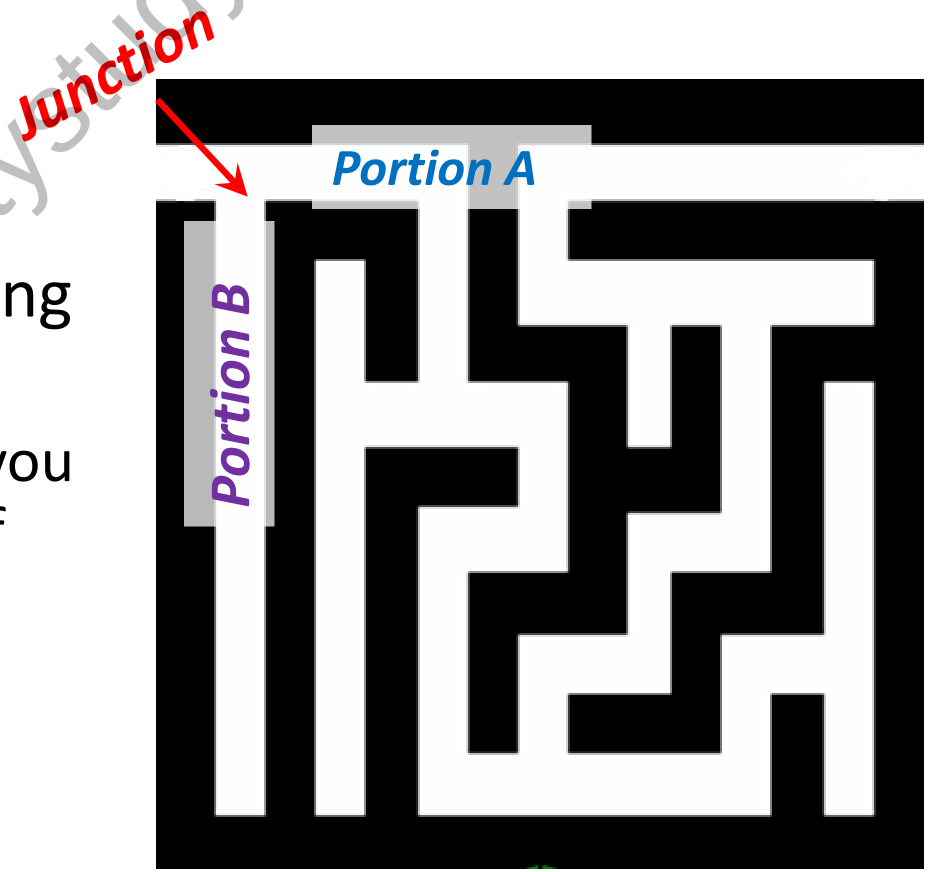
Back tracking Algorithm

Backtracking

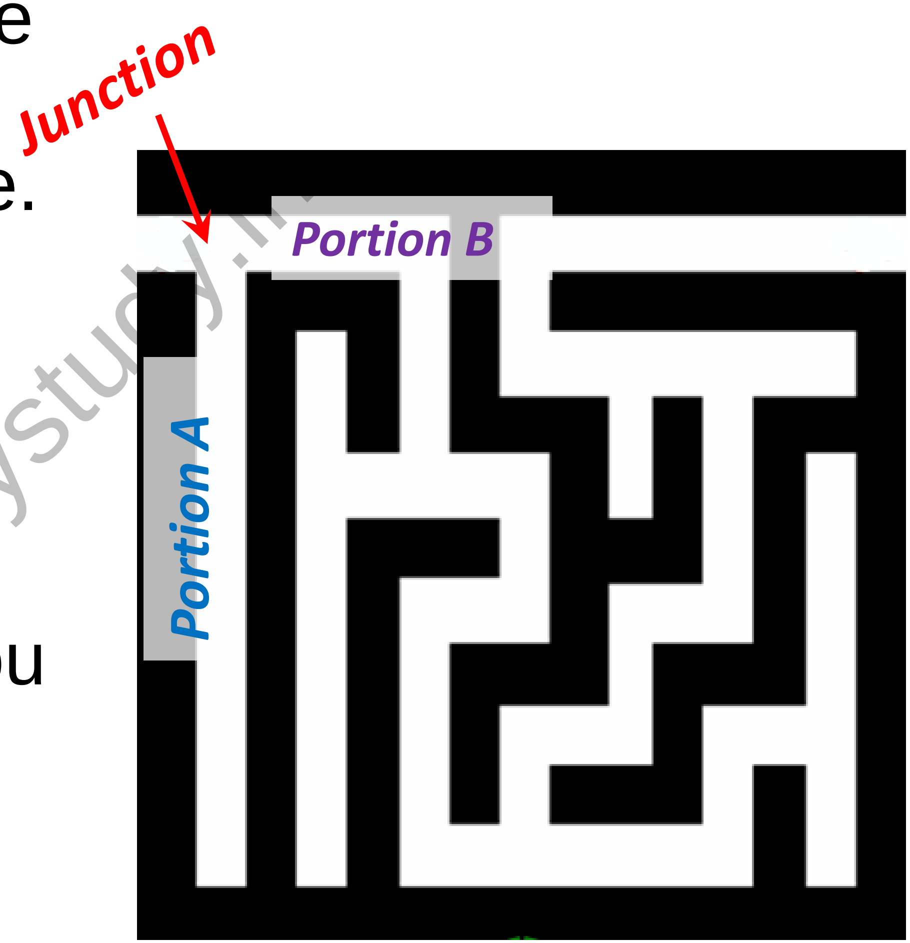
- Backtracking is a technique used to solve problems with a large search space, by systematically trying and eliminating possibilities.

- A standard example of backtracking would be going through a maze.
  - At some point in a maze, you might have two options of which direction to go:



Backtracking

