## **[[백준] 16637번 - 괄호 추가하기](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-16637%EB%B2%88-%EA%B4%84%ED%98%B8-%EC%B6%94%EA%B0%80%ED%95%98%EA%B8%B0)**

2020. 4. 1. 07:43

[BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [삼성A형](https://dheldh77.tistory.com/tag/%EC%82%BC%EC%84%B1A%ED%98%95), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

길이가 N인 수식이 있다. 수식은 0보다 크거나 같고, 9보다 작거나 같은 정수와 연산자(+, -, ×)로 이루어져 있다. 연산자 우선순위는 모두 동일하기 때문에, 수식을 계산할 때는 왼쪽에서부터 순서대로 계산해야 한다. 예를 들어, 3+8×7-9×2의 결과는 136이다.

수식에 괄호를 추가하면, 괄호 안에 들어있는 식은 먼저 계산해야 한다. 단, 괄호 안에는 연산자가 하나만 들어 있어야 한다. 예를 들어, 3+8×7-9×2에 괄호를 3+(8×7)-(9×2)와 같이 추가했으면, 식의 결과는 41이 된다. 하지만, 중첩된 괄호는 사용할 수 없다. 즉, 3+((8×7)-9)×2, 3+((8×7)-(9×2))은 모두 괄호 안에 괄호가 있기 때문에, 올바른 식이 아니다.

수식이 주어졌을 때, 괄호를 적절히 추가해 만들 수 있는 식의 결과의 최댓값을 구하는 프로그램을 작성하시오. 추가하는 괄호 개수의 제한은 없으며, 추가하지 않아도 된다.

### 입력 >

첫째 줄에 수식의 길이 N(1 ≤ N ≤ 19)가 주어진다. 둘째 줄에는 수식이 주어진다. 수식에 포함된 정수는 모두 0보다 크거나 같고, 9보다 작거나 같다. 문자열은 정수로 시작하고, 연산자와 정수가 번갈아가면서 나온다. 연산자는 +, -, \* 중 하나이다. 여기서 \*는 곱하기 연산을 나타내는 × 연산이다. 항상 올바른 수식만 주어지기 때문에, N은 홀수이다.

### 출력 >

첫째 줄에 괄호를 적절히 추가해서 얻을 수 있는 결과의 최댓값을 출력한다. 정답은 231보다 작고, -231보다 크다.

### 해결방법 >

DFS를 사용했다.

 n개의 연산기호가 주어졌을 때, 연산기호의 부분집합을 구하는 방법을 생각할 수 있다.

 괄호안에는 한 개의 연산기호만 들어갈 수 있기 때문이다.

 여기서 부분집합이란 괄호로 묶어 먼저 처리되는 연산을 의미한다.

 또한 i번째 연산자가 부분집합에 포함되었을때,

 중첩괄호가 허용되지 않기때문에 i+1번째 연산자는 부분집합에 포함될 수 없다.

 아래와 같은 과정으로 진행된다.

 1) 입력된 문자열을 연산자와 숫자로 나눈어 LinkedList에 저장한다.

 2) 연산자가 저장된 LinkedList를 기준으로 DFS탐색을 한다.

 3) 조합이기 때문에 i번째 원소를 포함하는 경우와 포함하지 않는 경우로 DFS탐색을 한다.

 4) i번째 원소를 포함할 경우, i번째 연산자를 제거하고 i번째, i+1번째 숫자를 피연산자로 연산결과를 i번째에 숫자에 삽입한다.

 5) i+1번째 연산자로 3)번 과정을 다시 반복한다.

 i번째와 i+1번째 연산자를 부분집합으로 함께 묶을 수 없지만, i번째 연산자를 제거했을때 i+1번째 연산자는 원래 제거한 연산자의 i+2번째 이므로 가능한 부분집합이 된다.

 6) 부분집합을 구했을 때, LinkedList에 남아있는 숫자와 연산자를 순서대로 계산한다.

 7) 최솟값인 결과값을 구한다.

### [JAVA]

package baekjoon;

import java.io.\*;

import java.util.\*;

public class BOJ\_16637 {

static int len;

static int max = -2147483648;

public static void cal(LinkedList<Integer> n, LinkedList<Character> o, int idx) {

char op = o.remove(idx);

int num1 = n.remove(idx);

int num2 = n.remove(idx);

if(op == '+') {

n.add(idx, num1 + num2);

}

else if(op == '-') {

n.add(idx, num1 - num2);

}

else if(op == '\*') {

n.add(idx, num1 \* num2);

}

}

public static void dfs(int x, int flag, LinkedList<Integer> n, LinkedList<Character> o) {

LinkedList<Integer> nums = new LinkedList<>();

LinkedList<Character> op = new LinkedList<>();

nums.addAll(n);

op.addAll(o);

if(x > op.size() - 1 || x > len - 1) {

while(nums.size() > 1) {

cal(nums, op, 0);

}

max = Math.max(nums.get(0), max);

return;

}

if(flag == 1) {

cal(nums, op, x);

}

dfs(x + 1, 1, nums, op);

dfs(x + 1, 0, nums, op);

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

len = n / 2;

String s = br.readLine();

LinkedList<Integer> nums = new LinkedList<>();

LinkedList<Character> op = new LinkedList<>();

for(int i = 0; i < n; i++) {

char c = s.charAt(i);

if(i % 2 == 0) nums.add(c - '0');

else op.add(c);

}

dfs(0, 0, nums, op);

dfs(0, 1, nums, op);

System.out.println(max);

}

}

## [**[백준] 17281번 - ⚾**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-17281%EB%B2%88-%E2%9A%BE)

2020. 3. 27. 06:28

[17281](https://dheldh77.tistory.com/tag/17281), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [삼성A형](https://dheldh77.tistory.com/tag/%EC%82%BC%EC%84%B1A%ED%98%95), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

⚾는 9명으로 이루어진 두 팀이 공격과 수비를 번갈아 하는 게임이다. 하나의 이닝은 공격과 수비로 이루어져 있고, 총 N이닝동안 게임을 진행해야 한다. 한 이닝에 3아웃이 발생하면 이닝이 종료되고, 두 팀이 공격과 수비를 서로 바꾼다.

두 팀은 경기가 시작하기 전까지 타순(타자가 타석에 서는 순서)을 정해야 하고, 경기 중에는 타순을 변경할 수 없다. 9번 타자까지 공을 쳤는데 3아웃이 발생하지 않은 상태면 이닝은 끝나지 않고, 1번 타자가 다시 타석에 선다. 타순은 이닝이 변경되어도 순서를 유지해야 한다. 예를 들어, 2이닝에 6번 타자가 마지막 타자였다면, 3이닝은 7번 타자부터 타석에 선다.

