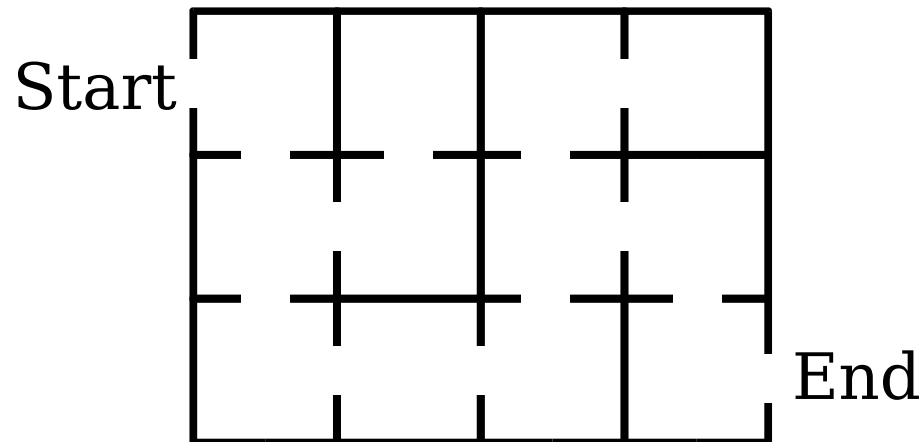


Backtracking Algorithms

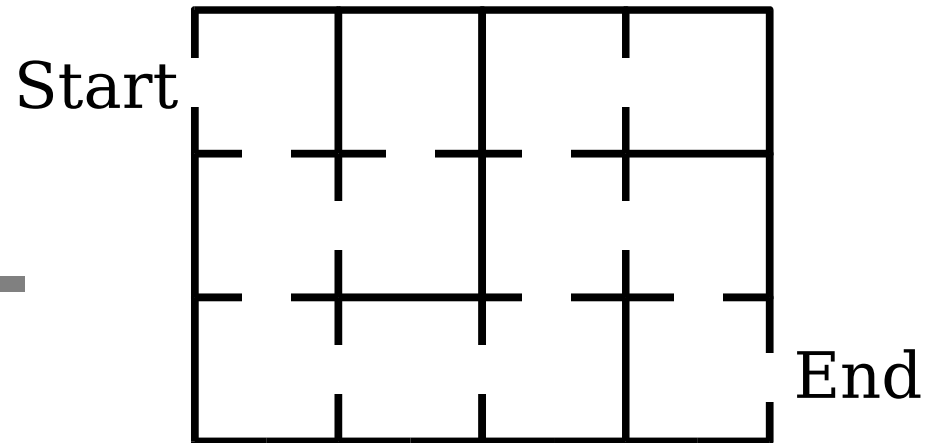
- Two situations:
 - Finding a solution to a problem can't be based on a *straight path* to the goal.
 - consider traversing a maze.
 - We need a better approach than brute force (independently evaluating all possible solutions).
 - Think of the TSP problem – many possible solutions share partial tours (why not treat identical partial tours as a single partial solution?).

Solving a maze

- Consider the following 2-D maze:

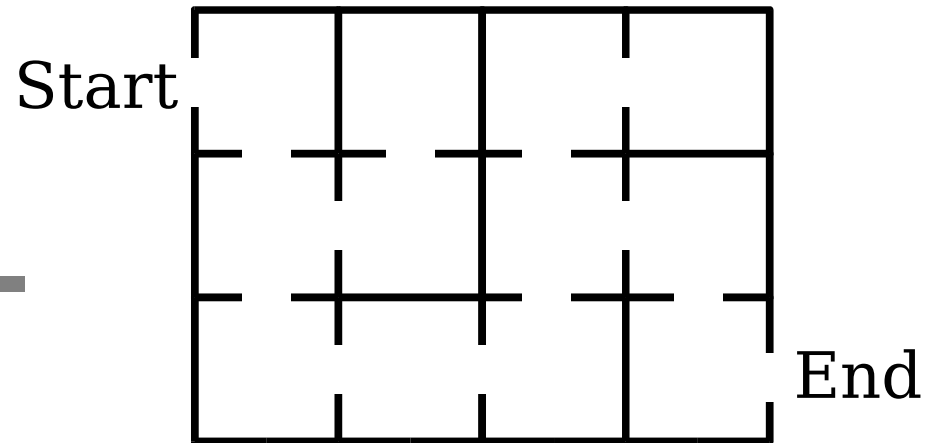


Maze Traversal



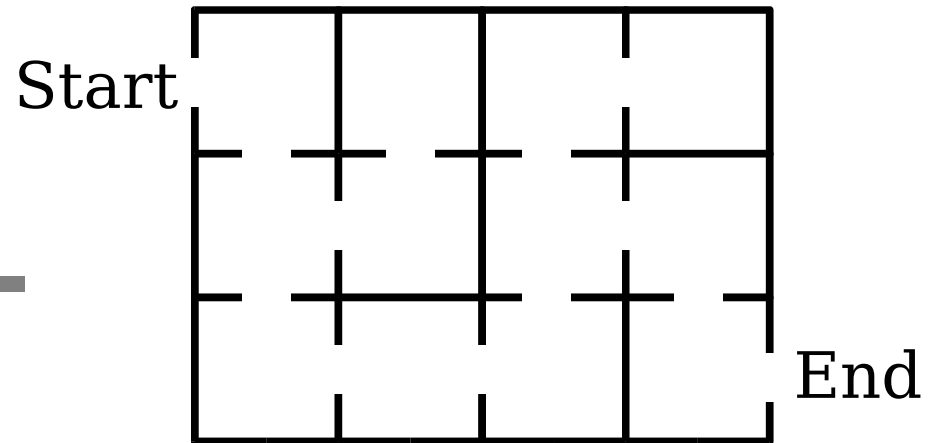
- 4x3 grid of *cells*
 - name the cells by row and column
 - cell_{0,0} is the start, and cell_{2,3} is the end.
- When in a cell, movement can be any of
 - up down left right.
 - some directions may be blocked.

Brute Force



- Try all possible sequences of moves:
 - most will fail (be blocked)
 - what does "all possible sequences" mean?
 - how many moves
 - need to avoid loops
 - generating *all possible sequences* may not be possible

Backtracking



- Rephrase the problem as finding a path from any cell $cell_{r,c}$ to the end.
- Given a starting point:
 - move up from $cell_{r,c}$ if possible. Try to solve (recursively from new starting point. If solution is found we are done.
 - move down from $cell_{r,c}$ if possible ... (then left, right)

Avoiding Loops

- The strategy described has a problem:
 - move from $cell_{r,c}$ up to $cell_{r-1,c}$.
 - when trying to solve from $cell_{r-1,c}$
 - we must avoid moving *back* to $cell_{r,c}$
 - this would result in an infinite loop...
- Idea: keep track of cells already visited.

Revised strategy

- Given a starting point $cell_{r,c}$:
 - if $cell_{r,c}$ has already been visited, report failure.
 - move up from $cell_{r,c}$ if possible. Try to solve (recursively from new starting point. If solution is found we are done.
 - move down from $cell_{r,c}$ if possible ... (then left, right)G

Possible Scheme Encoding of Maze

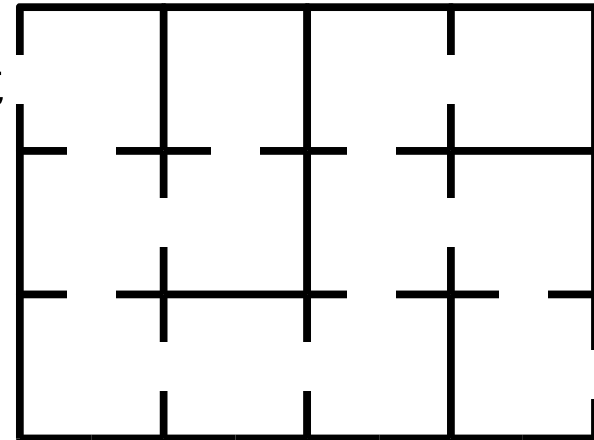
- For each cell, create a list that describes which moves are blocked:

```
' ( false true false false )  
    ↑   ↑   ↑   ↑  
    up  down left right
```

- We need a list of these lists (one for each cell).
- Arrange the lists to mimic the structure of the grid.

Encoding

Start



End

```
' ( (false true false false)
    (false true false false)
    (false true false true)
    (false false true false))
((true true false true)
 (true false true false)
 (true true false true)
 (false true true false))
((true false false true) (false false true true)
 (true false true false) (true false true true))
```

Exercise

- Play with "maze.scm" which can solve mazes.
- Data definitions:
 - cell is a row and column (like a posn).
 - maze is defined using lists of 'true, 'false.
 - as described in previous slide
 - solution is a list of cells (in order visited)
- Create your own 3x3 maze

TSP backtracking

- Partial solution is list of cities from the start along with minimum total cost found so far.
- *pruning*: use best solution found so far to eliminate some choices –
 - if smallest cost from current city to unvisited city results would result in a tour cost greater than best found so far – no need to consider any more tours from this city.

N Queens

- Chess Problem:
 - given $N \times N$ chess board, place N queens on the board so that no two queens threaten each other.
 - A queen threatens any piece on the same row, column or diagonal as the queen.
- Backtracking:
 - place a queen on the board, see if a solution can be found (by placing the rest of the queens recursively). If not – find a different place for the first queen.

N-Queens Exercise

- Try to solve a 4x4 N-Queens, here is one solution:

