Huffman Encoding

CHIDATA

Dictionary Encoding

Leverage redundancy in data collections

```
data = [‘red’, ‘blue’, ‘red’, ‘red’]
```

Dictionary Encode: Analyze data and build a lookup table with fixed size values.

```
‘red’ => 0
‘blue’ => 1

data = [0, 1, 0, 0]
```

Dictionary Encoding

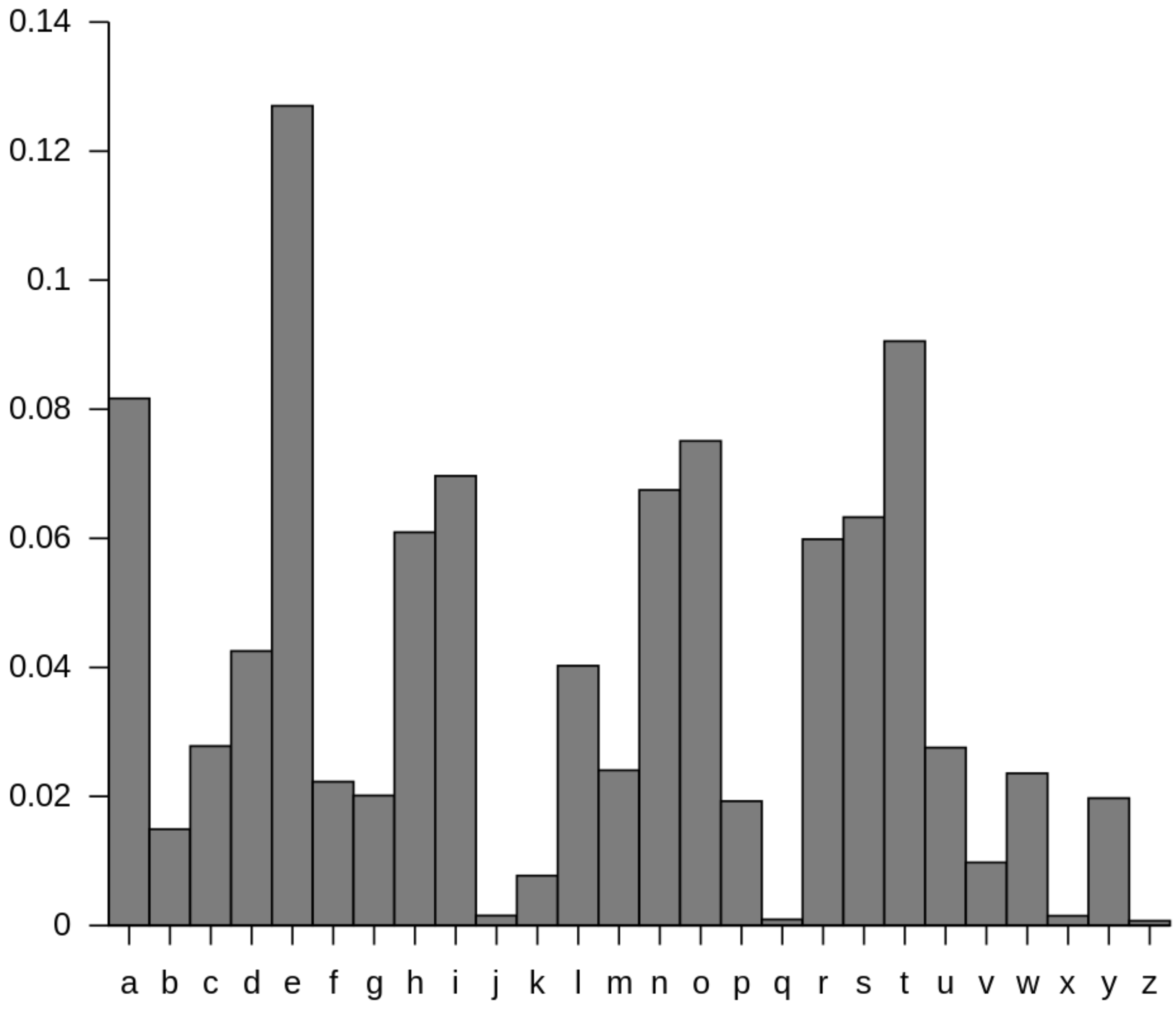
The problem with skew:

