Week 6

Introduction to Algorithms

What are algorithms?

algorithm
/ˈalgərɪð(ə)m/

noun
noun: **algorithm**; plural noun: **algorithms**

a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
"a basic **algorithm** for division"

Origin

late 17th century (denoting the Arabic or decimal notation of numbers): variant (influenced by Greek *arithmos* 'number') of Middle English *algorism*, via Old French from medieval Latin *algorismus* . The Arabic source, *al-Ḵwārizmī* 'the man of Ḵwārizm' (now Khiva), was a name given to the 9th-century mathematician Abū Jaʿfar Muhammad ibn Mūsa, author of widely translated works on algebra and arithmetic.

https://www.dictionary.com/browse/algorithm

British Dictionary definitions for algorithm

algorithm

noun

1. a logical arithmetical or computational procedure that if correctly applied ensures the solution of a problem: Compare heuristic
2. **logic maths** a recursive procedure whereby an infinite sequence of terms can be generated

French name: **algorism**

Derived Forms

algorithmic, adjective
algorithmically, adverb

Word Origin

C17: changed from algorism, through influence of Greek *arithmos* number

algorithm in Science

algorithm

[ălˈgə-rĭðʹəm]

1. A finite set of unambiguous instructions performed in a prescribed sequence to achieve a goal, especially a mathematical rule or procedure used to compute a desired result. Algorithms are the basis for most computer programming.

Combining definitions…

Algorithm (noun):

A *finite* sequence of *unambiguous* steps that *solves* a specific problem.

Combining definitions…

```
while True:
  print("foo")
```

Algorithm (noun):

A *finite* sequence of *unambiguous* steps that *solves* a specific problem.

Combining definitions…

~~while True:~~
~~ print("foo")~~

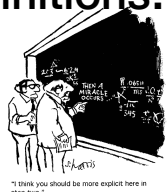
Algorithm (noun):

We want our algorithms to **terminate!**

A *finite* sequence of *unambiguous* steps that *solves* a specific problem.

Combining definitions…

Algorithm (noun):



"I think you should be more explicit here in step two."

A *finite* sequence of *unambiguous* steps that *solves* a specific problem.

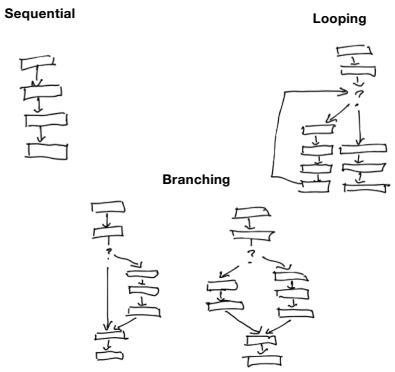
Cartoon from http://www.sciencecartoonsplus.com/gallery/math/math07.gif
Copied under fair use, see https://rationalwiki.org/wiki/File:Math07.gif

Combining definitions…

Algorithm (noun):

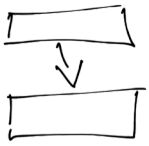
A *finite* sequence of *unambiguous* steps that *solves* a specific problem.

Building-blocks in Programming and in Algorithms



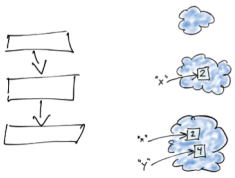
Sequential execution

```
print("Hello")
print("World")
```



Variables (program state)

```
x = 2
y = 2 * x
print(x, y)
```



Branching

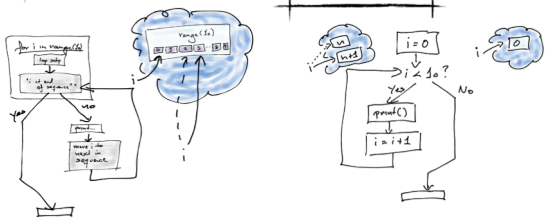
```
if x > 0:
  print("x is positive")
if x < 0:
  print("x is negative")
```

```
if x > 0:
  print("x is positive")
elif x < 0:
  print("x is negative")
else:
  print("x is zero")
```

