Public-Service Announcement

Lecture #7: Tree Recursion

"Autofocus is Berkeley's first mobile photography club. Join us as we build a community of phone photographers at Cal. All you need to be part is an interest in photography and a mobile phone!

Infosessions on 2/2 and 2/7. Details at tiny.cc/autofocus"

Last modified: Sun Feb 19 15:36:10 2017

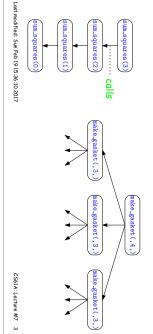
CS61A: Lecture #7 1

Last modified: Sun Feb 19 15:36:10 2017

CS61A: Lecture #7 2

Tree Recursion

- The make gasket function is an example of a tree recursion, where each call makes multiple recursive calls on itself.
- A linear recursion makes at most one recursive call per call
- A tail recursion has at most one recursive call per call, and it is the last thing evaluated.
- A linear recursion such as for sum_squares produces the pattern of calls on the left, while make_gasket produces the pattern on the right—an instance of what we call a tree in computer science.



What About This?

```
What kind of recursion is this?
def find.it(f, y, low, high):
    """Given that F is a nondecreasing function on integers,
    find a value of x between LOW and HIGH inclusive such that
    F(x) == Y. Return None if there isn't one."""
    if low > high:
        return None
    mid = (low + high) // 2
    val = f(mid)
    return val == y \
        or (val < y and find.it(f, y, low, mid-1)) \
        or (val > y and find.it(f, y, mid+1, high))
```

What About This?

```
What kind of recursion is this? Tail Recursion
def findit(f, y, low, high):
    """Given that f is a nondecreasing function on integers,
    find a value of x between LOW and HIGH inclusive such that
    F(x) == Y. Return None if there isn't one."""
    if low > high:
        return None
    mid = (low + high) // 2
    val = f(mid)
    return val == y \
        or (val < y and findit(f, y, low, mid-1)) \
        or (val > y and findit(f, y, mid+1, high))
```

What About This?

Last modified: Sun Feb 19 15:36:10 2017

CS61A: Lecture #7

```
What kind of recursion is this? Tree Recursion
def findit(f, y, low, high):
    """Given that F is a nondecreasing function on integers,
    find a value of x between LOW and HIGH inclusive such that
    F(x) == Y. Return None if there isn't one."""

if low > high:
    return None
mid = (low + high) // 2
val = f(mid)
    return val == y \
    or (val < y and find.it(f, y, low, mid-1)) \
    or (find.it(f, y, mid+1, high))</pre>
```

Last modified: Sun Feb 19 15:36:10 2017

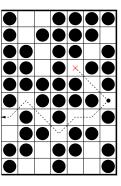
SOLA: Lecture #

Last modified: Sun Feb 19 15:36:10 2017

CS61A: Lecture #7 6

Finding a Path

Consider the problem of finding your way through a maze of blocks:



- From a given starting square, one can move down one level and up to one column left or right on each step, as long as the square moved to is unoccupied.
- Problem is to find a path to the bottom layer
- Diagram shows one path that runs into a dead end and one that es-

Last modified: Sun Feb 19 15:36:10 2017

CS61A: Lecture #7 7

Path-Finding Program

Translating the problem into a function specification:

