Abstract Data Structures

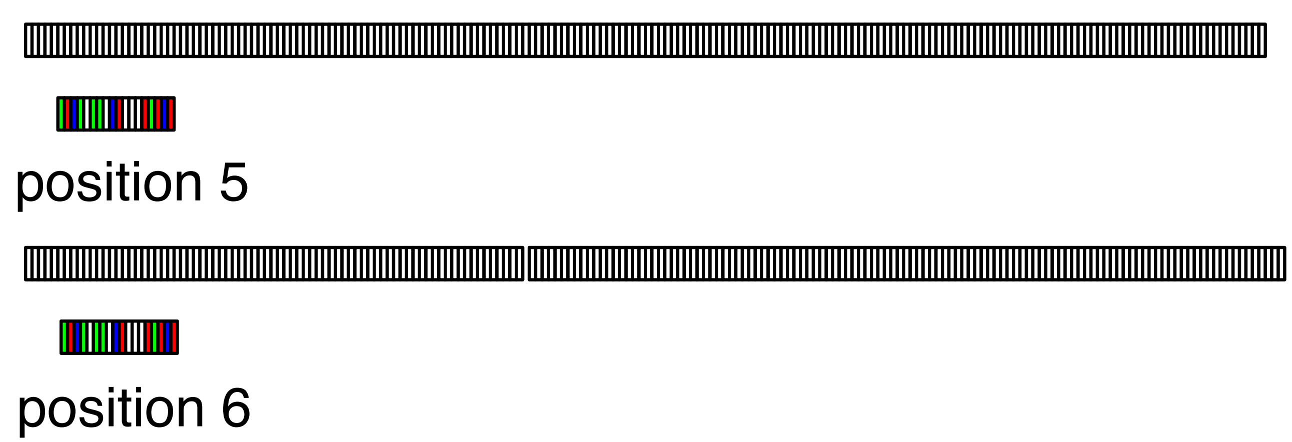
Thomas Schwarz, SJ

Algorithms

- Recipe for a computation
  - Can have a large effect on the resources a computation takes

Algorithms

- Example:
  - Finding a string of length 100 in a human genome
    - Naive method: Compare string letter for letter against all positions in about 3 billion letters



    - 300,000,000,000 comparisons

Algorithms

- Example:
  - Finding a string of length 100 in a human genome
    - Slightly better by breaking off comparisons when we know we cannot have a match
      - On average $(3/4 + 2/4 + 3/16 + 4/32 + \ldots) \times 3000000000$ comparisons
  - Can be done with **less than** 3000000000 comparisons!

Algorithms

- Two criteria
  - Correctness
    - Formal methods, testing
  - Resource consumption
    - Speed, memory use

Example: Collatz Conjecture

- Collatz sequence (a.k.a. hailstorm sequence)
  - Take a number
    - If the number is even, divide the number by 2
    - If the number is odd, multiply by 3 and add 1
  - Repeat to get the Collatz sequence
    - Example:
      - 9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
- Collatz Conjecture: All Collatz sequences terminate in 1

Example: Collatz Conjecture

- Implementing the Collatz conjecture tester:
  - First, implement the Collatz step

```
def collatz_step(n):
    if n%2:
        return 3*n+1
    else:
        return n//2
```

Example: Collatz Conjecture

- We can calculate a Collatz sequence:

```
def collatz_sequence(n):
    sequence = [n]
    while True:
        n = collatz_step(n)
        sequence.append(n)
        if n == 1:
            return sequence
```

Example: Collatz Conjecture

- Assume we want to test the Collatz conjecture for all numbers smaller than 1,000,000
  - Observation 1:
    - If a Collatz sequence contains a number, it will contain the sequence for that number
      - Collatz sequence for 11:
        - 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
      - Collatz sequence for 9:
        - 9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Example: Collatz Conjecture

- Consequence:
  - If we see a number for which the Collatz conjecture holds, we do not need to continue calculating the sequence

Example: Collatz Conjecture

- Smart algorithm:
  - Remember whether a number has already appeared in a Collatz sequence
  - Whenever we encounter such a number, we can stop
  - Therefore: keep a list of all Collatz numbers and update it

Example:
Collatz Conjecture

- When could the Collatz conjecture go wrong:
  - One possibility:
    - Just go off and get larger and larger numbers
  - Alternative:
    - We repeat the same sequence over and over again

Example: Collatz Conjecture

```
def is_collatz(i):
    sequence = set([i])
    while True:
        i = collatz_step(i)
        if i==1:
            return True
        if i in sequence:
            return False
        sequence.add(i)
        if i < NN and collatz[i]:
            for x in sequence:
                if x < NN:
                    collatz[x] = True
```

Example:
Collatz Conjecture

- Then check for all numbers between 2 and 1,000,000

```
print('starting')
for i in range(2,NN):
    if collatz[i]:
        continue
    if not is_collatz(i):
        print(i)
print('finishing')
```

Data Structures

- Fundamental objects of computation
  - Built from smaller objects such as integers, floating point numbers, pixels, codes (utf-8, ASCII)
  - Using basic aggregation such as arrays, unions, fields provided by software

Abstract Data Structures

- ADT encapsulate the behavior of a data structure
- Using an interface for interaction

