CSCI 204: Data Structures & Algorithms

Recursion 2

Recursive binary search

- Idea:
  - Compare the target with the list item in the middle
  - If found, stop;
  - If target is greater than the middle, search the second half, otherwise, search the first half
- Base case(s):
  - Found or the list is exhausted
- Recursive case:
  - Search the first half
  - Search the second half

Recursive binary search

```
def  bin_search(nums, target, left, right):

    if left > right:   # not found
        return False

    mid = (left + right) // 2
    if nums[mid] == target:
        return True
    elif nums[mid] < target:  # search for upper half
        left = mid + 1
        return bin_search(nums, target, left, right)
    else:
        right = mid -1        # search for lower half
        return bin_search(nums, target, left, right)

nums = [2, 5, 6, 7, 9, 10, 12]
print(bin_search(nums, 2, 0, len(nums)-1))    # True
print(bin_search(nums, 12, 0, len(nums)-1))   # True
print(bin_search(nums, 6, 0, len(nums)-1))    # True
print(bin_search(nums, 22, 0, len(nums)-1))   # False
print(bin_search(nums, 0, 0, len(nums)-1))    # False
```

Check if a number is a prime

- Ideas: to determine if b is a prime, we check if b % x == 0 (divisible) consecutively …
- E.g., 5: we check 5%4, 5%3, 5%2, 5%1, when x reaches 1, we know 5 is a prime
- E.g., 6: we check 6%5, 6%4, 6%3 which is 0, stop, 6 is not a prime

List all permutations

- Make every element in the list as a prefix, one at a time, do it recursively
- E.g., ‘abcd’
  - ‘a’ + recursively(‘bcd’)
  - ‘b’ + recursively(‘acd’)
  - ‘c’ + recursively(‘abd’)
  - ‘d’ + recursively(‘abc’)

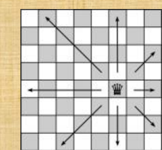
Now let’s do the workshop.

The 8-Queens Problem

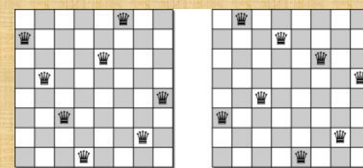
- The task is to place 8 queens onto a chessboard such that no queen can attack another queen.
  - Uses a standard 8 x 8 chess board.
  - There are 92 solutions to this problem.

Queen’s Moves

- The queen can move and attack any piece of the opponent by moving in any direction along a straight line.

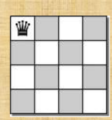


Sample Solutions



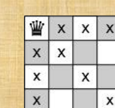
4-Queens Problem

- To develop an algorithm, we consider the smaller 4-queens problem.
  - Since no two queens can occupy the same column, we can proceed one column at a time.
  - Place a queen in position (0, 0).



4-Queens Problem

- This move eliminates a number of squares for the placement of additional queens.

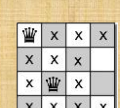


4-Queens Problem

- We move to the second column and place a queen at position (2, 1)

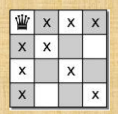
4-Queens Problem

- The 3rd queen should be placed in the 3rd column.
  - But there are no open cells in the third column.
  - So there is no solution based on the placement of the first 2 queens.



4-Queens Problem

- We have to backtrack:
  - go back to the previous column
  - pickup the last queen placed
  - try to find another valid cell in that column.



13

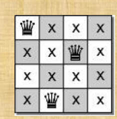
4-Queens Problem

- Place a queen at position (3,1) and move forward.

14

4-Queens Problem

- In the 3rd column, we can now place a queen at position (1,2).
- But now we have no open slots in the 4th column.



15

4-Queens Problem

- We again must backtrack and pick up the queen from the 3rd column.
- But there are no other empty cells in the 3rd column.

16

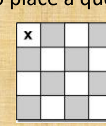
4-Queens Problem

- We must backtrack yet again and pick up the queen from the 2nd column.
- But there are no other empty cells in the 2nd column either.

17

4-Queens Problem

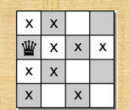
- So we backtrack one more time and pick up the queen from the 1st column.
- We then try again to place a queen in the 1st column.



18

4-Queens Problem

- In the 1st column, we can place a queen at position (1, 0).



4-Queens Problem

- We again continue with the process and attempt to find open positions in each of the remaining columns.
- We can use a similar approach to solve the original 8-queens problem.

N-Queens Board ADT

- The *n-queens board* is used for positioning queens on a square board for use in solving the n-queens problem.
  - consists of *n x n* squares.
  - each square is identified by index *[0...n)*

• NQueensBoard(n) • size() • numQueens() • unguarded(row, col)	• placeQueen(row, col) • removeQueen(row, col) • reset() • draw()

8-Queens Solution

```
def solveNQueens( board, col ):
  if board.numQueens() == board.size() :
    return True
  else :
    for row in range( board.size() ):
      if board.unguarded( row, col ):
        board.placeQueen( row, col )
        if board.solveNQueens( board, col+1 ) :
          return True
        else :
          board.removeQueen( row, col )

    return False
```