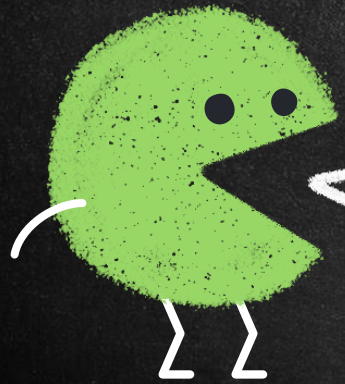
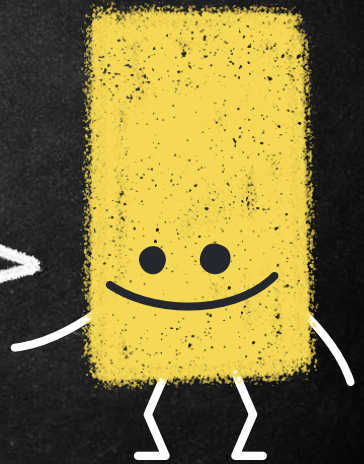
Algorithm Study For 2021

WEEK 5



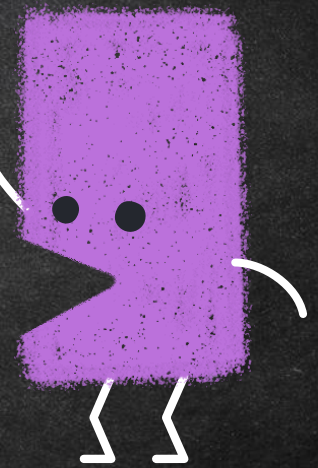


“

이번 주에는 그래프를 복습하고

BFS/DFS 구현 및 문제풀이를

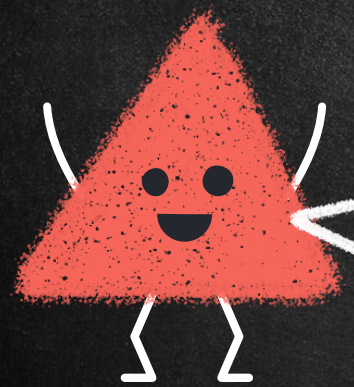
진행합니다아!

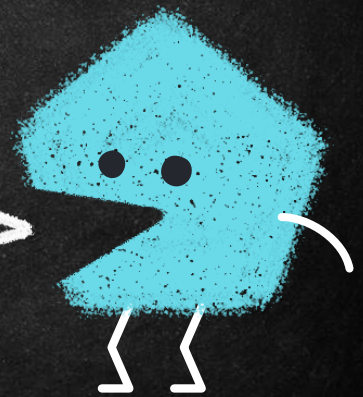


Week 5

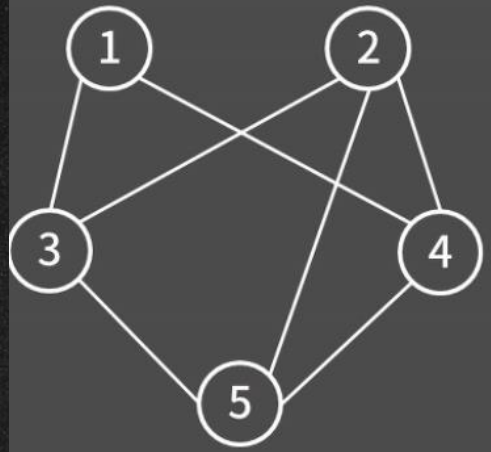
1.

GRAPH REVIEW !!!



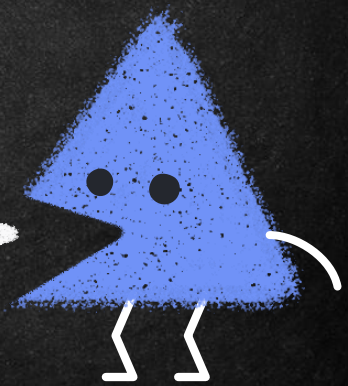


GRAPH 표현방법 1 - 인접행렬

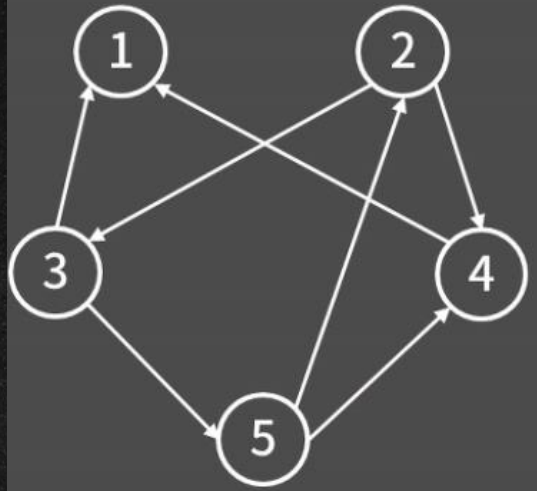


	1	2	3	4	5
1	0	0	1	1	0
2	0	0	1	1	1
3	1	1	0	0	1
4	1	1	0	0	1
5	0	1	1	1	0

- 첫번째 방법은 인접행렬입니다.
- 편의상 정점을 2차원으로 확장시켜서 연결되면 1, 연결되지 않으면 0을 줍니다.
- 이 그림에서는 무방향 그래프이기 때문에 대각선에 대해서 대칭인 형태가 나옵니다.

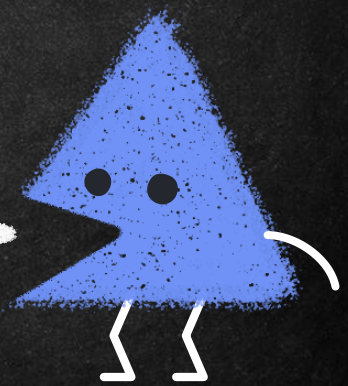


GRAPH 표현방법 1 - 인접행렬

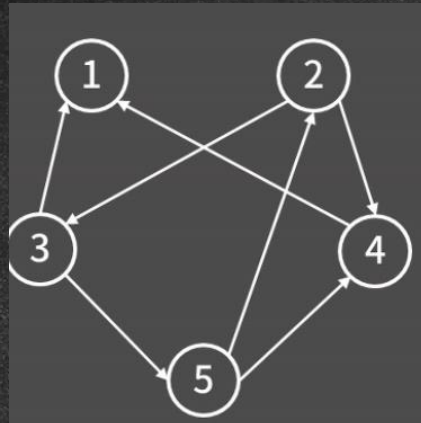


	1	2	3	4	5
1	0	0	0	0	0
2	0	0	1	1	0
3	1	0	0	0	1
4	1	0	0	0	0
5	0	1	0	1	0

- 첫번째 방법은 인접행렬입니다.
- 편의상 정점을 2차원으로 확장시켜서 연결되면 1, 연결되지 않으면 0을 줍니다.
- 이 그림에서는 방향 그래프이기 때문에 대각선에 대해서 대칭이 아닙니다.



GRAPH 표현방법 1 - 인접행렬 구현



입력

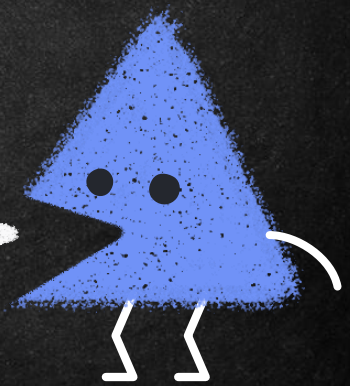
```
5 7
3 1
2 3
4 1
5 2
5 4
3 5
2 4
```

Directed Graph

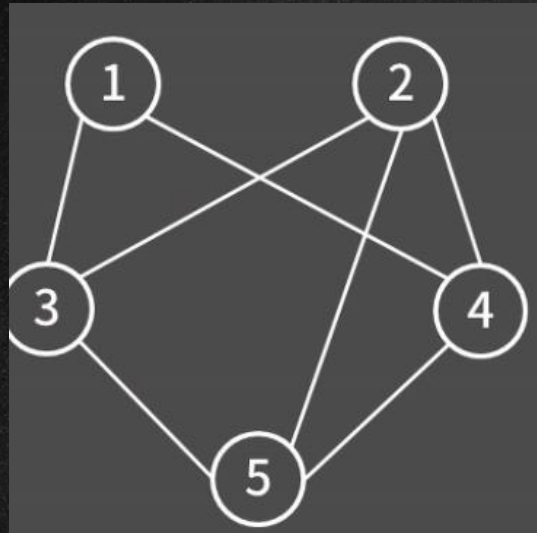
```
00 int adj_matrix[10][10]={};
01 int v, e;
02 cin >> v >> e;
03 for(int i = 0; i < e; i++){
04   int u, v;
05   cin >> u >> v;
06   adj_matrix[u][v] = 1;
07 }
```

Undirected Graph

```
00 int adj_matrix[10][10]={};
01 int v, e;
02 cin >> v >> e;
03 for(int i = 0; i < e; i++){
04   int u, v;
05   cin >> u >> v;
06   adj_matrix[u][v] = 1;
07   adj_matrix[v][u] = 1;
08 }
```

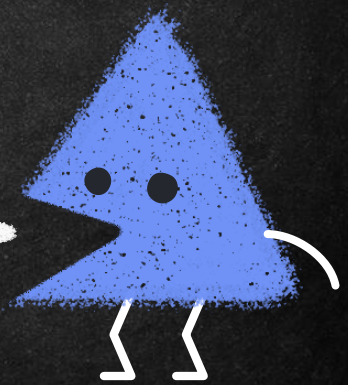


GRAPH 표현방법 2 - 인접리스트

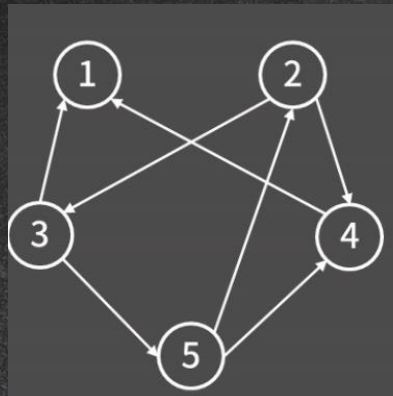


1	3	4	
2	3	5	4
3	1	2	5
4	2	1	5
5	3	2	4

- 두번째 방법은 인접 리스트입니다.
- 정점의 개수에 맞춰서 v개의 리스트를 만들어 각 리스트에 연결된 정점을 넣어주면 됩니다!



GRAPH 표현방법 2 - 인접리스트 구현



입력

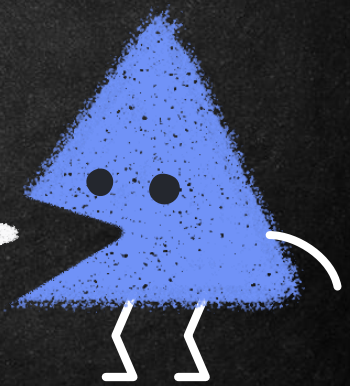
```
5 7
3 1
2 3
4 1
5 2
5 4
3 5
2 4
```

Directed Graph

```
00 vector<int> adj[10];
01 int v, e;
02 cin >> v >> e;
03 for(int i = 0; i < e; i++){
04   int u, v;
05   cin >> u >> v;
06   adj[u].push_back(v);
07 }
```

Undirected Graph

```
00 vector<int> adj[10];
01 int v, e;
02 cin >> v >> e;
03 for(int i = 0; i < e; i++){
04   int u, v;
05   cin >> u >> v;
06   adj[u].push_back(v);
07   adj[v].push_back(u);
08 }
```

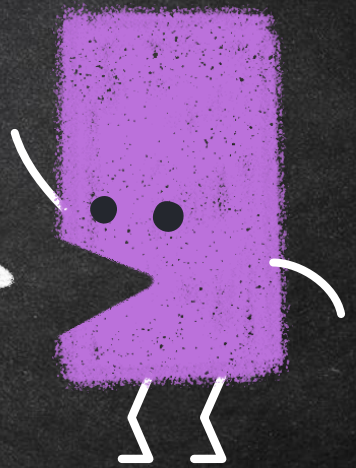


“

DFS/BFS가 뭔데요!!?

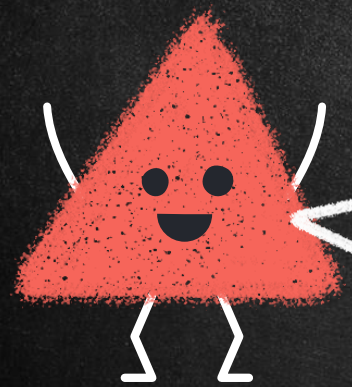
(필수시청-9분 쉬움)

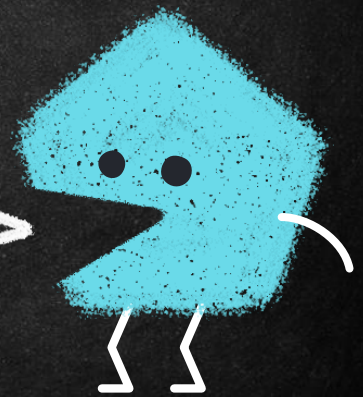
https://www.youtube.com/watch?v=-ws4Ytm0x3nw



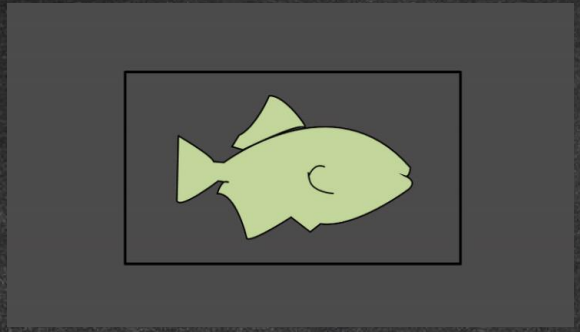
2.

BFS!!!

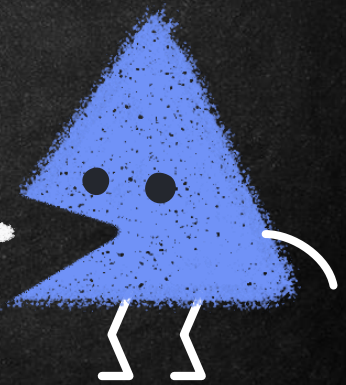




BFS 알고리즘 설명



- BFS를 알아보기 전에 익숙한 문제를 생각해봅시다.
- 그림판의 페인트 기능을 활용하면 물고기의 색을 바꿀 수 있습니다. 페인트 기능은 외부 윤곽선을 따라서 구분되는 영역의 색을 한번에 바꾸는 거고, 이거를 Flood Fill이라고 합니다.
- 클릭한 칸의 상하좌우를 보여 나와 색이 같은지 확인하는 과정의 반복을 진행하겠죠. BFS로 해결합니다.

