- Homework 5 is on stacks
- Go to megaminer.

1 Recursive Object

An object which potentially consists or defined in terms of itself. Recursion is used to define things such as:

- Sets
- Functions
- Other objects

Power of Recursion:

- Describe an infite object
- Through finite means.

Recursive Definition

- Base Case
- Recursive Case

Example: Set of all strings of balanced parenthesis.
Base Case: () is in the set.
Recursive Case: if s is in the set, then s(), ()s, and (s) are also in the set.

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

$$Factorial = \begin{cases} 1! = 1 \\ n! = n \times (n-1) \end{cases} \tag{2}$$

1.1 Recursive Algorithms

- Base Case
  - Direct solution to a small problem instance.
- Recursive Case
  - Decompose problem into smaller instances.
  - Solve smaller instances.
  - Construct solution from smaller solutions.

1.1.1 Triomino Problem

Suppose we have four possible tiles made of three squares.

Problem: Cover $2^n \times 2^n$ board, where one tiles is a hole with triominoes.

- $n = 4, 2^n = 16$

  Morales gives example.

- split board in 4 equal parts.
- Place tronmino across 3 split parts without a hole.
- Solve each subpart.

```
void foo() {
  int x;

  foo();
}

quicksort(array, left, right) { // assuming left < or = right
  if (left = right) {
    return; // Base Case
  }

  pivot = a[(left + right) / 2];
  int i = left;
  int j = right;

  repeat
    while (a[i] < pivot) { i++; }
    while (a[j] > pivot) { j--; }
    if (i < j) {
      swap(a[i], a[j]);
      i++;
      j--;
    }
  while (j > i);

  quicksort(a, i + 1, r);
  quicksort(a, l, i - 1)

}
```

2 Recursive Backtracking

```
try
    intialize choices
    do
        select choice
        if choice is valid
            record choice
            if solution complete
                success!
            else
                try next step
                if next step succeeds
                    success!
                else
                    cancel record
    while !success & more choices available

path_find(grid, int row, int col) {
    for choice c in {N, NE, E}
        nrow = row after c;
        ncol = col after c;

        if (grid[nrow][ncolumn]) != obstacle && nrow, ncol is in bounds)
        record nrow, ncol;

        if (grid[row][column] == cake!) {
            return true;
        } else {
            solve = path_find(grid, nrow, ncol)
            if (solve) {
                return true;
            } else {
                record C;
            }
        }

    return false
}

bool valid(grid, int r, int c) {
    if (c < 0 || r >= N) {
        return false;j
    }
    if (c < 0 || c >= N) {
        return false;
    }
    if (grid[r, c] = obstacle) {
        return false;
    }

    return true;
}

for (int i = 0; i < 3; c++) {
    nrow = col + dir[c][0];
    ncol = col + dir[c][1];
    if (valid(grid, nrow, ncol, n)
}
```