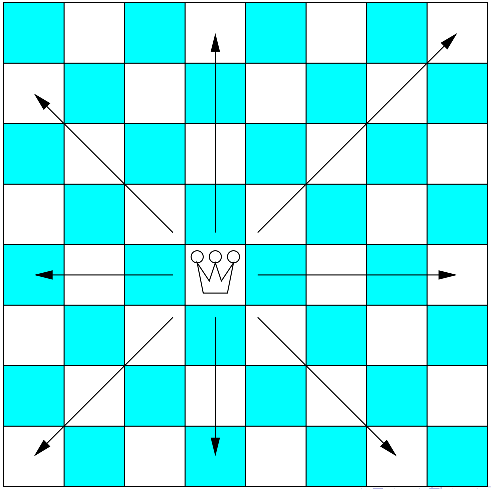
Backtracking with Recursion

Thumrongsak Kosiyatrakul
tkosiyat@cs.pitt.edu

Example 1: The Eight Queens Problem

- Suppose you have a chessboard and eight queens.
- Place all eight queens on the chessboard so that no queen piece is threatening another queen on the board.



Planning

- Put one queen in each column
  - Start at column 0
  - Stop when column is equal to 8
- Crate a helper method named `isLegel(int r, int c)`
  - Return true if it is okay to put a queen at row `r` column `c`.
  - Otherwise return false.
  - **Note** that this method should work with any number of queens on the board (1 to 8).
- Will use recursion and backtracking
  - Create a recursive method called `place(int c)` that place a queen in column `c`
  - This method return `true` if it can reach a solution
  - Otherwise, return false.

The method place(int c)

```
boolean place(int c)
{
    if(c == 8)
        return true;
    else
    {
        for(int r = 0; r < 8; r++)
        {
            if(isLegal(r,c))
            {
                put a queen at row r column c
                boolean p = place(c + 1);

                if(p)
                    return true;
                else
                    remove the queen from row r column c
            }
        }

        return false;
    }
}
```

Example 2: Solving a Maze

- Suppose we have a maze size `width` by `height`.
    - Starting position is at row 0 column 0.
    - Destination position is at row `height - 1` column `width - 1`.
- At each position, the algorithm will try to go north, east, south, and west (in that order).
- Will not move if there is a wall in that moving direction.
- Will not move if we just came from that direction.
- Assume that we have methods `isNorthWall(r,c)`, `isEastWall(r,c)`, `isSouthWall(r,c)`, and `isWestWall(r,c)` which return information about walls at row `r` column `c`.
- Create a recursive method called `solveMaze(r, c, pr, pc)` (backtracking)
    - Return true if it reach the destination
    - `pr` and `pc` are previous row and column

boolean solveMaze(r, c, pr, pc)

```
boolean solveMaze(r, c, pr, pc)
{
    if(r == height - 1 && c == width - 1)
        return true;
    else
    {
        if(!isNorthWall(r,c) && r - 1 != pr) {
            add (r - 1, c) to solution
            boolean p = solveMaze(r - 1, c, r, c);
            if(p)
                return true;
            else
                remove (r - 1, c) from solution
        }

        if(!isEastWall(r,c) && c + 1 != pc) {
            add (r, c + 1) to solution
            boolean p = solveMaze(r, c + 1, r, c);
            if(p)
                return true;
            else
                remove (r, c + 1) from solution
        }
        :
```

boolean solveMaze(r, c, pr, pc)

```
        :
        if(isSouthWall(r,c) && r + 1 != pr) {
            add (r + 1, c) to solution
            boolean p = solveMaze(r + 1, c, r, c);
            if(p)
                return true;
            else
                remove (r + 1, c) from solution
        }

        if(isWestWall(r,c) && c - 1 != pc) {
            add (r, c - 1) to solution
            boolean p = solveMaze(r, c - 1, r, c);
            if(p)
                return true;
            else
                remove (r, c - 1) from solution
        }

        return false;
    }
}
```

KenKen Puzzle

- Suppose we have a KenKen puzzle size n by n
- Planning
    - Try to put a number between 1 to n in every square
    - Have a method `isLegal()`
        - Return true if it is legal to put a number there
        - Otherwise, return false
        - This method should work with any number of numbers in the puzzle (1 to n * n). See board.
    - Definitely use recursion and backtracking
        - `boolean solveKenKen(int c)`
    - Environment: KenKen puzzle as ADT
        - Size, cages, target value, operator, etc
        - Allow you to add or remove a number from/to a row and a column
        - `isLegal()` method.

boolean solveKenKen(int c)

```
boolean solveKenKen(int r, int c)
{
    if(c == size) {
        c = 0;
        r = r + 1;
    }

    if(r == size)
        return true;
    else
    {
        for(int i = 1; i <= size; i++)
        {
            put i at row r column c
            boolean p = isLegal();
            if(p) {
                boolean q = solveKenKen(r, c + 1);
                if(q)
                    return true;
            }
        }
        return false;
    }
}
```