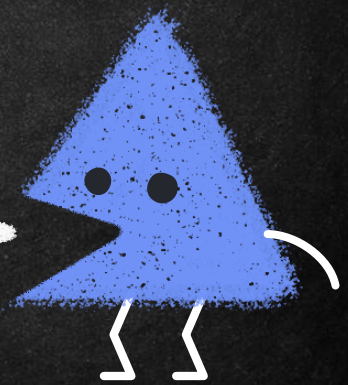


BFS 알고리즘 설명

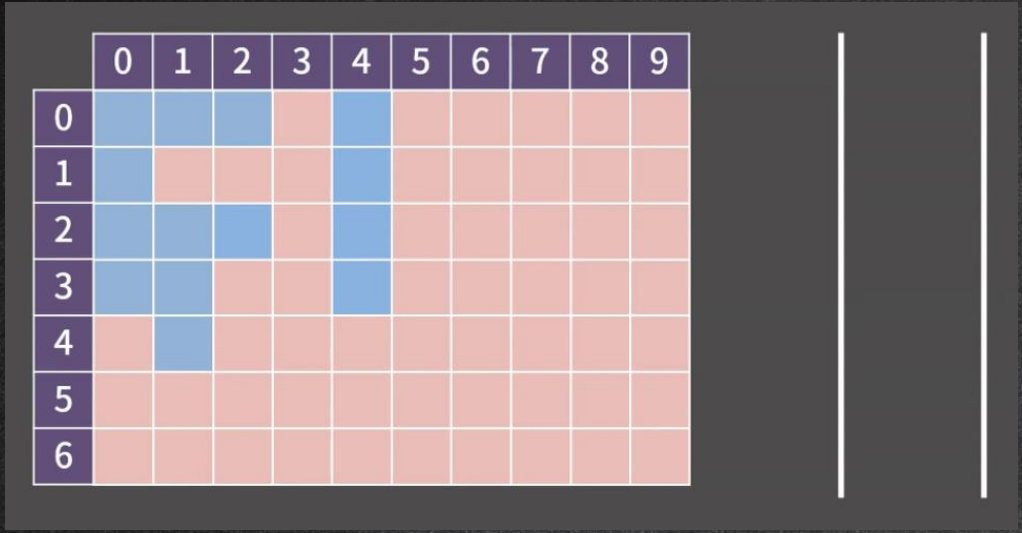
BFS(Breadth First Search)

- 다차원 배열에서 각 칸을 방문할 때 너비를 우선으로 방문하는 알고리즘
- *너비를 우선으로 방문한다...?!!???? 뭔 소리죠??*

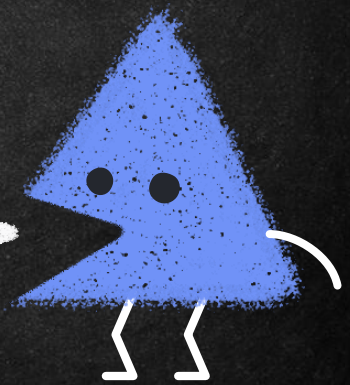
- *그림을 통해서 이해해봅시다!!!*



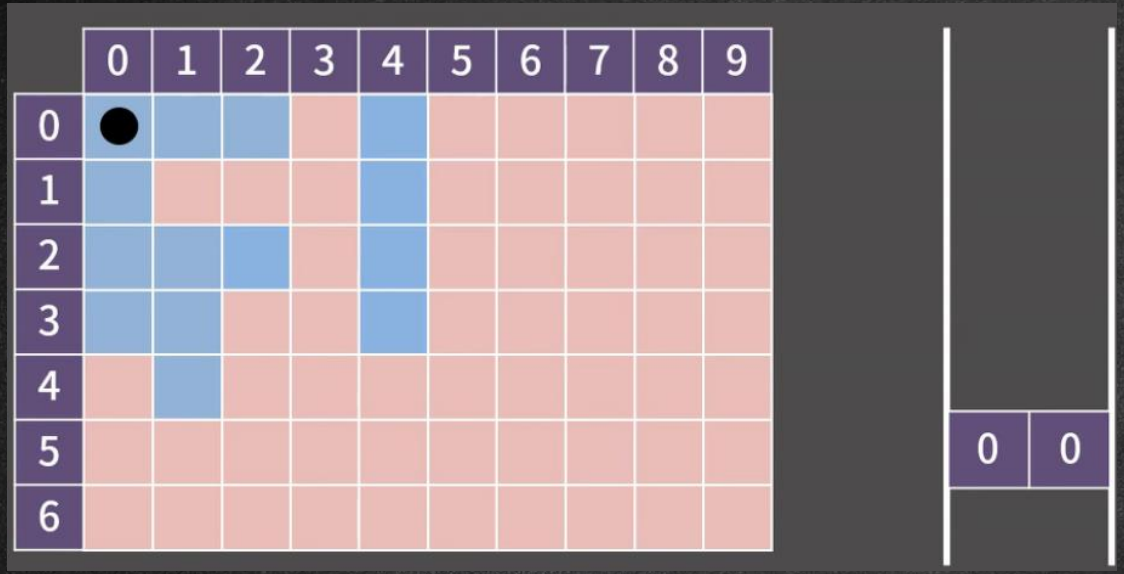
BFS 알고리즘 설명 그림으로!!!



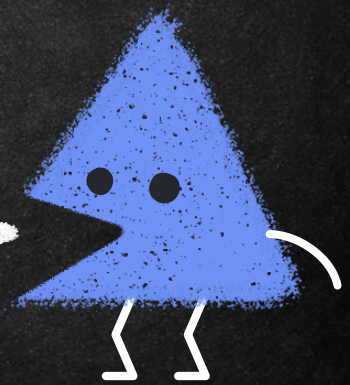
→ 우리의 목표는 (0,0)과 상하좌우로 이어진 모든 파란색 칸을 확인하는 것입니다. 이 문제를 BFS로 어떻게 해결하는지 보겠습니다.



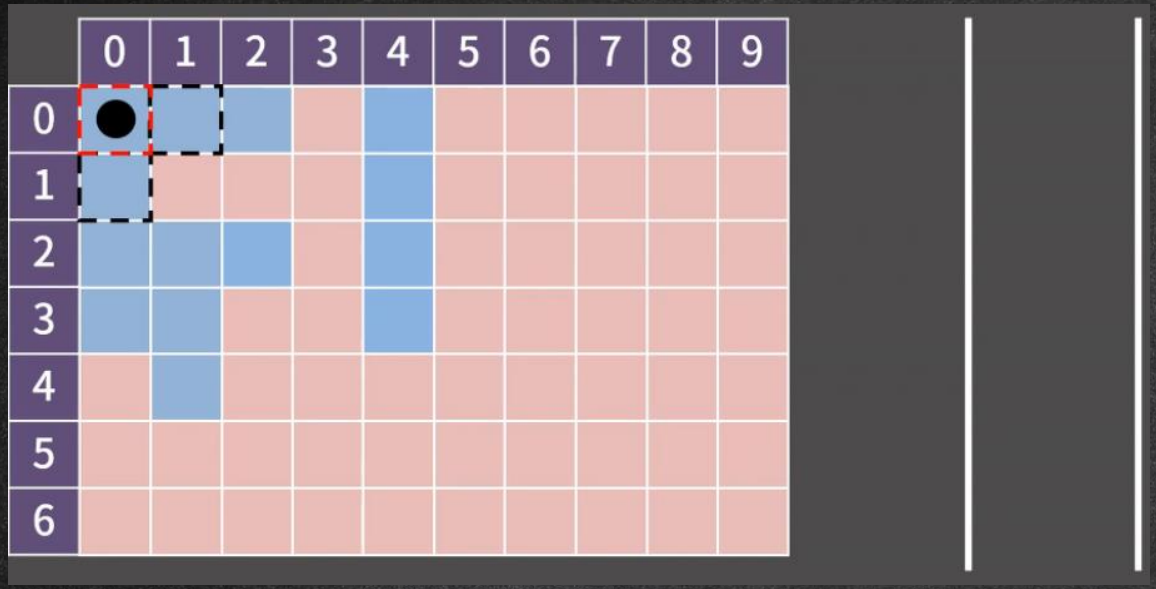
BFS 알고리즘 설명 그림으로!!!



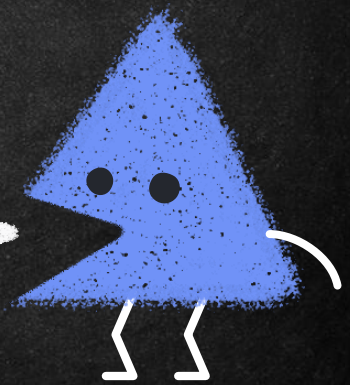
우선 BFS 알고리즘에서는 좌표를 담을 큐가 필요합니다. BFS 알고리즘이 시작되면 우선 (0, 0)에 방문했다는 표시를 남기고 해당 칸을 큐에 넣습니다. 이 초기 세팅이 끝난 후에는 큐가 빌 때까지 계속 큐의 front를 빼고 해당 좌표의 상하좌우를 살펴보면서 큐에 넣어주는 작업을 반복하게 됩니다.



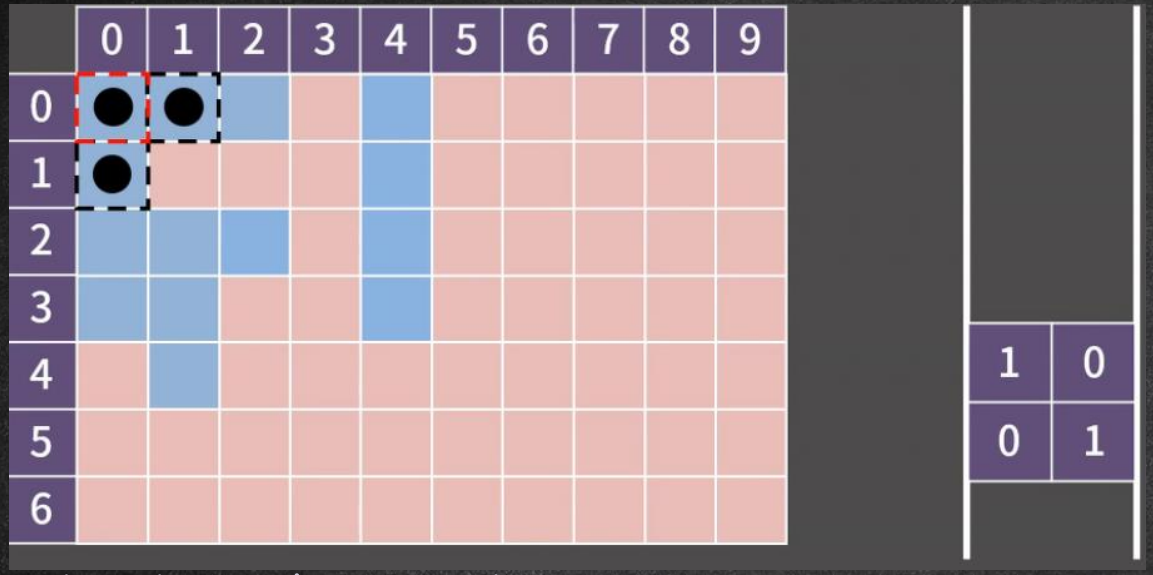
BFS 알고리즘 설명 그림으로!!!



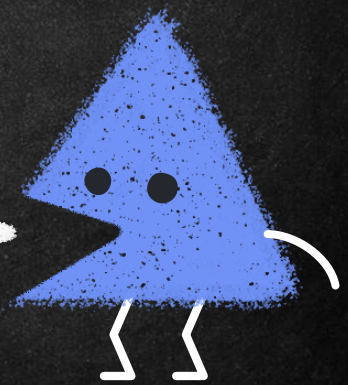
→ 지금 상황에서 q의 front는 (0,0)이었고, 이것의 상하좌우를 보는데, 이 중에서 우리는 파란색 칸이면서 아직 방문하지 않은 칸을 찾습니다. (0,1) (1,0)을 큐에!!!



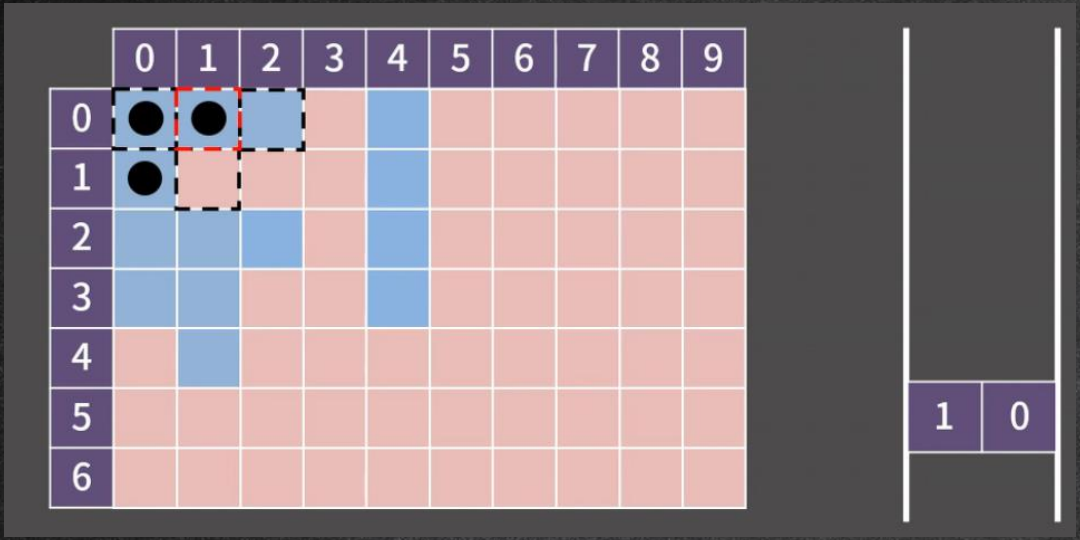
BFS 알고리즘 설명 그림으로!!!



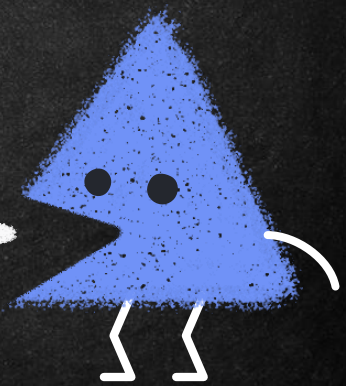
- (0,0)에서 할것은 다했고, 다음으로 넘어갑니다.
- 현재 큐의 front는(0,1)이고, pop을 합니다. 상하좌우를 다시 확인하고 (1,1)은 빨간칸, (0,0)은 방문했으므로 (0,2)만 방문 표시 후에 큐에 삽입!!!



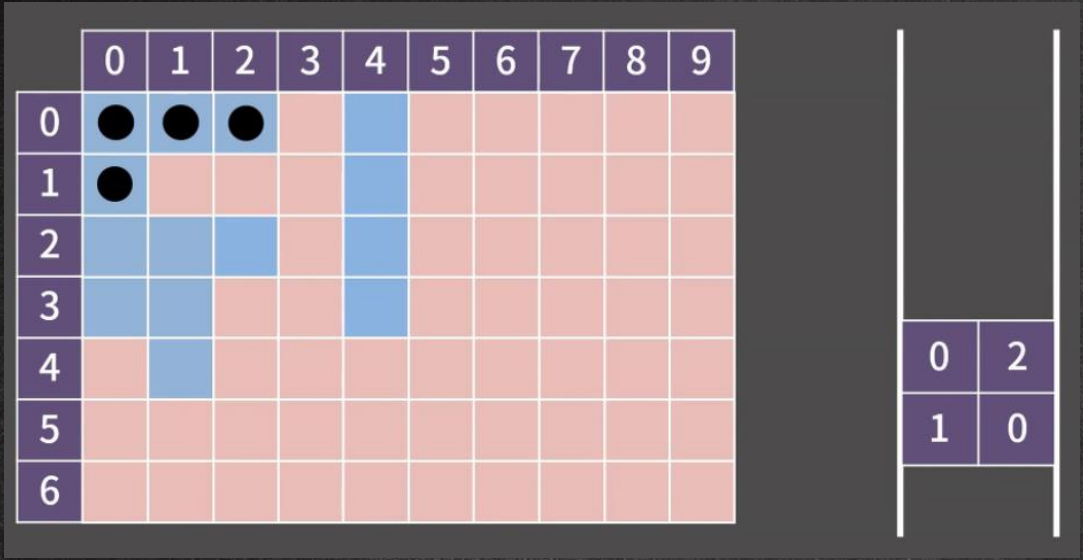
BFS 알고리즘 설명 그림으로!!!



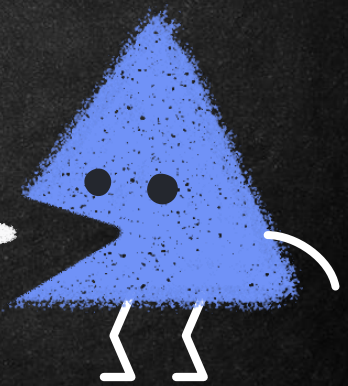
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



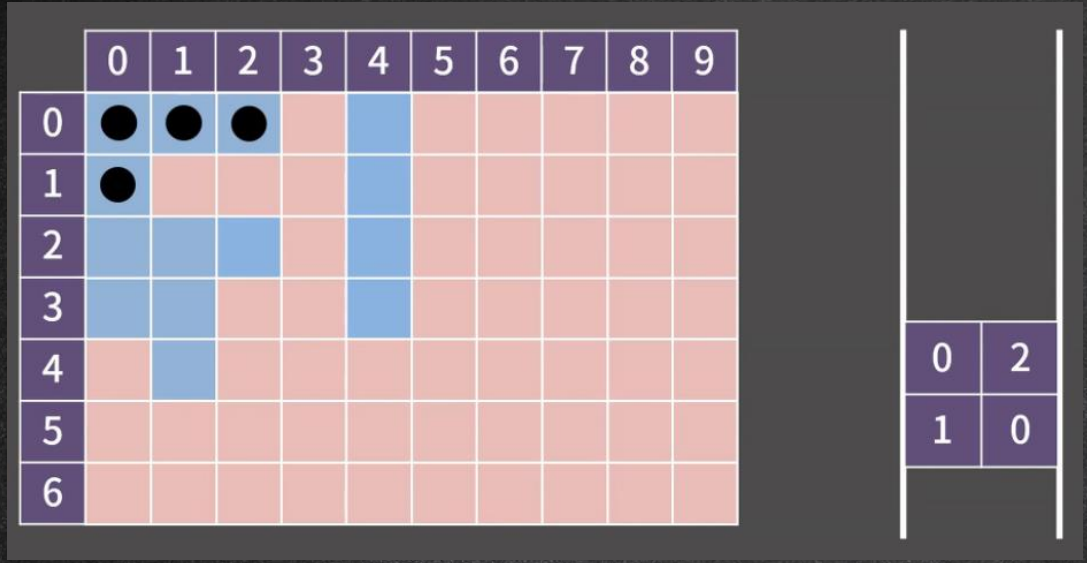
BFS 알고리즘 설명 그림으로!!!



