



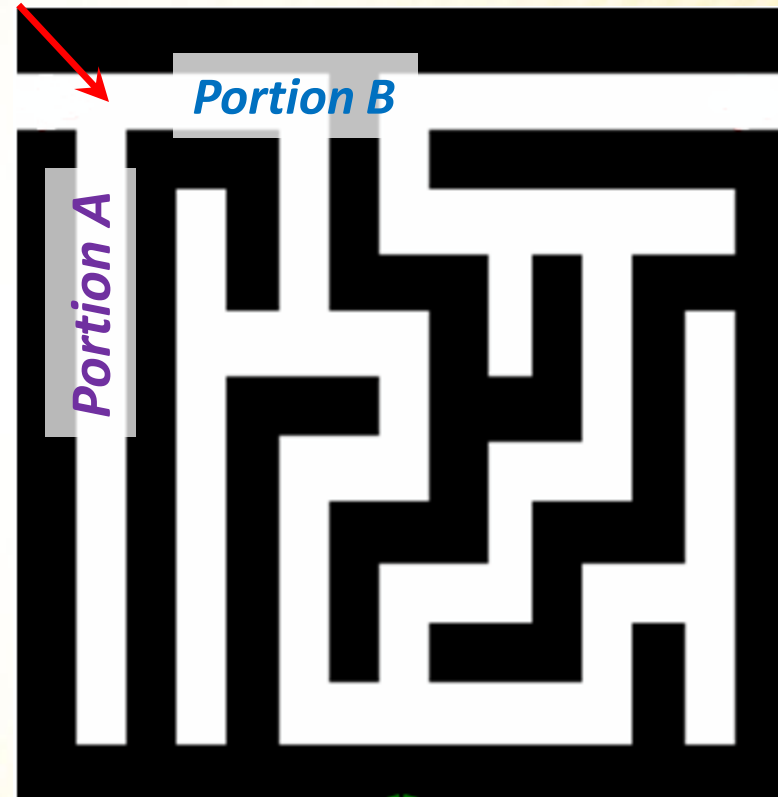
# **BACKTRACKING**

COP 3502

# 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:

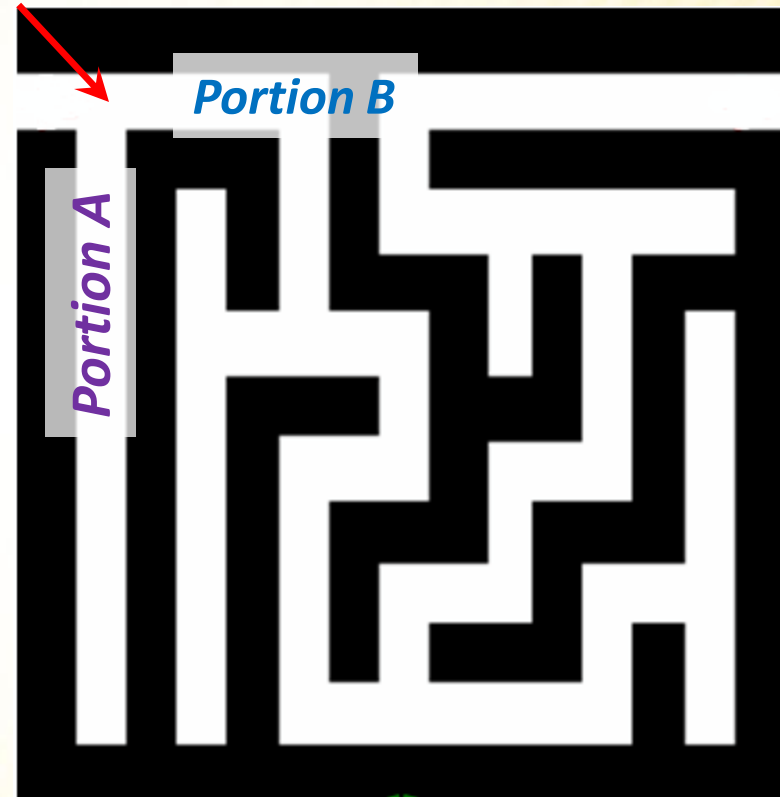
*Junction*



# Backtracking

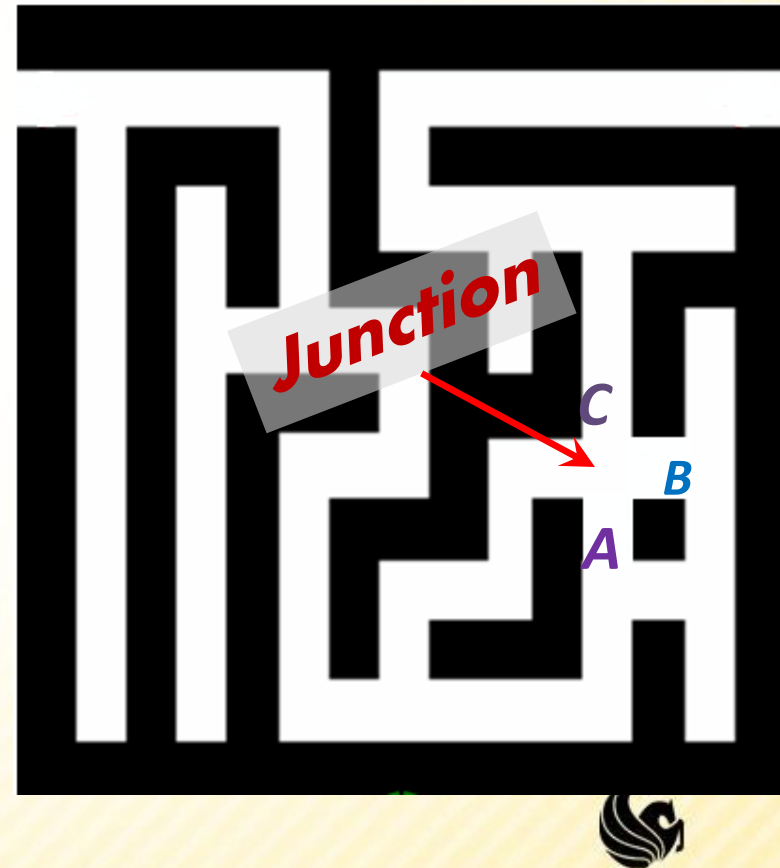
- One strategy would be to try going through **Portion A** of the maze.
  - If you get stuck before you find your way out, then you "**backtrack**" to the junction.
- At this point in time you know that **Portion A** will **NOT** lead you out of the maze,
  - so you then start searching in **Portion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

Junction



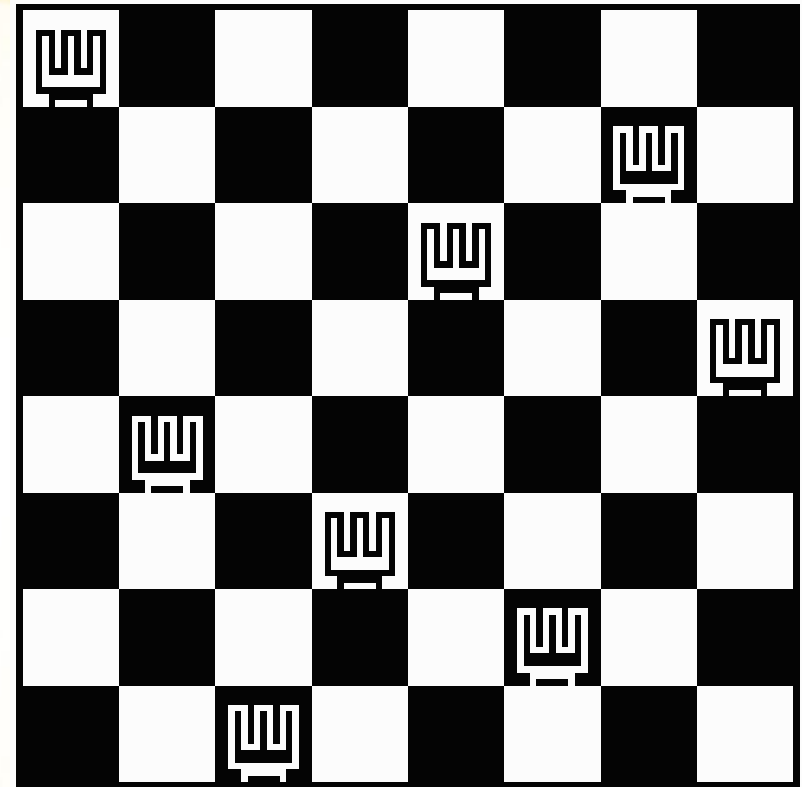
# Backtracking

- 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 – 8 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.
- This is also called the N Queens problem since we can solve this problem for any  $N \times N$  board with N Queens as well.



# Backtracking – 8 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 do 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 row we are placing the next queen

**usedList[]** – keeps track of the columns in which the queens have already been placed.

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

