JavaScript BFS 알고리즘
BFS 문제 풀이

BFS 문제 풀이 | 코딩 테스트에서 자주 등장하는 BFS 알고리즘 이해하기

강사 나동빈

JavaScript
BFS 알고리즘

BFS 문제 풀이

숨바꼭질: https://www.acmicpc.net/problem/1697

혼자 힘으로 풀어보기

문제 제목: 숨바꼭질

문제 난이도: ★★☆☆☆

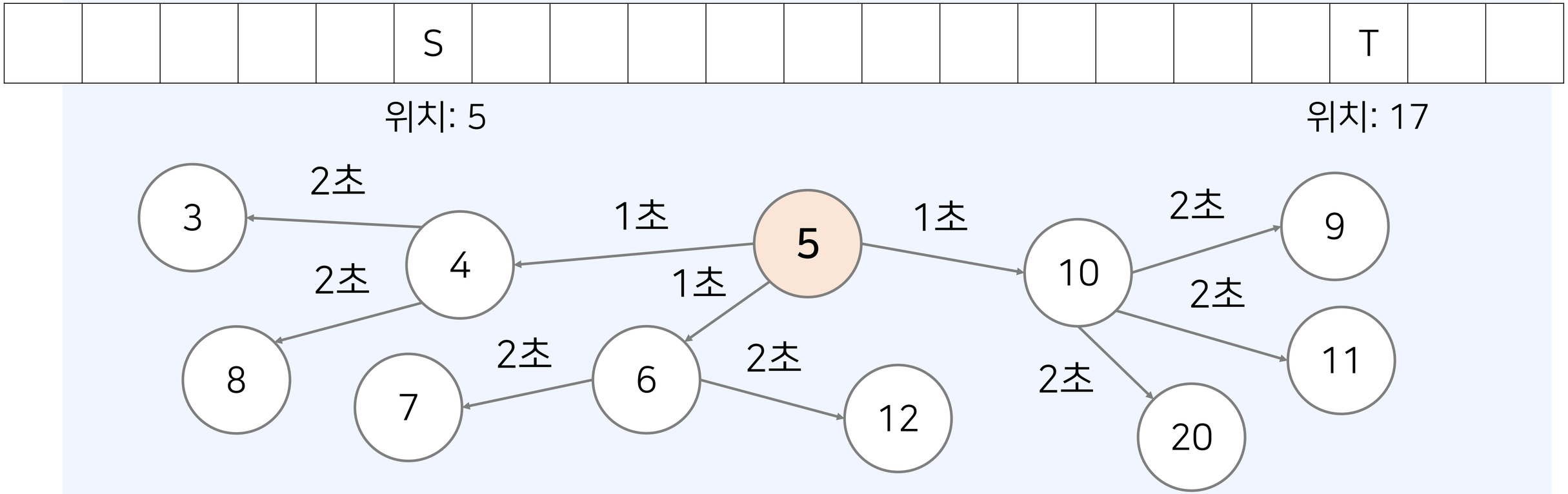
문제 유형: 너비 우선 탐색, 그래프 순회, 최단 거리

추천 풀이 시간: 40분

숨바꼭질: https://www.acmicpc.net/problem/1697

문제 해결 아이디어

- 초기 위치(N)에서 동생의 위치(M)에 도달하는 **최단 시간**을 계산하는 문제다.
- 모든 순간이동(간선)의 비용이 1초로 동일하므로, BFS로 최단 시간을 계산할 수 있다.



숨바꼭질: https://www.acmicpc.net/problem/1697

정답 코드 예시

JavaScript
BFS
BFS
문제 풀이

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

const MAX = 100001;
let [n, k] = input[0].split(' ').map(Number); // 초기 위치(N)와 동생의 위치(K)
let visited = new Array(MAX).fill(0); // 각 위치까지의 최단 시간

function bfs() { // 너비 우선 탐색(BFS)
  queue = new Queue();
  queue.enqueue(n);
  while (queue.getLength() != 0) { // 큐가 빌 때까지 반복
    let cur = queue.dequeue();
    if (cur == k) { // 동생의 위치에 도달한 경우
      return visited[cur]; // 최단 시간 출력
    }
    for (let nxt of [cur - 1, cur + 1, cur * 2]) {
      // 공간을 벗어난 경우 무시
      if (nxt < 0 || nxt >= MAX) continue;
      // 아직 방문하지 않은 위치라면
      if (visited[nxt] == 0) {
        visited[nxt] = visited[cur] + 1;
        queue.enqueue(nxt);
      }
    }
  }
}
console.log(bfs());
```

나이트의 이동: https://www.acmicpc.net/problem/7562

혼자 힘으로 풀어보기

문제 제목: 나이트의 이동

문제 난이도: ★★☆☆☆

문제 유형: 너비 우선 탐색, 그래프 순회, 최단 거리

추천 풀이 시간: 40분

나이트의 이동: https://www.acmicpc.net/problem/7562

문제 해결 아이디어

- 가중치가 없는 그래프에서의 최단 경로를 구하는 문제이므로 BFS를 사용한다.
  - 나이트가 존재하는 위치에서 BFS를 수행하여 **모든 칸까지의 최단 거리를 계산**한다.
- 각 위치에서 8가지 방향으로 이동할 수 있다는 차이점이 있다.

```
// 이동할 여덟 가지 방향 정의
dx = [-2, -2, -1, -1, 1, 1, 2, 2];
dy = [-1, 1, -2, 2, -2, 2, -1, 1];
```

나이트의 이동: https://www.acmicpc.net/problem/7562

정답 코드 예시

JavaScript
BFS
BFS
문제 풀이

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

dx = [-2, -2, -1, -1, 1, 1, 2, 2]; // 이동할 여덟 가지 방향 정의
dy = [-1, 1, -2, 2, -2, 2, -1, 1];

let testCases = Number(input[0]); // 테스트 케이스의 수
let line = 1;
while (testCases--) {
  let l = Number(input[line]);
  let [x, y] = input[line + 1].split(' ').map(Number); // 현재 위치
  let [targetX, targetY] = input[line + 2].split(' ').map(Number); // 목표 위치
  let visited = []; // 방문 정보
  for (let i = 0; i < l; i++) visited.push(new Array(l).fill(0));
  queue = new Queue(); // 너비 우선 탐색(BFS) 수행
  queue.enqueue([x, y]);
  visited[x][y] = 1;
  while (queue.getLength() != 0) {
    let cur = queue.dequeue();
    x = cur[0];
    y = cur[1];
    for (let i = 0; i < 8; i++) { // 현재 위치에서 이동하고자 하는 위치 확인
      let nx = x + dx[i];
      let ny = y + dy[i];
      if (nx < 0 || nx >= l || ny < 0 || ny >= l) continue; // 공간을 벗어난 경우 무시
      if (visited[nx][ny] == 0) { // 방문하지 않은 위치인 경우
        visited[nx][ny] = visited[x][y] + 1;
        queue.enqueue([nx, ny]);
      }
    }
  }
  line += 3; // 다음 테스트 케이스로 이동
  console.log(visited[targetX][targetY] - 1); // 항상 도달 가능
}
```