Overview. Use clingo to implement a “path finding” program for simple 2-dimensional grids with obstacles and blocked-out locations. Given a start and finish location (where a location is denoted by an x-y coordinate in the grid), the goal is to compute paths through the grid that begin at the start location and end at the finish location. Each move on a path is made between adjacent grid positions and requires an action that incurs a cost. The default action when no obstacle exists between two grid locations is to walk, which incurs a cost of 1. The size of the grid, the blocked grid locations, and the obstacles, actions, and costs are given as input to the program. The following is a visual example of a 2 × 2 grid input that contains:

- one blocked position at location (0,2)
- a start position at location (0,0)
- a finish position at location (1,2)
- water between (0,0) and (0,1) requiring swimming at a cost of 4; a fence between (1,0) and (1,1) requiring climbing at a cost of 2; brush between (1,1) and (1,2) requiring clearing at a cost of 3; and a wall between (1,1) and (2,1) that requires digging under at a cost of 5

```
% the value of n denotes the (n x n) grid height and width
#const n = 2.
% start and finish locations
start(at(0,0)).
finish(at(1,2)).
% blocked locations
blocked(at(0,2)).
% actions, obstacles, and costs
cost(swim, water, 4).
cost(climb, fence, 2).
cost(clear, brush, 3).
cost(dig, wall, 5).
% location of obstacles on the grid
object(at(0,0), at(0,1), water).
object(at(1,0), at(1,1), fence).
object(at(1,1), at(1,2), brush).
object(at(1,1), at(2,1), wall).
```

This corresponding grid input for the example would be given as follows.
Instructions. Do steps 1–6 below. Note that each program variant must adhere to the following.

- A path consists of a set of moves of the form \texttt{move(L1,L2,A,C)} stating that a move is made from location \texttt{L1} to location \texttt{L2} requiring action \texttt{A} at a cost of \texttt{C}. Locations \texttt{L1} and \texttt{L2} must be adjacent to each other in the grid.

- A path cannot contain a move either to or from a blocked location.

- A path cannot contain a loop.

- A path should not contain a move that leaves the finish location.

- If an obstacle is between two locations, moving from one location to the other in either direction incurs the corresponding cost.

1. Write a program that finds all possible paths. Each path should be output as its corresponding \texttt{move} relations. (In other words you only need to “show” the \texttt{move} predicate).

2. Write a program that finds the path with the smallest total path cost. The path cost should be calculated as the sum of each move cost along the path. You must use a “minimize” weak constraint in \texttt{clingo} to find the path with the smallest cost. In particular, if you store the path cost as \texttt{path\_cost(C)}, then the following statement will return the answer set with the smallest path (given as the “optimization” result).

\[
\texttt{#minimize \{C : path\_cost(C)\}.}
\]

In this case, the \texttt{C} before the colon denotes the weight that is being minimized such that the answer set with the smallest such weight is the optimized result. Your program should output both the path (i.e., the moves on the path) and the path cost.

3. Similar to question 2, write a program that outputs the path with the smallest number of moves. Your program should print out both the moves and the corresponding path length.

4. Write a program that outputs the path with the smallest overall “volume”, where the volume of a path is its length times its total cost. Note that you can include arithmetic operations in a minimize statements weight expression. Your program should print out the moves, the length, and the cost.

5. Come up with your own criteria for defining what constitutes a “good” path and then write a program to find such paths. You can define your own weak constraint or add new constraints to your program. Describe your metric and how you implemented it.

6. Come up with a test suite of grids with different types of obstacles and run your programs to see the results. Be sure to try different grid sizes (beyond $2 \times 2$ grids).

To Turn In: Hand in hardcopy of 1–6 above with output of tests showing your program works correctly, a discussion of issues and challenges, and a cover page. Also submit your program to the online submission system.