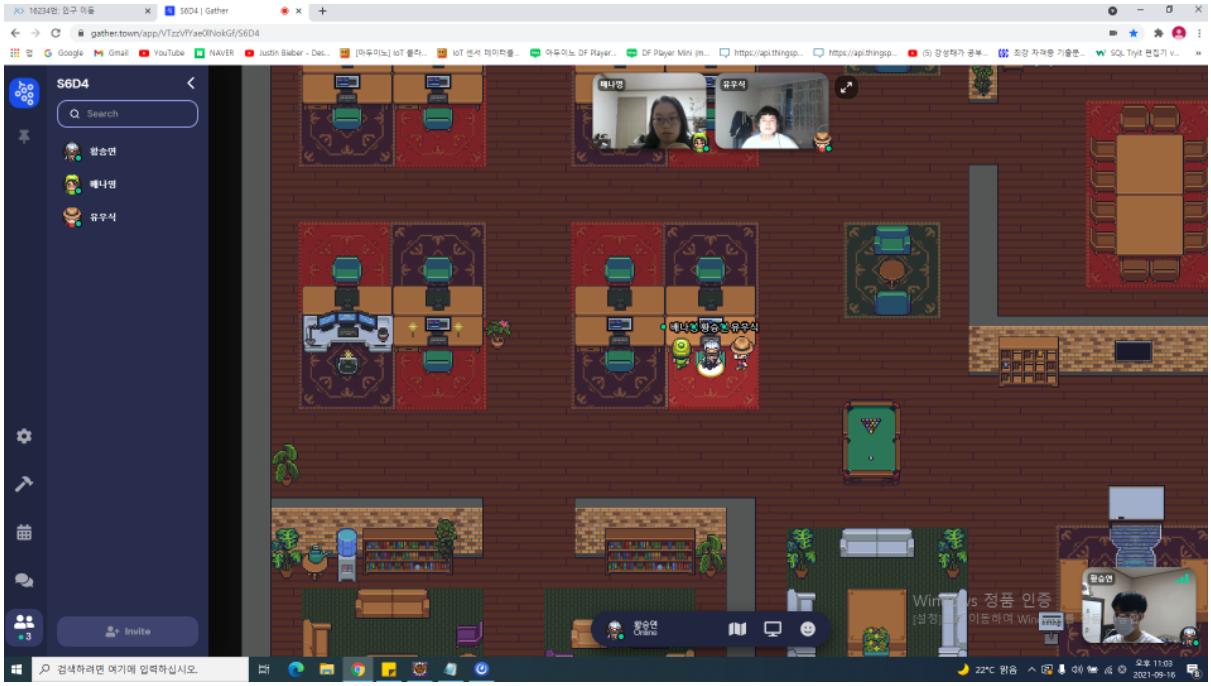
9월 2주차 스터디

- 발표 일자: 09.16 23:00
- 스터디원: 배나영,유우식,황승연
- 스터디 사진



1. 1697 숨바꼭질

```
package practice;

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

public class Boj_1697_숨바꼭질 {
public static void main(String[] args) {
Scanner sc = new Scanner([System.in](http://system.in/));

  int N = sc.nextInt();
  int K = sc.nextInt();

  System.out.println(bfs(N,K));
```

```
}

static int bfs(int N,int K) {     //가장 빠른 시간 찾아야하니까 bfs
  boolean[] visited = new boolean[100001];
  Queue<Integer> queue = new LinkedList<Integer>();

  queue.offer(N);
  visited[N] =true;
  int distance =0;    //시간 변수(답)

  while(!queue.isEmpty()) {
    int size = queue.size();    //현재시간에서 갈수 있는 곳 모두 방문해볼거니까 사이즈 생성

    for(int s=0;s<size;s++) {   //현재 시간에서 갈 수 있는 곳 확인

      int num = queue.poll();

      if(num==K) {        //동생 찾으면 나가기
        return distance;
      }
      if(num-1>=0 && !visited[num-1]) {   //1빼보고 방문 안한 곳이면 큐에 넣기(큐에서 대기하다가
 나중에 갈거)
        queue.offer(num-1);
        visited[num-1] = true;
      }
      if(num+1<=100000 && !visited[num+1]) {  ////1 더해보고 방문 안한 곳이면 큐에 넣기(큐에서
 대기하다가 나중에 갈거)
        queue.offer(num+1);
        visited[num+1] = true;
      }
      if(2*num<=100000 && !visited[2*num]) {  //*2해보고 방문 안한 곳이면 큐에 넣기(큐에서 대기
하다가 나중에 갈거)
        queue.offer(2*num);
        visited[2*num] = true;
      }
    }
    distance++;
  }
  return -1;
}
```

