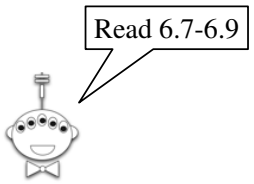
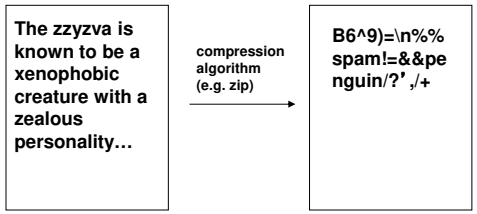
The CS 5 Herald

Goodwill Gesture Goes Awry

Claremont (AP): Seven rooms were damaged in a Harvey Mudd College dormitory Tuesday evening after a misguided attempt to cheer up sleep-deprived students. "We were approached by a group of three penguins who wanted to sing Christmas carols," explained a witness. "They promised that it would help us study." Instead, the raucous squawking of the untrained and untalented birds quickly led to a violent dispute between supporters and detractors. One student attempted to encase the singers in foam rubber, but a second set fire to the material in hopes of freeing the animals. The resulting explosion unleashed a conflagration that spread to North Dorm, where there was extensive damage. However, losses in North were estimated at only $35.47, due to the advanced age of the furniture



Data Compression



TEXT FILE
zzyzva.txt

58,254 bytes

TEXT FILE
zzyzva.txt.Z

23,124 bytes

Data Compression!

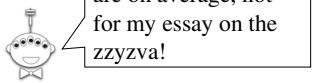
The zzyzva is known to be a xenophobic creature with a zealous personality…

TEXT FILE

Letter	ord(Letter)	Binary
T	84	01010100
h	104	01101000
e	101	01100101
z	122	01111010

- ' '– 1226754 19.04%
- E – 655257 10.17%
- T – 474521 7.37%
- A – 425718 6.61%
- … skipping a few …
- J – 5329 0.08%
- Q – 4923 0.08%
- Z – 3378 0.05%

English text letter frequencies

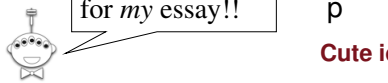


Variable Length Encodings

The zzyzva is known to be a xenophobic creature with a zealous personality…

TEXT FILE

Letter	frequency	Binary code
z	0.25	0
y	0.10	1
x	0.09	00
a	0.08	01
…		
r	0.02	10100111100
p	0.01	10100111101



Cute idea, but what's the problem here?

The Prefix Property

The zzyzva is known to be a xenophobic creature with a zealous personality…

TEXT FILE

Letter	frequency	Binary code
z	0.25	00
y	0.10	01
x	0.09	10
a	0.08	111
r	0.02	1100

101110100001100 = 10 111 01 00 00 1100

x a y z z r

Consider the Language "Spamish" which has only four letters in its alphabet…

Letter	freq	Fixed Length	Variable Length
s	0.6	00	0
p	0.2	01	10
a	0.1	10	110
m	0.1	11	111

Expected average number of bits per symbol = 2

1.6 is 80% of 2.0, so we expect 20% space savings!

Expected average number of bits per symbol = 0.6x1 + 0.2x2 + 0.1x3 + 0.1x3 = 1.6

The Variable Length Coding Problem…

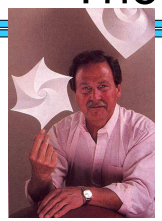
Letter	Frequency
a_1	freq(a_1)
a_2	freq(a_2)
a_3	freq(a_3)
…	
a_n	freq(a_n)

These frequencies are from the specific file that we're planning to compress!!

Objective: Find a binary prefix code that minimizes…

freq(a_1) × codelength(a_1) +
freq(a_2) × codelength(a_2) + …
freq(a_n) × codelength(a_n)

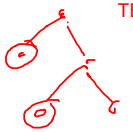
The David Huffman Story!



map smppam
ssampamsmam
…

TEXT FILE

Letter	freq
s	0.6
p	0.2
a	0.1
m	0.1

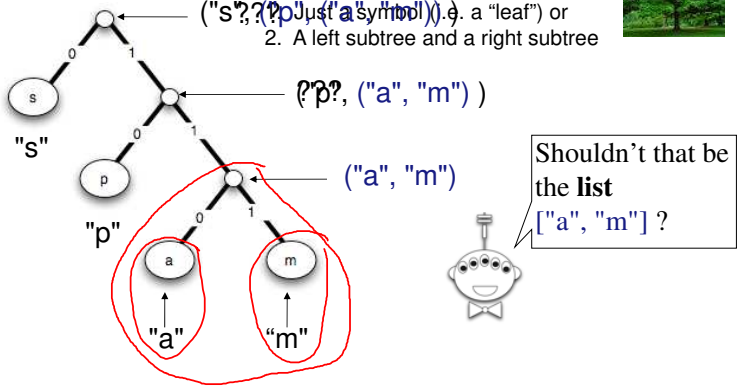


ENCODING:

1. Scan text file to compute frequencies
2. Build Huffman tree
3. Find code for every symbol (letter)—why is this a prefix code?
4. Create new compressed file by saving the entire code at the top of the file followed by the code for each symbol (letter) in the file

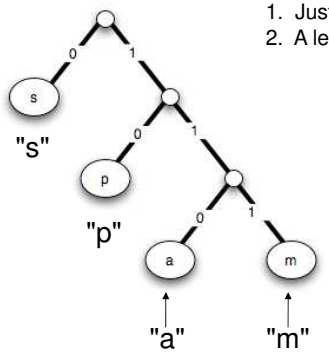
I wonder about trees—Robert Frost
We wonder about Robert Frost—Trees

Recursive definition of a binary tree…
A binary tree is:
1. Just a symbol (i.e. a "leaf") or
2. A left subtree and a right subtree



I wonder about trees—Robert Frost
We wonder about Robert Frost—Trees

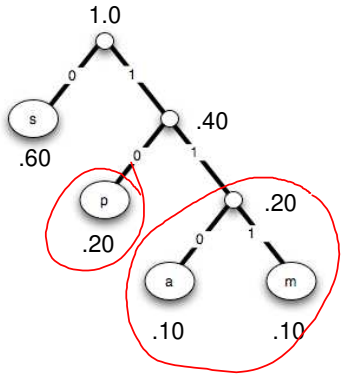
Recursive definition of a binary tree…
A binary tree is:
1. Just a symbol (i.e. a "leaf") or
2. A left subtree and a right subtree



Huffman's Algorithm

- Find two lowest-frequency symbols
- Combine them into a tree node
  - Add their frequencies
- Repeat until only one node left

Huffman Example



You Try It!

Worksheet

Letter	Frequency
h	0.40
a	0.20
r	0.15
v	0.15
e	0.06
y	0.04

Build the tree and write down the codes for each of the symbols

Then encode the string "haha" using this code

32 bits

Building the Huffman Tree!

Letter	Frequency
h	0.40
a	0.20
r	0.15
v	0.15
e	0.06
y	0.04

```
frequencies = {"h": 0.40,
"a": 0.20, "r": 0.15,
"v": 0.15, "e": 0.06,
"y": 0.04}
```

How do I get these!?

OBJECTIVE: Convert this into a tree…

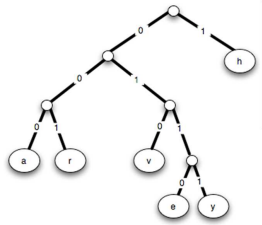
Building the Huffman Tree!

```
frequencies = {"h": 0.40,
"a": 0.20, "r": 0.15,
"v": 0.15, "e" :0.06,
"y": 0.04}
```

Assume a function `minfrequency(frequencies)` that returns the character (key) with min frequency!

```
def make_tree(self, counter):
```

(((a, r) , (v, (e, y))) , h)



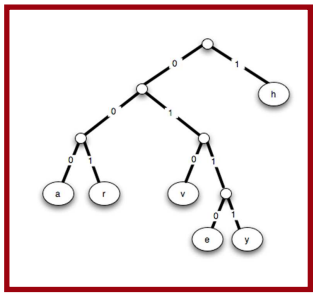
`minfrequency` can be written using `Counter`'s `most_common()` function with `[-1]` and the `del` keyword!