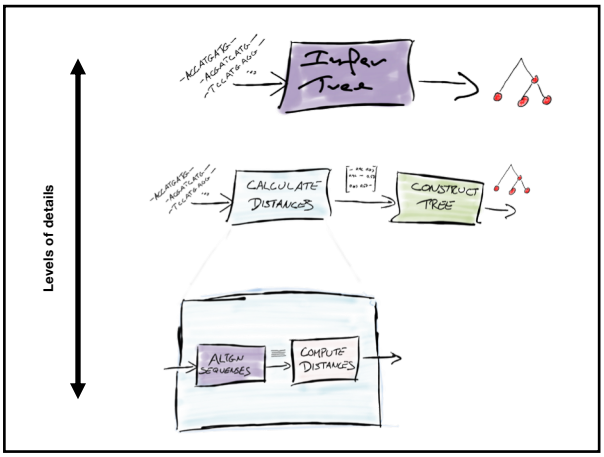
Looping

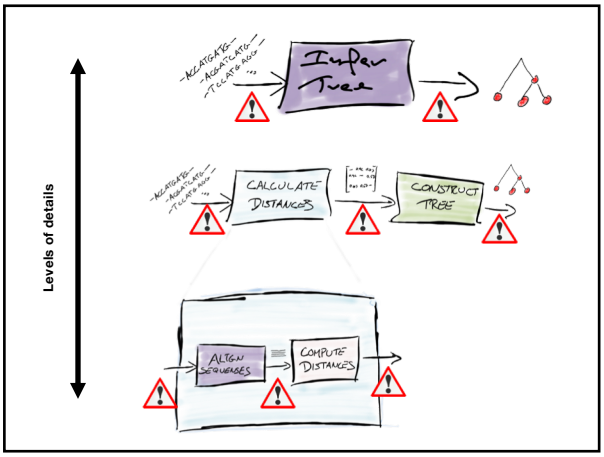
```
for i in range(10):
  print("i =", i)
```

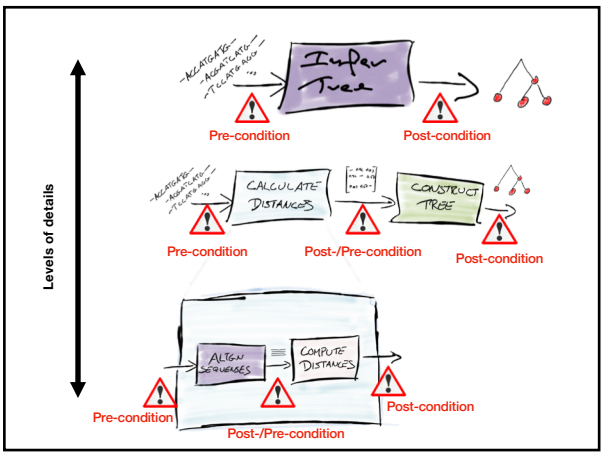
```
i = 0
while i < 10:
  print("i =", i)
  i = i + 1
```

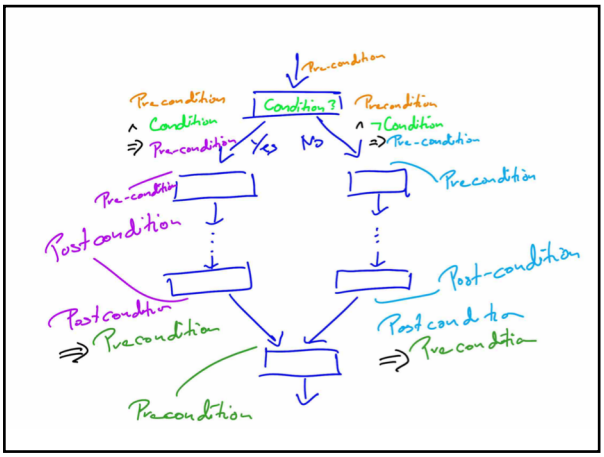


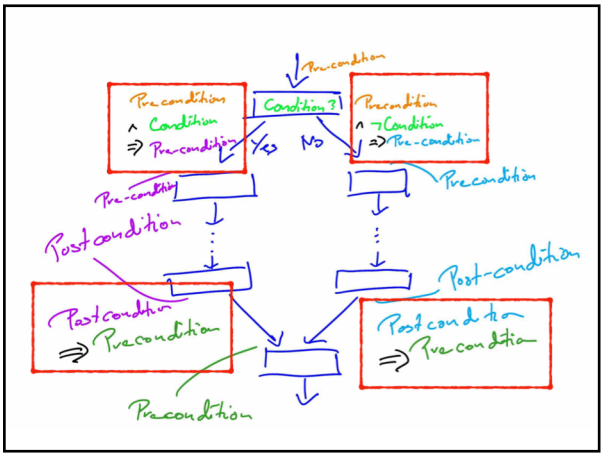
Designing algorithms: Breaking down problems to smaller parts

