11570: 환상의 듀엣

sianke1991

0. 이 문제는 1차원 배열로 풀 수 없다.
곡의 흐름이 1, 3, 8, 12, 13이고, 해당 음을 a가 불렀을 때 및 해당 음을 b가 불렀을 때로 나누어서 푼다고 생각해 보자.

	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
해당 음을 a가 불렀을 때 힘든 정도의 최소치	0 {1, null}	0 {3, 1}	5 {8, 1}	9 {12, 1}	10 {13, 1}
해당 음을 b가 불렀을 때 힘든 정도의 최소치	0 {null, 1}	0 {1, 3}	5 {1, 8}	9 {1, 12}	10 {1, 13}

* 각 칸에 대해서 검은색 값은 힘든 정도의 최소치, 녹색 값은 {최근에 a가 부른 음의 높이, 최근에 b가 부른 음의 높이}를 의미한다.
이 풀이는 잘못된 답을 내놓는데, 인덱스 2부터 답이 잘못되고 있다. (인덱스 2에서 힘든 정도의 최소치는 2가 되어야 한다.) 그리고 그 이유는 인덱스 1 시점에서 저장된 값의 한계 때문이다. 인덱스 1 시점에서 최적은 1과 3을 따로 부르는 것(그래서 힘든 정도는 0)이다. 하지만 인덱스 2의 곡 높이가 8이므로 (비록 국소적인 관점에서는 최적이 아니지만) 인덱스 1 시점에서는 1과 3을 한 사람이 불러야 한다. 각 시점에 대해 맨 마지막 음부터 이어 나가는 경우부터, 맨 처음 음부터 한 사람이 계속 부르는 경우까지 그 길이에 비례하는 경우를 고려해야 한다. 이렇게 각 시점에 대해 [해당 시점의 길이에 비례하는 개수의 값]을 저장해야 하므로 1차원 배열로는 풀기 어렵고 2차원 배열을 사용해야 한다.

1. 두 개의 2차원 배열

a가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {1, null}	0 {3, 1}	2 {8, 3}	7 {12, 8}	11 {13, 12}
[1]		2 {3, null}	5 {8, 1}	6 {12, 3}	8 {13, 8}
[2]			7 {8, null}	9 {12, 1}	7 {13, 3}
[3]				11 {12, null}	10 {13, 1}
[4]					12 {13, null}

b가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {null, 1}	0 {1, 3}	2 {3, 8}	7 {8, 12}	11 {12, 13}
[1]		2 {null, 3}	5 {1, 8}	6 {3, 12}	8 {8, 13}
[2]			7 {null, 8}	9 {1, 12}	7 {3, 13}
[3]				11 {null, 12}	10 {1, 13}
[4]					12 {null, 13}

두 개의 2차원 배열을 사용하여 문제를 해결하는 방법을 살펴본다. 2차원 배열에 대해 첫 번째 인덱스(이하 i 인덱스)는 곡의 흐름을, 두 번째 인덱스(이하 j 인덱스)는 a(또는 b)가 마지막에 연속으로 부른 곡의 길이를 의미한다. 예를 들어 dpArray[3][1] 은 인덱스 0, 1, 2, 3의 (길이 4 짜리) 곡을 부르는데, 마지막 두 부분(인덱스 2, 3) 부분을 a가 불렀을 때 힘든 정도의 최소치이다. (힘든 정도의 최소치 외에 최근에 a가 부른 음의 높이, 최근에 b가 부른 음의 높이를 저장해 둔다.) 이 표를 채울 때에는 j=0 인 경우와 j>0 인 경우에 그 방식이 서로 다르다.

a가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {1, null}	0 {3, 1}	2 {8, 3}	7 {12, 8}	11 {13, 12}
[1]		2 {3, null}	5 {8, 1}	6 {12, 3}	8 {13, 8}
[2]			7 {8, null}	9 {12, 1}	7 {13, 3}
[3]				11 {12, null}	10 {13, 1}
[4]					12 {13, null}

b가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {null, 1}	0 {1, 3}	2 {3, 8}	7 {8, 12}	11 {12, 13}
[1]		2 {null, 3}	5 {1, 8}	6 {3, 12}	8 {8, 13}
[2]			7 {null, 8}	9 {1, 12}	7 {3, 13}
[3]				11 {null, 12}	10 {1, 13}
[4]					12 {null, 13}