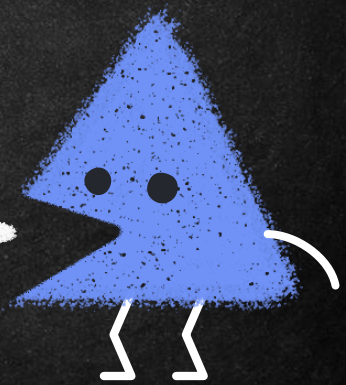
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



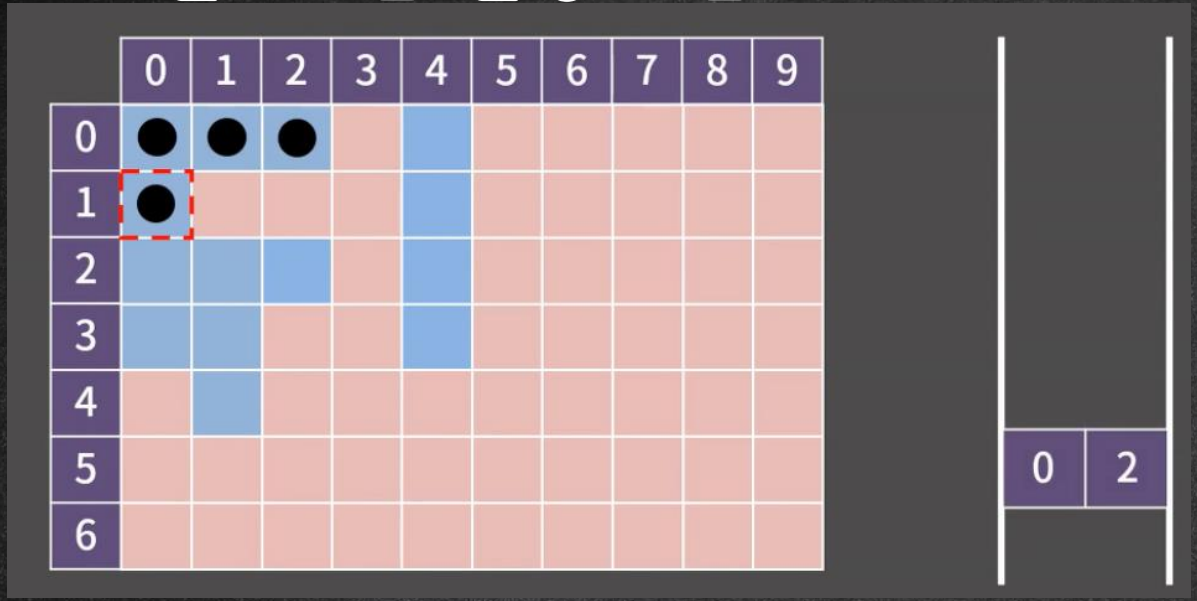
BFS 알고리즘 설명 그림으로!!!



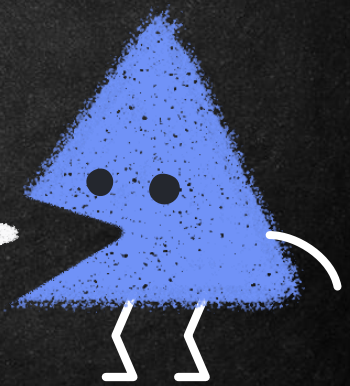
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



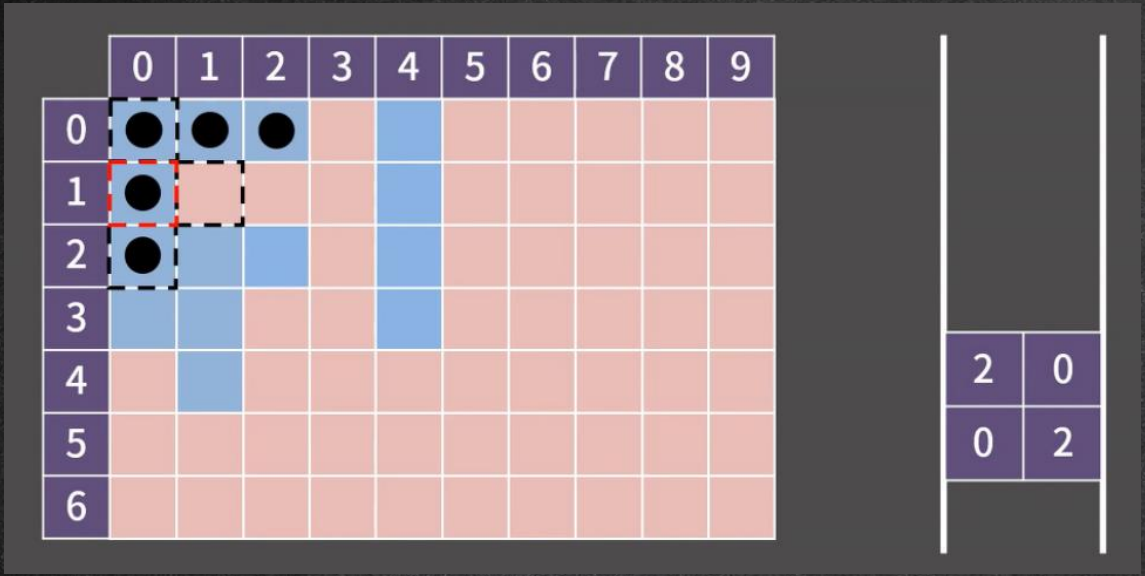
BFS 알고리즘 설명 그림으로!!!



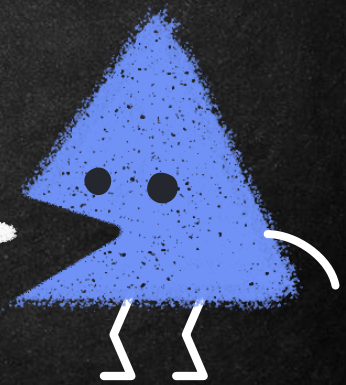
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



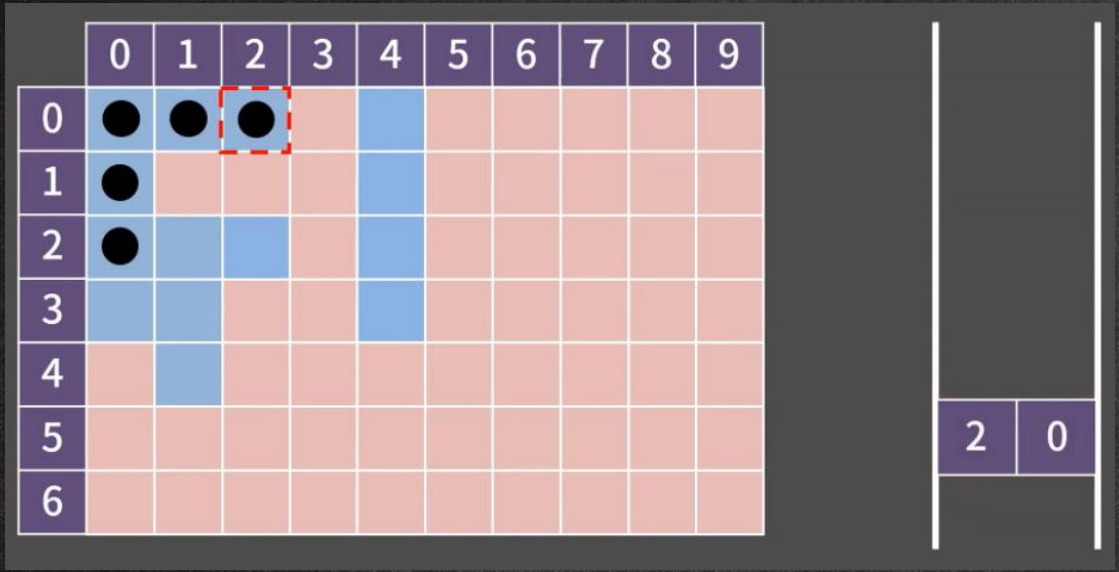
BFS 알고리즘 설명 그림으로!!!



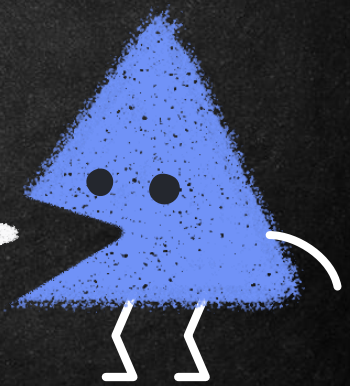
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



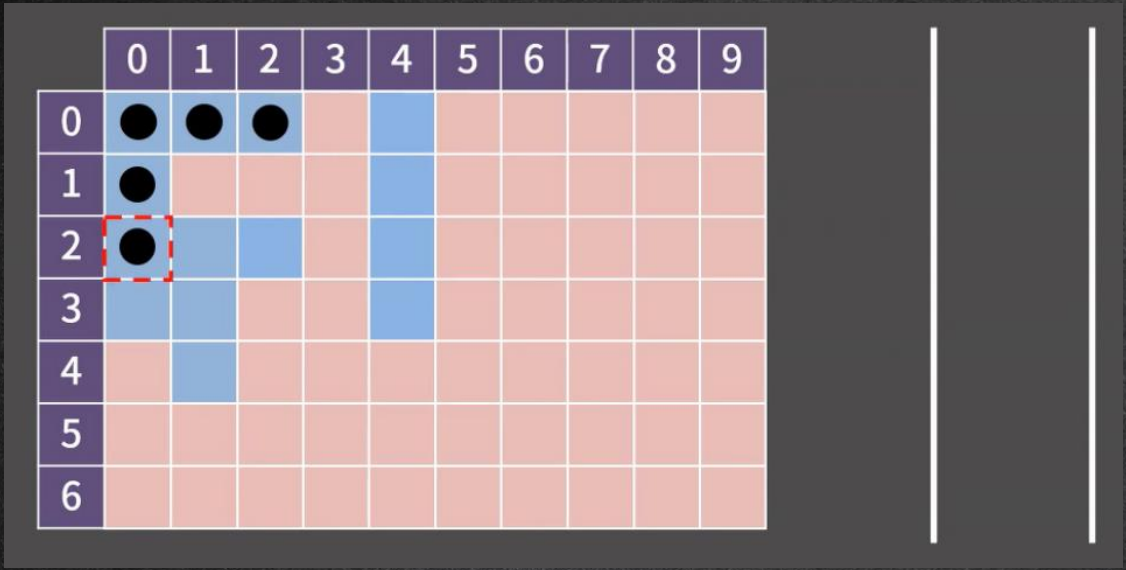
BFS 알고리즘 설명 그림으로!!!



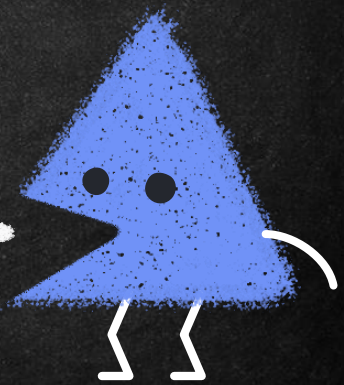
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



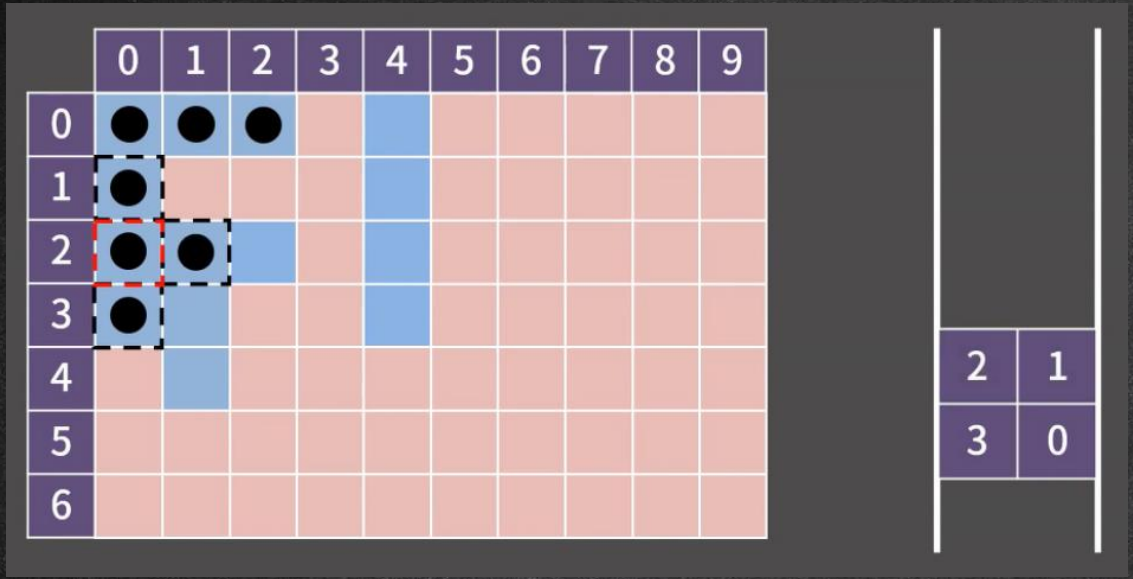
BFS 알고리즘 설명 그림으로!!!



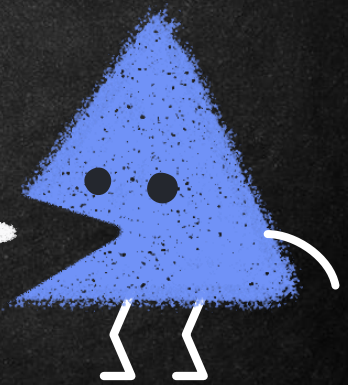
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



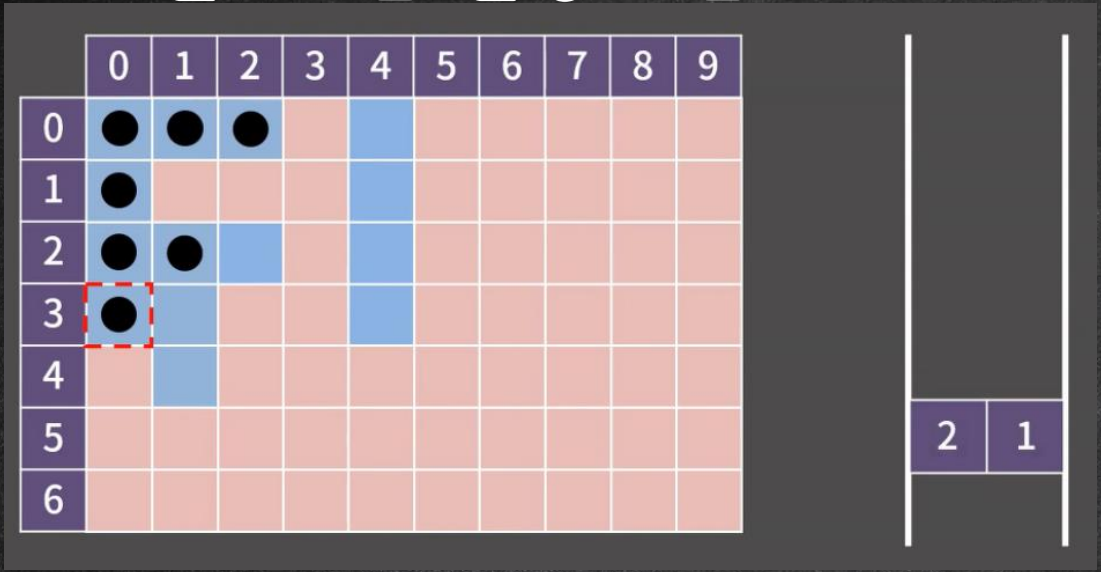
BFS 알고리즘 설명 그림으로!!!



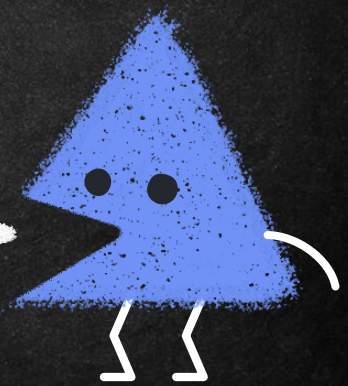
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



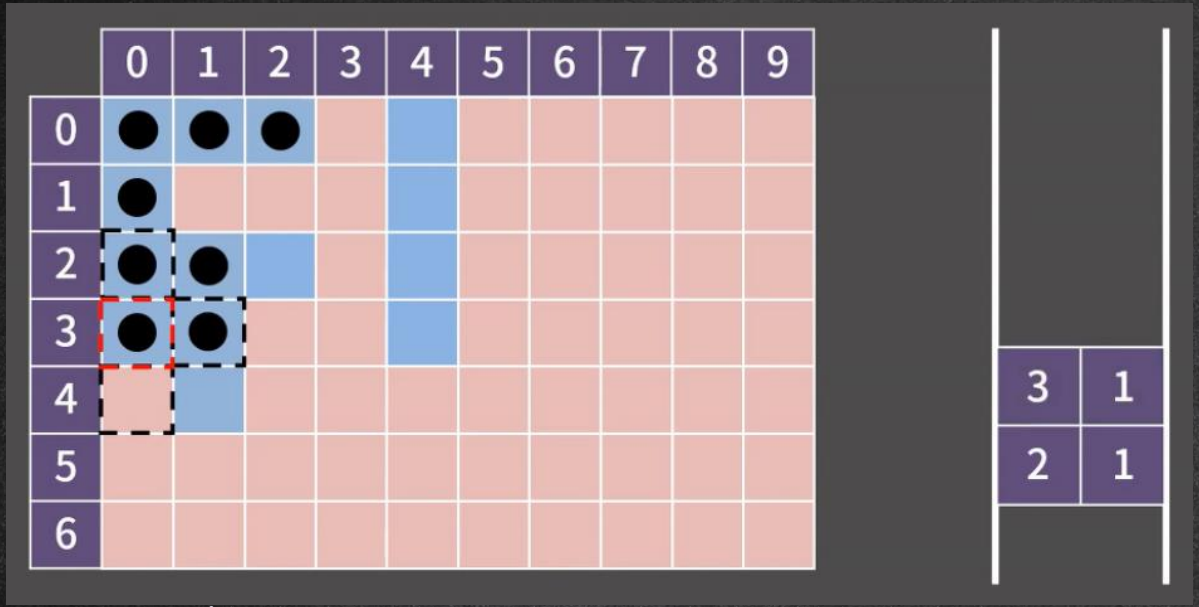
BFS 알고리즘 설명 그림으로!!!