Color	key
Red (Prob 0.8)	“00”
Green (Prob 0.02)	“01”
Blue (Prob 0.15)	“10”
Black (Prob 0.3)	“11”

Same number of bits to represent the keys even though Red is way more popular (40x) than Green

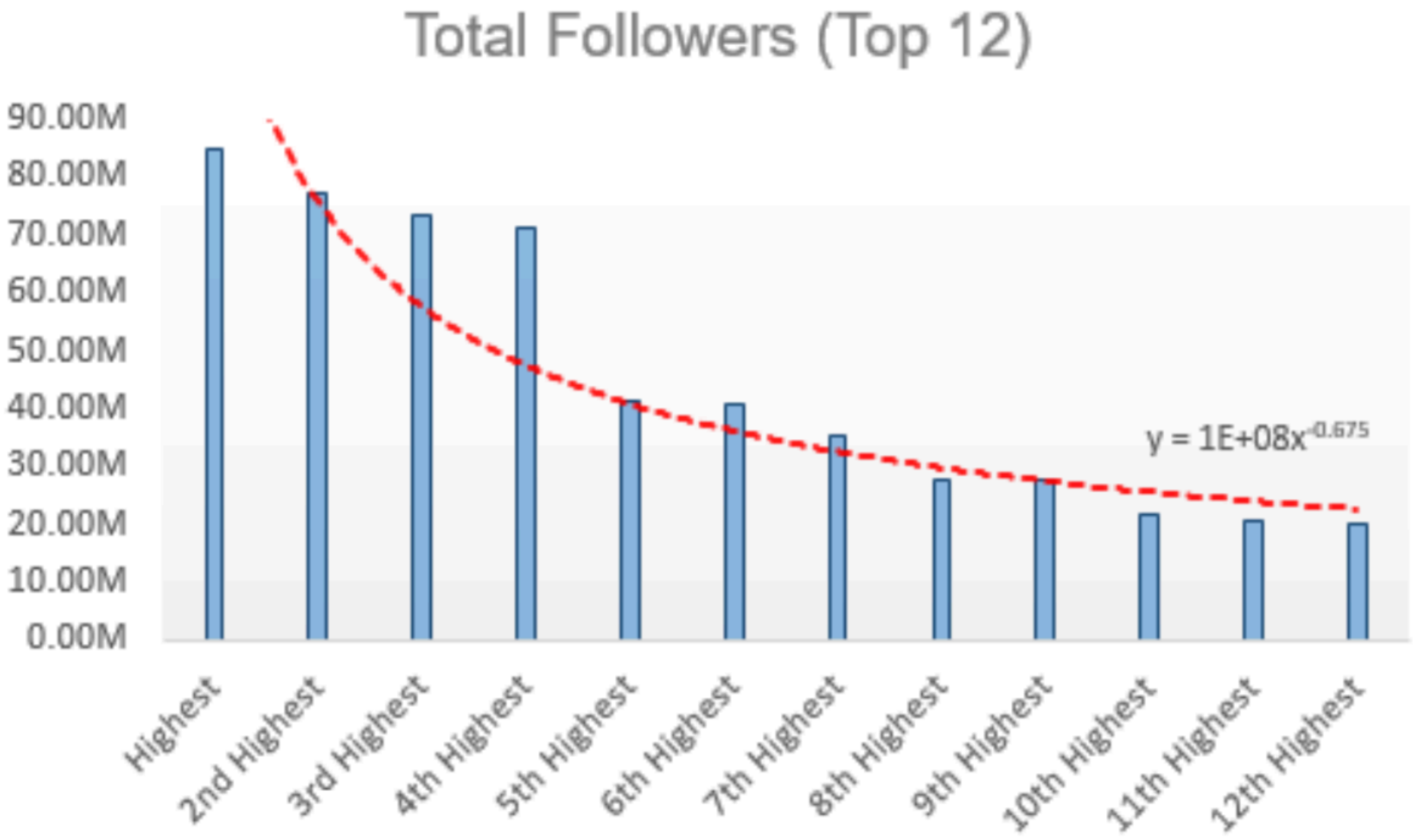
Skew in Real Data

Letter frequency in the English language



Skew in Real Data

Followers on Twitter



Naive Solution

Algorithm: Sort by frequency cut leading zeros

Color	key
Red (Prob 0.8)	“0”
Green (Prob 0.02)	“11”
Blue (Prob 0.15)	“1”
Black (Prob 0.03)	“10”

Problem?

```
data = [Blue,Red]
   enc = 10


  data = [Black]
   enc = 10
```

Delimiters?

