JooHyun – Lee (comkiwer)

TS큰퍼즐

Hancom Education Co. Ltd.

Problem

Problem

퍼즐을 맞추는 프로그램을 작성한다.

퍼즐 조각의 네 변은 각각 같은 간격으로 M개의 들어간 부분이나 나온 부분이 있다.

퍼즐 조각의 바깥으로 나온 부분을 산, 안으로 들어간 부분을 골이라 하는데,
편의상 산 하나로 표현한다.

퍼즐 조각의 가로, 세로의 크기는 동일하며 네 변의 산의 폭이나 개수도 동일하다.

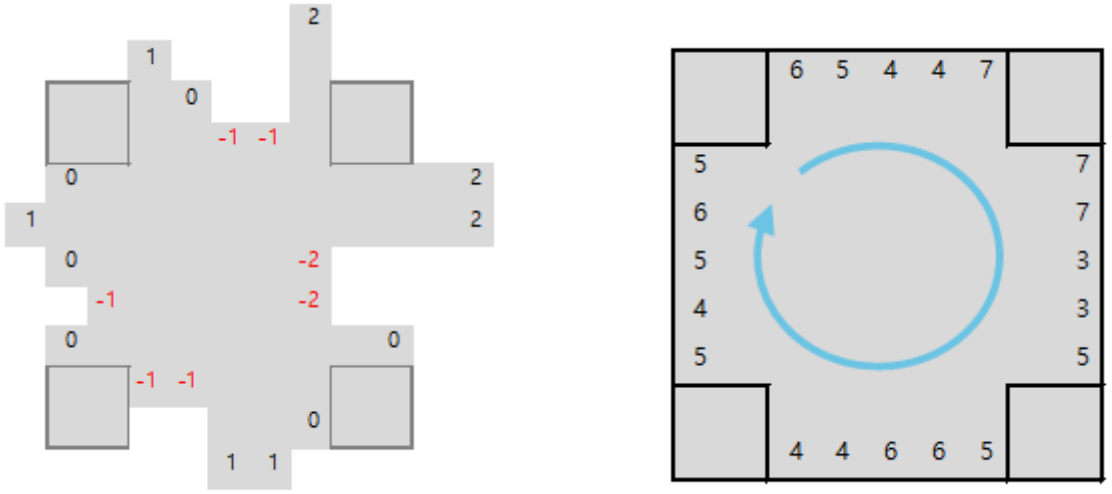
퍼즐 조각의 산의 범위는 -4 ~ +4 인데, 5를 더하여
1 ~ 9 까지의 범위로 표현하면 퍼즐 조각은 정사각형으로 간략히 표현이 가능하다.

Problem

TS큰퍼즐

[Fig. 1] 은 산의 개수가 5일때의 퍼즐 조각의 예이다.

왼쪽이 실제 퍼즐 조각의 모양이고, 오른쪽이 이를 간략하게 표현한 그림이다.



[Fig. 1]

Problem

각 산의 높이 하나는 10진수 한자리로 표현되며,
퍼즐 조각의 한 변의 모양은 10진수 M자리로 표현된다.
퍼즐 조각의 변의 모양은 상단 좌측부터 시계방향순으로 주어지며,
10진수 4개의 배열로 상우하좌 순으로 주어진다.

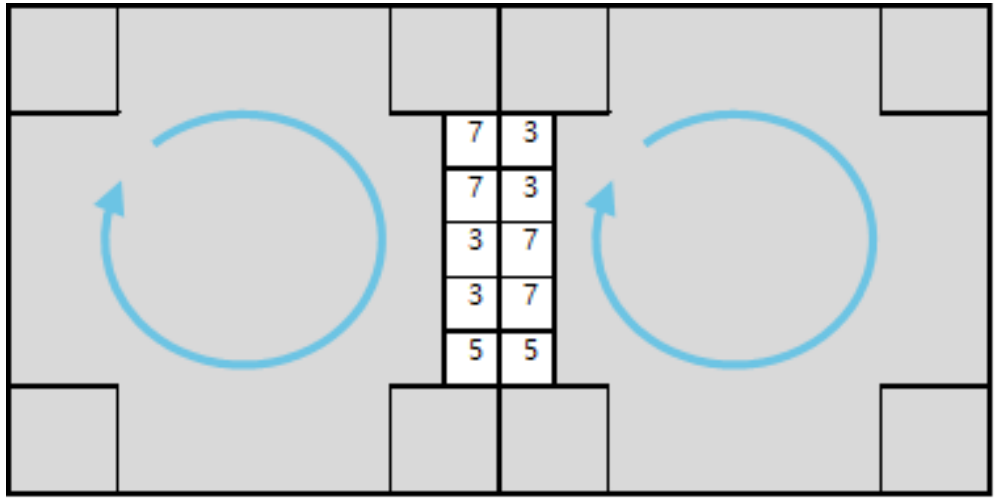
예를 들어[Fig. 1]의 퍼즐 조각은
{ 65447, 77335, 56644, 54565 } 이 입력으로 주어진다.

퍼즐 조각을 놓기 위해서는 인접한 퍼즐 조각과 맞닿는 변이 완벽히 맞물려야 한다.
완벽히 맞물리기 위해서는 모든 마주보는 산의 높이의 합이 10 이 되어야 한다.

Problem

TS큰퍼즐

[Fig. 2] 는[Fig. 1] 의 퍼즐 조각 오른쪽에 다른 퍼즐 조각을 놓는 경우의 예이다.
오른쪽 변 77335 와 완벽히 맞물리는 조각은 시계방향으로 표현되므로 왼쪽 변이 57733 인 조각이다.
퍼즐 조각을 놓기 위해서는 상하좌우 인접한 퍼즐 조각의 맞닿는 산이 완벽하게 맞물려야 한다.



[Fig. 2]

한 조각 내에는 산의 배열이 동일한 두 변이 존재하지 않는다.

Problem

퍼즐판의 전체 크기는 테두리를 포함하여 (N + 2)*(N + 2) 의 크기이며,
각 테스트 케이스 초기에 퍼즐 판의 외곽 테두리의 퍼즐 조각이 주어지므로,
맞추어야 하는 빈 퍼즐 판의 크기는 N*N 이다.

좌표는 퍼즐 조각을 놓을 수 있는 빈 공간의 왼쪽 상단부터
열과 행을 x와 y로 표현했을 때,
(x, y) 좌표를 사용하며, (1, 1) 에서부터 (N, N) 까지 순차적으로 부여된다.

mU, mR, mB, mL은 각 테스트 케이스 초기에 주어지는 외곽 테두리 퍼즐 조각의
정보이며, 순서는 x또는 y가 증가하는 방향으로 주어진다.

