Laboratório de Programação Avançada 2018/19

Week 2 – Recursion

· U C ·

Universidade de Coimbra

Outline

1. Introduction
2. Examples
3. Recursion and iteration
4. Time complexity

Reading about problem solving with recursion

- J. Erickson, Algorithms, Chapter 1
- J. Edmonds, How to think about algorithms, Chapter 8 (or Part II - recursion)
- S.S. Skiena, M.G. Revilla, Programming Challenges, Chapter 6

Problem solving

- In LPA, you can solve most of the problems by using reduction techniques. You need to recognize the underlying problem.
- Or use a general strategy: Break the problem down into smaller problems which you can solve, and devise how to recover the solution from the partial solutions found
- This is the main strategy of backtracking, dynamic programming, greedy algorithms and branch-&-bound
- To know how to break the problem in the most effective manner requires a lot of training

Introduction

Recursive program: A program that calls itself.

Main idea: We solve the problem by solving smaller sub-problems.

1. A base case (simple problem, not solved by recursion)
2. A recursive step (uses solutions of sub-problems)

Introduction

Recursive program: A program that calls itself.

Main idea: We solve the problem by solving smaller sub-problems.

1. A base case (simple problem, not solved by recursion)
2. A recursive step (uses solutions of sub-problems)

Proof by mathematical induction:

1. (Base case) It is true for the base case
2. (Inductive hypothesis) Assume that is true for k
3. (Inductive step) If it is true for k then it must be true for $k+1$.

Induction:

Show that $0 + 1 + 2 + \cdots + n = \dfrac{n(n+1)}{2}$

1. Base case: True for $n = 0$: $0 = \dfrac{0 \cdot (0+1)}{2}$
2. If it holds for k, then it also holds for $k+1$:

$$(0 + 1 + 2 + \cdots + k) + (k+1) = \frac{(k+1)((k+1)+1)}{2}$$

Under the induction hypothesis that is true for k:

$$\frac{k(k+1)}{2} + (k+1) = \frac{(k+1)((k+1)+1)}{2}$$

A recursive algorithm to compute the square of a number n

Function $SQ(n)$
  if $n = 0$ **then** {base case}
    $s = 0$
  else
    $s = SQ(n-1) + 2(n-1) + 1$ {recursive step}
  return s

Note that $n^2 = (n-1)^2 + 2(n-1) + 1$.

Correctness proof by induction

- The recursion terminates when $n = 0$
- Base case: After the last recursion, $s = 0$
- Inductive hypothesis: Assume that after returning from $k - 1$ recursions, $s = (k - 1)^2$
- Inductive step: After returning from k recursions, $s = (k - 1)^2 + 2(k - 1) + 1 = k^2$
- Then, after returning from n recursions, $s = (n - 1)^2 + 2(n - 1) + 1 = n^2$

Introduction

Patterns:

- Handle first or last and recur on remaining
- Divide in half, recur on one/both halves (D&C)

Pros: Smaller code, few or no local variables.

Cons: Less eficient than iterative because of the push and pop operations in the run-time stack. Can have problems of stack overflow.

Examples

Problem: Draw a Sierpiński triangle

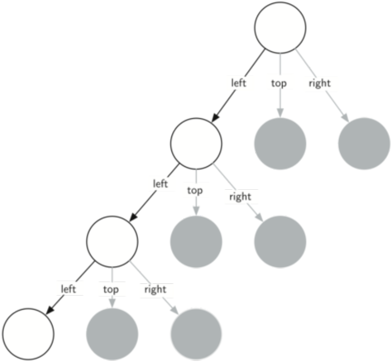
Problem: Draw a Sierpiński triangle

Recursion: Draw smaller triangles at the left, top and right of the large triangle

Base case: The triangle is small enough

Examples

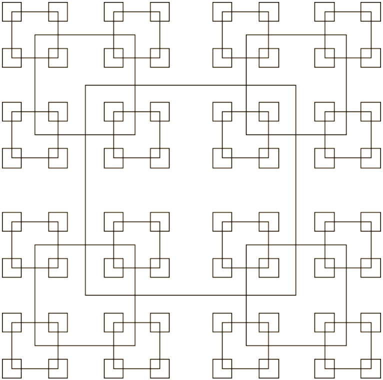
Recursive call tree



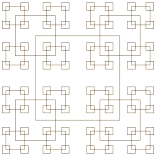
base case →

Examples

Problem: All Squares (modified UVa 155)



Examples



Function *Square*(x, y, s)
    drawSquare(x, y, s) $\{(x, y)$ is the centroid of the square$\}$
    if $s/2 \leq 1$ **then** $\{$base case$\}$
        return
    else $\{$recursive step$\}$
        Square$(x + s/2, y + s/2, s/2)$ $\{$top-right$\}$
        Square$(x - s/2, y + s/2, s/2)$ $\{$top-left$\}$
        Square$(x + s/2, y - s/2, s/2)$ $\{$bottom-right$\}$
        Square$(x - s/2, y - s/2, s/2)$ $\{$bottom-left$\}$

Problem: How many squares?