Algorithm 2: Designate a character as a delimiter character “00”

Color	**key**
Red (Prob 0.8)	“0”
Green (Prob 0.02)	“11”
Blue (Prob 0.15)	“10”
Black (Prob 0.03)	“1”

Problem?

```
data = [Black,Red]
enc = 1000
data = [Blue]
enc = 1000
```

Delimiters work when the atomic items have a fixed sizes (like chars)

Are there codes that work?

Color	key
Red (Prob 0.8)	“0”
Green (Prob 0.02)	“1110”
Blue (Prob 0.15)	“110”
Black (Prob 0.03)	“10”

```
enc = 0101101110
```

```
enc = [red,black,blue,green]
```

Is there any other possible decoding? Why?

Prefix-Free Codes

No code is a “prefix” of another code

Red	0
Black	10
Blue	110
Green	1110

Color	**key**
Red (Prob 0.8)	“0”
Green (Prob 0.02)	“1110”
Blue (Prob 0.15)	“110”
Black (Prob 0.03)	“10”

Prefix-Free Codes

Left-to-right decoding is always unambiguous

Red	0
Black	10
Blue	110
Green	1110

Color	**key**
Red (Prob 0.8)	“0”
Green (Prob 0.02)	“1110”
Blue (Prob 0.15)	“110”
Black (Prob 0.03)	“10”

Does it save on space?

Color	**key**
Red (Prob 0.8)	“0”
Green (Prob 0.02)	“1110”
Blue (Prob 0.15)	“110”
Black (Prob 0.03)	“10”

Expected Storage = 0.8*1 + 0.03*2 + 0.15*3 + 0.02*4

Expected Storage = 1.39

Small savings can add up!

Does it save on space?

Color	key
Red (Prob 0.9)	“0”
Green (Prob 0.001)	“1110”
Blue (Prob 0.049)	“110”
Black (Prob 0.05)	“10”