j = 0인 경우에는 마지막으로 연속으로 부른 음이 i 번째 음 바로 하나 이므로 바로 전 음(i-1 번째 음)은 상대가 부른 것이 된다. 따라서 dpArray[3][0]을 채우기 위해서는 상대방의 dpArray[2][j]를 참조해야 한다.
붉은 색의 경우 최근에 a가 부른 음이 3이므로 12의 음을 부르기 위해서는 힘든 정도가 9 만큼 증가해야 한다.
(최종 힘든 정도는 11)
푸른 색의 경우 최근에 a가 부른 음이 1이므로 12의 음을 부르기 위해서는 힘든 정도가 11 만큼 증가해야 한다.
(최종 힘든 정도는 17)
보라 색의 경우 최근에 a가 부른 음이 없으므로 12의 음을 부르는데 힘든 정도는 증가하지 않는다.
(최종 힘든 정도는 7)
이 세 경우 중 가장 낮은 값은 7이므로 7을 dpArray[3][0]에 기록한다.

a가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {1, null}	0 {3, 1}	2 {8, 3}	7 {12, 8}	11 {13, 12}
[1]		2 {3, null}	5 {8, 1}	6 {12, 3}	8 {13, 8}
[2]			7 {8, null}	9 {12, 1}	7 {13, 3}
[3]				11 {12, null}	10 {13, 1}
[4]					12 {13, null}

b가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {null, 1}	0 {1, 3}	2 {3, 8}	7 {8, 12}	11 {12, 13}
[1]		2 {null, 3}	5 {1, 8}	6 {3, 12}	8 {8, 13}
[2]			7 {null, 8}	9 {1, 12}	7 {3, 13}
[3]				11 {null, 12}	10 {1, 13}
[4]					12 {null, 13}

j > 0 인 경우에는 바로 전 음 역시 본인이 부른 것이다. 또한 표의 정의 상 dpArray[i][j]는 dpArray[i-1][j-1]을 참조해야 한다. (인덱스 3인 시점에서 마지막으로 길이 2만큼 연속으로 불렀다면 인덱스 2인 시점에는 마지막으로 길이 1만큼 연속으로 불렀을 것이다.)
붉은 색의 경우 최근에 a가 부른 음이 8이므로 12의 음을 부르기 위해서는 힘든 정도가 4 만큼 증가해야 한다.
(최종 힘든 정도는 6)
푸른 색의 경우 최근에 a가 부른 음이 8이므로 12의 음을 부르기 위해서는 힘든 정도가 4 만큼 증가해야 한다.
(최종 힘든 정도는 9)
보라 색의 경우 최근에 a가 부른 음이 8이므로 12의 음을 부르기 위해서는 힘든 정도가 4 만큼 증가해야 한다.
(최종 힘든 정도는 11)
이러한 방식으로 두 표를 채워 나갈 수 있다. 그러나...

a가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {1, null}	0 {3, 1}	2 {8, 3}	7 {12, 8}	11 {13, 12}
[1]		2 {3, null}	5 {8, 1}	6 {12, 3}	8 {13, 8}
[2]			7 {8, null}	9 {12, 1}	7 {13, 3}
[3]				11 {12, null}	10 {13, 1}
[4]					12 {13, null}

b가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {null, 1}	0 {1, 3}	2 {3, 8}	7 {8, 12}	11 {12, 13}
[1]		2 {null, 3}	5 {1, 8}	6 {3, 12}	8 {8, 13}
[2]			7 {null, 8}	9 {1, 12}	7 {3, 13}
[3]				11 {null, 12}	10 {1, 13}
[4]					12 {null, 13}

이 두 표는 서로 대칭이므로 두 표를 전부 채우는 것은 같은 일을 두 번 하는 것이다. (시간도 두 배, 공간도 두 배 차지하게 된다.) 이 두 표가 대칭이라는 사실에 집중하면 표 하나로 이 문제를 풀 수 있다. 또한...