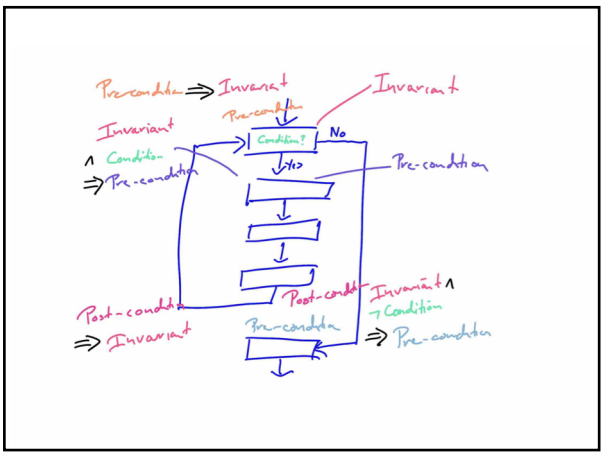


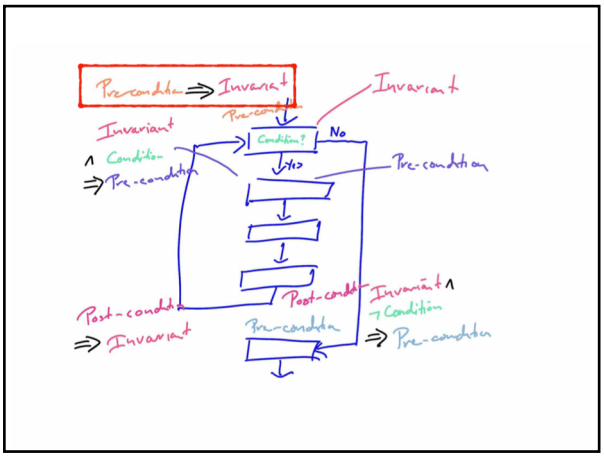

Levels of details
Infer Tree
CALCULATE DISTANCES
CONSTRUCT TREE
ALIGN SEQUENCES
COMPUTE DISTANCES

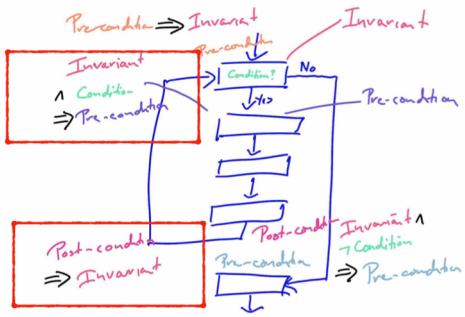

Levels of details
Infer Tree
Pre-condition
Post-condition
CALCULATE DISTANCES
CONSTRUCT TREE
Pre-condition
Post-/Pre-condition
Post-condition
ALIGN SEQUENCES
COMPUTE DISTANCES
Pre-condition
Post-condition
Post-/Pre-condition

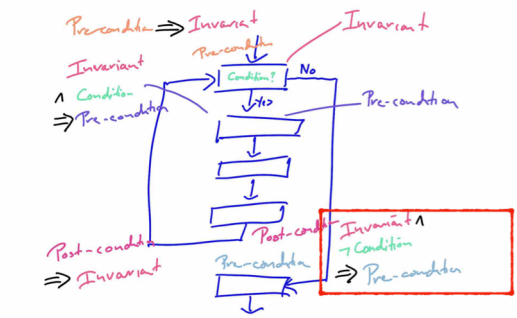

Pre-condition
Pre-conditions
Condition
Pre-condition
Condition ?
Pre-condition
¬Condition
Pre-condition
Yes
No
Pre-condition
Pre-condition
Post-condition
Post-condition
Post-condition
Post-condition
Pre-condition
Post-condition
Pre-condition
Pre-condition


