Objectives

Greedy Algorithms Wrapup

- Data Compression: Huffman Codes
- Clustering

Problem: Encoding Symbols

Computers use bits: 0s and 1s

Need to represent what we (humans) know as 0s and 1s

- Map symbol to unique sequence of 0s and 1s
- Process is called *encoding*

Fundamental problem for data compression: represent data as compactly as possible

Goal. Optimal encoding that takes advantage of *nonuniformity* of letter frequencies

Prefix Codes

Problem: Encoding of one character is a *prefix* of encoding of another

Solution: **Prefix Codes**: map letters to bit strings such that no encoding is a prefix of any other

- Won't need artificial devices like spaces to separate characters

Optimal Prefix Codes

Goal: minimize **Average number of Bits per Letter (ABL):**

$\Sigma_{x \in S}$frequency of x * length of encoding of x

For all characters in our alphabet

f_x: frequency that letter x occurs

$\gamma(x)$: encoding of x

- $|\gamma(x)|$: length of encoding of x

Minimize **ABL =** $\boxed{\Sigma_{x \in S} f_x \, |\gamma(x)|}$

Problem Statement

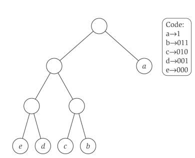
Given an alphabet and a set of frequencies for the letters, produce optimal (most efficient) prefix code

- ➢ Minimizes average number of bits per letter

Binary Trees to Represent Prefix Codes

Exposes structure better than list of mappings

- Each leaf node is a letter
- Follow path to the letter
  - –Going left: 0
  - –Going right: 1

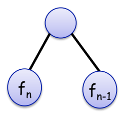


Combining Our Conclusions

The binary tree corresponding to the optimal prefix code is *full*, i.e., each internal node has two children

We want to label the leaf nodes of the binary tree corresponding to the optimal prefix code such that nodes with *greatest* depth have *least* frequency

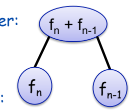
➜ Bottom of the tree:



How Can We Use This?

Two letters with least frequency are definitely going to be siblings

- Tie them together
- Their parent is a "meta-letter"
  - Frequency is sum of $f_n + f_{n-1}$



Constructing an Optimal Prefix Code

Huffman's Algorithm:

```
To construct a prefix code for an alphabet S, with given frequencies:
  If S has two letters then
    Encode one letter using 0 and the other letter using 1
  Else
    Let y* and z* be the two lowest-frequency letters
    Form a new alphabet S' by deleting y* and z* and
      replacing them with a new letter ω of frequency f_y* + f_z*
    Recursively construct a prefix code γ' for S', with tree T'
    Define a prefix code for S as follows:
      Start with T'
      Take the leaf labeled ω and add two children below it
        labeled y* and z*
  Endif
```

Reduce

Build up

Replace lowest-freq letters with meta letter

Alternative Description

Create a leaf node for each symbol, labeled by its frequency, and add to a queue

While there is more than one node in the queue

- Remove the two nodes of lowest frequency
- Create a new internal node with these two nodes as children and with frequency equal to the sum of the two nodes' probabilities
- Add the new node to the queue

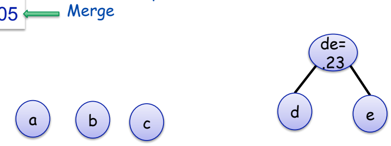
The remaining node is the tree's root node

Creating the Optimal Prefix Code

$f_a = .32$
$f_b = .25$
$f_c = .20$
$f_d = .18$
$f_e = .05$

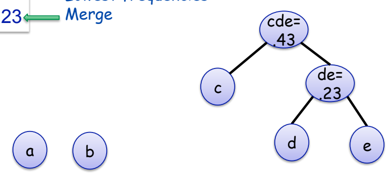
Creating the Optimal Prefix Code

$f_a = .32$
$f_b = .25$
$f_c = .20$
$f_d = .18$ Lowest frequencies
$f_e = .05$ Merge



