Greedy Algorithms: Huffman Coding

Data Compression

Q. Given a text that uses 32 symbols (26 different letters, space, and some punctuation characters), how can we encode this text in bits?
A. We can encode 2^5 different symbols using a fixed length of 5 bits per symbol. This is called fixed length encoding.

Q. Some symbols (e, t, a, o, i, n) are used far more often than others. How can we use this to reduce our encoding?
A. Encode these characters with fewer bits, and the others with more bits.

Q. How do we know when the next symbol begins?
A. Use a separation symbol (like the pause in Morse), or make sure that there is no ambiguity by ensuring that no code is a prefix of another one.

Ex. c(a) = 01 What is 0101?
c(b) = 010
c(e) = 1

3

Prefix Codes

Definition. A prefix code for a set S is a function c that maps each $x \in S$ to 1s and 0s in such a way that for $x,y \in S$, $x \neq y$, c(x) is not a prefix of c(y).

Ex. c(a) = 11
c(e) = 01
c(k) = 001
c(l) = 10
c(u) = 000

Q. What is the meaning of 1001000001 ?
A. "leuk"

Suppose frequencies are known in a text of 1G:
f_a=0.4, f_e=0.2, f_k=0.2, f_l=0.1, f_u=0.1
Q. What is the size of the encoded text?
A. $2*f_a + 2*f_e + 3*f_k + 2*f_l + 4*f_u = 2.4G$

5

Optimal Prefix Codes

Definition. The average bits per letter of a prefix code c is the sum over all symbols of its frequency times the number of bits of its encoding:

$$ABL(c) = \sum_{x \in S} f_x \cdot |c(x)|$$

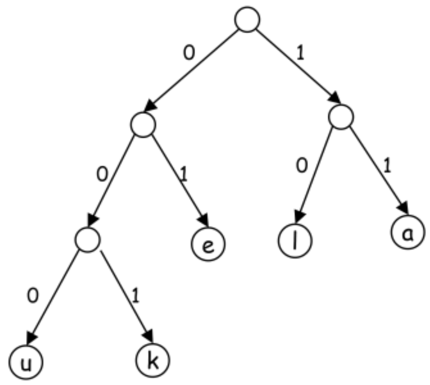
We would like to find a prefix code that is has the lowest possible average bits per letter.

Suppose we model a code in a binary tree...

6

Representing Prefix Codes using Binary Trees

Ex. c(a) = 11
c(e) = 01
c(k) = 001
c(l) = 10
c(u) = 000



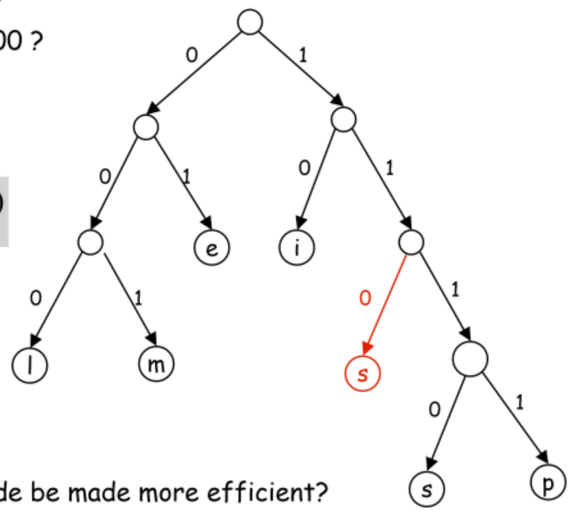
Q. How does the tree of a prefix code look?
A. Only the leaves have a label.
Pf. An encoding of x is a prefix of an encoding of y if and only if the path of x is a prefix of the path of y.

8

Representing Prefix Codes using Binary Trees

Q. What is the meaning of
111010001111101000 ?
A. "simpel"

$$ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)$$



Q. How can this prefix code be made more efficient?
A. Change encoding of p and s to a shorter one.
This tree is now full.

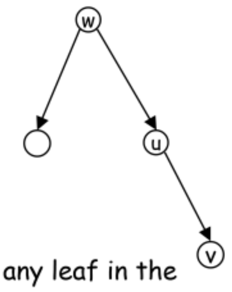
11

Representing Prefix Codes using Binary Trees

Definition. A tree is full if every node that is not a leaf has two children.

Claim. The binary tree corresponding to the optimal prefix code is full.
Pf. (by contradiction)

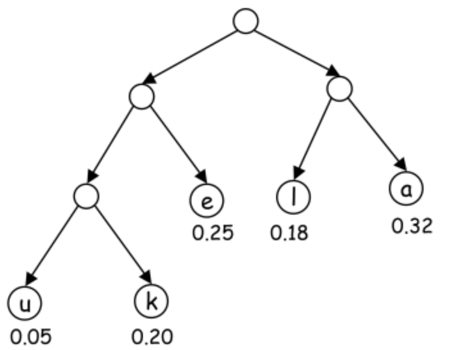
- Suppose T is binary tree of optimal prefix code and is not full.
- This means there is a node u with only one child v.
- Case 1: u is the root; delete u and use v as the root
- Case 2: u is not the root
  - let w be the parent of u
  - delete u and make v be a child of w in place of u
- In both cases the number of bits needed to encode any leaf in the subtree of v is decreased. The rest of the tree is not affected.
- Clearly this new tree T' has a smaller ABL than T. Contradiction.