Pre-condition
Pre-conditions
∧ Condition
⇒ Pre-condition
Condition?
Yes
No
Precondition
∧ ¬Condition
⇒ Pre-condition
Pre-condition
Precondition
Post-condition
Post-condition
Post-condition
Post condition
⇒ Precondition
Post condition
⇒ Precondition
Precondition

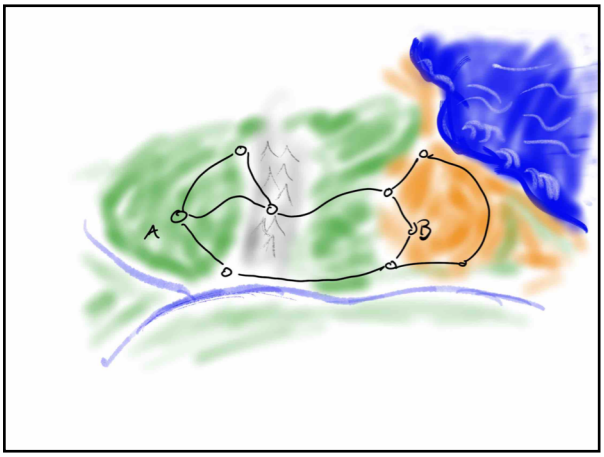

Pre-condition ⇒ Invariant
Invariant
Pre-condition
Invariant
∧ Condition
⇒ Pre-condition
Condition?
No
Yes
Pre-condition
Post-condition
⇒ Invariant
Post-condition
Invariant ∧
¬ Condition
Pre-condition
⇒ Pre-condition

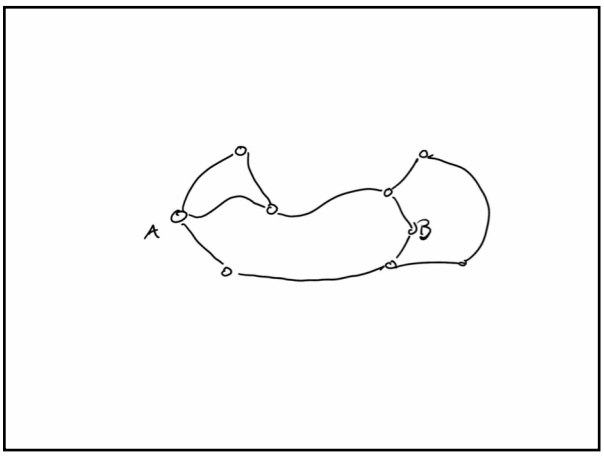

Pre-condition ⇒ Invariant
Invariant
Pre-condition
Invariant
∧ Condition
⇒ Pre-condition
Condition?
No
Yes
Pre-condition
Post-condition
⇒ Invariant
Post-condition
Invariant ∧
¬ Condition
Pre-condition
⇒ Pre-condition

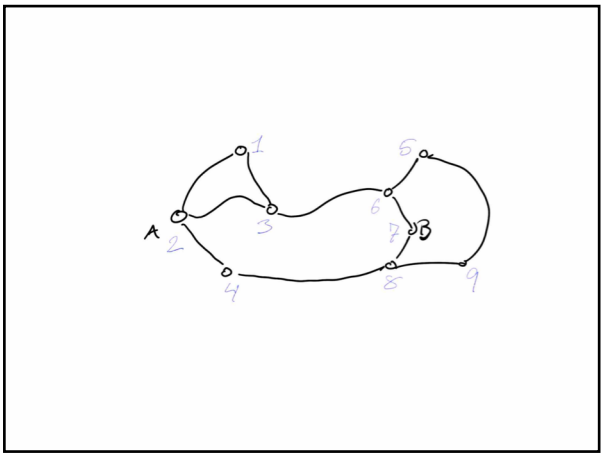


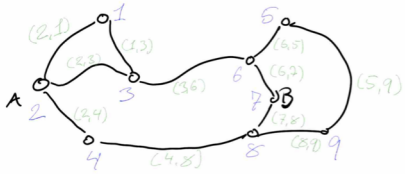


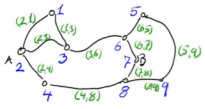
Example: are cities A and B connected?


