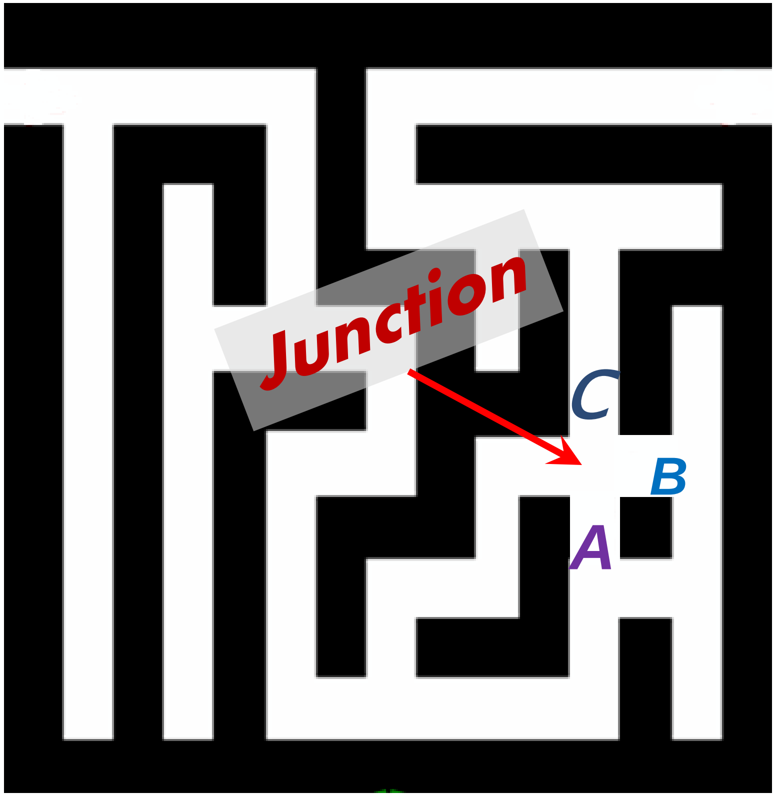
CS312-Computing Algorithms

Backtracking

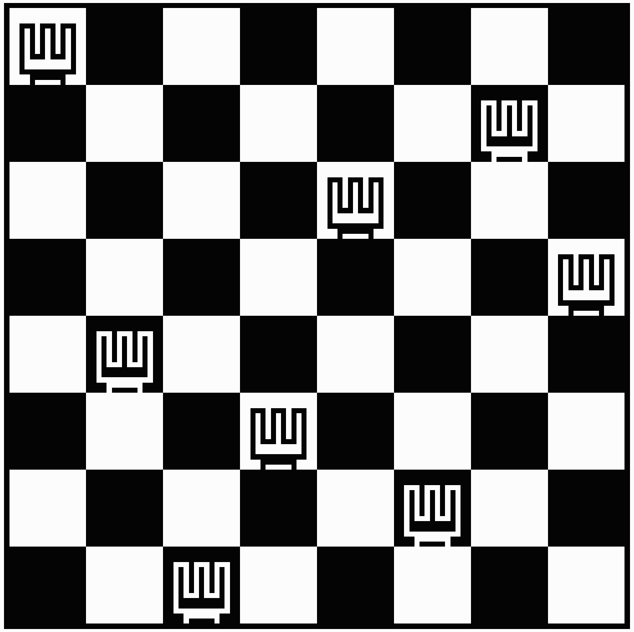
Backtracking

- Clearly, at a single junction you could have even more than 2 choices.
- The backtracking strategy says to try each choice, one after the other,
  - if you ever get stuck, ***"backtrack"*** to the junction and try the next choice.
- If you try all choices and never found a way out, then there IS no solution to the maze.