Expected Storage = 0.9*1 + 0.05*2 + 0.049*3 + 0.001*4

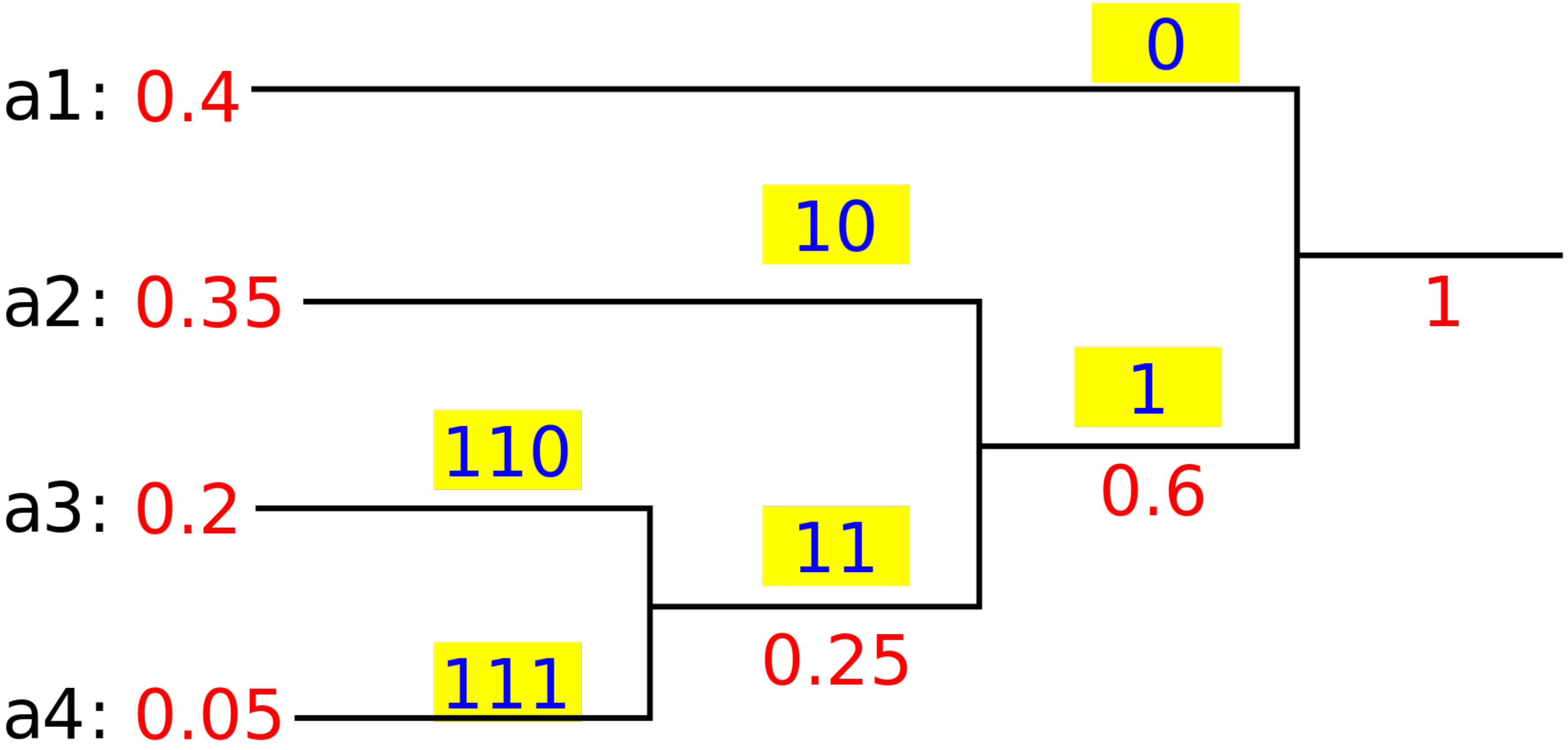
Expected Storage = 1.151