Problem

	mU[0] (1,0)	mU[1] (2,0)	mU[2] (3,0)	mU[3] (4,0)	
mL[0] (0,1)	(1,1)	(2,1)	(3,1)	(4,1)	mR[0] (5,1)
mL[1] (0,2)	(1,2)	(2,2)	(3,2)	(4,2)	mR[1] (5,2)
mL[2] (0,3)	(1,3)	(2,3)	(3,3)	(4,3)	mR[2] (5,3)
mL[3] (0,4)	(1,4)	(2,4)	(3,4)	(4,4)	mR[3] (5,4)
	mB[0] (1,5)	mB[1] (2,5)	mB[2] (3,5)	mB[3] (4,5)	

[Fig. 3]

초기 상태에 퍼즐 판의 외곽 테두리의 퍼즐 조각이 주어진다.

Problem

이후 퍼즐 조각이 하나씩 주어지면, 아래와 같은 규칙을 따라 퍼즐 조각을 놓는다.

1. 하나 이상의 변이 이미 놓여진 다른 퍼즐 조각과 맞닿아 있어야 하며,
   맞닿은 부분은 산이 모두 완벽히 맞물려야 한다.
2. 퍼즐 조각이 놓일 수 있는 자리가 여러 개일 경우, 맞닿는 변이 많은 쪽이 우선된다.
3. 맞닿는 변의 수가 같은 경우, 위쪽 자리가 우선된다.
4. 같은 행 안에서는 왼쪽 자리가 우선된다.

퍼즐 조각을 맞출 때는 90도, 180도, 270도 회전시킨 후 맞추는 것도 가능하다.

퍼즐 조각을 뒤집거나 비트는 것은 불가능하다.

한번 놓인 퍼즐 조각은 이동하지 않는다.

Problem

TS큰퍼즐

void init(int N, int M, int mU[][4],int mR[][4],int mB[][4],int mL[][4])

각 테스트 케이스의 처음에 호출된다.

퍼즐판의 크기는 (N + 2) x (N + 2) 이고, 각 퍼즐 조각의 한 변의 산의 개수는 M이다.

테두리의 위쪽 퍼즐 조각들 mU, 테두리의 오른쪽 퍼즐 조각들 mR,

테두리의 아래쪽 퍼즐 조각들 mB, 테두리의 왼쪽 퍼즐 조각들 mL 이 주어진다.

퍼즐 조각의 입력의 순서는 [Fig. 3] 순서를 참고한다.

	mU[0] (1,0)	mU[1] (2,0)	mU[2] (3,0)	mU[3] (4,0)	
mL[0] (0,1)	(1,1)	(2,1)	(3,1)	(4,1)	mR[0] (5,1)
mL[1] (0,2)	(1,2)	(2,2)	(3,2)	(4,2)	mR[1] (5,2)
mL[2] (0,3)	(1,3)	(2,3)	(3,3)	(4,3)	mR[2] (5,3)
mL[3] (0,4)	(1,4)	(2,4)	(3,4)	(4,4)	mR[3] (5,4)
	mB[0] (1,5)	mB[1] (2,5)	mB[2] (3,5)	mB[3] (4,5)	

[Fig. 3]

Problem

void init(int N, int M, int mU[][4],int mR[][4],int mB[][4],int mL[][4])

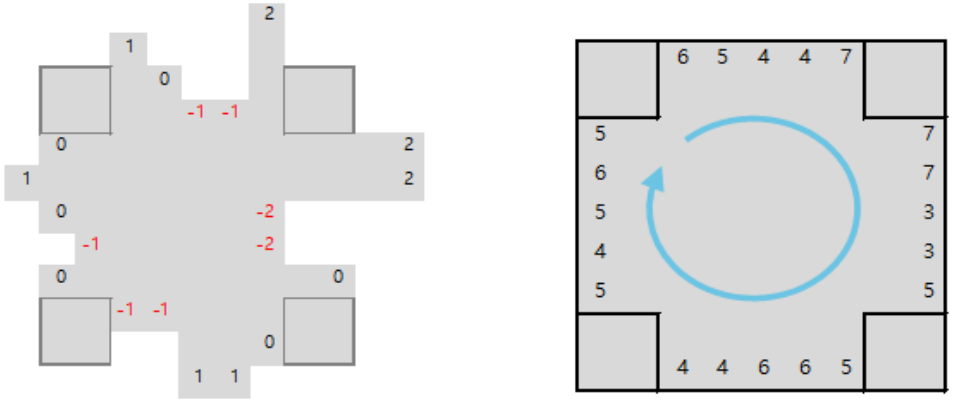
(계속)

mU[], mR[], mB[], mL[] 은 각 퍼즐 조각의 상, 우, 하, 좌 4개의 변의 모양이 0을 포함하지 않는 M자리 10진수의 형태로 순서대로 주어진다.

각 퍼즐 조각의 모양은[Fig. 1] 을 참고한다.

초기 상태에서 퍼즐은 N x N개를 놓을 수 있는 상태이다.

퍼즐 조각을 놓을 수 있는 좌표는(X, Y) 좌표를 사용하며, 각 좌표의 범위는 1 ~N 이다.



[Fig. 1]

Problem

TS큰퍼즐

void init(int N, int M, int mU[][4],int mR[][4],int mB[][4],int mL[][4])

(계속)

Parameters

N : 퍼즐 판의 크기(4 ≤ N ≤ 1000)

M : 퍼즐조각들의 한 변에 있는 산의 개수(3 ≤ M ≤ 8)

mU : 퍼즐판의 위쪽 퍼즐 조각들의 정보(111 ≤ mU[][] ≤ 99,999,999)

mR : 퍼즐판의 오른쪽 퍼즐 조각들의 정보(111 ≤ mR[][] ≤ 99,999,999)

mB : 퍼즐판의 아래쪽 퍼즐 조각들의 정보(111 ≤ mB[][] ≤ 99,999,999)

mL : 퍼즐판의 왼쪽 퍼즐 조각들의 정보(111 ≤ mL[][] ≤ 99,999,999)

void destroy()

각 테스트 케이스의 마지막에 호출된다.

빈 함수로 두어도 채점에는 영향을 주지 않는다.

Problem

int put(int mPiece[4])

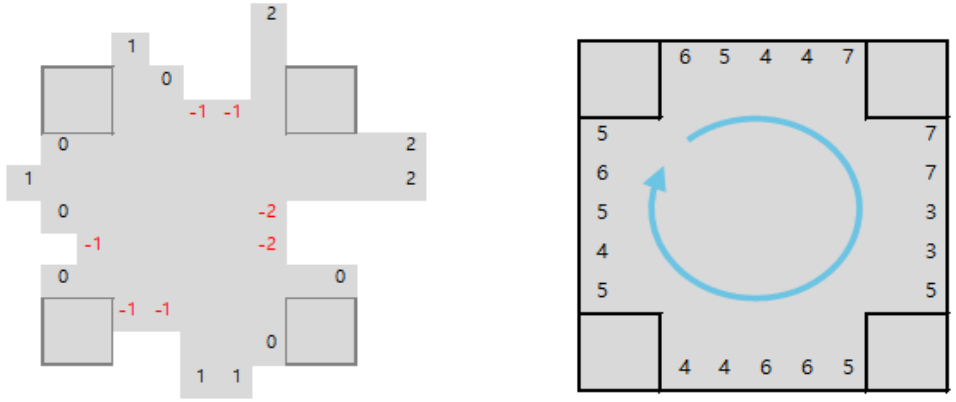
모양이 mPiece인 퍼즐 조각을 놓는다.

