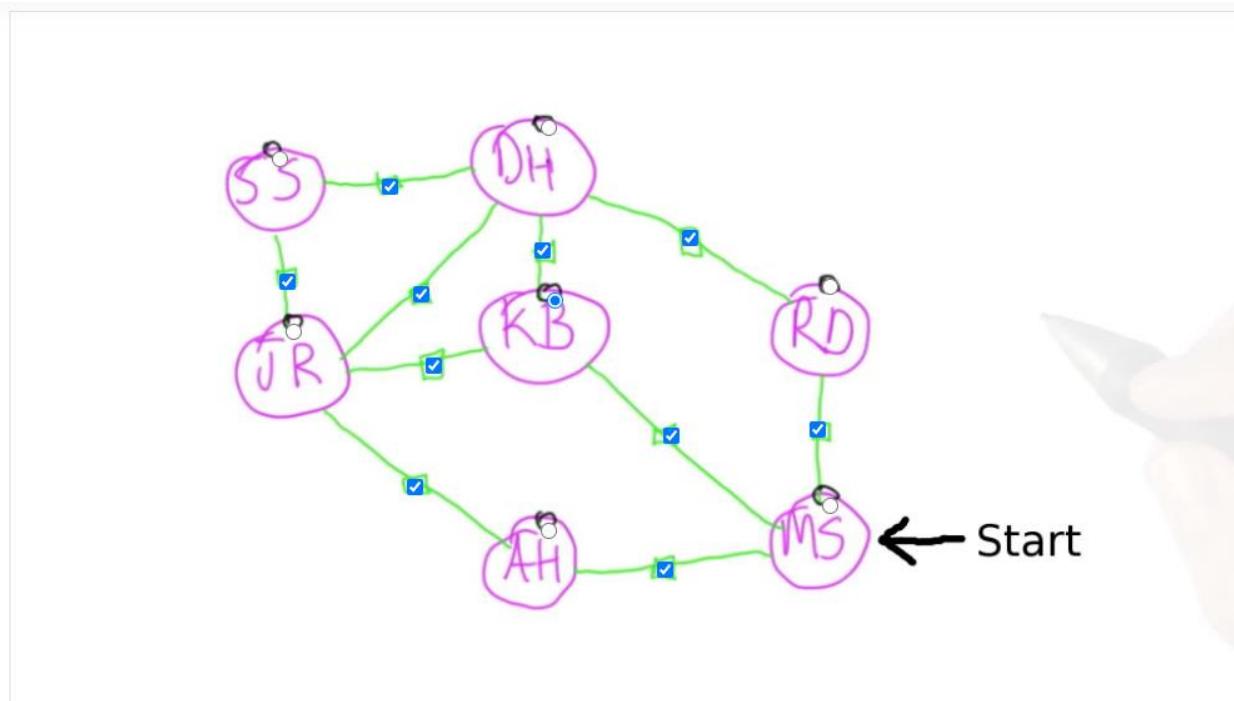


Fellhipe Gutierrez

Magic Trick



Eulerian Path

Can a graph with only even-degree nodes have an Eulerian path?

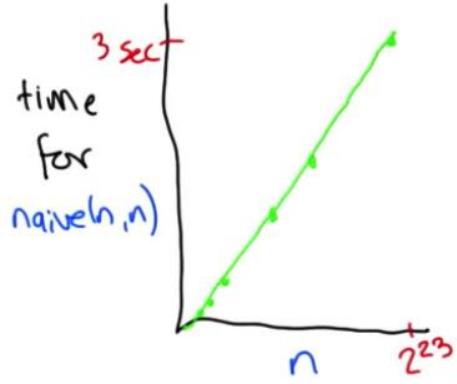
- No, the start and end nodes must have odd degree
- Yes, it depends on the graph, though
- Yes, all such graphs do.
- Yes, all graphs do.

## Case Study

What does  
 $\text{naive}(a, b)$   
compute as a  
function of  $a \& b$ ?

- o  $\max(a, b)$
- o  $a - b$
- o  $b - a$
- o  $a + b$
- o  $a * b$

## Running Time



How does running time +  
relate to input  $n$ ?

- o roughly constant  $t \approx c$
- o roughly logarithmic  $t \approx \log n$
- o roughly linear  $t \approx cn$
- o roughly exponential  $t \approx c^n$

## Russian Peasant Algorithm

Bit Shifts

What is  $17 \gg 1$ ?

- 171
- 9
- 8
- 8.5
- 34

## How Many Additions

How many additions for  $\text{Russian}(20, 7)$



## Measuring Time

```
1 # How many units of time will this python code take to run?
2
3 s = 0
4 for i in range(10):
5     s = s + i
6 print s
7
8
9
10
11
12
```



## Counting Steps

```
1 import math
2
3 def time(n):
4     """ Return the number of steps
5     necessary to calculate
6     `print countdown(n)`"""
7     steps = 0
8     steps = 3 + 2 * math.ceil(n/5.0)
9     return steps
10
11 def countdown(x):
12     y = 0
13     while x > 0:
14         x = x - 5
15         y = y + 1
16
17     return math.ceil(y*2)+3
18
19 print countdown(50)
20
```

## Steps for Naïve

```
1 # counting steps in naive as a function of a
2
3 def naive(a, b):
4     x = a
5     y = b
6     z = 0
7     while x > 0:
8         z = z + y
9         x = x - 1
10    return z
11
12 def time(a):
13     # The number of steps it takes to execute naive(a, b)
14     # as a function of a
15     steps = 0
16     steps = 2*a +3
17     return steps
18
19 print (time(5))
```

## Halving

How many times can you  
divide a number  $x$  in half  
(rounding down) before it hits zero?

$x$	1	2	3	4	5	6	7	8	9	10	11
# halvings	1	2	2	3	3	3	3	4	4	4	4
○ $x$								○ $\log_2 x$			
○ $x/2$								○ $\lfloor \log_2 x \rfloor + 1$			

## Recurrence Relation

$$T(a) = \begin{cases} \text{if } a=0, & 1 \\ \text{elif } a \text{ is even,} & 3 + T(a/2) \\ \text{else} & , 3 + T((a-1)/2) \end{cases}$$

$$\circ T(a) = \lfloor \log_2 a \rfloor + 1$$

$$\circ T(a) = 3 \lfloor \log_2 a \rfloor + 3$$

$$\circ T(a) = 3 \lfloor \log_2 a \rfloor + 1$$

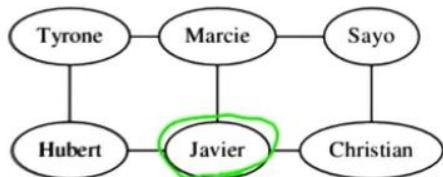
$$\circ T(a) = 3 \lfloor \log_2 a \rfloor + 4$$

## Lección 2

### Eulerian Path

# Eulerian Path

Here is a social network:



- Tyrone
- Marcie
- Sayo
- Hubert
- Javier
- Christian

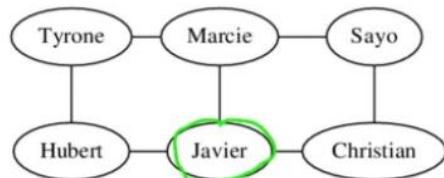
An Eulerian path starting with Javier will end at what node?



## Counting Eulerian Paths

# Eulerian Path

Here is a social network:



How many Eulerian paths does the graph have starting from Javier?

6

## Create Graph with Eulerian Tour

```
import itertools

def create_tour(nodes):
    # nodes = [1,2,3]
    # return [(1,3), (1,2), (2,3)]
    return list(itertools.permutations(nodes, 2))

#####
def get_degree(tour):
    degree = {}
    for x, y in tour:
        degree[x] = degree.get(x, 0) + 1
        degree[y] = degree.get(y, 0) + 1
    return degree

def check_edge(t, b, nodes):
    """
    t: tuple representing an edge
    b: origin node
    nodes: set of nodes already visited
    if we can get to a new node from `b` following `t`
    then return that node, else return None
    """
    if t[0] == b:
        if t[1] not in nodes:
            return t[1]
    elif t[1] == b:
        return t[0]
    return None
```

## Representing a Graph

### Representing a Graph

$$g = [(1,2), (1,4), \dots, (5,9)]$$

What are the advantages and disadvantages of representing a graph as a list of tuples?

Advantage: The simplistic structures allows to compute very easily any task necessary to the case

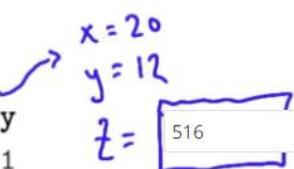
Disadvantage : Is too simple to represent a more complicated structure

## Naïve Multiplication Algorithm

# Naive Multiplication Algorithm

Recall the algorithm `naive` from lecture.

```
def naive(a, b):
    x = a; y = b
    z = 0
    while x > 0:
        z = z + y
        x = x - 1
    return z
```



Let's say we're computing `naive(63, 12)`. At some point during the execution we have  $x = 20$  and  $y = 12$ . What is  $z$ ?

## Recursive Naïve

# Recursive Naive Multiplication Algorithm

Here is a recursive version of the naive multiplication algorithm

```
def rec_naive(a, b):
    if a == 0:
        return 0
    return b + rec_naive(a-1, b)
```

How many additions does it take to compute that `rec_naive(17, 6) = 102?`

17

## Russian Multiplication Algorithm

# Russian Multiplication Algorithm

Recall the algorithm `russian` from the lecture.

```
def russian(a, b):
    x = a; y = b
    z = 0
    while x > 0:
        if x % 2 == 1:
            z = z + y
        y = y << 1
        x = x >> 1
    return z
```

Let's say we're computing `russian(63,12)`. At some point during the execution, we have  $x = 7$  and  $z = 84$ . What is  $y$  at this moment?

## Clique

# Clique

Here's a loop:

```
def clique(n):
    print "in a clique..."
    for j in range(n):
        for i in range(j):
            print i, "is friends with", j
```

How many units of time does it take to execute `clique(4)`? Count each `print` statement as one unit and count each time `range` is evaluated as one unit.

## General Clique

```
1 # Write a function, `count`  
2 # that returns the units of time  
3 # where each print statement is one unit of time  
4 # and each evaluation of range also takes one unit of time  
5  
6 7 def count(n):  
7     # Your code here to count the units of time  
8     # it takes to execute clique  
9     return 2 + (1+n)*n/2  
10  
11 12 def clique(n):  
12     print "in a clique..."  
13     for j in range(n):  
14         for i in range(j):  
15             print i, "is friends with", j  
16  
17 18 if __name__ == '__main__':  
19     print count(4)
```

## Challenge Find Eulerian Tour

```
1 def find_eulerian_tour(graph):  
2  
3     def _next_node(edge, current):  
4         return edge[0] if current == edge[1] else edge[1]  
5  
6     def _remove_edge(raw_list, discard):  
7         return [item for item in raw_list if item != discard]  
8  
9     search = [[[[], graph[0][0], graph]]]  
10    while search:  
11        path, node, unexplore = search.pop()  
12        path += [node]  
13  
14        if not unexplore:  
15            return path  
16  
17        for edge in unexplore:  
18            if node in edge:  
19                search += [[path, _next_node(edge, node), _remove_edge(unexplore, edge)]]  
20  
21 if __name__ == '__main__':  
22     graph = [(1, 2), (2, 3), (3, 1), (3, 4), (4, 3)]  
23     print find_eulerian_tour(graph)|
```

## LECCIÓN 3

### Divisible by Five

YES, the product is divisible by 5  
No, the product is not divisible by 5

Run

```
361 636 277 129 434 577 796 596 727 586
156 109 714 716 548 979 386 766 137 243
331 999 922 304 657 314 634 303 877 597
363 174 431 193 361 877 403 926 279 892
749 401 347 202 783 314 333 244 798 897
674 651 517 349 337 887 617 484 379 793
542 466 961 148 946 199 302 899 606 126
519 203 137 517 146 724 899 699 747 883
126 247 469 953 398 502 582 847 384 214
348 648 331 426 783 291 557 764 939 856
753 581 797 224 537 381 283 493 196 162
382 102 629 936 883 279 966 241 907 677
945 416 122 583 667 394 654 592 977 177
866 199 483 581 954 924 991 383 754 754
199 451 798 586 829 651 517 167 704 749
622 299 486 559 973 243 839 276 803 753
```

Clear IDE Clear Output Pick a template

### Chain Network

Chain Network

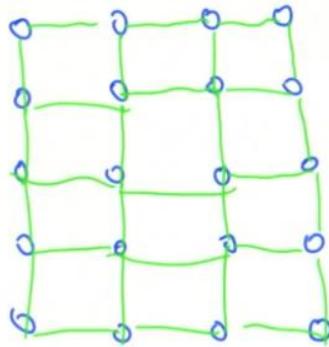
○ ○ ○ ○ ○

$n = 5$  nodes       $m = 4$  edges

○  $m = n - 1$   
○  $m = n$   
○  $m = n + 1$

## Grid Network

### Grid Network



$$n = 20$$

$$M = 15 + 16 = 31$$

If we have 256 nodes arranged as a square grid, how many edges?

480

## Big-Theta Reflexive

### Big $\Theta$ Reflexive?

If  $f(n) \in \Theta(g(n))$ , is  $g(n) \in \Theta(f(n))$ ?