Abstract Data Structures

- Example:
  - Counter
    - Counters can be incremented and decremented
    - They have an integer value that can be read
    - They are initialized with a zero value
    - Behavior is abstractly defined:
      - Decrementing a counter with zero value creates an error or has no effect
        - This is a design decision
      - Incrementing a counter always increases the value by one

Abstract Data Structures

- Standard Implementation of an ADT uses a Python class
  - But not necessarily

Abstract Data Structures

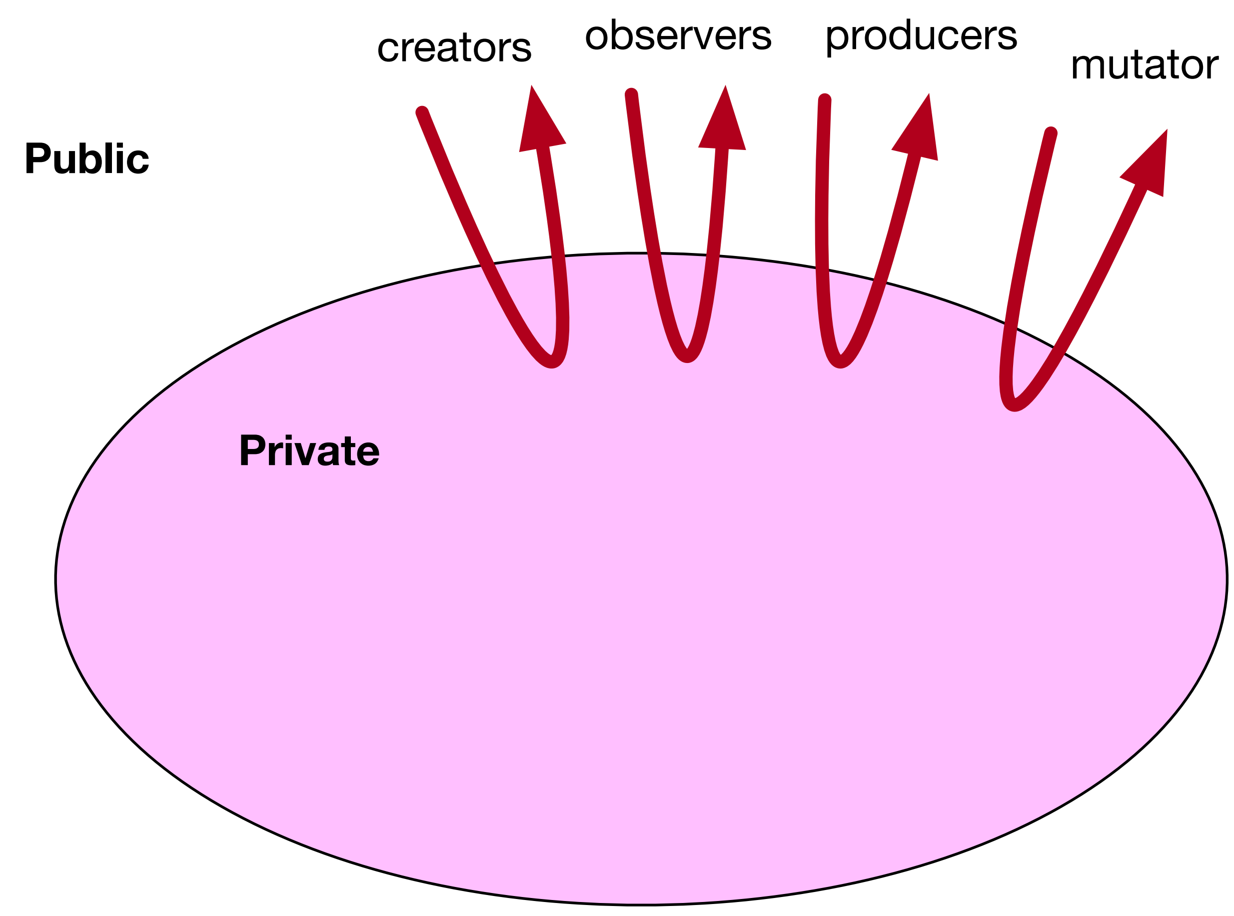
- Silly Example:
  - The Counter ADT in Python
    - Silly because we could just use an int with a couple of functions
  - Counters can be incremented and decremented
  - They have a value that can be returned
  - Null 0 is an absolute minimum

Abstract Data Structures

```
class Counter( ):
    def __init__(self):
        self.count = 0
    def add(self):
        self.count += 1
    def dec(self):
        if self.count >=1:
            self.count -= 1
    def get_count(self):
        return self.count
    def is_null(self):
        return self.count==0
```

Abstract Data Structures

- ADT provides an interface to the world
    - Creator: Create an instance
    - Observer: Get something from the ADT
    - Producer: Create new instances from old instances
    - Mutators: Change an instance of the ADT



Abstract Data Structures

- Example:
  - Strings:
    - Different implementations:
      - Pascal strings: an array of characters plus a length
      - Unix strings: an array of characters with a null symbol at the end

Abstract Data Structures

- Example:
  - Strings:
    - ADT String provides an interface that is:
      - Formal enough to reason about strings
      - Allows to re-implement the data structure

Abstract Data Structures

- Example:
  - Strings:
    - Creator: a constructor that creates a string
    - Observer: get character at second last position
    - Mutator: replace all substrings of a certain form with a different form
    - Producer: concatenate two strings