2기 / 1주차_구현

🌟 구현

구현이란 머릿속에 있는 알고리즘을 소스코드로 바꾸는 과정이다.

구현 문제 유형은 모든 범위의 코딩 테스트 문제 유형을 포함하는 개념이라고 할 수 있다.

1. 백준 17952_과제는 끝나지 않아!

💡 접근법

최근에 나온 순서대로 과제를 진행하고, 만약 과제가 끝나지 않으면 중단하고 새 과제를 시작한다는 말에서 스택을 사용해야겠다고 생각했다.
걱정된 부분이 있었는데 시간복잡도와 공간복잡도였다.
최악의 경우 매번 스택의 pop, push가 발생하기 때문이었고, 수업시간이 최대 1,000,000분이기 때문에 스택에 1,000,000개의 데이터가 쌓일 수도 있다는 것이었다.

🔑 풀이

과제가 있는 시간과 없는 시간으로 나눴다.

1. 과제가 있는 경우
    a. 입력 받은 과제의 소요시간이 1인 경우
    → 해당 시간에 완료되는 과제이기에 스택의 삽입, 삭제 없이 바로 총점에 더해준다.
    b. 아닌 경우
    → 스택에 삽입한다. (이 때, 시간은 1을 뺀 후 진행한다.)
2. 과제가 없는 경우
    a. 스택이 비어있는 경우
    → 아무 작업도 하지 않는다.
    b. 스택의 top 시간이 1인 경우
    → 1-a와 동일하다.
    c. 아닌 경우
    → 1-b와 동일하다.

시간이 1인지 아닌지 판단한 후 진행되는 작업은 과제가 있는 경우와 없는 경우가 동일하기 때문에 메서드를 따로 분리했다.

💻 코드

660 ms, 208952 KB

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

public class BOJ_17952 {
    private static int totalScore = 0;
    private static Stack<Assignment> stack;

    static class Assignment {
        private int score;
        private int time;

        public Assignment(int score, int time) {
            this.score = score;
            this.time = time;
        }
    }

    public static void main(String[] args) throws IOException {
        solve();
        output();
    }

    private static void output() {
        System.out.println(totalScore);
    }

    private static void solve() throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int minute = Integer.parseInt(br.readLine());

        stack = new Stack<>();
        StringTokenizer st;

        for (int i = 0; i < minute; i++) {
            st = new StringTokenizer(br.readLine());
            int check = Integer.parseInt(st.nextToken());

            // 과제가 없는 시간
            if (check == 0) {
                if (stack.isEmpty()) continue;

                Assignment current = stack.pop();
```

```
                checkTime(current);
            }

            // 과제가 있는 시간
            else {
                int score = Integer.parseInt(st.nextToken());
                int time = Integer.parseInt(st.nextToken());

                checkTime(new Assignment(score, time));
            }
        }
    }

    // 과제의 남은 소요시간 확인
    private static void checkTime(Assignment assignment) {
        int score = assignment.score;
        int time = assignment.time;

