2

그래프의 저장과 탐색

2.1 DFS와 BFS BOJ #1260

DFS와 BFS를 연습해보는 문제입니다. 단 전제조건이 있는데, 방문할 수 있는 정점이 여러 개인 경우에는 정점 번호가 작은 것을 먼저 방문해야 합니다.

인접 행렬을 사용하면 정점 번호를 순서대로 볼 수 있지만, 공간 복잡도와 시간 복잡도가 조금 걱정입니다. 대신에 인접 리스트를 정렬하는 방식을 쓸 수 있습니다.

정렬은 많이 활용되기 때문에 배열을 정렬하는 방법은 확실히 알아두는 게 좋습니다.

정답 코드

실행 시간 0ms, 메모리 2,268KB

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm> // sort()
#include <cstring> // memset()

using namespace std;

vector<vector<int>> graph;
bool visit[1001];

void dfs(int u) {
    cout << u << ' ';
    for (int v : graph[u]) {
        if (visit[v]) continue;
        visit[v] = true;
        dfs(v);
```

```cpp
    }
}

void bfs(int start) {
    queue<int> q;
    q.emplace(start);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        cout << u << ' ';
        for (int v : graph[u]) {
            if (visit[v]) continue;
            visit[v] = true;
            q.emplace(v);
        }
    }
}

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

    int n, m, v;
    cin >> n >> m >> v;
    graph.resize(n + 1);

    while (m--) {
        int x, y;
        cin >> x >> y;
        graph[x].emplace_back(y);
        graph[y].emplace_back(x);
    }

    for (int i = 1; i <= n; i++) {
        sort(graph[i].begin(), graph[i].end()); // sort by index
    }

    visit[v] = true;
    dfs(v);
    cout << '\n';

    memset(visit, 0, sizeof(visit)); // reset visit to false
    visit[v] = true;
    bfs(v);

    return 0;
}
```

2.2 적록색약 BOJ #10026

연결 요소의 개수를 구하면 됩니다. 강의에서 언급한 BOJ #1012 유기농 배추 문제와 비슷하게 접근하면 됩니다.

처음 한 번의 탐색에서는 연결 요소의 개수를 그냥 구하고, 다음 한 번의 탐색

에서는 R과 G가 같다고 생각하고 탐색하면서 연결 요소의 개수를 구하면 됩니다. 아래 코드의 경우 첫 번째 탐색에서 연결 요소의 개수는 cc1에, 두 번째 탐색에서 연결 요소의 개수는 cc2에 저장했습니다.

정답 코드

실행 시간 0ms, 메모리 2,164KB

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <tuple>
#include <string>
#include <cstring> // memset()

using namespace std;
using pii = pair<int, int>;

int dx[4] = {-1, 1, 0, 0};
int dy[4] = {0, 0, -1, 1};

string pic[100];
int visit[100][100];

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

    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> pic[i];
    }

    queue<pii> q;
    int cc1 = 0, cc2 = 0;

    // 1st bfs
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (visit[i][j]) continue;
            cc1++;
            q.emplace(pii(i, j));

            char c = pic[i][j];
            while (!q.empty()) {
                int x = q.front().first, y = q.front().second;
                q.pop();
                for (int d = 0; d < 4; d++) {
                    int nx = x + dx[d], ny = y + dy[d];
                    if (0 > nx || nx >= n) continue; // boundary check
                    if (0 > ny || ny >= n) continue;
                    if (visit[nx][ny]) continue; // visit check
                    if (pic[nx][ny] != c) continue; // connectivity check
                    visit[nx][ny] = true;
                    q.emplace(pii(nx, ny));
                }
            }
        }
```

```
    }

    memset(visit, 0, sizeof(visit)); // reset visit to false

    // 2nd bfs
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (visit[i][j]) continue;
            cc2++;
            q.emplace(pii(i, j));

            char c = pic[i][j];
            while (!q.empty()) {
                int x = q.front().first, y = q.front().second;
                q.pop();
                for (int d = 0; d < 4; d++) {
                    int nx = x + dx[d], ny = y + dy[d];
                    if (0 > nx || nx >= n) continue; // boundary check
                    if (0 > ny || ny >= n) continue;
                    if (visit[nx][ny]) continue; // visit check
                    if (c == 'R' || c == 'G') { // connectivity check
                        if (pic[nx][ny] == 'B') continue;
                    } else if (pic[nx][ny] != c) continue;
                    visit[nx][ny] = true;
                    q.emplace(pii(nx, ny));
                }
            }
        }
    }

    cout << cc1 << ' ' << cc2;

    return 0;
}
```

BFS 부분은 코드가 비슷한 부분이 많아, 함수로 따로 만들 수도 있을 것 같아 보입니다.

