JavaScript BFS 알고리즘
BFS 문제 풀이

BFS 문제 풀이 | 코딩 테스트에서 자주 등장하는 BFS 알고리즘 이해하기

강사 나동빈

JavaScript
BFS 알고리즘

BFS 문제 풀이

인구 이동: https://www.acmicpc.net/problem/16234

혼자 힘으로 풀어보기

문제 제목: 인구 이동

문제 난이도: ★★★☆☆

문제 유형: BFS

추천 풀이 시간: 60분

인구 이동: https://www.acmicpc.net/problem/16234

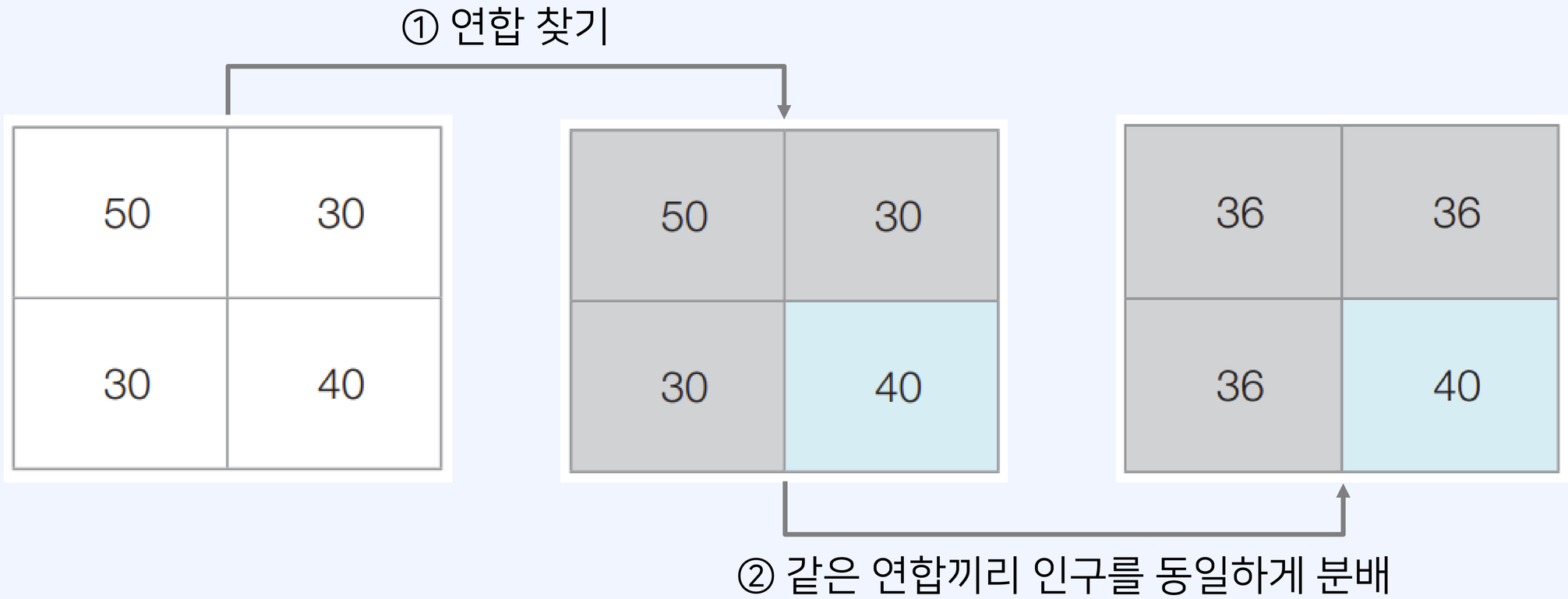
문제 풀이 핵심 아이디어

- 모든 나라의 위치에서 상, 하, 좌, 우로 국경선을 열 수 있는지 확인해야 합니다.
  - 따라서 모든 나라에서 DFS 혹은 BFS를 수행하여 인접한 나라의 인구수를 확인합니다.
  - 가능하다면 국경선을 열고 인구 이동 처리를 진행합니다.

인구 이동: https://www.acmicpc.net/problem/16234

문제 풀이 핵심 아이디어

- 문제에서 제시된 세 번째 예시를 확인해 봅시다.
  - $L = 20, R = 50$



인구 이동: https://www.acmicpc.net/problem/16234

정답 코드 예시


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

let [n, l, r] = input[0].split(' ').map(Number); // 땅의 크기(N), L, R 값을 입력받기
let graph = []; // 전체 나라의 정보(N x N)를 입력 받기
for (let i = 1; i <= n; i++) {
  let row = input[i].split(' ').map(Number);
  graph.push(row);
}
let dx = [-1, 0, 1, 0];
let dy = [0, -1, 0, 1];
let totalCount = 0;

while (true) { // 더 이상 인구 이동을 할 수 없을 때까지 반복
  let union = Array.from(Array(n), () => Array(n).fill(-1));
  let index = 0;
  for (let i = 0; i < n; i++) {
    for (let j = 0; j < n; j++) {
      if (union[i][j] == -1) { // 해당 나라가 아직 처리되지 않았다면
        bfs(i, j, index, union);
        index++;
      }
    }
  }
  if (index == n * n) break; // 모든 인구 이동이 끝난 경우
  totalCount += 1;
}
console.log(totalCount); // 인구 이동 횟수 출력
```

인구 이동: https://www.acmicpc.net/problem/16234

정답 코드 예시


