2주차 - 1926: 그림 풀이

202055524 김의현

2025-3-28

1 문제 설명

어떤 색칠된 영역이 가로/세로/대각선으로 인접한 경우, 연결되었다고 정의하자.

색칠된 영역이 서로 연결된 경우, 그 모두를 하나의 그림이라고 보자.(단, 색칠된 영역이 하나만 있어도 그림으로 본다.)

그림의 넓이는 그림이 포함하는 색칠된 영역의 개수이다.

이때 그림의 개수와 가장 넓은 그림의 넓이를 출력하시오. 단, 그림이 없는 경우 넓이는 0이다.

1.1 입력

첫째 줄에 도화지의 세로 크기 $n(1 \leq n \leq 500)$과 가로 크기 $m(1 \leq m \leq 500)$이 차례로 주어진다.

둘째 줄부터 $n+1$번째 줄까지 그림 정보가 주어진다. 0은 빈 도화지이며 1은 색칠이 된 부분이다.

2 접근법

입력이 그리 크지 않으므로 시간복잡도를 복잡하게 따질 필요는 없다. dfs/bfs를 사용하면 연속된 구간을 쉽게 찾아낼 수 있다.

도화지의 모든 점을 순회한다. 만일 해당 점이 색칠되지 않은 영역이거나, 이전에 이미 탐색하였던 영역이면 넘어간다. 그렇지 않다면 dfs/bfs로 순회하며 인접한 영역의 크기를 구하면 될 것이다.

3 풀이

```
#include <algorithm>
#include <deque>
#include <ios>
#include <iostream>
#include <vector>

int main() {
  std::ios_base::sync_with_stdio(false);
  std::cin.tie(nullptr);

  int width, height;

  std::cin >> height >> width;

  std::vector<int> board;

  board.reserve(width * height);

  for (auto i = 0; i < height; ++i) {
    for (auto j = 0; j < width; ++j) {
      int input = 0;
      std::cin >> input;
      board.push_back(input);
    }
  }

  // check pos
  std::deque<std::pair<int, int>> queue;

  size_t max_area = 0, num_area = 0;
  for (auto i = 0; i < height; ++i) {
    for (auto j = 0; j < width; ++j) {
      const auto pos = i * width + j;

      auto &cur = board[pos];
```

```
if (!cur) {
  continue;
}

++num_area;

board[i * width + j] = 0;
queue.push_back({i, j});

size_t area = 0;
while (!queue.empty()) {
  const auto [y, x] = queue.front();
  queue.pop_front();
  if (y - 1 >= 0 && board[(y - 1) * width + x]) {
    board[(y - 1) * width + x] = 0;
    queue.push_back({y - 1, x});
  }

  if (y + 1 < height && board[(y + 1) * width + x]) {
    board[(y + 1) * width + x] = 0;
    queue.push_back({y + 1, x});
  }

  if (x - 1 >= 0 && board[y * width + x - 1]) {
    board[y * width + x - 1] = 0;

    queue.push_back({y, x - 1});
  }

  if (x + 1 < width && board[y * width + x + 1]) {
    board[y * width + x + 1] = 0;
    queue.push_back({y, x + 1});
  }

  ++area;
```

```
        }

        max_area = std::max(max_area, area);
      }
    }

    std::cout << num_area << "\n" << max_area;
}
```