a가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {1, null}	0 {3, 1}	2 {8, 3}	7 {12, 8}	11 {13, 12}
[1]		2 {3, null}	5 {8, 1}	6 {12, 3}	8 {13, 8}
[2]			7 {8, null}	9 {12, 1}	7 {13, 3}
[3]				11 {12, null}	10 {13, 1}
[4]					12 {13, null}

b가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {null, 1}	0 {1, 3}	2 {3, 8}	7 {8, 12}	11 {12, 13}
[1]		2 {null, 3}	5 {1, 8}	6 {3, 12}	8 {8, 13}
[2]			7 {null, 8}	9 {1, 12}	7 {3, 13}
[3]				11 {null, 12}	10 {1, 13}
[4]					12 {null, 13}

표의 정의 상 본인이 가장 마지막에 부른 음의 높이는 해당 i 인덱스의 곡의 높이와 같게 된다. 따라서 본인이 가장 마지막에 부른 음의 높이는 굳이 dpArray에 적을 필요는 없다. 이 사실을 이용하면 dpArray의 각 칸에 값 3 개 대신 2 개를 적음으로써 공간 사용을 더 줄일 수 있다.

2. 한 개의 2차원 배열

a가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {null}	0 {1}	2 {3}	7 {8}	11 {12}
[1]		2 {null}	5 {1}	6 {3}	8 {8}
[2]			7 {null}	9 {1}	7 {3}
[3]				11 {null}	10 {1}
[4]					12 {null}

* 각 칸에 대해서 중괄호 내의 값은 상대방(b)가 마지막으로 부른 음의 높이를 나타낸다.
두 개의 2차원 배열을 하나로 줄이면 위와 같은 모습이 된다. 가장 최근에 a가 부른 음의 높이는 해당 시점의 음악의 음의 높이와 같다. (예를 들어 dpArray[3][2]에서 가장 최근에 a가 부른 음의 높이는 12 이다.) 여기에서도 j=0일 때와 j>0일 때 표를 채워나가는 방식이 서로 다르다.

a가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {null}	0 {1}	2 {3}	7 {8}	11 {12}
[1]		2 {null}	5 {1}	6 {3}	8 {8}
[2]			7 {null}	9 {1}	7 {3}
[3]				11 {null}	10 {1}
[4]					12 {null}

j=0일 때에는 상대방 표를 참조해야 하지만, 여기서는 표를 하나로 줄였으므로 본인의 표를 참조해야 한다. 따라서 진한 녹색 사각형에서 연한 녹색 사각형으로 가는 흐름을 볼 때 진한 녹색 사각형은 [b가 해당 음을 부르는 경우, {a가 마지막으로 부른 음의 높이}]를 의미하는 것으로 간주해야 한다.
붉은 색의 경우 최근에 a가 부른 음이 3이므로 12의 음을 부르기 위해서는 힘든 정도가 9 만큼 증가해야 한다.
(최종 힘든 정도는 11)
푸른 색의 경우 최근에 a가 부른 음이 1이므로 12의 음을 부르기 위해서는 힘든 정도가 11 만큼 증가해야 한다.
(최종 힘든 정도는 17)
보라 색의 경우 최근에 a가 부른 음이 없으므로 12의 음을 부르는데 힘든 정도는 증가하지 않는다.
(최종 힘든 정도는 7)
이 세 경우 중 가장 낮은 값은 7이므로 7을 dpArray[3][0]에 기록한다.

