

JooHyun – Lee (comkiwer)

TS테트리스

Hancom Education Co. Ltd.

Problem

TS테트리스는 두 명의 플레이어가
폭이 W 이고 높이가 H 인 2차원 수직 공간의 게임판에서 진행되는 게임이다.

플레이어 1과 플레이어 2가 한 턱씩 번갈아 가며
자신의 블록을 떨어뜨리거나 상대의 블록을 변경하여

자신의 블록을 많이 없애는 플레이어가 이기는 게임이다.

게임은 항상 플레이어 1부터 시작하며, 매 턴마다 진행되는 게임 방식은 다음과 같다.

- ① 플레이어는 아래 두가지 방법 중 하나를 선택하여 해당 액션을 한다.
 - ⓐ 최상단 특정 위치에서 자신의 블록 3개를 가로로 연결하여 아래로 떨어뜨린다. ([그림. 1]의 A참조)
 - ⓑ 최하단 특정 위치의 블록로부터 시작하여 상하좌우로 연결되어 있는 상대의 블록들을 내 블록으로 바꾼다. ([그림. 2]의 B 참조)
- ② 게임판에서 동일 플레이어의 블록이 가로, 세로, 대각 방향 중 한 방향으로 5개 이상 연결되면 해당 블록들은 동시에 사라진다.
- ③ 사라진 블록 위의 블록들은 비어 있는 공간만큼 아래로 떨어진다.
- ④ 동일한 플레이어의 블록이 5개 이상 연결되어 사라진 블록이 없을 때까지 ② ~ ④를 반복 수행하고, 더 이상 사라질 블록이 없으면 상대 플레이어의 턴으로 넘어간다.

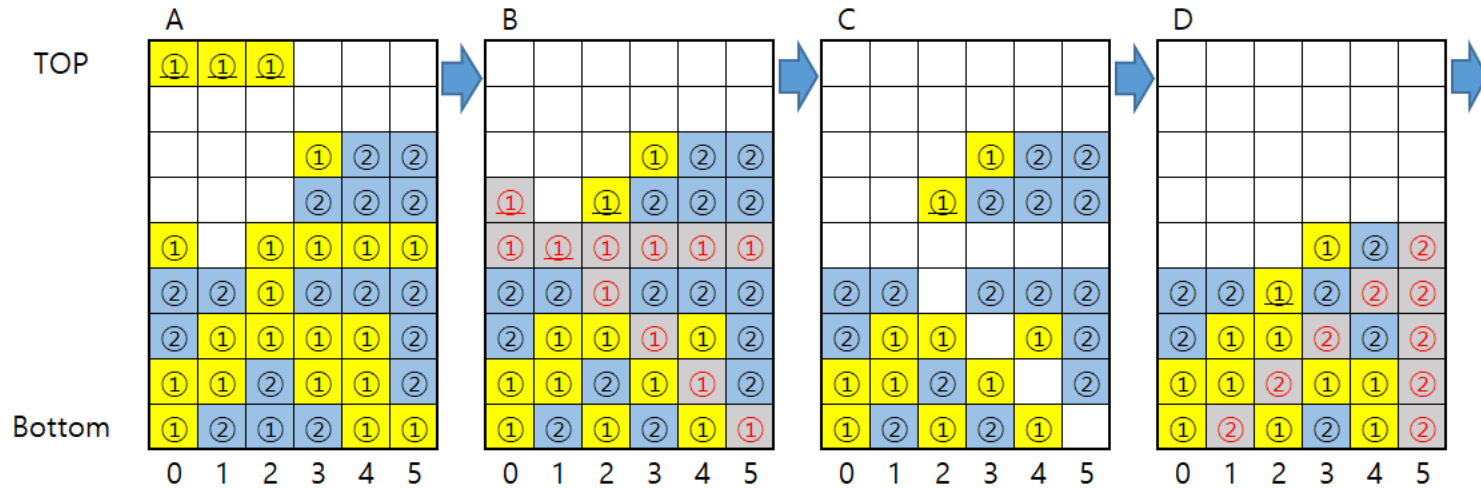
이때, 각 플레이어는 자신의 턴에서 없앤 자신의 블록 개수만큼 점수를 얻게 된다.

플레이어 1의 턴에서 없앤 플레이어 2의 블록은 점수에 포함되지 않고
플레이어 2의 턴에서 없앤 플레이어 1의 블록은 점수에 포함되지 않음을 유의하라.

다음 그림 1은 방법 ㉠로 게임을 진행한 예이다.

Problem

TS테트리스

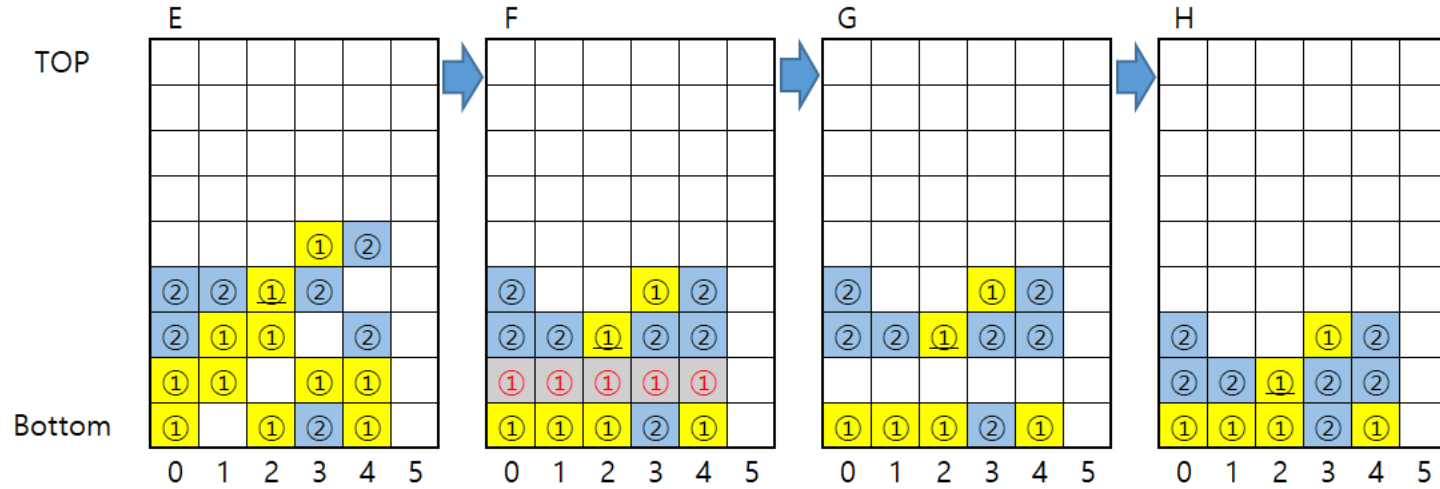


플레이어 1이 최상단의 열 0에서부터 가로로 3 개의 블록을 떨어뜨렸다. ([그림. 1]의 A를 참조)
그 결과로 플레이어 1의 블록이 가로로 6개, 대각으로 6개 연결되어 사라져서 11점을 얻었다.
여기서 한 개의 블록은 중복이다. ([그림. 1]의 B를 참조)

[그림. 1]의 D에서 사라진 플레이어 2의 블록 9개는 점수에 포함되지 않는다.