A
B


A
B

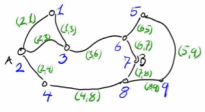

1
5
A
2
3
6
7
B
4
8
9





	1	0
A	2	0
	3	0
	4	0
	5	0
	6	0
B	7	0
	8	0
	9	0

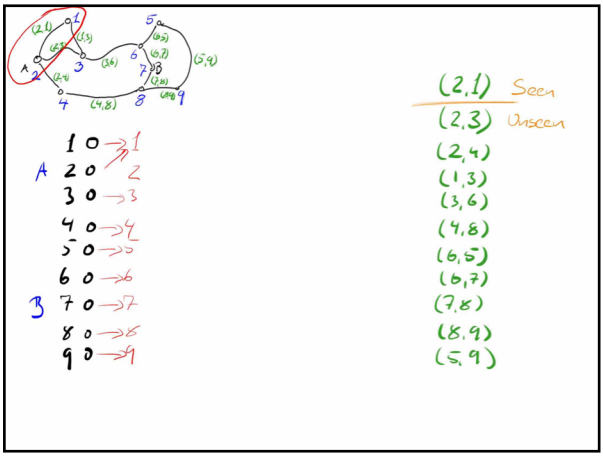
(2,1)
(2,3)
(2,4)
(1,3)
(3,6)
(4,8)
(6,5)
(6,7)
(7,8)
(8,9)
(5,9)



	1	0 → 1
A	2	0 → 2
	3	0 → 3
	4	0 → 4
	5	0 → 5
	6	0 → 6
B	7	0 → 7
	8	0 → 8
	9	0 → 9

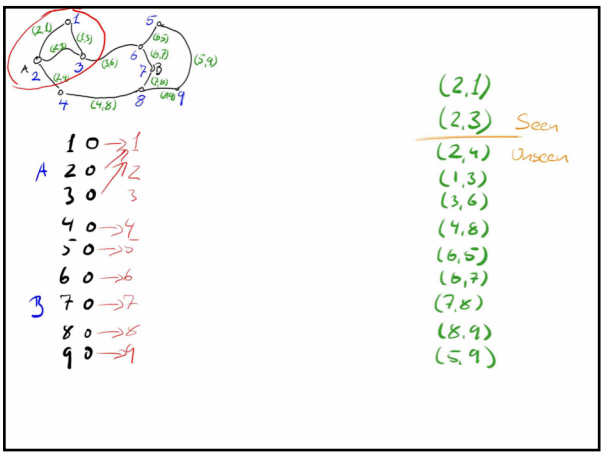
Seen

(2,1) Unseen
(2,3)
(2,4)
(1,3)
(3,6)
(4,8)
(6,5)
(6,7)
(7,8)
(8,9)
(5,9)



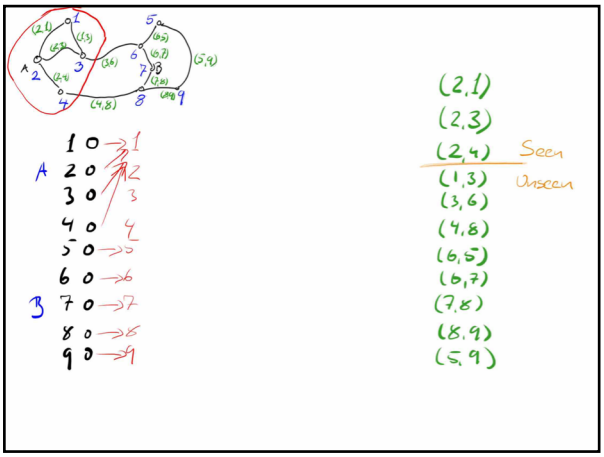
1 0 → 1
A 2 0 → 2
3 0 → 3
4 0 → 4
5 0 → 5
6 0 → 6
B 7 0 → 7
8 0 → 8
9 0 → 9

(2,1) Seen
(2,3) Unseen
(2,4)
(1,3)
(3,6)
(4,8)
(6,5)
(6,7)
(7,8)
(8,9)
(5,9)