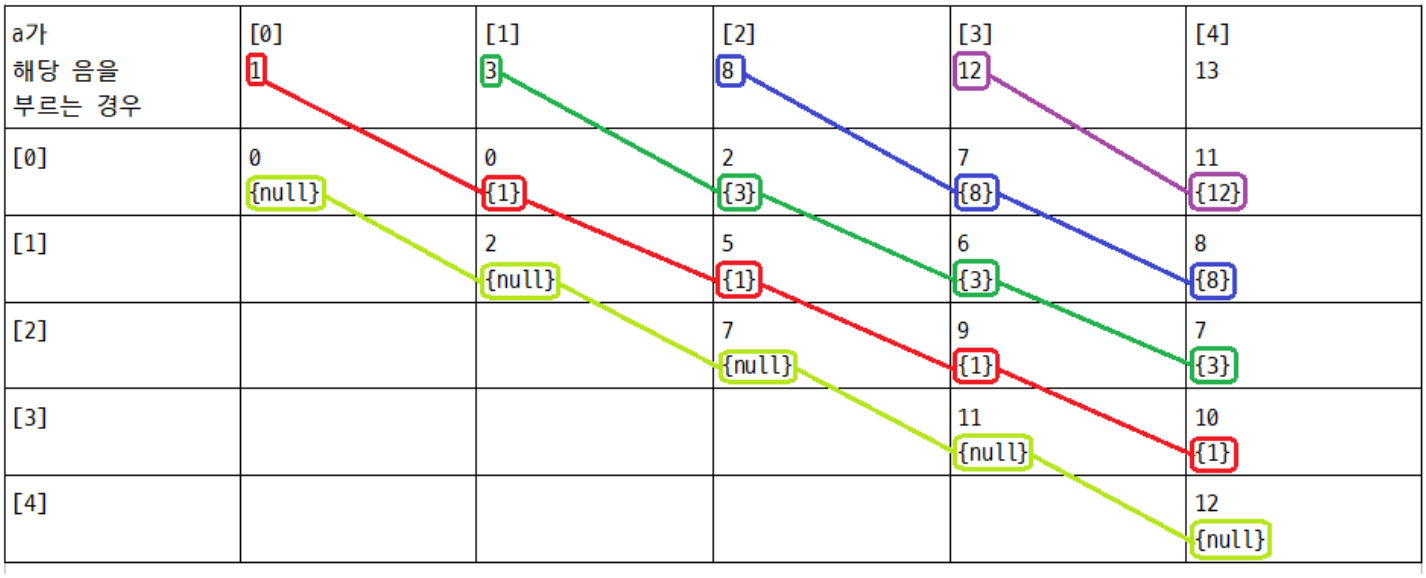
a가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {null}	0 {1}	2 {3}	7 {8}	11 {12}
[1]		2 {null}	5 {1}	6 {3}	8 {8}
[2]			7 {null}	9 {1}	7 {3}
[3]				11 {null}	10 {1}
[4]					12 {null}

j>0일 때에는 표를 채워나가는 방식이 (표 2개를 사용할 때와) 비슷하다.
붉은 색의 경우 최근에 a가 부른 음이 8이므로 12의 음을 부르기 위해서는 힘든 정도가 4 만큼 증가해야 한다.
(최종 힘든 정도는 6)
푸른 색의 경우 최근에 a가 부른 음이 8이므로 12의 음을 부르기 위해서는 힘든 정도가 4 만큼 증가해야 한다.
(최종 힘든 정도는 9)
보라 색의 경우 최근에 a가 부른 음이 8이므로 12의 음을 부르기 위해서는 힘든 정도가 4 만큼 증가해야 한다.
(최종 힘든 정도는 11)

a가 해당 음을 부르는 경우	[0] 1	[1] 3	[2] 8	[3] 12	[4] 13
[0]	0 {null}	0 {1}	2 {3}	7 {8}	11 {12}
[1]		2 {null}	5 {1}	6 {3}	8 {8}
[2]			7 {null}	9 {1}	7 {3}
[3]				11 {null}	10 {1}
[4]					12 {null}

표를 전부 채워 나갔다면 dpArray[곡의 길이-1]에서 최소치를 찾는다. (길이가 5인 곡의 경우 dpArray[4]에서 최소치를 찾는다. 위의 예시의 경우 7이 그 정답이 된다.)

3. 공간을 더 줄이고 싶다면...
위에서 제시한 방법은 표의 각 칸에 값을 두 개씩 채웠다. 하지만 표의 정의, 특징을 잘 살펴보면 표의 각 칸에 {상대방이 마지막으로 부른 음의 높이}를 기록할 필요가 없음을 알 수 있다.



{상대방이 마지막으로 부른 음의 높이}에 규칙성이 존재한다.

dpArray[i][j]에 대해 a는 마지막으로 j-i 부터 j 까지 연속으로 노래를 불렀다. 이는 인덱스 j-i-1은 상대방(b)가 불렀음을 의미하고, 또한 이 위치는 b가 마지막으로 노래를 부른 위치이다. 두 인덱스 i와 j를 알면 상대방이 마지막으로 부른 음의 높이를 알 수 있으므로 이를 굳이 dpArray에 적을 필요가 없음을 알 수 있다.

[END OF DOCUMENT]