Better compression when there is higher skew

Huffman Coding

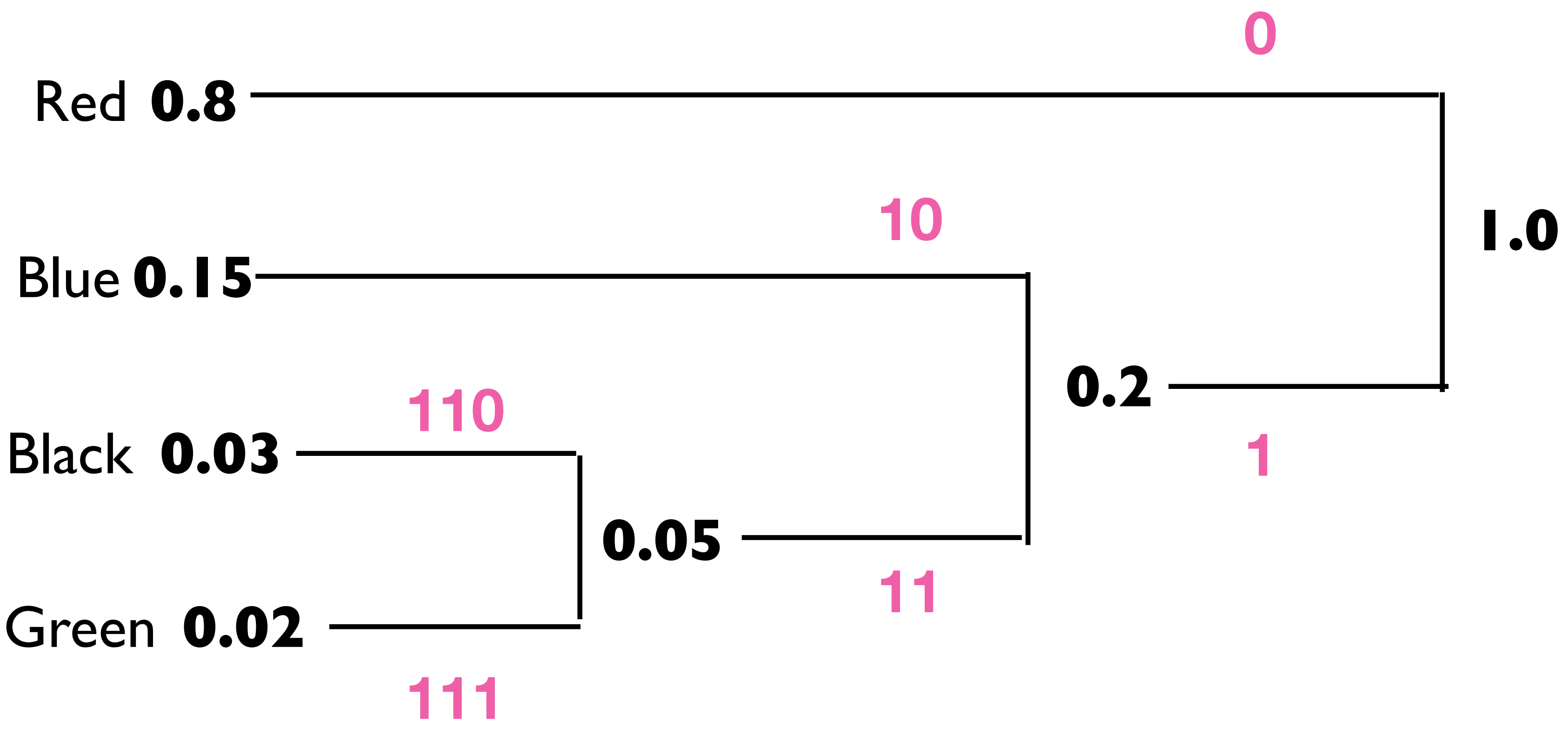
An algorithm to find optimal prefix-free codes.