공격은 투수가 던진 공을 타석에 있는 타자가 치는 것이다. 공격 팀의 선수가 1루, 2루, 3루를 거쳐서 홈에 도착하면 1점을 득점한다. 타자가 홈에 도착하지 못하고 1루, 2루, 3루 중 하나에 머물러있을 수 있다. 루에 있는 선수를 주자라고 한다. 이닝이 시작될 때는 주자는 없다.

타자가 공을 쳐서 얻을 수 있는 결과는 안타, 2루타, 3루타, 홈런, 아웃 중 하나이다. 각각이 발생했을 때, 벌어지는 일은 다음과 같다.

* 안타: 타자와 모든 주자가 한 루씩 진루한다.
* 2루타: 타자와 모든 주자가 두 루씩 진루한다.
* 3루타: 타자와 모든 주자가 세 루씩 진루한다.
* 홈런: 타자와 모든 주자가 홈까지 진루한다.
* 아웃: 모든 주자는 진루하지 못하고, 공격 팀에 아웃이 하나 증가한다.

한 야구팀의 감독 아인타는 타순을 정하려고 한다. 아인타 팀의 선수는 총 9명이 있고, 1번부터 9번까지 번호가 매겨져 있다. 아인타는 자신이 가장 좋아하는 선수인 1번 선수를 4번 타자로 미리 결정했다. 이제 다른 선수의 타순을 모두 결정해야 한다. 아인타는 각 선수가 각 이닝에서 어떤 결과를 얻는지 미리 알고 있다. 가장 많은 득점을 하는 타순을 찾고, 그 때의 득점을 구해보자.

### 입력 >

첫째 줄에 이닝 수 N(2 ≤ N ≤ 50)이 주어진다. 둘째 줄부터 N개의 줄에는 각 선수가 각 이닝에서 얻는 결과가 1번 이닝부터 N번 이닝까지 순서대로 주어진다. 이닝에서 얻는 결과는 9개의 정수가 공백으로 구분되어져 있다. 각 결과가 의미하는 정수는 다음과 같다.

* 안타: 1
* 2루타: 2
* 3루타: 3
* 홈런: 4
* 아웃: 0

각 이닝에는 아웃을 기록하는 타자가 적어도 한 명 존재한다.

### 출력 >

아인타팀이 얻을 수 있는 최대 점수를 출력한다.

### 해결방법 >

DFS(깊이우선탐색) 해결할 수 있는 문제이다.

9명의 선수를 순서대로 배치하는 모든 경우의 수를 찾고 이 때의 점수를 계산한다.

1) 9명의 선수의 순서 배치

DFS를 통해 순서를 배치하는데 이 때, 문제에서 1번 선수는 4번째 타자로 고정되어있으므로 이 경우를 고려해서 탐색한다.

2) 점수 계산

점수 계산의 경우는 일반적인 야구와 같은 방식으로 계산한다.

쓰리아웃이 발생하기 전까지 진루한 선수의 수를 카운트하고, 쓰리아웃이 발생했을 때 루에 남아있는 선수의 수를 빼준다.

스택을 이용해 루에 남아있는 선수의 수를 저장했다

선수들의 결과를 순서대로 스택에 담고

쓰리아웃이 발생했을 때 ,다시 마지막 선수부터 결과를 더해 이 값이 3을 초과할때까지

스택에서 뽑은 선수의 수를 총 진루한 선수의 수에서 빼준다.

이러한 방식으로 각 선수 배치에 따른 결과를 구하고 이 중 최대값을 출력한다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_17281\_2 {

static int max = 0;

static int[] visit = new int[9];

static int[] order = new int[9];

static int[][] record;

static int n;

public static void play() {

int score = 0;

int j = 6;

for(int i = 0; i < n; i++) {

int out = 0;

int run = 0;

Stack <Integer> st = new Stack<>();

while(out < 3) {

if(record[i][order[j]] == 0) out++;

else {

run++;

st.push(record[i][order[j]]);

}

j++;

if(j > 8) j = 0;

}

int k = 0;

int remain = 0;

while(!st.isEmpty()) {

k += st.pop();

if(k > 3) break;

remain++;

}

run -= remain;

score += run;

}

max = Math.max(score, max);

}

public static void dfs(int x, int depth) {

visit[x] = 1;

order[depth] = x;

if(depth == 8) {

play();

visit[x] = 0;

order[depth] = 0;

return;

}

for(int i = 0; i < 9; i++) {

if(visit[i] == 0) dfs(i, depth + 1);

}

order[depth] = 0;

visit[x] = 0;

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

StringTokenizer tk = new StringTokenizer(br.readLine());

n = Integer.parseInt(tk.nextToken());

record = new int [n][9];

int[] scores = new int[9];

for(int i = 0; i < n; i++) {

tk = new StringTokenizer(br.readLine());

for(int j = 0; j < 9; j++) {

record[i][j] = Integer.parseInt(tk.nextToken());

scores[j] += record[i][j];

}

}

int best = 0;

int idx = 0;

for(int i = 0; i < 9; i++) {

if(scores[i] > best) {

idx = i;

best = scores[i];

}

}

order[0] = idx;

visit[idx] = 1;

for(int i = 0; i < 9; i++) {

if(i == idx) continue;

dfs(i, 1);

}

System.out.println(max);

}

}

## [**[백준] 17070번 - 파이프 옮기기 1**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-17070%EB%B2%88-%ED%8C%8C%EC%9D%B4%ED%94%84-%EC%98%AE%EA%B8%B0%EA%B8%B0-1)

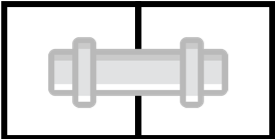
2020. 3. 27. 06:17

[17070](https://dheldh77.tistory.com/tag/17070), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [삼성A형](https://dheldh77.tistory.com/tag/%EC%82%BC%EC%84%B1A%ED%98%95), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

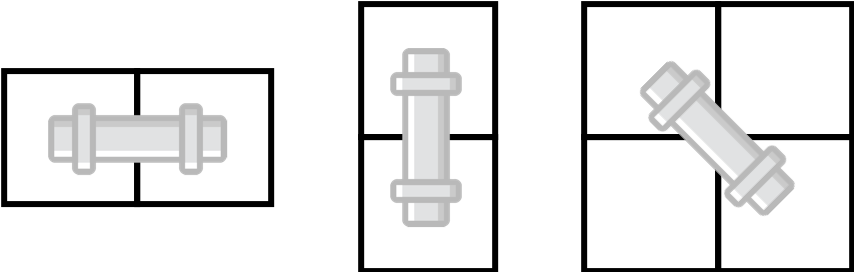
### 문제 >

유현이가 새 집으로 이사했다. 새 집의 크기는 N×N의 격자판으로 나타낼 수 있고, 1×1크기의 정사각형 칸으로 나누어져 있다. 각각의 칸은 (r, c)로 나타낼 수 있다. 여기서 r은 행의 번호, c는 열의 번호이고, 행과 열의 번호는 1부터 시작한다. 각각의 칸은 빈 칸이거나 벽이다.

오늘은 집 수리를 위해서 파이프 하나를 옮기려고 한다. 파이프는 아래와 같은 형태이고, 2개의 연속된 칸을 차지하는 크기이다.



파이프는 회전시킬 수 있으며, 아래와 같이 3가지 방향이 가능하다.

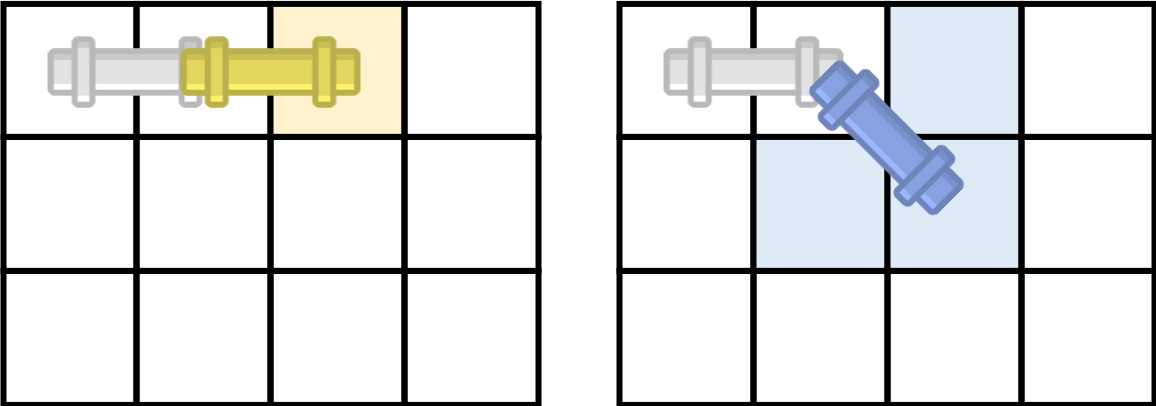


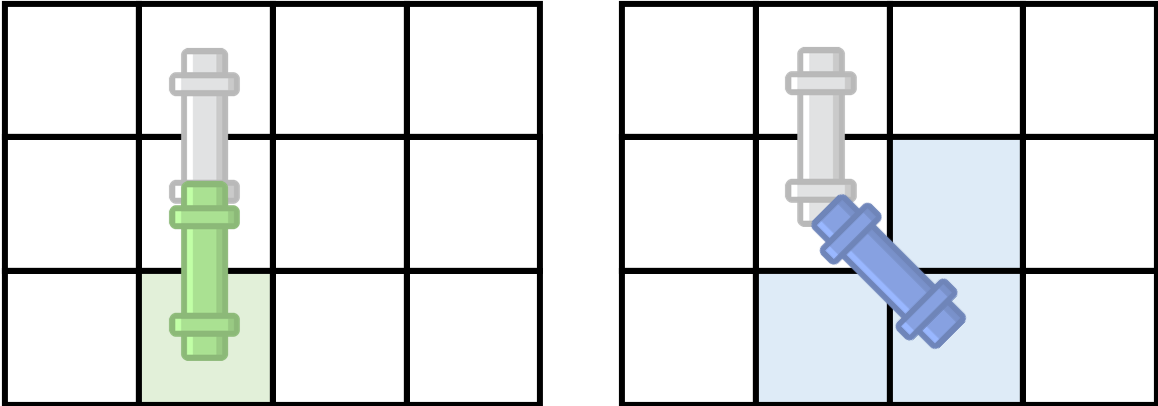
파이프는 매우 무겁기 때문에, 유현이는 파이프를 밀어서 이동시키려고 한다. 벽에는 새로운 벽지를 발랐기 때문에, 파이프가 벽을 긁으면 안 된다. 즉, 파이프는 항상 빈 칸만 차지해야 한다.

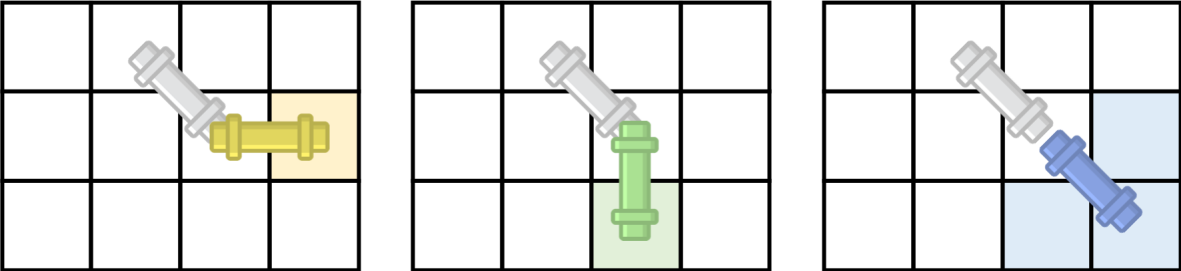
파이프를 밀 수 있는 방향은 총 3가지가 있으며, →, ↘, ↓ 방향이다. 파이프는 밀면서 회전시킬 수 있다. 회전은 45도만 회전시킬 수 있으며, 미는 방향은 오른쪽, 아래, 또는 오른쪽 아래 대각선 방향이어야 한다.

파이프가 가로로 놓여진 경우에 가능한 이동 방법은 총 2가지, 세로로 놓여진 경우에는 2가지, 대각선 방향으로 놓여진 경우에는 3가지가 있다.

아래 그림은 파이프가 놓여진 방향에 따라서 이동할 수 있는 방법을 모두 나타낸 것이고, 꼭 빈 칸이어야 하는 곳은 색으로 표시되어져 있다.

가로

세로

대각선

가장 처음에 파이프는 (1, 1)와 (1, 2)를 차지하고 있고, 방향은 가로이다. 파이프의 한쪽 끝을 (N, N)로 이동시키는 방법의 개수를 구해보자.

### 입력 >

첫째 줄에 집의 크기 N(3 ≤ N ≤ 16)이 주어진다. 둘째 줄부터 N개의 줄에는 집의 상태가 주어진다. 빈 칸은 0, 벽은 1로 주어진다. (1, 1)과 (1, 2)는 항상 빈 칸이다.

### 출력 >

첫째 줄에 파이프의 한쪽 끝을 (N, N)으로 이동시키는 방법의 수를 출력한다. 이동시킬 수 없는 경우에는 0을 출력한다. 방법의 수는 항상 1,000,000보다 작거나 같다.

### 해결방법 >

DFS(깊이우선탐색) 해결할 수 있는 문제이다.

파이프가 놓여있는 방향에 따라 이동할 수 있는 경우의 수가 달라진다.