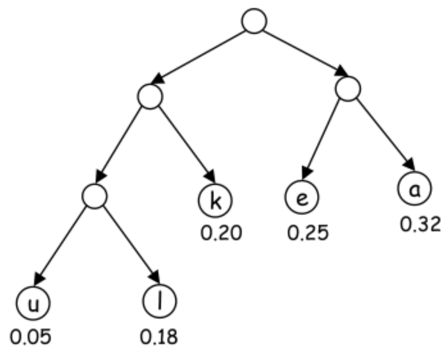


Optimal Prefix Codes: False Start

Q. Where in the tree of an optimal prefix code should letters be placed with a high frequency?
A. Near the top.

Greedy template. Create tree top-down, split S into two sets S_1 and S_2 with (almost) equal frequencies. Recursively build tree for S_1 and S_2.
[Shannon-Fano, 1949] f_a=0.32, f_e=0.25, f_k=0.20, f_l=0.18, f_u=0.05





Optimal Prefix Codes: Huffman Encoding

Observation. Lowest frequency items should be at the lowest level in tree of optimal prefix code.

Observation. For n > 1, the lowest level always contains at least two leaves.

Observation. The order in which items appear in a level does not matter.

Claim. There is an optimal prefix code with tree T* where the two lowest-frequency letters are assigned to leaves that are siblings in T*.

Greedy template. [Huffman, 1952] Create tree bottom-up.
Make two leaves for two lowest-frequency letters y and z.
Recursively build tree for the rest using a meta-letter for yz.

Optimal Prefix Codes: Huffman Encoding

```
Huffman(S) {
   if |S|=2 {
      return tree with root and 2 leaves
   } else {
      let y and z be lowest-frequency letters in S
      S' = S
      remove y and z from S'
      insert new letter ω in S' with fω=fy+fz
      T' = Huffman(S')
      T = add two children y and z to leaf ω from T'
      return T
   }
}
```

Q. What is the time complexity?

Optimal Prefix Codes: Huffman Encoding

```
Huffman(S) {
   if |S|=2 {
      return tree with root and 2 leaves
   } else {
      let y and z be lowest-frequency letters in S
      S' = S
      remove y and z from S'
      insert new letter ω in S' with fω=fy+fz
      T' = Huffman(S')
      T = add two children y and z to leaf ω from T'
      return T
   }
}
```

Q. What is the time complexity?
A. $T(n) = T(n-1) + O(n)$
so $O(n^2)$
Q. How to implement finding lowest-frequency letters efficiently?
A. Use priority queue for S: $T(n) = T(n-1) + O(\log n)$ so $O(n \log n)$

18

Huffman Encoding: Greedy Analysis

Claim. Huffman code for S achieves the minimum ABL of any prefix code.
Pf. by induction, based on optimality of T' (y and z removed, ω added)
(see next page)

Claim. $ABL(T')=ABL(T)-f_\omega$
Pf.

$$
\begin{aligned}
\mathrm{ABL}(T) &= \sum_{x\in S} f_x \cdot \mathrm{depth}_T(x) \\
&= f_y \cdot \mathrm{depth}_T(y) + f_z \cdot \mathrm{depth}_T(z) + \sum_{x\in S, x\neq y,z} f_x \cdot \mathrm{depth}_T(x) \\
&= (f_y + f_z)\cdot(1+\mathrm{depth}_T(\omega)) + \sum_{x\in S, x\neq y,z} f_x \cdot \mathrm{depth}_T(x) \\
&= f_\omega \cdot (1+\mathrm{depth}_T(\omega)) + \sum_{x\in S, x\neq y,z} f_x \cdot \mathrm{depth}_T(x) \\
&= f_\omega + \sum_{x\in S'} f_x \cdot \mathrm{depth}_{T'}(x) \\
&= f_\omega + \mathrm{ABL}(T')
\end{aligned}
$$

20

Huffman Encoding: Greedy Analysis

Claim. Huffman code for S achieves the minimum ABL of any prefix code.
Pf. (by induction)
Base: For n=2 there is no shorter code than root and two leaves.
Hypothesis: Suppose Huffman tree T' for S' of size n-1 with ω instead of y and z is optimal. (IH)
Step: (by contradiction)
- *Idea of proof:*
  - Suppose other tree Z of size n is better.
  - Delete lowest frequency items y and z from Z creating Z'
  - Z' cannot be better than T' by IH.