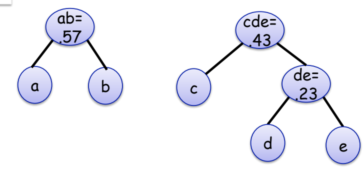
Creating the Optimal Prefix Code

$f_a = .32$
$f_b = .25$
$f_c = .20$ Lowest frequencies
$f_{de} = .23$ Merge



Creating the Optimal Prefix Code

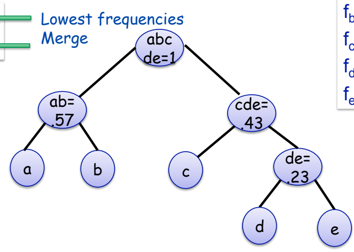
$f_a = .32$ Lowest frequencies
$f_b = .25$ Merge
$f_{cde} = .43$



Creating the Optimal Prefix Code

$f_{ab} = .57$ Lowest frequencies
$f_{cde} = .43$ Merge

$f_a = .32$
$f_b = .25$
$f_c = .20$
$f_d = .18$
$f_e = .05$

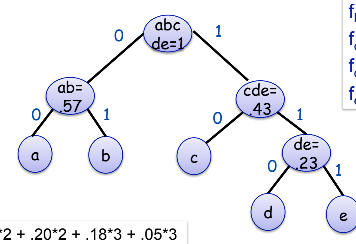


What are the resulting encodings?
What is the ABL?

Creating the Optimal Prefix Code

```
a: 00
b: 01
c: 10
d: 110
e: 111
```

$f_a = .32$
$f_b = .25$
$f_c = .20$
$f_d = .18$
$f_e = .05$



ABL=.32*2 + .25*2 + .20*2 + .18*3 + .05*3
= .64 + .5 + .4 + .54 + .15
= 2.23

Building to Solution

We built tree bottom up

May have thought top down would work better

- See book for discussion

Implementation

What are the data structures we need?

- Binary tree for the prefix codes
- Priority queue for choosing the node with lowest frequency

Where are the costs?

Running Time

Costs

- Inserting and extracting node into PQ: O(log n)
- Number of insertions and extractions: O(n)

➔ O(n log n)

Analysis of Algorithm's Optimality

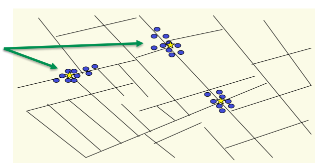
2 page proof in book

Real-life Compression

Text can be compressed well because of known frequencies

- Algorithms can be optimized to languages
- More than just "z doesn't happen very often"
  - "z doesn't happen after q"

Intersections with polluted wells



Outbreak of cholera deaths in London in 1850s.
Reference: Nina Mishra, HP Labs

CLUSTERING

Review: Clustering

Given a set *U* of *n* objects labeled $p_1, \ldots, p_n$, classify into coherent groups

- Example objects: photos, documents, micro-organisms

Distance function. Numeric value specifying "closeness" of two objects

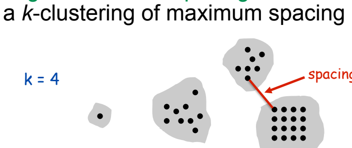
- Assume it satisfies several natural properties:
  - $d(p_i, p_j) = 0$ iff $p_i = p_j$ (identity of indiscernibles)
  - $d(p_i, p_j) \geq 0$ (nonnegativity)
  - $d(p_i, p_j) = d(p_j, p_i)$ (symmetry)

Problem: Clustering of Maximum Spacing

k-clustering. Divide objects into *k* non-empty groups

Spacing. Min distance between any pair of points in different clusters

Clustering of maximum spacing. Given an integer *k*, find a *k*-clustering of maximum spacing



Our Proposed Solution

Start with each node in its own cluster

Sort edges by their distance, ascending

For each edge, combine its nodes' clusters into one cluster until we have *k* clusters

Greedy Clustering Algorithm

Single-link *k*-clustering algorithm

- Form a graph on the vertex set *U*, corresponding to *n* clusters
- Find the closest pair of objects such that *each object is in a different cluster*, and add an edge between them
- Repeat *n-k* times until there are exactly *k* clusters

How relates to our algorithm?