def is_path(blocked, x0, y0):
 """True iff there is a pat

square (x1, 0) such that all squares on the path (including first

there is a path of squares from (XO, YO) to some

			~				[F]	μ.	Д
0	1	2	ω	4	Ŋ	6	ach	s t	ast
•			lacksquare		lacksquare		23	rue	9
					lacksquare		ep	H	rre
•	lacksquare		lacksquare				of	H	m
							р	the	OCC.
	lacksquare		lack				pat	100	upi
•							3	id	ed.
	lacksquare		lacksquare				000	squ	ш
							d d	are	00 T8
•	lacksquare		lacksquare				nwo	ρ.	KE
•			lacksquare				on	is true iff the grid square at (x, y) is occupied or off the edge.	<u>į</u> .
							е	×	es es
	٠,		_				WO	G	pr
Č	not M(1 1) not M(2 2)	M(0,0) $M(1,0)$ $M(1,2)$	ξ.	5			and	1S	edi
1	₹ .	ء ح	, כ	Δ.			H	000	cat
É	3 ;	7	5 5	3.			Or	du	Ö m
ŀ	٠,	<u>₹</u> }	<u> </u>	<u> </u>			0	Led	sucl
,	7 5	_ {	3 5	2			201	or	ı t
1	3 3	2 7	9 6	<u> </u>			mm	of	hat
ò	+ :	≤ 3	≥ 8	8			8 1	f t	В
	Z /	<u> </u>	<u> </u>	9			eft	he	DCK
,		<u>ه</u> و	2 6	2			0	edg	8
	ت ر	Ç	j o	0			H	še.	last) are unoccupied. BLOCKED is a predicate such that $BLOCKED(x, y)$
		?	by a predicate M where each	This arid would be represented			Each step of a path goes down one row and 1 or 0 columns left or right."""		Y)
		á	י כ	2			. =		
							- 1		

by a predicate M where, e.g. M(0,0), M(1,0), M(1,2), not M(1,1), not M(2,2). This grid would be represented

Last modified: Sun Feb 19 15:36:10 2017

Here, is path(M, 5, 6) is true; is path(M, 1, 6) and is path(M, 6, 6) are false.

is_path Solution (I)

def is_path(blocked, x0, y0): """True iff there is a path of squares from (XO, YO) to some square (x1, O) such that all squares on the path (including first and last) are unoccupied. BLOCKED is a predicate such that BLOCKED(x, y) is true iff the grid square at (x, y) is occupied or off the edge. Each step of a path goes down one row and 1 or 0 columns left or right."""

elif return

else:

return

return

Last modified: Sun Feb 19 15:36:10 2017

CS61A: Lecture #7 9

is_path Solution (II)

def is_path(blocked, x0, y0):
 """True iff there is a path of squares from (X0, Y0) to some
 square (x1, 0) such that all squares on the path (including first and
 last) are unoccupied. BLOCKED is a predicate such that BLOCKED(x, y)
 is true iff the grid square at (x, y) is occupied or off the edge.
 Each step of a path goes down one row and 1 or 0 columns left or right.""" else: elif return True return False

Last modified: Sun Feb 19 15:36:10 2017 CS61A: Lecture #7 10

is_path Solution (III)

def is_path(blocked, x0, y0):
 """True iff there is a path of squares from (X0, Y0) to some
 square (x1, 0) such that all squares on the path (including first and
 last) are unoccupied. BLOCKED is a predicate such that BLOCKED(x, y)
 is true iff the grid square at (x, y) is occupied or off the edge.
 Each step of a path goes down one row and 1 or 0 columns left or right.""" if blocked(x0, y0):

else: elif return False return True

CS61A: Lecture #7

Last modified: Sun Feb 19 15:36:10 2017

is_path Solution (IV)

def is.path(blocked, x0, y0):
 """True iff there is a path of squares from (X0, Y0) to some
 square (x1, 0) such that all squares on the path (including first and
 last) are unoccupied. BLOCKED is a predicate such that BLOCKED(x, y)
 is true iff the grid square at (x, y) is occupied or off the edge.
 Each step of a path goes down one row and i or 0 columns left or right.""" else: elif y0 == 0: if blocked(x0, y0): return True return False

