JavaScript 누적합

누적합 알고리즘 문제 풀이

누적합 알고리즘 문제 풀이 | 코딩 테스트에서 자주 등장하는 누적합 이해하기

강사 나동빈

JavaScript
누적합

누적합 알고리즘 문제 풀이

2차원 배열의 합: https://www.acmicpc.net/problem/2167

혼자 힘으로 풀어보기

문제 제목: 2차원 배열의 합

문제 난이도: ★★☆☆☆

문제 유형: 누적합

추천 풀이 시간: 50분

2차원 배열의 합: https://www.acmicpc.net/problem/2167

문제 풀이 핵심 아이디어

- 본 문제의 **요구 사항**은 다음과 같다.

→ 2차원 배열에서 (i, j)부터 (x, y) 위치까지의 **직사각형**에 저장된 수들의 합을 계산한다.

- 예를 들어 (2, 2)부터 (3, 3)까지의 합은 11이다.

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

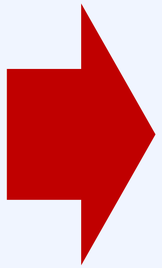
2차원 배열의 합: https://www.acmicpc.net/problem/2167

문제 풀이 핵심 아이디어

[Step 1] 기본 누적 합 계산하기

- (1, 1)부터 (x, y) 위치까지의 **직사각형**에 저장된 수들의 **누적 합**을 계산한다.
- $sum[x][y] = arr[x][y] + sum[x-1][y] + sum[x][y-1] - sum[x-1][y-1]$

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



누적 합

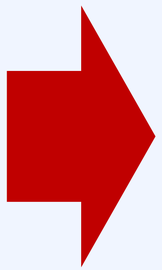
3	8	15	18
7	14	27	34
16	24	39	51
17	25	44	59

문제 풀이 핵심 아이디어

[Step 2] 쿼리 처리하기

- (i, j)부터 (x, y) 위치까지의 **직사각형**에 저장된 수들의 합을 계산할 수 있다.
- 구간 합: $sum[x][y] - sum[i-1][y] - sum[x][j-1] + sum[i-1][j-1]$

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



누적 합

3	8	15	18
7	14	27	34
16	24	39	51
17	25	44	59

예시: 11
(39 – 15 – 16 + 3)

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

정답 코드 예시


```
let file = require('fs').readFileSync('/dev/stdin');
let input = file.toString().split('\n');

let [n, m] = input[0].split(" ").map(Number); // 배열 크기(N X M)
let arr = [new Array(m + 1).fill(0)]; // 초기 배열
for (let i = 1; i <= n; i++) {
  arr.push([0, ...input[i].split(" ").map(Number)]);
}
let k = Number(input[n + 1]); // 쿼리의 수(K)
let queries = []; // 각 쿼리 정보 배열
for (let line = n + 2; line <= n + 1 + k; line++) {
  let [i, j, x, y] = input[line].split(" ").map(Number);
  queries.push([i, j, x, y]);
}
```

정답 코드 예시


```
// (1, 1)부터의 누적 합(sum) 계산
let sum = [];
for (let i = 0; i <= n; i++) sum.push(new Array(m + 1).fill(0));
for (let i = 1; i <= n; i++) {
  for (let j = 1; j <= m; j++) {
    sum[i][j] = arr[i][j] + sum[i - 1][j] + sum[i][j - 1] - sum[i - 1][j - 1];
  }
}

// (i, j)부터 (n, m)까지의 구간 합 계산
for (let index = 0; index < k; index++) {
  let [i, j, x, y] = queries[index];
  let current = sum[x][y] - sum[i - 1][y] - sum[x][j - 1] + sum[i - 1][j - 1];
  console.log(current);
}
```

부분합: https://www.acmicpc.net/problem/1806

혼자 힘으로 풀어보기

문제 제목: 부분합

문제 난이도: ★★★☆☆

문제 유형: 누적합

추천 풀이 시간: 50분

부분합: https://www.acmicpc.net/problem/1806

문제 풀이 핵심 아이디어

- 길이 N짜리 수열이 주어진다.
- 이 수열에서 구간 합 중에 그 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 계산한다.

[예시]

- $N = 10, S = 15, arr = [5, 1, 3, 5, 10, 7, 4, 9, 2, 8]$인 경우를 가정하자.

→ 이 경우 $arr[3] + arr[4] \geq S = 15$이며, 길이는 2가 된다.

부분합: https://www.acmicpc.net/problem/1806

문제 풀이 핵심 아이디어

- **투 포인터**를 사용하여 **구간 합**을 체크하여 문제를 해결할 수 있다.
- 다음과 같이 초기화를 진행한다.

1. 시작점$(start)$ = 0
2. 끝점(end) = 0

- 시작점$(start)$이 증가하면 구간 합이 증가한다.
- 끝점(end)이 증가하면 구간 합이 감소한다.

