Laberinto con Pista

Este es un problema de comunicación

Hay un laberinto en un tablero de tamaño $N \times N$. Algunas celdas están bloqueadas y otras están vacías. Todas las celdas del borde del tablero están bloqueadas. Te puedes mover por las celdas vacías. En cada momento, solo puedes ver el estado de las 4 celdas vecinas, y debes decidir a qué celda vecina moverte. Tu objetivo es llegar de la esquina inferior izquierda a la esquina superior derecha del laberinto, en la menor cantidad de movimientos posibles. Se garantiza que dichas dos celdas están vacías, y que existe un camino de celdas vacías que las une.
Además, tendrás la ayuda de Cesar. Al inicio del juego, Cesar te dará una pista inicial, que consiste de un arreglo binario. Cesar tendrá acceso al laberinto, y a partir de observarlo, generará una pista, que recibirás al inicio del juego.

Problema

Implementa dos funciones. Una que reciba la información del laberinto y genere una pista, y otra función que dada la pista, y solamente la información de los vecinos de la celda actual, traverse el laberinto desde la esquina inferior izquierda hasta la esquina superior derecha.

Detalles de Implementación

Implementa dos funciones. La primera función, $Pista()$, recibe una variable N, el tamaño del laberinto, y un vector de vectores, $Laberinto$, con la información de las celdas bloqueadas y vacías. Esta función debe regresar la pista, como un vector de booleanos. La segunda función, $Juego()$, recibe un vector de vectores la pista que generó el primer programa, y tiene acceso a la función $mover()$ para interactuar con el laberinto. Cuando llamas dicha función, debes indicar un parámetro, que detalla el tipo de movimiento. Los parámetros y procesos disponibles son los siguientes:

Parámetro	**Proceso**
$mover(-1)$	Se inicia el juego.
$mover(0)$	Te mueves de la casilla (x, y) a la casilla $(x, y+1)$.
$mover(1)$	Te mueves de la casilla (x, y) a la casilla $(x+1, y)$.
$mover(2)$	Te mueves de la casilla (x, y) a la casilla $(x, y-1)$.
$mover(3)$	Te mueves de la casilla (x, y) a la casilla $(x-1, y)$.
$mover(4)$	Se acaba el juego.

Cada vez que llames la función mover, el proceso puede ser exitoso o fallar. Si te moviste a una casilla vacía, o finalizaste el juego en la casilla adecuada el proceso es exitoso. En cualquier otro caso, el proceso falla y el jurado mata el programa, y recibes 0 puntos en el caso que se esté procesando.

independientemente del parámetro, la función regresará un arreglo de 4 booleanos. LLamemosle a. $a[i]$ es verdadero si $mover(i)$ sería un movimiento válido.
Lo siguiente es un template del programa que debes implementar:

```
#include "Laberinto.h"
using namespace std;
vector<bool> Pista(int N, vector<vector<bool>> Laberinto) {
    vector<bool> Pista;
    // Programa esta función.
    return pista;
}
void Juego(vector<bool> Pista) {
    // Programa esta función.
}
```

Subtareas

Digamos que la cantidad de Llamadas a la función $mover()$ es M, y el tamaño de la pista P.

- (5 puntos) No hay ningún límite sobre M o P.
- (15 puntos) Los valores de M y P deben ser mínimos para el laberinto dado. Se garantiza que no existe una celda vacía con cuatro celdas vecinas vacías.
- (20 puntos) El valor de M debe ser mínimo para el laberinto dado. No hay restricciones respecto a P.
- (60 puntos)
  - Para obtener los 60 puntos, los valores de M y P deben ser mínimos para el laberinto dado.
  - Cualquier solución que tenga un P mayor, será normalizada entre el valor óptimo y $2M$.