JavaScript BFS 알고리즘
BFS 문제 풀이

BFS 문제 풀이 | 코딩 테스트에서 자주 등장하는 BFS 알고리즘 이해하기

강사 나동빈

JavaScript
BFS 알고리즘

BFS 문제 풀이

경쟁적 전염: https://www.acmicpc.net/problem/18405

혼자 힘으로 풀어보기

문제 제목: 경쟁적 전염

문제 난이도: ★★★☆☆

문제 유형: 너비 우선 탐색, 그래프 순회, 최단 거리

추천 풀이 시간: 60분

경쟁적 전염: https://www.acmicpc.net/problem/18405

문제 해결 아이디어

- N × N 크기의 시험관이 있으며, 시험관은 1 × 1 크기의 칸으로 나누어진다.
- 특정한 위치에는 바이러스가 존재할 수 있다. 바이러스의 종류는 1 ~ K번까지 K가지가 있으며 모든 바이러스는 이 중 하나에 속한다.
- 시험관에 존재하는 모든 바이러스는 1초마다 [상, 하, 좌, 우]의 방향으로 증식하는데, 매초 **번호가 낮은 종류의 바이러스부터 먼저 증식**한다.
- 또한 증식 과정에서 특정한 칸에 이미 어떠한 바이러스가 있다면, 그곳에는 다른 바이러스가 들어갈 수 없다.
- 시험관의 크기와 바이러스의 위치 정보가 주어졌을 때, S초가 지난 후에 (X, Y)에 존재하는 바이러스의 종류를 출력하는 프로그램을 작성해야 한다.

경쟁적 전염: https://www.acmicpc.net/problem/18405

문제 해결 아이디어

초기 상태

1		2
3		

1초 후

1	1	2
1		2
3	3	

2초 후

1	1	2
1	1	2
3	3	2

경쟁적 전염: https://www.acmicpc.net/problem/18405

문제 해결 아이디어

- 모든 간선의 비용은 1이다.
- 따라서 너비 우선 탐색(BFS)을 이용하여 최단 거리를 계산할 수 있다.
  - 다만 바이러스가 낮은 번호부터 증식한다는 점이 특징이다.
  - 따라서 초기 **큐(Queue)에 원소를 삽입**할 때 낮은 바이러스의 번호부터 삽입한다.

경쟁적 전염: https://www.acmicpc.net/problem/18405

정답 코드 예시

JavaScript
BFS
BFS
문제 풀이

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

let [n, k] = input[0].split(' ').map(Number);
let graph = []; // 전체 보드 정보를 담는 리스트
let data = []; // 바이러스에 대한 정보를 담는 리스트

for (let i = 0; i < n; i++) {
  // 보드 정보를 한 줄 단위로 입력
  graph.push(input[i + 1].split(' ').map(Number));
  for (let j = 0; j < n; j++) {
    // 해당 위치에 바이러스가 존재하는 경우
    if (graph[i][j] != 0) {
      // (바이러스의 종류, 시간, 위치 X, 위치 Y) 삽입
      data.push([graph[i][j], 0, i, j]);
    }
  }
}

// 정렬 이후에 큐로 옮기기(낮은 번호의 바이러스가 먼저 증식)
data.sort((a, b) => a[0] - b[0]);
queue = new Queue();
for (let x of data) {
  queue.enqueue(x);
}
```

경쟁적 전염: https://www.acmicpc.net/problem/18405

정답 코드 예시


```
let [targetS, targetX, targetY] = input[n + 1].split(' ').map(Number);

// 바이러스가 퍼져나갈 수 있는 4가지의 위치
let dx = [-1, 0, 1, 0];
let dy = [0, 1, 0, -1];

// 너비 우선 탐색(BFS) 진행
while (queue.getLength() != 0) {
  let [virus, s, x, y] = queue.dequeue();
  // 정확히 s초가 지나거나, 큐가 빌 때까지 반복
  if (s == targetS) break;
  // 현재 노드에서 주변 4가지 위치를 각각 확인
  for (let i = 0; i < 4; i++) {
    let nx = x + dx[i];
    let ny = y + dy[i];
    // 해당 위치로 이동할 수 있는 경우
    if (0 <= nx && nx < n && 0 <= ny && ny < n) {
      // 아직 방문하지 않은 위치라면, 그 위치에 바이러스 넣기
      if (graph[nx][ny] == 0) {
        graph[nx][ny] = virus;
        queue.enqueue([virus, s + 1, nx, ny]);
      }
    }
  }
}
console.log(graph[targetX - 1][targetY - 1]);
```

혼자 힘으로 풀어보기

문제 제목: 특정 거리의 도시 찾기

문제 난이도: ★★☆☆☆

문제 유형: 너비 우선 탐색, 그래프 순회, 최단 거리

추천 풀이 시간: 40분

문제 해결 아이디어

- 모든 간선의 비용은 1이다.
- 따라서 너비 우선 탐색(BFS)을 이용하여 최단 거리를 계산할 수 있다.
- 먼저 특정한 도시 X를 시작점으로 **BFS를 수행하여 모든 도시까지의 최단 거리를 계산**한다.
- 이후에 각 노드로의 최단 거리를 하나씩 확인하며 값이 K인 경우 해당 도시의 번호를 출력한다.

정답 코드 예시


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

// 도시의 개수(N), 도로의 개수(M), 거리(K), 출발 도시(X)
let [n, m, k, x] = input[0].split(' ').map(Number);
let graph = []; // 그래프 정보
for (let i = 1; i <= n; i++) {
  graph[i] = [];
}
for (let i = 1; i <= m; i++) {
  let [x, y] = input[i].split(' ').map(Number);
  graph[x].push(y);
}
// 모든 도시에 대한 최단 거리 초기화
let distance = new Array(n + 1).fill(-1);
distance[x] = 0; // 출발 도시까지의 거리는 0으로 설정
```

정답 코드 예시


```
// 너비 우선 탐색(BFS) 수행
let queue = new Queue();
queue.enqueue(x);
while (queue.getLength() != 0) { // 큐가 빌 때까지 반복하기
  let now = queue.dequeue();
  // 현재 도시에서 이동할 수 있는 모든 도시를 확인
  for (let nextNode of graph[now]) {
    if (distance[nextNode] == -1) { // 방문하지 않은 도시라면
      distance[nextNode] = distance[now] + 1;
      queue.enqueue(nextNode);
    }
  }
}
// 최단 거리가 K인 모든 도시의 번호를 오름차순으로 출력
let check = false;
for (let i = 1; i <= n; i++) {
  if (distance[i] == k) {
    console.log(i);
    check = true;
  }
}
if (!check) console.log(-1); // 최단 거리가 K인 도시가 없다면
```