Problem

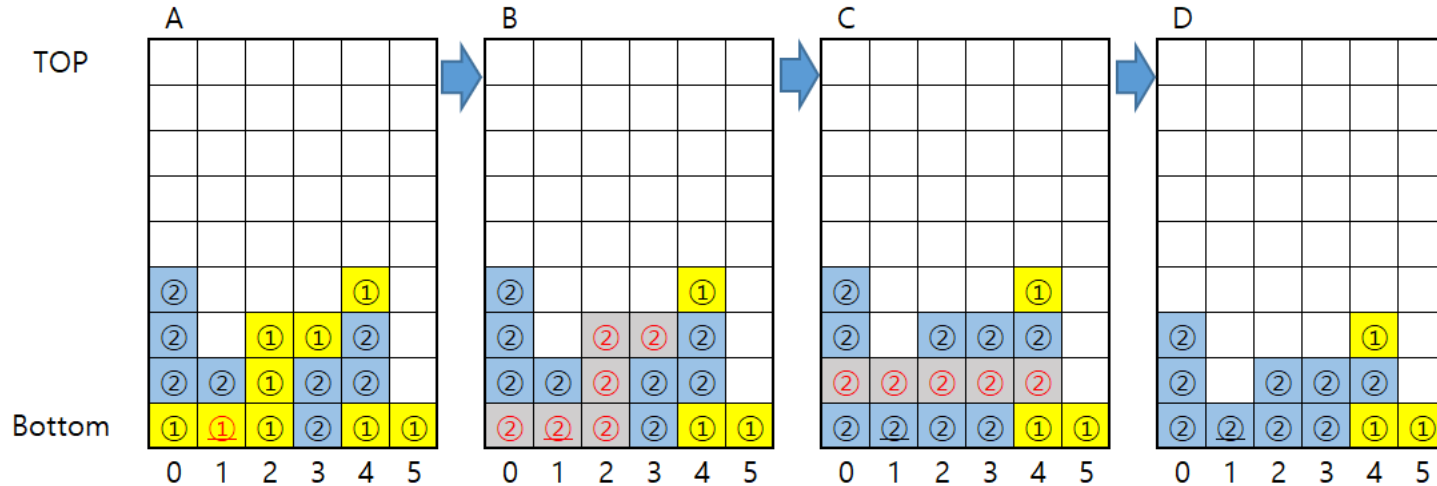
TS테트리스



또 다시 가로로 5개 연결되어 사라져서 5점을 얻었다. ([그림. 1]의 F를 참조)

한 턴에 총 16점을 얻었다.

아래 [그림. 2]는 방법 ㉑로 게임을 진행한 예이다.



플레이어 2가 열 1의 최하단에 있는 플레이어 1의 블록을 공격하였다. ([그림. 2]의 A를 참조)

공격 결과로 열 1의 맨 아래의 있는 플레이어 1의 블록으로부터 시작하여

가로와 세로로 연결된 플레이어 1의 블록들은 플레이어 2의 블록으로 변경되었다.

그리고, 플레이어 2의 블록이 가로로 5개 연결되어 사라지고 플레이어 2는 5점을 얻었다.

([그림. 2]의 C를 참조)

상기와 같은 방식으로 TS테트리스 게임을 진행할 때,
플레이어 1과 플레이어 2가 획득한 점수와 승자를 구하는 프로그램을 작성하라.

※ 아래 API 설명을 참조하여 각 함수를 구현하라.

1. `void init(int W, int H)` :

테스트 케이스에 대한 초기화하는 함수. 각 테스트 케이스의 처음에 1회 호출된다.

W와 H는 각각 게임판의 폭과 높이이다.

게임판에는 초기에 어떤 블록도 없다.

게임판에서 맨 왼쪽 열은 열 0이고 맨 오른쪽 열은 열 " $W - 1$ "이다.

Parameters

W : 게임판의 폭 ($10 \leq W \leq 200$)

H : 게임판의 높이 ($10 \leq H \leq 200$)

2. `int` dropBlocks(`int` player, `int` col) :

player인 플레이어가 게임판의 최상단의 col 열에서부터
가로로 연결된 3개의 블록을 떨어뜨린다.

해당 턴에서 플레이어가 얻은 점수를 반환한다.

예로, player = 2, col = 3 이라고 하면 플레이어 2가

최상단 열 3, 4, 5인 세 개의 칸으로부터 플레이어 2의 블록을 각각 떨어뜨린다.

주어진 테스트 케이스에서 블록을 떨어뜨리고자 하는 해당 칸에
아무 블록이 없음을 보장한다.

Parameters

player : 게임을 진행하는 플레이어 번호 ($1 \leq \text{player} \leq 2$)

col : 상단에서 떨어뜨릴 3개의 블록 중 왼쪽 블록의 위치 ($0 \leq \text{col} \leq W - 3$)

Returns

해당 턴에서 얻은 점수

3. `int` changeBlocks(`int` player, `int` col) :

`player`인 플레이어가 게임판의 최하단 `col` 열에 있는 블록과 그 블록로부터 시작하여
상하좌우로 연결된 상대 플레이어의 블록을 해당 플레이어의 블록로 변환시킨다.

해당 턴에서 얻은 점수를 반환한다.

대각으로만 연결된 상대 플레이어의 블록을 변환시킬 수 없다.

만약 최하단 `col` 열에 있는 블록이 자신의 블록이거나 블록이 없는 경우
아무런 변화도 일어나지 않는다.

Parameters

`player` : 게임을 진행하는 플레이어 번호 ($1 \leq \text{player} \leq 2$)

`col` : 변환시킬 상대 플레이어 블록의 위치 ($0 \leq \text{col} \leq W - 1$)

Returns

해당 턴에서 얻은 점수

4. `int getResult(int blockCnt[2])` :

지금까지 각 플레이어가 얻은 점수를 계산하여 게임 결과를 반환하고
게임판에 남아 있는 각 플레이어의 블록의 개수를 `blockCnt`에 저장한다.
게임 결과는 다음과 같다.

- 플레이어 1이 플레이어 2 보다 높은 점수를 얻은 경우 1을 반환한다.
- 플레이어 2가 플레이어 1 보다 높은 점수를 얻은 경우 2를 반환한다.
- 두 플레이어의 점수가 같은 경우 0을 반환한다.

게임판에 남아 있는 플레이어 1의 블록의 개수는 `blockCnt[0]`에 저장한다.

게임판에 남아 있는 플레이어 2의 블록의 개수는 `blockCnt[1]`에 저장한다.

Parameters

`blockCnt` : 각 플레이어의 남아 있는 블록의 개수를 저장해야 할 배열

Returns

게임 결과

[제약사항]

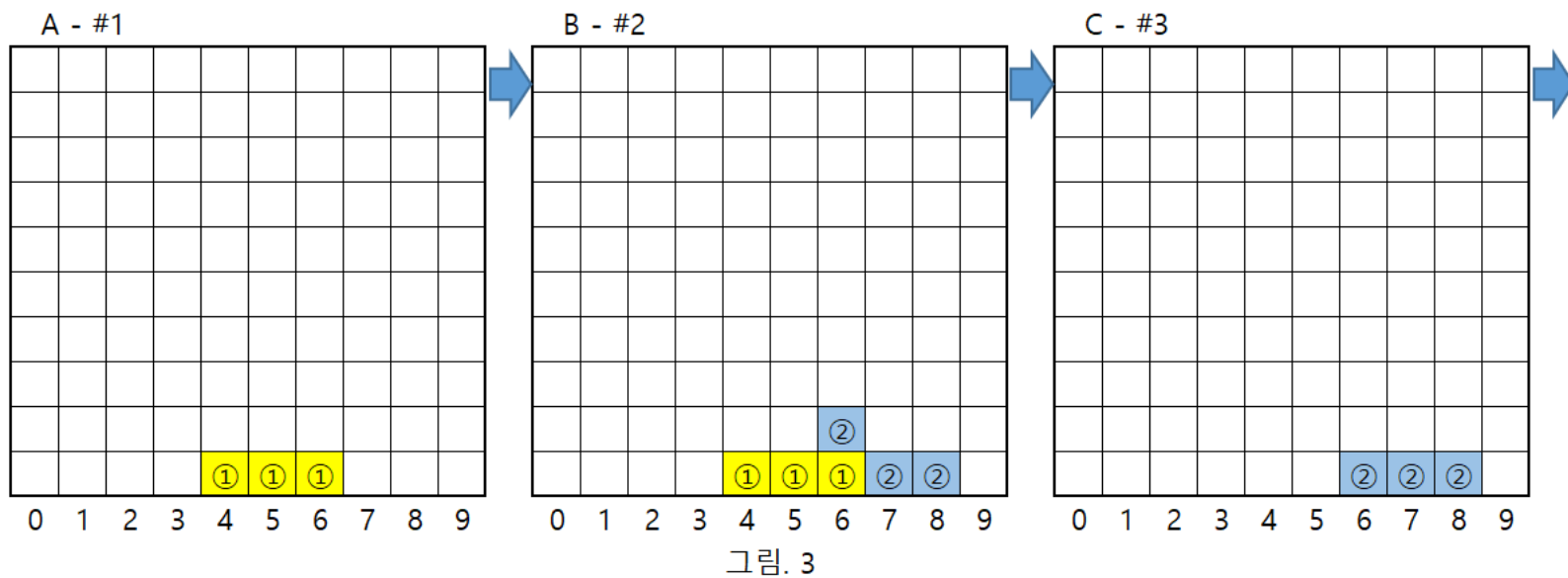
1. 각 테스트 케이스 시작 시 `init()` 함수가 한 번 호출된다.
2. 게임판의 너비 `W`는 10 이상 200 이하의 정수이다.
3. 게임판의 높이 `H`는 10 이상 200 이하의 정수이다.
4. `dropBlocks()` 함수가 호출 될 때 돌을 떨어트리는 위치에 돌이 없음을 보장한다.
5. `dropBlocks()`와 `changeBlocks()` 함수의 `mPlayer` 값은 1과 2가 서로 반복된다.
시작은 1부터 한다.
6. 각 테스트 케이스에서 `dropBlocks()` 함수가 최대 10,000회 호출된다.
7. 각 테스트 케이스에서 `changeBlocks()`와 `getResult()` 함수는
각각 최대 1,000회 호출된다.