- No, it doesn't follow from the definition.
- Yes, big  $\Theta$  is like " $=$ ", and equality is reflexive
- Yes, because  $\frac{1}{c_1}$  &  $\frac{1}{c_2}$  sandwich  $g$  by  $f$ .

### Big-Theta Practice

$$\underline{2n^2 + 6n + 20\log n} \in \Theta(?)$$

$2n^2 + 6n + 20\log n$

$20\log n$

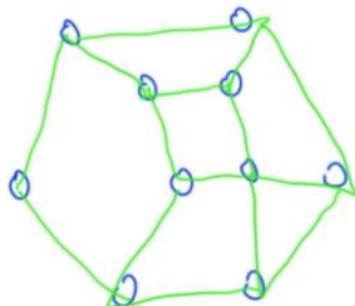
$20n^2$

$n^2$

$n^2 \log n$

### Regions on a Planar Graph

Regions in a Planar Graph



How many regions  
in this graph?

7

## Complete Graph

```
1 def make_link(G,node1,node2):
2     if node1 not in G:
3         G[node1]={}
4         (G[node1])[node2]=1
5     if node2 not in G:
6         G[node2]={}
7         (G[node2])[node1]=1
8     return G
9 #
10 # How many edges in a complete graph on n nodes?
11 #
12
13 def clique(n):
14     # Return the number of edges
15     # Try to use a mathematical formula...
16     #Graph
17     G={}
18     #edges
19     for i in range(n):
20         for j in range(n):
21             if i<j: make_link(G,i,j)
22
23     return sum([len(G[node]) for node in G.keys()])/2
24
25 for n in range (1,10):
26     print n, clique(n),n*(n-1)/2
27
```

## Hypercube Edges

How many edges in an  
n-node hypercube?

- $\Theta(n)$
- $\Theta(n^2)$
- $\Theta(n \log n)$
- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(\sqrt{n})$
- $\Theta(\log^2 n)$



## Tree Graphs

Tree Graphs

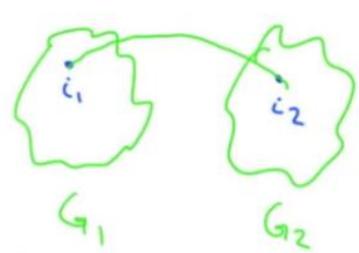
Which graphs are trees?

- 
- 
- 
- 
- 
- 
- 
- 

## Recursive Graphs

What Graph Is This?

```
def makeG(n):  
    if n==1: return a single node  
    G1 = makeG(n/2)  
    G2 = makeG(n/2)  
    i1 = random node from G1  
    i2 = random node from G2  
    make-link(G1, i1, i2)
```



tree  
 chain  
 ring

## Recurrence Relation

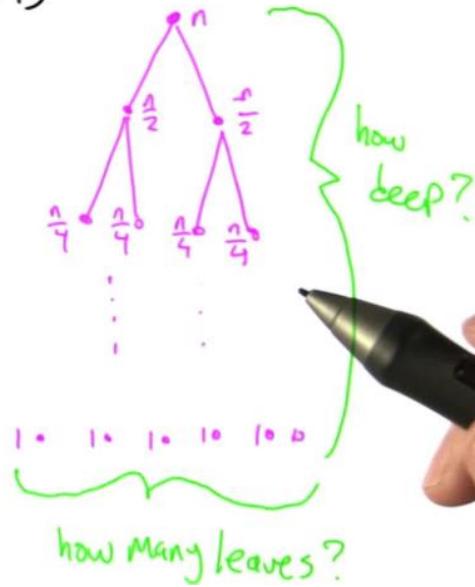
### Recurrence Relations

$T$ : # of edges

$$T(1) = 0$$

$$T(n) = 2T(n/2) + 1$$

depth	leaves
<input checked="" type="checkbox"/> $\log n$	<input type="checkbox"/> $\log n$
<input type="checkbox"/> $n$	<input checked="" type="checkbox"/> $n$
<input type="checkbox"/> $2^n$	<input type="checkbox"/> $2^n$



## Tangled Hypercube

$G1 = \text{makeG}(n/2)$

$G1$

$G2$

$G2 = \text{makeG}(n/2)$

$i1 = \text{list of nodes of } G1 \text{ in random order}$

$i2 = \text{list of nodes of } G2 \text{ in random order}$

for  $i$  in range( $n/2$ ):  $\text{make\_link}(G, i1[i], i2[i])$

return  $G$

What Graph Structure Did we Make?

ring

tree

hypercube

none of

the above

## LECCIÓN 4

### Star Network

```
1 def star_network(n):
2     # return number of edges
3     return n-1
4
5 print star_network(5)
```

### Subsets

<u>Subsets</u>					
<u>X</u>	<u>Y</u>	<u>X c Y</u>	<u>Y c X</u>	<u>Both</u>	<u>Neither</u>
Star	tree	●	○	○	○
planar graphs	trees	○	●	○	○
trees	rings	○	○	○	●
rings	chains	○	○	○	●
chains	trees	●	○	○	○
hypercubes	rings	○	○	○	●
grids	chains	○	●	○	○
planar graphs	hypercubes	○	○	○	●

## Function Comparison

# Function Comparision

$4n^2 + (\log n)^7 + 9n(\log n)^2 + n^{\frac{2}{3}}$  is

- $\Theta(n^2)$
- $\Theta((\log n)^7)$
- $\Theta(n^{\frac{2}{3}})$
- $O(n)$
- $O(n^2)$
- $O(n^3)$

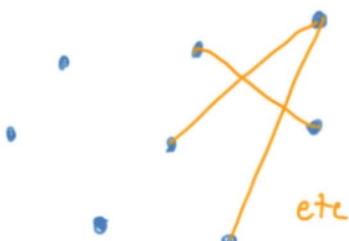
## Planar Graphs

# Planar Graphs

Draw yourself a planar graph with 8 nodes and 15 edges.

How many regions does the graph have?

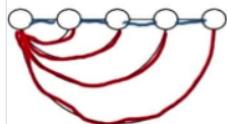
9



## Combination Locks

# Combination Lock

A “combination lock” graph on  $n$  nodes consists of the edges in a chain **and** the edges in a star, with the left end of the chain as the center of the star. Here’s the five node version:



The number of edges is

- $\Theta(n)$
- $\Theta(n \log n)$
- $\Theta(n^2)$



## Make a Combination Lock

```
10 def create_combo_lock(nodes):
11     G = []
12     # your code here
13     make_link(G, nodes[0], nodes[1])
14     for a in range(2, len(nodes)):
15         make_link(G, nodes[a-1], nodes[a])
16         make_link(G, 0, nodes[a])
17     return G
18
```

Erdos-Renyi

## Erdos-Renyi Graph

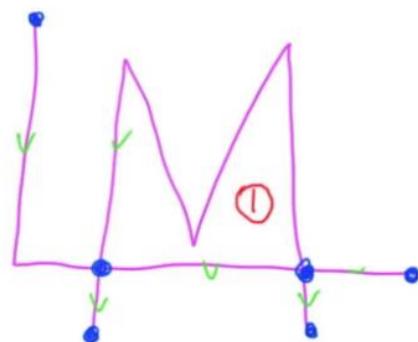
Imagine generating an Erdos-Renyi graph with  $n = 256$  and  $p = 0.25$ .  
On average, how many edges will it have?

8160

LECCIÓN 6

Initial Foray

Initial Foray



DOTS  
+ REGIONS  
- SEGMENTS  
=

## Properties of Social Networks

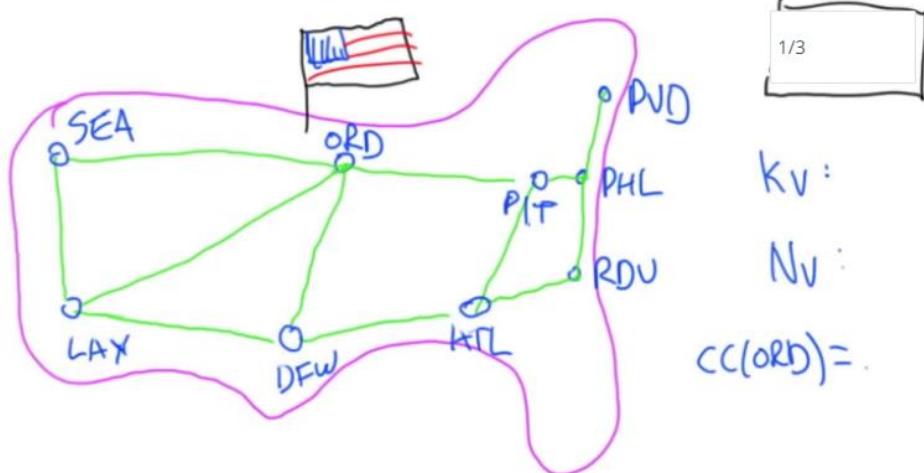
### QUIZ : Degree & Paths in Graphs

	clique	ring	balanced tree	hypercube
degree path	☒	☒	☒	☒
$\Theta(1)$	☒	☒	☒	☒
$\Theta(\log n)$	☒	☒	☒	☒
$\Theta(n)$	☒	☒	☒	☒
	☒	☒	☒	☒

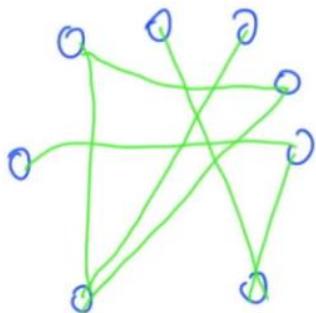
## Clustering Coefficient Quiz

### Quiz $CC(ORD)$



## Connected Components

### Counting Component Sizes



How many disconnected  
Components?

2|  
— .

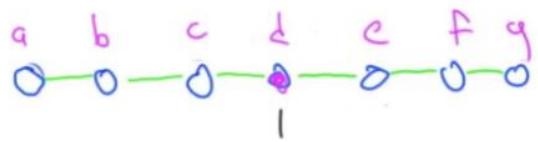
## Checking Pairwise Connectivity

```
1 def check_connection(G, v1, v2):
2     marked = []
3     mark_component (G,v1,marked)
4
5     return v2 in marked
```

## Depth First without Recursion

### Depth First without recursion

- Grab last element of open list



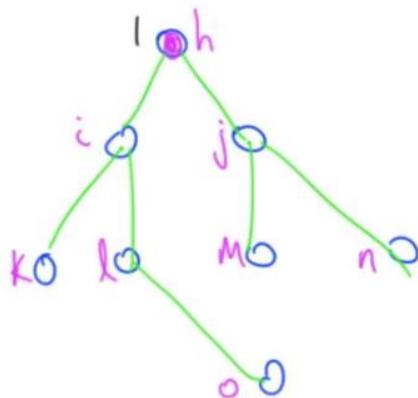
- mark any unmarked neighbors and add to open list  
- repeat until nothing open

OPEN LIST: d  
(TODO LIST)

$$a + g = \boxed{12}$$

## Searching a Tree

When Expanded?



Starting from h  
and using breadth  
first search, when  
will node o be  
added to the open list?



## Single Source Shortest Paths

All targets in one shot

How can we find all distances from  
v<sub>i</sub> to the rest of the graph faster?

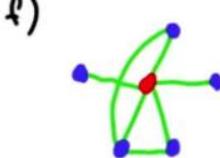
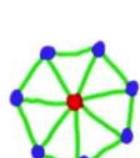
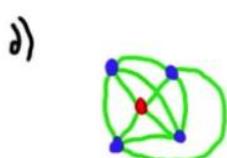
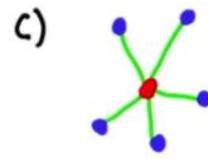
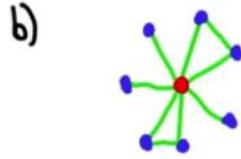
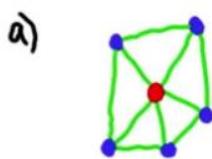
$\Theta(n^2 + nm)$

- Ⓐ You can't. Deal with it.
- Ⓑ Use a smaller graph
- Ⓒ Search for something that's not there
- Ⓓ Do the searches backwards

## LECCIÓN 7

### Clustering Coefficient

## Clustering Coefficient



Order by clustering coefficient of the red node of each graph - lowest to highest

cbfead

### Bipartite I

## Bipartite Graphs

Consider a bipartite graph,  $B$ , that has five nodes in one group and three in the other

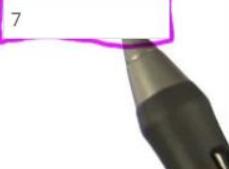
LEFT



RIGHT



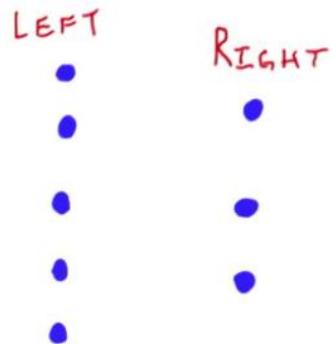
- i) What is the smallest number of edges needed to make  $B$  have a single reachable component consisting of all the nodes?



