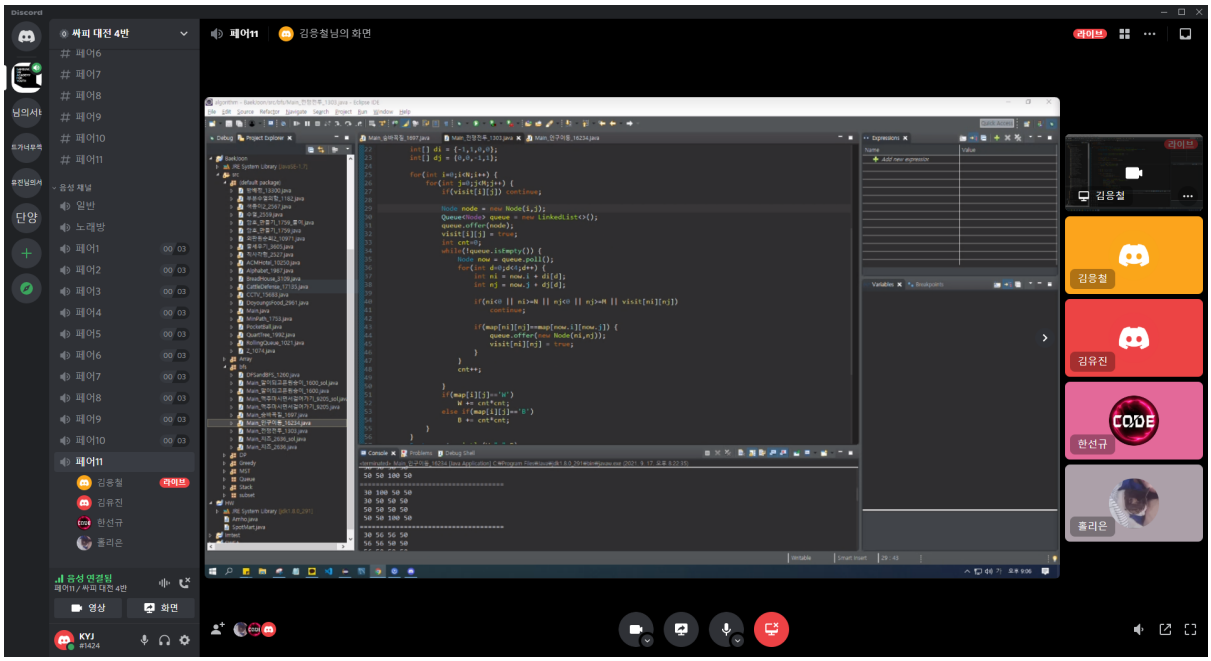
9월 2주차 스터디 발표

2021년 9월 17일 금요일 21:00 ~ 24:05

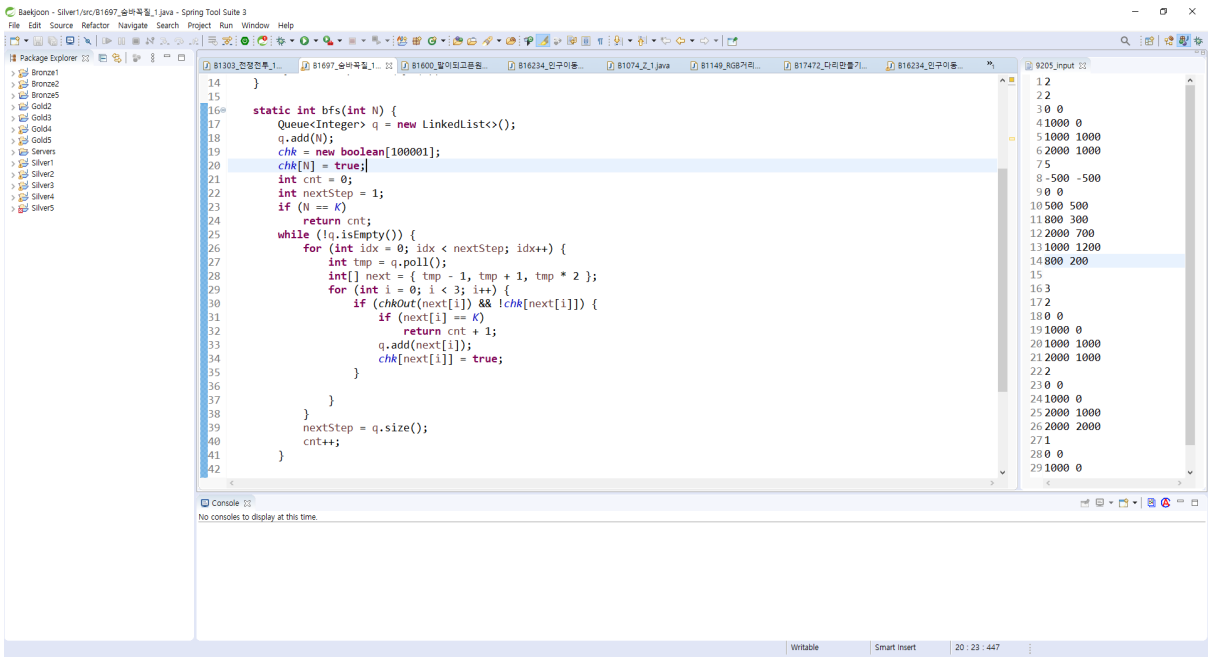
백준 1303 전쟁 전투 (발표자 : 김응철)




```
package bfs;
import java.io.*;
import java.util.*;

public class Main_전쟁전투_1303 {
  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    StringTokenizer st = new StringTokenizer(br.readLine());

    int M = Integer.parseInt(st.nextToken()); //가로
    int N = Integer.parseInt(st.nextToken()); //세로

    int B = 0, W = 0;
    char[][] map = new char[N][M];
    boolean[][] visit = new boolean[N][M];

    for(int i=0;i<N;i++) {
      String tmp = br.readLine();
      map[i] = tmp.toCharArray();
    }//input

    int[] di = {-1,1,0,0};
    int[] dj = {0,0,-1,1};

    for(int i=0;i<N;i++) {
      for(int j=0;j<M;j++) {//배열을 돌면서 상하좌우탐색
        if(visit[i][j]) continue;//들렸던 곳이면 continue

        Node node = new Node(i,j);
        Queue<Node> queue = new LinkedList<>();
        queue.offer(node);
        visit[i][j] = true;

        int cnt=0;
        while(!queue.isEmpty()) {//큐가 다 비워질때까지 탐색.
          Node now = queue.poll();
          for(int d=0;d<4;d++) {
            int ni = now.i + di[d];
            int nj = now.j + dj[d];

            if(ni<0 || ni>=N || nj<0 || nj>=M || visit[ni][nj])//다음 바라본 곳이 배열 밖이거나 한번 탐색한 곳이면 continue
```

```
              continue;

            if(map[ni][nj]==map[now.i][now.j]) {//바라본곳이 현재인덱스의 값과 같다면
              queue.offer(new Node(ni,nj));//큐에 추가
              visit[ni][nj] = true;//발자국남기기
            }
          }
          cnt++;//갯수 체크

        }
        if(map[i][j]=='W')//W면 W에 더하고 (처음 시작한 인덱스는 그대로 있으므로 그걸 기준으로 한다)
          W += cnt*cnt;
        else if(map[i][j]=='B')//B면 B에 더하고
          B += cnt*cnt;
      }
    }
    System.out.println(W+" "+B);

  }

  static class Node{
    int i,j;

    public Node(int i, int j) {
      super();
      this.i = i;
      this.j = j;
    }

  }

}
```