Problem analysis

Problem analysis : 예제

TS테트리스

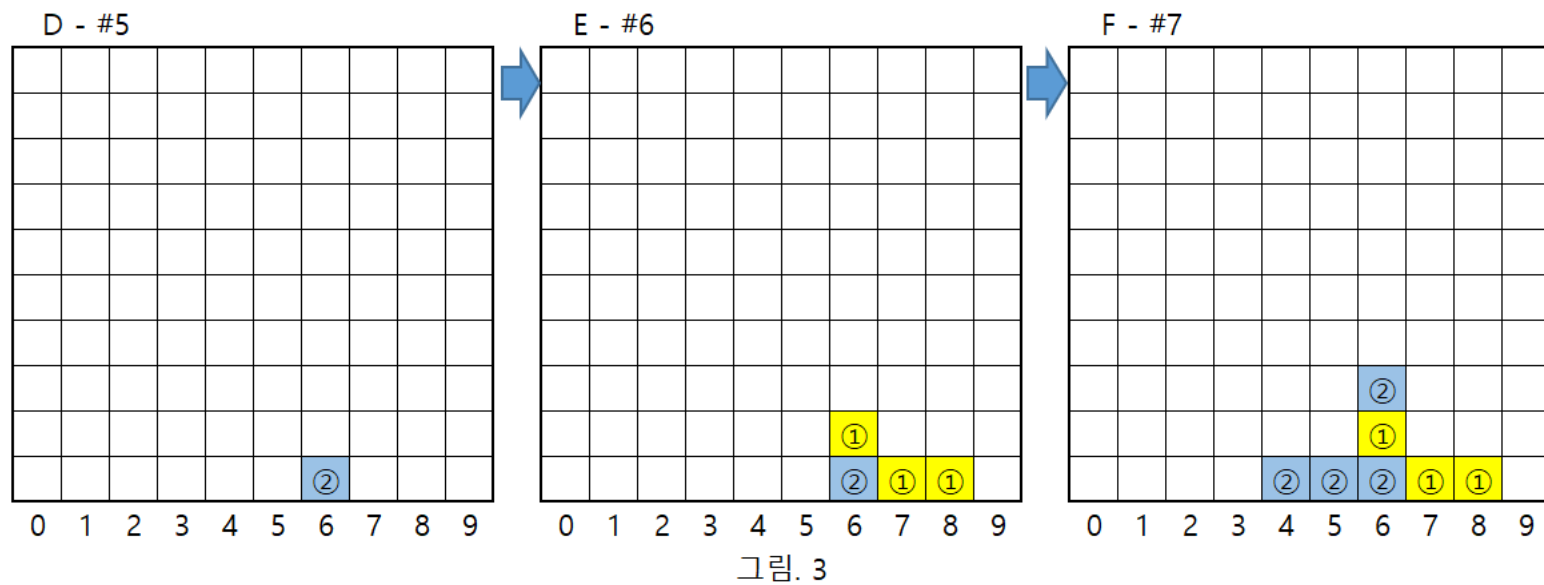
순서	함수	반환값	설명	그림
0	init(10, 10)		W = 10, H = 10	
1	dropBlocks(1, 4)	0		[그림. 3-A]
2	dropBlocks(2, 6)	0		[그림. 3-B]
3	dropBlocks(1, 1)	6	플레이어 1이 6점을 얻는다.	[그림. 3-C]
4	getResult()	1 mStoneCnt[] = {0, 3}	플레이어 1, 2가 얻은 점수는 각각 6점과 0점이다.	



Problem analysis : 예제

TS테트리스

순서	함수	반환값	설명	그림
5	dropBlocks(2, 4)	5	플레이어 2가 5점을 얻는다.	[그림. 3-D]
6	dropBlocks(1, 6)	0		[그림. 3-E]
7	dropBlocks(2, 4)	0		[그림. 3-F]



Problem analysis : 예제

TS테트리스

순서	함수	반환값	설명	그림
8	dropBlocks(1, 7)	0		[그림. 4-A]
9	changeBlocks(2, 0)	0	해당 칸에 돌이 없으므로 변화가 없다.	[그림. 4-B]
10	dropBlocks(1, 3)	5	플레이어 1이 5점을 얻는다.	[그림. 4-C]

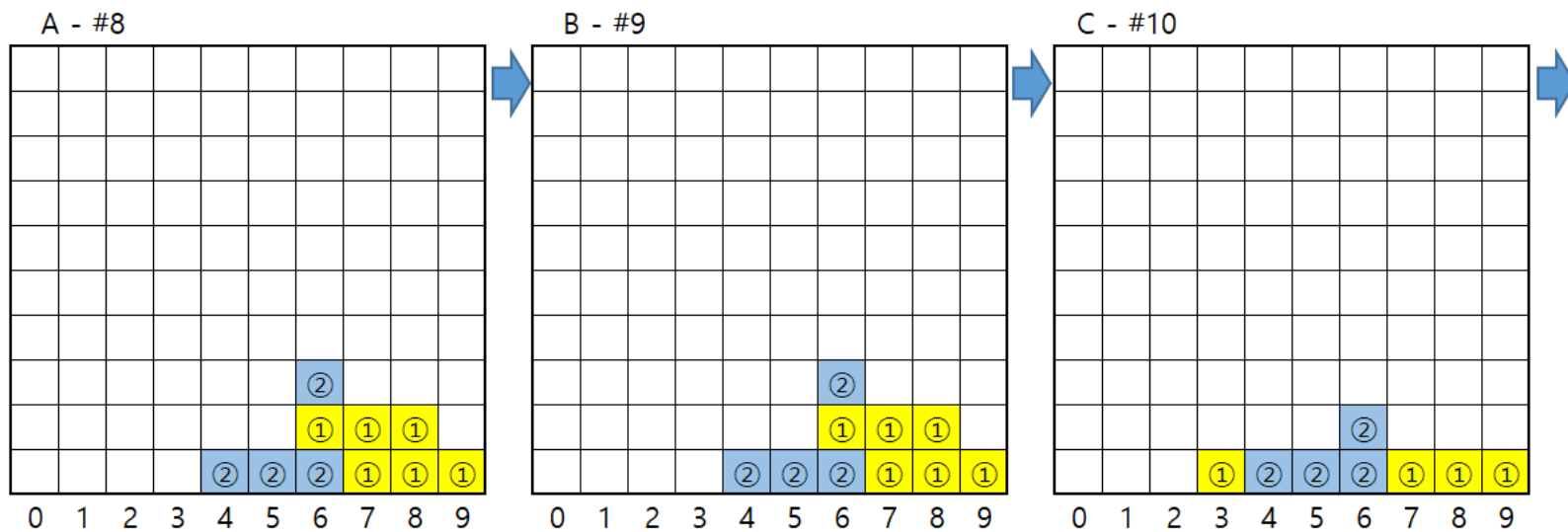
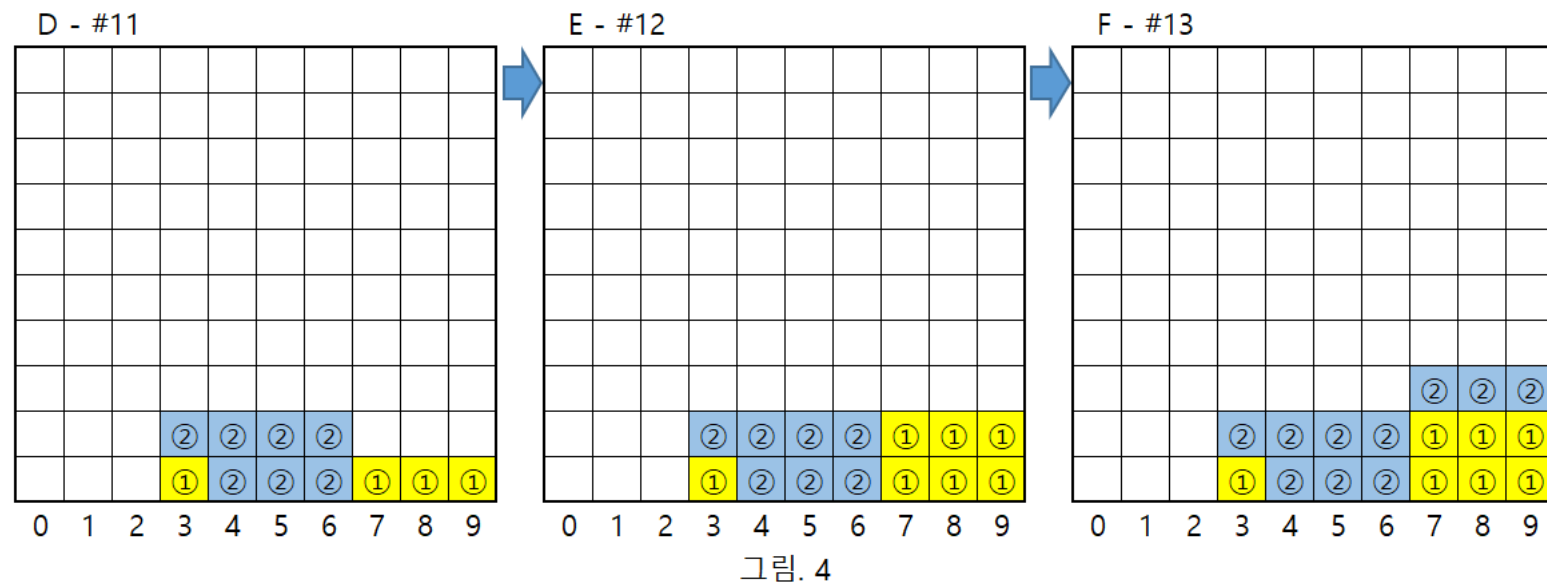


