Huffman Trees

Greedy Algorithm for Data Compression

Tyler Moore

CS 2123, The University of Tulsa

Some slides created by or adapted from Dr. Kevin Wayne. For more information see

https://www.cs.princeton.edu/courses/archive/fall12/cos226/lectures.php

Data compression

Compression reduces the size of a file:

- To save space when storing it.
- To save time when transmitting it.
- Most files have lots of redundancy.

Who needs compression?

- Moore's law: # transistors on a chip doubles every 18–24 months.
- Parkinson's law: data expands to fill space available.
- Text, images, sound, video, ...

> *" Everyday, we create 2.5 quintillion bytes of data—so much that 90% of the data in the world today has been created in the last two years alone. "* — *IBM report on big data (2011)*

Basic concepts ancient (1950s), best technology recently developed.

Applications

Generic file compression.

- Files: GZIP, BZIP, 7z.
- Archivers: PKZIP.
- File systems: NTFS, HFS+, ZFS.

Multimedia.

- Images: GIF, JPEG.
- Sound: MP3.
- Video: MPEG, DivX™, HDTV.



Communication.

- ITU-T T4 Group 3 Fax.
- V.42bis modem.
- Skype.


Databases. Google, Facebook, ...

Google facebook

Lossless compression and expansion

Message. Binary data B we want to compress.
Compress. Generates a "compressed" representation $C(B)$.
Expand. Reconstructs original bitstream B.

uses fewer bits (you hope)

Basic model for data compression

Compression ratio. Bits in $C(B)$ / bits in B.

Ex. 50–75% or better compression ratio for natural language.

Rdenudcany in Enlgsih lnagugae

Q. How mcuh rdenudcany is in the Enlgsih lnagugae?

> *" ... randomising letters in the middle of words [has] little or no effect on the ability of skilled readers to understand the text. This is easy to denmtrasote. In a pubiltacion of New Scnieitst you could ramdinose all the letetrs, keipeng the first two and last two the same, and reibadailty would hadrly be aftcfeed. My ansaylis did not come to much beucase the thoery at the time was for shape and senqeuce retigcionon. Saberi's work sugsegts we may have some pofrweul palrlael prsooscers at work. The resaon for this is suerly that idnetiyfing coentnt by paarllel prseocsing speeds up regnicoiton. We only need the first and last two letetrs to spot chganes in meniang. "* — Graham Rawlinson

A. Quite a bit.

Variable-length codes

Use different number of bits to encode different chars.

Ex. Morse code: • • • – – – • • •

Issue. Ambiguity.

SOS ?

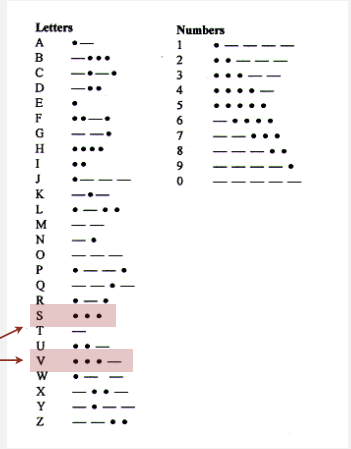
V7 ?

IAMIE ?

EEWNI ?

In practice. Use a medium gap to separate codewords.

codeword for S is a prefix of codeword for V



Variable-length codes

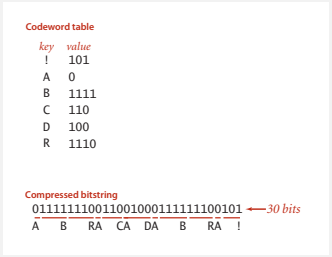
Q. How do we avoid ambiguity?

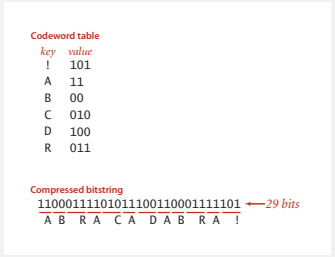
A. Ensure that no codeword is a prefix of another.

Ex 1. Fixed-length code.

Ex 2. Append special stop char to each codeword.

Ex 3. General prefix-free code.



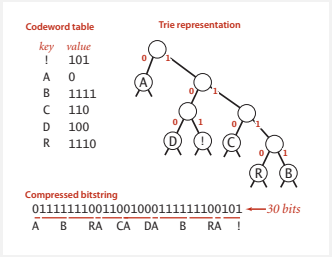


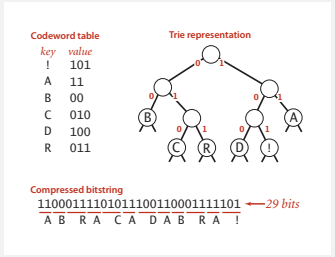
Prefix-free codes: trie representation

Q. How to represent the prefix-free code?

A. A binary trie!

- Chars in leaves.
- Codeword is path from root to leaf.