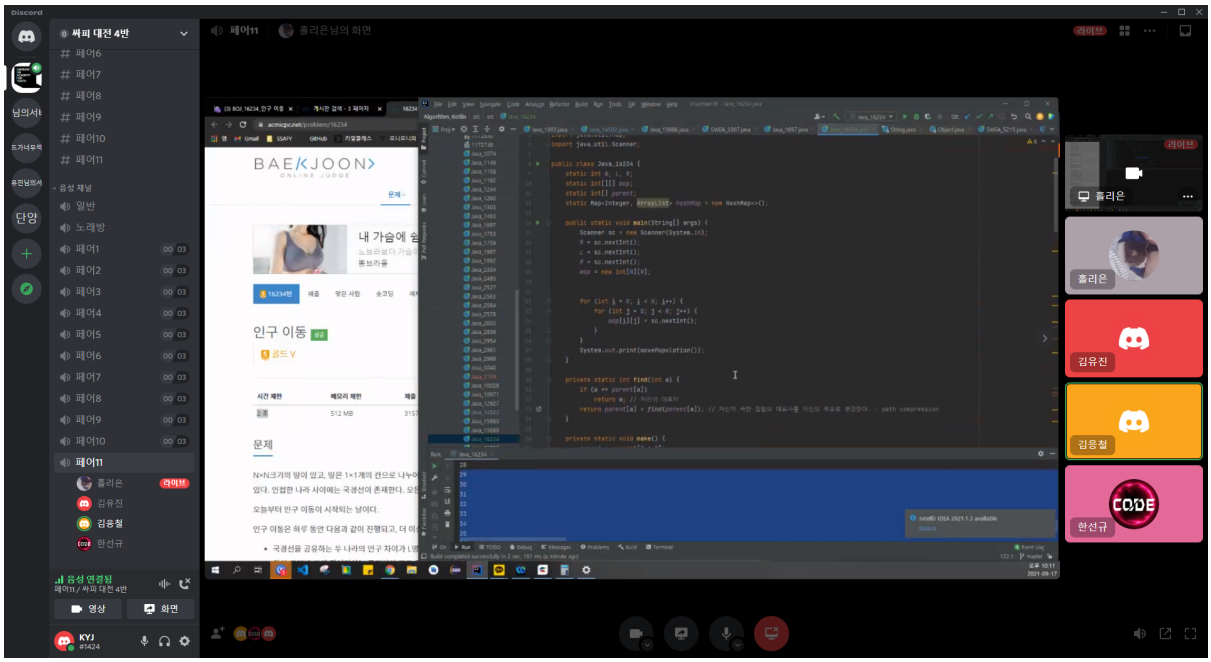
백준 1697 숨바꼭질 (발표자 : 김유진)




```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class B1697_숨바꼭질_1 {
  static boolean[] chk; // 체크용
  static int N, K; // 내 위치와 동생 위치를 저장할 변수

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    N = sc.nextInt(); // 내 위치 입력 받기
    K = sc.nextInt(); // 동생 위치 입력 받기
    System.out.println(bfs(N)); //리턴해준 값 출력
  }
```

```
  static int bfs(int N) {
    Queue<Integer> q = new LinkedList<>();
    q.add(N);
    chk = new boolean[100001]; // 1차원 맵 생성
    chk[N] = true; // 처음 위치 체크
    int cnt = 0; // 처음 스테므 0
    int nextStep = 1; // 처음에는 시작위치 1개라서
    if (N == K)
      return cnt;
    while (!q.isEmpty()) {
      for (int idx = 0; idx < nextStep; idx++) { // 해당 스텝에서 탐색해야 하는 위치를 전부 탐색한다.
        int tmp = q.poll();
        int[] next = { tmp - 1, tmp + 1, tmp * 2 }; // -1,+1, *2 칸 이동
        for (int i = 0; i < 3; i++) {
          if (chkOut(next[i]) && !chk[next[i]]) { // 이동 가능 범위를 벗어나지 않았다면
            if (next[i] == K) // 만약 동생의 위치와 동일하다면 끝
              return cnt + 1; // cnt는 1만큼 증가한 값을 리턴
            q.add(next[i]); // 다음에 탐색할 위치를
            chk[next[i]] = true;
          }

        }
      }
      nextStep = q.size(); // 다음에 탐색해야할 위치개수 갱신
      cnt++; // 스텝 증가
    }

    return Integer.MAX_VALUE; // 리턴이 필요해서 만듬
  }

  static boolean chkOut(int i) { // 맵 벗어나는지 확인
    if (i < 0 || i > 100000)
      return false; // 벗어나면 false 리턴
    return true; // 안 벗어나면 true 리턴
  }
}
```