퍼즐 조각의 상, 우, 하, 좌 4개의 변의 모양이

mPiece[0], mPiece[1], mPiece[2], mPiece[3] 순서로 주어진다.

각 변에서 모양은 퍼즐 조각의 중심에서 밖을 바라보는 방향으로

왼쪽에서 오른쪽 순으로 산의 높이가 주어진다. ([Fig. 1] 참고)



[Fig. 1]

int put(int mPiece[4]) (계속)

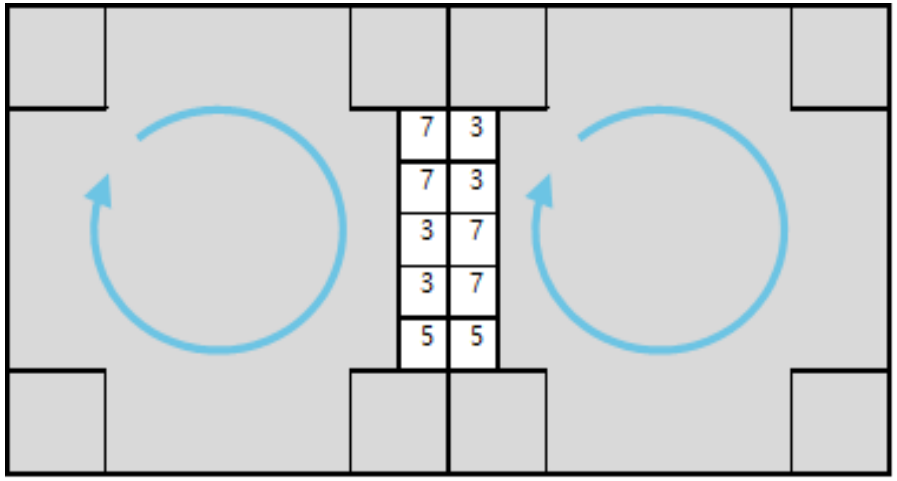
산은 각 변당 0을 포함하지 않는 M자리 10진수로 입력된다.

산의 높이는 1~9로 주어지며, 산의 높이가 1인 경우는 맞닿는 산의 높이가 9, 2인 경우는 8과 같이 맞닿는 산의 높이의 합이 10인경우 맞물릴 수 있다. ([Fig. 2] 참고)

맞닿는 두 퍼즐 조각의 마주보는 변의 산이 모두 맞물리는 경우 완벽하게 맞물린다고 한다.

퍼즐 조각은 퍼즐 판의 빈 위치에 놓아야 하며,

적어도 한 변이 이미 놓여진 다른 퍼즐 조각과 인접해야 한다.



[Fig. 2]

Problem

int put(int mPiece[4]) (계속)

4개의 변 모두 인접한 다른 퍼즐 조각이 없거나,

인접한 퍼즐 조각과 한 변이라도 완벽하게 맞물리지 않는다면 퍼즐 조각을 놓을 수 없다.

퍼즐 조각을 놓을 수 있으면, 퍼즐을 놓은 곳의 x 좌표와 y 좌표의 합을 반환한다.

퍼즐 조각을 놓을 수 없으면 -1 을 반환한다.

Parameters

mPiece : 새 퍼즐 조각의 모양 (111 ≤ mPiece[] ≤ 99,999,999)

Returns

퍼즐을 놓은 곳의 x 좌표와 y 좌표의 합을 반환한다.

Problem

[제약사항]

1. **각 테스트 케이스 시작 시** init() **함수가, 종료 시** destroy() **함수가 호출된다.**
2. **각 테스트 케이스에서** put() **함수는 최대** 10,000**회 호출된다.**

Problem analysis

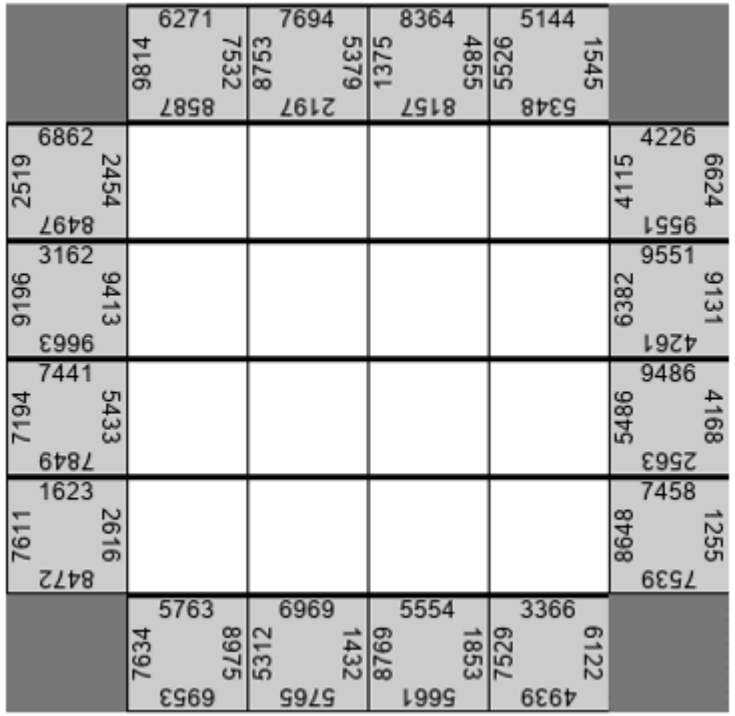
Problem analysis : 예제

TS큰퍼즐

아래는 각 함수 호출 이후 퍼즐 판에 조각이 놓여진 상태를 나타낸다.

#	Function	Description	return
1	init(4,4, {{6271, 7532, 8587, 9814},..... }, {{4226, 6624, 9551, 4115},..... }, {{5763, 8975, 6953, 7634},..... }, {{6862, 2454, 8497, 2519},..... })	빈 퍼즐 판의 크기 N 은 4, 각 퍼즐 조각의 산의 개수 M 은 4이다. 퍼즐 판의 외곽 퍼즐 조각들이 상우하좌 각각 N 개씩 입력된다.	