## Bipartite II

### Bipartite Graphs

Consider a bipartite graph,  $B$ , that has five nodes in one group and three in the other

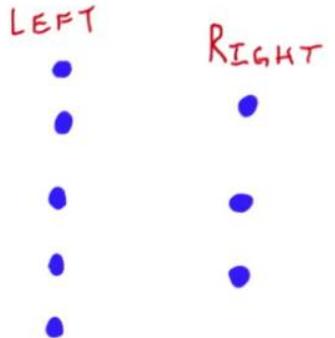


- 1) What is the smallest number of edges needed to make  $B$  have a single reachable component consisting of all the edges?
- 2) What is the maximum number of edges  $B$  can have? 15

## Bipartite III

### Bipartite Graphs

Consider a bipartite graph,  $B$ , that has five nodes in one group and three in the other



- 3) What is the maximum possible path length in  $B$ ? 6



## Bipartite IV

Bipartite Graphs

Consider a bipartite graph,  $B$ , that has five nodes in one group and three in the other

LEFT	RIGHT
•	•
•	•
•	•
•	•

3) What is the maximum possible path length in  $B$ ?  

4) What is the maximum possible clustering coefficient for a node in  $B$ ? 0

ONLY CONSIDER NODES DEGREE  $\geq 2$ .

## Mark Component

```
5
6 def mark_component(G, node, marked):
7     open_list = [node]
8     total_marked = 1
9     marked[node] = True
10    while len(open_list) > 0:
11        node = open_list.pop()
12        for neighbor in G[node]:
13            if neighbor not in marked:
14                open_list.append(neighbor)
15                marked[neighbor] = True
16                total_marked += 1
17    return total_marked
18
19
```

## Centrality

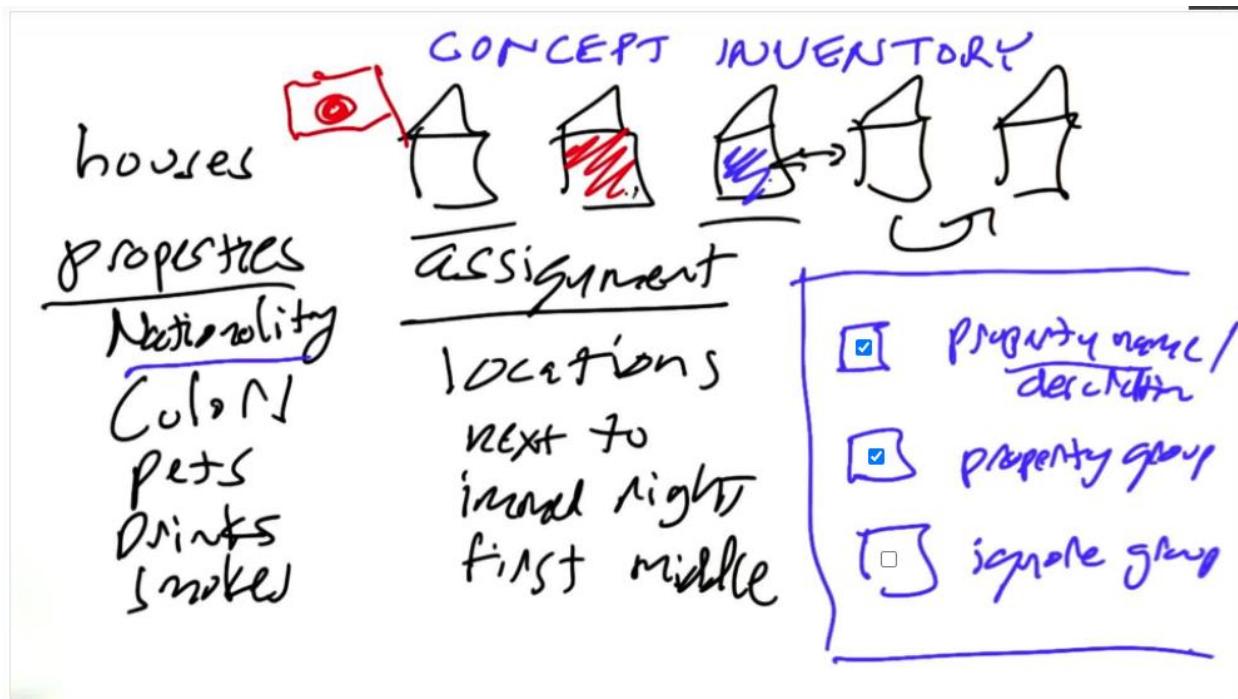
```
6 def centrality_max(G, v):
7     distance_from_start = {}
8     open_list = [v]
9     distance_from_start[v] = 0
10    while len(open_list) > 0:
11        current = open_list[0]
12        del open_list[0]
13        for neighbor in G[current].keys():
14            if neighbor not in distance_from_start:
15                distance_from_start[neighbor] = distance_from_start[current] + 1
16                open_list.append(neighbor)
17    return max(distance_from_start.values())
18
```

## Bridge Edges

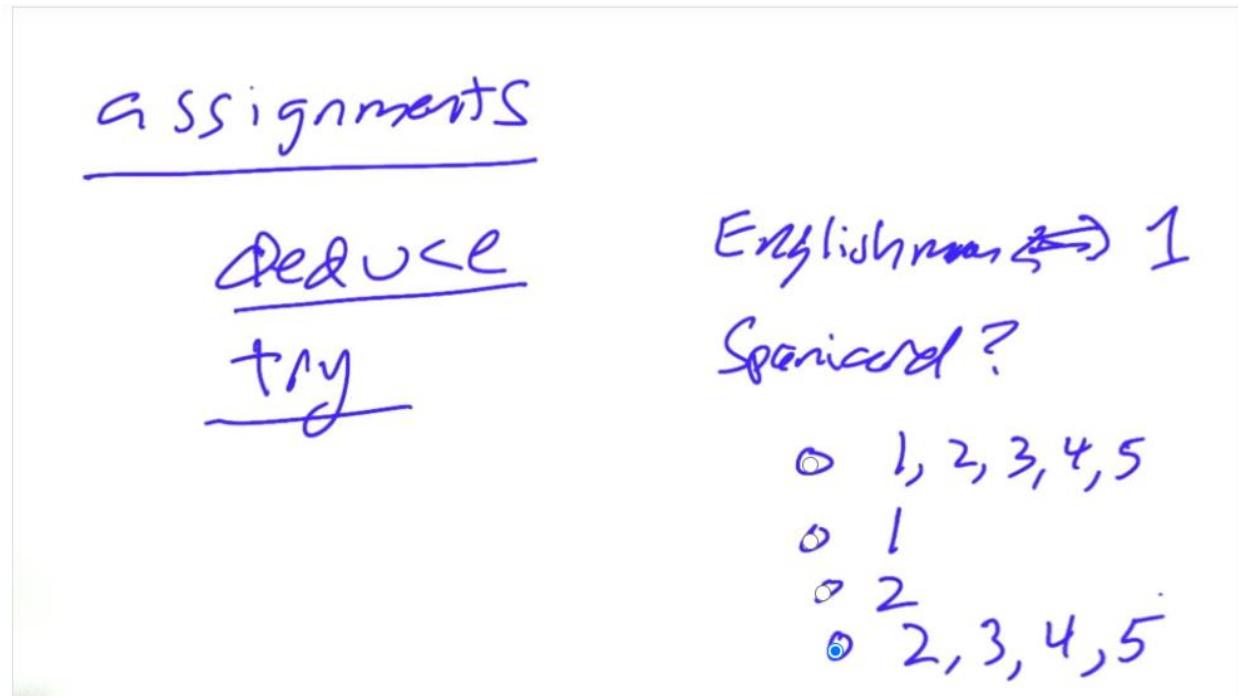
```
1 #
2 # First some utility functions
3 #
4
5 def make_link(G, node1, node2, r_or_g):
6     # modified make_link to apply
7     # a color to the edge instead of just 1
8     if node1 not in G:
9         G[node1] = {}
10    (G[node1])[node2] = r_or_g
11    if node2 not in G:
12        G[node2] = {}
13    (G[node2])[node1] = r_or_g
14    return G
15
16 def get_children(S, root, parent):
17     """returns the children from following the
18     green edges"""
19     return [n for n, e in S[root].items()
20             if ((not n == parent) and
21                 (e == 'green'))]
22
23 def get_children_all(S, root, parent):
24     """returns the children from following
25     green edges and the children from following
26     red edges"""
27     green = []
28     red = []
29     for n, e in S[root].items():
30         if e == 'green':
```

## LECCIÓN 8

### Zebra Puzzle

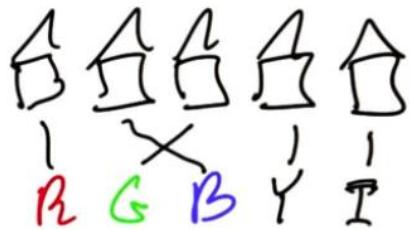


Where The Spaniard



## Counting Assignments

Assignments



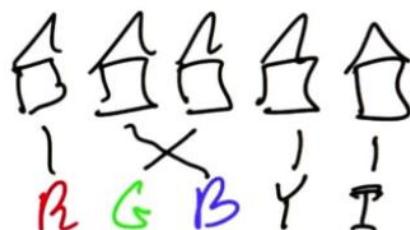
- o 5
- o  $5^2 = 25$
- o  $2^5 = 32$
- o  $5! = 120$

## Multiple Properties

Assignments

All 5 properties

- o  $5 \cdot 5!$
- o  $5!^2$
- o  $5!^5$
- o  $5!^3$



one property

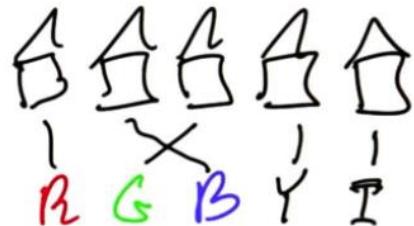
- o 5
- o  $5^2 = 25$
- o  $2^5 = 32$
- o  $\textcircled{5} 5! = 120$

Back of the Envelope

$5!^5 \approx$  0 1 million  
o 20 billion  
o 5 trillion

All 5 properties

- o  $5 \cdot 5!$
- o  $5!^2$
- o  $\textcircled{0} 5!^5$
- o  $5!^3$



one property

- o  $5^5 = 25$
- o  $2^5 = 32$
- o  $\textcircled{0} 5! = 120$

Leaving Happy Valley

Assignment

red  $\leftrightarrow$

- $\text{house}[1].add('red')$  set()
- $\text{house}[1].color = 'red'$  House()
- $\text{red} = 1$  ~

## Ordering Houses

houses = [1, 2, 3, 4, 5]

orderings =  $\text{F}(\text{houses})$

for (red, green, ivory, yellow, blue) in orderings:

- o permutations
- o combinations
- o factorial
- o other

## Length Of Orderings

houses = [1, 2, 3, 4, 5]

orderings =  $\text{F}(\text{houses})$

for (red, green, ivory, yellow, blue) in orderings:

- o permutations
- o combinations
- o factorial
- o other

import  
len(orderings)

120

## Estimating Runtime

```
1
2
3
4 import itertools
5
6 houses = [1, 2, 3, 4, 5]
7 orderings = list(itertools.permutations(houses))
8
9 for (red, green, ivory, yellow, blue) in orderings:
10    for (Englishman, Spaniard, Ukrainian, Japanese, Norwegian) in orderings:
11        for (dog, snails, fox, horse, ZEBRA) in orderings:
12            for (coffee, tea, milk,oj, WATER) in orderings:
13                for (OldGold, Kools, Chesterfields, LuckyStrike, Parliaments) in orderings:
14                    # constraints go here
15
16
17          ○ 1 sec
18          ○ 1 min
19          ○ 1 hour
20          ○ 1 day
```

### Red Englishman

```
4 import itertools
5
6 houses = [1, 2, 3, 4, 5]
7 orderings = list(itertools.permutations(houses))
8
9 for (red, green, ivory, yellow, blue) in orderings:
10    for (Englishman, Spaniard, Ukrainian, Japanese, Norwegian) in orderings:
11        for (dog, snails, fox, horse, ZEBRA) in orderings:
12            for (coffee, tea, milk,oj, WATER) in orderings:
13                for (OldGold, Kools, Chesterfields, LuckyStrike, Parliaments) in orderings:
14                    if ( Englishman == red ): #2|
15
16
17          # 1 sec
18          # 1 min
19          # 1 hour
20          # 1 day
```

## Neighbors

```
7
8 import itertools
9
10 houses = [1, 2, 3, 4, 5]
11 orderings = list(itertools.permutations(houses))
12
13 def imright(h1, h2):
14     "House h1 is immediately right of h2 if h1-h2 == 1."
15     return h1-h2 == 1
16
17 def nextto(h1, h2):
18     "Two houses are next to each other if they differ by 1."
19     return abs(h1-h2) == 1
20
```

## Generator Expressions

GENERATOR EXPRESSION  
(terms for-clause optional-for-ifs...)