```
function bfs(x, y, index, union) { // 특정 위치에서 출발하여 모든 연합을 체크한 뒤에 데이터 갱신
  let united = [[x, y]]; // (x, y)의 위치와 연결된 나라(연합) 정보를 담는 리스트
  let q = new Queue(); // 너비 우선 탐색(BFS)을 위한 큐 라이브러리 사용
  q.enqueue([x, y]);
  union[x][y] = index; // 현재 연합의 번호 할당
  let summary = graph[x][y]; // 현재 연합의 전체 인구 수
  let cnt = 1; // 현재 연합의 국가 수
  while (q.getLength() != 0) { // 큐가 빌 때까지 반복(BFS)
    let [x, y] = q.dequeue();
    for (let i = 0; i < 4; i++) { // 현재 위치에서 4가지 방향을 확인하며
      let nx = x + dx[i];
      let ny = y + dy[i];
      // 바로 옆에 있는 나라를 확인하여
      if (0 <= nx && nx < n && 0 <= ny && ny < n && union[nx][ny] == -1) {
        let dif = Math.abs(graph[nx][ny] - graph[x][y]); // 옆에 있는 나라와 인구 차이가 L명 이상, R명 이하라면
        if (l <= dif && dif <= r) {
          q.enqueue([nx, ny]);
          union[nx][ny] = index; // 연합에 추가하기
          summary += graph[nx][ny];
          cnt += 1;
          united.push([nx, ny]);
        }
      }
    }
  }
  for (let unit of united) { // 연합 국가끼리 인구를 분배
    let [i, j] = unit;
    graph[i][j] = parseInt(summary / cnt);
  }
}
```

뱀: https://www.acmicpc.net/problem/3190

혼자 힘으로 풀어보기

문제 제목: 뱀

문제 난이도: ★★★☆☆

문제 유형: 큐 자료구조, 시뮬레이션

추천 풀이 시간: 50분

뱀: https://www.acmicpc.net/problem/3190

문제 해결 아이디어

- 전형적인 시뮬레이션(simulation) 문제 유형이다.
  - 문제에서 요구하는 대로 실수 없이 구현할 수 있다면 정답 판정을 받을 수 있다.
- 2차원 배열상의 맵에서 뱀이 이동한다.
  - 따라서 동, 남, 서, 북의 위치로 이동하는 기능을 구현해야 한다.
  - 매 시점마다 뱀이 존재하는 위치를 항상 2차원 리스트에 기록해야 한다.
- 시뮬레이션 문제를 가장 쉽게 풀기 위해서는 그림으로 그려보는 것이 좋다.
  - 코딩 테스트에 참여할 때는 종이에 자신만의 그림으로 그려보도록 한다.

뱀: https://www.acmicpc.net/problem/3190

정답 코드 예시


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

let n = Number(input[0]); // 보드의 크기(N)
let k = Number(input[1]); // 사과의 개수(K)
let data = []; // [N + 1][N + 1] 크기의 맵 정보
for (let i = 0; i < n + 1; i++) {
  data.push(new Array(n + 1).fill(0));
}
for (let i = 2; i <= k + 1; i++) {
  let [a, b] = input[i].split(' ').map(Number);
  data[a][b] = 1; // 사과가 있는 곳은 1로 표시
}
let l = Number(input[k + 2]); // 뱀의 방향 변환 횟수
let info = [];
for (let i = k + 3; i < k + 3 + l; i++) {
  let [x, c] = input[i].split(' ');
  info.push([Number(x), c]);
}
```

정답 코드 예시

백: https://www.acmicpc.net/problem/3190


```
// 처음에는 오른쪽을 보고 있으므로 (동, 남, 서, 북)
let dx = [0, 1, 0, -1];
let dy = [1, 0, -1, 0];

function turn(direction, c) {
  if (c == 'L') {
    direction = direction - 1;
    if (direction == -1) direction = 3;
  }
  else direction = (direction + 1) % 4;
  return direction;
}
```

뱀: https://www.acmicpc.net/problem/3190

정답 코드 예시

JavaScript
BFS
BFS
문제 풀이

```
let [x, y] = [1, 1]; // 뱀의 머리 위치
data[x][y] = 2; // 뱀이 존재하는 위치는 2로 표시
let direction = 0; // 처음에는 동쪽을 보고 있음
let time = 0; // 시작한 뒤에 지난 '초' 시간
let index = 0; // 다음에 회전할 정보
let q = new Queue();
q.enqueue([x, y]); // 뱀이 차지하고 있는 위치 정보 (꼬리가 앞쪽)
while (true) {
  let nx = x + dx[direction];
  let ny = y + dy[direction];
  if (1 <= nx && nx <= n && 1 <= ny && ny <= n && data[nx][ny] != 2) { // 맵 범위 안에 있고, 뱀의 몸통이 없는 위치라면
    if (data[nx][ny] == 0) { // 사과가 없다면 이동 후에 꼬리 제거
      data[nx][ny] = 2;
      q.enqueue([nx, ny]);
      let [px, py] = q.dequeue();
      data[px][py] = 0;
    }
    if (data[nx][ny] == 1) { // 사과가 있다면 이동 후에 꼬리 그대로 두기
      data[nx][ny] = 2;
      q.enqueue([nx, ny]);
    }
  }
  else { // 벽이나 뱀의 몸통과 부딪혔다면
    time += 1;
    break;
  }
  [x, y] = [nx, ny]; // 다음 위치로 머리를 이동
  time += 1;
  if (index < l && time == info[index][0]) { // 회전할 시간인 경우 회전
    direction = turn(direction, info[index][1]);
    index += 1;
  }
}
console.log(time);
```