[#1]

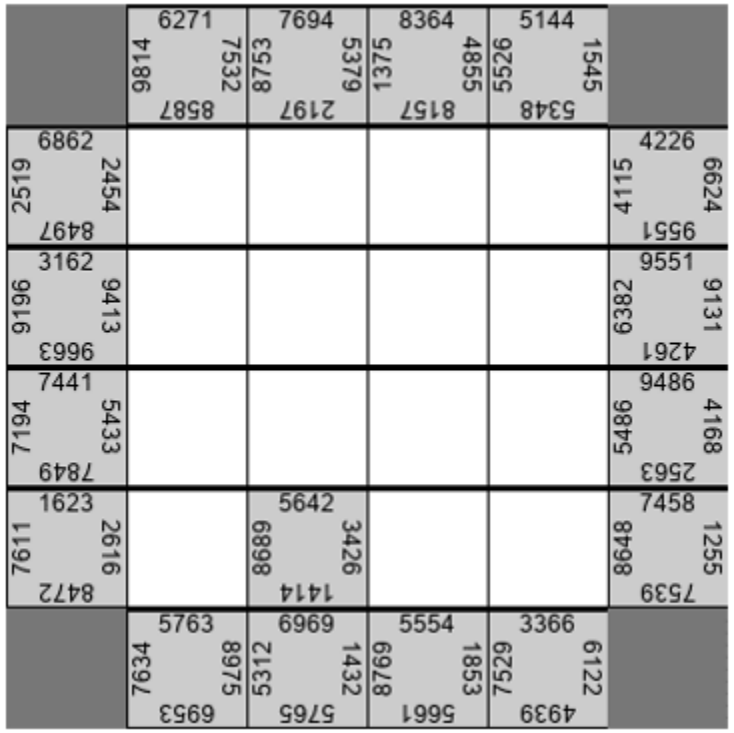


Problem analysis : 예제

TS큰퍼즐

#	Function	Description	return
2	put({3426, 1414, 8689, 5642})	상우하좌 각 변의 모양이 각각 시계방향으로 {3426, 1414, 8689, 5642} 의 퍼즐 조각이 입력된다. 퍼즐 조각은 시계방향으로 90도 회전하여 (2,4)위치에 놓여진다.	6

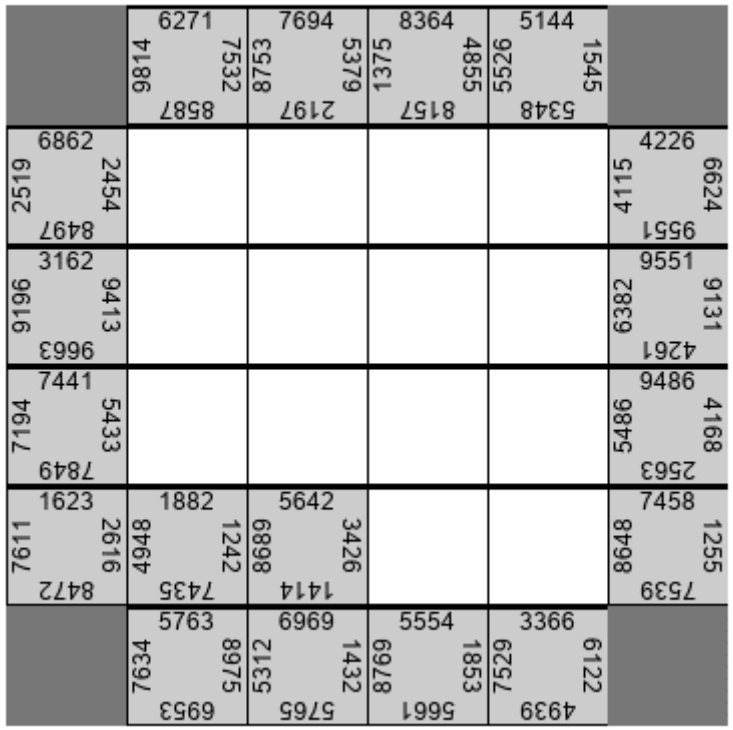
[#2]



Problem analysis : 예제

#	Function	Description	return
3	put({4948, 1882, 1242, 7435})	{4948, 1882, 1242, 7435} 의 퍼즐 조각이 입력된다. 시계방향으로 270도 회전하여 (1,4)위치에 놓여진다.	5

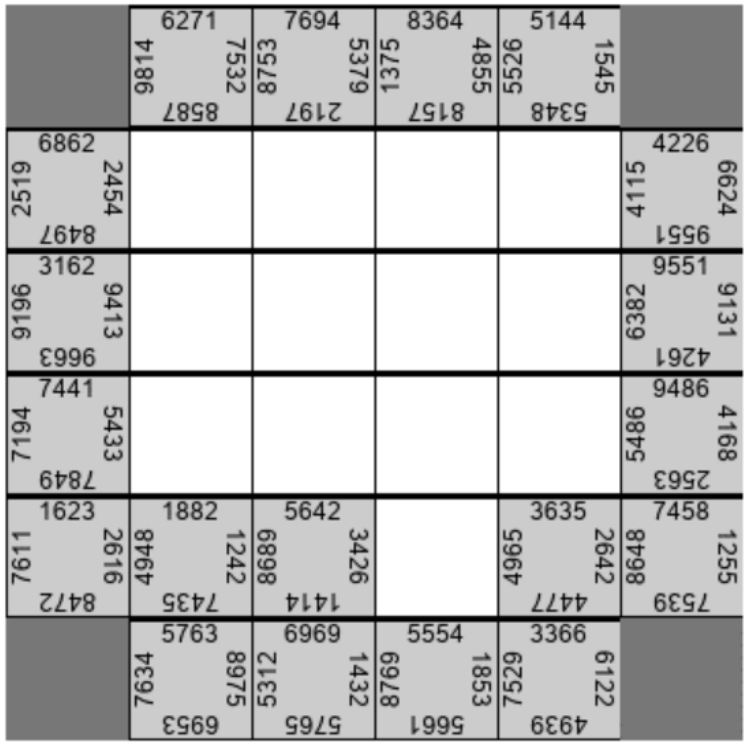
[#3]



Problem analysis : 예제

#	Function	Description	return
4	put({3635, 2642, 4477, 4965})	{3635, 2642, 4477, 4965} 의 퍼즐 조각이 입력된다. 시계방향으로 0도 회전하여 (4,4)위치에 놓여진다.	8

[#4]