Last modified: Sun Feb 19 15:36:10 2017 CS61A: Lecture #7 12

is_path Solution (V)

```
def is.path(blocked, x0, y0):
    """True iff there is a path of squares from (X0, Y0) to some
    square (x1, 0) such that all squares on the path (including first and
    last) are unoccupied. BLOCKED is a predicate such that BLOCKED(x, y)
    is true iff the grid square at (x, y) is occupied or off the edge.
    Each step of a path goes down one row and 1 or 0 columns left or right."""
                                                                              elif y0 == 0:
                                                                                                                                                                                                                                                        if blocked(x0, y0):
                                                                                                                                                                     return False
return True
```

return is_path(blocked, x0-1, y0-1) or is_path(blocked, x0, y0-1) \ or is_path(blocked, x0+1, y0-1)

Last modified: Sun Feb 19 15:36:10 2017

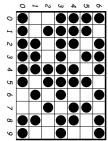
CS61A: Lecture #7

13

Variation I

def num_paths(blocked, x0, """Return the number of unoccupied paths that run from (XO, YO) to some square (x1, O). BLOCKED is a predicate such that BLOCKED(x, y) is true iff the grid square at (x, y) is occupied or off the edge. """ y0):

For the previous predicate M, the result of numpaths (M, 5, 6) is 1. For the predicate M2, denoting this grid (missing (7,1)):



the result of num_paths(M2, 5, 6) is 5.

Last modified: Sun Feb 19 15:36:10 2017

CS61A: Lecture #7 14

num_paths Solution (I)

```
def num_paths(blocked, x0, y0):
    """Return the number of unoccupied paths that run from (X0, Y0)
    to some square (x1, 0). BLOCKED is a predicate such that BLOCKED(x, y)
    is true iff the grid square at (x, y) is occupied or off the edge. """
else:
                                                                                                     elif y0 == 0:
                                                                                                                                                                                                                                             if blocked(x0, y0):
                                  return
                                                                                                                                                                              return
```

Last modified: Sun Feb 19 15:36:10 2017

return

CS61A: Lecture #7 15

num_paths Solution (II)

```
def num_paths(blocked, x0, y0):
    """Return the number of unoccupied paths that run from (X0, Y0)
    to some square (x1, 0). BLOCKED is a predicate such that BLOCKED(x, y)
    is true iff the grid square at (x, y) is occupied or off the edge. """
                                                                    else:
                                                                                                                                                                   elif y0 == 0:
                                                                                                                                                                                                                                                                                                         if blocked(x0, y0):
                                                                                                     return 1
                                                                                                                                                                                                                                 return 0
return
```

Last modified: Sun Feb 19 15:36:10 2017

CS61A: Lecture #7 16

num_paths Solution (III)

```
def num_paths(blocked, x0, y0):
    """Return the number of unoccupied paths that run from (X0, Y0)
    to some square (x1, 0). BLOCKED is a predicate such that BLOCKED(x, y)
    is true iff the grid square at (x, y) is occupied or off the edge. """
                                                                                                                      else:
                                                                                                                                                                                                                                          elif y0 == 0:
                                                                                                                                                                                                                                                                                                                                                                                                       if blocked(x0, y0):
                                                                                                                                                             return 1
                                                                                                                                                                                                                                                                                                                       return 0
\label{eq:num_paths} $$ num_paths(blocked, x0-1, y0-1) + num_paths(blocked, x0, y0-1) + num_paths(blocked, x0+1, y0-1) $$
```

Variation II

def find_path(blocked, x0, y0):
 """Return a string containing the steps in an unoccupied
 path from (X0, Y0) to some unoccupied square (x1, 0),
 or None if not is.path(BLOCKED, X0, Y0). BLOCKED is a
 predicate such that BLOCKED(x, y) is true iff the grid square at (x, y) is occupied or off the edge. predicate such 0 1 2 3 4 5 6 •

Possible result of find_path(M, 5, 6):

6

"(5, 6) (6, 5) (6, 4) (7, 3) (6, 2) (5, 1) (6, 0)"