Function *Square*(x, y, s)

  if $s/2 \leq 1$ **then** {base case}

    return 1

  else {recursive step}

    return $1 + \mathit{Square}(x + s/2, y + s/2, s/2) +$ {top-right}

      $\mathit{Square}(x - s/2, y + s/2, s/2) +$ {top-left}

      $\mathit{Square}(x + s/2, y - s/2, s/2) +$ {bottom-right}

      $\mathit{Square}(x - s/2, y - s/2, s/2)$ {bottom-left}

Examples

Problem: How many squares contain a given point (p_x, p_y)?

Function *Square*(x, y, s)

$k = 0$

if $p_x \in [x - s/2, x + s/2]$ **and** $p_y \in [y - s/2, y + s/2]$ **then** $\{$in$\}$

$k = 1$

if $s/2 \leq 1$ **then** $\{$base case$\}$

return k

else $\{$recursive step$\}$

return $k + Square(x + s/2, y + s/2, s/2) +$ $\{$top-right$\}$

$Square(x - s/2, y + s/2, s/2) +$ $\{$top-left$\}$

$Square(x + s/2, y - s/2, s/2) +$ $\{$bottom-right$\}$

$Square(x - s/2, y - s/2, s/2)$ $\{$bottom-left$\}$

Examples

Problem: Gray Code

1-gray code		2-gray code		3-gray code			4-gray code			
							0	0	0	0
							0	0	0	1
							0	0	1	1
							0	0	1	0
				0	0	0	0	1	1	0
				0	0	1	0	1	1	1
		0	0	0	1	1	0	1	0	1
0		0	1	0	1	0	0	1	0	0
1		1	1	1	1	0	1	1	0	0
		1	0	1	1	1	1	1	0	1
				1	0	1	1	1	1	1
				1	0	0	1	1	1	0
							1	0	1	0
							1	0	1	1
							1	0	0	1
							1	0	0	0

Examples

Problem: Gray Code

0	0	0	0
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	1	1
0	1	0	1
0	1	0	0
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0
1	0	1	0
1	0	1	1
1	0	0	1
1	0	0	0

1 and 2-gray code

0	0	0	0
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	1	1
0	1	0	1
0	1	0	0
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0
1	0	1	0
1	0	1	1
1	0	0	1
1	0	0	0

2 and 3-gray code

0	0	0	0
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	1	1
0	1	0	1
0	1	0	0
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0
1	0	1	0
1	0	1	1
1	0	0	1
1	0	0	0

3 and 4-gray code

Problem: Gray Code

Recursion: The n-bit Gray code is defined recursively as follows:

- the n-1 bit code, with 0 prepended to each word, followed by
- the n-1 bit code in reverse order, with 1 prepended to each word.

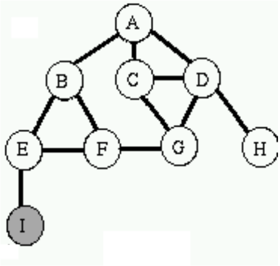
Base case: The 1-bit code is 0 followed by 1.

Examples

Function *grayCode*(n)

```
if n = 1 then                                   {base case}
    bits = [0, 1]
else                                            {recursive step}
    grayCode(n − 1)                             {compute (n-1)-bit GC}
    rbits = reverse(bits)                       {reverse (n-1)-bit GC}
    prepend(0, bits)                            {prepend 0's to (n-1)-bit GC}
    prepend(1, rbits)                           {prepend 1's to reversed (n-1)-bit GC}
    bits = bits + rbits                         {n-bit GC}
```

Problem: Depth First Search (DFS)