2.3 연결 요소의 개수 BOJ #11724

적록색약보다 쉬운 문제인데 실수로 적록색약보다 뒤에 배치했습니다. 같은 방식으로 연결 요소의 개수를 세 주면 됩니다. 다만 이 경우 격자가 아니라 그냥 그래프이므로, 1번 노드부터 N번 노드까지 하나하나 확인하면서 방문하지 않았다면 연결 요소의 개수를 증가시켜 주는 식으로 접근하면 되겠습니다.

정답 코드

실행 시간 0ms, 메모리 2,164KB

```
#include <iostream>
#include <vector>
#include <queue>
```

```cpp
#include <tuple>
#include <cstring> // memset()

using namespace std;

bool visit[1001];

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

    int n, m;
    cin >> n >> m;

    vector<vector<int>> graph(n + 1);
    while (m--) {
        int u, v;
        cin >> u >> v;
        graph[u].emplace_back(v);
        graph[v].emplace_back(u);
    }

    int cc = 0;
    queue<int> q;

    // count connected components
    for (int i = 1; i <= n; i++) {
        if (visit[i]) continue;
        visit[i] = true;
        cc++;
        q.emplace(i);

        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int v : graph[u]) {
                if (visit[v]) continue;
                visit[v] = true;
                q.emplace(v);
            }
        }
    }

    cout << cc;

    return 0;
}
```

2.4 토마토 BOJ #7576

상자 안에 토마토가 있습니다. 하루가 지나면 인접한 토마토가 익는다고 합니다. 토마토가 전부 익는 데 며칠이 걸리는지 계산해야 합니다.

이번에는 방문 여부를 저장하는 `visit` 배열 대신 토마토가 익은 시점을 저장하는 `days` 배열을 만드는 게 좋아 보입니다. 그리고 처음에 익은 토마토가 있던

곳들부터 BFS를 진행합시다.

만약 현재 확인하고 있는 칸에 있는 토마토가 익는 데 x일 걸렸다면, 인접한 안 익은 토마토는 익는 데 $x+1$일 걸릴 것입니다. `days` 배열에서 인접한 칸들을 업데이트합시다. BFS를 마쳤을 때 `days` 배열에 저장되어 있는 최댓값이 정답입니다.

다만 주의해야 될 점은 토마토가 모두 익지는 못하는 상황이면 `-1`을 출력해야 된다는 점입니다. 이를 확인하기 위해, BFS를 진행하면서

- 토마토를 확인하는 즉시 상자 배열에서 `0`을 `1`으로 바꿔 주고
- 마지막에 토마토 상자 배열에 `0`이 하나라도 있으면 그 토마토는 익지 않았다는 뜻이 되므로 `-1`을 출력하면 되겠습니다.

안타깝게도 이 문제는 DFS로는 풀 수 없습니다. 조금 생각해 보면 알 수 있습니다.

정답 코드

실행 시간 96ms, 메모리 14,080KB

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <tuple>
#include <algorithm> // max()

using namespace std;
using pii = pair<int, int>;

int dx[4] = {-1, 1, 0, 0};
int dy[4] = {0, 0, -1, 1};

int tomato[1000][1000], days[1000][1000];