1. 가로로 놓여있을 경우 :

가로 - 끝점을 기준으로 오른쪽 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

대각선 - 끝점을 기준으로 오른쪽, 아래쪽, 대각선 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

2. 세로로 놓여있을 경우 :

세로 - 끝점을 기준으로 아래쪽 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

대각선 - 끝점을 기준으로 오른쪽, 아래쪽, 대각선 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

3. 대각선으로 놓여있을 경우 :

가로 - 끝점을 기준으로 오른쪽 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

세로 - 끝점을 기준으로 아래쪽 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

대각선 - 끝점을 기준으로 오른쪽, 아래쪽, 대각선 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

각 방향에서 가능한 경우를 모든 탐색한다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_17070 {

static int[] dx = {0, 1, 1};

static int[] dy = {1, 0, 1};

static int cnt = 0;

static int n;

static int[][] map;

public static boolean possible(int x, int y, int d) {

int nx = x + dx[d];

int ny = y + dy[d];

if(d == 0 || d == 1) {

if(nx >= 0 && ny >= 0 && nx < n && ny < n && map[nx][ny] == 0) return true;

}

else {

if(nx >= 0 && ny >= 0 && nx < n && ny < n) {

if(map[nx - 1][ny] == 0 && map[nx][ny - 1] == 0 && map[nx][ny] == 0) return true;

}

}

return false;

}

public static void dfs(int x, int y, int d) {

int nx = x + dx[d];

int ny = y + dy[d];

if(nx == n-1 && ny == n-1) {

cnt++;

return;

}

if(d == 0) {

if(possible(nx, ny, 0)) dfs(nx, ny, 0);

if(possible(nx, ny, 2)) dfs(nx, ny, 2);

}

else if(d == 1) {

if(possible(nx, ny, 1)) dfs(nx, ny, 1);

if(possible(nx, ny, 2)) dfs(nx, ny, 2);

}

else {

if(possible(nx, ny, 0)) dfs(nx, ny, 0);

if(possible(nx, ny, 1)) dfs(nx, ny, 1);

if(possible(nx, ny, 2)) dfs(nx, ny, 2);

}

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

StringTokenizer tk = new StringTokenizer(br.readLine());

n = Integer.parseInt(tk.nextToken());

map = new int[n][n];

for(int i = 0; i < n; i++) {

tk = new StringTokenizer(br.readLine());

for(int j = 0; j < n; j++) {

map[i][j] = Integer.parseInt(tk.nextToken());

}

}

dfs(0, 0, 0);

System.out.println(cnt);

}

}

## [**[백준] 11559번 - Puyo Puyo**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-11559%EB%B2%88-Puyo-Puyo)

2020. 3. 26. 06:14

[11559](https://dheldh77.tistory.com/tag/11559), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [시뮬레이션](https://dheldh77.tistory.com/tag/%EC%8B%9C%EB%AE%AC%EB%A0%88%EC%9D%B4%EC%85%98), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

뿌요뿌요의 룰은 다음과 같다.

*필드에 여러 가지 색깔의 뿌요를 놓는다. 뿌요는 중력의 영향을 받아 아래에 바닥이나 다른 뿌요가 나올 때까지 아래로 떨어진다.*

*뿌요를 놓고 난 후, 같은 색 뿌요가 4개 이상 상하좌우로 연결되어 있으면 연결된 같은 색 뿌요들이 한꺼번에 없어진다.*

*뿌요들이 없어지고 나서 위에 다른 뿌요들이 있다면, 역시 중력의 영향을 받아 차례대로 아래로 떨어지게 된다.*

*아래로 떨어지고 나서 다시 같은 색의 뿌요들이 4개 이상 모이게 되면 또 터지게 되는데, 터진 후 뿌요들이 내려오고 다시 터짐을 반복할 때마다 1연쇄씩 늘어난다.*

*터질 수 있는 뿌요가 여러 그룹이 있다면 동시에 터져야 하고 여러 그룹이 터지더라도 한번의 연쇄가 추가된다.*

남규는 최근 뿌요뿌요 게임에 푹 빠졌다. 이 게임은 1:1로 붙는 대전게임이라 잘 쌓는 것도 중요하지만, 상대방이 터뜨린다면 연쇄가 몇 번이 될지 바로 파악할 수 있는 능력도 필요하다. 하지만 아직 실력이 부족하여 남규는 자기 필드에만 신경 쓰기 바쁘다. 상대방의 필드가 주어졌을 때, 연쇄가 몇 번 연속으로 일어날지 계산하여 남규를 도와주자!

### 입력 >

12\*6의 문자가 주어진다.

이때 .은 빈공간이고 .이 아닌것은 각각의 색깔의 뿌요를 나타낸다.

R은 빨강, G는 초록, B는 파랑, P는 보라, Y는 노랑이다.(모두 대문자로 주어진다.)

입력으로 주어지는 필드는 뿌요들이 전부 아래로 떨어진 뒤의 상태(즉 뿌요 아래에 빈 칸이 있는 경우는 없음) 이다.

### 출력 >

현재 주어진 상황에서 몇연쇄가 되는지 출력하라. (하나도 터지지 않는다면 0을 출력하면 된다.)

### 해결방법 >

DFS(깊이우선탐색) 또는 BFS(너비우선탐색)알고리즘을 활용해 해결할 수 있는 시뮬레이션 문제이다.

두 가지 과정이 필요하다.

1) 삭제

2) 이동

1)의 삭제의 경우는 문자가 같은 원소 4개 이상이 인접한 경우를 찾는 과정으로

BFS나 DFS를 활용해서 4개 이상 인접한 경우를 모두 찾고,

제거해준다.

2)의 이동의 경우는 삭제된 원소간 빈 공간을 없애주는 과정으로,

열을 기준으로 빈 공간이 아닌 원소 즉 '.'이 아닌 문자가 입력된 원소들을 스택이나 큐에 담고

맨아래 행부터 다시 쌓아준다.

비슷한 문제로 프렌즈4블록을 참고하면 좋을 듯하다.