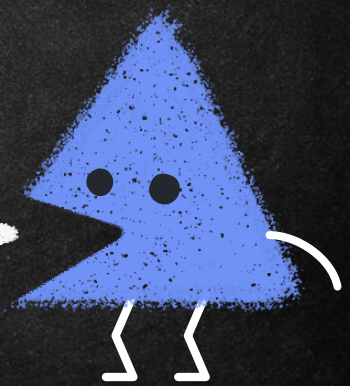
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



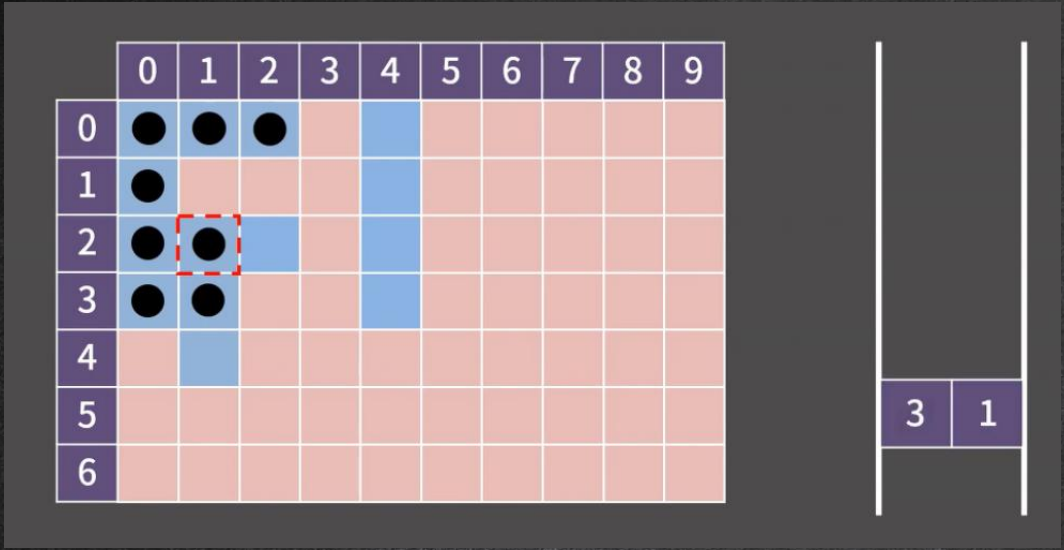
BFS 알고리즘 설명 그림으로!!!



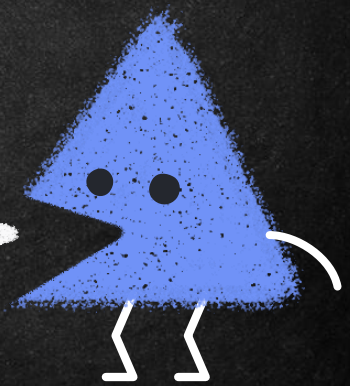
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



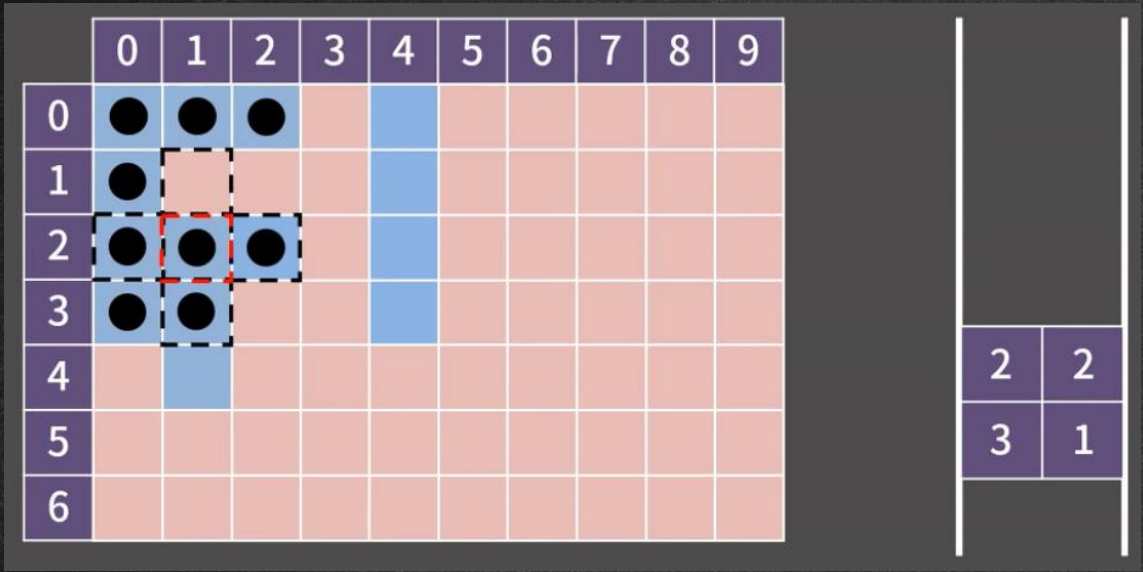
BFS 알고리즘 설명 그림으로!!!



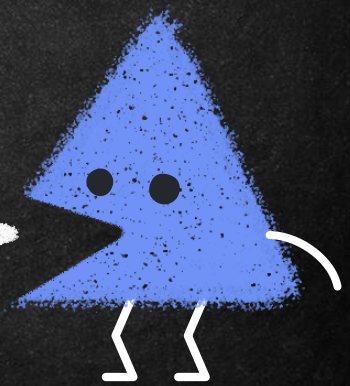
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



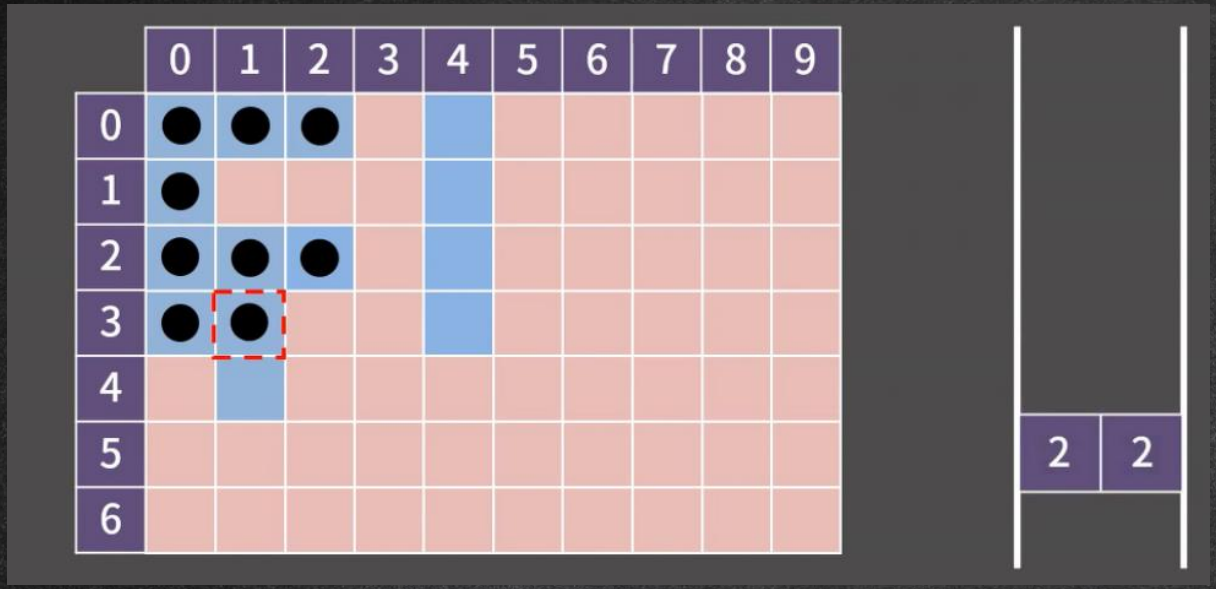
BFS 알고리즘 설명 그림으로!!!



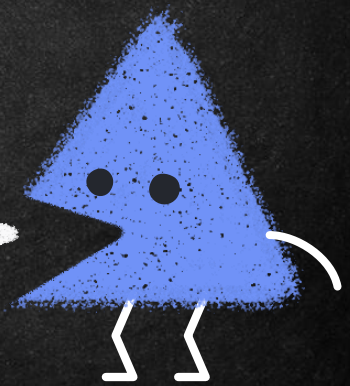
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



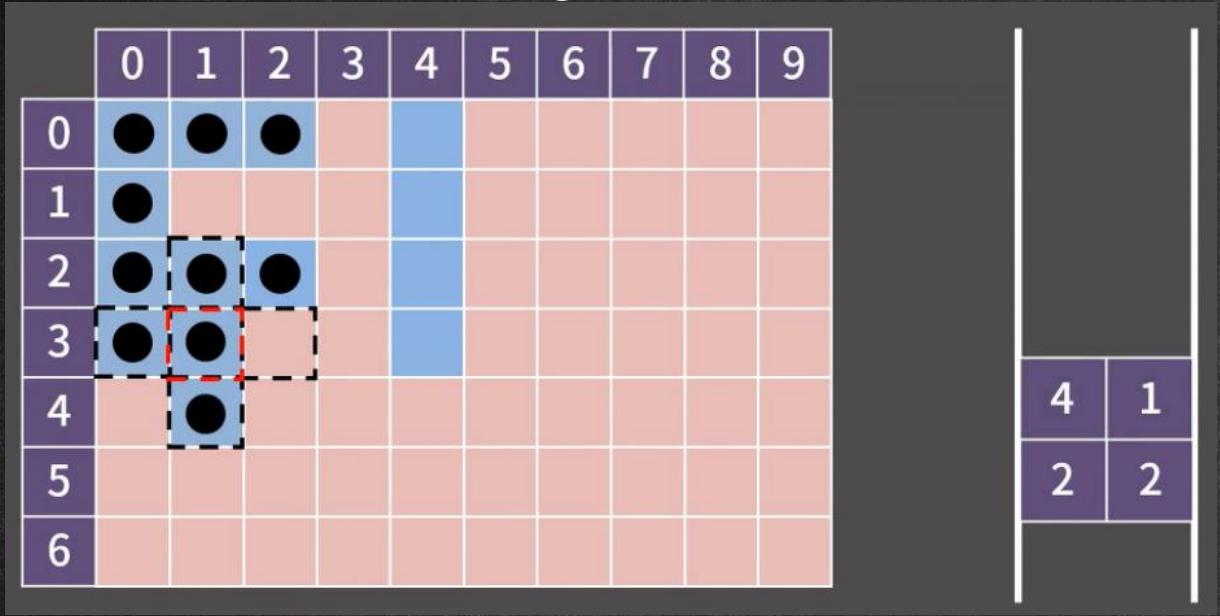
BFS 알고리즘 설명 그림으로!!!



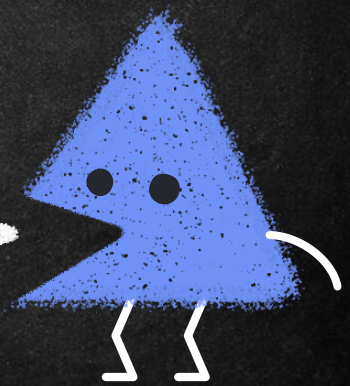
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



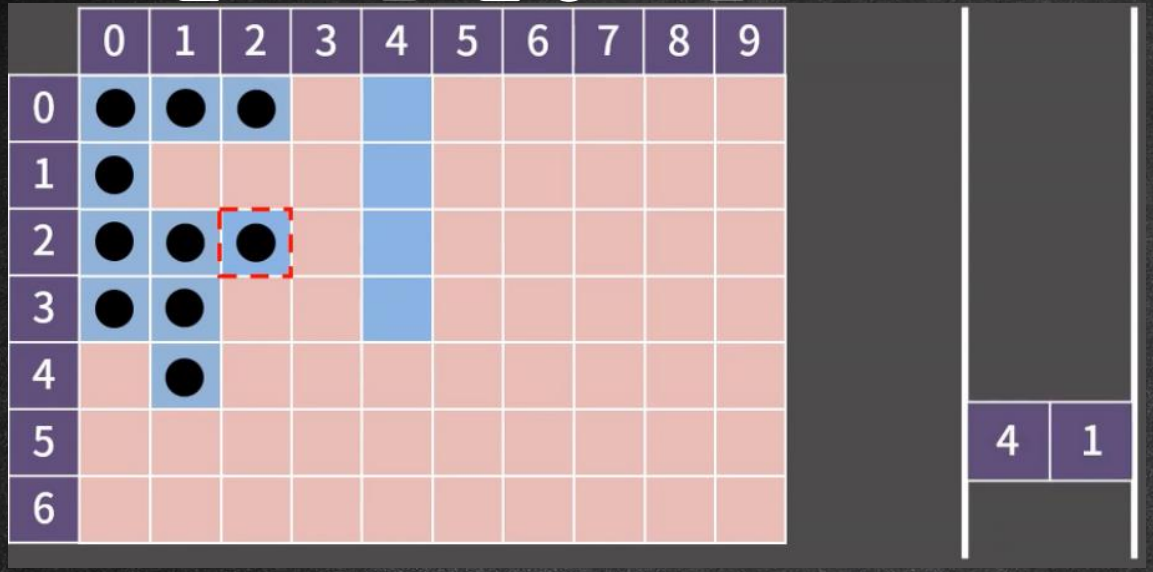
BFS 알고리즘 설명 그림으로!!!



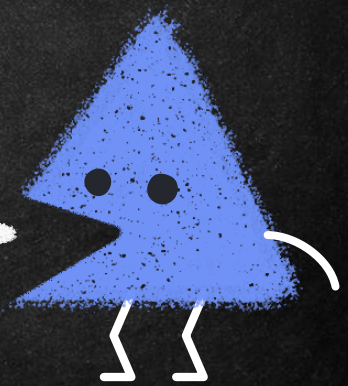
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



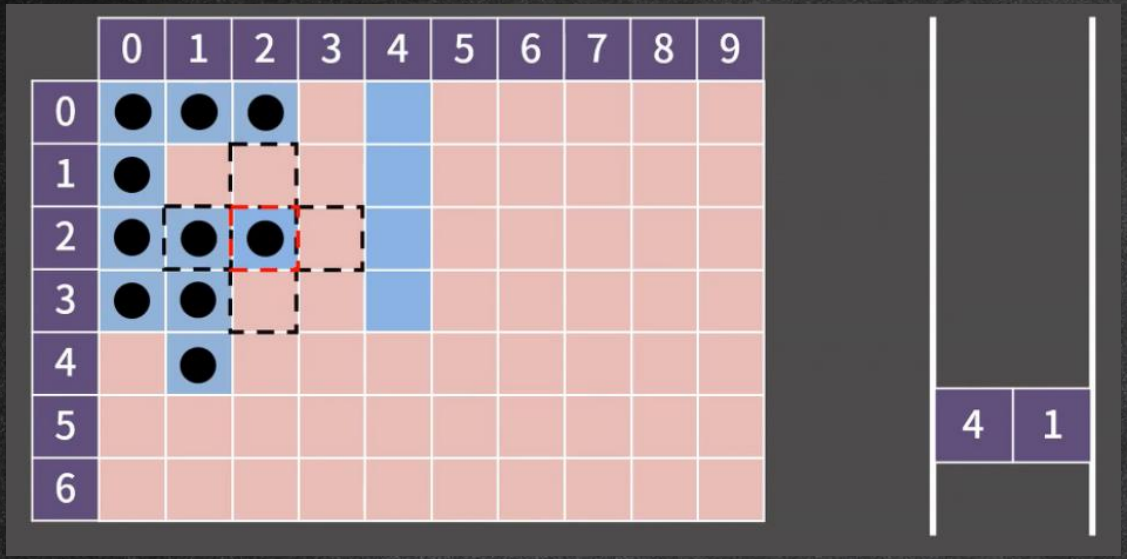
BFS 알고리즘 설명 그림으로!!!