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

    int m, n;
    cin >> m >> n;

    queue<pii> q;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> tomato[i][j];
            if (tomato[i][j] == 1) {
                q.emplace(pii(i, j));
            }
        }
    }

    int ans = 0;
```

```
35
36      while (!q.empty()) {
37          int x = q.front().first, y = q.front().second;
38          q.pop();
39
40          for (int d = 0; d < 4; d++) {
41              int nx = x + dx[d], ny = y + dy[d];
42              if (0 > nx || nx >= n) continue;
43              if (0 > ny || ny >= m) continue;
44              if (tomato[nx][ny] != 0) continue; // visit check
45              tomato[nx][ny] = 1;
46              days[nx][ny] = days[x][y] + 1;
47              q.emplace(pii(nx, ny));
48              ans = max(ans, days[nx][ny]);
49          }
50      }
51
52      for (int i = 0; i < n; i++) {
53          for (int j = 0; j < m; j++) {
54              if (tomato[i][j] == 0) {
55                  ans = -1;
56                  break;
57              }
58          }
59      }
60
61      cout << ans;
62
63      return 0;
64  }
```

조금 더 어려운 문제로 연습하고 싶다면 아래 문제들을 추천합니다.

- BOJ #7569 토마토: 3차원 버전의 문제입니다.
- BOJ #17114 하이퍼 토마토: 11차원 버전의 문제입니다.

모두 같은 방법으로 해결할 수 있습니다.

2.5 경로 찾기 BOJ #11403

인접 행렬이 주어집니다. 모든 정점 (i, j)에 대해 i에서 j로 갈 수 있는지 없는지를 판단해야 합니다.

인접 행렬에서 BFS나 DFS를 하는 시간은 $\mathcal{O}\left(n^2\right)$ 입니다. 다행히도 $N \leq 100$이기 때문에 모든 정점 i에서부터 BFS 혹은 DFS를 돌리면서 경로가 있는지 없는지를 확인해 볼 수 있습니다.

강의에서는 인접 행렬에서 DFS/BFS를 하는 건 다루지 않았습니다만, 아래 코드에서는 인접 행렬에서 DFS를 다루고 있습니다. 줄 11과 같은 식으로 진행하면 됩니다. `start`는 단순히 `path`가 있고 없음의 여부를 저장하기 위한 배열입니다.

정답 코드

실행 시간 4ms, 메모리 2,036KB

```cpp
#include <iostream>
#include <cstring> // memset()

using namespace std;

int n;
int graph[100][100];
bool visit[100], path[100][100];

void dfs(int u, int start) {
    for (int v = 0 ; v < n; v++) {
        if (visit[v]) continue;
        if (!graph[u][v]) continue;
        visit[v] = true;
        path[start][v] = true;
        dfs(v, start);
    }
}

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

    cin >> n;

    for (int i = 0 ; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> graph[i][j];
        }
    }

    for (int i = 0; i < n; i++) {
        memset(visit, 0, sizeof(visit));
        dfs(i, i);
    }

    for (int i = 0 ; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << path[i][j] << ' ';
        }
        cout << '\n';
    }

    return 0;
}
```

사실 이 코드는 조금 더 최적화할 수 있습니다. 줄 14–15를 보고 잘 생각해 보면 `path` 배열은 필요없음을 알 수 있습니다. 대신 DFS가 끝나는 줄 36에 `visit` 배열의 내용만 출력해 줘도 됩니다.

2.6 케빈 베이컨의 6단계 법칙 BOJ #1389

가중치가 없는 그래프의 경우, BFS를 이용하면 최단 거리를 구할 수 있습니다.

예를 들어 어떤 노드부터 현재 확인하고 있는 노드까지의 거리가 x라고 하면, 거기에 인접해 있으면서 아직 방문하지 않은 노드까지의 거리는 $x+1$입니다. 이름부터 '너비 우선 탐색'인 BFS는 현재 노드에서 가까운 정점들부터 확인한다는 특징을 가지고 있기 때문입니다. (DFS는 대부분의 상황에서는 이런 걸 할 수 없습니다.)

위의 '토마토' 문제의 경우 각 칸의 `days` 배열에 저장된 수는 실제로 그 칸에서부터 처음에 익어 있던 토마토까지의 최단 거리가 저장되어 있음을 알 수 있습니다.

이번에도 $N \leq 100$이기 때문에, 위에 등장했던 '경로 찾기' 문제와 '토마토' 문제의 아이디어를 섞어 BFS를 N번 돌리면서 계산한 거리를 다 더해 주면 되겠습니다.

정답 코드

실행 시간 0ms, 메모리 1,988KB

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <cstring> // memset()

using namespace std;

int n, m, dist[101];
vector<vector<int>> graph;

int bacon(int start) {
    memset(dist, 0, sizeof(dist));
    // bfs
    queue<int> q;
    q.emplace(start);
    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (int v : graph[u]) {
            if (dist[v] > 0) continue;
            if (v == start) continue;
            dist[v] = dist[u] + 1;
            q.emplace(v);
        }
    }

    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += dist[i];
    }
    return sum;
}

int main() {
    cin.tie(nullptr);
```