- One strategy would be to try going through **Portion A** of the maze.
  - If you get stuck before you find your way out, then you ***"backtrack"*** to the junction.
- At this point in time you know that **Portion A** will ***NOT*** lead you out of the maze,
  - so you then start searching in **Portion B**

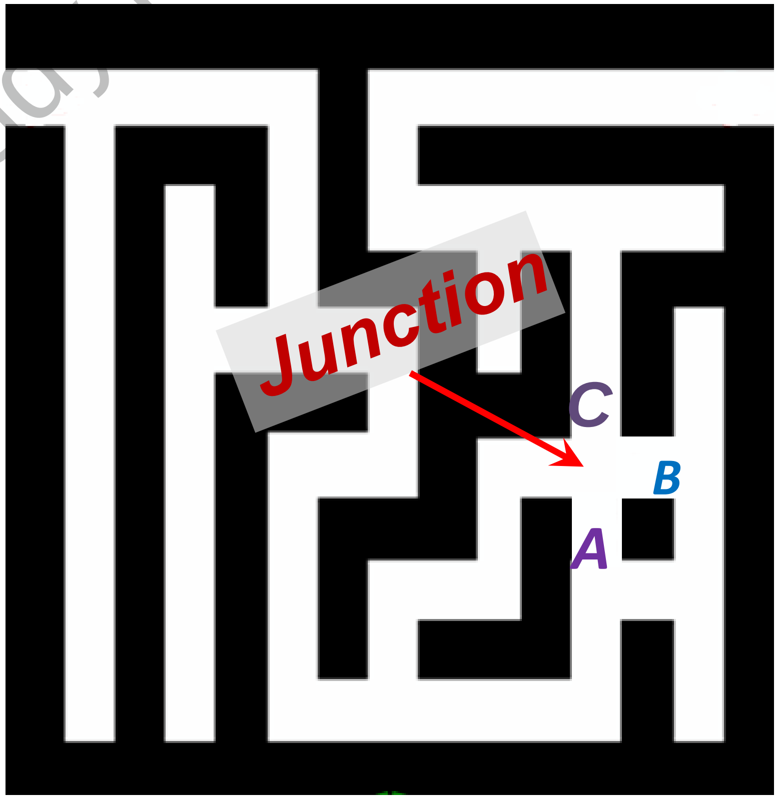


Backtracking

- 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.