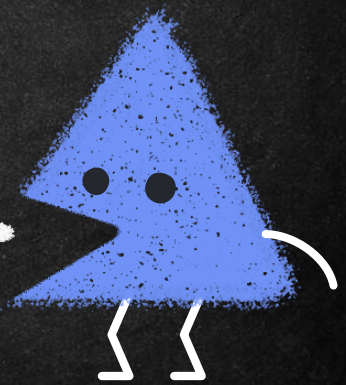
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



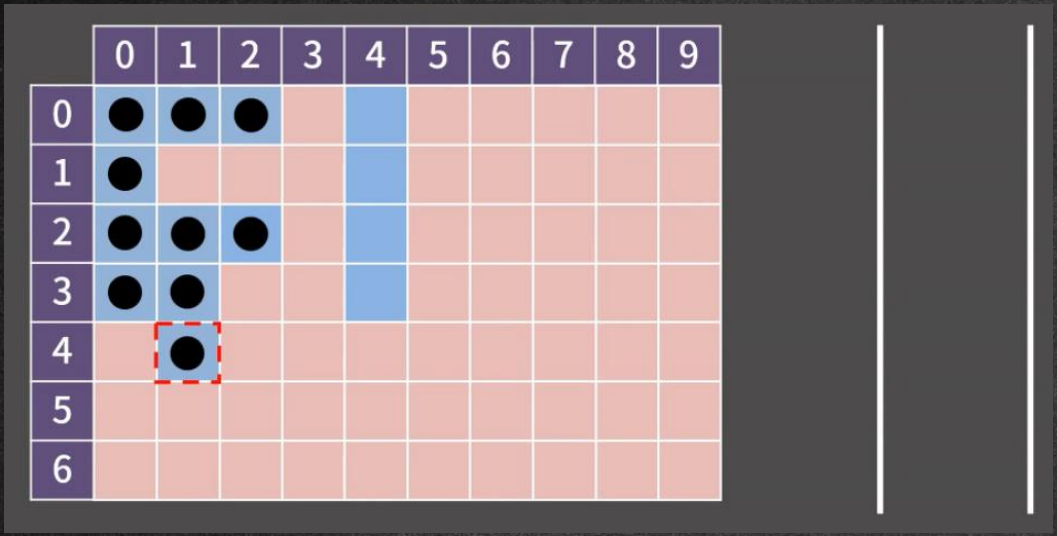
BFS 알고리즘 설명 그림으로!!!



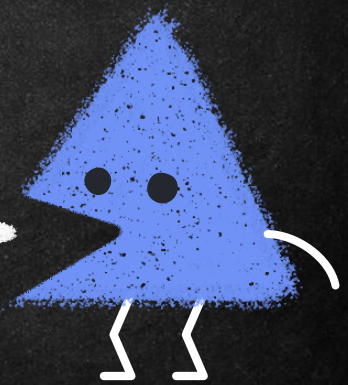
→ 스스로 확인해보세요! 같은 과정이 반복됩니다!!!



BFS 알고리즘 설명 그림으로!!!



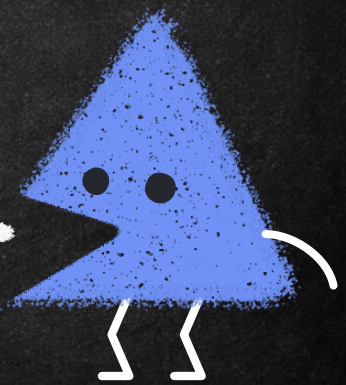
- → 스스로 확인해보세요! 같은 과정이 반복됩니다!!!
- → 이제 상하좌우를 살펴도 없네요!!!
- → 정리해봅시다!!!



BFS 알고리즘 설명

1. 시작하는 칸을 큐에 넣고 방문했다는 표시를 남김.
2. 큐에서 원소를 꺼내서 그 칸에 상하좌우로 인접한 칸에 대해서 3번을 진행함.
3. 해당 칸을 이전에 방문했다면 패스, 처음으로 방문했다면 방문했다는 표시 후에 큐에 삽입
4. 큐가 빌때까지 2번을 반복함.

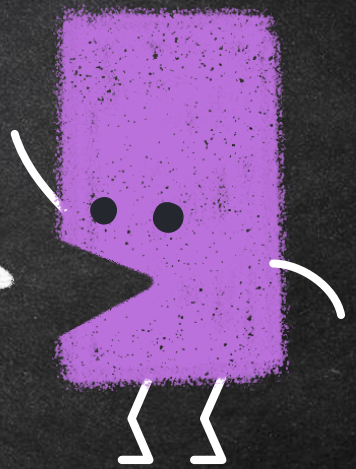
➔ 코드를 짜면서 생각을 해봅시다.



“

Pair 기억나시나요..?

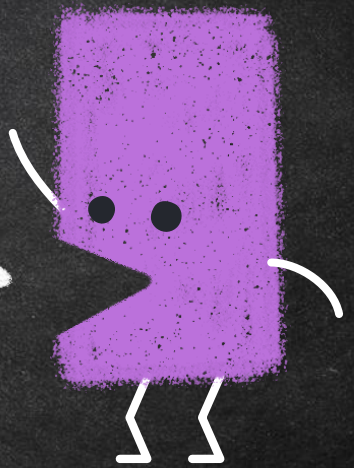
https://github.com/Dcom-KHU/2021_Algorithm_basic/blob/main/Dcom-KHU%20Week5/pair_practice.cpp



“

값의 접근은 각각 first, second를 부름으로써 가능하고 또 pair에는 미리 대소 관계가 설정되어 있어서 편합니다. 알아서 앞쪽의 값을 먼저 비교하고, 이후 뒤쪽의 값을 비교합니다.

BFS를 구현할 때 큐에 좌표를 넣어야 하는데, 이때 pair를 쓸 것입니다.



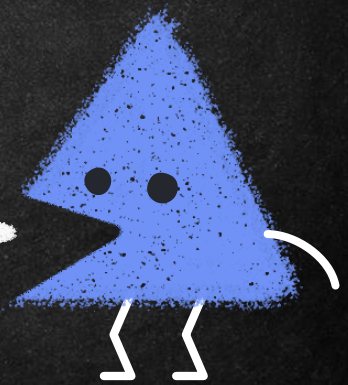
BFS 알고리즘 설명 코드로!!!

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

using namespace std;
#define X first
#define Y second // pair에서 first, second를 줄여서 쓰기 위해서 사용
int board[502][502] =
{ {1,1,1,0,1,0,0,0,0,0},
 {1,0,0,0,1,0,0,0,0,0},
 {1,1,1,0,1,0,0,0,0,0},
 {1,1,0,0,1,0,0,0,0,0},
 {0,1,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0} }; // 1이 파란 칸, 0이 빨간 칸에 대응
bool vis[502][502]; // 해당 칸을 방문했는지 여부를 저장
int n = 7, m = 10; // n = 행의 수, m = 열의 수
int dx[4] = { 1,0,-1,0 };
int dy[4] = { 0,1,0,-1 }; // 상하좌우 네 방향을 의미
```

→ 상하좌우를 왜 저렇게 하나요??

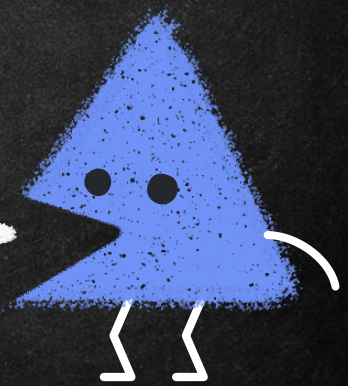
→ X좌표 기준 +1 -1, Y좌표 기준 +1 -1을 해준다고 생각!!!



BFS 알고리즘 설명 코드로!!!