The Huffman Encoder

```
frequencies = {"h": 0.40,
"a": 0.20, "r": 0.15,
"v": 0.15, "e": 0.06,
"y": 0.04}
```

Read input file into string S
Count letter frequencies in S
Build the Huffman tree
Find the Huffman code for each character
binary_sequence = ""
for each character c in S
** binary_sequence += Huffman code for c**
Write encoding to codes file
Write output to compressed file

foo.txt.HUFF



h: 1 a : 000 r: 001
v: 010 e: 0110 y: 0111

The Huffman Decoder

Read compressed file into string E
Read Huffman table from codes file
Expand E to original text string S
Save S to file

```
6
2001
h: 1      a : 000    r: 001
v: 010    e: 0110    y: 0111
$a!*&spam^>\n):^)
pen*guin!*blah/~.\cs5!.<-42
blahblahblah
```

OOPs! (Object-Oriented Programs)

```
>>> today = Date(11, 10, 2020)
>>> due = Date(11, 16, 2020)
>>> due - today
6
>> if due > today:
       print("Go watch a movie!")
```

One Implementation

```
class Date(object):
  def __init__(self, m, d, y):
    self.month = m
    self.day = d
    self.year = y

>>> d = Date(1, 21, 1969)
```

Another Implementation…

```
class Date(object):
  def __init__(self, m, d, y):
    self.daysSince1900 = …

>>> d = Date(1, 21, 1969)
```

Why would any sane person ***want*** to store the date as the number of days since January 1, 1900?

Getters and Setters

```
class Date(object):
  def __init__(self, m, d, y):
    self._daysSince1900 = …

  def setDay(self, d):
    if d <= 0 or d > 31:
      …
    else:
      self._daysSince1900 = …
```

```
>>> d = Date(1, 21, 1969)
>>> d.setDay(28)     # SETTER
>>> x = d.getDay()   # GETTER
```

Date "Abstraction"

```
Date
  __init__(self, month, day, year)
 setDay(self, day)
 setMonth(self, month)
 setYear(self, year)
 getDay(self)
 getMonth(self)
 getYear(self)
 ==, >, <, >=, <=, +, -
```

The Advantage of Abstraction

Rack-and-pinion?
Recirculating ball?
Worm-and-sector?
Steer-by-wire?

An *Import*ant Point

```
import turtle
import math
import Date

turtle.forward(100)
print(math.cos(math.pi))
today = Date.Date(11, 9, 2011)
```

```
from turtle import *
from math import cos, pi
from Date import *

forward(100)
print(cos(pi))
today = Date(11, 9, 2011)
```

Default Arguments

In my experience, arguments are usually default of deperson who started them!

```
class Student(object):

    def __init__(self, firstName, lastName,
        school = "HMC", major = "undeclared")
```

```
>>> where = Student("Carmen", "Sandiego")
>>> stu = Student("Stu", "Dious", "PIT")
>>> anna = Student("Anna", "Litik", major = "Physics")
>>> elmo = Student("Elmo")
>>> bigBird = Student("Big", "Bird", firstName = "Tweety")
>>> bart = Student(school="PIT", "Bart", "Simpson")
```

Inheritance

```
class Person(object):
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

    def asleep(self, time):
        return 0 <= time <= 7  # MILITARY TIME IN HOURS

    def __repr__(self):
        return self.firstName + " " + self.lastName
```

```
>>> geoff = Person("Geoff", "Kuenning")
>>> geoff
Geoff Kuenning
>>> geoff.asleep(2)
True
```

```
class Person(object):
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

    def asleep(self, time):
        return 0 <= time <= 7

    def __repr__(self):
        return self.firstName + " " + self.lastName

class Student(Person):
    def __init__(self, first, last, age):
        super(Student, self).__init__(self, first, last)
        self.age = age

    def asleep(self, time):
        return 3 <= time <= 11
    def __repr__(self):
        return Person.__repr__(self) + ", " + str(self.age) + " years old"
```

Sleeping until 11 AM!?

```
>>> s = Student("Sue", "Persmart", 18)
>>> s
Sue Persmart, 18 years old
>>> s.asleep(2)
False
```

```
class Person(object):
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

    def asleep(self, time):
        return 0 <= time <= 7

    def __repr__(self):
        return self.firstName + " " + self.lastName

class Student(Person):
    def __init__(self, first, last, age):
        super(Student, self).__init__(self, first, last)
        self.age = age

    def asleep(self, time):
        return 3 <= time <= 11

    def __repr__(self):
        return Person.__repr__(self) + ", " + str(self.age) + " years old"

class Mudder(Student):
    def __init__(self, first, last, age, dorm):
        super(Mudder, self).__init__(self, first, last, age)
        self.dorm = dorm

    def asleep(self, time):
        return False
```

```
>>> wally = Mudder("Wally", "Wart",
                   42, "West")
>>> wally
?
>>> wally.asleep(4)
?
```

Get some sleep!!!

The Dangers of Inheritance