부분합: https://www.acmicpc.net/problem/1806

정답 코드 예시


```
let file = require('fs').readFileSync('/dev/stdin');
let input = file.toString().split('\n');

let [n, s] = input[0].split(' ').map(Number);
let arr = input[1].split(' ').map(Number);

let result = 1e9;
let start = 0; // 시작점(start)
let end = 0; // 끝점(end)
let summary = arr[0]; // 구간 합
```

부분합: https://www.acmicpc.net/problem/1806

정답 코드 예시


```
// i가 고정된 상태에서 j를 최대한 오른쪽으로 이동시키는 구현 방식
while (true) {
  // 현재 합이 s보다 작다면, 합을 키우기 위해 end를 증가
  while (end < n - 1 && summary < s) {
    end += 1;
    summary += arr[end];
  }
  if (summary >= s) { // 현재 합이 s 이상인 경우
    result = Math.min(result, end - start + 1); // 최소 길이 계산
  }
  summary -= arr[start];
  start += 1;
  // [유의] 탈출 조건에 유의
  if (start >= n) break;
}
if (result == 1e9) result = 0;
console.log(result);
```

컬러볼: https://www.acmicpc.net/problem/10800

혼자 힘으로 풀어보기

문제 제목: 컬러볼

문제 난이도: ★★★☆☆

문제 유형: 누적합, 정렬

추천 풀이 시간: 50분

컬러볼: https://www.acmicpc.net/problem/10800

문제 풀이 핵심 아이디어

- 문제에서 주어진 예시를 그림으로 그려보면 다음과 같다.

공 번호	색	크기
1	1	10
2	3	15
3	1	3
4	4	8

- 자기 공보다 **크기가 작고 색이 다른 공을 사로잡을** 수 있다.
- 예를 들어 1번 공은 4번 공만 잡을 수 있다.

컬러볼: https://www.acmicpc.net/problem/10800

문제 풀이 핵심 아이디어

- 공의 개수 N이 **최대 20만**까지 들어올 수 있다.
- 시간 복잡도 $O(NlogN)$의 알고리즘을 작성해야 한다.
  - $O(N^2)$으로 푸는 경우: 20만 X 20만 = 400억
  - $O(N)$으로 푸는 경우: 20만
  - $O(NlogN)$으로 푸는 경우: 20만 X 18 = 360만
- 따라서 $O(NlogN)$의 시간 복잡도를 가지는 $sort()$를 사용할 수 있다.

컬러볼: https://www.acmicpc.net/problem/10800

문제 풀이 핵심 아이디어

- **색상과 상관 없이** 항상 자기보다 작은 공들을 잡을 수 있다고 하면?

1. 단순히 크기만 고려해서 정렬한다.
   - 입력이 [10, 15, 3, 8]이라면, 정렬해서 [3, 8, 10, 15]를 만들자.
2. 이후에 **누적합**을 구해준다.
   - 누적합이란 앞에서부터 계속 더해준 값을 모아 놓은 것을 말한다.
   - 누적합: [3, 11, 21, 36]
   - 누적합을 한 칸씩 오른쪽으로 이동(shift)해주면 다음과 같다.
     - [0, 3, 11, 21]
     - 이것은 각 공에 대하여 "**자기보다 작은 공들의 크기 합**"과 같다.

컬러볼: https://www.acmicpc.net/problem/10800

문제 풀이 핵심 아이디어

- 이제 “자기와 같은 색상은 잡을 수 없다”는 조건도 함께 고려한다면?
- **[핵심]** 똑같은 방식으로 해결하면 되는데, **색상마다 누적합을 계산**해주면 된다.

컬러볼: https://www.acmicpc.net/problem/10800

문제 풀이 핵심 아이디어

- 자기 공보다 **크기가 작고 색이 다른 공을 사로잡을** 수 있다.

- 모든 공을 **크기를 기준으로 오름차순 정렬**한 뒤에 다음 공식을 계산한다.
  - 전체 공들의 크기 누적 합 – 같은 색상인 공들의 크기 누적 합

컬러볼: https://www.acmicpc.net/problem/10800

정답 코드 예시


```
let input = require('fs').readFileSync('/dev/stdin');
input = input.toString().split('\n');

let n = Number(input[0]);
let arr = [];
for (let i = 0; i < n; i++) {
  // 색상(c)와 크기(s)를 입력받기
  let c = Number(input[i + 1].split(' ')[0]);
  let s = Number(input[i + 1].split(' ')[1]);
  arr.push([c, s, i]);
}
// 크기를 기준으로 오름차순 정렬
arr.sort((a, b) => a[1] - b[1]);

let summary = 0; // 전체 누적 합
let colorSummary = Array(200001).fill(0); // 색상별 누적 합
let result = Array(n).fill(0) // 공의 등장 순서(i)별 최종 결과
```

컬러볼: https://www.acmicpc.net/problem/10800

정답 코드 예시


```
let start = 0;
while (start < n) {
  // 크기가 같은 공의 마지막 인덱스 찾기(start는 시작 인덱스 end는 끝 인덱스)
  let end = start;
  while (end < n && arr[start][1] == arr[end][1]) end += 1;
  // 자기보다 작은 공들의 크기 합 - 같은 색상인 공들의 크기 합
  for (let i = start; i < end; i++) {
    result[arr[i][2]] = summary - colorSummary[arr[i][0]];
  }
  // 합계 값(누적 합) 갱신
  for (let i = start; i < end; i++) {
    colorSummary[arr[i][0]] += arr[i][1]; // 색상별 누적 합
    summary += arr[i][1]; // 전체 누적 합
  }
  start = end;
}
console.log(result.join('\n'));
```