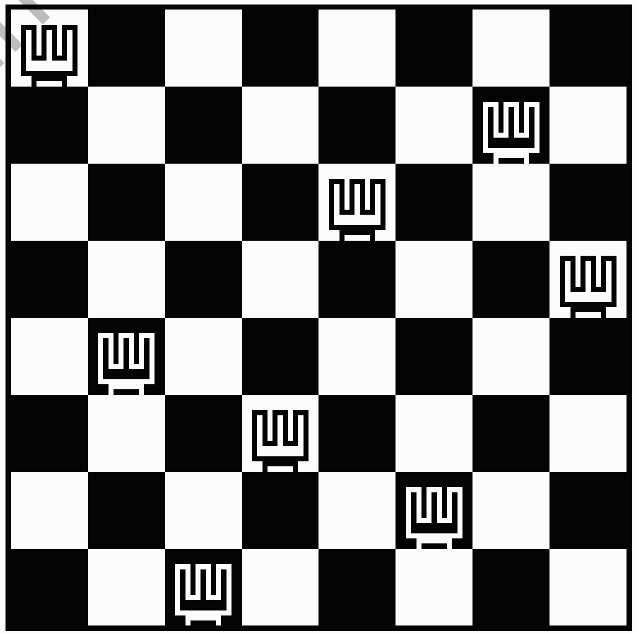
- If you try all choices and never found a way out, then there IS no solution to the maze.



Backtracking – Eight Queens Problem

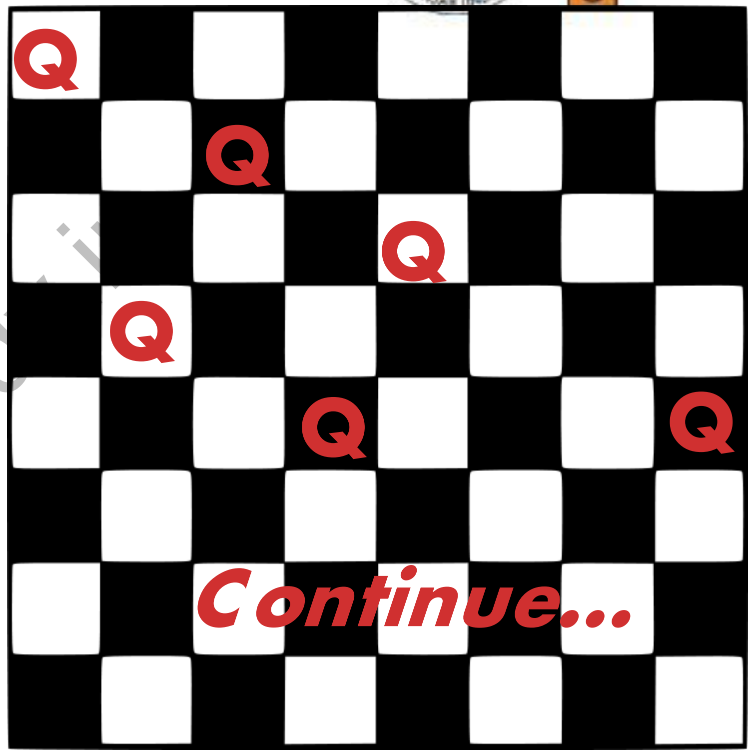
- Find an arrangement of **8** queens on a single chess board such that no two queens are attacking one another.

- 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.



Animated Example:
http://www.hbmeyer.de/backtrack/achtdamen/eight.htm#up

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.)

perm[] - stores a valid permutation of queens from index 0 to location-1.
location – the column we are placing the next queen
usedList[] – keeps track of the rows in which the queens have already been placed.

```
void solveItRec(int perm[], int location, struct onesquare usedList[])  {

  if (location == SIZE) {
    printSol(perm);
  }
 for (int i=0; i<SIZE; i++) {

    if (usedList[i] == false) {

       if (!conflict(perm, location, i)) {

           perm[location] = i;
           usedList[i] = true;
           solveItRec(perm, location+1, usedList);
           usedList[i] = false;
        }
      }
   }
}
```

Found a solution to the problem, so print it!

Loop through possible rows to place this queen.

Only try this row if it hasn't been used

Check if this position conflicts with any previous queens on the diagonal

1) mark the queen in this row
2) mark the row as used
3) solve the next column location recursively
4) un-mark the row as used, so we can get ALL possible valid solutions.

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

Thank You !!!