23

Huffman Encoding: Greedy Analysis

Claim. Huffman code for S achieves the minimum ABL of any prefix code.
Pf. (by induction)
Base: For n=2 there is no shorter code than root and two leaves.
Hypothesis: Suppose Huffman tree T' for S' with ω instead of y and z is optimal. (IH)
Step: (by contradiction)
- Suppose Huffman tree T for S is not optimal.
- So there is some tree Z such that ABL(Z) < ABL(T).
- Then there is also a tree Z for which leaves y and z exist that are siblings and have the lowest frequency (see observation).
- Let Z' be Z with y and z deleted, and their former parent labeled ω.
- Similar T' is derived from S' in our algorithm.
- We know that ABL(Z')=ABL(Z)-f_ω, as well as ABL(T')=ABL(T)-f_ω.
- But also ABL(Z) < ABL(T), so ABL(Z') < ABL(T').
- Contradiction with IH.

24

Divide & Conquer : Merge Sort

Divide & Conquer: Merge Sort

Divide-and-conquer paradigm

Divide-and-conquer.

- Divide up problem into several subproblems (of the same kind).
- Solve (conquer) each subproblem recursively.
- Combine solutions to subproblems into overall solution.

Most common usage.

- Divide problem of size n into two subproblems of size $n/2$. ← $O(n)$ time
- Solve (conquer) two subproblems recursively.
- Combine two solutions into overall solution. ← $O(n)$ time

Consequence.

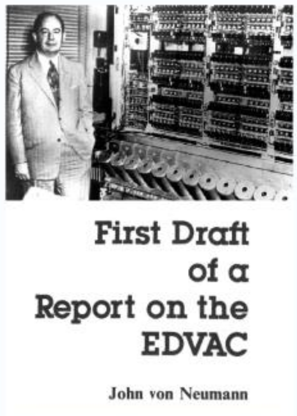
- Brute force: $\Theta(n^2)$.
- Divide-and-conquer: $O(n \log n)$.

attributed to Julius Caesar

2

Mergesort

- Recursively sort left half.
- Recursively sort right half.
- Merge two halves to make sorted whole.



input

A	L	G	O	R	I	T	H	M	S

sort left half

A	G	L	O	R	(I)	(T)	(H)	(M)	(S)

sort right half

(A)	(G)	(L)	(O)	(R)	H	I	M	S	T

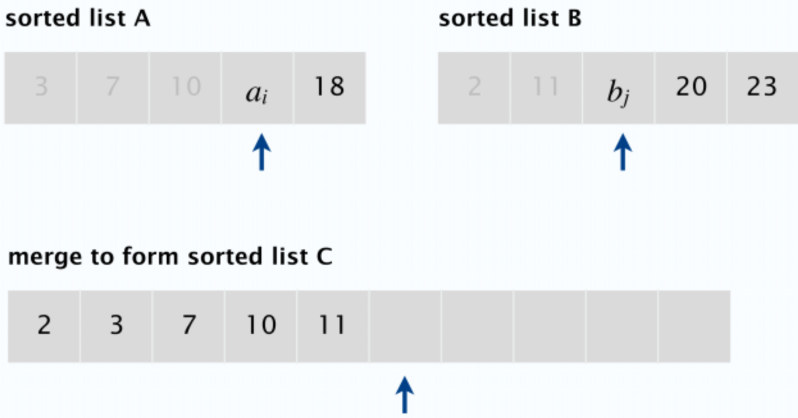
merge results

A	G	H	I	L	M	O	R	S	T

Merging

Goal. Combine two sorted lists A and B into a sorted whole C.

- Scan A and B from left to right.
- Compare a_i and b_j.
- If $a_i \leq b_j$, append a_i to C (no larger than any remaining element in B).
- If $a_i > b_j$, append b_j to C (smaller than every remaining element in A).



Mergesort implementation

Input. List L of n elements from a totally ordered universe.

Output. The n elements in ascending order.

```
MERGE-SORT(L)
_______________________________________
IF (list L has one element)
    RETURN L.

Divide the list into two halves A and B.
A ← MERGE-SORT(A).    ←——— T(n / 2)
B ← MERGE-SORT(B).    ←——— T(n / 2)
L ← MERGE(A, B).      ←——— Θ(n)
RETURN L.
_______________________________________
```

A useful recurrence relation

Def. $T(n)$ = max number of compares to mergesort a list of length n.

Recurrence.

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n & \text{if } n > 1 \end{cases}$$

between $\lfloor n / 2 \rfloor$ and $n - 1$ compares

Solution. $T(n)$ is $O(n \log_2 n)$.

Assorted proofs. We describe several ways to solve this recurrence. Initially we assume n is a power of 2 and replace ≤ with = in the recurrence.

9

Divide-and-conquer recurrence: recursion tree

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$$