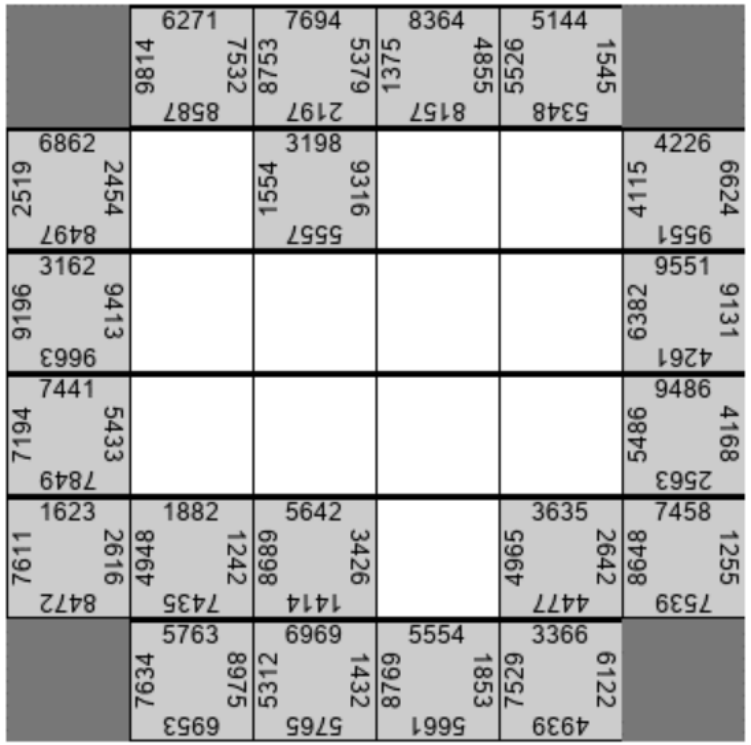


Problem analysis : 예제

TS큰퍼즐

#	Function	Description	return
5	put({9316, 5557, 1554, 3198})	{9316, 5557, 1554, 3198} 의 퍼즐 조각이 입력된다. 시계방향으로 90도 회전하여 (2,1)위치에 놓여진다.	3

[#5]



Problem analysis : 예제

TS큰퍼즐

#	Function	Description	return
6	put({2675, 5996, 7924, 1396})	{2675, 5996, 7924, 1396} 의 퍼즐 조각이 입력된다. 시계방향으로 0도 회전하여 (4,1)위치에 놓여진다.	5
7	put({6813, 8274, 9291, 7965})	{6813, 8274, 9291, 7965} 의 퍼즐 조각이 입력된다. 시계방향으로 0도 회전하여 (4,2)위치에 놓여진다.	6
8	put({6555, 4867, 5585, 5416})	{6555, 4867, 5585, 5416} 의 퍼즐 조각이 입력된다. 시계방향으로 180도 회전하여 (3,4)위치에 놓여진다.	7
9	put({4179, 2715, 4971, 3592})	{4179, 2715, 4971, 3592} 의 퍼즐 조각이 입력된다. 시계방향으로 90도 회전하여 (3,1)위치에 놓여진다.	4
10	put({6568, 3252, 6559, 8421})	{6568, 3252, 6559, 8421} 의 퍼즐 조각이 입력된다. 시계방향으로 270도 회전하여 (1,1)위치에 놓여진다.	2
11	put({5413, 6913, 2175, 5938})	{5413, 6913, 2175, 5938} 의 퍼즐 조각이 입력된다. 시계방향으로 90도 회전하여 (3,2)위치에 놓여진다.	5
12	put({3143, 7914, 3456, 5255})	{3143, 7914, 3456, 5255} 의 퍼즐 조각이 입력된다. 시계방향으로 270도 회전하여 (3,3)위치에 놓여진다.	6
13	put({4265, 5747, 4567, 9181})	{4265, 5747, 4567, 9181} 의 퍼즐 조각이 입력된다. 시계방향으로 90도 회전하여 (4,3)위치에 놓여진다.	7
14	put({5398, 9358, 1539, 3555})	{5398, 9358, 1539, 3555} 의 퍼즐 조각이 입력된다. 시계방향으로 90도 회전하여 (2,2)위치에 놓여진다.	4

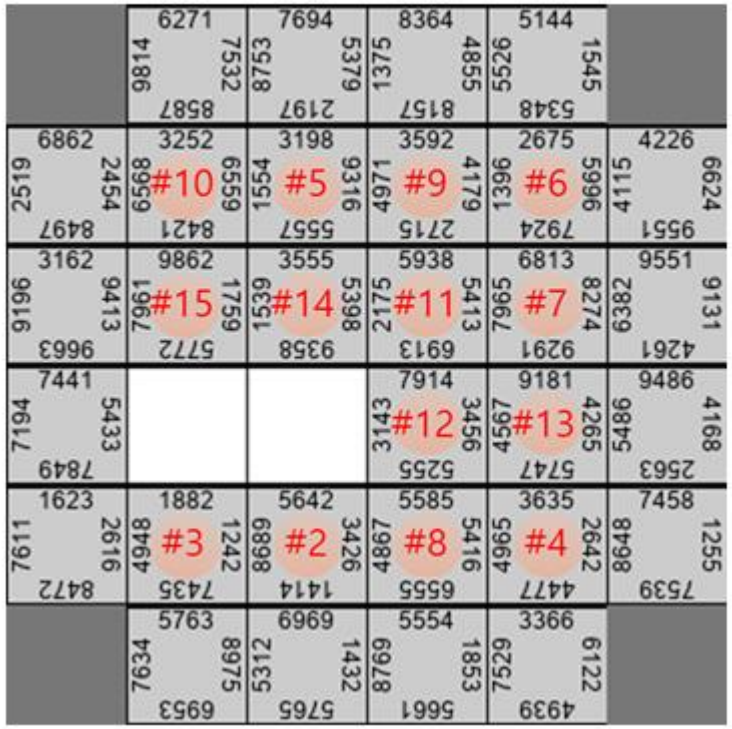
Problem analysis : 예제

TS큰퍼즐

#	Function	Description	return
15	put({7961, 9862, 1759, 5772})	{7961, 9862, 1759, 5772} 의 퍼즐 조각이 입력된다. 시계방향으로 270도 회전하여 (1,2)위치에 놓여진다.	3

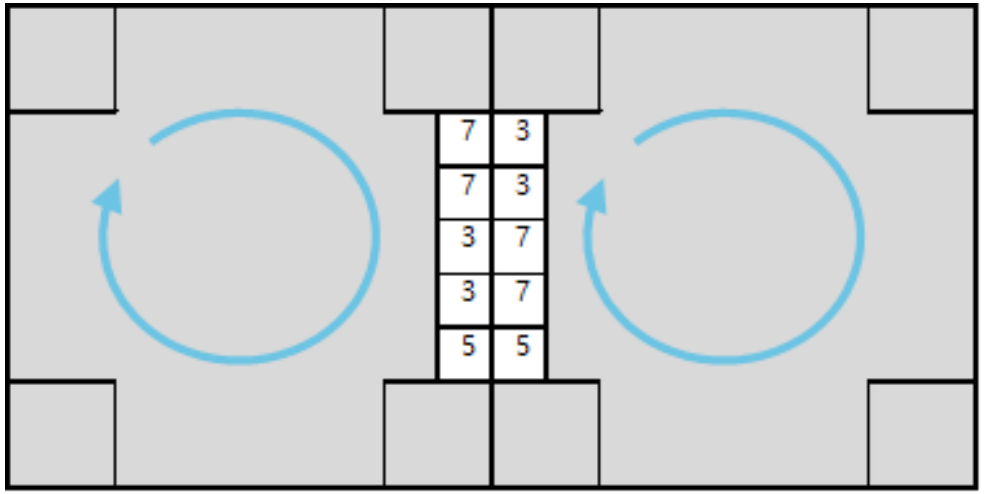
[#15]

붉은색으로 표시된 숫자는 해당 호출 순서에 추가된 조각을 의미한다.