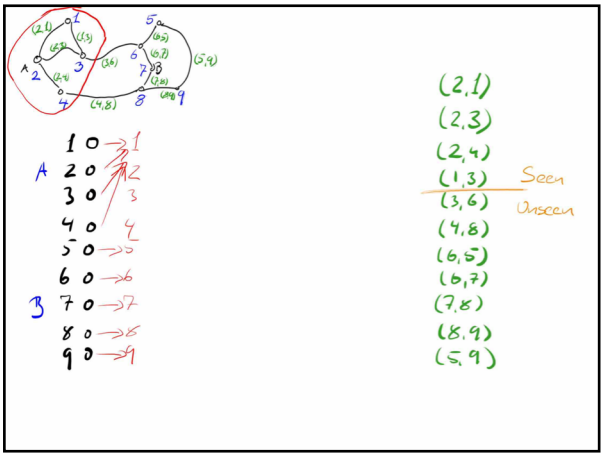
1 0 → 1
A 2 0 → 2
3 0 → 3
4 0 → 4
5 0 → 5
6 0 → 6
B 7 0 → 7
8 0 → 8
9 0 → 9

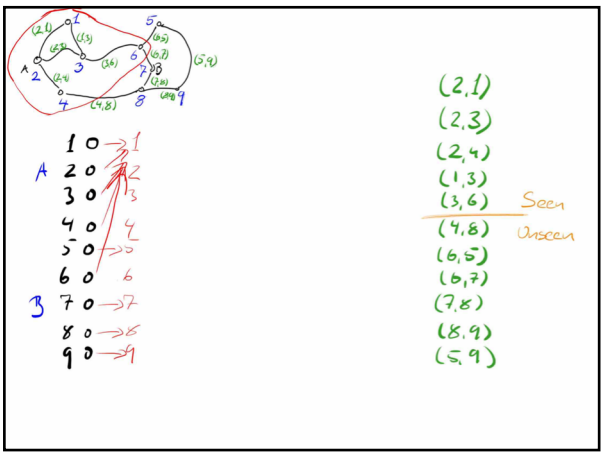
(2,1)
(2,3) Seen
(2,4) Unseen
(1,3)
(3,6)
(4,8)
(6,5)
(6,7)
(7,8)
(8,9)
(5,9)

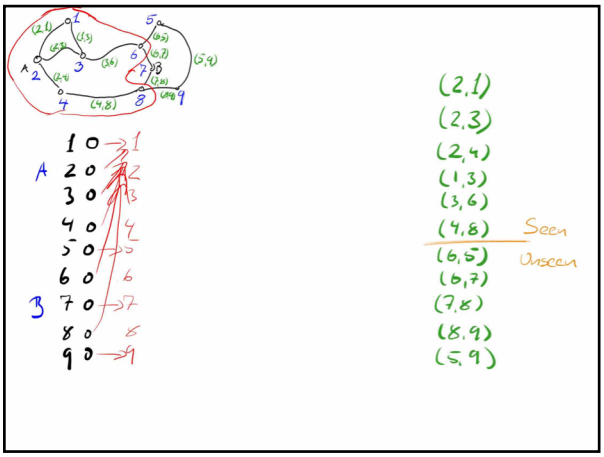


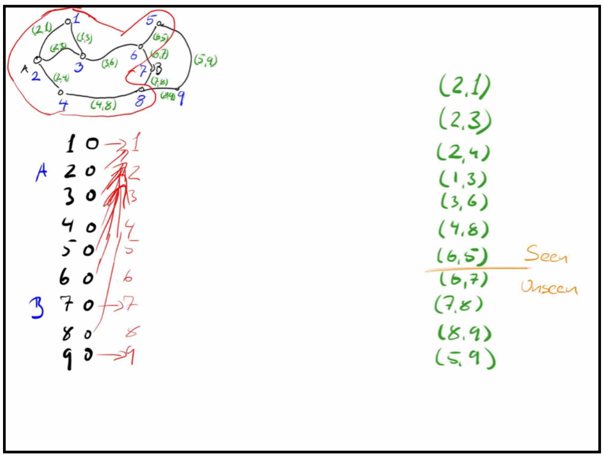
1 0 → 1
A 2 0 → 2
3 0 → 3
4 0 → 4
5 0 → 5
6 0 → 6
B 7 0 → 7
8 0 → 8
9 0 → 9