백준 16234 인구이동 (발표자 : 정은이)




```
package src;

import java.util.*;

public class Java_16234_인구이동 {
    static int N, L, R;
    static int[][] map;
    static int[] parent;
    static Map<Integer, ArrayList> hashMap = new HashMap<>();

    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        N = sc.nextInt();
        L = sc.nextInt();
        R = sc.nextInt();
        map = new int[N][N];

        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                map[i][j] = sc.nextInt();
            }
        }
        System.out.print(movePopulation());
    }
// 1. union set 알고리즘을 활용
// 2. 인접해있고, 같은 무리라면 `union` 으로 합치면서 한바퀴 돌기
// 3. 다시 한 바퀴 돌면서 `HashMap` 자료구조를 이용하여 key : 부모의 인덱스 value : key를 부모로 가지는 자식인덱스들을 줄줄이 추가
// 4. `while (hashMap.size() < N * N)`  연합이 한개라도 없다면 `HashMap` 의 부모의 크기가 N*N 일거다.
// 5. `HashMap`을 key 중심으로 한바퀴 돌면서 (value인 부모가 같은 자식들의 어레이리스트)의 사이즈가 1 (본인) 이상일 경우에 sum 과 각 국가들이 가질 수 있는 인구
// 6. 다시 탐색하면서 연합이 없을때까지 2-5 반복

    private static int movePopulation() {
        int dayCnt = 0;

        bfs();

        // 6. 다시 탐색하면서 연합이 없을때까지 2-5 반복
        while (hashMap.size() < N * N) { //4. `while (hashMap.size() < N * N)`  연합이 한개라도 없다면 `HashMap` 의 부모의 크기가 N*N 일거다.
            dayCnt++;
            // 5. `HashMap`을 key 중심으로 한바퀴 돌면서 (value인 부모가 같은 자식들의 어레이리스트)의 사이즈가 1 (본인) 이상일 경우에 sum 과 각 국가들이 가질
            for (Integer key : hashMap.keySet()) {
                ArrayList<Integer> que = hashMap.get(key); // 키값을 부모로 가진 자식들의 어레이
                int queSize = que.size();
                if (queSize <= 1) continue;
                int sum = 0;
                // 연합의 인구 합 구하기
                for (int i = 0; i < queSize; i++) {
                    int tmp = que.get(i);
                    sum += map[tmp / N][tmp % N];
                }

                int people = sum / queSize;
                // 연합에 인구 할당
                for (int i = 0; i < queSize; i++) {
                    int tmp = que.get(i);
                    map[tmp / N][tmp % N] = people;
                }
            }
            bfs();
        }
        return dayCnt;
    }

    private static int find(int a) {
        if (a == parent[a])
            return a; // 자신이 대표자
        return parent[a] = find(parent[a]); // 자신이 속한 집합의 대표자를 자신의 부모로 변경한다.
    }

    private static void make() {
        parent = new int[N * N];
        // 모든 원소를 자신을 대표로 만듦.
        for (int i = 0; i < N * N; i++) {
            parent[i] = i;
        }
    }

    // 두 원소를 하나의 집합으로 합치기
    private static boolean union(int a, int b) {
        int aRoot = find(a);
        int bRoot = find(b);

        if (aRoot == bRoot)
            return false; // 이미 같은 집합으로 합치지 않는다.
        parent[bRoot] = aRoot;
        return true;
    }

    static void bfs() {
        hashMap.clear();
        make();

        int[] dx = {-1, 1, 0, 0};
        int[] dy = {0, 0, -1, 1};
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                for (int k = 0; k < 4; k++) {
                    int tx = i + dx[k];
```

```
                    int ty = j + dy[k];

                    if (tx < 0 || tx >= N || ty < 0 || ty >= N) continue;
                    int dif = Math.abs(map[i][j] - map[tx][ty]);
                    if (dif >= L && dif <= R) {
                        // 2. 인접해있고, 같은 무리라면 `union` 으로 합치면서 한바퀴 돌기
                        // 인접한 국가와 인구의 차이가 L 이상 R 이하면 union
                        union(i * N + j, tx * N + ty);
                    }
                }
            }
        }
        // 3. 다시 한 바퀴 돌면서 `HashMap` 자료구조를 이용하여 key : 부모의 인덱스 value : key를 부모로 가지는 자식인덱스들을 줄줄이 추가
        int p = 0;
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                p = find(i * N + j);
                if (!hashMap.containsKey(p)) hashMap.put(p, new ArrayList<>());
                hashMap.get(p).add(i * N + j); //  부모의 인덱스를 키 값, value 로 자식들의 어레이에 추가
            }
        }
    }
}
```