        if (time == 1) {
            totalScore += score;
        } else {
            stack.push(new Assignment(score, time - 1));
        }
    }
}
```

2. 백준 24954_물약 구매

💡 접근법

예제를 이해하는 게 제일 어려웠다🥺

막상 이해하고 나니 순열 문제임을 알 수 있었다.

🔑 풀이

현재 구매한 물약에 해당하는 다른 물약 할인이 있을 경우 할인을 적용한다.

그리고 DFS로 모든 경우를 전부 확인한다.

💻 코드

816 ms, 298108 KB

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
```

```
import java.util.Arrays;
import java.util.List;
import java.util.StringTokenizer;

public class BOJ_24954 {
    private static int minPrice = Integer.MAX_VALUE;
    private static int cntPotion; // 물약 개수
    private static int[] price; // 각 물약의 가격
    private static List<List<Potion>> graph; // 할인 정보 리스트
    private static boolean[] isVisited;

    private static class Potion{
        private int number;
        private int discount;

        public Potion(int number, int discount) {
            this.number = number;
            this.discount = discount;
        }
    }

    public static void main(String[] args) throws IOException {
        input();
        for (int i = 1; i <= cntPotion; i++) {
            isVisited[i] = true;
            solve(i, price[i], 1, price);
            isVisited[i] = false;
        }
        output();
    }

    private static void output() {
        System.out.println(minPrice);
    }

    private static int[] copyArray(int[] array) {
        int[] copy = new int[array.length];
        System.arraycopy(array, 0, copy, 0, array.length);

        return copy;
    }

    private static void solve(int start, int totalPrice, int depth, int[] originPrice) {
        if (depth == cntPotion) {
            minPrice = Math.min(minPrice, totalPrice);
        }

        int[] copyPrice = copyArray(originPrice);

        for (Potion potion : graph.get(start)) {
            int number = potion.number;
            int discount = potion.discount;

            copyPrice[number] -= discount;
            if (copyPrice[number] <= 0) copyPrice[number] = 1;
        }

        for (int k = 1; k <= cntPotion; k++) {
```

```
            if (!isVisited[k]) {
                isVisited[k] = true;
                solve(k, totalPrice + copyPrice[k], depth + 1, copyPrice);
                isVisited[k] = false;
            }
        }

    }

    private static void input() throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        cntPotion = Integer.parseInt(br.readLine());

        graph = new ArrayList<>();
        for (int i = 0; i <= cntPotion; i++) {
            graph.add(new ArrayList<>());
        }

        price = new int[cntPotion + 1];
        StringTokenizer st = new StringTokenizer(br.readLine());
        for (int i = 0; i < cntPotion; i++) {
            price[i + 1] = Integer.parseInt(st.nextToken());
        }

        for (int i = 0; i < cntPotion; i++) {
            int cnt = Integer.parseInt(br.readLine());

            for (int k = 0; k < cnt; k++) {
                st = new StringTokenizer(br.readLine());

                int discountNum = Integer.parseInt(st.nextToken());
                int discountPrice = Integer.parseInt(st.nextToken());
                graph.get(i + 1).add(new Potion(discountNum, discountPrice));
            }
        }

        isVisited = new boolean[cntPotion + 1];
    }
}
```

3. 백준 14500_테트로미노

💡 접근법

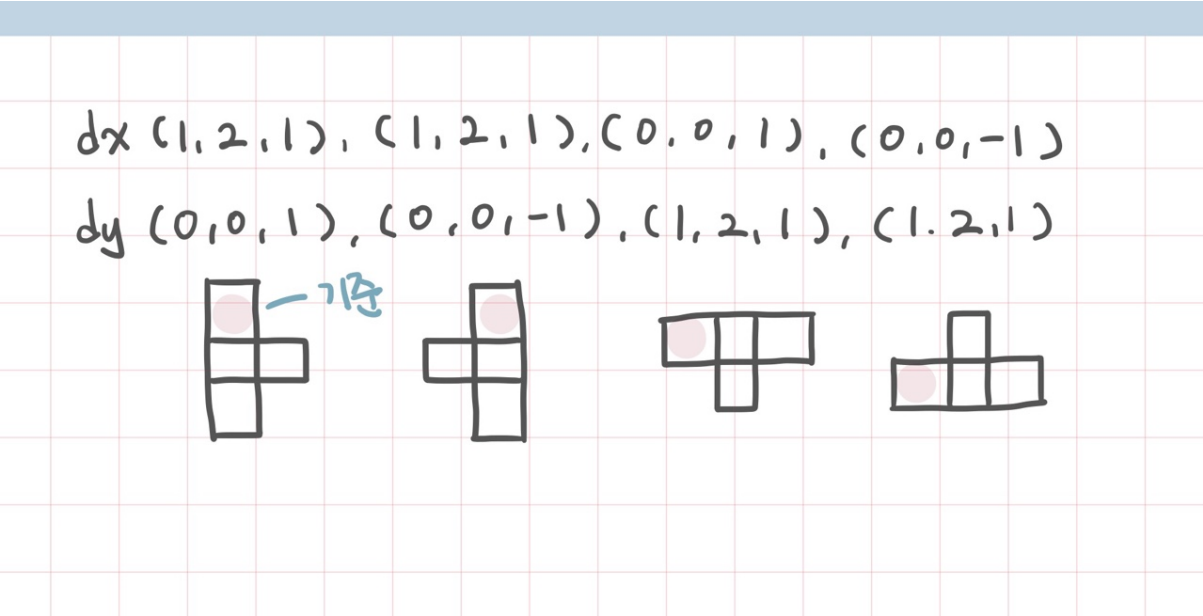
조각이 회전이나 대칭이 가능하다는 부분에서 그럼 연속된 4칸을 확인했을 때 조각을 못 만드는 경우가 있을까를 생각해봤다.
아닐 수도 있지만 내가 계산했을 땐 없었기 때문에 완전탐색으로 4칸만 확인하면 되겠구나 생각했다.