“Bottom up” binary tree construction where you merge least frequent items successively and assign codes to each of the branches of the tree.



Does it save on space?

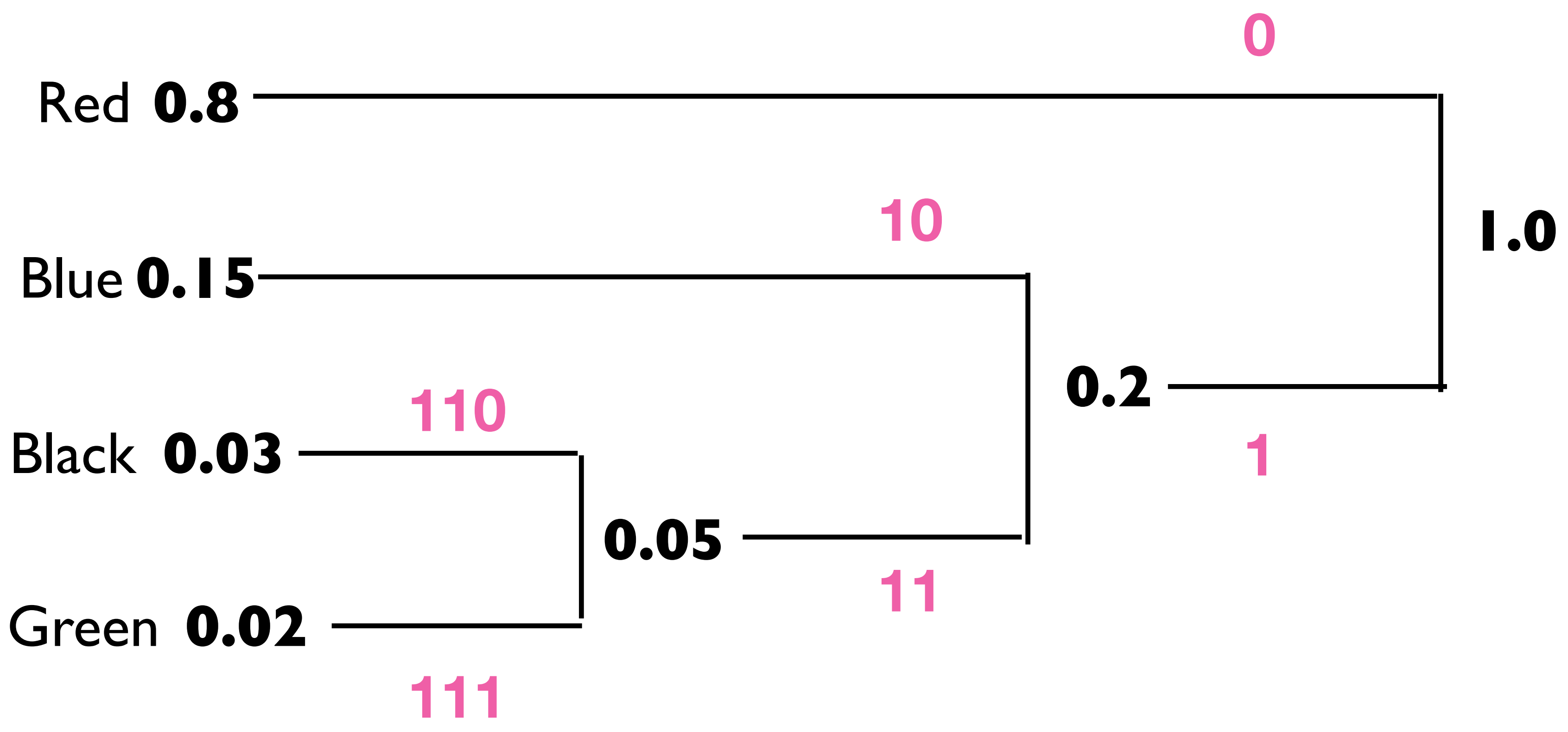
Color
Red (Prob 0.8)
Green (Prob 0.02)
Blue (Prob 0.15)
Black (Prob 0.03)