- config students
- less indentation
- stop early
- easier to edit

## Eliminating Redundancy

```
def zebra_puzzle():
    "Return a tuple (WATER, ZEBRA) indicating their house numbers."
    houses = first, _, middle, _, _ = [1, 2, 3, 4, 5]
    orderings = list(itertools.permutations(houses)) #1
    return next((WATER, ZEBRA)
        for (red, green, ivory, yellow, blue) in orderings
            for (Englishman, Spaniard, Ukrainian, Japanese, Norwegian) in orderings
                if Englishman is red #2
                for (dog, snails, fox, horse, ZEBRA) in orderings
                    for (coffee, tea, milk, oj, WATER) in orderings
                        for (OldGold, Kools, Chesterfields, LuckyStrike, Parliaments) in orderings
                            if Spaniard is dog #3
```

## Good Science

### TIMING — MEASUREMENTS

mathematics)

experimental science

REPEAT 0.0

- reduce external error
- reduce randomness
- reduce errors



## Timed Calls

```
6
7 def timedcalls(n, fn, *args):
8     """Call fn(*args) repeatedly: n times if n is an int, or up to
9     n seconds if n is a float; return the min, avg, and max time"""
10    if isinstance(n, int):
11        times = [timedcall(fn, *args)[0] for _ in range(n)]
12    else:
13        times = []
14        while sum(times) < n:
15            times.append(timedcall(fn, *args)[0])
16    return min(times), average(times), max(times)
17
```

## Yielding Results

GENERATOR FUNCTIONS

def ints(start, end=None):  
 i = start  
 while i <= end:  
 yield i  
 i = i + 1

ints(0)

## All Ints

```
14 def all_ints():
15     "Generate integers in the order 0, +1, -1, +2, -2, +3, -3, ..."
16     yield 0
17     for i in ints(1):
18         yield +i
19         yield -i
```

## Cryptarithmetic

$$\begin{array}{r} \text{O D D} \\ + \text{O D D} \\ \hline \text{E V E N} \end{array}$$

- 1
- 2
- 3
- 4
- 9

Cryptarithmetic

## Odd or Even

- o odd
- o even

$$\begin{array}{r} \textcircled{O} \text{ } \textcircled{D} \text{ } \textcircled{D} \\ + \textcircled{O} \text{ } \textcircled{D} \text{ } \textcircled{D} \\ \hline \text{E} \text{ } \text{V} \text{ } \text{E} \text{ } \text{N} \end{array}$$

o 1  
o 2  
o 3  
o 4  
o 9

Cryptarithmetic

## Solving Cryptarithmetic

```
1 def solve(formula):
2     """Given a formula like 'ODD + ODD == EVEN', fill in digits to
3     Input formula is a string; output is a digit-filled-in string
4     for f in fill_in(formula):
5         if valid(f):
6             return f
```

## Filling In Fill In

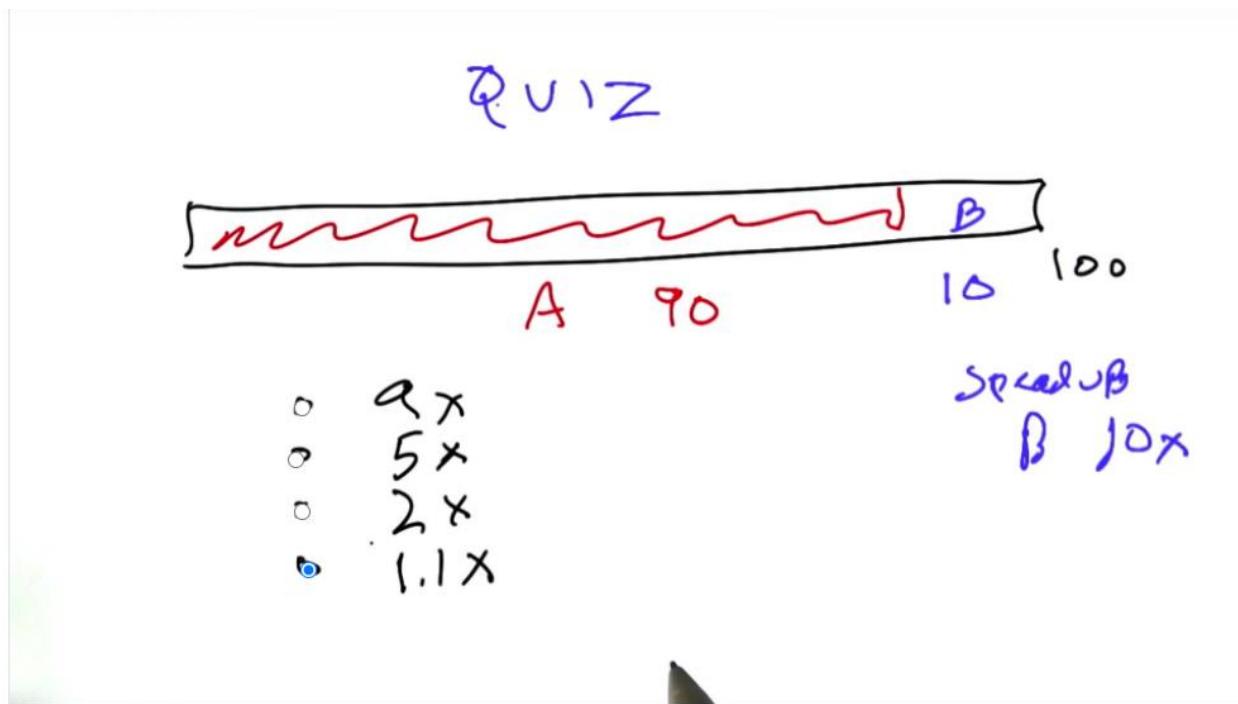
```
def fill_in(formula):
    "Generate all possible fillings-in of letters in formula with digits."
    letters = ''.join(set(re.findall('[A-Z]', formula))) #should be a string
    for digits in itertools.permutations('1234567890', len(letters)):
        table = string.maketrans(letters, ''.join(digits))
        yield formula.translate(table)
```

## Find All Values

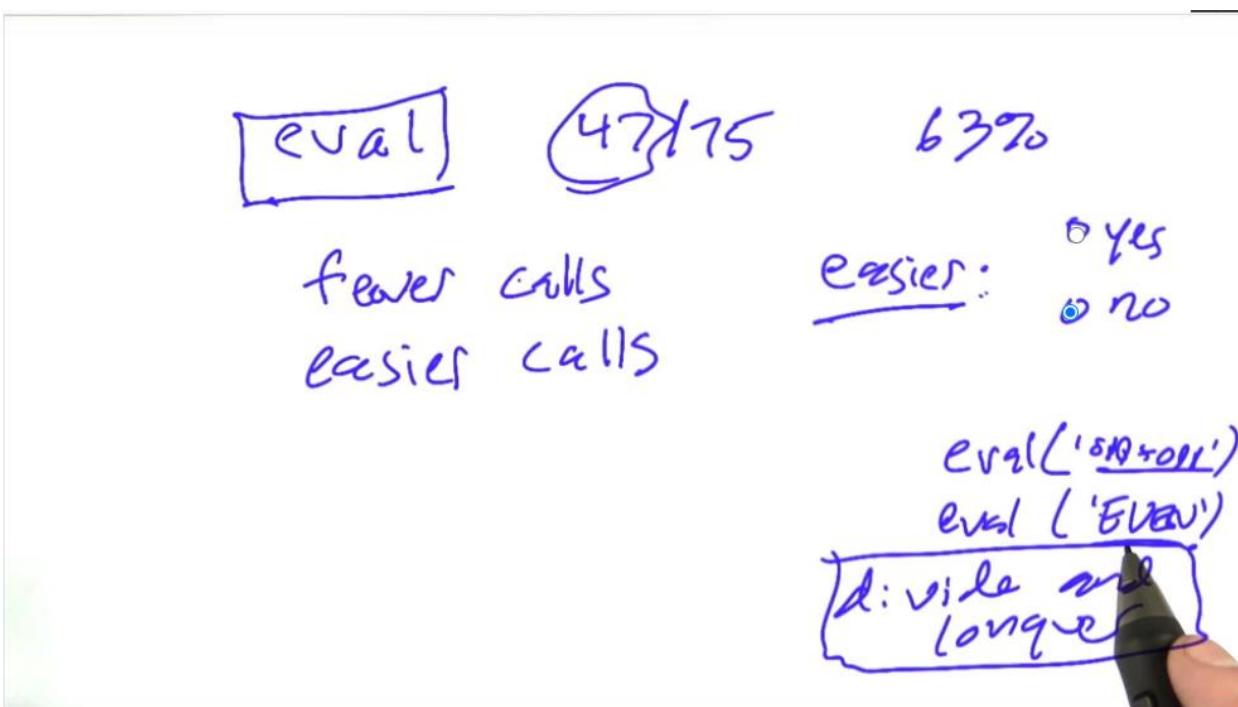
all values

- o fill-in  $\rightarrow$  list
- o "return" to "print" in solve
- o change valid
- o add new function

## Increasing Speed



## Rethinking Eval



## Making Fewer Calls

eval

47/75

63%

fewer calls  
~~easier~~ calls

easier:  
① Yes  
② No

- ① one big formula
- ② fill in one digit (zero)
- ③ eval formats once as addition with priorities

eval('5\*10+100')

eval('EVAL')

divide and conquer

## Compile Word

```
def compile_word(word):
    """Compile a word of uppercase letters as numeric digits.
    E.g., compile_word('YOU') => '(1*U+10*O+100*Y)'
    Non-uppercase words unchanged: compile_word('+') => '+''''
    if word.isupper():
        terms = [('%s%s' % (10**i, d))
                 for (i, d) in enumerate(word[::-1])]
        return '(' + ' + '.join(terms) + ')'
    else:
        return word
```

## LECCIÓN 9

### Introduction

#### Magic trick

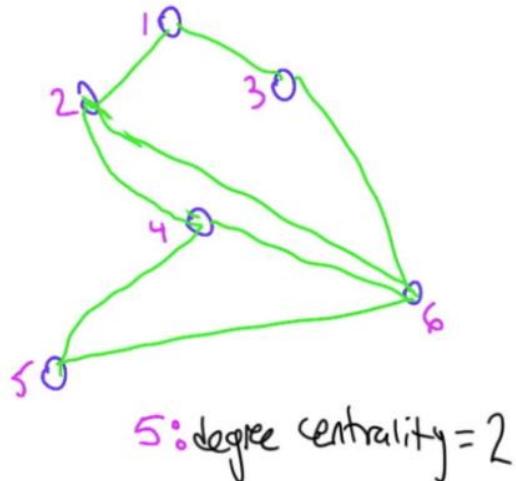
0  
1  
2  
3  
4  
5  
:  
10

$\Theta(-n)$

- frog 
- rabbit 
- elephant 
- horse 
- shark 

### Degree Centrality

#### Degree Centrality



Which node has the highest degree centrality in the network?



## Statistics By Sorting

### Quiztistics

21 43 48 49 50 51 75 77 79 87 93

Find the midpoint and the median.

What's their mean?

54

## Mean

### Computing the Mean

$$\mu = \frac{\sum_{i=0}^{n-1} L[i]}{n}$$



def mean(L):

total = 0

for i in range(len(L)):

    total += L[i]

return (total + 0.0) / len(L)

What is the  
running time of  
mean?

Θ(1)

Θ(log n)

Θ(n)

Θ(n log n)

Θ(n^2)

## Extreme Values

```
1 #  
2 # Write `max`  
3 #  
4  
5 def max(L):  
6     max_so_far = L[0]  
7     for i in range(1, len(L)):  
8         if L[i] > max_so_far:  
9             max_so_far = L[i]  
10    return max_so_far  
11  
12 def test():  
13     L = [1, 2, 3, 4]  
14     assert 4 == max(L)  
15     L = [3, 6, 10, 9, 3]  
16     assert 10 == max(L)  
17  
18  
19
```

## Order Statistics

2<sup>nd</sup> Most Popular Name  
yob1995.txt

write python code to find the 2<sup>nd</sup> most popular female name given in the US in 1995.

- Michael
- Ashley
- Jessica
- Matthew
- Zuzanna
- Taylor

## Top K Problem

	n/2	$\sqrt{n}$	$\log n$	$\log \log n$	100
SELECTION /	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
INSERTION					
SORT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

$n$  elements in L, want top  $k$

## Partitioning Around V

```
7 def partition(L, v):
8     smaller = []
9     bigger = []
10    for each in L:
11        if each < v:
12            smaller.append(each)
13        if each > v:
14            bigger.append(each)
15    # your code here
16    return smaller+[v]+bigger
17
```

## Heaps of Fun

	Heaps via Lists	$\Theta(1)$	$\Theta(\log n)$	$\Theta(n)$
Insert				
ordered list	☒	☒	☒	
unordered list	☒	☒	☒	
find/remove min				
ordered list	☒	☒	☒	
unordered list	☒	☒	☒	

## Heap Height

Properties of a Heap 1	
Nodes are filled in left to right, top to bottom. Height, root to leaf, is:	
• $\Theta(1)$	• $\Theta(\sqrt{n})$
• $\Theta(\log n)$	• $\Theta(n)$

$n=20$

## Properties of a Heap

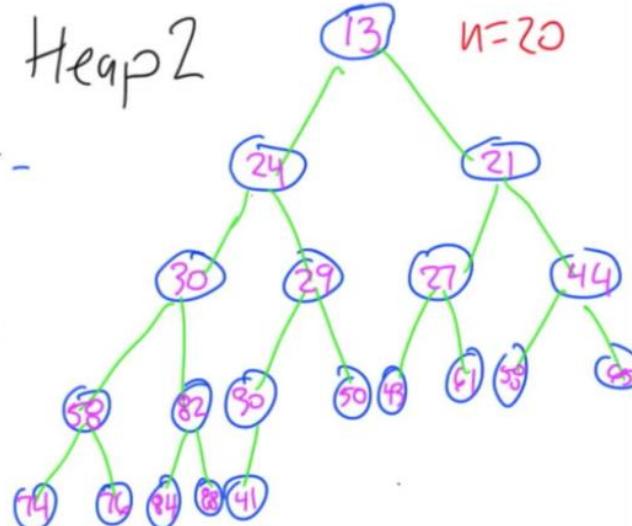
### Properties of a Heap 2

Smallest value is at root -

easy to find!