그림. 4

Problem analysis : 예제

TS테트리스

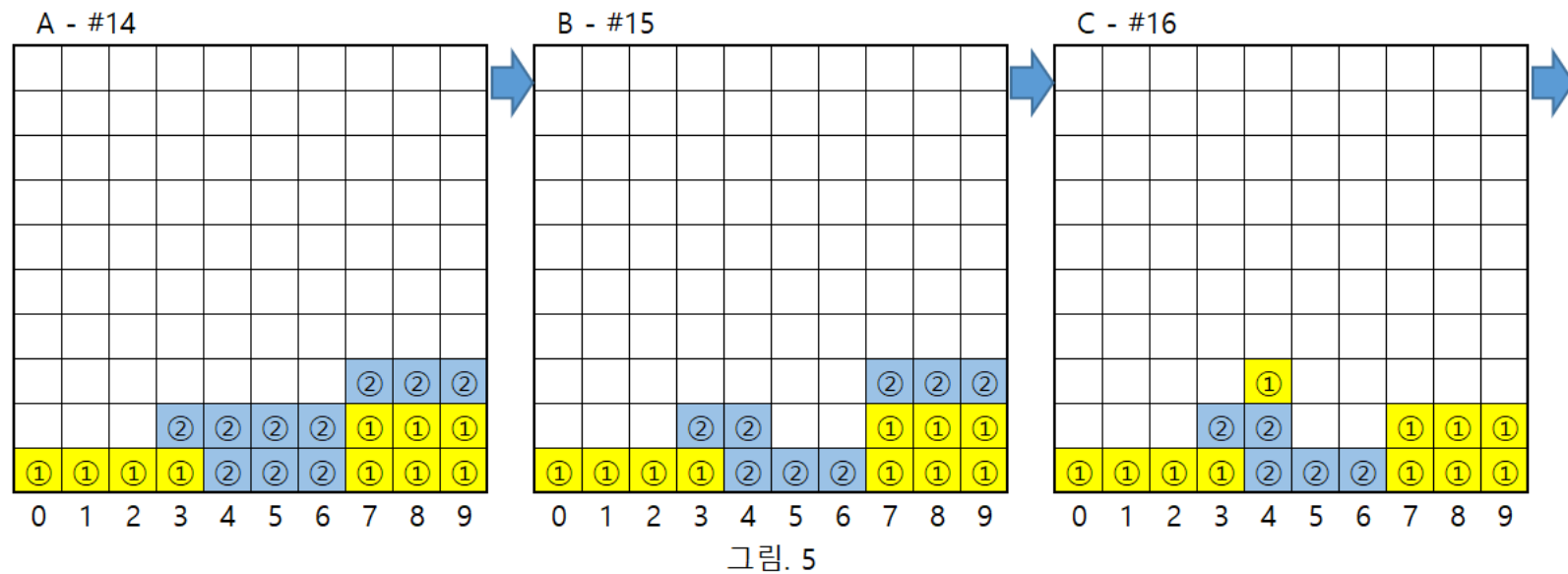
순서	함수	반환값	설명	그림
11	dropBlocks(2, 3)	0		[그림. 4-D]
12	dropBlocks(1, 7)	0		[그림. 4-E]
13	dropBlocks(2, 7)	0		[그림. 4-F]



Problem analysis : 예제

TS테트리스

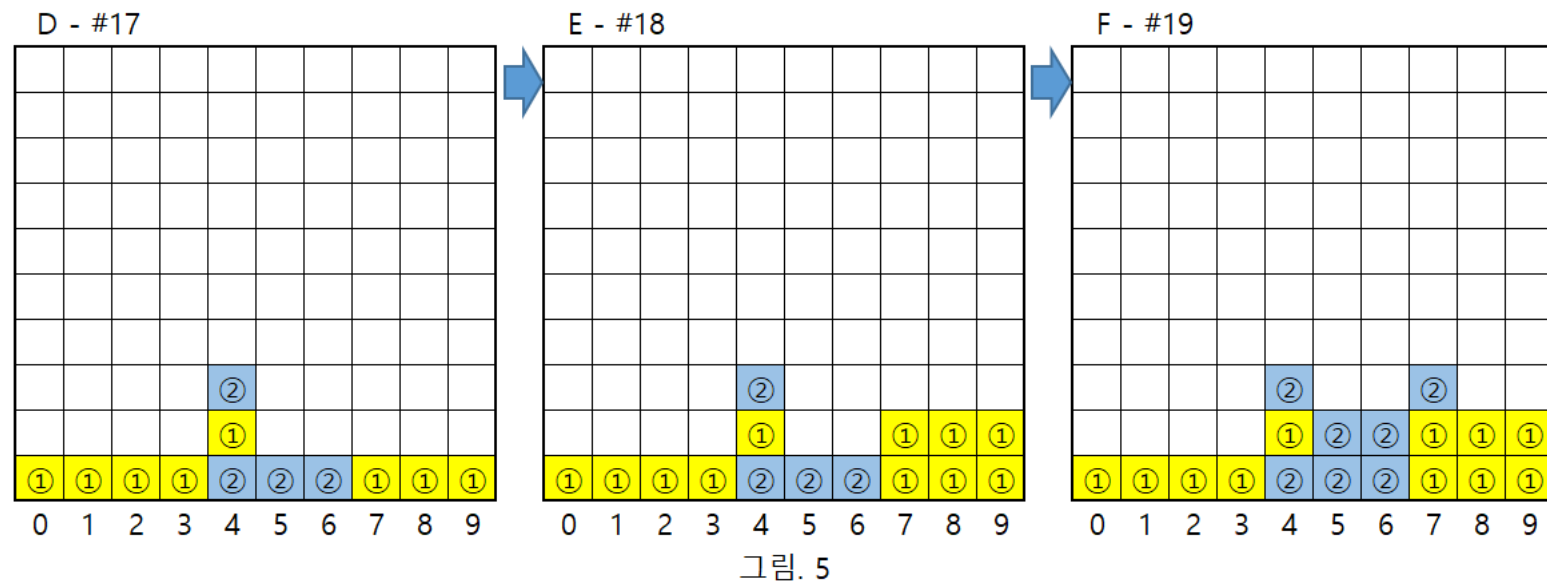
순서	함수	반환값	설명	그림
14	dropBlocks(1, 0)	0		[그림. 5-A]
15	dropBlocks(2, 2)	5	플레이어 2가 5점을 얻는다.	[그림. 5-B]
16	dropBlocks(1, 4)	5	플레이어 1이 5점을 얻는다.	[그림. 5-C]