Does it save on space?

Color	Key
Red (Prob 0.8)	"0"
Green (Prob 0.02)	"111"
Blue (Prob 0.15)	"10"
Black (Prob 0.03)	"110"

Expected Storage
0.8*1 + 0.03*3 + 0.15*2 + 0.02*3 = 1.25

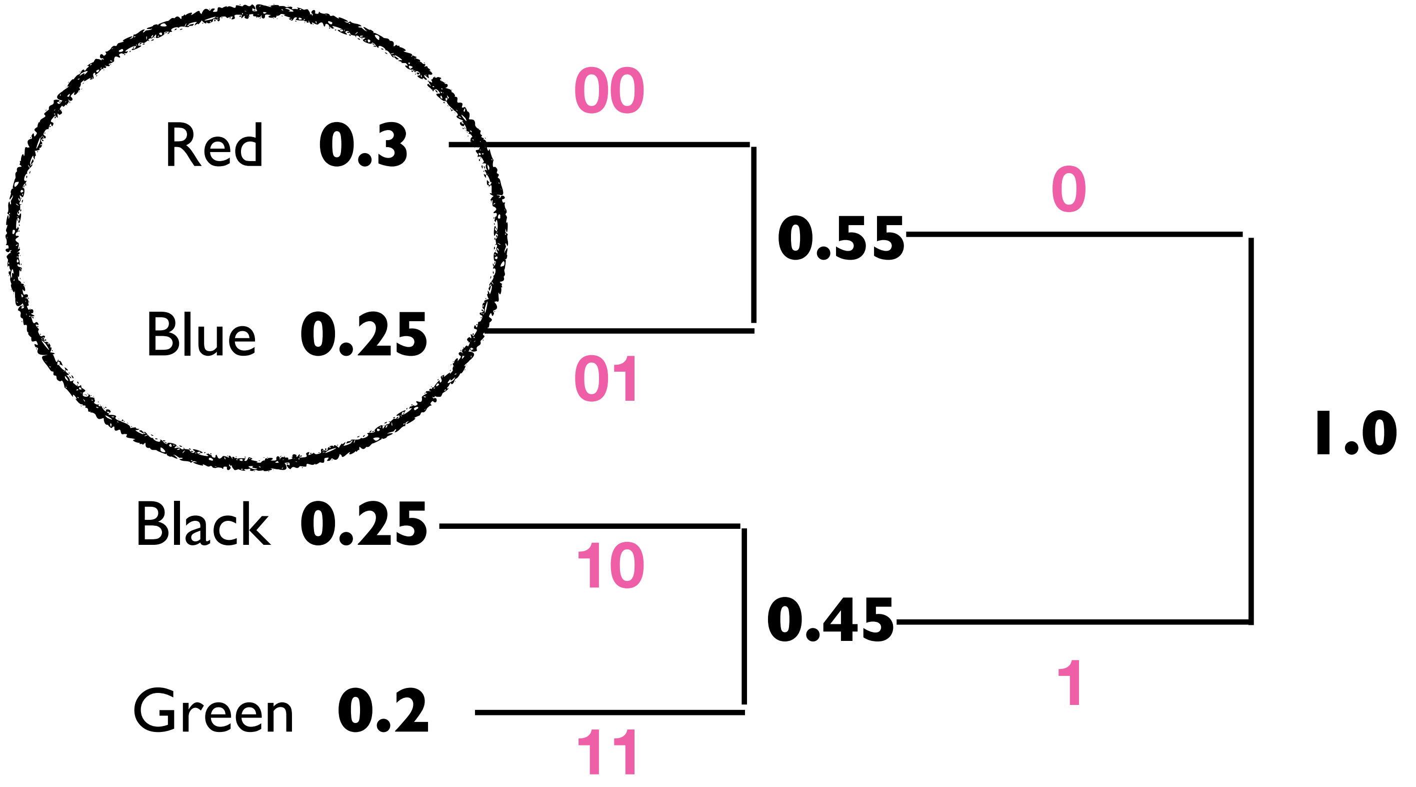


Another Example

Color
Red (Prob 0.3)
Green (Prob 0.2)
Blue (Prob 0.25)
Black (Prob 0.25)

Degenerates into a fixed-length code when there isn't enough skew

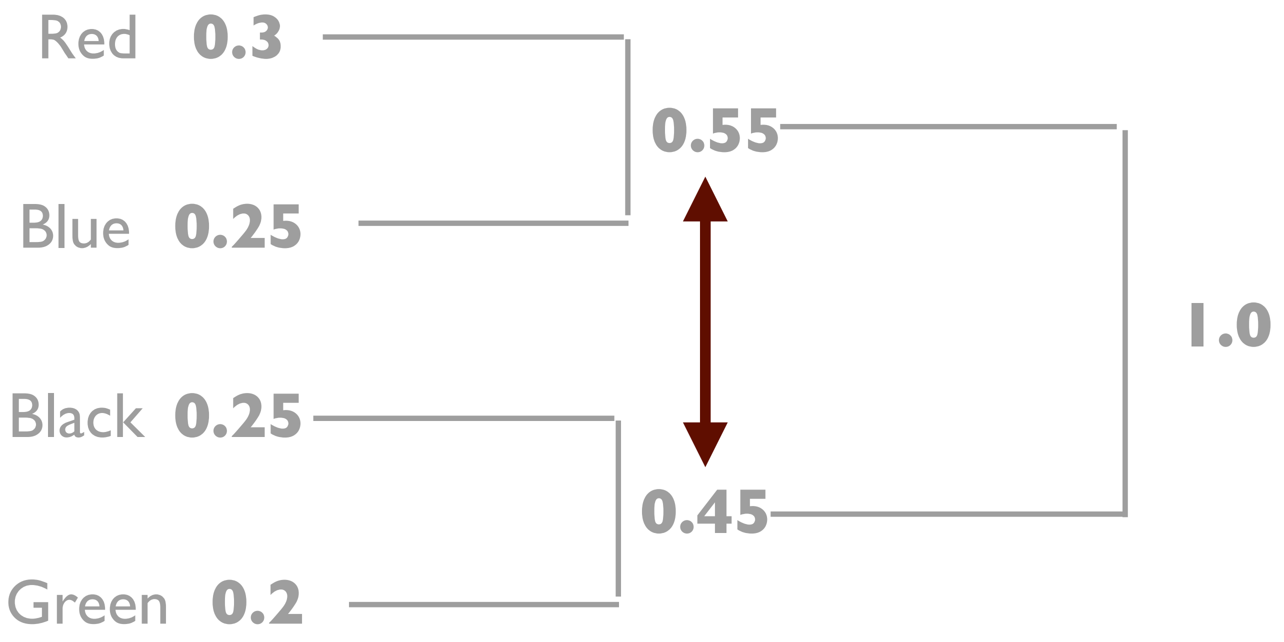
Now the two lowest



Another Example

Color
Red (Prob 0.3)
Green (Prob 0.2)
Blue (Prob 0.25)
Black (Prob 0.25)

Intuition: Variable length codes are preferred when at least one item has a higher probability than a combination of two items.


Red 0.3
Blue 0.25
Black 0.25
Green 0.2
0.55
0.45
1.0