Problem analysis : 제약조건 및 API

- 퍼즐판의 최대 크기는 1000*1000이다.
- 퍼즐 한조각은 4개의 변으로 이루어지며
  12시 방향부터 시계방향으로 변의 정보가 주어진다.
- 한 변의 최대 길이는 8이며 각 자리에 사용되는 숫자는 1~9이다.
- 한 변과 맞닿는 다른 한 변은 아래 그림과 같이 각 자리의 합이 10을 이루어야 한다.



Problem analysis : 제약조건 및 API

- init()함수를 통하여 초기 퍼즐판의 모습이 아래 그림과 같이 주어진다.
  이후 put() 함수를 통하여 퍼즐 조각 맞추기를 최대 10,000번 수행할 수 있다.
- 아래 퍼즐판의 크기는 4인 경우이다.

	mU[0] (1,0)	mU[1] (2,0)	mU[2] (3,0)	mU[3] (4,0)	
mL[0] (0,1)	(1,1)	(2,1)	(3,1)	(4,1)	mR[0] (5,1)
mL[1] (0,2)	(1,2)	(2,2)	(3,2)	(4,2)	mR[1] (5,2)
mL[2] (0,3)	(1,3)	(2,3)	(3,3)	(4,3)	mR[2] (5,3)
mL[3] (0,4)	(1,4)	(2,4)	(3,4)	(4,4)	mR[3] (5,4)
	mB[0] (1,5)	mB[1] (2,5)	mB[2] (3,5)	mB[3] (4,5)	

Problem analysis : 제약조건 및 API

- 퍼즐판의 크기 N이 1,000 이고 퍼즐조각 한 변의 길이가 8이라고 할 때,
  push()를 통하여 퍼즐 맞추기를 실행하는데 단순히 전체를 탐색하는 경우에
  시간 복잡도는 어떻게 될까?

- (퍼즐판에 놓인 변의 수) * (새로운 퍼즐의 변수) * (push()호출 수) * (TC수)
  = 4,000이상 * 4 * 10,000 * 50 = 8,000,000,000 이상
  이 될 수도 있다.

- 이는 너무 많은 시간을 필요로 한다.
  어떻게 해야 할 까?

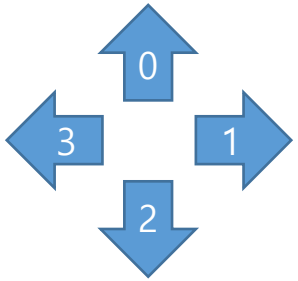
Solution sketch

Solution sketch

- 퍼즐조각의 변은 3자리에서 8자리 10진 정수로 이루어진다.
  이를 이용하여 탐색대상을 그룹 짓는 방법을 생각할 수 있다.

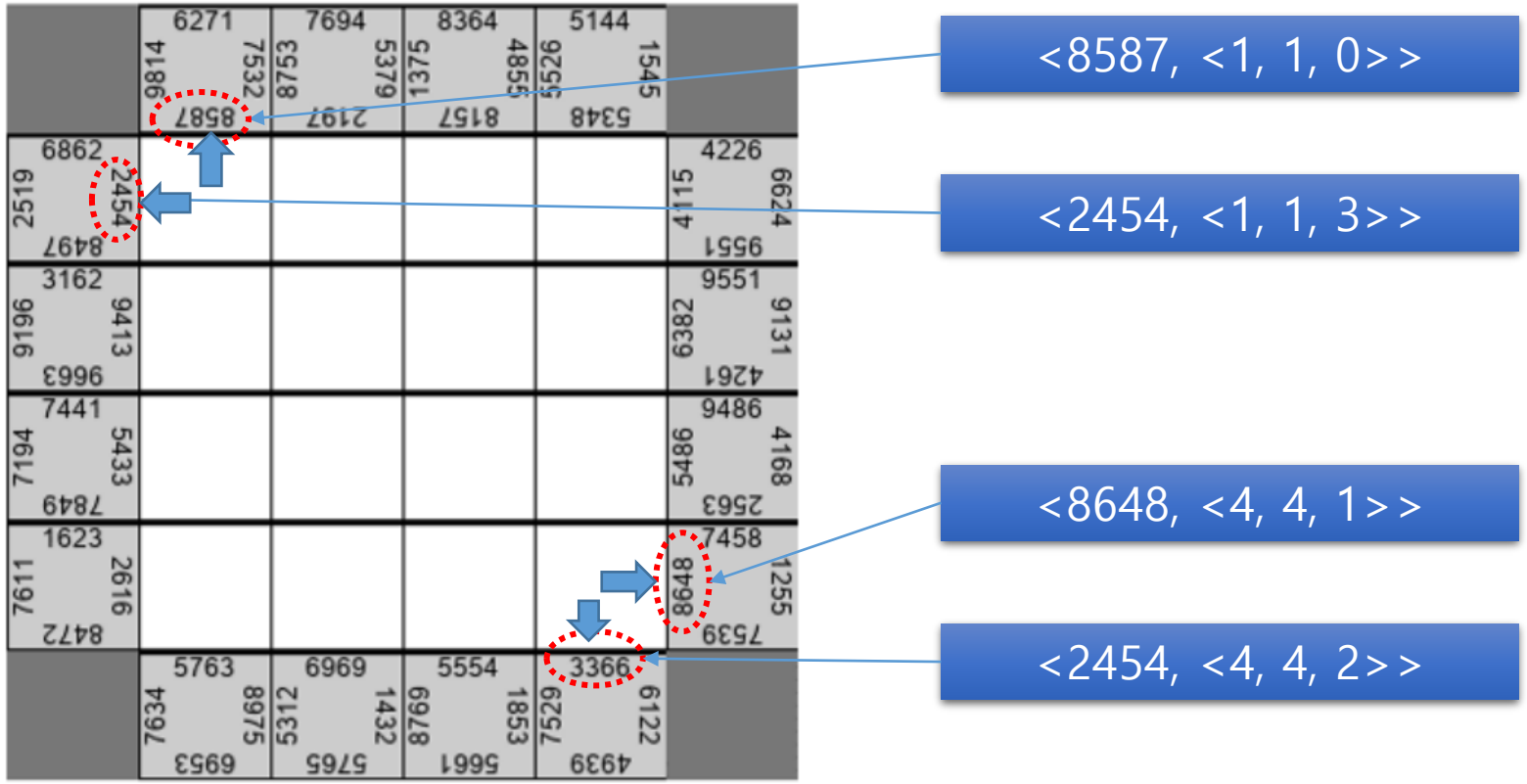
  이 경우 시간 복잡도는 다음과 같이 줄어든다.
  (퍼즐판에 놓인 변의 수) * (새로운 퍼즐의 변수) * (push()호출 수) * (TC수)
  = 1(평균) * 4 * 10,000 * 50 = 2,000,000 이상
  이 될 수도 있다.

- 방향에는 아래 그림과 같이 번호를 부여하자.