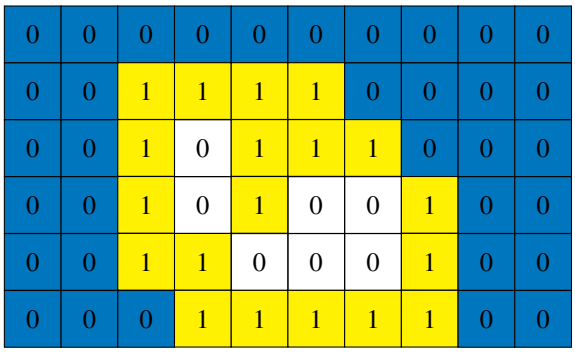
```
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    cin >> n >> m;
    graph.resize(n + 1);

    while (m--) {
        int u, v;
        cin >> u >> v;
        graph[u].emplace_back(v);
        graph[v].emplace_back(u);
    }

    int mn = 987654321, mni;

    for (int i = 1; i <= n; i++) {
        int bc = bacon(i);
        if (mn > bc) {
            mn = bc;
            mni = i;
        }
    }

    cout << mni;

    return 0;
}
```

2.7 치즈 BOJ #2636

2000년 정보올림피아드 초등부 3번 문제입니다. 치즈가 녹아 없어집니다. 몇 시간이 지나야 치즈가 전부 녹아 없어지는지를 계산해야 합니다. 크기가 최대 100×100인데, 만약 치즈가 100×100칸에 꽉 차 있다면 녹는 데에는 50시간이 걸릴 것이라고 짐작할 수 있습니다.

빈 칸에 인접해 있는 치즈를 녹이면 될 거 같은데, 단순히 그렇게 하기에는 빈 칸이 치즈 안쪽에 있는지(치즈 안의 구멍인지) 바깥쪽에 있는지 구별해야 할 거 같습니다. 이를 어떻게 효율적으로 구별할 수 있을까요?

생각해 보면, 바깥쪽의 빈 공간은 연결 요소를 이루고 있습니다.



따라서 일단 바깥쪽의 빈 공간을 연결 요소로 구해 주고, 거기에 인접해 있는 치즈들만 녹여 주면 됩니다. 이 작업은 $\mathcal{O}(n^2)$이고 최대 50번 녹이므로 충분히 시간 안에 풀 수 있습니다.

정답 코드

실행 시간 0ms, 메모리 2,040KB

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <cstring> // memset()

using namespace std;
using pii = pair<int, int>;

int dx[4] = {-1, 1, 0, 0};
int dy[4] = {0, 0, -1, 1};

int n, m;

int ch[102][102];
bool visit[102][102];

int melt() {
    memset(visit, 0, sizeof(visit));

    queue<pii> q;
    visit[0][0] = true;
    q.emplace(pii(0, 0));

    while (!q.empty()) {
        int x = q.front().first, y = q.front().second;
        q.pop();

        for (int d = 0; d < 4; d++) {
            int nx = x + dx[d], ny = y + dy[d];
            if (0 > nx || nx >= n + 2) continue;
            if (0 > ny || ny >= m + 2) continue;
            if (visit[nx][ny]) continue;
            visit[nx][ny] = true;
            if (ch[nx][ny] == 0) {
                q.emplace(pii(nx, ny));
            }
```

```
        }
    }

    int melted = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (visit[i][j] && ch[i][j] == 1) {
                ch[i][j] = 0;
                melted++;
            }
        }
    }

    return melted;
}

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

    cin >> n >> m;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> ch[i][j];
        }
    }

    int hours = 0, last = 0;
    while (true) {
        int melted = melt();
        if (!melted) break;
        last = melted;
        hours++;
    }

    cout << hours << '\n' << last;

