Backtracking

Comp Sci 1575 Data Structures

MISSOURI S&T | Computer Science

Recursive backtracking

Life can only be understood backwards;
but it must be lived forwards.
-Soren Kierkegaard

Outline

1. Introduction
    - Problem: Hindsight
    - Solution: Backtracking
    - General procedure
    - General pseudocode

2. Examples
    - Sudoku
    - Mazes

Outline

1 Introduction
- Problem: Hindsight
- Solution: Backtracking
- General procedure
- General pseudocode

2 Examples
- Sudoku
- Mazes

Problem: fixing your past mistakes

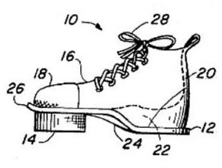
- You are faced with repeated sequences of options and must choose one each step
- After you make your choice you will get a new set of options.
- Which set of options you get depends on which choice you made.
- Procedure is repeated until you reach a final state.
- If you made a good sequence of choices, your final state may be a goal state.
- If you didn't, you can go back and try again

For example:
Games such as, n-Queens, Knapsack problem, Sudoku, Maze, etc

Outline

1. Introduction
   - Problem: Hindsight
   - Solution: Backtracking
   - General procedure
   - General pseudocode
2. Examples
   - Sudoku
   - Mazes

Solution: backtracking



- General meta-heuristic that incrementally builds candidate solutions by a sequence of candidate extension steps, one at a time, and abandons each partial candidate, c, (by backtracking) as soon as it determines that c cannot possibly be extended to a valid solution.
- Can be completed in various ways to give all the possible solutions to the given problem.
- Can be implemented with a form of recursion or stacks
- If at some step it becomes clear that the current path that you are on can't lead to a solution, you go back to the previous step (backtrack) and choose a different path.

Outline

1 Introduction
- Problem: Hindsight
- Solution: Backtracking
- General procedure
- General pseudocode

2 Examples
- Sudoku
- Mazes

General procedure

```
Pick a starting point.
while(Problem is not solved)
  For each path from the starting point.
    check if selected path is valid,
    if yes
      select it
      and make recursive call to rest of problem
    if recursive calls returns true, then
      return true.
    else
      undo the current move and
      return false.
  If none of the moves work out,
    return false, NO SOLUTION.
```

Outline

1 Introduction
- Problem: Hindsight
- Solution: Backtracking
- General procedure
- General pseudocode

2 Examples
- Sudoku
- Mazes

With data, P, call backtrack(root(P))

- root(P): return partial candidate at root of search tree.
- reject(P,c): return true only if partial candidate c is not worth completing.
- accept(P,c): return true if c is a solution of P, and false otherwise.
- first(P,c): generate the first extension of candidate c.
- next(P,s): generate next alternative extension of a candidate, after extension s.
- output(P,c): use solution c of P, as appropriate to application.

```
procedure backtrack(c)
  if reject(P,c) then return
  if accept(P,c) then output(P,c)
  s = first(P,c)
  while s is not NULL do
    backtrack(s)
    s = next(P,s)

backtrack(root(P))
```

Outline

1 Introduction
- Problem: Hindsight
- Solution: Backtracking
- General procedure
- General pseudocode

2 Examples
- Sudoku
- Mazes

Outline

1. Introduction
   - Problem: Hindsight
   - Solution: Backtracking
   - General procedure
   - General pseudocode
2. Examples
   - Sudoku
   - Mazes

Sudoku: decompose into smaller problem?

Start

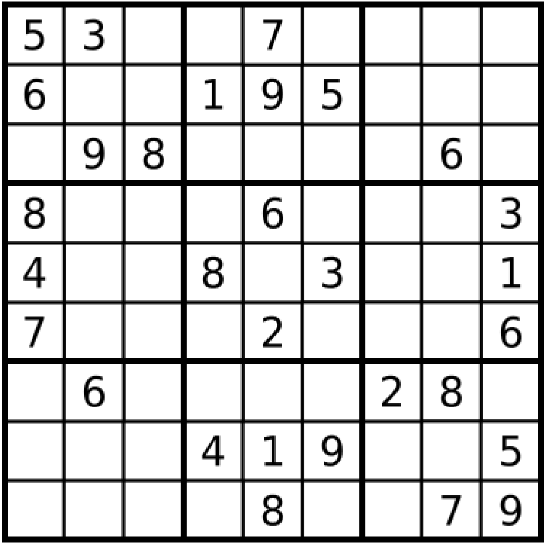
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Finish

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

- 81 cells, in a 9 by 9 grid, with 9 zones, each zone being the intersection of the first, middle, or last 3 rows, and the first, middle, or last 3 columns.
- Each cell may contain a number from one to nine; each number can only occur once in each zone, row, and column of the grid.
- At the beginning of the game, some cells begin with numbers in them, and the goal is to fill in the remaining cells.