<https://dheldh77.tistory.com/315>

[[프로그래머스] 프렌즈4블록](https://dheldh77.tistory.com/315" \t "_blank)

[문제설명 블라인드 공채를 통과한 신입 사원 라이언은 신규 게임 개발 업무를 맡게 되었다. 이번에 출시할 게임 제목은 프렌즈4블록. 같은 모양의 카카오프렌즈 블록이 2×2 형태로 4개가 붙어있을 경우 사..](https://dheldh77.tistory.com/315" \t "_blank)

[dheldh77.tistory.com](https://dheldh77.tistory.com/315" \t "_blank)

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_11559 {

static char[][] map;

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

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

static int depth = 0;

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void dfs(char c, int x, int y, int[][] visit, ArrayList<Node> list) {

visit[x][y] = 1;

list.add(new Node(x, y));

depth++;

for(int i = 0; i < 4; i++) {

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < 12 && ny < 6 && visit[nx][ny] == 0 && map[nx][ny] == c) {

dfs(c, nx, ny, visit, list);

}

}

}

public static int bomb() {

int cnt = 0;

for(int i = 0; i < 12; i++) {

for(int j = 0; j < 6; j++) {

if(map[i][j] != '.') {

int[][] visit = new int[12][6];

ArrayList <Node> list = new ArrayList<>();

depth = 0;

dfs(map[i][j], i, j, visit, list);

if(depth >= 4) {

for(int k = 0; k < list.size(); k++) {

Node node = list.get(k);

map[node.x][node.y] = '.';

}

cnt += list.size();

}

}

}

}

return cnt;

}

public static char[][] move() {

char[][] n\_map = new char[12][6];

for(int i = 0; i < 12; i++) {

for(int j = 0; j < 6; j++) {

n\_map[i][j] = '.';

}

}

for(int i = 0; i < 6; i++) {

Stack <Character> st = new Stack<>();

for(int j = 0; j < 12; j++) {

if(map[j][i] != '.') st.push(map[j][i]);

}

int row = 11;

while(!st.isEmpty()) {

n\_map[row][i] = st.pop();

row--;

}

}

return n\_map;

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

map = new char[12][6];

for(int i = 0; i < 12; i++) {

String s = br.readLine();

for(int j = 0; j < 6; j++) {

map[i][j] = s.charAt(j);

}

}

int cnt = 0;

while(true) {

int flag = bomb();

if(flag == 0) break;

cnt++;

map = move();

}

System.out.println(cnt);

}

}

## [**[백준] 6603번 - 로또**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-6603%EB%B2%88-%EB%A1%9C%EB%98%90)

2020. 3. 26. 06:04

[6603](https://dheldh77.tistory.com/tag/6603), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

독일 로또는 {1, 2, ..., 49}에서 수 6개를 고른다.

로또 번호를 선택하는데 사용되는 가장 유명한 전략은 49가지 수 중 k(k>6)개의 수를 골라 집합 S를 만든 다음 그 수만 가지고 번호를 선택하는 것이다.

예를 들어, k=8, S={1,2,3,5,8,13,21,34}인 경우 이 집합 S에서 수를 고를 수 있는 경우의 수는 총 28가지이다. ([1,2,3,5,8,13], [1,2,3,5,8,21], [1,2,3,5,8,34], [1,2,3,5,13,21], ..., [3,5,8,13,21,34])

집합 S와 k가 주어졌을 때, 수를 고르는 모든 방법을 구하는 프로그램을 작성하시오.

### 입력 >

입력은 여러 개의 테스트 케이스로 이루어져 있다. 각 테스트 케이스는 한 줄로 이루어져 있다. 첫 번째 수는 k (6 < k < 13)이고, 다음 k개 수는 집합 S에 포함되는 수이다. S의 원소는 오름차순으로 주어진다.

입력의 마지막 줄에는 0이 하나 주어진다.

### 출력 >

각 테스트 케이스마다 수를 고르는 모든 방법을 출력한다. 이때, 사전 순으로 출력한다.

각 테스트 케이스 사이에는 빈 줄을 하나 출력한다.

### 해결방법 >

n개의 숫자가 주어졌을 때, n개 중 6개의 조합 즉, C(n, 6)을 구하는 문제이다.

DFS(깊이우선탐색)으로 가능한 모든 조합을 찾는다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_6603 {

static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

static BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));

public static void dfs (int x, int[] nums, int depth, int[] visit) throws IOException{

visit[x] = 1;

if(depth == 6) {

for(int i = 0; i < visit.length; i++) {

if(visit[i] == 1) bw.write(nums[i] + " ");

}

bw.write("\n");

visit[x] = 0;

return;

}

for(int i = x; i < nums.length; i++) {

if(visit[i] == 0) dfs(i, nums, depth + 1, visit);

}

visit[x] = 0;

}

public static void main(String[] args) throws IOException {

StringTokenizer tk;

while(true) {

tk = new StringTokenizer(br.readLine());

int n = Integer.parseInt(tk.nextToken());

if(n == 0) break;

int[] nums = new int[n];

int[] visit = new int[n];

for(int i = 0; i < n; i++) {

nums[i] = Integer.parseInt(tk.nextToken());

}

for(int i = 0; i < n; i++) {

visit[i] = 1;

dfs(i, nums, 1, visit);

}

bw.write("\n");

}

bw.flush();

}

}

## [**[백준] 13460번 - 구슬 탈출 2**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-13460%EB%B2%88-%EA%B5%AC%EC%8A%AC-%ED%83%88%EC%B6%9C-2)

2020. 3. 19. 03:16

[13460](https://dheldh77.tistory.com/tag/13460), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [시뮬레이션](https://dheldh77.tistory.com/tag/%EC%8B%9C%EB%AE%AC%EB%A0%88%EC%9D%B4%EC%85%98), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

스타트링크에서 판매하는 어린이용 장난감 중에서 가장 인기가 많은 제품은 구슬 탈출이다. 구슬 탈출은 직사각형 보드에 빨간 구슬과 파란 구슬을 하나씩 넣은 다음, 빨간 구슬을 구멍을 통해 빼내는 게임이다.

보드의 세로 크기는 N, 가로 크기는 M이고, 편의상 1×1크기의 칸으로 나누어져 있다. 가장 바깥 행과 열은 모두 막혀져 있고, 보드에는 구멍이 하나 있다. 빨간 구슬과 파란 구슬의 크기는 보드에서 1×1크기의 칸을 가득 채우는 사이즈이고, 각각 하나씩 들어가 있다. 게임의 목표는 빨간 구슬을 구멍을 통해서 빼내는 것이다. 이때, 파란 구슬이 구멍에 들어가면 안 된다.

이때, 구슬을 손으로 건드릴 수는 없고, 중력을 이용해서 이리 저리 굴려야 한다. 왼쪽으로 기울이기, 오른쪽으로 기울이기, 위쪽으로 기울이기, 아래쪽으로 기울이기와 같은 네 가지 동작이 가능하다.

각각의 동작에서 공은 동시에 움직인다. 빨간 구슬이 구멍에 빠지면 성공이지만, 파란 구슬이 구멍에 빠지면 실패이다. 빨간 구슬과 파란 구슬이 동시에 구멍에 빠져도 실패이다. 빨간 구슬과 파란 구슬은 동시에 같은 칸에 있을 수 없다. 또, 빨간 구슬과 파란 구슬의 크기는 한 칸을 모두 차지한다. 기울이는 동작을 그만하는 것은 더 이상 구슬이 움직이지 않을 때 까지이다.

보드의 상태가 주어졌을 때, 최소 몇 번 만에 빨간 구슬을 구멍을 통해 빼낼 수 있는지 구하는 프로그램을 작성하시오.