    return 0;
}
```

배열을 입력받을 때 상하좌우 1칸씩 여유를 줬습니다. 바깥쪽의 빈 공간을 연결 요소로 구하고자 하는데, 바깥쪽에 빈 공간이 없으면 연결 요소를 구하기 곤란해지기 때문입니다.

이 문제는 한 변만 실내 공기와 접촉하면 치즈가 녹았지만, 같은 해 중등부에서는 BOJ #2638 두 변이 실내 공기와 접촉해야 치즈가 녹는 문제가 나왔습니다. 접근은 비슷하지만 조금 더 생각해야 풀리는 문제입니다.

2.8 숨바꼭질 BOJ #1697

수빈이가 동생을 찾을 수 있는 가장 빠른 시간을 구해야 합니다. 전혀 그래프같지 않아 보이지만 놀랍게도 BFS로 풀 수 있습니다.

점들을 전부 노드로 생각합시다. 그리고, 모든 노드 x에 대해

$$x \to x-1 \qquad x \to x+1 \qquad x \to 2x$$

의 단방향 간선들이 있다고 생각합시다. 그러면 이 문제는 n에서 k까지 가는 최단 경로를 찾는 문제로 바뀝니다. 이는 위의 문제들처럼 BFS로 쉽게 풀 수 있습니다.

정답 코드

실행 시간 0ms, 메모리 2,512KB

```cpp
#include <iostream>
#include <queue>

using namespace std;

int dist[100001];

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

    int n, k;
    cin >> n >> k;
    if (n == k) {
        cout << 0;
        return 0;
    }

    queue<int> q;
    q.emplace(n);

    while (!q.empty()) {
        int x = q.front();
        q.pop();

        int next[3] = {x - 1, x + 1, 2 * x};
        for (int d = 0; d < 3; d++) {
            int y = next[d];
            if (y < 0 || y > 100000) continue;
            if (dist[y] > 0 || y == n) continue;
            dist[y] = dist[x] + 1;
            q.emplace(y);
        }

        if (dist[k] > 0) {
            cout << dist[k];
            break;
```

```
        }
    }

    return 0;
}
```

2.9 DSLR BOJ #9019

카메라와 관련된 문제일 줄 알았는데 웬 레지스터가 어쩌고 명령어가 어쩌고 해서 당황하셨을 겁니다. 결론은 위의 '숨바꼭질'과 같은 식으로 접근하면 됩니다. 하지만 이 문제는 명령의 개수뿐만 아니라 명령어를 어떻게 써야 최소가 되는지를 나열하라고 합니다.

경로를 저장하는 배열 v를 하나 만듭니다. 만약에 $x \to_{\mathrm{L}} y$라면, `v[y] = {x, 'L'}`과 같은 식으로 y에 도달하기 직전의 값과 명령어를 저장합니다.

BFS를 이용해 A에서 B까지 도달했다면 이제 이 배열을 이용해 B에서부터 다시 이전 한 명령씩 돌아옵니다. 명령 목록을 스택에 쌓고, 스택에서 pop 하면서 하나 하나 출력해 주면 됩니다. 이 부분은 줄 79–90에 구현되어 있습니다.

정답 코드

실행 시간 816ms, 메모리 2,196KB

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <tuple>
#include <queue>
#include <stack>
#include <cstring>

using namespace std;
using pic = pair<int, char>;

int d(int n) {
    return (n * 2) % 10000;
}

int s(int n) {
    if (n > 0) return n - 1;
    return 9999;
}

int l(int n) {
    int left = n / 1000;
    return (n * 10 + left) % 10000;
}

int r(int n) {
    int right = n % 10;
```

```cpp
    return n / 10 + right * 1000;
}

pic v[10000];

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

    int t;
    cin >> t;

    while (t--) {
        int a, b;
        cin >> a >> b;

        memset(v, 0, sizeof(v));

        queue<int> q;
        v[a] = pic(-1, '_');
        q.emplace(a);

        while (!q.empty()) {
            int x = q.front();
            q.pop();

            int dx = d(x), sx = s(x), lx = l(x), rx = r(x);

            if (v[dx].second == 0) {
                v[dx] = pic(x, 'D');
                if (dx == b) break;
                q.emplace(dx);
            }
            if (v[sx].second == 0) {
                v[sx] = pic(x, 'S');
                if (sx == b) break;
                q.emplace(sx);
            }
            if (v[lx].second == 0) {
                v[lx] = pic(x, 'L');
                if (lx == b) break;
                q.emplace(lx);
            }
            if (v[rx].second == 0) {
                v[rx] = pic(x, 'R');
                if (rx == b) break;
                q.emplace(rx);
            }
        }

        stack<char> rs;
        int i = b;
        while (i != a) {
            pic x = v[i];
            rs.emplace(x.second);
            i = x.first;
        }

        while (!rs.empty()) {
            cout << rs.top();
            rs.pop();
        }
```

```
91
92          cout << '\n';
93      }
94
95      return 0;
96  }
```