```
int main(void) {
    queue<pair<int, int> > Q;
    vis[0][0] = 1; // (0, 0)을 방문했다고 명시
    Q.push({ 0,0 }); // 큐에 시작점인 (0, 0)을 삽입.
    while (!Q.empty()) {
        pair<int, int> cur = Q.front(); Q.pop();
        cout << '(' << cur.X << ", " << cur.Y << ") -> ";
        for (int dir = 0; dir < 4; dir++) { // 상하좌우 칸을 살펴볼 것이다.
            int nx = cur.X + dx[dir];
            int ny = cur.Y + dy[dir]; // nx, ny에 dir에서 정한 방향의 인접한 칸의 좌표가 들어감
            if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
            // 범위 밖일 경우 넘어감
            if (vis[nx][ny] || board[nx][ny] != 1) continue;
            // 이미 방문한 칸이거나 파란 칸이 아닐 경우
            vis[nx][ny] = 1;
            // (nx, ny)를 방문했다고 명시
            Q.push({ nx,ny });
        }
    }
}
```

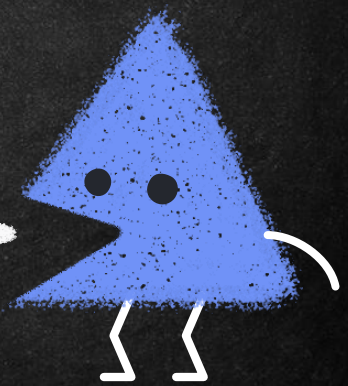
- → 실행결과를 예상해보세요!!!
- → 앞에 board에서 1 적힌 것 기준으로 BFS 진행함.



BFS 알고리즘 설명 코드로!!!

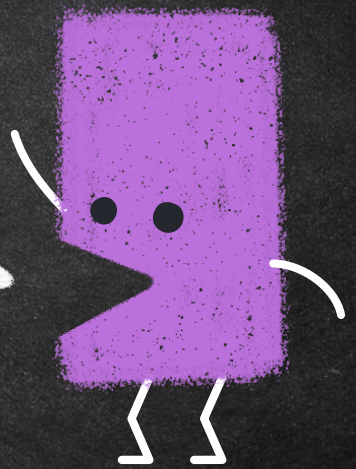
```
(0, 0) ->
(1, 0) ->
(0, 1) ->
(2, 0) ->
(0, 2) ->
(3, 0) ->
(2, 1) ->
(3, 1) ->
(2, 2) ->
(4, 1) ->
```

제대로 너비 우선으로 탐색이 진행되었음을 확인할 수 있습니다!!!



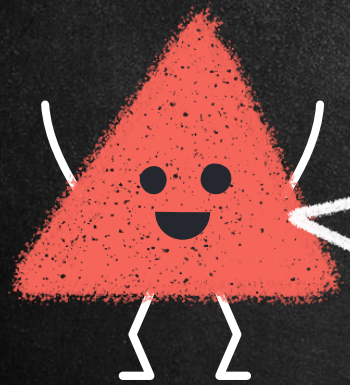
BFS Source Code

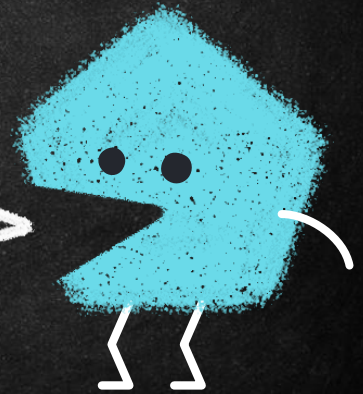
https://github.com/Dcom-KHU/2021_Algorithm_basic/blob/main/Dcom-KHU%20Week5/bfs.cpp



3.

DFS!!!



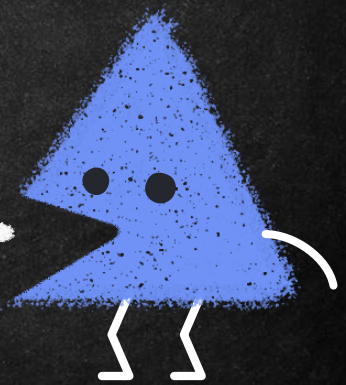


DFS 알고리즘 설명

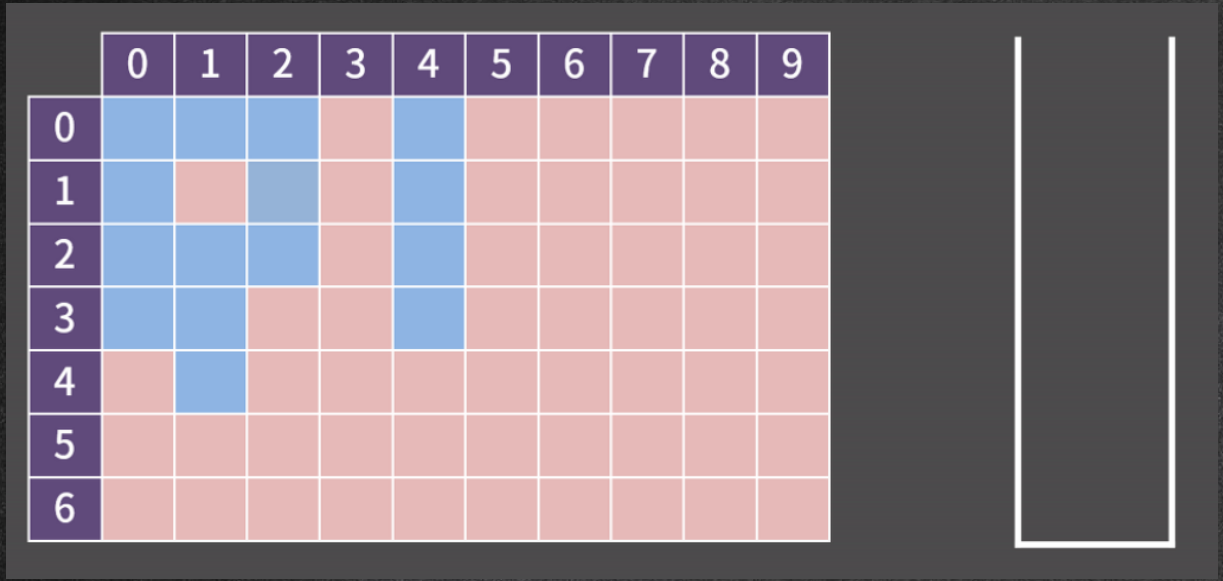
DFS(Depth First Search)

- 다차원 배열에서 각 칸을 방문할 때 깊이를 우선으로 방문하는 알고리즘
- *깊이를 우선으로 방문한다...?!!???? 뭔 소리죠??*

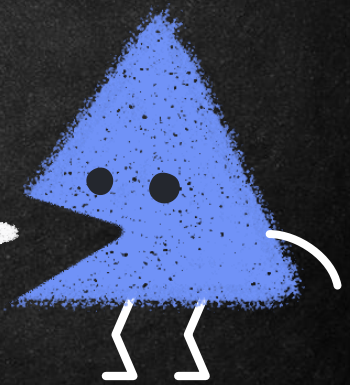
- *그림을 통해서 이해해봅시다!!!*



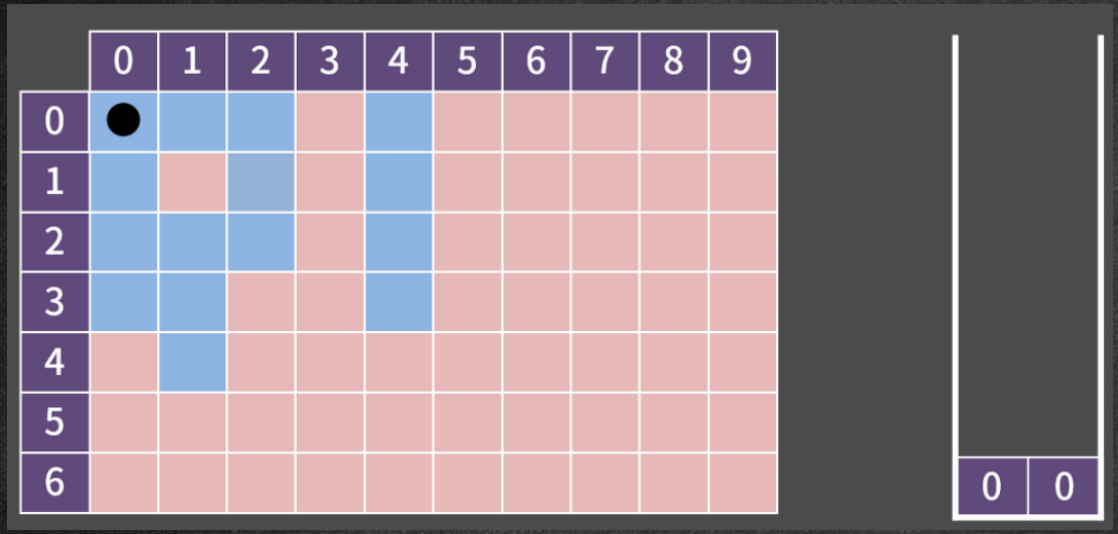
DFS 알고리즘 설명 그림으로!!!



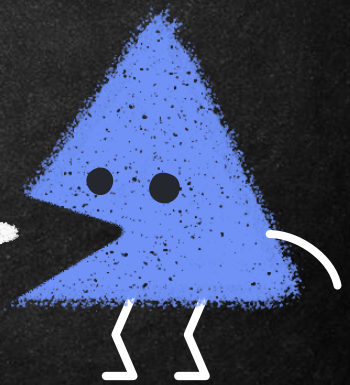
- 우리의 목표는 (0,0)과 상하좌우로 이어진 모든 파란색 칸을 확인하는 것입니다. 이 문제를 DFS로 어떻게 해결하는지 보겠습니다.



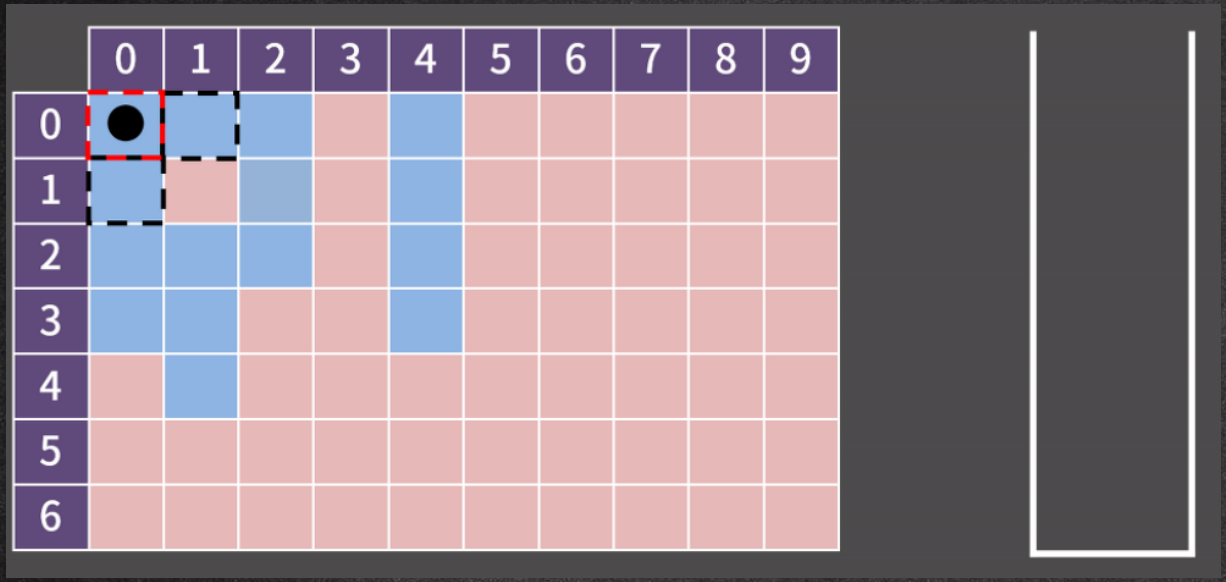
DFS 알고리즘 설명 그림으로!!!



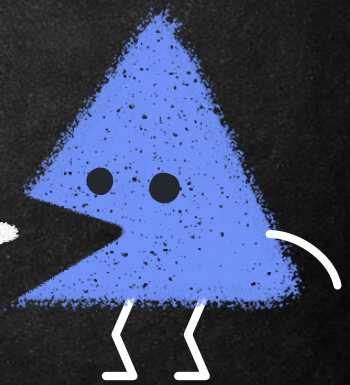
→ 우선 (0,0)에 방문했다는 표시를 한 후 해당 칸을 스택에 넣습니다. 이게 초기세팅이고, 이후에는 스택이 빌때까지 계속 스택의 top을 빼고 상하좌우를 살피면서 반복합니다.



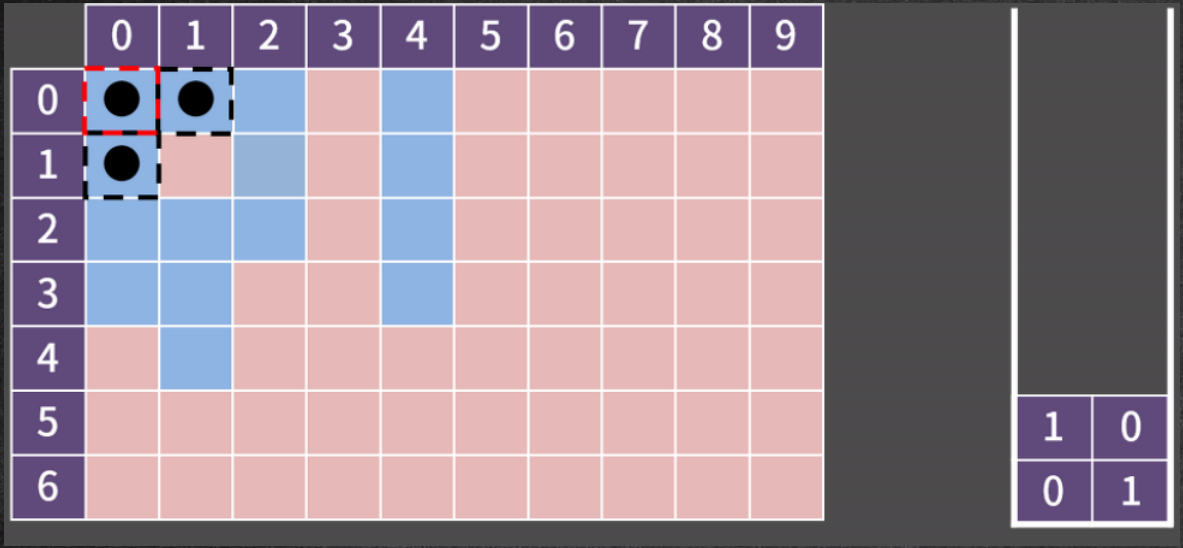
DFS 알고리즘 설명 그림으로!!!



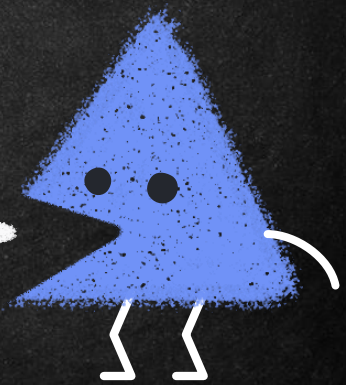
- 지금 스택의 top은 (0,0)이고 pop을 합니다.
- 그리고 (0,0)의 상하좌우를 보는데 이 중에서 우리는 파란색 칸이면서 아직 방문하지 않은 칸을 찾습니다.



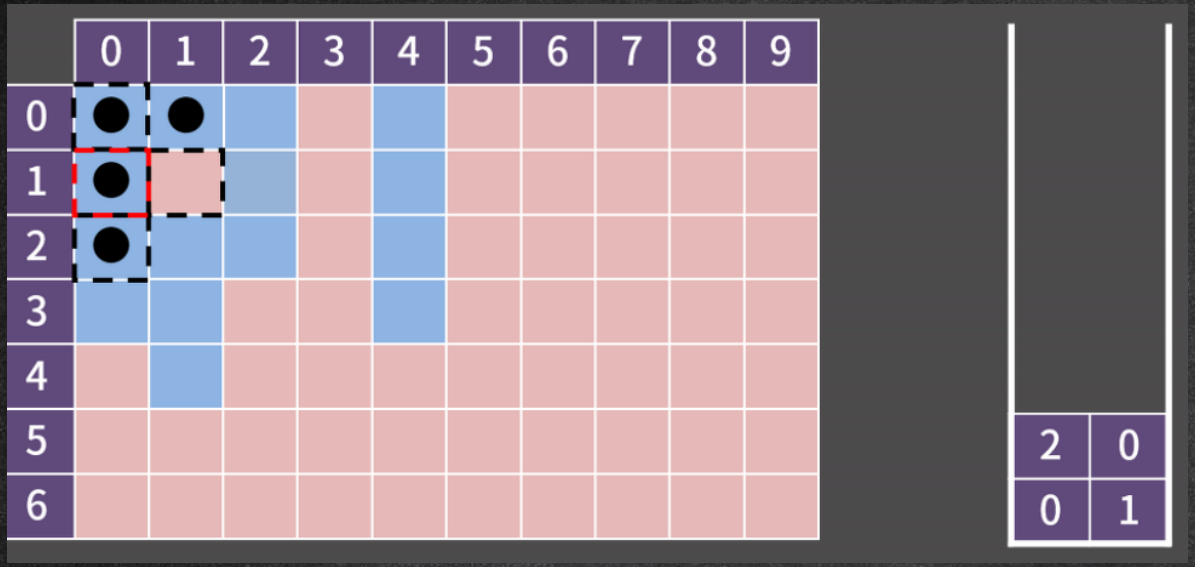
DFS 알고리즘 설명 그림으로!!!



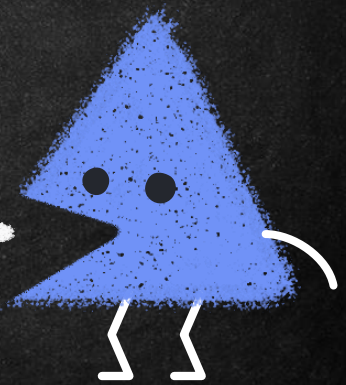
→ 보면 (0,1)과 (1,0)이 모두 파란 칸이면서 아직 방문하지 않은 칸이니 방문했다는 표시를 남기고 스택에 넣습니다.



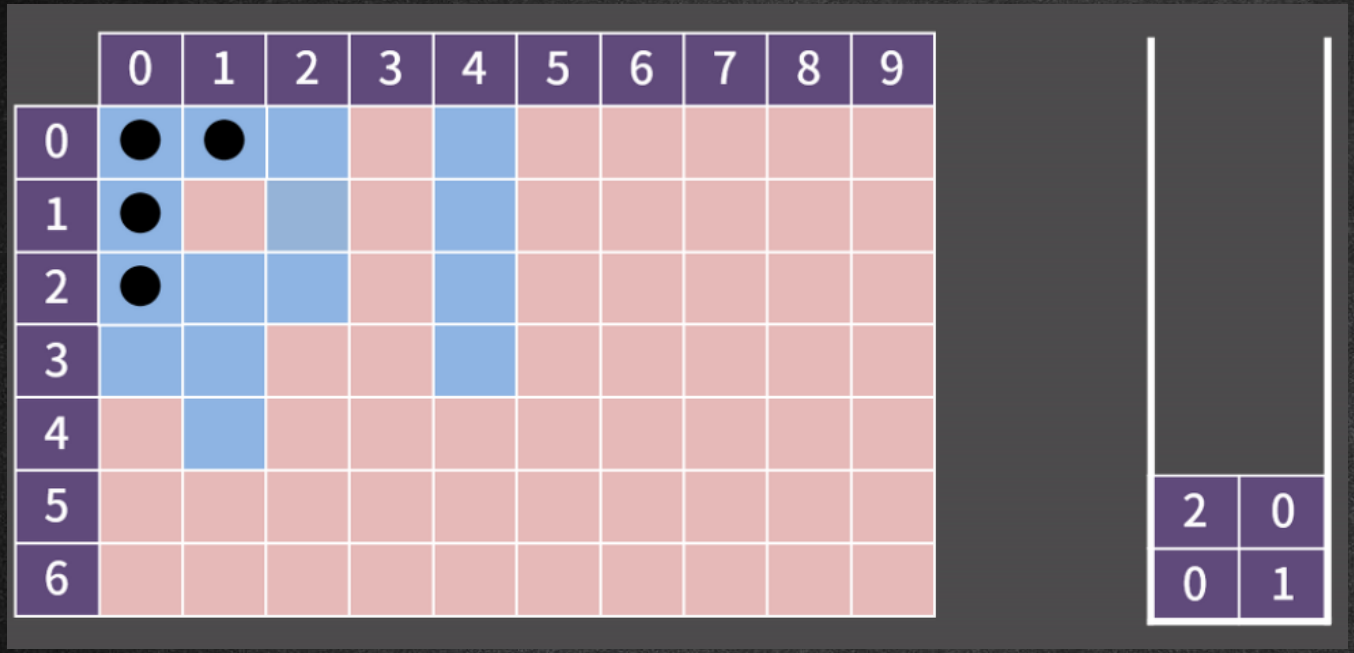
DFS 알고리즘 설명 그림으로!!!



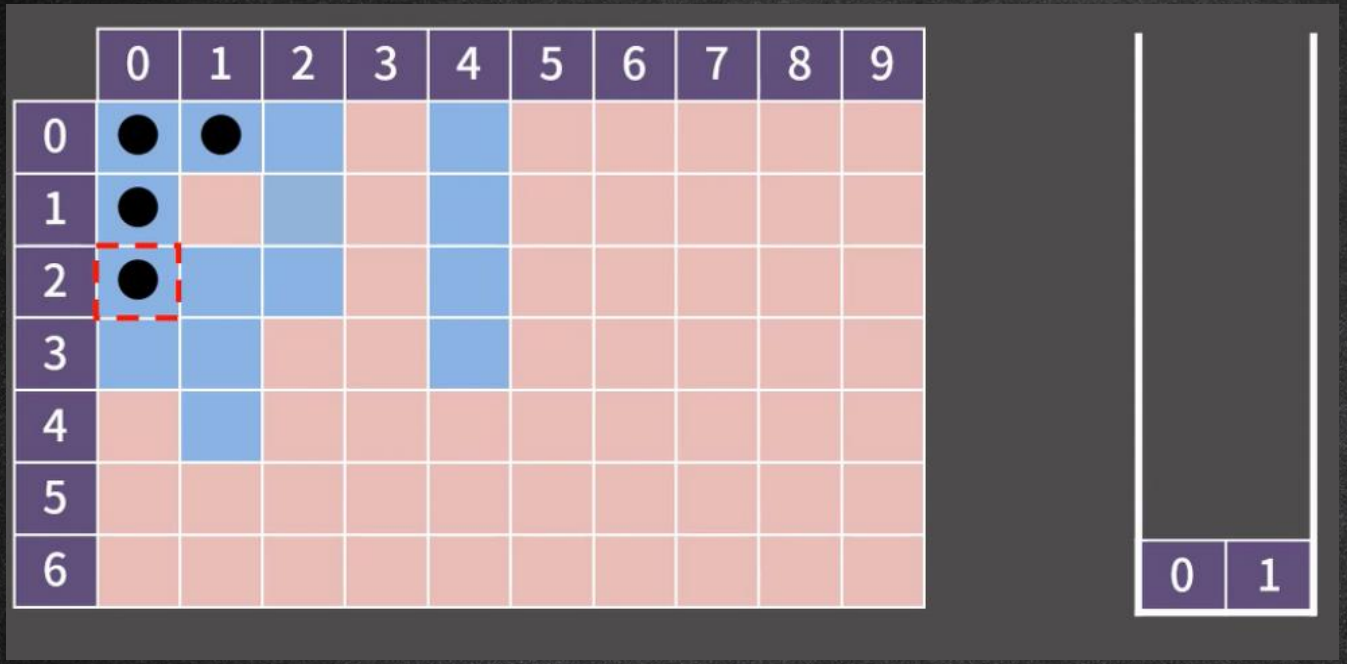
- 지금 스택의 top은 (1,0)이고 pop을 합니다.
- 상하좌우칸을 살펴보면 (2,0)을 찾을 수 있고 방문처리한 후에 스택에 넣습니다! 이제 반복해볼게요~



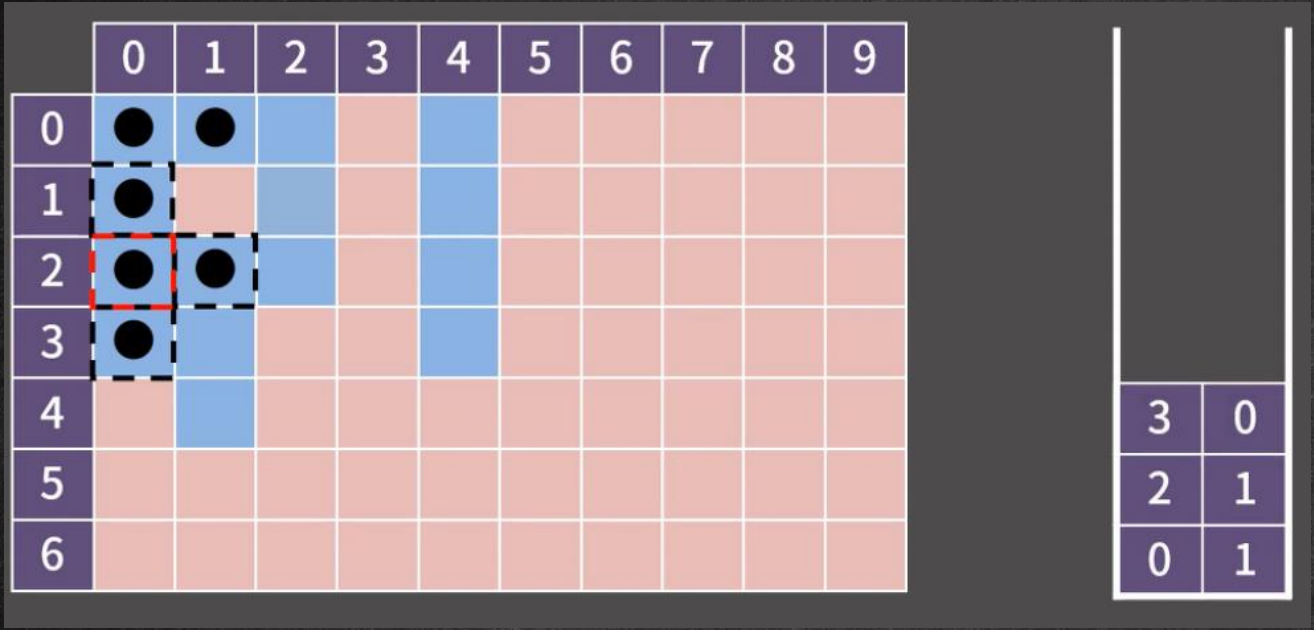
DFS 알고리즘 설명 그림으로!!!

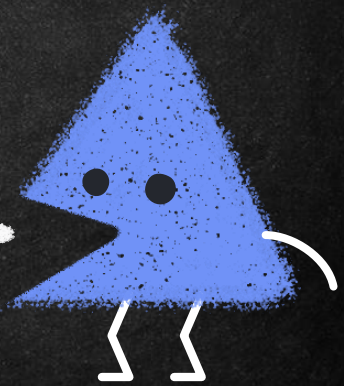


DFS 알고리즘 설명 그림으로!!!

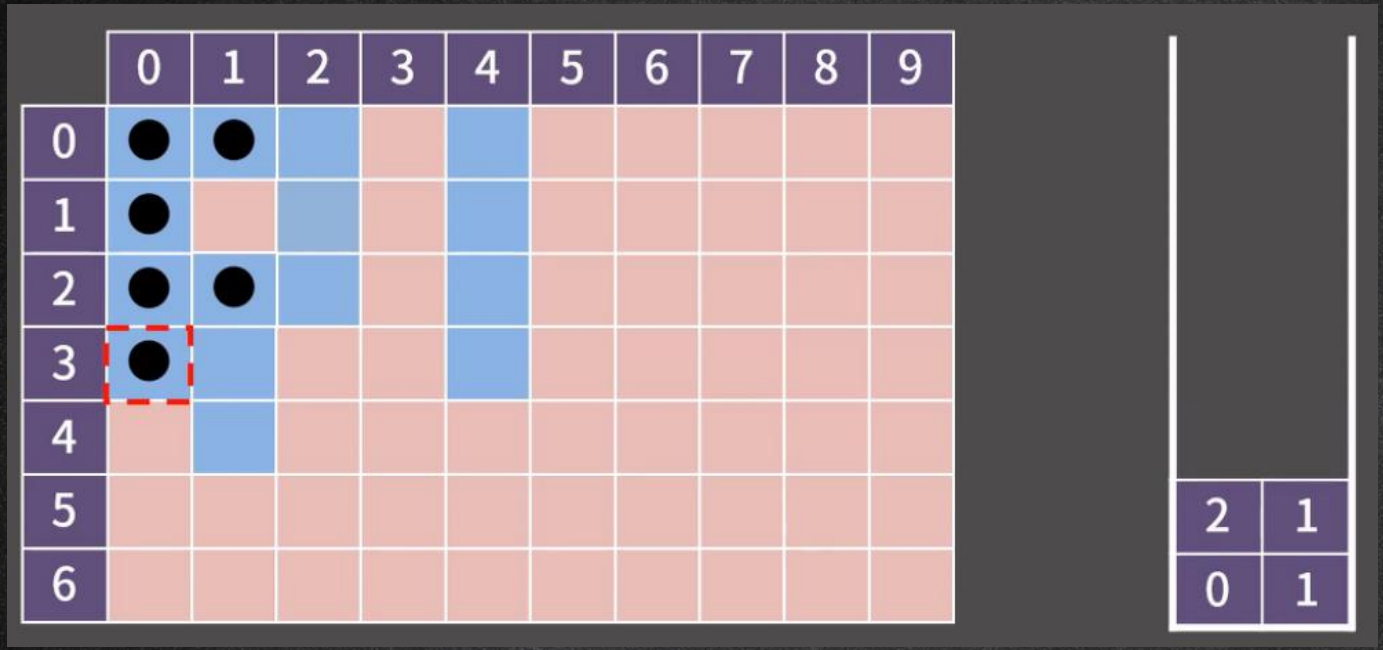


DFS 알고리즘 설명 그림으로!!!

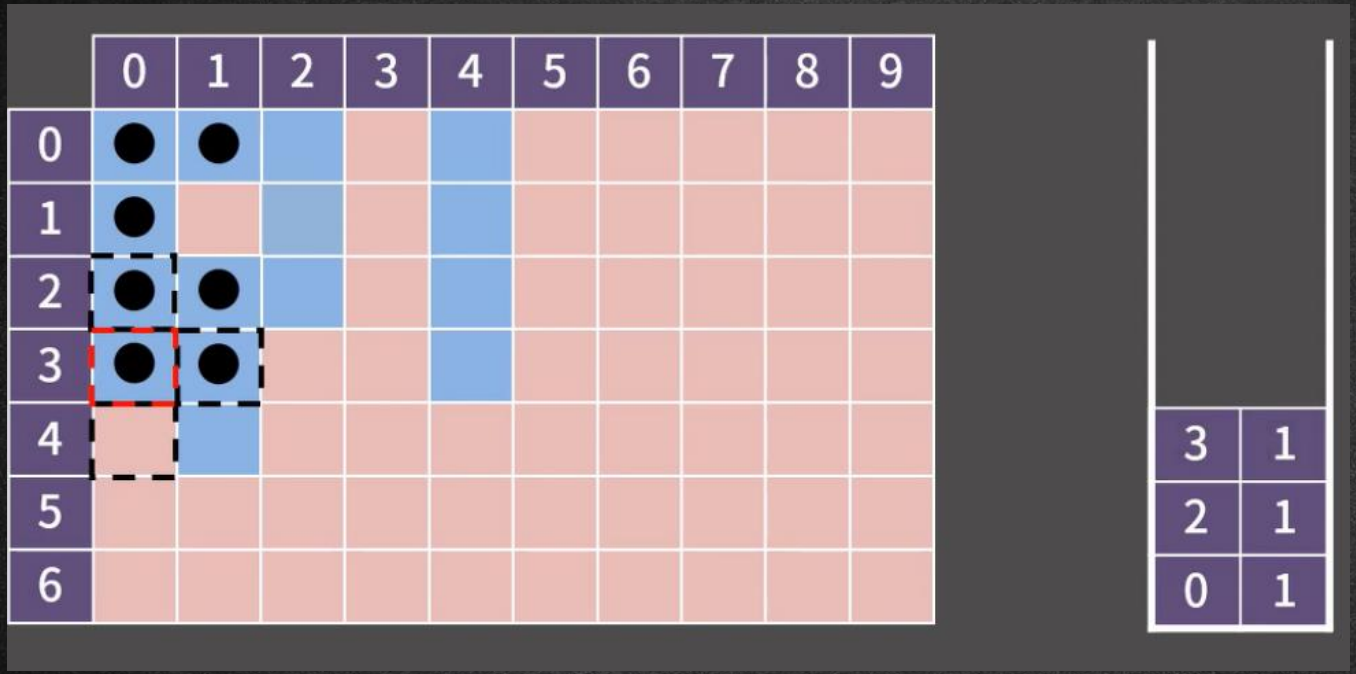


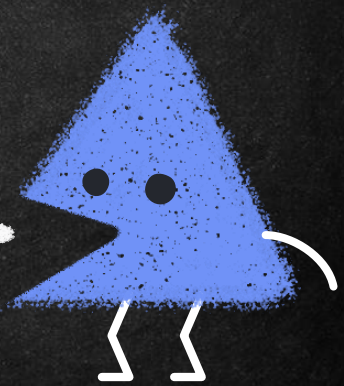


DFS 알고리즘 설명 그림으로!!!



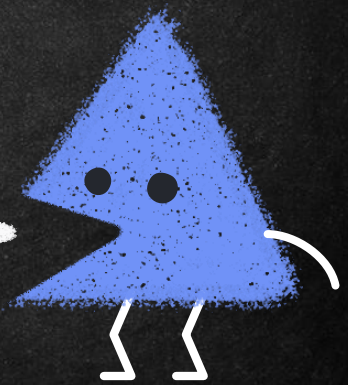
DFS 알고리즘 설명 그림으로!!!



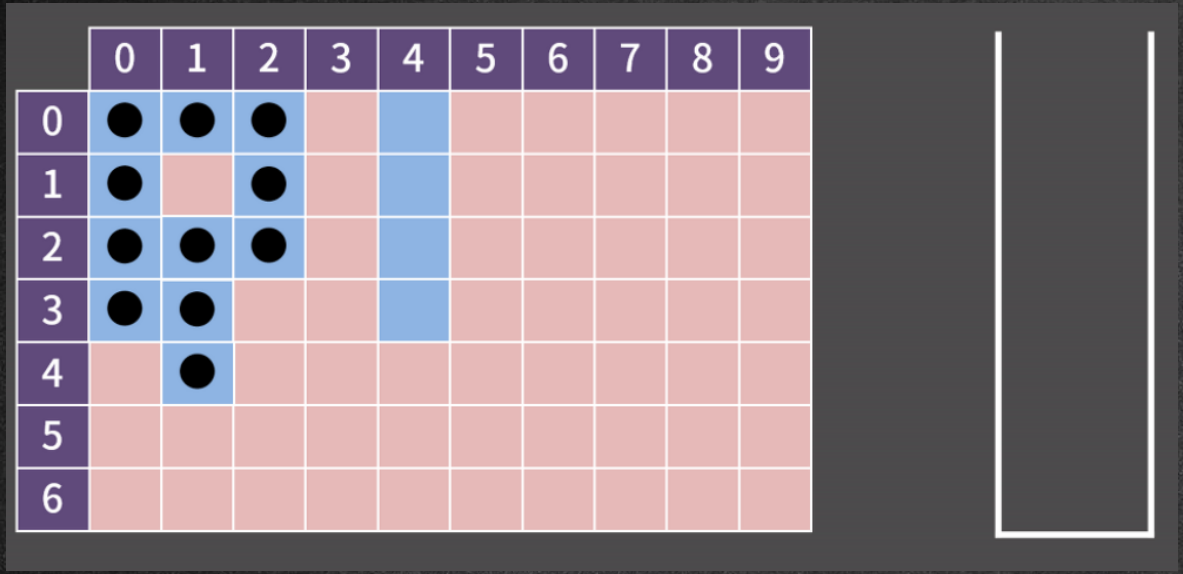


DFS 알고리즘 설명 그림으로!!!

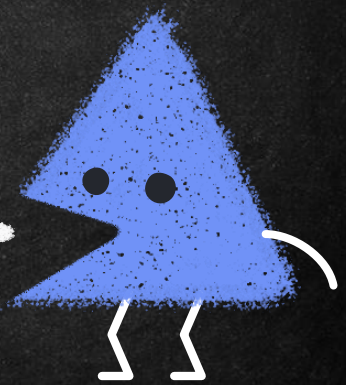
→ 계에에에에에에속 반복합니다!!!!



DFS 알고리즘 설명 그림으로!!!



- 이렇게 스택이 비는 순간 종료됩니다.
- BFS는 큐를 사용해서, DFS는 스택을 사용해서 생각해보았습니다!!!

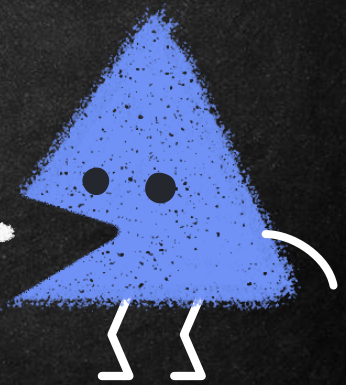


DFS 알고리즘 설명 코드로!!!