### 입력 >

첫 번째 줄에는 보드의 세로, 가로 크기를 의미하는 두 정수 N, M (3 ≤ N, M ≤ 10)이 주어진다. 다음 N개의 줄에 보드의 모양을 나타내는 길이 M의 문자열이 주어진다. 이 문자열은 '.', '#', 'O', 'R', 'B' 로 이루어져 있다. '.'은 빈 칸을 의미하고, '#'은 공이 이동할 수 없는 장애물 또는 벽을 의미하며, 'O'는 구멍의 위치를 의미한다. 'R'은 빨간 구슬의 위치, 'B'는 파란 구슬의 위치이다.

입력되는 모든 보드의 가장자리에는 모두 '#'이 있다. 구멍의 개수는 한 개 이며, 빨간 구슬과 파란 구슬은 항상 1개가 주어진다.

### 출력 >

최소 몇 번 만에 빨간 구슬을 구멍을 통해 빼낼 수 있는지 출력한다. 만약, 10번 이하로 움직여서 빨간 구슬을 구멍을 통해 빼낼 수 없으면 -1을 출력한다.

### 해결방법 >

DFS(깊이우선탐색) 알고리즘으로 해결할 수 있는 시뮬레이션 문제이다.

문제를 해결하기 위해서 크게 세 가지를 생각해줘야 했다.

**1) 보드의 회전 방향**

  한 방향으로 두 구슬 중 하나라도 움직일 수 있다면 회전시켜줘고 DFS탐색을 시작한다.

  구슬이 움직일 수 있는 조건은 그 방향으로 벽이 없고, 다른 구슬이 존재하지 않아야 한다.

  또 한 가지, 코드 실행 속도를 위해서 바로 이전에 이동했던 방향의 역방향으로 다시 이동하지 않는다.

**2) 두 구슬의 움직임**

  구슬은 각각 독립적으로 움직이면서 서로의 영향을 받는다.

  즉 구슬 주변 상황에 따라서 움직이는 길이는 달라지지만,

  서로 인접한 위치에 있다면 앞에 있는 구슬이 움직이기 전까지 뒤에 있는 구슬이 움직일 수 없다.

  처음에는 고민을 많이 했다.

  만약 왼쪽으로 움직여야하는데 구슬이 파란색, 빨간색 순서로 붙어 있다면 파란색 부터 움직여줘야하고,

  반대라면 또 빨간색부터 옮겨줘야하고 그럼 그때마다 조건문을 달아줘야하는 건가...

  깊게 생각할 필요가 없었다.

  그냥 두 구슬이 더 이상 이동할 방향이 없을 때까지 계속해서 이동시켜주며면되는 문제였다.

  다른 구슬 때문에 막혀있더라고 다른 구슬이 움직이고 나면 이동할 자리가 생긴다.

**3) 탐색의 종료조건**

  처음에는 생각했던 조건은 세 가지였다.

  1) 10번 이상 회전을 시켰을 때 -> 실패 조건

  2) 파란색이 빠져나왔을 때 -> 실패 조건

  3) 빨간색만 빠져나왔을 때 -> 성공 조건

  이렇게 단순하게 생각한게 문제였다.

  예제에 없는 반례로, 빨간색이 먼저 빠져나오고 난 뒤에 파란색 구슬이 계속해서 이동해 빠져나오는 경우이다.

  빨간색 구슬이 빠져나왔다면 빨간색 구슬을 제거해주고 파란색 구슬을 계속해서 이동시킨 뒤

  파란색 구슬도 빠져나오는지 확인을 해야한다.

  이러한 반례나 공의 이동방식 등 여러가지로 도움이 되는 문제였다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_13460\_2 {

static int n;

static int m;

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

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

static int min = -1;

static int tx;

static int ty;

public static boolean possible(int x, int y, int d, char[][] map) {

int nx = x + dx[d];

int ny = y + dy[d];

if(nx > 0 && ny > 0 && nx < n - 1 && ny < m - 1 && (map[nx][ny] == '.' || map[nx][ny] == 'O')) return true;

else return false;

}

public static void dfs(int rx, int ry, int bx, int by, int d, char[][] map, int cnt) {

if(cnt > 10) return;

char[][] n\_map = new char[n][m];

for(int i = 0; i < n; i++) {

for(int j = 0; j < m; j++) {

n\_map[i][j] = map[i][j];

}

}

int nrx = rx, nry = ry;

int nbx = bx, nby = by;

int[] flag = new int[2];

if(rx == tx && ry == ty) flag[0] = 1;

if(bx == tx && by == ty) flag[1] = 1;

while((flag[0] == 0 && possible(rx, ry, d, n\_map)) || possible(bx, by, d, n\_map)) {

if(flag[0] == 0 && possible(rx, ry, d, n\_map)) {

nrx = rx + dx[d];

nry = ry + dy[d];

n\_map[rx][ry] = '.';

n\_map[nrx][nry] = 'R';

rx = nrx;

ry = nry;

if(rx == tx && ry == ty) {

flag[0] = 1;

n\_map[rx][ry] = 'O';

}

}

if(possible(bx, by, d, n\_map)) {

nbx = bx + dx[d];

nby = by + dy[d];

n\_map[bx][by] = '.';

n\_map[nbx][nby] = 'B';

bx = nbx;

by = nby;

if(bx == tx && by == ty) {

flag[1] = 1;

return;

}

}

}

if(flag[0] == 1) {

if(min == -1) min = cnt;

else min = Math.min(cnt, min);

return;

}

for(int i = 0; i < 4; i++) {

if(d == 0 && i == 2) continue;

if(d == 1 && i == 3) continue;

if(d == 2 && i == 4) continue;

if(d == 3 && i == 1) continue;

if(possible(nrx, nry, i, n\_map) || possible(nbx, nby, i, n\_map)) {

dfs(nrx, nry, nbx, nby, i, n\_map, cnt + 1);

}

}

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

StringTokenizer tk = new StringTokenizer(br.readLine());

n = Integer.parseInt(tk.nextToken());

m = Integer.parseInt(tk.nextToken());

char[][] map = new char[n][m];

int rx = 0, ry = 0;

int bx = 0, by = 0;

for(int i = 0; i < n; i++) {

tk = new StringTokenizer(br.readLine());

String s = tk.nextToken();

for(int j = 0; j < m; j++) {

map[i][j] = s.charAt(j);

if(map[i][j] == 'R') {

rx = i;

ry = j;

}

if(map[i][j] == 'B') {

bx = i;

by = j;

}

if(map[i][j] == 'O') {

tx = i;

ty = j;

}

}

}

for(int i = 0; i < 4; i++) {

if(possible(rx, ry, i, map) || possible(bx, by, i, map)) {

dfs(rx, ry, bx, by, i, map, 1);

}

}

System.out.println(min);

