24 Interstellar

21 points

Introduction

Do you like solving mazes? We are sure you don't, they are too easy to solve, and they can be solved by mice! But this is in part because we make them too simple, we make them in two dimensions to be able to draw them on a 2D surface. But universe is 3D, well, or 4D if we include time.
So we need to write a program to solve a 4D or space-time maze, but remember the fourth dimension is time and time dimension has something special in our universe, you cannot go back in time. Of course, you can change the speed while moving in this space-time maze, this means that when in a specific time slice you can move as much as you want through the 3D space maze or even not move at all before you jump to the next time slice. Traveling in diagonal is not allowed, neither in space, nor in time.
The maze may have multiple solutions, but you need to return the solution that is the shortest in the number of required movements in space.

Input

The first line will be the size of the space time of our maze: Nx, Ny, Nz, Nt, all of them are positive integers. All space-time values are integers and start by 0, if Nx = 2, for example, possible x positions are 0, 1 and 2. Second line is the starting point: Sx, Sy, Sz, St. St will be always 0, start of time. Third line is the ending point: Ex, Ey, Ez, Et. Where Et = Nt – 1, end of time.
After this, there are Nx * Ny * Nz rows, each row start by 3 numbers than are the x, y, z coordinates of one point in space and later there are Nt values representing the evolution in time of this point in the maze, possible values are an asterisk (*) for a wall you can´t cross or an underscore (_) for a space.

Output

Output the number of steps followed by the path through space-time in x, y, z, t coordinates.

Input example:

```
3 2 3 4
0 1 1 0
1 1 0 3
0 0 0 **_*
0 0 1 ___*
0 0 2 ___*
0 1 0 *__*
0 1 1 ___*
0 1 2 __**
1 0 0 ****
1 0 1 ****
1 0 2 *___
1 1 0 **__
1 1 1 *_**
1 1 2 ****
2 0 0 ****
2 0 1 ****
2 0 2 ****
2 1 0 **_*
2 1 1 __**
2 1 2 ****
```

Output Example:

```
6
0 1 1 0
0 1 1 1
0 1 1 2
0 1 0 2
1 1 0 2
1 1 0 3
```