2.10 불 BOJ #5427

일단 불이 몇 초 후에 어디에 도달할 수 있는지를 먼저 계산해 놓습니다. BFS 한 번으로 가능합니다.

그 후에는 상근이가 몇 초 후에 어디에 도달할 수 있는지를 계산합니다. 미리 불이 몇 초 후에 어디에 도달하는지 계산해 놓은 결과를 이용해, 상근이가 어떤 칸에 도달하는 시점에 불이 이미 퍼져 있었거나 상근이와 불이 동시에 같은 칸에 도달하는 경우는 무시하고 BFS를 진행합니다. (줄 72–73)

마지막에는 가장자리 칸들을 검사하면서 가장자리 칸에 도달한 최단 시간 + 1을 출력해 주면 됩니다.

정답 코드

실행 시간 100ms, 메모리 13,032KB

```
1   #include <iostream>
2   #include <vector>
3   #include <algorithm>
4   #include <tuple>
5   #include <queue>
6   #include <stack>
7   #include <cstring>
8
9   using namespace std;
10  using pii = pair<int, int>;
11
12  string b[1000];
13  int distf[1000][1000], dists[1000][1000];
14
15  int dx[4] = {-1, 1, 0, 0};
16  int dy[4] = {0, 0, -1, 1};
17
18  int main() {
19      cin.tie(nullptr);
20      cout.tie(nullptr);
21      ios_base::sync_with_stdio(false);
22
23      int t;
24      cin >> t;
25
26      while (t--) {
27          memset(distf, -1, sizeof(distf));
28          memset(dists, -1, sizeof(dists));
```

```
29
30      queue<pii> qf, qs;
31
32      int m, n;
33      cin >> m >> n;
34      for (int i = 0; i < n; i++) {
35          cin >> b[i];
36          for (int j = 0; j < m; j++) {
37              if (b[i][j] == '*') {
38                  qf.emplace(pii(i, j));
39                  distf[i][j] = 0;
40              } else if (b[i][j] == '@') {
41                  qs.emplace(pii(i, j));
42                  dists[i][j] = 0;
43              }
44          }
45      }
46
47      while (!qf.empty()) {
48          int x = qf.front().first, y = qf.front().second;
49          qf.pop();
50
51          for (int d = 0; d < 4; d++) {
52              int nx = x + dx[d], ny = y + dy[d];
53              if (0 > nx || nx >= n) continue;
54              if (0 > ny || ny >= m) continue;
55              if (distf[nx][ny] != -1) continue;
56              if (b[nx][ny] == '#') continue;
57              distf[nx][ny] = distf[x][y] + 1;
58              qf.emplace(pii(nx, ny));
59          }
60      }
61
62      while (!qs.empty()) {
63          int x = qs.front().first, y = qs.front().second;
64          qs.pop();
65
66          for (int d = 0; d < 4; d++) {
67              int nx = x + dx[d], ny = y + dy[d];
68              if (0 > nx || nx >= n) continue;
69              if (0 > ny || ny >= m) continue;
70              if (dists[nx][ny] != -1) continue;
71              if (b[nx][ny] == '#') continue;
72              if (distf[nx][ny] != -1
73                  && distf[nx][ny] <= dists[x][y] + 1) continue;
74              dists[nx][ny] = dists[x][y] + 1;
75              qs.emplace(pii(nx, ny));
76          }
77      }
78
79      int mn = 987654321;
80      for (int i = 0; i < n; i++) {
81          if (dists[i][0] != -1) mn = min(mn, dists[i][0]);
82          if (dists[i][m - 1] != -1) mn = min(mn, dists[i][m - 1]);
83      }
84      for (int j = 0; j < m; j++) {
85          if (dists[0][j] != -1) mn = min(mn, dists[0][j]);
86          if (dists[n - 1][j] != -1) mn = min(mn, dists[n - 1][j]);
87      }
88
89      if (mn == 987654321) {
90          cout << "IMPOSSIBLE\n";
91      } else {
```

```
            cout << mn + 1 << '\n';
        }
    }

    return 0;
}
```

2.11 벽 부수고 이동하기 BOJ #2206

최단 거리를 저장하는 배열을 두 개 만듭니다. 하나에는 아직 벽을 부수지 않았을 때의 최단 거리, 다른 하나에는 벽을 한 번 부수고 난 후의 최단 거리를 저장합니다.