}

}

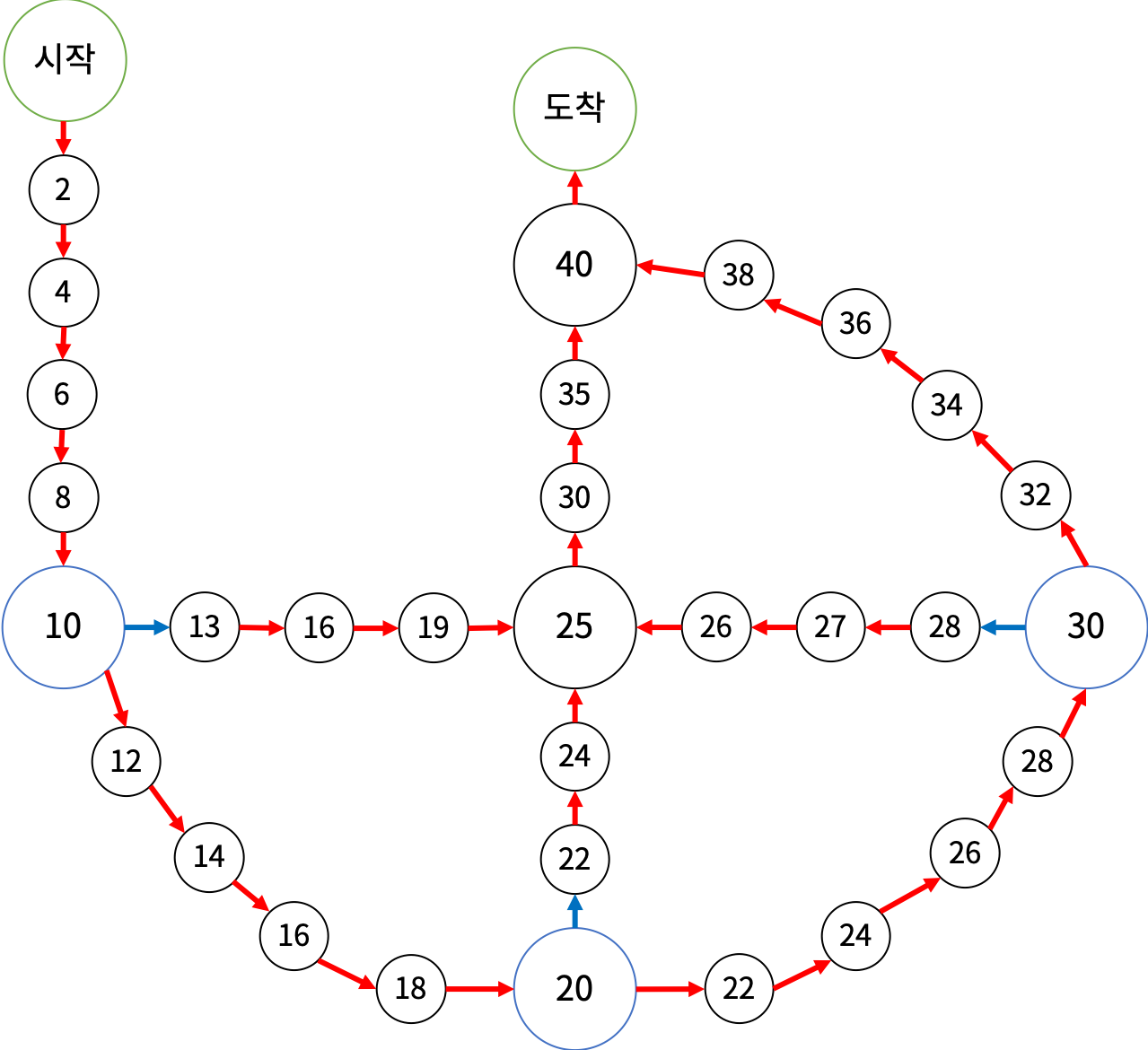
## [**[백준] 17825번 - 주사위 윷놀이**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-17825%EB%B2%88-%EC%A3%BC%EC%82%AC%EC%9C%84-%EC%9C%B7%EB%86%80%EC%9D%B4)

2020. 3. 18. 02:35

[17825](https://dheldh77.tistory.com/tag/17825), [Backtracking](https://dheldh77.tistory.com/tag/Backtracking), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

주사위 윷놀이는 다음과 같은 게임판에서 하는 게임이다.



* 처음에는 시작 칸에 말 4개가 있다.
* 말은 게임판에 그려진 화살표의 방향대로만 이동할 수 있다. 말이 파란색 칸에서 이동을 시작하면 파란색 화살표를 타야 하고, 이동하는 도중이거나 파란색이 아닌 칸에서 이동을 시작하면 빨간색 화살표를 타야 한다. 말이 도착 칸으로 이동하면 주사위에 나온 수와 관계 없이 이동을 마친다.
* 게임은 10개의 턴으로 이루어진다. 매 턴마다 1부터 5까지 한 면에 하나씩 적혀있는 5면체 주사위를 굴리고, 도착 칸에 있지 않은 말을 하나 골라 주사위에 나온 수만큼 이동시킨다.
* 말이 이동을 마치는 칸에 다른 말이 있으면 그 말은 고를 수 없다. 단, 이동을 마치는 칸이 도착 칸이면 고를 수 있다.
* 말이 이동을 마칠 때마다 칸에 적혀있는 수가 점수에 추가된다.

주사위에서 나올 수 10개를 미리 알고 있을 때, 얻을 수 있는 점수의 최댓값을 구해보자.

### 입력 >

첫째 줄에 주사위에서 나올 수 10개가 순서대로 주어진다.

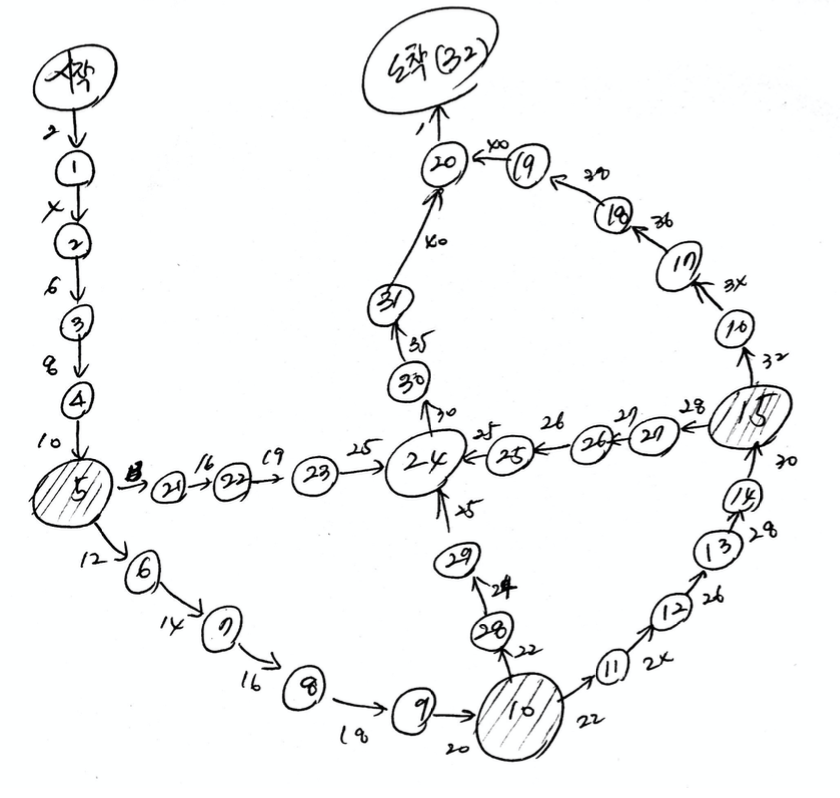
### 출력 >

얻을 수 있는 점수의 최댓값을 출력한다.