Problem analysis : 예제

TS테트리스

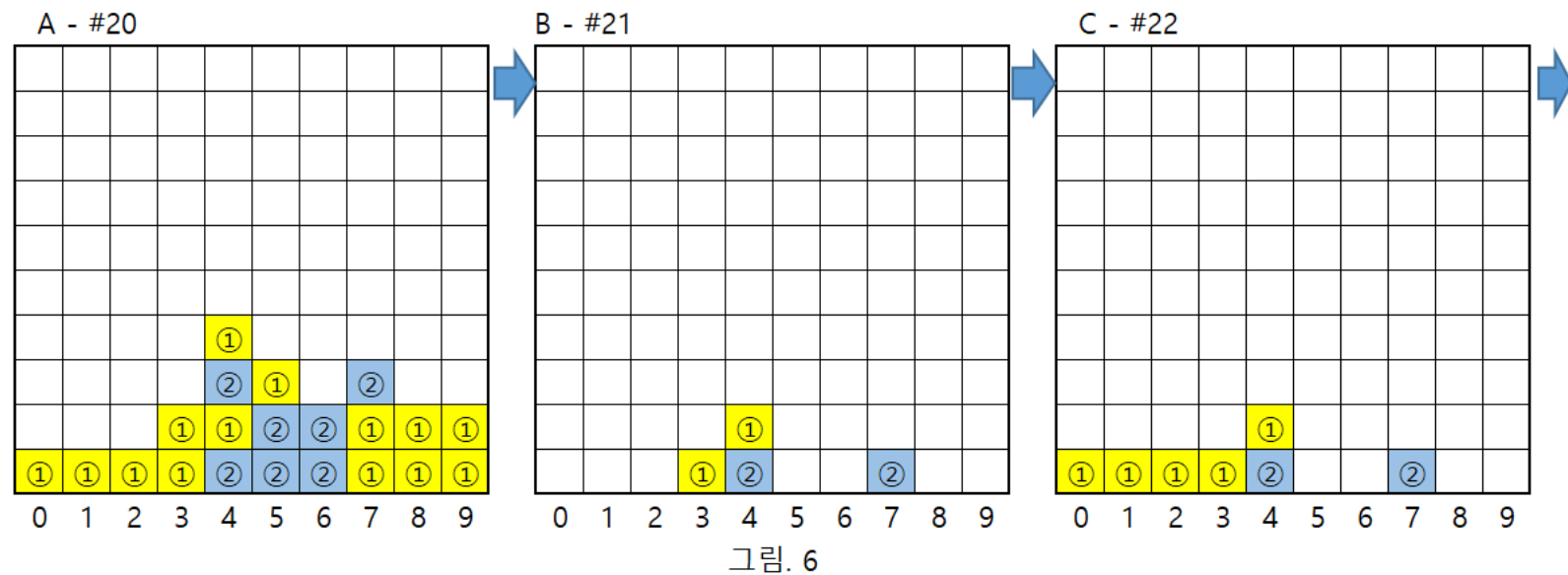
순서	함수	반환값	설명	그림
17	dropBlocks(2, 4)	7	플레이어 2가 7점을 얻는다.	[그림. 5-D]
18	dropBlocks(1, 7)	0		[그림. 5-E]
19	dropBlocks(2, 5)	0		[그림. 5-F]



Problem analysis : 예제

TS테트리스

순서	함수	반환값	설명	그림
20	dropBlocks(1, 3)	0		[그림. 6-A]
21	changeBlocks(2, 9)	11	플레이어 2가 11점을 얻는다.	[그림. 6-B]
22	dropBlocks(1, 0)	0		[그림. 6-C]



Problem analysis : 예제

TS테트리스

순서	함수	반환값	설명	그림
23	dropBlocks(2, 3)	0		[그림. 6-D]
24	dropBlocks(1, 6)	0		[그림. 6-E]
25	dropBlocks(2, 2)	0		[그림. 6-F]

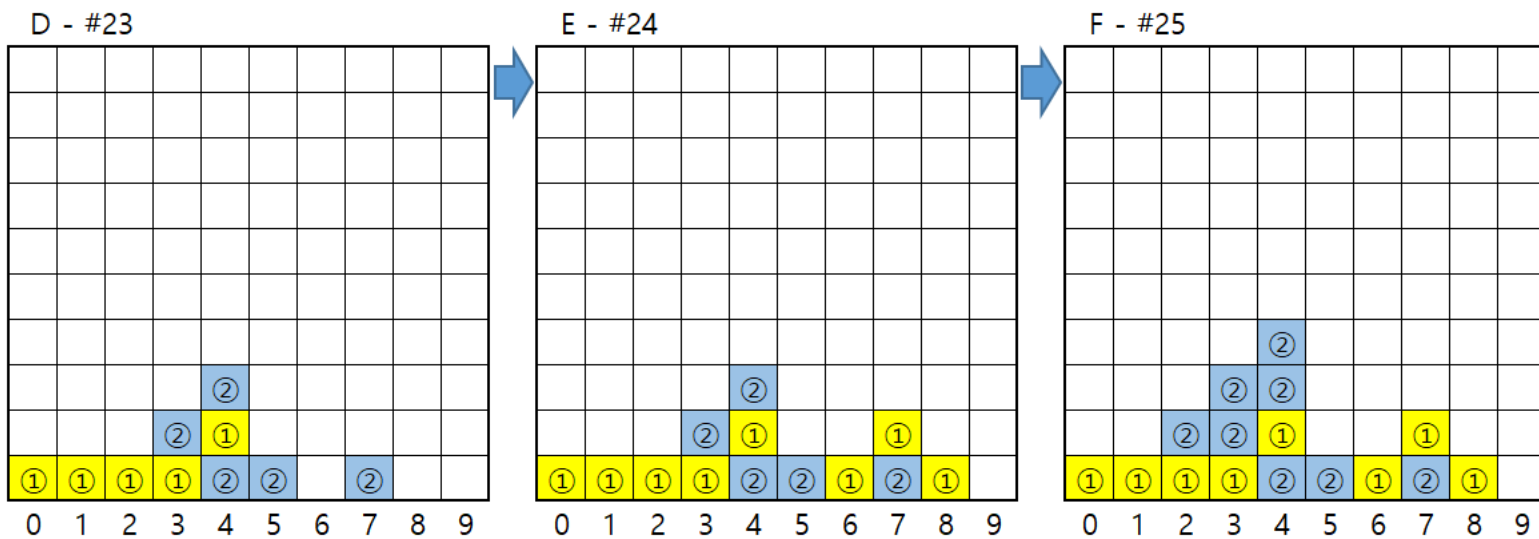


그림. 6

Problem analysis : 예제

TS테트리스

순서	함수	반환값	설명	그림
26	dropBlocks(1, 7)	0		[그림. 7-A]
27	dropBlocks(2, 2)	0		[그림. 7-B]
28	dropBlocks(1, 6)	0		[그림. 7-C]