Sudoku strategies?



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

How might we solve this non-recursively?

Sudoku: decompose into smaller problem?

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

What is our pseudocode for a recursive solution?

Which functions do we need?

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

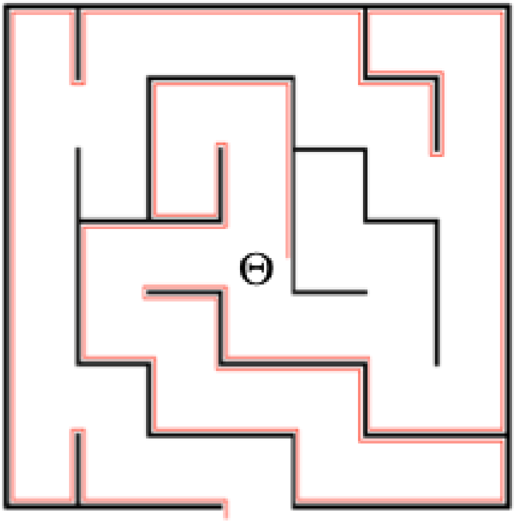
```
Find row, col of an unassigned cell
If there are none, return true
For digits from 1 to 9
  a) If no conflict for digit at row,col
       assign digit to row,col
       and recursively try to fill rest of grid
  b) If recursion successful, return true
  c) Else, remove digit and try another
If all digits were tried and nothing worked,
  return false
```

Outline

1. Introduction
   - Problem: Hindsight
   - Solution: Backtracking
   - General procedure
   - General pseudocode
2. Examples
   - Sudoku
   - Mazes

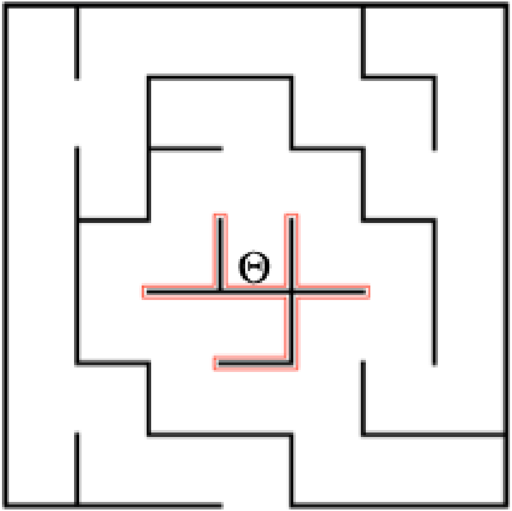
General rules to solve a maze?

Right hand rule? Start in center, and try to escape.



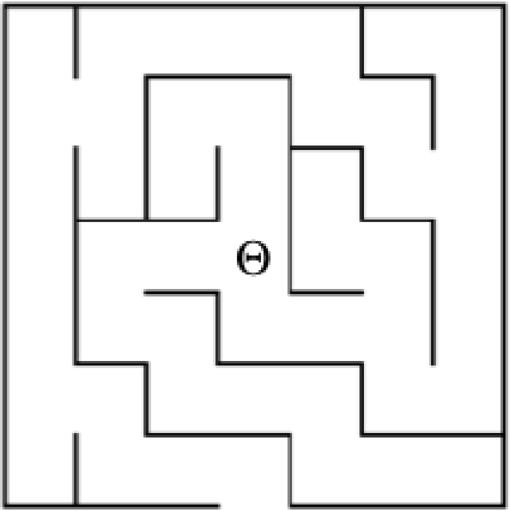
General rules to solve a maze?

Right hand rule doesn't work with this kind of loop

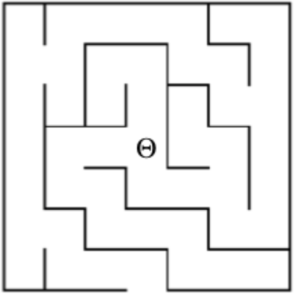


Recursive maze-finding

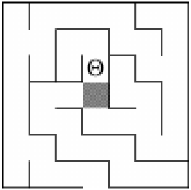
Is there a smaller version of a maze problem?

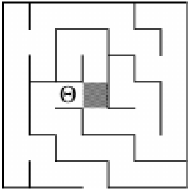


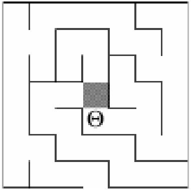
Recursive maze-finding



Is there a smaller version of a maze problem?







Recursive pseudocode: maze with prize at center

Goal: start outside of maze, obtain prize, find your way out

```
Face left, the first direction to explore
For i in each of the three directions
  if(!found && direction being faced is not a dead end)
    then explore the direction now being faced,
    returning to this exact spot after the exploration,
    and setting found to true.
  Turn 90 degrees right
Step forward, turn around
```