- 황승연:

가장 짧은 시간을 구하기에 bfs를 이용해서 풀었다.

나영님은 bfs로 비슷한 풀이를 진행하셨는데 큐에 넣을 때 num이 배열 범위를 벗어나는 조건을 처리하지 않아서 에러가 났다고 하셨다.

방문체크와 배열범위를 벗어나지 않는 지 조건을 확인해주는 것을 잊지 않도록 해야겠다.

- 배나영

팀원의 코드를 보고 내가 고려하지 않은 조건들이 많다는 것을 깨달았다. 앞으로 코드를 적기 전에 좀 더 꼼꼼하게 여러 경우를 생각해봐야겠다.

2. 1303 전쟁

```
package practice;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

public class Boj_1303_전쟁 {

static int[] di = {-1,1,0,0};
static int[] dj = {0,0,-1,1};
static boolean[][] visited;
static int N;
static int M;
static char[][] map;
static int cnt;
public static void main(String[] args) throws IOException {
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

  StringTokenizer st = new StringTokenizer(br.readLine());
  N = Integer.parseInt(st.nextToken());
  M = Integer.parseInt(st.nextToken());
  map = new char[M][N];

  for(int i=0;i<M;i++) {
    String str = br.readLine();
    for(int j=0;j<N;j++) {
      map[i][j] = str.charAt(j);
    }
  }
  //입력 끝
  visited = new boolean[M][N];
  cnt = 0;    //집합 각각의 수 저장할 변수
  int resultW =0;   //흰색 위력(답)
  int resultB =0;   //블루 위력(답)

  for(int i=0;i<M;i++) {
    for(int j=0;j<N;j++) {
      if(!visited[i][j] && map[i][j] =='W') {   //맵 모두 탐색하면서 방문 안했고 흰색이면
        cnt = 0;                //집합 초기화
        dfs(i,j,'W');             //흰색 매개변수로 주면서 탐색
        resultW += cnt*cnt;           //위력 더해주기
      }
      else if(!visited[i][j] && map[i][j] =='B') {  //맵 모두 탐색하면서 방문 안했고 블루면
        cnt=0;
        dfs(i,j,'B');
        resultB += cnt*cnt;
      }
    }
  }
```

```
  System.out.println(resultW+" "+resultB);
}

 static void dfs(int nowi,int nowj,char C) {
   visited[nowi][nowj] = true;        //방문체크
   cnt++;             //나라개수 더해주기
   for(int d=0;d<4;d++) {
     int nexti = nowi+di[d];
     int nextj = nowj+dj[d];

     if(nexti<0 || nextj<0 || nexti>=M || nextj >=N) {
       continue;
     }
     if(!visited[nexti][nextj] && map[nexti][nextj] == C) {   //매개변수로 받은 나라면 계속
 탐색 이어가기
       dfs(nexti,nextj,C);
     }
   }
 }
}
```

- 황승연

2차원 배열을 탐색하며 흩어져있는 진영별로 더해주고 각각 위력을 계산해서 합쳐주었다. 흰색과 파란색을 따로 탐색해야 했기에 dfs함수에 매개변수로 char를 주었다.

3. 16234 인구이동

```
package practice;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.StringTokenizer;

public class Boj_16234_인구이동 {

static int[] di = {-1,1,0,0};
static int[] dj = {0,0,-1,1};
static int N;
static int L;
static int R;
static int[][] population;
static int union;
static int sum;
static Queue<Point> queue;
public static void main(String[] args) throws IOException {
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

```
  StringTokenizer st = new StringTokenizer(br.readLine());
  N = Integer.parseInt(st.nextToken());
  L = Integer.parseInt(st.nextToken());
  R = Integer.parseInt(st.nextToken());

  population = new int[N][N];       // 각각 나라 인구
  for(int i=0;i<N;i++) {
    st = new StringTokenizer(br.readLine());
    for(int j=0;j<N;j++) {
      population[i][j] = Integer.parseInt(st.nextToken());
    }
  }
  //입력 끝

  queue = new LinkedList<>();     //연합별로 인구수 갱신해줘야해서 필요
  boolean[][] visited = new boolean[N][N];    //방문체크
  int day =0;             //날짜 변수(답)

  for (int k = 0; k < 2000; k++) {    //날짜 증가
    boolean flag = false;       // 이 날에 인구이동이 있는지 체크할 불린변수
    visited = new boolean[N][N];    //새로운 날이 되면 방문체크 새로만들기

    for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) { //나라들 돌면서
        if (!visited[i][j]) {   //방문하지 않은 나라라면
          union = 0;        //연합인 나라 수 초기화
          sum = 0;        //연합인 나라 인구 합 초기화
          dfs(i, j, visited);   // 연합찾기
          if (union >1) {     //연합인 나라수가 2이상이면 그날은 인구이동을 하니까(1이면 연합없이 독
고다이)
            flag = true;    //인구이동 트루
          }
          int each = sum / union;   //연합인 나라 가각에 배치할 인구
          while (!queue.isEmpty()) {    //연합인 나라들 저장된 큐
            Point point = queue.poll();
            population[point.i][point.j] = each; // 연합국 인구 갱신
          }
        }

      }
    }
    if(!flag) {       //이 날에 인구이동이 없었다면
      System.out.println(day);    //답 출력
      break;
    }
    day++;
  }



}

static void dfs(int nowi,int nowj, boolean[][] visited) {     //연합찾기

  visited[nowi][nowj] = true;         //방문체크
  union++;                //연합증가
  sum += population[nowi][nowj];      //연합 인구 증가
  queue.offer(new Point(nowi, nowj));   //큐에 연합인 나라들 넣기
```

```
  for(int d=0;d<4;d++) {          //현재 나라에서 인접나라 보면서 연합되는지 체크
    int nexti = nowi+di[d];
    int nextj = nowj+dj[d];

    if(nexti<0 || nextj <0 || nexti>=N || nextj >=N) {
      continue;
    }
    if(!visited[nexti][nextj] && Math.abs(population[nexti][nextj]-population[nowi][no
wj])>=L && Math.abs(population[nexti][nextj]-population[nowi][nowj])<=R) {
      dfs(nexti,nextj,visited);
      //연합 가능하고 아직 방문 안한 나라라면 그 나라에서 계속 탐색하러 들어가기
    }


  }


}

static class Point{
  int i;
  int j;
  public Point(int i, int j) {
    super();
    this.i = i;
    this.j = j;
  }

}
}
```

- 유우식

인구이동을 할 때 인구수를 연합 마다 일정하게 나누는게 결과에 영향이 미치는지 궁금해졌고,

결과에 영향이 없다면 굳이 인구수를 나눌 필요가 있을 까 고민을 해보고 다시 풀어보았지만,

연합에 따라 인구수를 나누지 않으면 시간초과가 떠서 엄청 큰 영향을 주었다.

- 황승연

우식님과 풀이가 비슷한데 나는 메인에서 많은 것을 처리해서 가독성이 떨어진다고 느꼈다.

우식님은 인구이동 확인과 국경선 체크를 각각 함수로 만드셔서 좀 더 함수의 역할이 잘 와닿는다고 생각했다. 나도 다음부터는 각각의 역할에 따라 함수를 나누어 사용해봐야겠다고 생각했다

또한, 2차원 배열 탐색에 날짜가 증가할때마다 반복해줘야해서 3중 for문을 쓰게 됐는데 시간초과가 날까 걱정했지만 N이 작아서인지 다행히 시간초과는 나지 않았다.