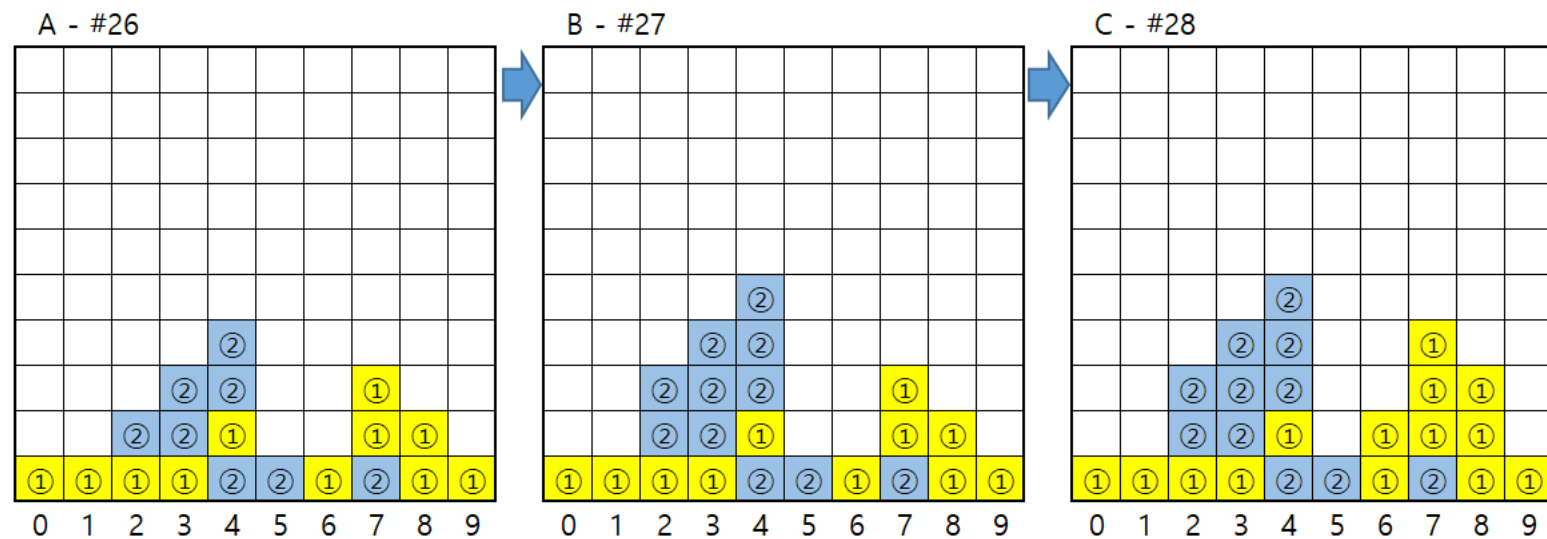
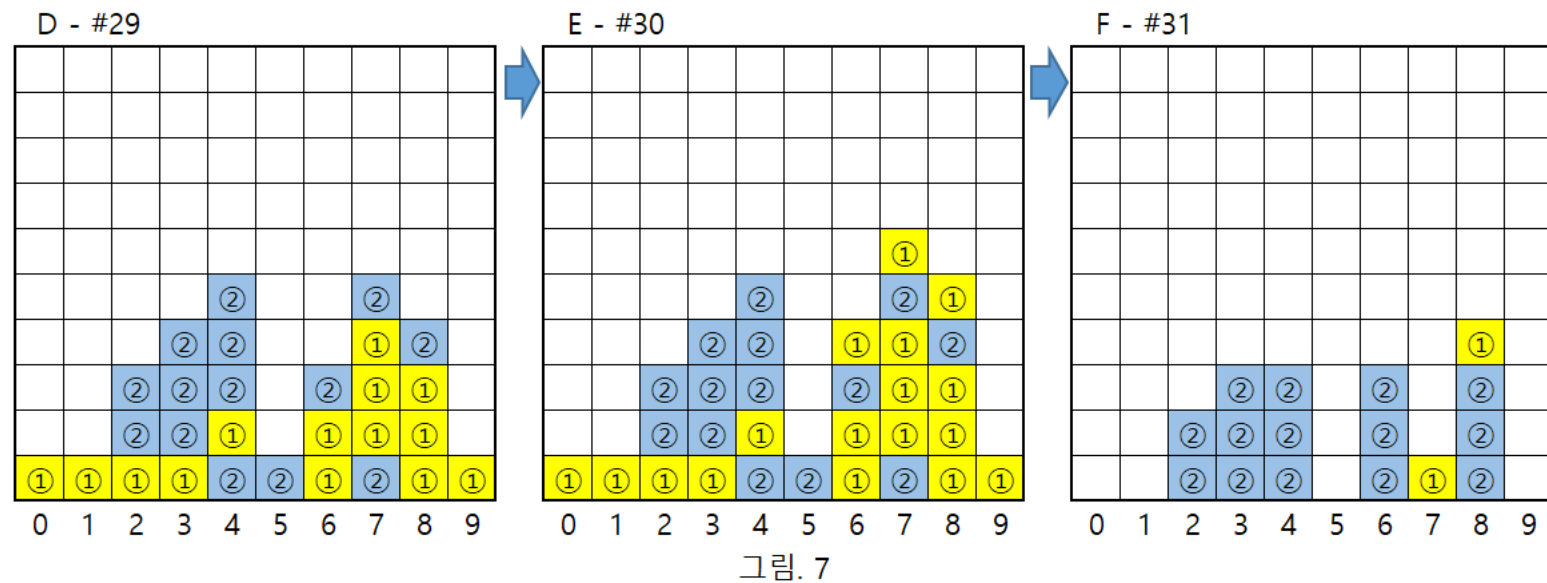


그림. 7

Problem analysis : 예제

TS테트리스

순서	함수	반환값	설명	그림
29	dropBlocks(2, 6)	0		[그림. 7-D]
30	dropBlocks(1, 6)	0		[그림. 7-E]
31	changeBlocks(2, 6)	10	플레이어 2가 10점을 얻는다.	[그림. 7-F]



Problem analysis : 예제

TS테트리스

순서	함수	반환값	설명	그림
32	getResult()	2 mStoneCnt[] = {2, 14}	플레이어 1, 2가 얻은 점수는 각각 16점과 38점이다	
33	dropBlocks(1, 4)	0		[그림. 8-A]
34	dropBlocks(2, 5)	7	플레이어 2가 7점을 얻는다.	[그림. 8-B]
35	dropBlocks(1, 4)	0		[그림. 8-C]
36	getResult()	2 mStoneCnt[] = {8, 10}	플레이어 1, 2가 얻은 점수는 각각 16점과 45점이다	