What's the deepest level  
you can find the  
3rd smallest?

3



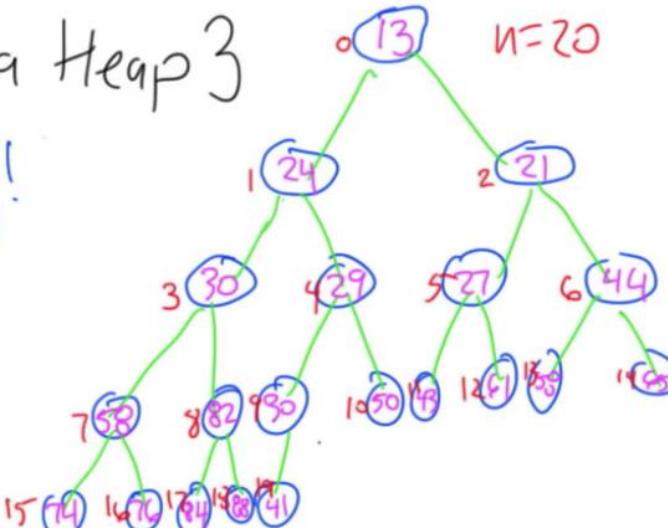
## Heap Number

### Properties of a Heap 3

Can store in an array!

If this tree were  
bigger, what node  
would be the right  
child of 72?

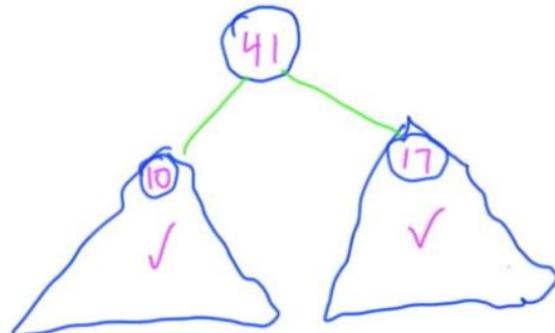
146



## Establishing the Heap Property

### Establishing the Heap Property

Any one-node tree satisfies the heap property. Imagine we have two subtrees satisfying the property - how add a root?

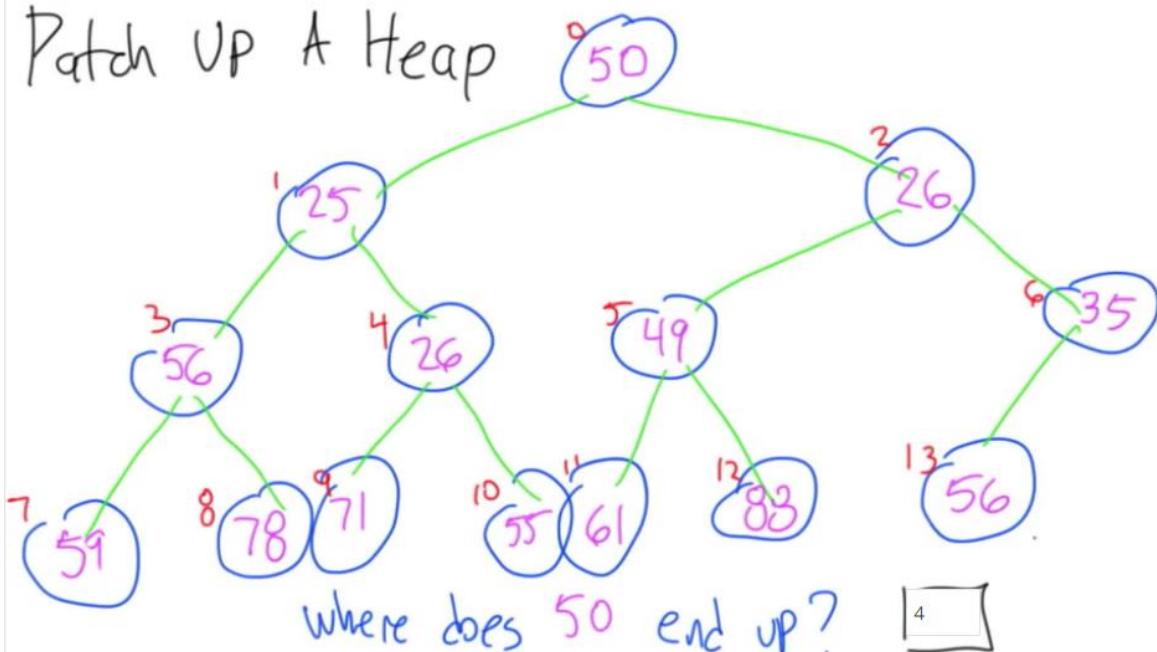


Which value can we swap to the top?

10

## Patch Up A Heap

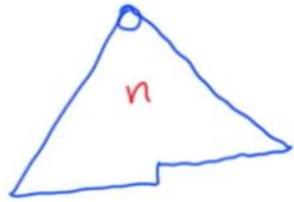
### Patch Up A Heap



## Running Time of Down Heapify

Running Time of down-heapify( $L, i$ )?

- $\Theta(1)$
- $\Theta(\sqrt{n})$
- $\Theta(\log n)$
- $\Theta(n)$



## Remove Min

```
4
5 def remove_min(L):
6     L[0] = L.pop()
7     down_heapify(L, 0)
8     return L
9
```

## Heap Sort Performance

heap sort

```
def heap_sort(L):
    build_heap(L, 0)
    while len(L) > 0:
        print L[0]
        remove_min(L)
```

running time?

- $\Theta(n \log n)$
- $\Theta(n)$
- $\Theta(n \log n)$
- $\Theta(n^2)$

## LECCIÓN 10

### Build a Heap

## BUILD A HEAP

Another way to build a heap out a set of values, `vals`, is to insert the items one at a time into the heap.

```
1 heap = []
2 for v in vals:
3     insert_heap(heap, v)
```

What is the running time?

- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(n \log n)$
- $\Theta(n^2)$

## Minimize Sum of Absolute Value

```
9 def minimize_absolute(L):
10    L2 = L[:]
11    L2.sort()
12    if len(L2) % 2 == 0:
13        return (L2[(len(L2) - 1)/ 2] + L2[(len(L2) - 1)/ 2]) / 2.
14    else:
15        return L2[(len(L2) - 1)/ 2]
16
```

## Minimize Sum of Squares

```
7 #
8 def minimize_square(L):
9     return sum(L) * 1. / len(L)
```

## Mode

```
9 def mode(L):
10    counts = {}
11    for each in L:
12        if each not in counts:
13            counts[each] = 0
14            counts[each] += 1
15    x = 0
16    y = 0
17    for each in counts:
18        if counts[each] > x:
19            x = counts[each]
20            y = each
21    return y
22
```

## Up Heapify

```
9 def up_heapify(L, i):
10    while i > 0 and L[i] < L[parent(i)]:
11        L[parent(i)], L[i] = L[i], L[parent(i)]
12        i = parent(i)
```

## Actor Centrality

# Actor Centrality

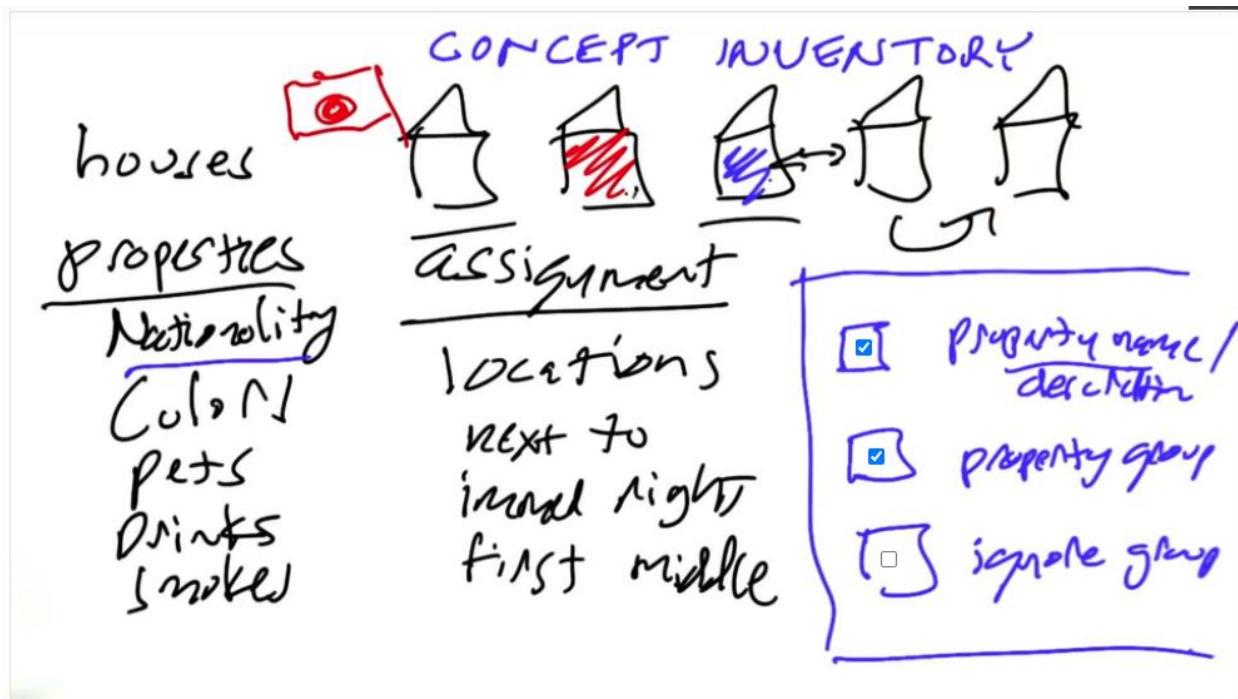
Using the supplied file 'imdb-1.tsv' calculate the top 20 central actors, using the average centrality measurement given in Unit 3. The top central actor is *Tatasciore, Fred*. Who is the 20th?

- Jackson, Samuel L.
- Hoffman, Dustin
- De Niro, Robert
- Morrison, Rana

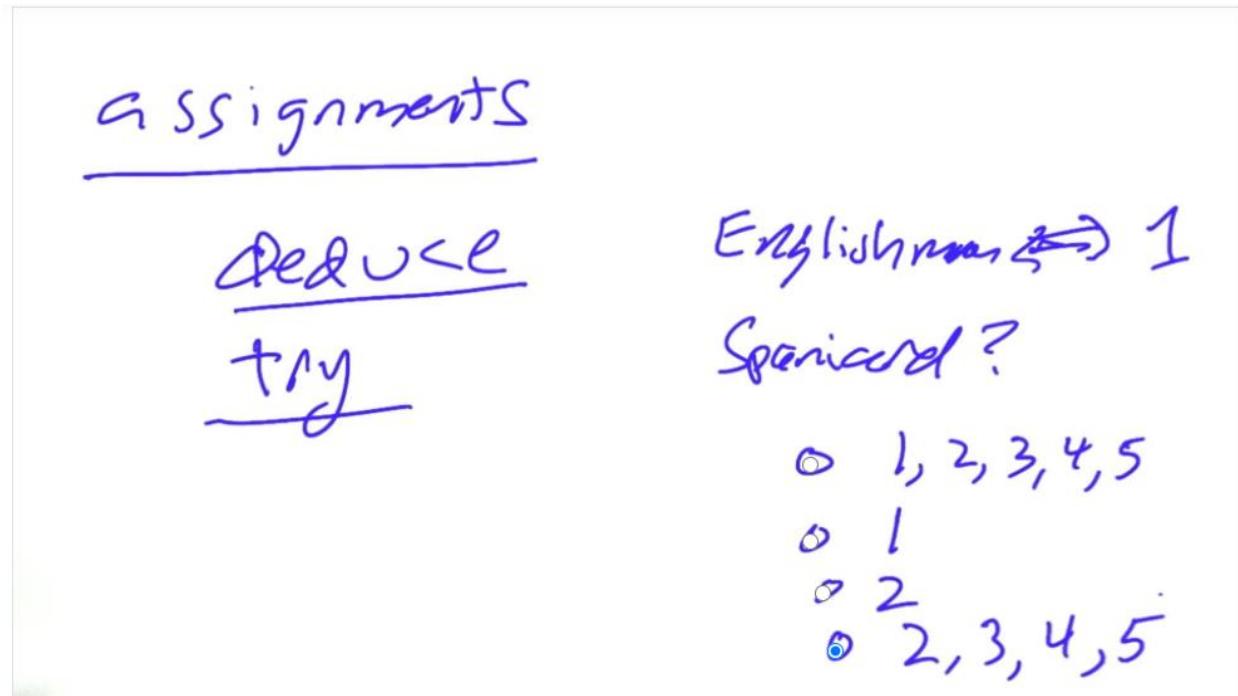
What is his/her centrality?