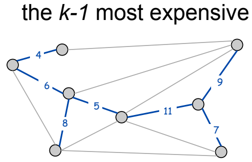
How is this related to the MST?

Greedy Clustering Algorithm

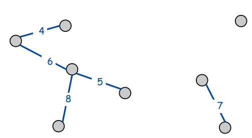
Key observation. Same as Kruskal's algorithm

- Except we stop when there are *k* connected components

Remark. Equivalent to finding an MST and deleting the *k-1* most expensive edges



MST



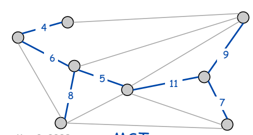
K=3

Greedy Clustering Algorithm: Analysis

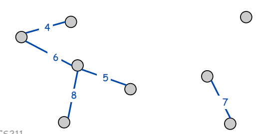
Theorem. Let C denote the clustering $C_1, \ldots, C_k$ formed by deleting the *k-1* most expensive edges of a MST. C is a *k*-clustering of *max spacing*.

Pf Idea.

- What can we say about C's spacing?
  - Within clusters and between clusters
- What if C isn't optimal?
  - What does that mean about C's clusters vs (optimal) C*'s clusters?



MST



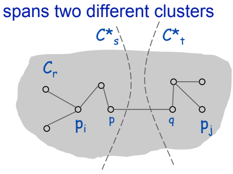
K=3

Greedy Clustering Algorithm: Analysis

Theorem. Let C denote the clustering $C_1, \ldots, C_k$ formed by deleting the *k-1* most expensive edges of a MST. C is a *k*-clustering of *max spacing*.

Pf Sketch. Let C* denote some other clustering $C^*_1, \ldots, C^*_k$

- The spacing of *C* is length *d* of (k-1)st most expensive edge
- Let p_i, p_j be in the same cluster in *C* (say C_r) but different clusters in C*, say C^*_s and C^*_t
- Some edge (*p*, *q*) on p_i-p_j path in C_r spans two different clusters in C*
- What do we know about *(p, q)*?

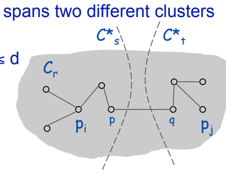


Greedy Clustering Algorithm: Analysis

Theorem. Let C denote the clustering $C_1, \ldots, C_k$ formed by deleting the *k-1* most expensive edges of a MST. C is a *k*-clustering of *max spacing*.

Pf. Let C* denote some other clustering $C^*_1, \ldots, C^*_k$

- The spacing of *C* is length *d* of (k-1)st most expensive edge
- Let p_i, p_j be in the same cluster in *C* (say C_r) but different clusters in C*, say C^*_s and C^*_t
- Some edge (*p*, *q*) on p_i-p_j path in C_r spans two different clusters in C*
- All edges on p_i-p_j path have length ≤ d since Kruskal chose them
- Spacing of C is ≤ d since *p* and *q* are in different clusters



The Plan

Tue-Fri: Open-book midterm

- I'll be at a conference Tuesday through Saturday
  - Available by email

Next Monday

- Helping the registrar assign students to courses

Problem Set 3

Much better!

Common problems

- Not showing sufficient amount of work
  - E.g., Make sure you're applying Prim's algorithm rather than Kruskal's
- Explain what you're trying to prove
  - Introduces your variables/notation
- Say what proof technique you're going to use
  - E.g., proof by induction
  - Give reader a head's up of approach
- Notation confusion
  - If I misunderstood what you were saying because of your notation, come talk to me. Possibility to get partial credit

Midterm Expectations

Covers chapters 1—4 of book

Similar to problem set

Turned into my mailbox in CS office by Friday or under my office door

For each problem

- Clear description of solution
  - Reference similar problems/solving technique
  - Use algorithms terminology
- State and explain running time
- State proof technique
- State intuition, show work when appropriate