의 불허용 직전학원

$D(i,j)$ = $A(i,j)$ 강좌를 듣기 바로 직전에 다른 학원에 왔을 때 드는 최소비용

$RS(i,j)$ = 배열 $A(i)$의 뒤로부터의 누적합 $RS(i,j+1)+A(i,j)$

그럼 다음과 같은 점화식이 성립한다.

$D(i,0)=0$

$D(i,j)=\min(D(p,q)+RS(p,q)-RS(p,j))+T\,(1\le p\le n, j-e\le q\le j-s\,)\,(p\neq i, X(i))$

그 이후, $D(i,j)+RS(i,j)$ $(1\le i\le n, m-e+1\le j\le m)$ 중 최솟값을 택하면 해가 되고, $O(n^4)$으로 부분문제 1번을 해결할 수 있다.

여기서 각 p에 대해 $[j-e,j-s]$구간의 $D(p,q)$을 계속 참조하는 성질을 이용해서, $D(i,j)$를 채울 때, 각 p의 $[j-e,j-s]$구간의 n개의 최솟값을 각 j에 대해 저장해주고 참조하는 식으로 문제를 해결하면 $O(nm(n+m))$으로 부분문제 1, 2번을 해결할 수 있다.

여기서 각 i에 대해 자신으로 올 수 없는 p값이 최대 2개라는 성질을 이용해서 최솟

값을 3개를 저장해주는 식으로 처리하면 $O(nm^2)$의 시간복잡도로 부분문제 1,2,3번을 해결할 수 있다.

여기서 각 j가 $j+1$로 진행함에 따라 $[j-e,j-s]$ 구간이 $[j+1-e,j+1-s] = [j-e+1,j-s+1]$ 로 바뀌고, 모든 참조하는 값에서 더해지는 값이 A[p][j-s+1]로 동일함으로 그 내부의 대소 관계는 바뀌지 않고 추가되는 값은 $q=j-s+1$ 에 대한 값, 제거되는 값은 $q=j-e$에 대한 값이라는 성질을 이용해서 시간복잡도를 더 줄일 수 있다.

점화식을 다시 한 번 관찰하면

$D(i,j) = \min(D(p,q)+(RS(p,q)-RS(p,j)),\infty)+T$

에서 각 p,q에 대해 $(-RS(p,q))$ 는 공통이므로

$D(i,j)=\min((D(p,q)+RS(p,q))-RS(p,j),\infty)+T$

로 묶어서 생각할 수 있다. ($-RS(p,j)$ 를 따로 두면 각 p에 대해 최솟값을 가져갈 때, j에 구속을 받지 않는다.)

따라서 $D(p,q)+RS(p,q)$ 의 값보다 작은 deque의 위의 값을 다 빼주고 deque의 맨 앞에 있는 값의 위치가 $[j-e,j-s]$ 에 속하지 않는다면 pop을 해주는 식으로 deque를 관리해주면 deque의 맨 앞에 있는 값 $-RS(p,j)$는 각 p의 $[j-e,j-s]$ 구간의 최솟값이 된다. 여기서 위에서 설명한 최솟값을 3개 가져가는 식으로 처리하면 총 시간복잡도 $O(nm)$에 문제를 해결할 수 있다. 만약 최솟값 3개를 가져가지 않고 전체탐색을 한다면 총 시간복잡도가 $O(n^2m)$이 되므로 부분문제 1, 2, 4번을 해결할 수 있다.

(4) 조개 줍기(shell)

질의가 없는 문제의 경우 간단한 동적계획법을 이용하여 문제를 해결할 수 있다. 이는 매우 잘 알려진 문제이며, 간단한 형태로 점화식이 나타난다. 이를 이용하면 질의 하나를 $O(N^2)$의 시간복잡도로 해결할 수 있으며, 질의가 총 N개이므로 전체 시간복잡도 $O(N^3)$으로 1번 부분문제를 해결할 수 있다.

어떤 위치의 숫자가 하나 바뀔 때 일어나는 일에 대해 관찰하여 보자. 한 위치가 바뀔 때는 행 번호가 보다 크거나 같고, 열 번호가 보다 크거나 같은 위치의 값만 영향을 받는다는 것을 알 수 있다. 마찬가지로, 한 위치의 값이 영향을 받지 않으려면 이나 중 하나는 바뀌지 않았어야 한다. 따라서 영향을 받는 칸들은 모두 인접해 있을 수밖에 없으며, 추가적으로 값이 바뀌는 구간과 바뀌지 않는 구간의 경계는 항상 행과 열이 증가하는 방향으로 나타난다.

부분문제 2의 경우 초기상태의 답이 매우 작고, 몇 번의 질의를 받아도 답이 0보다 작아질 수 없다. 이를 이용하면 값이 바뀌는 위치의 칸을 모두 찾은 뒤, 값을 1 감소

시키는 방식을 사용하면 답을 구할 수 있다.

행 번호가 증가하는 방향으로 경계를 관찰하면, 영향을 받는 구간의 시작점과 끝점 모두 열 번호가 증가하는 방향으로 이동한다는 것을 알 수 있으며, 이를 이용하여 슬라이딩 윈도우 기법과 유사한 방식으로 문제를 해결할 수 있다. 질의의 개수가 총 $O(N)$개이므로 전체 시간복잡도 $O(N^2)$에 문제를 해결할 수 있다.