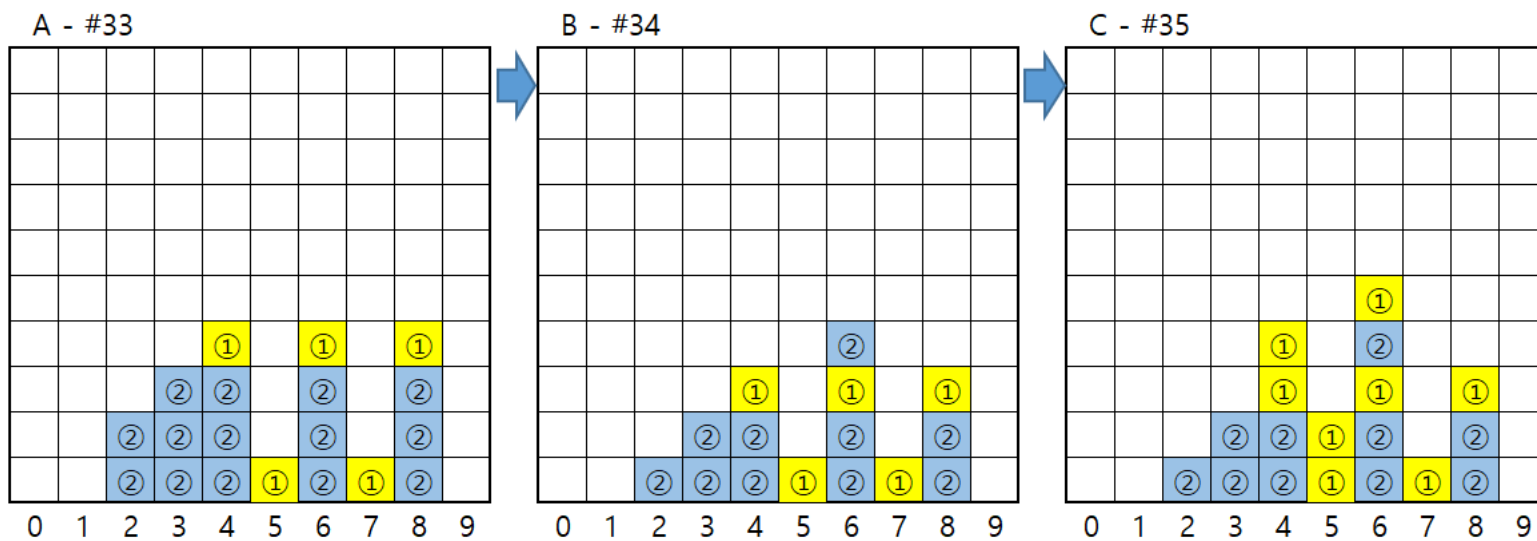
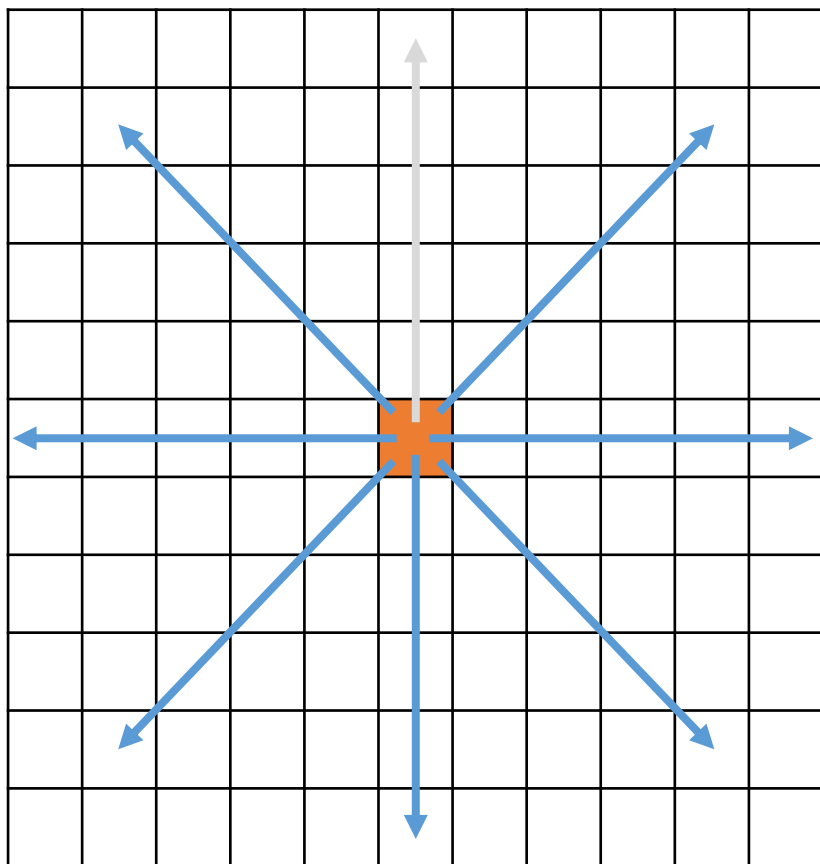


그림. 8

- 두명의 플레이어가 200 * 200 보드에 1 * 3 블록을 놓는 게임이다.
 - 테트리스처럼 블록을 떨어뜨리는 dropBlocks() 함수가 최대 10,000회 호출된다.
 - 블록과 상하좌우로 연결된 블록을 바꾸는 changeBlocks() 함수는 각각 최대 1,000회 호출된다.
 - 두 플레이어의 점수 비교 결과와 보드에 남은 블록 수를 반환하는 getResult() 함수는 각각 최대 1,000회 호출된다.
-
- 게임 유형의 문제로 주어진 조건에 맞게 구현하면 되는 문제이다.
 - 문제의 설명과 예제 설명이 그림과 함께 상세하게 주어진다.
 - 이렇게 상세하게 주어지는 문제는 문제 이해에 시간이 필요할 수 있다. 또한 구현하기에 코드 양과 구현 방법이 쉽지 않기 때문인 경우가 많다.

- `dropBlocks()` 함수에서 1×3 블록이 주어지면 기존에 보드에 있는 블록 위에 놓이게 된다.
그리고 7방향을 탐색해 보아야 한다. `changeBlocks()` 라면 8방향을 탐색해 보아야 한다.



- 인접한 블록을 바꾸는 `changeBlocks()` 함수가 호출된 경우
주어진 블록과 상하좌우로 연결된 블록을 따라 탐색해야 한다.
어떻게 할 수 있을까?

		1	1	1		1				
		2	2	1	1	1	2			
	2	2	2	1	1	1	1	2		
	2	1	1	1	2	2	2	1	2	

이 값을 2로 바꾸는 경우 1개를 제외한 모든 1이 2로 바뀐다.

- dropBlocks() 함수와 changeBlocks() 함수 호출시에
블록이 지워지는 경우 위쪽에 남은 블록은 아래로 이동한다.
여러 개의 블록을 여러 칸 이동할 수도 있다. 어떻게 할 것인가?

		2	2	1	1	1				
		1	1	1	1	1		1		
	2	2	2	1	1	1	2	2	1	
...	2	1	1	1	2	1	1	2	2	...
...	2	1	1	1	2	2	2	1	2	...


				
			
		2	2	1		1		
		1	1	1	1	1	...	1	...	
	2	2	2	1	1	1	2	2	1	
...	2	1	1	1	2	1	1	2	2	...
...	2	1	1	1	2	2	2	1	2	...

Solution sketch

- 게임 진행을 시뮬레이션 하는 구현문제이다.
 - 제한된 시간에 문제를 분석하고 주어진 제한 사항을 모두 구현하는 것이 쉽지 않다.
 - 단순 구현으로는 시간 초과가 예상된다고 해서
처음부터 최적화를 고민한다면 더욱 힘들다.
 - 먼저 구현 후에 시간이 오래 걸리는 부분을 찾아 수정하는 선택이
더 좋을 수 있다.
-
- 8방향으로 연속하여 5개 이상이 같은 플레이어의 블록이라면 제거할 수 있다.
 - interactive하게 진행되므로 dropBlocks() 함수와 changeBlocks()에 의해서만
5개 이상의 블록이 발생한다.
 - 따라서 한 방향으로서는 많아야 12개가 연속하여 같은 블록이 발견될 수 있다.
여러 행을 거치면서 더 많은 연속한 같은 블록이 발생하는 경우가 있을 수 있다.

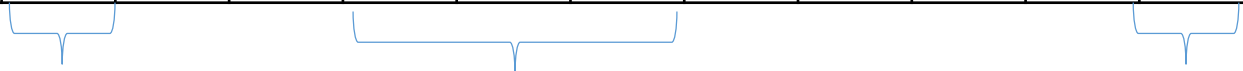
- 아래의 예에서 중앙에 4개의 2중에 하나를 1로 바꾼다면 12개의 블록이 1이 된다.

...	1	1	1	1	2	2	2	2	1	1	1	1	...



- 위와 같이 한 방향 탐색으로는 13개는 있을 수 없다. 5개가 이전에 있었다면 이미 제거되어야 했기 때문이다.
- 하지만 여러 행을 거치며 13개 이상이 발생하는 경우가 생길 수 있다.

												2	2	2
				2	2	2	2	1	2	2	2	2	1	2
1	1	1	1	2	1	1	2	2	2	1	1	1	1	2



- dropBlocks() 함수가 주어진 경우 새로 추가된 3개의 블록과 연결된 블록들이 제거되는지 검토한다. 그 결과 제거되는 블록이 없다면 추가 작업은 필요 없다. 제거되는 블록이 있는 경우 전체 블록에 대하여 제거 될 수 있는지 검토해야 한다. 제거가 연속하여 일어날 수 있다. 블록이 지워질수록 탐색 범위는 줄어든다.
- 연속한 5개 이상의 블록을 지우는 프로세스는 다음과 같다.
 1. 지울 블록 탐색하기 :
 - 1) 전체 탐색의 경우 : 가장 왼쪽 아래에서부터 시작 하므로 4방향이면 충분하다. $\uparrow, \nearrow, \rightarrow, \searrow$
 - 2) 3개의 블록을 놓고 이들과 인접한 블록을 탐색하는 경우에는 7방향을 탐색해야 한다.
 $\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow$
 2. 지울 블록이 있다면 표시하기
 3. 표시된 블록 지우기, 점수 업데이트, 보드에 남은 플레이어 별 블록 수 업데이트
지우는 작업에 있어 **tow pointer**를 이용하여 자리 이동 프로세스를 효율적으로 수행할 수 있다.