```cpp
#include <iostream>
#include <stack>
using namespace std;
#define X first
#define Y second // pair에서 first, second를 줄여서 쓰기 위해서 사용
int board[502][502] =
{ {1,1,1,0,1,0,0,0,0,0},
 {1,0,0,0,1,0,0,0,0,0},
 {1,1,1,0,1,0,0,0,0,0},
 {1,1,0,0,1,0,0,0,0,0},
 {0,1,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0} }; // 1이 파란 칸, 0이 빨간 칸에 대응
bool vis[502][502]; // 해당 칸을 방문했는지 여부를 저장
int n = 7, m = 10; // n = 행의 수, m = 열의 수
int dx[4] = { 1,0,-1,0 };
int dy[4] = { 0,1,0,-1 }; // 상하좌우 네 방향을 의미
```

→ 상하좌우를 왜 저렇게 하나요??

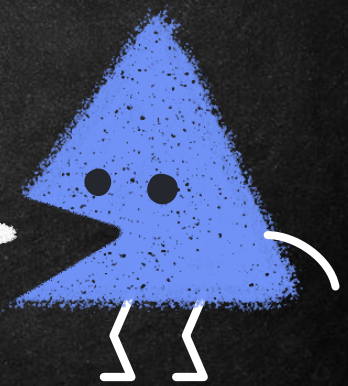
→ X좌표 기준 +1 -1, Y좌표 기준 +1 -1을 해준다고 생각!!!



DFS 알고리즘 설명 코드로!!!