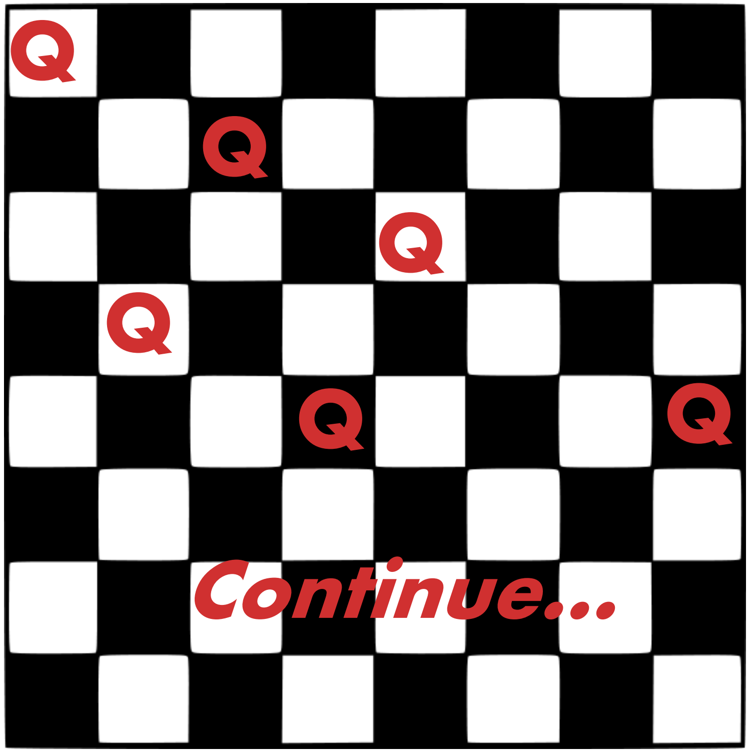
Backtracking – Eight Queens Problem

- Find an arrangement of **8** queens on a single chess board such that no two queens are attacking one another.
- In chess, queens can move all the way down any row, column or diagonal (so long as no pieces are in the way).
  - Due to the first two restrictions, it's clear that each row and column of the board will have exactly one queen.



Backtracking – Eight Queens Problem

- The backtracking strategy is as follows:
  1) Place a queen on the first available square in row 1.
  2) Move onto the next row, placing a queen on the first available square there (that doesn't conflict with the previously placed queens).
  3) Continue in this fashion until either:
     a) you have solved the problem, or
     b) you get stuck.
        - When you get stuck, remove the queens that got you there, until you get to a row where there is another valid square to try.

