문제 설명

땅따먹기 게임을 하려고 합니다. 땅따먹기 게임의 땅(land)은 총 N행 4열로 이루어져 있고, 모든 칸에는 점수가 쓰여 있습니다. 1행부터 땅을 밟으며 한 행씩 내려올 때, 각 행의 4칸 중 한 칸만 밟으면서 내려와야 합니다. **단, 땅따먹기 게임에는 한 행씩 내려올 때, 같은 열을 연속해서 밟을 수 없는 특수 규칙이 있습니다.**

예를 들면,

| 1 | 2 | 3 | 5 |

| 5 | 6 | 7 | 8 |

| 4 | 3 | 2 | 1 |

로 땅이 주어졌다면, 1행에서 네번째 칸 (5)를 밟았으면, 2행의 네번째 칸 (8)은 밟을 수 없습니다.

마지막 행까지 모두 내려왔을 때, 얻을 수 있는 점수의 최대값을 return하는 solution 함수를 완성해 주세요. 위 예의 경우, 1행의 네번째 칸 (5), 2행의 세번째 칸 (7), 3행의 첫번째 칸 (4) 땅을 밟아 16점이 최고점이 되므로 16을 return 하면 됩니다.

제한사항

* 행의 개수 N : 100,000 이하의 자연수
* 열의 개수는 4개이고, 땅(land)은 2차원 배열로 주어집니다.
* 점수 : 100 이하의 자연수

입출력 예

| **land** | **answer** |
| --- | --- |
| [[1,2,3,5],[5,6,7,8],[4,3,2,1]] | 16 |

입출력 예 설명

입출력 예 #1  
문제의 예시와 같습니다.

#include <iostream>

#include <vector>

#include<algorithm>

using namespace std;

int hopscotch(vector<vector<int> > land)

{

int pivot = 0;

int max = 0;

for (int j = 0; j < 4; j++)

{

if (max < land[pivot][j] + compare(land, pivot, j))

max = land[pivot][j] + compare(land, pivot, j);

}

return max;

}

int compare(vector<vector<int>> land, int row, int col)

{

int answer = 0;

int max = 0;

int tmpRow = 0;

int tmpCol = 0;

if (row + 1 == land.size() - 1)

{

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

{

if (i == col) continue;

if (max < land[row + 1][i])

max = land[row + 1][i];

}

return max;

}

for (int j = 0; j < 4; j++)

{

if (col == j) continue;

else

{

for (int k = 0; k < 4; k++)

{

if (k == j) continue;

else

{

if (max < land[row + 1][j] + land[row + 2][k])

{

max = land[row + 1][j] + land[row + 2][k];

tmpRow = row + 1;

tmpCol = j;

}

else if(max == land[row + 1][j] + land[row + 2][k])

{

if (row + 2 == land.size() - 1) continue;

if (land[row + 2][j] < land[row + 2][tmpCol])

tmpCol = j;

}

}

}

}

}

return land[tmpRow][tmpCol] + compare(land, tmpRow, tmpCol);

}

/\* 과정 설명

0번째 배열에서 밑으로 내려가면서 0 + 1 번째 배열에 값에 다음 0 + 2 배열의 값을 더해서 그중 큰 것을 골라준다

문제는 너무 많은 시간이 걸린다. 최적화를 찾아보자

\*/