```
int main(void) {
    stack<pair<int, int> > S;
    vis[0][0] = 1; // (0, 0)을 방문했다고 명시
    S.push({ 0,0 }); // 스택에 시작점인 (0, 0)을 삽입.
    while (!S.empty()) {
        pair<int, int> cur = S.top(); S.pop();
        cout << '(' << cur.X << ", " << cur.Y << ") -> ";
        for (int dir = 0; dir < 4; dir++) { // 상하좌우 칸을 살펴볼 것이다.
            int nx = cur.X + dx[dir];
            int ny = cur.Y + dy[dir]; // nx, ny에 dir에서 정한 방향의 인접한 칸의 좌표가 들어감
            if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
            // 범위 밖일 경우 넘어감
            if (vis[nx][ny] || board[nx][ny] != 1) continue;
            // 이미 방문한 칸이거나 파란 칸이 아닐 경우
            vis[nx][ny] = 1; // (nx, ny)를 방문했다고 명시
            S.push({ nx,ny });
        }
    }
}
```

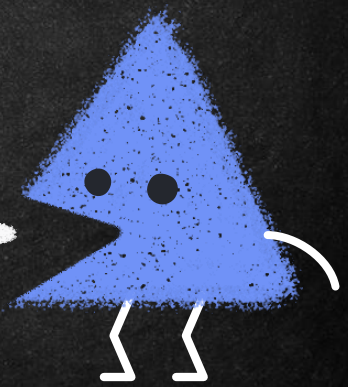
- → 실행결과를 예상해보세요!!!
- → 앞에 board에서 1 적힌 것 기준으로 DFS 진행함.



DFS 알고리즘 설명 코드로!!!

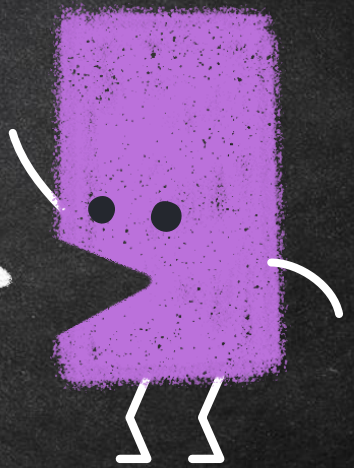
```
(0, 0) ->
(0, 1) ->
(0, 2) ->
(1, 0) ->
(2, 0) ->
(2, 1) ->
(2, 2) ->
(3, 1) ->
(4, 1) ->
(3, 0) ->
```

제대로 깊이 우선으로 탐색이 진행되었음을 확인할 수 있습니다!!!



DFS Source Code

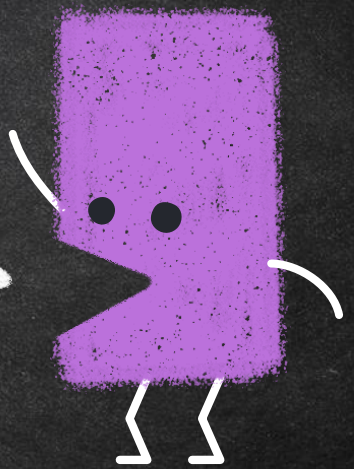
https://github.com/Dcom-KHU/2021_Algorithm_basic/blob/main/Dcom-KHU%20Week5/dfs.cpp



“

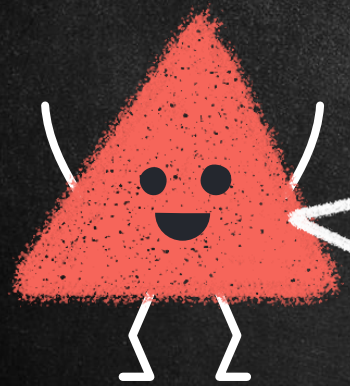
DFS는 재귀로도 구현이 가능합니다!!!

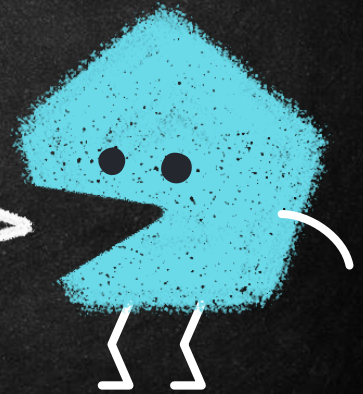
https://velog.io/@choiiis/%EC%9E%AC%EA%B7%80%ED%95%A8%EC%88%98-stack%EC%9D%84-%EC%9D%B4%EC%9A%A9%ED%95%9C-DFS-%EA%B8%B0%EB%B3%B8-%EA%B5%AC%ED%98%84



4.

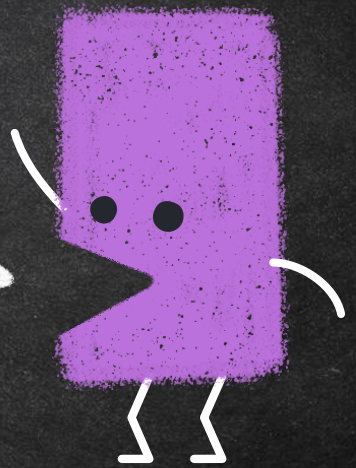
Workout!!!





“

백준에 있는 문제 2개를 함께
풀어봅니다!!!



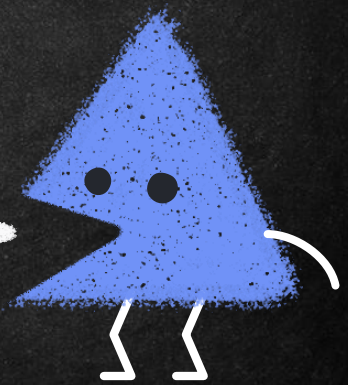
백준(1926번-그림 문제)

https://www.acmicpc.net/problem/1926

꼭 문제와 입출력을 확인하세요!!!

문제에서 난관은 다음과 같습니다.

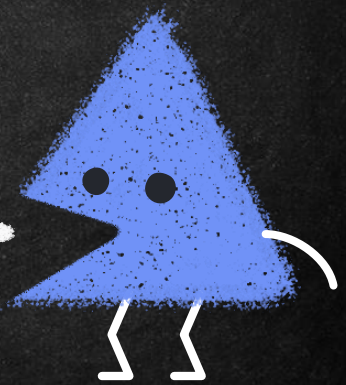
➔ 상하좌우로 연결된 그림의 크기를 알아내기
➔ 도화지에 있는 모든 그림을 찾아내기



백준(1926번-그림 문제)

우리는 지금 하나의 시작점에서 BFS를 도는 방법만
익혔으나,이 문제에서는 도화지에서 모든 그림을 찾아야하고,
크기도 알아내야 합니다.
꽤 어렵지 않나 싶지만, 조금만 생각을 하면 됩니다.

크기의 경우에는 POP을 몇 번하는지를 세주면 됩니다.
(생각해보세요!!!)
모든 그림을 찾는 경우에는 이중 for문을 사용하면 됩니다.
(???)

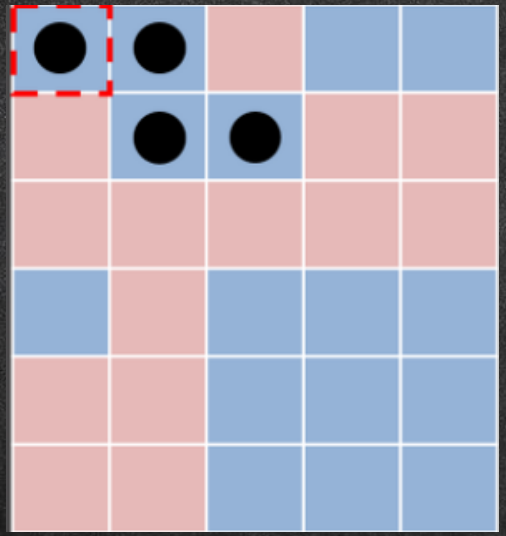


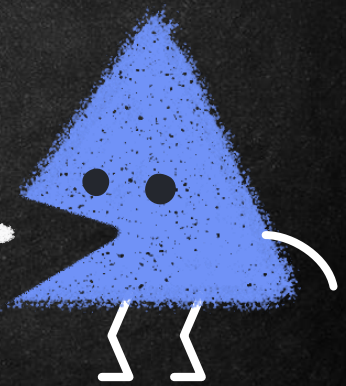
백준(1926번-그림 문제)

모든 그림을 찾는 경우에는 이중 for문을 사용하면 됩니다(??)

아래 그림을 생각해보면 1번 BFS를 마친 상태입니다.

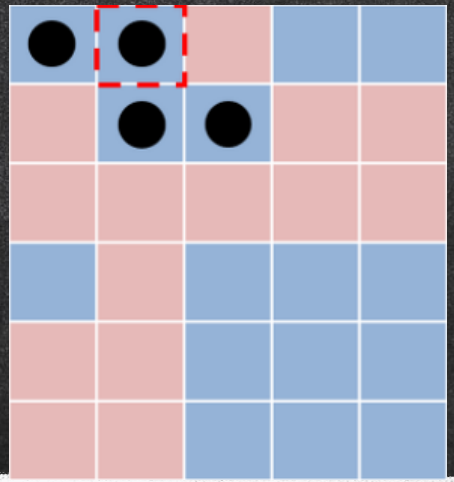
BFS를 마쳤으니 방문처리가 되었겠죠!!!

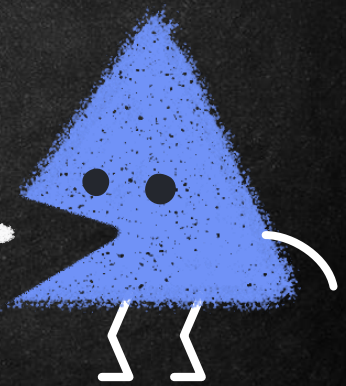




백준(1926번-그림 문제)

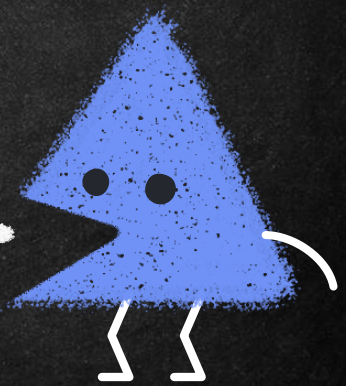
(0,0)에 이어 오른쪽으로 한 칸 이동해서 BFS가 가능한지 살펴봅니다. 이미 전 슬라이드에서 BFS를 완료했으니 방문처리가 되었으므로 시작점이 될 수 없습니다!!!
이중 for문은 결국 시작점을 찾기 위해 사용되는 것입니다.





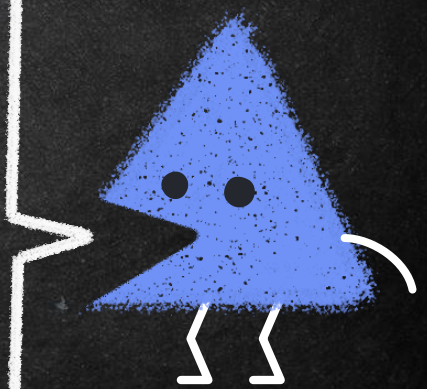
백준(1926번-그림 문제) CODE

```cpp
#include <iostream>
#include <queue>
using namespace std;
#define X first
#define Y second // pair에서 first, second를 줄여서 쓰기 위해서 사용
int board[502][502]; // 1이 파란 칸, 0이 빨간 칸에 대응
bool vis[502][502]; // 해당 칸을 방문했는지 여부를 저장
int n, m;
int dx[4] = { 1,0,-1,0 };
int dy[4] = { 0,1,0,-1 }; // 상하좌우 네 방향을 의미
```



백준(1926번-그림 문제) CODE

```cpp
int main(void) {
    cin >> n >> m;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> board[i][j];
    int mx = 0; // 그림의 최댓값
    int num = 0; // 그림의 수
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) { // (i, j)를 시작점으로 하고 싶은 상황
            if (board[i][j] == 0 or vis[i][j]) continue;
            // 해당 칸이 색칠이 안된 부분(0)이거나 이미 (i, j)를 방문했을 경우 넘어감
            // (i,j)는 새로운 그림에 속해있는 시작점
            num++; // 그림의 수 1 증가
            queue<pair<int, int> > Q;
            vis[i][j] = 1; // (i,j)로 BFS를 시작하기 위한 준비
            Q.push({ i,j });
            int area = 0; // 그림의 넓이
            while (!Q.empty()) {
                area++; // 큐에 들어있는 원소를 하나 뺄 때 마다 넓이를 1 증가시킴
                auto cur = Q.front(); Q.pop();
                for (int dir = 0; dir < 4; dir++) { // 상하좌우 칸을 살펴볼 것이다.
                    int nx = cur.X + dx[dir];
                    int ny = cur.Y + dy[dir];
                    // nx, ny에 dir에서 정한 방향의 인접한 칸의 좌표가 들어감
                    if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
                    // 범위 밖일 경우 넘어감
                    if (vis[nx][ny] || board[nx][ny] != 1) continue;
                    // 이미 방문한 칸이거나 파란 칸이 아닐 경우
                    vis[nx][ny] = 1; // (nx, ny)를 방문했다고 명시
                    Q.push({ nx,ny });
                }
            }
            // (i, j)를 시작점으로 하는 BFS가 종료됨
            mx = max(mx, area);
            // area가 mx보다 클 경우 mx에 area를 대입. max는 STL에 정의된 함수
        }
    }
    cout << num << '₩n' << mx;
}
```

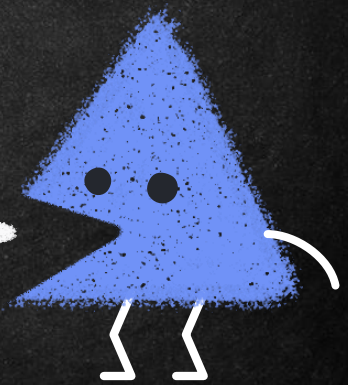


백준(2178번-미로 탐색)

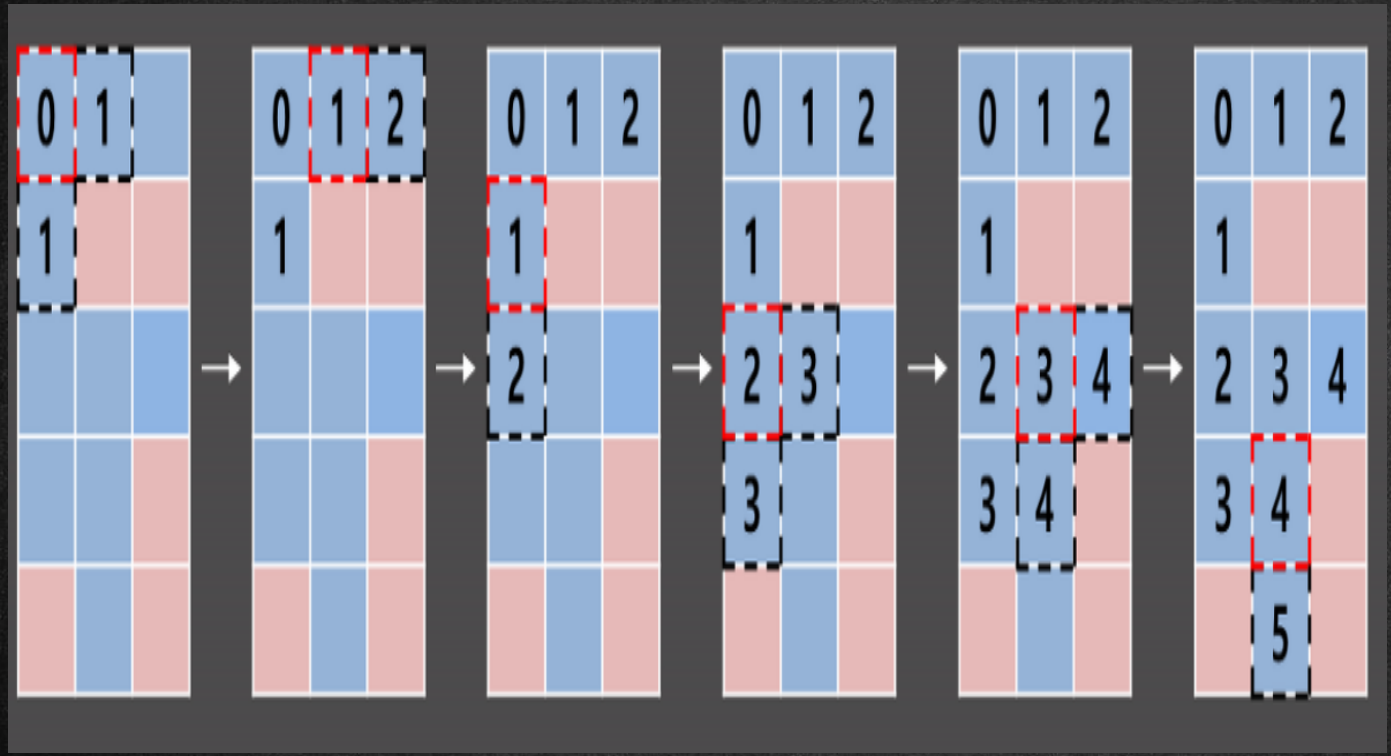
https://www.acmicpc.net/problem/2178

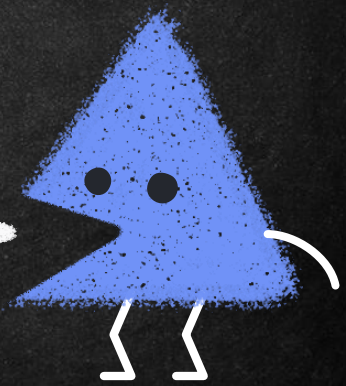
꼭 문제와 입출력을 확인하세요!!!

미로의 좌측 상단으로부터 우측 하단으로 가는 최단 경로의 길이를 찾는 문제입니다.
(0,0)에서 시작해서 사방으로 퍼져나가는 느낌이 들지 않나요...?
그림과 함께 살펴보죠!!!



백준(2178번-미로 탐색)

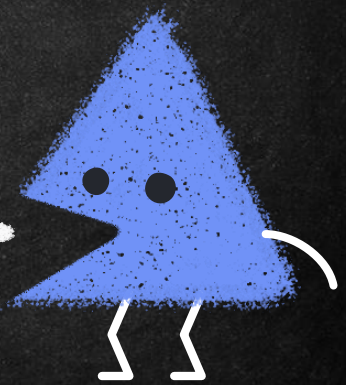




백준(2178번-미로 탐색)

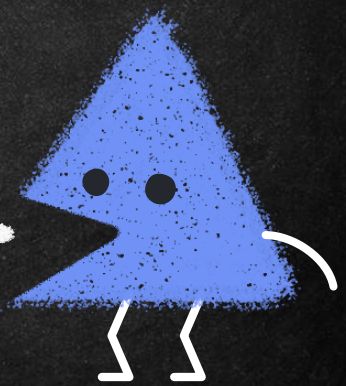
방문했다는 표시 대신에 (0,0)에서부터 깊어질수록
거리를 계산하면 됩니다.

단순히 BFS를 통해서 방문하는 것에서 끝나는 것이 아니라
시작점과의 거리를 전부 계산할 수 있습니다!!!



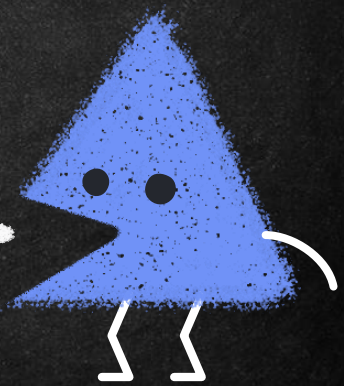
백준(2178번-미로 탐색) CODE

```cpp
#include <iostream>
#include <queue>
using namespace std;
#define X first
#define Y second
string board[102];
int dist[102][102];
int n, m;
int dx[4] = { 1,0,-1,0 };
int dy[4] = { 0,1,0,-1 };
```



백준(2178번-미로 탐색) CODE

```
int main(void) {
    cin >> n >> m;
    for (int i = 0; i < n; i++)
        cin >> board[i];
    for (int i = 0; i < n; i++) fill(dist[i], dist[i] + m, -1);
    queue<pair<int, int> > Q;
    Q.push({ 0,0 });
    dist[0][0] = 0;
    while (!Q.empty()) {
        auto cur = Q.front(); Q.pop();
        for (int dir = 0; dir < 4; dir++) {
            int nx = cur.X + dx[dir];
            int ny = cur.Y + dy[dir];
            if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
            if (dist[nx][ny] >= 0 || board[nx][ny] != '1') continue;
            dist[nx][ny] = dist[cur.X][cur.Y] + 1;
            Q.push({ nx,ny });
        }
    }
    cout << dist[n - 1][m - 1] + 1; // 문제의 특성상 거리+1이 정답
}
```

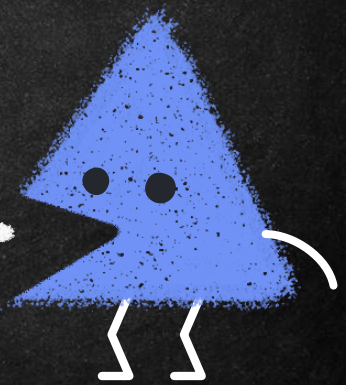


함께 풀어보면 좋은 문제들

백준 문제집

➔ Week 5에 들어있는 문제들 많지 않으니 학습한 내용들 참고해서 꼭 시도해보세요~!

BFS/ DFS는 정말 문제가 많습니다.
꼭 시간 내서 구현 방법이라도 알아가셨으면 좋겠습니다.



THANK YOU