## LECCIÓN 11

### Zebra Puzzle

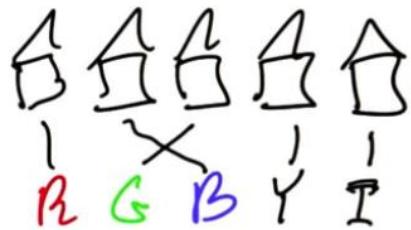


Where The Spaniard



## Counting Assignments

Assignments

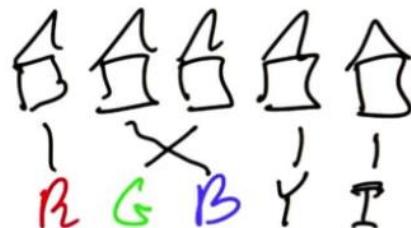


- o 5
- o  $5^2 = 25$
- o  $2^5 = 32$
- o  $5! = 120$

## Multiple Properties

Assignments

- All 5 properties
- o  $5 \cdot 5!$
  - o  $5!^2$
  - o  $5!^5$
  - o  $5!^3$



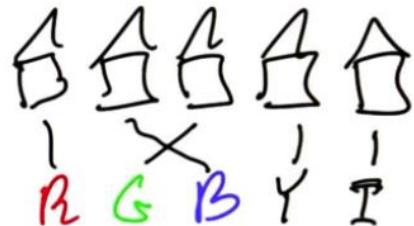
- one property
- o 5
  - o  $5^2 = 25$
  - o  $2^5 = 32$
  - o  $5! = 120$

Back of the Envelope

$5!^5 \approx$  0 1 million  
0 20 billion  
0 5 trillion

All 5 properties

- 0  $5 \cdot 5!$
- 0  $5!^2$
- 0  $5!^5$
- 0  $5!^3$



one property

$$\begin{aligned}0 & 5 \\0 & 5^2 = 25 \\0 & 2^5 = 32 \\0 & 5! = 120\end{aligned}$$

Leaving Happy Valley

Assignment

red  $\leftrightarrow$

- |                                     |                        |         |
|-------------------------------------|------------------------|---------|
| <input checked="" type="checkbox"/> | house[1].add('red')    | set()   |
| <input checked="" type="checkbox"/> | house[1].color = 'red' | House() |
| <input checked="" type="checkbox"/> | red = 1                | $\sim$  |

## Ordering Houses

houses = [1, 2, 3, 4, 5]

orderings =  $\text{F}(\text{houses})$

for (red, green, ivory, yellow, blue) in orderings:

- o permutations
- o combinations
- o factorial
- o other

## Length Of Orderings

houses = [1, 2, 3, 4, 5]

orderings =  $\text{F}(\text{houses})$

for (red, green, ivory, yellow, blue) in orderings:

- o permutations
- o combinations
- o factorial
- o other

import  
len(orderings)

120

## Estimating Runtime

```
1
2
3
4 import itertools
5
6 houses = [1, 2, 3, 4, 5]
7 orderings = list(itertools.permutations(houses))
8
9 for (red, green, ivory, yellow, blue) in orderings:
10    for (Englishman, Spaniard, Ukrainian, Japanese, Norwegian) in orderings:
11        for (dog, snails, fox, horse, ZEBRA) in orderings:
12            for (coffee, tea, milk,oj, WATER) in orderings:
13                for (OldGold, Kools, Chesterfields, LuckyStrike, Parliaments) in orderings:
14                    # constraints go here
15
16
17          ○ 1 sec
18          ○ 1 min
19          ○ 1 hour
20          ○ 1 day
```

### Red Englishman

```
4 import itertools
5
6 houses = [1, 2, 3, 4, 5]
7 orderings = list(itertools.permutations(houses))
8
9 for (red, green, ivory, yellow, blue) in orderings:
10    for (Englishman, Spaniard, Ukrainian, Japanese, Norwegian) in orderings:
11        for (dog, snails, fox, horse, ZEBRA) in orderings:
12            for (coffee, tea, milk,oj, WATER) in orderings:
13                for (OldGold, Kools, Chesterfields, LuckyStrike, Parliaments) in orderings:
14                    if ( Englishman == red ): #2|
15
16
17          # 1 sec
18          # 1 min
19          # 1 hour
20          # 1 day
```

## Neighbors

```
7
8 import itertools
9
10 houses = [1, 2, 3, 4, 5]
11 orderings = list(itertools.permutations(houses))
12
13 def imright(h1, h2):
14     "House h1 is immediately right of h2 if h1-h2 == 1."
15     return h1-h2 == 1
16
17 def nextto(h1, h2):
18     "Two houses are next to each other if they differ by 1."
19     return abs(h1-h2) == 1
20
```

## Generator Expressions

GENERATOR EXPRESSION  
(terms for-clause optional-for-ifs...)

- config students
- less indentation
- stop early
- easier to edit

## Eliminating Redundancy

```
def zebra_puzzle():
    "Return a tuple (WATER, ZEBRA) indicating their house numbers."
    houses = first, _, middle, _, _ = [1, 2, 3, 4, 5]
    orderings = list(itertools.permutations(houses)) #1
    return next((WATER, ZEBRA)
        for (red, green, ivory, yellow, blue) in orderings
            for (Englishman, Spaniard, Ukrainian, Japanese, Norwegian) in orderings
                if Englishman is red #2
                for (dog, snails, fox, horse, ZEBRA) in orderings
                    for (coffee, tea, milk, oj, WATER) in orderings
                        for (OldGold, Kools, Chesterfields, LuckyStrike, Parliaments) in orderings
                            if Spaniard is dog #3
```

## Good Science

### TIMING — MEASUREMENTS

mathematics)

experimental science

REPEAT

0.0

- reduce external error
- reduce randomness
- reduce errors



## Timed Calls

```
6
7 def timedcalls(n, fn, *args):
8     """Call fn(*args) repeatedly: n times if n is an int, or up to
9     n seconds if n is a float; return the min, avg, and max time"""
10    if isinstance(n, int):
11        times = [timedcall(fn, *args)[0] for _ in range(n)]
12    else:
13        times = []
14        while sum(times) < n:
15            times.append(timedcall(fn, *args)[0])
16    return min(times), average(times), max(times)
17
```

## Yielding Results

GENERATOR FUNCTIONS

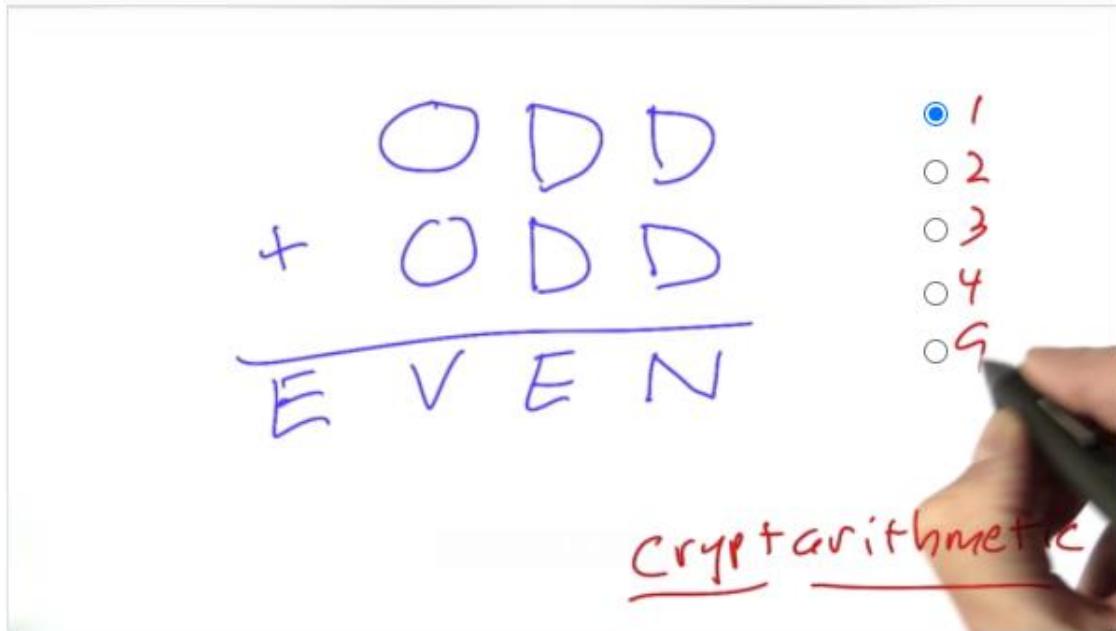
def ints(start, end=None):  
 i = start  
 while i <= end:  
 yield i  
 i = i + 1

ints(0)

## All Ints

```
14 def all_ints():
15     "Generate integers in the order 0, +1, -1, +2, -2, +3, -3, ..."
16     yield 0
17     for i in ints(1):
18         yield +i
19         yield -i
```

## Cryptarithmetic



## Odd or Even

- o odd
- o even

$$\begin{array}{r} \textcircled{O} \text{ } \textcircled{D} \text{ } \textcircled{D} \\ + \textcircled{O} \text{ } \textcircled{D} \text{ } \textcircled{D} \\ \hline \text{E} \text{ } \text{V} \text{ } \text{E} \text{ } \text{N} \end{array}$$

o 1  
o 2  
o 3  
o 4  
o 9

Cryptarithmetic

## Solving Cryptarithmetic

```
1 def solve(formula):
2     """Given a formula like 'ODD + ODD == EVEN', fill in digits to
3     Input formula is a string; output is a digit-filled-in string
4     for f in fill_in(formula):
5         if valid(f):
6             return f
```

## Filling In Fill In

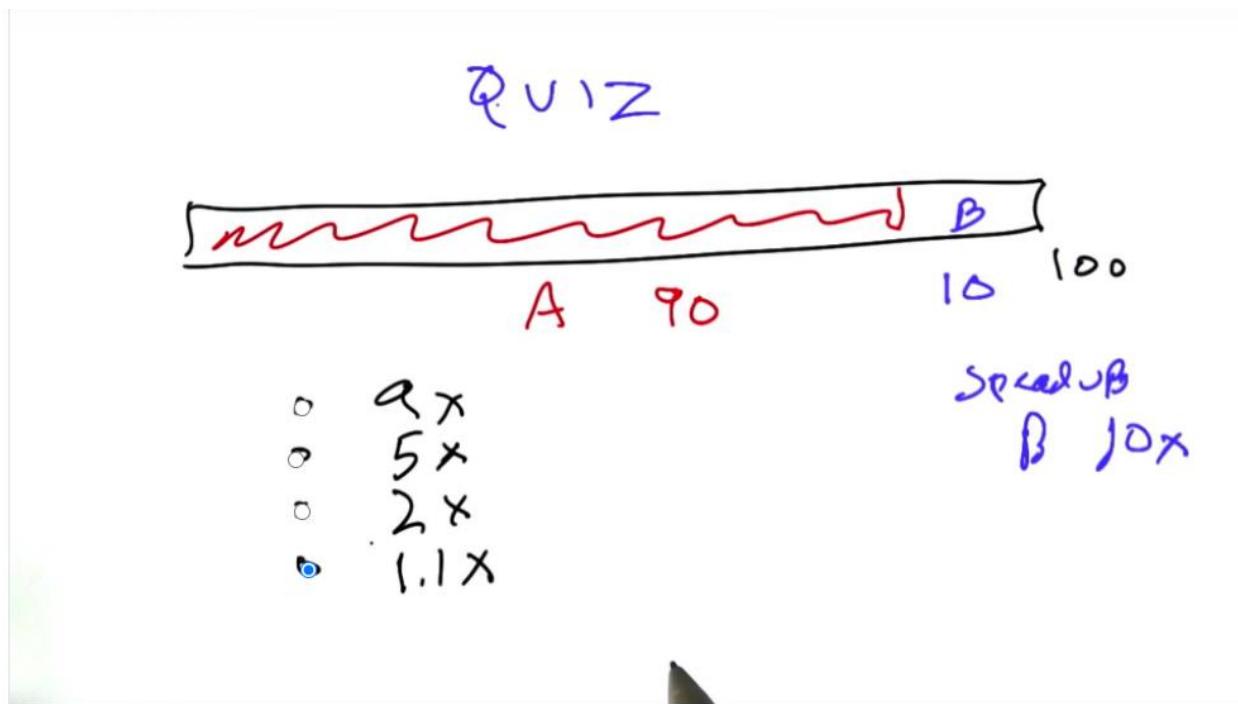
```
def fill_in(formula):
    "Generate all possible fillings-in of letters in formula with digits."
    letters = ''.join(set(re.findall('[A-Z]', formula))) #should be a string
    for digits in itertools.permutations('1234567890', len(letters)):
        table = string.maketrans(letters, ''.join(digits))
        yield formula.translate(table)
```

