SICP

God's Programming Book

Lecture-20 Decomposition





Decomposition

Slides Adapted from cs61a of UC Berkeley



Modular Design



Separation of Concerns

A design principle: Isolate different parts of a program that address different concerns

A modular component can be developed and tested independently

Hog

Hog Game Simulator

- Game rules
- Ordering of events
- State tracking to determine the winner

Game Commentary

- Event descriptions
- State tracking to generate commentary

Player Strategies

- Decision rules
- Strategy parameters (e.g., margins & number of dice)

Ants

Ants Game Simulator

- Order of actions
- Food tracking
- Game ending conditions

Actions

 Characteristics of different ants & bees Tunnel Structure

- Entrances & exits
- Locations of insects

Set Intersection



Linear-Time Intersection of Sorted Lists

Given two sorted lists with no repeats, return the number of elements that appear in both.



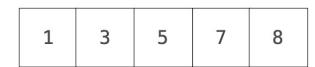


```
def fast_overlap(s, t):
    """Return the overlap between sorted S and sorted T.
    >>> fast_overlap([3, 4, 6, 7, 9, 10], [1, 3, 5, 7, 8])
    .....
    i, j, count = 0, 0, 0
   while
        if s[i] == t[j]:
            count, i, j = _{-}
        elif s[i] < t[j]:
        else:
    return count
```

Linear-Time Intersection of Sorted Lists

Given two sorted lists with no repeats, return the number of elements that appear in both.





```
def fast_overlap(s, t):
    """Return the overlap between sorted S and sorted T.
    >>> fast_overlap([3, 4, 6, 7, 9, 10], [1, 3, 5, 7, 8])
    .....
    i, j, count = 0, 0, 0
                 i < len(s) and j < len(t)
   while
        if s[i] == t[j]:
            count, i, j = count + 1, i + 1, j + 1
        elif s[i] < t[j]:
                             i = i + 1
        else:
                             j = j + 1
    return count
```

Sets



Sets

One more built-in Python container type

- Set literals are enclosed in braces
- Duplicate elements are removed on construction
- Sets have arbitrary order

```
>>> s = {'one', 'two', 'three', 'four', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
>>> 'three' in s
True
>>> len(s)
4
>>> s.union({'one', 'five'})
{'three', 'five', 'one', 'four', 'two'}
>>> s.intersection({'six', 'five', 'four', 'three'})
{'three', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
```

Thanks for Listening