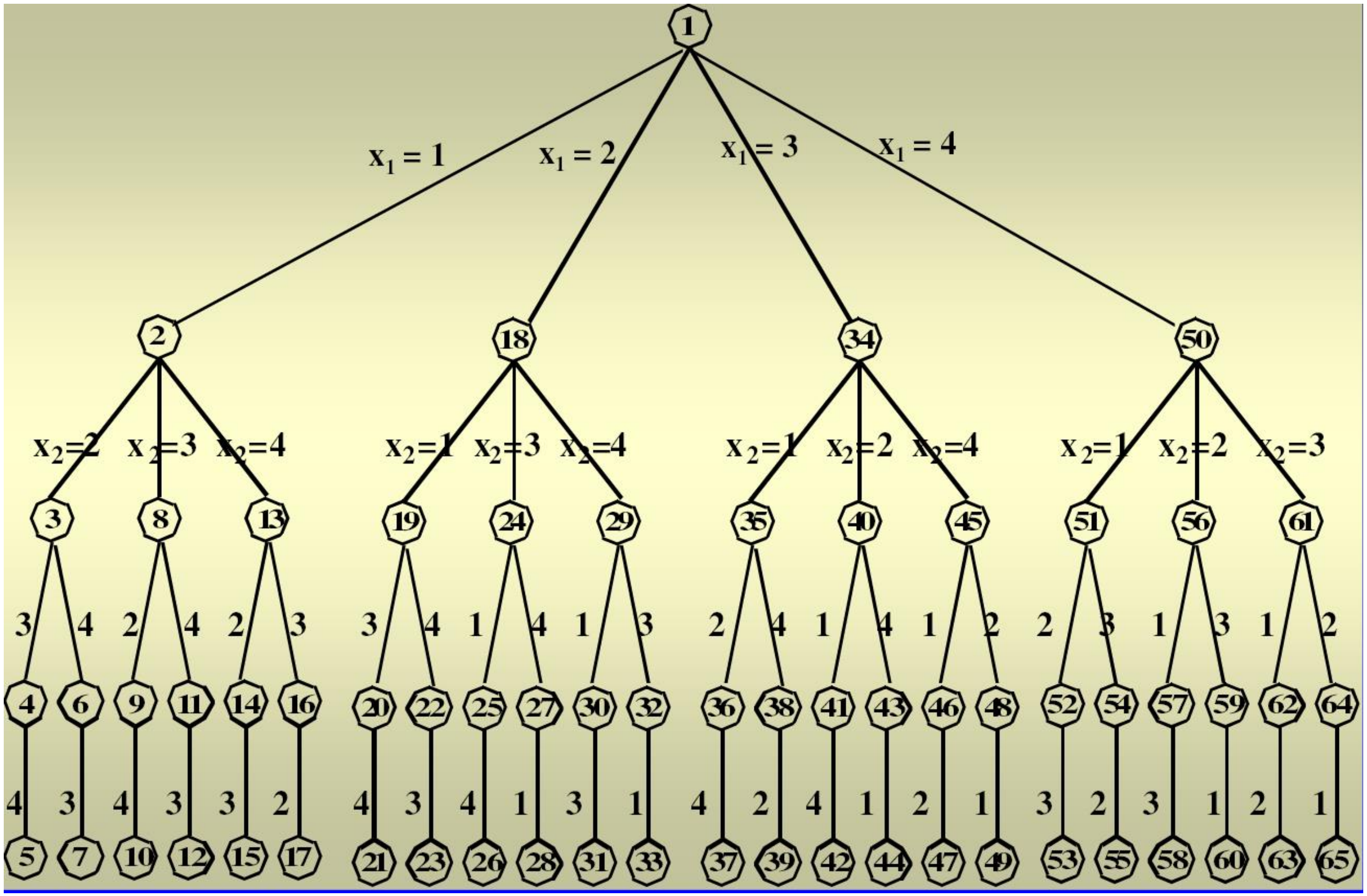


Backtracking – Eight Queens Problem

- When we carry out backtracking, an easy way to visualize what is going on is a tree that shows all the different possibilities that have been tried.
- On the board we will show a visual representation of solving the 4 Queens problem (placing 4 queens on a 4x4 board where no two attack one another).

Backtracking – Eight Queens Problem

- The neat thing about coding up backtracking, is that it can be done recursively, without having to do all the bookkeeping at once.
  - Instead, the stack or recursive calls does most of the bookkeeping
  - (ie, keeping track of which queens we've placed, and which combinations we've tried so far, etc.)


1
x1 = 1
x1 = 2
x1 = 3
x1 = 4
2
18
34
50
x2=2 x2=3 x2=4
x2=1 x2=3 x2=4
x2=1 x2=2 x2=4
x2=1 x2=2 x2=3
3 8 13 19 24 29 35 40 45 51 56 61
3 4 2 4 2 3 3 4 1 4 1 3 2 4 1 4 1 2 2 3 1 3 1 2
4 6 9 11 14 16 20 22 25 27 30 32 36 38 41 43 46 48 52 54 57 59 62 64
4 3 4 3 3 2 4 3 4 1 3 1 4 2 4 1 2 1 3 2 3 1 2 1
5 7 10 12 15 17 21 23 26 28 31 33 37 39 42 44 47 49 53 55 58 60 63 65

Backtracking – 8 queens problem – Analysis

- Another possible brute-force algorithm is generate the permutations of the numbers 1 through 8 (of which there are 8! = 40,320),
  - and uses the elements of each permutation as indices to place a queen on each row.
  - Then it rejects those boards with diagonal attacking positions.

- The backtracking algorithm, is a slight improvement on the permutation method,
  - constructs the search tree by considering one row of the board at a time, eliminating most non-solution board positions at a very early stage in their construction.
  - Because it rejects row and diagonal attacks even on incomplete boards, it examines only 15,720 possible queen placements.

- A further improvement which examines only 5,508 possible queen placements is to combine the permutation based method with the early pruning method:
  - The permutations are generated depth-first, and the search space is pruned if the partial permutation produces a diagonal attack

Sudoku and Backtracking

- Another common puzzle that can be solved by backtracking is a Sudoku puzzle.

- The basic idea behind the solution is as follows:
  1) Scan the board to look for an empty square that could take on the fewest possible values based on the simple game constraints.
  2) If you find a square that can only be one possible value, fill it in with that one value and continue the algorithm.
  3) If no such square exists, place one of the possible numbers for that square in the number and repeat the process.
  4) If you ever get stuck, erase the last number placed and see if there are other possible choices for that slot and try those next.

Mazes and Backtracking

- A final example of something that can be solved using backtracking is a maze.
  - From your start point, you will iterate through each possible starting move.
  - From there, you recursively move forward.
  - If you ever get stuck, the recursion takes you back to where you were, and you try the next possible move.

- In dealing with a maze, to make sure you don't try too many possibilities,
  - one should mark which locations in the maze have been visited already so that no location in the maze gets visited twice.
  - (If a place has already been visited, there is no point in trying to reach the end of the maze from there again.

Questions?