Examples

Recursion: Visit neighbors of a node in G that were not yet visited

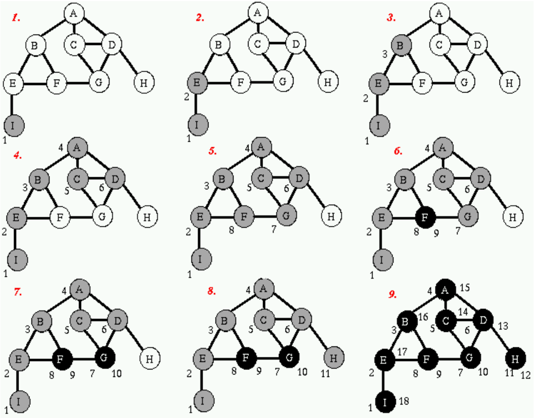
Base case: All neighbors were already visited

Function $dfs(G, u)$
    $color(u) = gray$ {node u is in progress}
    for each $(u, v) \in G$ **and** $color(v) = white$ **do**
        $dfs(G, v)$ {run dfs on v}
    $color(u) = black$ {node u is visited}

Note: all nodes in G are marked white (unvisited)

Examples

Problem: Depth First Search (DFS)



Problem: Find node with label ℓ with dfs

Function $\mathit{dfs}(G, u, \ell)$

  if $\mathit{label}(u) = \ell$ **then** {base case}

    return `true`

  else {recursive step}

    $\mathit{color}(u) = \mathit{gray}$ {node u is in progress}

    for each $(u, v) \in G$ **and** $\mathit{color}(v) = \mathit{white}$ **do**

      if $\mathit{dfs}(G, v, \ell) =$ `true` **then** {if dfs on v found the node}

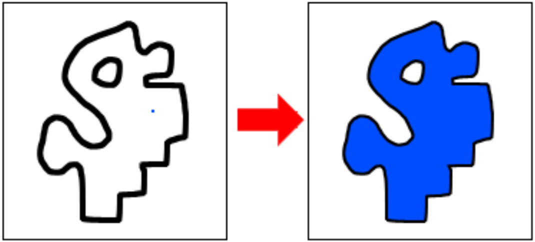
        return `true` {stop recursion}

    $\mathit{color}(u) = \mathit{black}$ {node u is visited}

    return `false`

Examples

Problem: Flood Fill



Recursion: Visit neighbors of a cell that were not yet colored

Base case: All neighbors were already colored

Function *flood*(M, x, y)

  if $color(M[x][y]) = \texttt{true}$ **then** {base case}

    return

  else {recursive step}

    $paint(M, x, y)$ {paint in (x, y)}

    $flood(M, x, y-1)$ {down}

    $flood(M, x, y+1)$ {up}

    $flood(M, x-1, y)$ {left}

    $flood(M, x+1, y)$ {right}

Examples

Problem: Exploring a maze

	A	B	C	D	E	F	G	H
1	*	*	*	*	*			
2	*				*			
3	*	S	*	*	*			
4	*				*	*	*	*
5	*		*					*
6	*				*			*
7	*	*	*	*	*		E	*
8					*	*	*	*

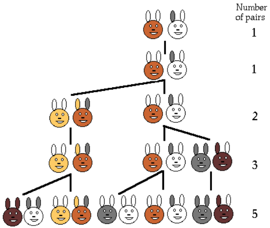
Examples

Function $Maze(M, x, y)$

```
if y > 8 or y < 1 or x <'A' or x >'H' then          {base case: limits}
    return false
if M[x][y] = '*' then                               {base case: wall}
    return false
if M[x][y] = 'E' then                               {base case: exit}
    return true
M[x][y] = "*"
if Maze(M, x, y − 1) = true then                    {recursive step: down}
    return true
if Maze(M, x, y + 1) = true then                    {recursive step: up}
    return true
if Maze(M, x − 1, y) = true then                    {recursive step: left}
    return true
if Maze(M, x + 1, y) = true then                    {recursive step: right}
    return true
return false
```

Problem: Fibonacci numbers

A man has one pair of rabbits at a certain place entirely surrounded by a wall. We wish to know how many pairs can be bred from it in one year, if the nature of these rabbits is such that they breed every month one pair (male and female), that in turn will begin to breed in the second month after their birth.



Examples

Recursion: $fib(n) = fib(n-1) + fib(n-2)$

Base case: $fib(0) = 0, fib(1) = 1$

Function $fib(n)$
  if $n = 0$ **or** $n = 1$ **then** {base case}
    return n
  else {recursive step}
    return $fib(n-1) + fib(n-2)$

Bad example of recursion: Excessive recomputation since it does not take into account that $fib(n-2)$ was already computed.

Recursion tree for *fib*(5)