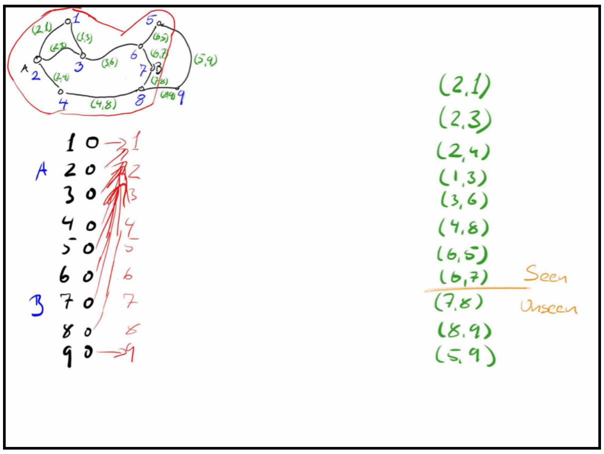
(2,1)
(2,3)
(2,4) Seen
(1,3) Unseen
(3,6)
(4,8)
(6,5)
(6,7)
(7,8)
(8,9)
(5,9)

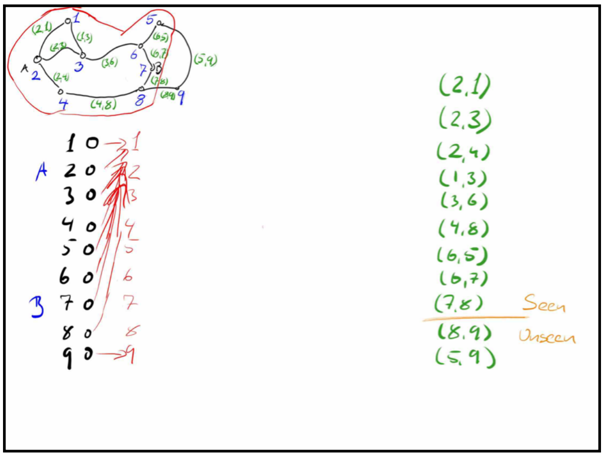

(2,1)
(1,3)
(4,2)
(3,6)
(4,8)
(6,5)
(6,7)
(7,8)
(8,9)
(5,9)
A
B
1 0 → 1
A 2 0 → 2
3 0 → 3
4 0 → 4
5 0 → 5
6 0 → 6
B 7 0 → 7
8 0 → 8
9 0 → 9
(2,1)
(2,3)
(2,4)
(1,3) Seen
(3,6) Unseen
(4,8)
(6,5)
(6,7)
(7,8)
(8,9)
(5,9)

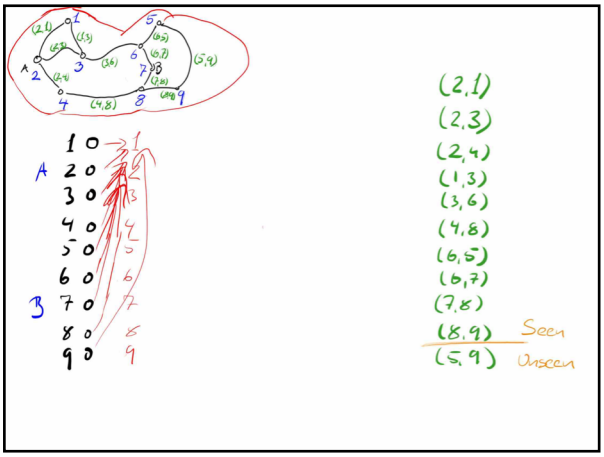

(2,1)
(1,3)
(4,2)
(3,6)
(4,8)
(6,5)
(6,7)
(7,8)
(8,9)
(5,9)
A
B
1 0 → 1
A 2 0 → 2
3 0 → 3
4 0 → 4
5 0 → 5
6 0 → 6
B 7 0 → 7
8 0 → 8
9 0 → 9
(2,1)
(2,3)
(2,4)
(1,3)
(3,6) Seen
(4,8) Unseen
(6,5)
(6,7)
(7,8)
(8,9)
(5,9)