Solution sketch

- 아래 그림과 같이 퍼즐 조각들이 주어지는 경우
  새로운 퍼즐 조각이 놓일 위치를 변이 가진 값을 키로 , (행, 열, 방향)을 속성으로 저장할 수 있다. 방향은 새롭게 놓일 퍼즐에서 보는 것으로 할 수 있다.



Solution sketch

- 이를 위하여 unordered_map과 벡터를 사용할 수 있다.
  예시 코드는 다음과 같다.

```
struct Data {
    int r, c, d;
    bool operator<(const Data&t)const {  // 위쪽이 우선, 왼쪽이 차선
        return r == t.r ? c < t.c : r < t.r;
    }
};
unordered_map<int, vector<Data>> htab;  // <key(퍼즐한면의 값), (r(행), c(열), d(방향))목록>
```

Solution sketch

- 퍼즐의 짝을 찾은 경우 vector<Data> 항목에서 제거되거나
  탐색에 제외시킬 필요가 있다.

  우리는 찾아 지우기보다는 탐색에서 제외시키는 전략을 사용해보자.
  이를 위하여
  1) 퍼즐조각의 네 변 정보는 별도의 배열에 저장하고 배열 인덱스를 번호로 한다.
  2) 퍼즐판을 준비하여 퍼즐 조각 번호를 저장하는 전략을 사용할 수 있다.

```
const int LM = 1004;
int B[LM][LM];              // 보드판에 놓여진 퍼즐번호를 저장
int pn, prr[14005][4];  // pn : 퍼즐의 수, prr[][4] : 퍼즐 상태 정보
```

- 이제 각 함수 별 할 일을 정리해보자.

Solution sketch

void init(int N, int M, int mU[][4],int mR[][4],int mB[][4],int mL[][4])

- 해시 테이블, 퍼즐판, 퍼즐정보와 퍼즐수 등을 초기화 한다.

- 퍼즐판의 4방향 테두리 정보를 세팅한다.
  1. 퍼즐판에 퍼즐조각 번호를 저장한다.
  2. 퍼즐조각의 상태 정보를 저장한다.
  3. 새로운 퍼즐을 놓을 수 있는 경우라면 해시테이블에 정보를 저장한다.

- 퍼즐판의 테두리 상단 정보를 세팅하는 예시 코드는 다음과 같다.

```
for (i = 1; i <= N; ++i) { // 상단에 퍼즐 배치하기
    B[0][i] = ++pn;          // 퍼즐판에 퍼즐조각 번호 저장
    memcpy(prr[pn], mU[i - 1], sizeof(mU[i - 1])); // 퍼즐상태정보 저장
    // (1, i)에 새로운 퍼즐을 놓는 경우 0방향(↑)이 기준이다.
    htab[prr[pn][2]].push_back({ 1, i, 0 });
}
```

Solution sketch

int put(int mPiece[4])

- 퍼즐 조각 mPeice[]의 짝이 되는 변 정보를 int t[4]에 구한다.
- t[] 에는 각 변의 정보가 “역순으로 10의 보수가” 저장된다.
- 예시 코드는 다음과 같다.

```
int rev(int n, int rn = 0) { // 역순으로 10의 보수 구하기
    for (; n; n /= 10) rn = rn * 10 + (10 - n % 10);
    return rn;
}
int put(int mPiece[4])
{
    ...
    for (i = 0; i < 4; ++i) t[i] = rev(mPiece[i]);  // 변별 짝이 되는 코드 값 구하기
    ...
}
```

Solution sketch

int put(int mPiece[4]) (계속)

- t[]의 각 변 별로 맞는 짝을 찾아 본다.
- mPiece[] 조각이 놓일 경우 조각 주변 4방향을 모두 확인해야 한다.
  이웃한 변에 퍼즐조각이 있는데 짝이 맞지 않는 경우 하나라도 있다면 제외시킨다.
- mPiece[]퍼즐조각이 놓일 수 있는 곳이 여러 곳이라면 다음 우선순위를 따른다.
  1. 이웃한 변에 퍼즐조각이 가장 많은 곳이 우선이다.
  2. 1번이 같다면 퍼즐 판 위쪽에 놓는 것이 우선이다.
  3. 2번도 같다면 퍼즐 판 왼쪽에 놓는 것이 우선이다.

- mPiece[] 퍼즐조각을 놓을 곳을 찾았다면 회전을 고려하여 놓고
  이웃한 셀에 새로운 퍼즐이 놓일 수 있는지 검토한다.
  놓일 수 있다면 init()에서 했던 방법으로 해시 테이블에 추가한다.

Solution sketch

int put(int mPiece[4]) (계속)

- p위치에 –rot회전하여 퍼즐 조각을 놓을 때 결과를 구하는 함수의 예는 다음과 같다.

```
int getResult(Data&p, int rot) { // p위치에 ccw -rot번 회전하여 놓을 때 결과 구하기
    int mc = 0;
    for (int i = 0; i < 4; ++i) {
        int nr = p.r + dr[i], nc = p.c + dc[i];  // 주변 셀 좌표
        if (B[nr][nc] == 0) continue;            // 빈 셀인 경우
        int*pi = prr[B[nr][nc]];                 // 주변 셀에 놓인 퍼즐

        // t[]의 ((i + rot) % 4)번 값을 i번으로 가지고 와서 비교
        if (pi[i ^ 2] != t[(i + rot) & 3])
            return -1; // 맞지 않는 주변 퍼즐이 하나라도 있다면 놓을 수 없다.
        else
            mc++;      // 모양이 맞는 퍼즐이 나온 경우
    }
    return mc;
}
```

Solution sketch

int put(int mPiece[4]) (계속)

- t[]의 네 변과 맞닿는 변을 찾아 놓을 위치를 확인하는 코드는 다음과 같다.
  맞닿는 변이 놓인 방향에 맞게 회전하여 처리하지만 직접 회전하지 않고 처리한다.

```
rint i, meetCnt = 0, rot = 0;
for (i = 0; i < 4; ++i) t[i] = rev(mPiece[i]);  // 변별 짝이되는 코드값 구하기
    Data ret{ LM, LM, 0 };  //새로운 퍼즐 조각이 놓일 좌표

for (i = 0; i < 4; ++i) {
    if (htab.count(t[i]) == 0) continue;
    for (auto&v : htab[t[i]]) {
        if (B[v.r][v.c] == 0) {
            // v.d가 기준이므로 v.d번에 i번 값이 와야 한다.
            int mc = getResult(v, (i - v.d + 4) & 3);
            // 1.가장 많은 인접한 블록을 가진 2. 가장 상단 3. 가장 왼쪽
            if (mc > meetCnt || (mc == meetCnt && v < ret)) {
                meetCnt = mc, ret = v;
                rot = (i - v.d + 4) & 3; // 회전 가중치
            }
        }
    }
}
```

Solution sketch

int put(int mPiece[4]) (계속)

- ret위치에 놓는 것으로 확정된 경우 새로운 퍼즐이 놓일 곳을 다음과 같이 처리할 수 있다.

```
B[ret.r][ret.c] = ++pn;                    // 새로운 퍼즐조각 놓기
for (i = 0; i < 4; ++i) {
    // 회전시켜 가져오기:mPiece[]를 -rot회전시켜 저장하기
    prr[pn][i] = mPiece[(i + rot) & 3];
    int nr = ret.r + dr[i], nc = ret.c + dc[i];
    if (B[nr][nc] == 0) {                  // 새로운 퍼즐 주변의 빈 셀 등록
        // (nr, nc)입장에서 i^2방향에 변의 값이 prr[pn][i]인 새로운 조각이 놓임
        htab[prr[pn][i]].push_back({ nr, nc, i ^ 2 });
    }
}
```

Solution sketch

[Summary]

- 데이터를 적절히 분류하는 용도로 해시를 사용할 수 있다.

