Recursion:

Friday, October 25, 2019 4:50 PM

- An object is said to be recursive if it partially consists or is defined in terms of itself
- Recursion is a powerful means of Mathematical definition.

Example
- Recursive definition of a set E
  - 2 is in E (Base-Case)
  - if x is in E then x+2 is also in E (Recursive Case)

$E = \{2, 4, 6, 8, \dots\}$ even positive numbers.

- A recursive definition has 2 parts
  - Base Case; A statement that can be resolved directly
  - Recursive case; in which the object is defined in terms of itself

Example: the set of all strings of balanced parenthesis.

'(' ')' '(())' ok '(()' No

1) • '()' is in the set.

2) • if S1 is in the set, then so is '(S1)'

3) • if S1 and S2 are in the set, then so is 'S1 S2'

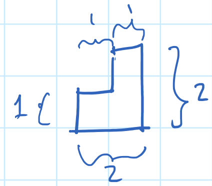
'((())())'

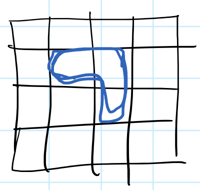
() by 1) then so '(())' by 2) then so '(())()' by 3) then so `((())())' by 2)

- The power of recursion is that it allows the definition of infinite objects by finite means

- Recursive Algorithms:-
  - Base Case:- an instance of the problem that can be solved directly
  - Recursive Case:- decomposing problem into simpler instances. contructing solution from the solutions of the simpler instances.

Example: "the triomino Problem"

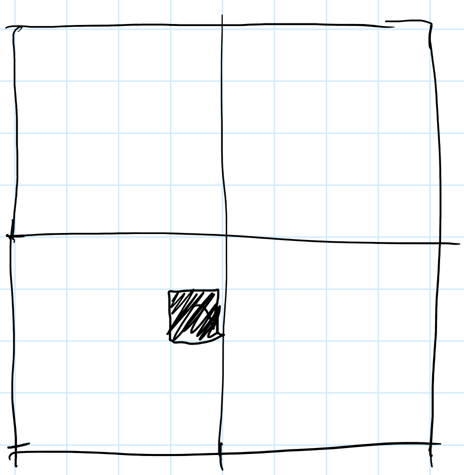


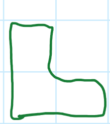


place triominos on the board.

problem: given a board of size 2^n where there is one hole of size 1×1, cover the board with triominoes

$n=3$

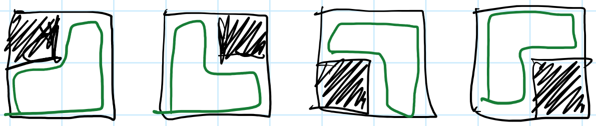




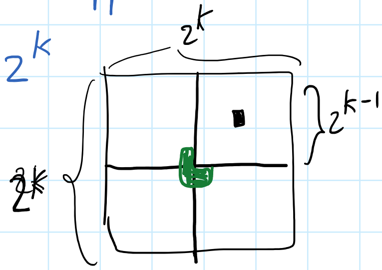
Recursive approach: Base case.

Recursive approach: Base case.

n=1

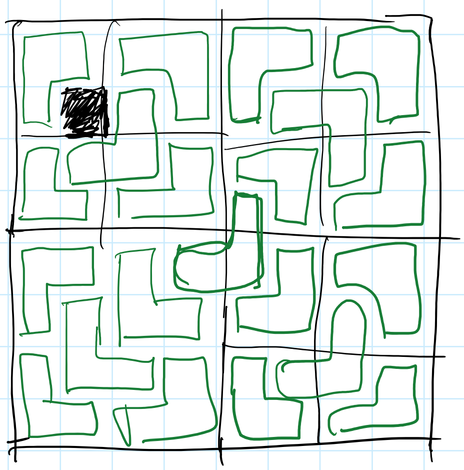


Recursive Approach : Recursive case



- split in 4
- place a triomino accross the boards without a hole.

n=3



• Recursive Programs:

```
foo( x )
{
   if x is the base case
      return direct solution
   else
      decompose x into sub-problems x'
      foo( x' )
      return solution constructed from solution to x'
}
```

Note: Recursion has an (undeserved) bad-rep.

1) "Every Recursion can be re-written as Iteration"
"Every Iteration can be re-written as Recursion"

2) Bad Examples:
fibonacci

$$\begin{cases} fib(1) = 1 \\ fib(2) = 1 \\ fib(n) = fib(n-1) + fib(n-2) & n>2 \end{cases}$$

```
fib( n )
{
  if( n==1 || n==2 )
    return 1;
  return fib( n-1 ) + fib( n-2 )
}
```

Good Example: Power $(x, y) = x^y$

```
pow( x, y )
{
   r = 1;
   for (i = 0; i < y; i++)
     r = r * x;
   return r;
}
```

$y=3$ $x * x * x * x$

$y=k$ $x * x * \dots$ k multiplications

y is $O(n)$

Recursion:

$$x^y \begin{cases} 1 & \text{if } y \text{ is } 0 \\ x * x^{y-1} \end{cases}$$

$x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x$

$$x^y = \begin{cases} 1 & \text{if } y \text{ is } 0 \\ x^{y/2} * x^{y/2} & \text{when } y \text{ is even} \\ x^{y/2} \cdot x^{y/2} \cdot x & \text{when } y \text{ is odd} \end{cases}$$

```
pow( x, y )
{
   if (y == 0 ) return 1;
   r = pow ( x , y/2 );
   if ( y is even )
      return r * r;
```

pow(2, 4) = 16
↓
pow(2, 2) = 4
↓

```
      r = pow ( x , y/2 );
      if ( y is even )
         return r * r;
      else
         return r * r * x;
   }
```

$pow(2,1) = 2$

$pow(2,0) = 1$

Multiplications. $2 * \log_2 y$ is $O(\log n)$

Recursive Backtracking

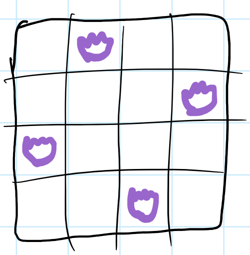
- Many problems do not have a "fixed rule" solution

Strategy = Decompose problem into a sequence of trail & error tasks.

Example: The n-queens problem.

Given an $n \times n$ chessboard, place n queens in the board such that the queens do not attack each other.

$n=4$



1850's Gauss. ≡ no general rule

$16 \times 15 \times 14 \times 13 = 43,680$

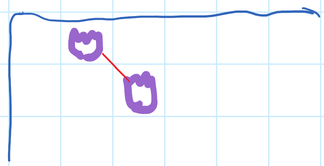
trick one queen per row.

$4 \times 4 \times 4 \times 4 = 4^4 = 256$

$6^6 = 46656$

$8^8 = 16,777,216$

- trick one queen per row.
- place one queen at a time and stop when conflicts arise:



Note:

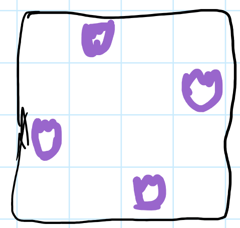
On Object Oriented Programming

- class Board
- class Queen, derived from class piece.

then ???

Algorithm:

```
try_queen ( i )
  repeat
    place i'th queen
      if no more queens to place.      ] Base - Case
        return success!
      else
        try = try_queen ( i+1 )
        if try is succesfull
          return success!             ] recursive case.
        else
          retract queen;
  until out of places for i'th queen.
  return Fail! ☹
```



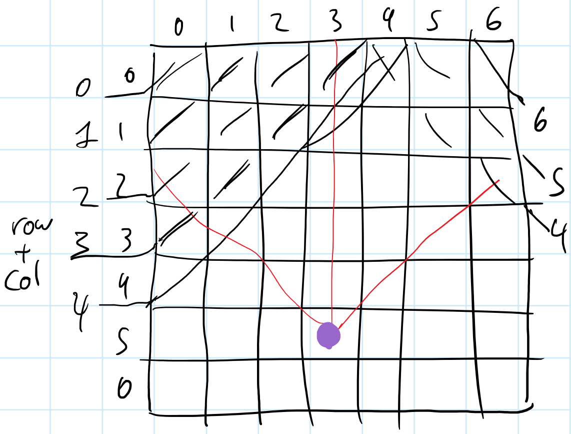
Refining algorithm:

```
bool try-queens ( int row, board , int n )
    bool try;
    for ( int col = 0 ; col < n ; col ++ )
    {
        if valid ( row, col, board )
            record ( board, row, col )
            if ( col == n-1 )
                return true;
```

```
        else
            try = try-queen( row+1, board, n)
            if (try)
                return true;
            else
                retract (board, row, col)
    }
    return false;
```

the board

1) idea 2d Array:



col-row

col - row
col + row

2) 1-d Array ; store column

0	1	2	3	4	5	6
4	1	5	2	0		

General form:-

```
try
    initialize choices
    do
        select choice
        if choice is acceptable
            record choice
            if solution complete
                return success!
            else
                try next step
                if next step succeeds
                    return success!
                else
                    retract choice.
    while more choices available.
```

retract choice.

while move choices available.

return fail.