### 해결방법 >

DFS(깊이우선탐색)알고리즘으로 문제를 해결했다.

일단은 문제에 주어진 판을 아래와 같이 다시 만들었다.



각 노드마다 번호를 매기고, 문제에서 주어진 숫자는 간선의 가중치로 생각하는 방법이다.

따라서 33 X 33 크기의 인접행렬을 만들고 간선의 가중치를 저장했다.

지도 정보를 만들었다면 이제 최대값을 찾아야하는데 두 가지 과정이 필요했다.

1) 깊이 우선 탐색을 통해 말이 움직이는 순서의 모든 경우의 수를 찾는다.

2) 각 경우마다 깊이 우선 탐색을 사용해서 주사위의 숫자만큼 이동한다.

각 과정을 자세히 살펴보면

**1) 깊이 우선 탐색을 통해 말이 움직이는 순서의 모든 경우의 수를 찾는다.**

1)번 과정 같은 경우는 주사위 숫자 10개에 따라 4개의 말을 선택할 수 있기 때문에 중복을 허용하는 순열과 같다.

경우의 수는 4^10으로 연산 수가 크다.

2)번 과정에서도 깊이우선탐색을 해야하는데 접근 방식이 잘못되었는가 생각을 많이 했지만 백트레킹으로 해결할 수 있었다.

**2) 각 경우마다 깊이 우선 탐색을 사용해서 주사위의 숫자만큼 이동한다.**

2)번 과정은 선택된 말로 i번째 주사위 수만큼 움직이는 것이다. 다시말해 주사위 수만큼 인접한 노드로 깊이있게 방문한다.

이 부분을 깊이우선탐색으로 해결한 이유 인접한 노드가 순서대로 있지않고,

5, 10, 15번째 노드, 즉 경로가 두 갈래로 나뉘는 노드가 있기  때문이다.

노드가 두 갈래로 나뉜다는 말은 간선의 정보가 두 개가 존재한다는 것인데, 두 가지 조건으로 해결할 수 있다

a) 탐색을 시작한 노드가 두 갈래로 나뉘는 노드라면, 노드 값이 큰 노드를 방문한다.

게임판에 대한 정보를 새로 만들 때, 두 갈래에서 우선으로 방문해야 하는 노드의 원소 값을 크게 만들었기 때문에 가능하다.

b) 탐색 중간에 방문한 노드가 두 갈래로 나뉘는 노드라면, 노드 값이 작은 노드를 방문한다.

지금까지 말한 부분만 구현한다면 연산 시간이 오래 걸린다. 따라서 1)번 과정의 백트레킹할 조건을 만들어준다. 조건은 아래와 같다.

a) 주사위 수만큼 이동했을 때, 다른 말이 존재한다면 경우(문제에서 요구한 조건)

b) 현재까지 이동한 간선의 합이 최대값이 될 가능성이 없을 경우

c) 도착지점에 도착한 말을 움직이려하는 경우

이 문제는 조금 억지로 풀었다는 느낌을 많이 받았다. 코드를 작성하는 내내 DFS를 중첩해서 사용하니까 연산시간에 대한

압박감도 많이 받았고, 이렇게 구현하는게 맞는가하는 생각도 많이 들었다.

처음 제출했을 때 연산 시간이 너무 커서 백트레킹 조건도 덕지덕지 붙여넣은 느낌이다.

다시 생각해봐야할 문제인 것 같다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_17825\_2 {

static int[] dice = new int[10];

static int[][] map = new int[33][33];

static int next = 0;

static int weight = 0;

static int max = 0;

public static class Node implements Comparable<Node>{

int v;

int w;

Node(int v, int w) {

this.v = v;

this.w = w;

}

@Override

public int compareTo(Node n) {

if(this.v < n.v) return 1;

else return -1;

}

}

public static void set() {

for(int i = 0; i < 20; i++) {

map[i][i + 1] = 2 \* (i + 1);

}

map[5][21] = 13; map[21][22] = 16; map[22][23] = 19; map[23][24] = 25;

map[15][27] = 28; map[27][26] = 27; map[26][25] = 26; map[25][24] = 25;

map[10][28] = 22; map[28][29] = 24; map[29][24] = 25;

map[24][30] = 30; map[30][31] = 35; map[31][20] = 40;

map[20][32] = 1;

}

public static void move(int cnt, int start) {

if(cnt == 0) return;

if(next == 32) {

weight = 1;

return;

}

ArrayList<Node> l = new ArrayList<>();

for(int i = 0; i < 33; i++) {

if(map[next][i] != 0) {

l.add(new Node(i, map[next][i]));

}

}

if(l.size() == 1) {

next = l.get(0).v;

weight = l.get(0).w;

}

else if(l.size() > 1) {

Collections.sort(l);

if(next == start) {

next = l.get(0).v;

weight = l.get(0).w;

}

else {

next = l.get(1).v;

weight = l.get(1).w;

}

}

move(cnt-1, start);

}

public static void select(int h, int[] ph, int x, int sum) {

if(x == 10) {

for(int i = 0; i < 4; i++) {

if(ph[i] == 32) sum--;

}

max = Math.max(max, sum);

return;

}

if(ph[h] == 32) return;

int[] horse = ph.clone();

next = horse[h];

move(dice[x], next);

for(int i = 0; i < 4; i++) {

if(next != 32 && horse[i] == next) return;

}

horse[h] = next;

sum += weight;

if(sum + (9 - x) \* 40 < max) return;

for(int i = 0; i < 4; i++) {

select(i, horse, x + 1, sum);

}

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

StringTokenizer tk = new StringTokenizer(br.readLine());

for(int i = 0; i < 10; i++) {

dice[i] = Integer.parseInt(tk.nextToken());

}

set();

for(int i = 0; i < 4; i++) {

int[] horse = new int[4];

select(i, horse, 0, 0);

}

System.out.println(max);

}

}