## Find All Values

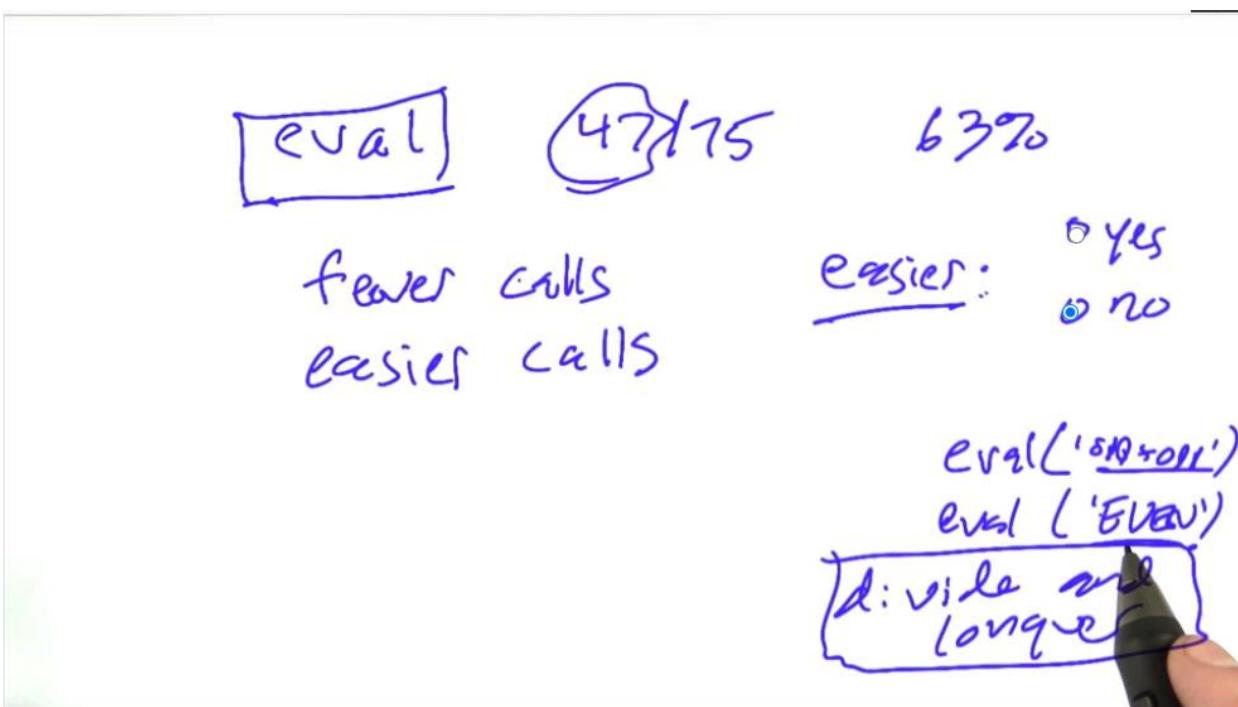
all values

- o fill-in  $\rightarrow$  list
- o "return" to "print" in solve
- o change valid
- o add new function

## Increasing Speed



## Rethinking Eval



## Making Fewer Calls

eval

47/75

63%

fewer calls  
~~easier~~ calls

easier:  
① Yes  
② No

- ① one big formula
- ② fill in one digit (zero)
- ③ eval formats once as addition with priorities

eval('5\*10+100')

eval('EVAL')

divide and conquer

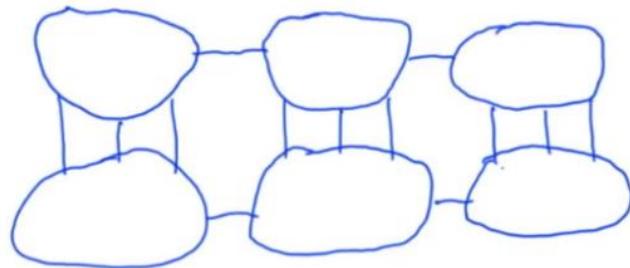
## Compile Word

```
def compile_word(word):  
    """Compile a word of uppercase letters as numeric digits.  
    E.g., compile_word('YOU') => '(1*U+10*O+100*Y)  
    Non-uppercase words unchanged: compile_word('+') => '+'  
    if word.isupper():  
        terms = [('%s%s' % (10**i, d))  
                 for (i, d) in enumerate(word[::-1])]  
        return '(' + ' + '.join(terms) + ')'  
    else:  
        return word
```

## LECCIÓN 12

### Save the Turtles

#### Save The Turtles



14

cut the bridges so plastic stays in one piece but with no loops (turtle traps).

### Strength of Connections

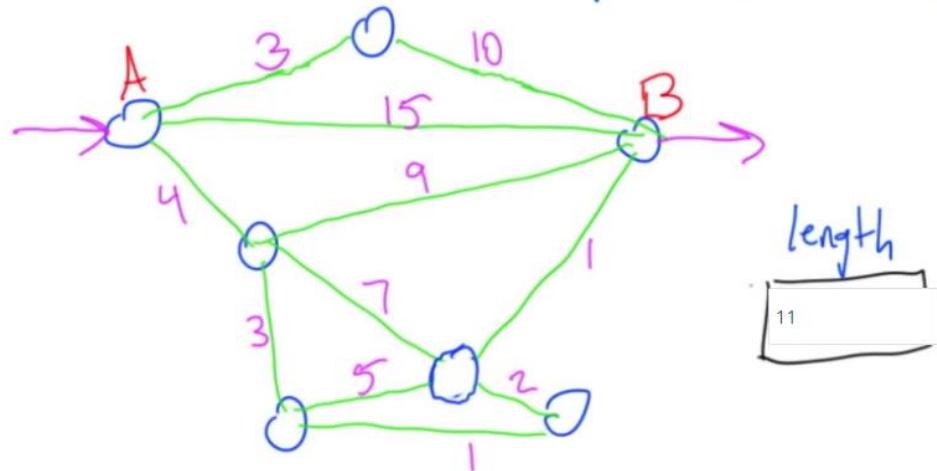
## Compute Connection Strength

Write a program to read the Marvel graph and put a strength value on each link. Which link has the highest strength value?

- HUMAN TORCH/JOHNNY S - THING/BENJAMIN J. GR
- INVISIBLE WOMAN/SUE - MR. FANTASTIC/REED R
- SPIDER-MAN/PETER PARKER - WATSON-PARKER, MARY
- CAPTAIN AMERICA - IRON MAN/TONY STARK

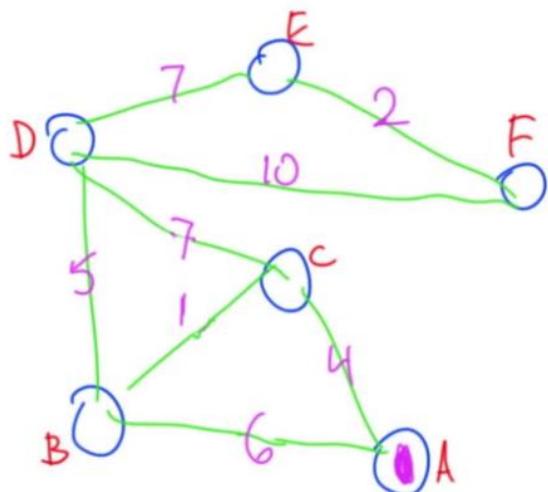
Find the Shortest Path

Find Shortest Path  
length is sum of weights



Simulate this Algorithm

Q112



Q112 A

Simulate this algorithm -  
what is the first number associated with node E?

17

What is Missing?

# Running Time of Dijkstra $n, m$

For each node :

- find the shortest dist-so-far
- remove it
- check each of its neighbors

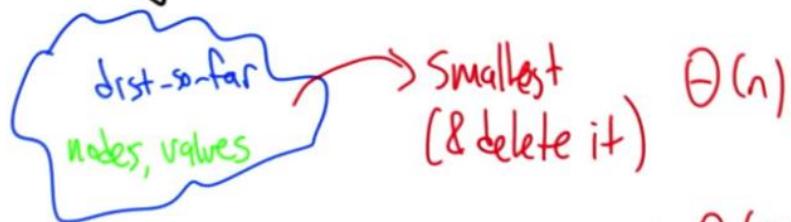
• depends on how  
"find shortest"  
is implemented

• depends on the  
degree

• depends on graph  
is dense or sparse

## Implementing Shortest Distance

### Implementing shortest-dist-node(dist-so-far)



loop through list -  $\Theta(n)$

•  $\Theta(n+m)$

•  $\Theta(nm)$

Running time of dijkstra  
with this implementation?

•  $\Theta(n^2)$

•  $\Theta(n^2+nm)$

## Using Heaps

### Running Time of Dijkstra $n, m$ With Heaps!

For each node

- find the shortest dist-so-far
- remove it
- check each of its neighbors  
possibly reducing distance

- $\Theta(n^3)$
- $\Theta(n^2)$
- $\Theta(n \log n)$
- $\Theta(m \log n)$
- $\Theta(nm \log n)$

## Randomizing Clustering Coefficient

```
9 def expected_C(G, v):  
0     # G[v].keys() is the set of neighbors of v  
1     neighbors = G[v].keys()  
2     degree = len(neighbors)  
3     # x in G[w][x] if x and w are connected in the graph (C[w,x])  
4     return clustering_coefficient(G, v)|  
5
```

## LECCIÓN 13

### Implementing Dijkstra with Heaps

```
9
10 import heapq
11
12 def dijkstra(HG, v):
13     dist_so_far = {v: 0}
14     final_dist = {}
15     heap = [(0, v)]
16     while dist_so_far:
17         (w, k) = heapq.heappop(heap)
18         if k in final_dist or (k in dist_so_far and w > dist_so_far[k]):
19             continue
20         else:
21             del dist_so_far[k]
22             final_dist[k] = w
23         for neighbor in [nb for nb in HG[k] if nb not in final_dist]:
24             nw = final_dist[k]+ HG[k][neighbor]
25             if neighbor not in dist_so_far or nw < dist_so_far[neighbor]:
26                 dist_so_far[neighbor] = nw
27                 heapq.heappush(heap, (nw, neighbor))
28     return final_dist
29
```

### Weighted Marvel Graph

## Different Paths

Following the instructions below, and then fill in the box with the number of paths that you found.

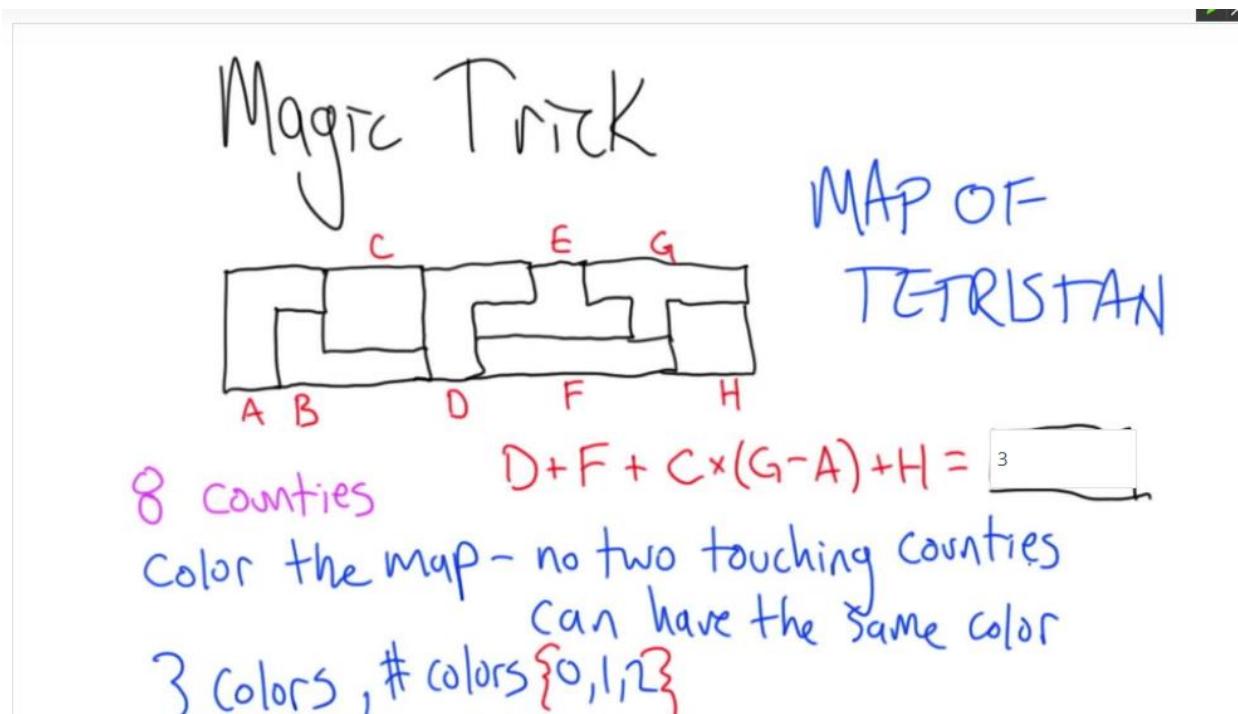
Number of paths:

## Least Obscure Path

```
121     return final_dist
122
123 def solve(graph, actor_0, actor_1):
124
125     return amended_dijkstra(graph, actor_0)[actor_1]
126
127 if __name__ == '__main__':
128     total_graph = read_graph("imdb-1.tsv")
129     movie_obscurity = read_obscurity("imdb-weights.tsv")
130
131     HG = make_hop_graph(total_graph, movie_obscurity)
132
133     print "answer = {"
134     for ch1, ch2 in answer:
135         #ch1, ch2 = t[0], t[1]
136         #routes = amended_dijkstra(HG, ch1)
137         answer[ch1, ch2] = solve(HG, ch1, ch2)
138
139         print '\t(\\"' + ch1 + '\\", \\"' + ch2 + '\"):' , answer[ch1, ch2], ","
140     print "}"|
```

## LECCIÓN 14

### Tetristan



## Degrees of Hardness

Quiz: What's a good way to find an upper bound on a problem's hardness?

- Ⓐ Devise an algorithm to solve it. Run it on a bunch of inputs. The shape of the graph is the bound.
- Ⓑ Devise an algorithm to solve it. Run on all possible inputs - running time is the upper bound.
- Ⓒ Devise an algorithm to solve it. Analyze it. Big O/ $\Theta$  is UB.
- Ⓓ No idea - is it studied in some other class?