스터디 소감

김응철 :

BFS에 대해 더 확실하게 다져볼 수 있는 기회였습니다.
인구이동 문제를 풀면서 bfs도 메모리나 O(n)관리를 하지 않으면 시간초과가 난다는 것도 알게되었습니다.
은이님과 유진님과 서로의 풀이도 공유하면서 제가 몰랐던 새로운 풀이법들도 알게되었습니다.
견문을 넓힐수 있었던 좋은 스터디였습니다.

김유진 :

BFS를 사용한 풀이법만 떠올랐는데 은이님의 HashMap을 이용한 풀이를 보고 견문을 넓혔습니다.

HashMap 구조는 직접 사용해 본 적이 없었어 유용함을 잘 몰랐는데 은이님의 풀이를 보고 배우고 사용할 정도로 익혀두면

다양한 문제에 적용할 수 있다고 생각했습니다.

오랜만에 알고리즘을 시작해서 기억에 남아있는 방법만 사용해서 문제를 풀고 있습니다.

알고리즘 및 자료구조 복습을 통해서 머리 속에 있는 지식을 한번 정리하고 좀 더 다양한 방법을 사용할 수 있도록 생각해봐야겠습니다.

정은이 :

스터디를 신청한 이유는 약간의 강제성과 팀원들과 함께 코드 리뷰를 하는 것이었습니다.
9월 2주차 스터디가 시작함에 앞서 주어진 문제를 다 풀어보자 목표를 가지고 임했습니다.
다행히 스터디 1분 전에 가까스로 문제를 다 풀어가 저의 목표를 달성할 수 있었습니다.
또, 응철님과 유진님과 함께 많은 이야기를 나눌 수 있었습니다.
한 문제씩 맡아서 진행한 후, 다른 풀이가 있다면 그분이 다른 방법을 소개하는 방식으로 이루어졌습니다.
그로 인해, 저는 hashMap을 이용한 풀이를 소개할 수 있었고, 제가 접근을 잘 못 하는 bfs 을 이용한 문제 접근, 풀이 과정을
접할 수 있어서 유익하고 좋았습니다.

디스코드에서 직접 코드를 보면서 설명을 진행했기 때문에, 의문점을 가진 부분에 대해서는 직접 실행해보는 시간도 가졌습니다.
그 의문을 같이 해결하기 위해서 스터디원끼리 많은 이야기를 나눴고,
시간이 길어져 그 부분에 대해서는 각자 알아보기로 한 후 스터디를 마쳤습니다.
스터디를 마친 후에 그 부분에 대해서 연락을 통해 해결하였고, 스터디 시간 후에도 서로의 상황을 공유하며 이야기를 나눌 수 있다는 것이 너무 좋았습니다.
동지가 생겨 든든하고 많은 자극을 받게 되었습니다. 알고리즘 더 열심히 풀어야겠습니다.