```
    if (location == SIZE) {  
        printSol(perm);  
    }
```

← Found a solution to the problem, so print it!

```
    for (int i=0; i<SIZE; i++) {
```

← Loop through possible columns to place this queen.

```
        if (usedList[i] == false) {
```

← Only try this column if it hasn't been used

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

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

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

←

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



# Backtracking – 8 Queens Problem

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



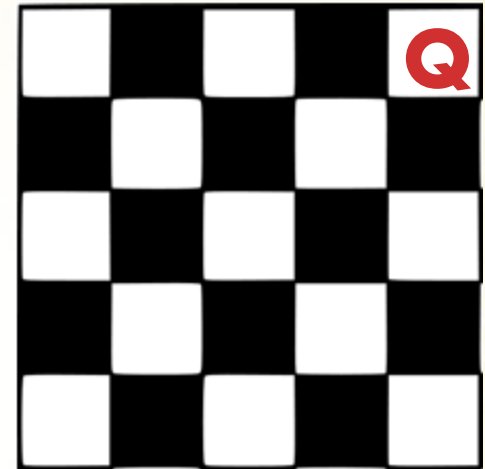
# Example Problem

- Show the first 2 solutions to the 5 queens problem that this algorithm would create.
  - Also show the decision tree as shown in class.




# Example Problem

- Finish the 5 Queens solution from this point, how many times do you have to backtrack?



# 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. 

# Mazes and Backtracking

- An 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.



# 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.