## Lower Bound on Complexity

Finding the max of a list ( $n$ )

Upper bound:  $\Theta(n)$

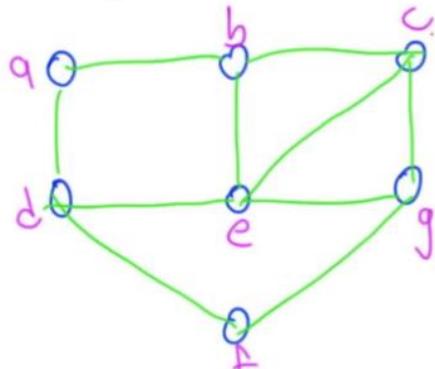
lower bound?

- Ⓐ  $\Theta(1)$ : need to give the answer. Maybe guessable...
- Ⓑ  $\Theta(n)$ : must at least look at all  $n$  items (or could miss the max)
- Ⓒ  $\Theta(\log n)$ : like a tournament, might be able to eliminate half each time.

## Longest Simple Path

- 1
- 2
- 3
- 4
- 5
- 6
- 7

### Long & Simple Path



Simple Path:  
no repeated nodes

a - g

any shortest path - simple  
longest simple path - n

Check box K if there is  
a simple path of K nodes from a to g

## Reduction: Long and Simple Path

```
def long_and_simple_path(G,u,v,l):
    """
    G: Graph
    u: starting node
    v: ending node
    l: minimum length of path
    """
    if not long_and_simple_decision(G,u,v,l):
        return False
    for node1 in G:
        neighbors = G[node1].keys()
        for node2 in neighbors:
            G = break_link(G, node1, node2)
            if not long_and_simple_decision(G,u,v,l):
                G = make_link(G, node1, node2)
    path = [u]
    node = u
    next_node = (G[node].keys())[0]
    while next_node != v:
        path.append(next_node)
        next_next_0 = (G[next_node].keys())[0]
        next_next_1 = (G[next_node].keys())[1]
        if next_next_0 == node:
            node, next_node = next_node, next_next_1
        else:
            node, next_node = next_node, next_next_0
    return path + [v]
```

## Accepting Certificate

$P \subseteq NP?$

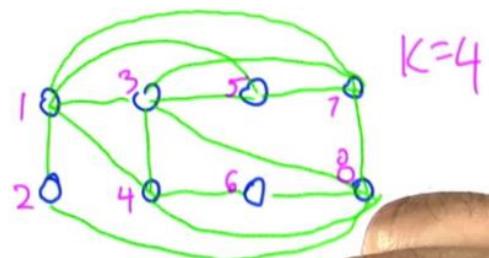
- Ⓐ If I knew that, I'd win a Fields Medal!
- Ⓑ No, if it is decidable in Polynomial time, no certificate is needed.
- Ⓒ Yes, if it can be decided in Polynomial time, no certificate is needed.

## Clique Problem

### Clique Problem

Given graph  $G$  and number  $k$ , is there a clique of size  $k$  in  $G$ ?

16



P=NP?

What if P=NP?

- Some cryptographic protocols based on problems like factoring could be cracked.
- A lot of CS theoreticians will be out of work.
- Computers will be smarter than people

## Find the Strangers

$s$ -independent set Problem Find the strangers!

Given graph  $H$  and number  $s$  is there a set of nodes of size  $s$  in  $H$  such that no two nodes in the set are connected in  $H$ ? (they are independent of each other)

Reduce to  $K$ -clique: Show how a poly time solution to  $K$ -clique solves

- $s$ -independent can be solved by guessing a set of nodes and it seeing if no pair is connected. Check  $S^2$  edges, so it is poly time.
- Run  $s$ -clique algorithm on  $H$ . Return the opposite of the answer.
- Let  $G$  be the complement of  $H$ . Run  $s$ -clique on  $G$ , return opposite.
- Let  $G$  be the complement of  $H$ . Run  $s$ -clique on  $G$ , return answer.

## NP-Completeness

Graph Partitioning is NP complete (GP)

Even if you don't know what it is,  
check all you know

- GP is NP-hard - a Poly time solution solves everything in NP
- clique can be reduced to it
- GP is in NP - an exponential-time solution exists
- It can be reduced to SAT.

### Solving 3-Colorability

```
def verify(G, cert, k):
    if len(cert) != len(G):
        return False
    for node in cert:
        if cert[node] not in range(k):
            return False
        for neighbor in G[node]:
            if cert[node] == cert[neighbor]:
                return False
    return True
```

### Generating a Formula

Formula Size

K Colors	3
n Nodes	8
M edges	20

How many clauses in the formula?  
(added together)



## 4-Colorability

### 4 colorability Quiz

- It's in NP because we can quickly verify a certificate that lists the color assignments
- It's not necessarily in NP because the number of colors is bigger
- It's not necessarily NP hard because having 4 colors to work with makes things easier
- It's NP hard because a solution to 4 colorability solves 3 colorability - add a node & connect it to all
- Since any graph that is 3 colorable is 4 colorable - equivalent

## LECCIÓN 15

### Programming a Reduction

```
by
70 # This function should use the k_clique_decision function
71 # to solve the independent set decision problem
72 def independent_set_decision(H, s):
73     return k_clique_decision(reform_graph(H), s)
74
75 if __name__ == '__main__':
76     flights = [(1,2),(1,3),(2,3),(2,6),(2,4),(2,5),(3,6),(4,5)]
77     G = {}
78 for (x,y) in flights:
79     make_link(G,x,y)
80
81 for node in G:
82     print node, "--",
83     for x in G[node]:
84         print x,
85     print
86
87 print
88 print "independent set:"
89 for i in range(1, len(flights)):
90     print i, "-", independent_set_decision(G, i)
```

## Reduction: k-Clique to Decision

```
1 def k_clique(G, k):
2     if not k_clique_decision(G, k):
3         return False
4     if len(G) is k:
5         return G.keys()
6
7     for edge in get_all_edges(G):
8         break_link(G, edge[0], edge[1])
9         if k_clique_decision(G, k):
10            return k_clique(G, k)
11            make_link(G, edge[0], edge[1])
12
13 if __name__ == '__main__':
14     edges = [(1,2),(1,3),(1,4),(2,4)]
15     G = {}
16     for (x,y) in edges:
17         make_link(G,x,y)
18
19     for x in G:
20         print x, G[x].keys()
21     print
22
23 k = 3
24 print "k =", k
25 print k_clique(G, k)
```

## Polynomial vs. Exponential

## Polynomial vs. Exponential

Theoreticians would say that an algorithm with a running time of  $n^{100}$  is efficient, but one with a running time of  $1.1^n$  is not. Assuming these running times are exact, what's the smallest  $n$  for which the efficient algorithm is faster?

9623

### From Clauses to Colors

## From Clauses to Colors

In the reduction from 3-SAT to 3-COLORABILITY, we talked about a way of converting a 3-SAT problem with  $x$  variables and  $y$  clauses into a graph with  $n$  nodes and  $m$  edges. Give a formula for  $n$  and  $m$ . (Fill in the boxes to complete the equation. See the example given below.)

$$\begin{array}{rcl} n & = & \boxed{0}x + \boxed{0}y + \boxed{0} \\ m & = & \boxed{0}x + \boxed{0}y + \boxed{0} \end{array}$$

(ex.  $n = 4x + 10y + 8$ )

NP or Not NP?

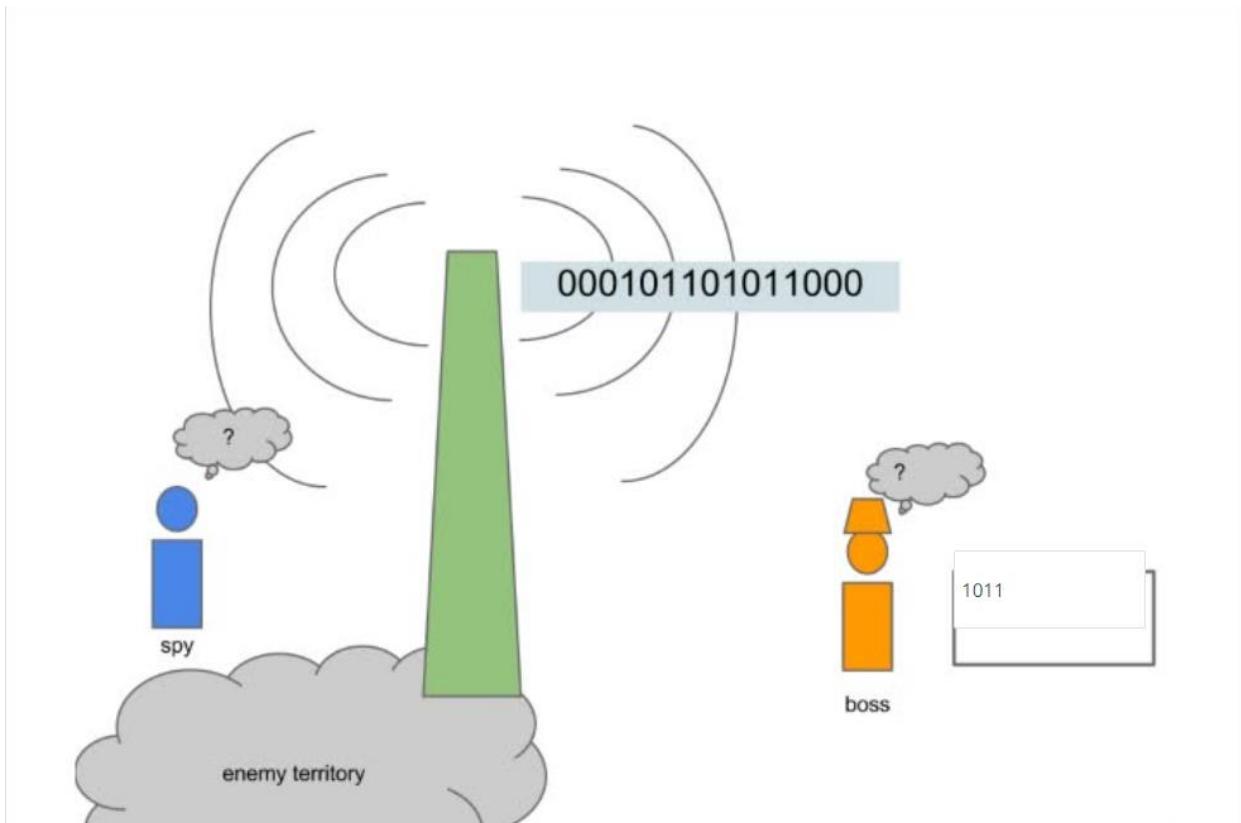
## NP or Not NP? That is the Question

Select all the problems below that are in NP. Hint: Think about whether or not each one has a short accepting certificate.

- Connectivity:** Is there a path from  $x$  to  $y$  in  $G$ ?
- Short path:** Is there a path from  $x$  to  $y$  in  $G$  that is no more than  $k$  steps long?
- Fewest colors:** Is  $k$  the absolute minimum number of colors with which  $G$  can be colored?
- Near Clique:** Is there a group of  $k$  nodes in  $G$  that has at least  $s$  pairs that are connected?
- Partitioning:** Can we group the nodes of  $G$  into two groups of size  $n/2$  so that there are no more than  $k$  edges between the two groups?
- Exact coloring count:** Are there exactly  $s$  ways to color graph  $G$  with  $k$  colors?

## LECCIÓN 16

### Spy Control



## Homophily

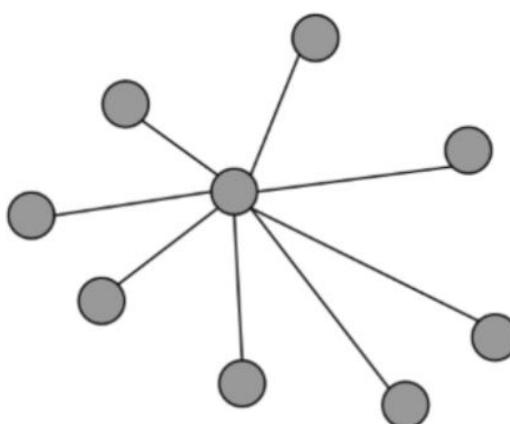
### Homophily is:

- the observation that two words tend to be pronounced similarly if they are in a social network.
- the fear that two people connected in a social network may develop a blood disorder that interferes with clotting.
- the political movement that suggests that people should be allowed to marry people of the same sex.
- the idea that nodes that are connected in a social network are also more likely to share other characteristics.
- the term used to refer to individuals who reside in the largest city in Pennsylvania.

## Networks

How many hubs do we need to list in each node of a star network to satisfy the property that we can compute shortest path lengths efficiently?

1
---



## LECCIÓN 18

### Top Two

```
11 def top_two(graph, start, topX=2):
12     heap = [((0, [start]), start)]
13     best = defaultdict(list)
14     best[start] = [(0, [start])]
15     while heap:
16         ((cost, path), node) = heapq.heappop(heap)
17         for (other, weight) in graph[node].items():
18             if other in path:
19                 continue
20
21             new = (cost + weight, path + [other])
22             best[other] = sorted(best[other] + [new])[:topX]
23             if new in best[other]:
24                 heapq.heappush(heap, (new, other))
25
26     return best
27
```