

JavaScript

다이나믹 프로그래밍

다이나믹 프로그래밍 문제 풀이

다이나믹 프로그래밍 이해하기 | 코딩 테스트에서 자주 등장하는 다이나믹 프로그래밍 이해하기

강사 나동빈

JavaScript

다이나믹 프로그래밍

다이나믹 프로그래밍 문제 풀이

JavaScript 동적 계획법 **혼자 힘으로 풀어보기** 다이나믹 프로그래밍

JavaScript
동적 계획법
다이나믹
프로그래밍

문제 제목: 연속부분최대곱

문제 난이도: ★★☆☆☆

문제 유형: 다이나믹 프로그래밍

추천 풀이 시간: 40분

JavaScript 동적 계획법 문제 해결 아이디어

다이나믹 프로그래밍

JavaScript
동적 계획법
다이나믹
프로그래밍

- $dp[i]$ 를 i 를 마지막 인덱스로하는 연속곱의 최대값으로 정의한다.
- 각 원소의 위치에서 이전 원소와 곱셈을 수행할 지 결정할 수 있다.
 - 점화식: $dp[i] = \max(dp[i], dp[i] * dp[i - 1])$
- 편의상 하나의 배열만 사용하여 DP 테이블에 초기 데이터를 담을 수 있다.
- 이후에 해당 DP 테이블의 값을 점화식에 따라서 갱신할 수 있다.

JavaScript 동적 계획법

정답 코드 예시

다이나믹 프로그래밍

JavaScript
동적 계획법
다이나믹
프로그래밍

```
// readline 모듈보다는 fs를 이용해 파일 전체를 읽기
let file = require('fs').readFileSync('/dev/stdin');
let input = file.toString().split('\n');

let n = Number(input[0]);

dp = [] // 다이나믹 프로그래밍을 위한 DP 테이블 초기화
for (let i = 1; i <= n; i++) {
  dp.push(Number(input[i]));
}

// 다이나믹 프로그래밍 수행
for (let i = 1; i < n; i++) {
  dp[i] = Math.max(dp[i], dp[i] * dp[i - 1]);
}

console.log(Math.max(... dp).toFixed(3));
```

JavaScript 동적 계획법 다이나믹 프로그래밍

혼자 힘으로 풀어보기

JavaScript
동적 계획법
다이나믹
프로그래밍

문제 제목: 극장 좌석

문제 난이도: ★★☆☆☆

문제 유형: 다이나믹 프로그래밍

추천 풀이 시간: 50분

JavaScript 동적 계획법 문제 해결 아이디어

다이나믹 프로그래밍

JavaScript
동적 계획법
다이나믹
프로그래밍

- VIP 좌석으로 구분된 각 좌석 묶음마다 경우의 수를 계산할 수 있다.



각 테이블의 값은 피보나치 수열과 동일

좌석 묶음 개수: [3, 2, 2]
 $dp[3] \times dp[2] \times dp[2] = 12$

JavaScript 동적 계획법 정답 코드 예시

다이나믹 프로그래밍

JavaScript
동적 계획법
다이나믹
프로그래밍

```
// readline 모듈보다는 fs를 이용해 파일 전체를 읽기
let file = require('fs').readFileSync('/dev/stdin');
let input = file.toString().split('\n');

let n = Number(input[0]);
let m = Number(input[1]);

let d = new Array(50).fill(0)
d[0] = 1;
d[1] = 1;
d[2] = 2;

// 다이나믹 프로그래밍 (피보나치 수열)
function dp(x) {
  if (d[x] !== 0) return d[x];
  d[x] = dp(x - 1) + dp(x - 2);
  return d[x];
}
```


JavaScript 동적 계획법 다이나믹 프로그래밍

정답 코드 예시

JavaScript
동적 계획법
다이나믹
프로그래밍

```
// VIP 좌석을 기준으로 몇 개씩 묶이는지 확인
let arr = [];
let start = 0;
for (let i = 2; i < m + 2; i++) {
    end = Number(input[i]);
    arr.push(end - 1 - start);
    start = end;
}
arr.push(n - start);

// 각 묶음의 개수에 대하여 DP 테이블의 값 가져오기
let res = 1;
for (let x of arr) res *= dp(x);
console.log(res);
```