- 도형을 회전하는 상황에 적절히 대처할 수 있다.
  1. 직접 회전시켜보기
  2. 직접 회전시키지 않고도 회전한 것으로 처리하기

- 저장된 데이터를 직접 삭제하지 않고
  삭제 표시하여 적절히 처리할 수 있다.

Code example

Code example

TS큰퍼즐

```
#include <cstring>
#include <vector>
#include <unordered_map>
using namespace std;

#define rint register int
const int LM = 1004;
int B[LM][LM];                  // 보드판에 놓여진 퍼즐번호를 저장
int pn, prr[14005][4];          // 추가되는 퍼즐의 수는 init포함하여 14000개
int dr[] = { -1, 0, 1, 0 }, dc[] = { 0, 1, 0, -1 };  // clockwise
int t[4];                       // 새로운 블록의 대응하는 짝의 코드를 rev()로 생성하여 저장

int rev(int n, int rn = 0) { // 역순으로 10의 보수 구하기
    for (; n; n /= 10) rn = rn * 10 + (10 - n % 10);
    return rn;
}

struct Data {
    int r, c, d;
    bool operator<(const Data&t)const {
        return r == t.r ? c < t.c : r < t.r;
    }
};
unordered_map<int, vector<Data>> htab;  // <key(퍼즐한면의 값), (r(행), c(열), d(방향))목록>
```

Code example

TS큰퍼즐

```
void init(int N, int M, int mU[][4], int mR[][4], int mB[][4], int mL[][4])
{
    // 초기화
    rint i, j;
    for (i = 1; i <= N; ++i) for (j = 1; j <= N; ++j) B[i][j] = 0;
    htab.clear(), pn = 0;

    // 새로운 퍼즐판의 테두리 정보 셋팅
    for (i = 1; i <= N; ++i) { // 상단에 퍼즐 배치하기
        B[0][i] = ++pn;
        memcpy(prr[pn], mU[i - 1], sizeof(mU[i - 1]));
        // (1, i)에 새로운 퍼즐을 놓는 경우 0방향(↑)이 기준이다.
        htab[prr[pn][2]].push_back({ 1, i, 0 });
    }
    for (i = 1; i <= N; ++i) {
        B[i][N + 1] = ++pn;
        memcpy(prr[pn], mR[i - 1], sizeof(mR[i - 1]));
        // (1, N)에 새로운 퍼즐을 놓는 경우 1방향(→)이 기준이다.
        htab[prr[pn][3]].push_back({ i, N, 1 });
    }
```

Code example

TS큰퍼즐

```
    for (i = 1; i <= N; ++i) {
        B[N + 1][i] = ++pn;
        memcpy(prr[pn], mB[i - 1], sizeof(mB[i - 1]));
        // (N, i)에 새로운 퍼즐을 놓는 경우 2방향(↓)이 기준이다.
        htab[prr[pn][0]].push_back({ N, i, 2 });
    }
    for (i = 1; i <= N; ++i) {
        B[i][0] = ++pn;
        memcpy(prr[pn], mL[i - 1], sizeof(mL[i - 1]));
        // (i, 1)에 새로운 퍼즐을 놓는 경우 3방향(←)이 기준이다.
        htab[prr[pn][1]].push_back({ i, 1, 3 });
    }
}

void destroy() {}
```

Code example


```
int getResult(Data&p, int rot) { // p위치에 -rot회전하여 놓을때 결과 구하기
    int mc = 0;
    for (int i = 0; i < 4; ++i) {
        int nr = p.r + dr[i], nc = p.c + dc[i];  // 주변셀 좌표
        if (B[nr][nc] == 0) continue;            // 빈셀인 경우
        int*pi = prr[B[nr][nc]];                 // 주변셀에 놓인 퍼즐

        // t[]의 ((i + rot) % 4)번 값을 i번으로 가지고 와서 비교
        if (pi[i ^ 2] != t[(i + rot) & 3])
            return -1; // 주변의 퍼즐과 맞지 않는 경우가 하나라도 있다면 놓을 수 없다.
        else
            mc++;      // 모양이 맞는 주변 퍼즐 개수가 나온경우
    }
    return mc;
}
```

Code example

TS큰퍼즐

```
int put(int mPiece[4])
{
    rint i, meetCnt = 0, rot = 0;
    for (i = 0; i < 4; ++i) t[i] = rev(mPiece[i]);  // 변별 짝이되는 코드값 구하기
    Data ret{ LM, LM, 0 };  //새로운 퍼즐 조각이 놓일 좌표

    for (i = 0; i < 4; ++i) {
        if (htab.count(t[i]) == 0) continue;
        for (auto&v : htab[t[i]]) {
            if (B[v.r][v.c] == 0) {
                // v.d번에 i번값이 와야 한다. (v.d로 i를 찾아가야 한다.)
                int mc = getResult(v, (i - v.d + 4) & 3);
                // 1.가장 많은 인접한 블록을 가진 2. 가장 상단, 3. 가장 왼쪽
                if (mc > meetCnt || (mc == meetCnt && v < ret)) {
                    meetCnt = mc, ret = v;
                    rot = (i - v.d + 4) & 3; // 회전 가중치
                }
            }
        }
    }
```

Code example

TS큰퍼즐

```
    B[ret.r][ret.c] = ++pn;                    // 새로운 퍼즐조각 놓기
    for (i = 0; i < 4; ++i) {
        prr[pn][i] = mPiece[(i + rot) & 3]; // 회전시켜 가져오기 : mPiece[]를 -rot회전시키기
        int nr = ret.r + dr[i], nc = ret.c + dc[i];
        if (B[nr][nc] == 0) {                  // 새로운 퍼즐 주변의 빈 셀 등록
            // (nr, nc)입장에서 i^2방향에 변의 값이 prr[pn][i]인 새로운 조각이 놓임
            htab[prr[pn][i]].push_back({ nr, nc, i ^ 2 });
        }
    }
    return ret.r + ret.c; //새로운 퍼즐 조각이 놓인 좌표
}
```

Thank you.