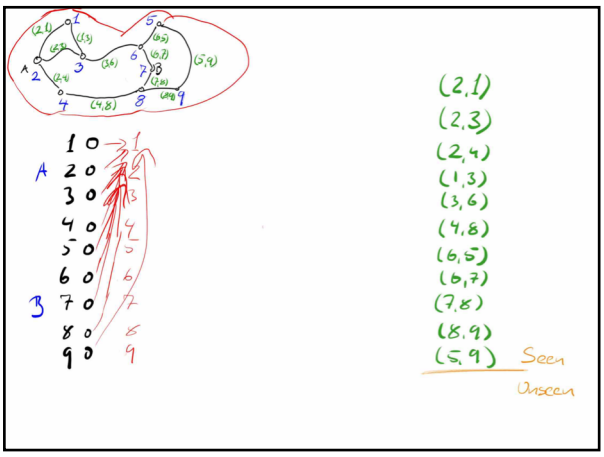

(2,1)
(1,3)
(4,2)
(3,6)
(4,8)
(6,5)
(6,7)
(7,8)
(8,9)
(5,9)
A
B
1 0 → 1
A 2 0 → 2
3 0 → 3
4 0 → 4
5 0 → 5
6 0 → 6
B 7 0 → 7
8 0 → 8
9 0 → 9
(2,1)
(2,3)
(2,4)
(1,3)
(3,6)
(4,8) Seen
(6,5) Unseen
(6,7)
(7,8)
(8,9)
(5,9)


1 0
A 2 0
3 0
4 0
5 0
6 0
B 7 0
8 0
9 0
(2,1)
(2,3)
(2,4)
(1,3)
(3,6)
(4,8)
(6,5)
Seen
Unseen
(6,7)
(7,8)
(8,9)
(5,9)


1 0
A 2 0
3 0
4 0
5 0
6 0
B 7 0
8 0
9 0
(2,1)
(2,3)
(2,4)
(1,3)
(3,6)
(4,8)
(6,5)
(6,7)
Seen
Unseen
(7,8)
(8,9)
(5,9)


1 0
A 2 0
3 0
4 0
5 0
6 0
B 7 0
8 0
9 0
(2,1)
(2,3)
(2,4)
(1,3)
(3,6)
(4,8)
(6,5)
(6,7)
(7,8)
Seen
Unseen
(8,9)
(5,9)



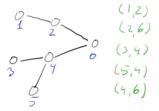


Intermezzo

- How would you represent the components in Python?
- How would you formalise the loop invariant to take into account the representation of the components?

Intermezzo

- Run the algorithm on these two graphs:

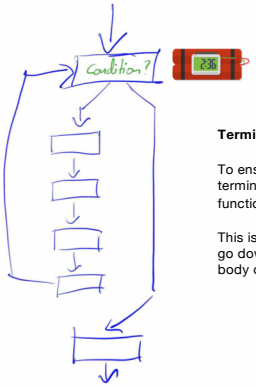


Correctness

How do we prove correctness?

- We prove that all pre- and post-conditions are satisfied through the steps in the algorithm.
- We prove that the post-condition of the last step implies that the overall problem is solved.
- *Correctness is just a special case of post-conditions*

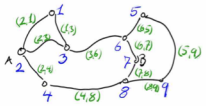
Termination



Termination functions:

To ensure that our algorithm terminations, we add a "termination" function to each loop.

This is a count-down that should go down each time we execute the body of the loop.



1 0 → 1
A 2 0 → 2
3 0 → 3
4 0 → 4
5 0 → 5
6 0 → 6
B 7 0 → 7
8 0 → 8
9 0 → 9

Seen / Unseen

(2,1)
(2,3)
(2,4)
(1,3)
(3,6)
(4,8)
(6,5)
(6,7)
(7,8)
(8,9)
(5,9)

Get the binary representation of a number *n*

```
reverse_bits = []
while n > 0:
    reverse_bits.append(n % 2)
    n //= 2
print(reverse_bits[::-1])
```

Termination function:
$t(n) = n$

Thats it!

Now it is time to do the exercises to test that you now know how to construct algorithms