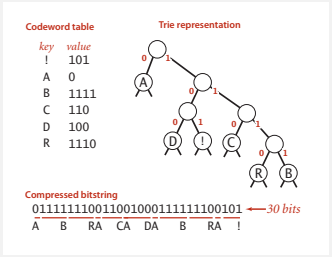
Prefix-free codes: compression and expansion

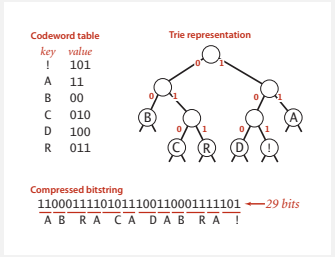
Compression.

- Method 1: start at leaf; follow path up to the root; print bits in reverse.
- Method 2: create ST of key-value pairs.

Expansion.

- Start at root.
- Go left if bit is 0; go right if 1.
- If leaf node, print char and return to root.





22

Average weighted code length

Definition

Given a set of symbols $s \in S$ and corresponding frequencies f_s where $\sum_{s \in S} f_s = 1$, the average weighted code length using a binary trie is $\sum_{s \in S} f_s \cdot \mathrm{Depth(s)}$.

Shannon-Fano codes

Q. How to find best prefix-free code?

Shannon-Fano algorithm:

- Partition symbols S into two subsets S_0 and S_1 of (roughly) equal freq.
- Codewords for symbols in S_0 start with 0; for symbols in S_1 start with 1.
- Recur in S_0 and S_1.

char	freq	encoding
A	5	0...
C	1	0...

S_0 = codewords starting with 0

char	freq	encoding
B	2	1...
D	1	1...
R	2	1...
!	1	1...

S_1 = codewords starting with 1

Problem 1. How to divide up symbols?

Problem 2. Not optimal!

27

Shannon-Fano Codes Exercise

Procedure for Creating Huffman Tree and Codes (RLW)

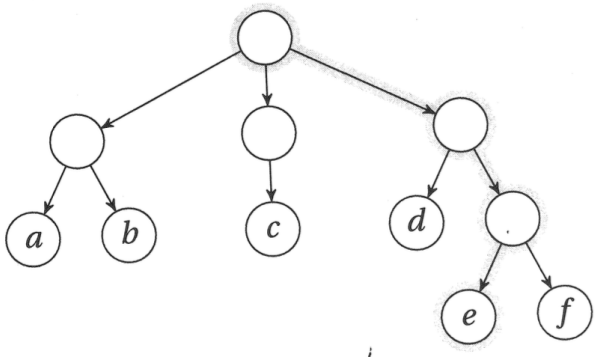
1. Initialize each symbol into a one-node tree with weight corresponding to the probability of the symbol's occurrence
2. REPEAT
   a. Select the two trees with smallest weights (break ties randomly).
   b. Combine two trees into one tree whose root is the sum of weights of two trees
   UNTIL one tree remains
3. Assign 0 to left edge and 1 to right edge in tree.
4. Huffman code is the binary value constructed from the path from root to leaf

Huffman Code Examples

Huffman Coding: Implementation

- We will use nested lists to represent the Huffman tree structure in Python

```
>>> T = [["a", "b"], ["c"], ["d", ["e", "f"]]]
>>> T[0][1]
'b'
>>> T[2][1][0]
'e'
```



Huffman Coding: Implementation

- To efficiently implement Huffman codes, must use a priority queue
- Place trees onto the queue with associated frequency
- Add merged trees onto the priority queue with updated frequencies

Huffman Coding: Implementing the Tree

```
from heapq import heapify, heappush, heappop
def huffman(seq, frq):

    trees = list(zip(frq, seq))
    heapify(trees)              # A min-heap based on freq
    while len(trees) > 1:       # Until all are combined
        fa, a = heappop(trees)  # Get the two smallest trees
        fb, b = heappop(trees)
        heappush(trees, (fa+fb, [a, b])) # Combine and re-add
    return trees[0][-1]
```


Huffman Coding: Implementing the Codes

```
def codes(tree, prefix = ""):
    if len(tree) == 1:
        return [tree, prefix]
    return codes(tree[0],prefix+"0")+ \
           codes(tree[1],prefix+"1")

def codesd(tr):
    cmap = {}
    def codesh(tree, prefix = ""):
        if len(tree) == 1:
            cmap[tree]=prefix
        else:
            codesh(tree[0],prefix+"0")
            codesh(tree[1],prefix+"1")
    codesh(tr,"")
    return cmap
```


Huffman Coding: Implementation

```
    seq = "abcdefghi"
    frq = [4,5,6,9,11,12,15,16,20]
    htree = huffman(seq,frq)
    print htree
    print codes(htree)
    ch = codesd(htree)
    print ch
    text = "abbafabgee"
    print text, "␣encodes␣to:"
    print "".join([ch[x] for x in text])
    """
[['i',[['a','b'],'e']],[['f','g'],[['c','d'],'h']]]
['i','00','a','0100','b','0101','e','011','f','100',
'g','101','c','1100','d','1101','h','111']
{'a':'0100','c':'1100','b':'0101','e':'011',
'd':'1101','g':'101','f':'100','i':'00','h':'111'}
abbafabgee␣␣encodes␣to:
0100010101010100100010001011010110 11
```