🔑 풀이

문제는 “ㅗ” 모양 블럭이었다.

연속된 탐색으로는 만들 수 없는 모양이라서 따로 계산한 후, dfs 결과 최댓값과 “ㅗ” 블럭 결과 최댓값 중 더 큰 값을 정답으로 출력했다.

“ㅗ” 블럭 확인할 때 좌표에 대해서 조금 더 설명하자면,



이렇게 4가지 경우를 나타낸 것이다.

💻 코드

680 ms, 93944 KB

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

public class BOJ_14500 {
    private static int row, column, answer;
    private static int[][] board;
    private static int max = 0;
    private static final int TETROMINO_LENTH = 4;
    private static boolean[][] isVisited;
    private static int[] dx = {1, 0, -1, 0};
    private static int[] dy = {0, 1, 0, -1};

    public static void main(String[] args) throws IOException {
        input();
        solve();
```

```
        output();
    }

    private static void output() {
        System.out.println(max);
    }

    private static void solve() {
        for (int i = 0; i < row; i++) {
            for (int k = 0; k < column; k++) {
                isVisited[i][k] = true;
                dfs(i, k, board[i][k], 1);
                goSpecialBlock(i, k, board[i][k]);
                isVisited[i][k] = false;
            }
        }
    }

    private static void goSpecialBlock(int x, int y, int total) {
        int[][] dx = {{1, 2, 1}, {1, 2, 1}, {0, 0, 1}, {0, 0, -1}};
        int[][] dy = {{0, 0, 1}, {0, 0, -1}, {1, 2, 1}, {1, 2, 1}};

        for (int i = 0; i < 4; i++) {
            int sum = 0;
            for (int k = 0; k < 3; k++) {
                int nextX = x + dx[i][k];
                int nextY = y + dy[i][k];

                if (nextX < 0 || nextY < 0 || nextX >= row || nextY >= column) {
                    break;
                }

                sum += board[nextX][nextY];
                if (k == 2) max = Math.max(max, total + sum);
            }
        }
    }

    private static void dfs(int x, int y, int total, int depth) {
        if (depth == TETROMINO_LENTH) {
            max = Math.max(max, total);
            return;
        }

        for (int i = 0; i < 4; i++) {
            int nextX = x + dx[i];
            int nextY = y + dy[i];

            if (nextX >= 0 && nextY >= 0 && nextX < row && nextY < column) {
                if (!isVisited[nextX][nextY]) {
                    isVisited[nextX][nextY] = true;
                    dfs(nextX, nextY, total + board[nextX][nextY], depth + 1);
                    isVisited[nextX][nextY] = false;
                }
            }
        }
    }
```

```
    private static void input() throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StringTokenizer st = new StringTokenizer(br.readLine());

        row = Integer.parseInt(st.nextToken());
        column = Integer.parseInt(st.nextToken());

        board = new int[row][column];
        isVisited = new boolean[row][column];

        for (int i = 0; i < row; i++) {
            st = new StringTokenizer(br.readLine());
            for (int k = 0; k < column; k++) {
                board[i][k] = Integer.parseInt(st.nextToken());
            }
        }
    }
}
```