JavaScript 동적 계획법
다이나믹 프로그래밍

혼자 힘으로 풀어보기

JavaScript
동적 계획법
다이나믹
프로그래밍

문제 제목: 함께 블록 쌓기

문제 난이도: ★★☆☆☆

문제 유형: 다이나믹 프로그래밍

추천 풀이 시간: 50분

JavaScript 동적 계획법 문제 해결 아이디어

다이나믹 프로그래밍

JavaScript
동적 계획법
다이나믹
프로그래밍

- 1번부터 N번까지의 학생들은 각각 블록들을 가지고 있다.
- 학생마다 최대 M개의 블록을 가지고 있을 수 있으며, 한 명의 학생이 가지고 있는 모든 블록들의 높이는 서로 다르다.
- 이때 1번부터 N번까지의 학생들이 가진 블록을 차례대로 사용하여 바닥에서부터 쌓아 올려 하나의 탑을 만들고자 한다.
- 단, 어떤 학생의 블록은 사용하지 않아도 되며 한 학생당 최대 1개의 블록만을 사용할 수 있다.
- 1번부터 N번까지의 학생들이 가지고 있는 블록들에 대한 정보가 주어졌을 때, 높이가 정확히 H인 탑을 만들 수 있는 경우의 수를 계산하는 프로그램을 작성하여라.

JavaScript 동적 계획법 문제 해결 아이디어

다이나믹 프로그래밍

JavaScript
동적 계획법
다이나믹
프로그래밍

- 예를 들어 $N = 3, M = 3, H = 5$ 인 상황을 가정하자.
- 또한 각 학생마다 가지고 있는 블록들의 높이가 다음과 같다고 가정하자.
 - 1번 학생: 2, 3, 5
 - 2번 학생: 3, 5
 - 3번 학생: 1, 2, 3
- 이때는 다음 그림과 같이 6가지가 존재한다.
 - (블록을 사용하지 않는 경우는 X로 표시하였다.)

1번 학생	2번 학생	3번 학생
X	3	2
X	5	X
2	X	3
2	3	X
3	X	2
5	X	X

JavaScript 동적 계획법 문제 해결 아이디어

다이나믹 프로그래밍

JavaScript
동적 계획법
다이나믹
프로그래밍

- i 는 각 학생 인덱스, j 는 각 높이, k 는 해당 학생이 가지고 있는 각 블록의 인덱스이다.
- 점화식: $D[i][j + A[i][k]] += D[i-1][j]$ if $D[i-1][j]$ 가 0이 아닌 경우

- 1번 학생: 2, 3, 5
- 2번 학생: 3, 5
- 3번 학생: 1, 2, 3

높이	0	1	2	3	4	5
초기 상태	1	0	0	0	0	0
1번 학생	1	0	1	1	0	1
2번 학생	1	0	1	2	0	3
3번 학생	1	1	2	4	3	6

JavaScript 동적 계획법 다이나믹 프로그래밍

정답 코드 예시

JavaScript
동적 계획법
다이나믹
프로그래밍

```
// readline 모듈보다는 fs를 이용해 파일 전체를 읽기
let file = require('fs').readFileSync('/dev/stdin');
let input = file.toString().split('\n');

let [n, m, h] = input[0].split(' ').map(Number);
let a = [];
let d = new Array(h + 1).fill(0);
for (let i = 1; i <= n; i++) {
  a.push(input[i].split(' ').map(Number));
}
```

JavaScript 동적 계획법 다이나믹 프로그래밍

정답 코드 예시

JavaScript
동적 계획법
다이나믹
프로그래밍

```
d[0] = 1;
// 모든 학생에 대하여 처리
for (let i = 0; i < n; i++) {
  let data = [];
  // 0부터 h까지의 모든 높이에 대하여 처리
  for (j = 0; j <= h; j++) {
    for (let k = 0; k < a.length; k++) { // 해당 학생의 모든 블록을 확인하며
      // 현재 학생의 블록으로 특정 높이의 탑을 쌓을 수 있는 경우
      if (d[j] != 0 && j + a[i][k] <= h) {
        data.push([j + a[i][k], d[j]]);
      }
    }
  }
  // 쌓을 수 있는 높이에 대하여, 경우의 수 증가
  for ([height, value] of data) {
    d[height] = (d[height] + value) % 10007;
  }
}
console.log(d[h]);
```