두 그래프에서 동시에 BFS를 하되 인접한 칸이 벽일 경우에는 벽을 한 번 부수고 난 후의 최단 거리를 벽을 부수지 않았을 때의 최단 거리 + 1로 업데이트해 줍니다. 줄 51–55를 참고하면 됩니다.

정답 코드

실행 시간 84ms, 메모리 13,844KB

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <tuple>
#include <queue>
#include <cstring>

using namespace std;
using pii = pair<int, int>;

int b[1000][1000], r1[1000][1000], r2[1000][1000];

int dx[] = {0, 0, 1, -1};
int dy[] = {1, -1, 0, 0};

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

    memset(r1, -1, sizeof(r1));
    memset(r2, -1, sizeof(r2));

    int n, m;
    cin >> n >> m;

    for (int i = 0; i < n; i++) {
        string s;
        cin >> s;
        for (int j = 0; j < m; j++) {
            b[i][j] = s[j] - '0';
        }
    }
```

```
34
35      queue<pii> q; // x, y
36      r1[0][0] = 1;
37      q.emplace(0, 0);
38
39      while (!q.empty()) {
40          int x = q.front().first, y = q.front().second;
41          q.pop();
42
43          for (int d = 0; d < 4; d++) {
44              int nx = x + dx[d], ny = y + dy[d];
45
46              if (nx < 0 || n <= nx) continue;
47              if (ny < 0 || m <= ny) continue;
48
49              bool flag = false;
50
51              // destroy wall - update r2 based on r1
52              if (b[nx][ny] && r1[x][y] != -1 && r2[nx][ny] == -1) {
53                  r2[nx][ny] = r1[x][y] + 1;
54                  flag = true;
55              }
56
57              // don't destroy wall - update r1 based on r1
58              if (!b[nx][ny] && r1[x][y] != -1 && r1[nx][ny] == -1) {
59                  r1[nx][ny] = r1[x][y] + 1;
60                  flag = true;
61              }
62
63              // don't destroy wall - update r2 based on r2
64              if (!b[nx][ny] && r2[x][y] != -1 && r2[nx][ny] == -1) {
65                  r2[nx][ny] = r2[x][y] + 1;
66                  flag = true;
67              }
68
69              if (flag) {
70                  q.emplace(pii(nx, ny));
71              }
72          }
73      }
74
75      if (r1[n - 1][m - 1] == -1) {
76          if (r2[n - 1][m - 1] == -1) {
77              cout << -1;
78          } else {
79              cout << r2[n - 1][m - 1];
80          }
81      } else {
82          if (r2[n - 1][m - 1] == -1) {
83              cout << r1[n - 1][m - 1];
84          } else {
85              cout << min(r1[n - 1][m - 1], r2[n - 1][m - 1]);
86          }
87      }
88
89      return 0;
90  }
```

이 문제도 다른 버전이 있습니다.

- BOJ #14442 벽 부수고 이동하기 2: 벽을 최대 $K \leq 10$개까지 부숴도 괜찮습니다.
- BOJ #16933 벽 부수고 이동하기 3: 벽을 최대 $K \leq 10$개까지 부숴도 괜찮지만 벽은 낮에만 부술 수 있습니다.
- BOJ #16946 벽 부수고 이동하기 4: 벽을 부쉈을 때 그 칸에서 이동할 수 있는 칸의 개수를 세야 합니다.

벽 부수고 이동하기 2와 3은 조금 더 생각해 이 문제와 비슷한 방법으로 접근하면 해결할 수 있습니다. 벽 부수고 이동하기 4는 이 문제와는 다른 방법으로 접근해야 하지만, DFS/BFS만으로 충분히 해결할 수 있는 문제입니다.