- changeBlocks() 함수가 주어진 경우 상하좌우 인접한 블록에 대한 탐색이 필요하다.
이때, DFS 또는 BFS 탐색을 사용할 수 있다.
전체 블록에 대하여 제거 될 수 있는지 검토해야 한다.
제거가 연속하여 일어날 수 있다.
블록이 지워질수록 탐색 범위는 줄어든다.
- dropBlocks() 함수와 changeBlocks() 함수를 진행하면서
각 플레이어의 점수와 보드에 남은 블록 수를 관리 할 수 있다.
따라서 getResult() 함수에는 $O(1)$ 에 답할 수 있다.

[summary]

- 게임의 진행상황을 구현하는 문제이다.
- 문제 분석과 구현에 시간이 많이 걸리는 경향이 있다.
빠른 문제 분석과 함께 빠르고 정확한 구현력이 요구된다.
- 처음부터 최적화를 고민하다가 실패하는 경향이 있다.
적절한 선에서 타협하고 구현 후에 최적화하는 전략이 필요하다.
- 출제 빈도는 낮은 편이다.
비슷한 문제로 jungol 3065 : 테트리스 문제가 출제된 적이 있다.
- 그래프 탐색(DFS, BFS)은 철저하게 준비해 두어야 한다.
- tow pointer, sliding window 기법 등을 익혀두자.

Code example

▪
▪

Code example

TS테트리스

```
#include <cstring>
#include <queue>
using namespace std;

using pii = pair<int, int>;
const int LM = 201;
int H, W;           // 보드의 높이와 너비
int board[LM][LM];  // 블록에 플레이어 번호를 표시
int mark[LM][LM];   // 삭제될 블록 표시
int score[3];       // 플레이어별 점수
int remain[3];      // 플레이어별 보드에 남은 블록의 수
int height[LM];     // 각 열별 보드에 쌓인 블록의 높이(바닥:H - 1, 꼭대기:0)

int dr[4] = { -1, -1, 0, 1 }; // ↑, ↗, →, ↘
int dc[4] = { 0, 1, 1, 1 };
int drc[4][2] = { {-1, 0}, {0, 1}, {1, 0}, {0, -1} }; // ↑, →, ↓, ←

queue<pii> que;
```

Code example

TS테트리스

```
void push(int nr, int nc, int player) {  
    // 경계체크, 범위를 벗어나거나  
    if (nr < 0 || nr >= H || nc < 0 || nc >= W || board[nr][nc] != (player ^ 3)) return;  
    que.push({ nr, nc });  
    --remain[board[nr][nc]]; // 상대 플레이어의 남은 블록수 감소  
    ++remain[board[nr][nc] = player]; // 현재 플레이어의 남은 블록수 증가  
}  
  
void changeBFS(int r, int c, int player) { // flood fill  
    que = {}; // 큐 초기화  
    push(r, c, player); // 시작위치  
    while (!que.empty()) {  
        pii t = que.front(); que.pop();  
        for (int i = 0; i < 4; ++i) // 상하좌우로 탐색  
            push(t.first + drc[i][0], t.second + drc[i][1], player);  
    }  
}
```

Code example

TS테트리스

```
int eraseBlock(int player) { // mark[][]에 표시된 블록 제거하기
    int cnt = 0;
    for (int c = 0; c < W; ++c) {
        // two pointer 준비
        int r1 = H - 1; // 선행 pointer
        int r2 = H - 1; // 후행 pointer
        while (r1 >= height[c]) {
            if (mark[r1][c]) { // 블록 제거하기
                if (board[r1][c] == player) // player블록이라면 점수 증가
                    ++cnt;
                mark[r1][c] = 0; // 체크 해제
                --remain[board[r1][c]]; // board[][]상에 player의 남은 블록수 감소
                --r1; // 위로 올라가며 검사
            }
            else { // 제거되지 않은 블록은 아래로 이동 : tow pointer : 이때 실제 삭제가 일어남
                board[r2--][c] = board[r1--][c];
            }
        }

        height[c] = r2 + 1; // new height
        while (r2 > r1) {
            board[r2--][c] = 0;
        }
    }
    return cnt;
}
```


Code example

TS테트리스

```
int search(int r, int c, int rr, int cc, int color) {
    int cnt = 0;
    while (r >= 0 && r < H && c >= 0 && c < W && board[r][c] == color) {
        cnt++, r += rr, c += cc;
    }
    return cnt;
}

void eraseMark(int r, int c, int rr, int cc, int len) {
    for (int i = 0; i < len; ++i, r += rr, c += cc) {
        mark[r][c] = 1;
    }
}
```

Code example

TS테트리스

```
int updateAll(int player){
    int ret = 0;
    bool flag = true;
    while (flag){
        flag = false;
        for (int c = 0; c < W; ++c) {
            for (int r = H - 1; r >= height[c]; --r){
                for (int k = 0; k < 4; ++k) { // k:
                    int len = search(r, c, dr[k], dc[k], board[r][c]); // 1. 지울 블록 탐색
                    if (len >= 5){
                        eraseMark(r, c, dr[k], dc[k], len); // 2. 지울 블록 표시
                        flag = true;
                    }
                }
            }
        }
        if (!flag)
            break;
        ret += eraseBlock(player); // 3. 표시된 블록 지우기 + 점수계산 + 보드에 남은 블록 계산
    }
    score[player] += ret;
    return ret;
}
```

Code example

TS테트리스

```
int updateTop(int player, int col){
    int ret = 0;
    bool flag = false;
    for (int c = col; c < col + 3; ++c) {
        int r = height[c];
        for (int k = 0; k < 4; ++k){
            int len1 = search(r, c, dr[k], dc[k], player);
            int len2 = search(r, c, -dr[k], -dc[k], player);

            if (len1 + len2 > 5){ // board[r][c]를 두번 세므로 합이 6 이상이어야 한다.
                eraseMark(r, c, dr[k], dc[k], len1);
                eraseMark(r, c, -dr[k], -dc[k], len2);
                flag = true;
            }
        }
    }

    if (!flag) return 0;
    ret = eraseBlock(player);
    score[player] += ret;
    return ret;
}
```

Code example

TS테트리스

```
void init(int w, int h){
    H = h, W = w;
    memset(board, 0, sizeof(board));
    for (int c = 0; c < W; ++c) // 보드의 각 열에 놓인 블록의 높이 초기화
        height[c] = H;
    score[1] = score[2] = 0;      // 각 player의 점수 초기화
    remain[1] = remain[2] = 0;   // 각 player의 보드에 남은 블록의 개수 초기화
}

int dropBlocks(int player, int col){
    for (int c = col; c < col + 3; ++c) { // drop block
        board[--height[c]][c] = player;
        ++remain[player];
    }
    int ret = updateTop(player, col);      // 놓인 자리 3곳을 우선 탐색
    if (ret == 0) return 0;                // 지워지는 블록이 없다면
    ret += updateAll(player);              // 전체 탐색
    return ret;
}
```

Code example

TS테트리스

```
int changeBlocks(int player, int col){    // 블록의 소유자를 바꾸기
    // 목표위치에 블록이 없거나 이미 player의 블록이라면 할 일이 없다.
    if (board[H - 1][col] == 0 || board[H - 1][col] == player) return 0;

    changeBFS(H - 1, col, player);
    int ret = updateAll(player);          // 전체 탐색
    return ret;
}

int getResult(int stoneCnt[2]){
    stoneCnt[0] = remain[1];
    stoneCnt[1] = remain[2];
    int ret = score[1] == score[2] ? 0 : (score[1] > score[2] ? 1 : 2);
    return ret;
}
```

Thank you.