Last modified: Sun Feb 19 15:36:10 2017

CS61A: Lecture #7

17

Last modified: Sun Feb 19 15:36:10 2017

CS61A: Lecture #7 18

find_path Solution (I)

```
def find_path(blocked, x0, y0):
    """Return a string containing the steps in an unoccupied
    path from (X0, Y0) to some unoccupied square (x1, 0),
    or None if not is.path(BLOCKED, X0, Y0). BLOCKED is a
    predicate such that BLOCKED(x, y) is true iff the
    grid square at (x, y) is occupied or off the edge. """
    if blocked(x0, y0):
        return
    elif y0 == 0:
        return
        return
```

CS61A: Lecture #7 19

Last modified: Sun Feb 19 15:36:10 2017

find_path Solution (II)

```
def find_path(blocked, x0, y0):
    """Return a string containing the steps in an unoccupied
    path from (X0, Y0) to some unoccupied square (x1, 0),
    or None if not is.path(BLOCKED, X0, Y0). BLOCKED is a
    predicate such that BLOCKED(x, y) is true iff the
    grid square at (x, y) is occupied or off the edge. """
    step = "({}, {})".format(x0, y0)
    # Alternative: step = str((x0, y0))
    if blocked(x0, y0):
        return None
    elif y0 == 0:
        return step
    else:
        return step
```

Last modified: Sun Feb 19 15:36:10 2017

CS61A: Lecture #7 20

find_path Solution (III)

```
def find_path(blocked, x0, y0):
    """Return a string containing the steps in an unoccupied
    path from (X0, Y0) to some unoccupied square (x1, 0),
    or None if not is_path(BLDCXED, X0, Y0). BLDCXED is a
    predicate such that BLDCXED(x, y) is true iff the
    grid square at (x, y) is occupied or off the edge. """
    step = "({}, {})".format(x0, y0)
    if blocked(x0, y0):
        return None
    elif y0 == 0:
        return None
    elif y0 == 0:
        return tetp
    p = find_path(blocks, x0-1, y0-1)
        if p is not None: return p + " " + step
        p = find_path(blocks, x0, y0-1)
        if p is not None: return p + " " + step
        p = find_path(blocks, x0+1, y0-1)
        if p is not None: return step + " " + p
        return None
```

Last modified: Sun Feb 19 15:36:10 2017

CS61A: Lecture #7 21

find_path Solution (IV)

```
def find_path(blocked, x0, y0):
    """Return a string containing the steps in an unoccupied
    path from (X0, Y0) to some unoccupied square (x1, 0),
    or None if not is.path(BLOCKED, X0, Y0). BLOCKED is a
    predicate such that BLOCKED(x, y) is true iff the
    grid square at (x, y) is occupied or off the edge. """
    step = "({}, {})".format(x0, y0)
    if blocked(x0, y0):
        return None
    elif y0 == 0:
        return None
else:
    for x in (x0-1, x0, x0+1):
        p = find.path(blocks, x, y0-1)
        if p is not None: return p + " " + step
        return None
```

A Change in Problem

- Suppose we changed the definition of "path" for the maze problems to allow paths to go left or right without going down.
- ullet And suppose we changed solutions in the obvious way, adding clauses for the (x_0-1,y_0) and (x_0+1,y_0) cases.
- Will this work? What would happen?

And a Little Analysis

- All our linear recursions took time proportional (in some sense) to the size of the problem.
- What about is_path?

Last modified: Sun Feb 19 15:36:10 2017 C561A: Lecture #7 23

Last modified: Sun Feb 19 15:36:10 2017

CS61A: Lecture #7 24

	Lest modified: Sun Feb 19 15:36:10 2017	 And a Little Analysis All our linear recursions took time proportional (in some sense) to the size of the problem. What about is path? Each call spawns up to three others, up to yo "generations." That means the number of possible calls could be as many as 3³⁰⁰—exponential growth.
	C561A: Lecture #7 25	in some sense) to e others, up to y0 e calls could be as