Some greedy algorithms you have seen before.

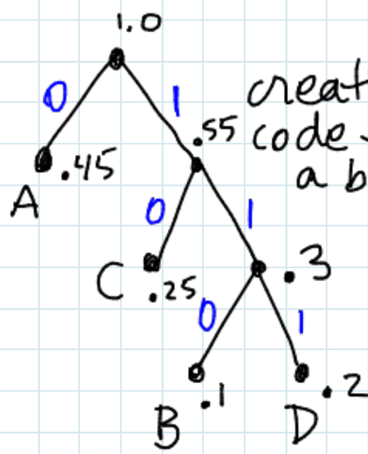
Huffman coding from CS 240

Given symbols (e.g. a, b, c ·· z) each with a frequency (e.g. e frequent, q infrequent) encode them in binary so that

- encodings are short — so use short string for e, longer for q
- can decode — so use a <u>prefix</u> code — no code is a prefix of another.

Example

	<u>freq.</u>	<u>code</u>
A	.45	0
B	.1	110
C	.25	10
D	.2	111



creating the code from leaves of a binary tree ensures the prefix property

To decode:

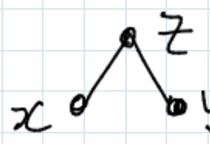
$\underbrace{0}_{A}\,\underbrace{10}_{C}\,\underbrace{111}_{D}\,\underbrace{0}_{A}$

Average code length:

$$.35(1) + .1(3) + .25(2) + .2(3) = \sum_{\substack{\text{internal} \\ \text{nodes } z}} f(z) = .3 + .55 + 1.0$$

Greedy Algorithm

- Let x, y be two least frequent letters
- construct



new letter z

$freq(z) = freq(x) + freq(y)$

- Remove x, y. Add z. Repeat

You saw a proof that this greedy algorithm gives the prefix code that minimizes average code length.

Idea of proof: an exchange proof that if x or y is not a child of the lowest internal node, we could swap and average code length does not go up.

Thus

any solution $\xrightarrow[\text{swap}]{}$ . . . $\xrightarrow[\text{swap}]{}$ greedy solution

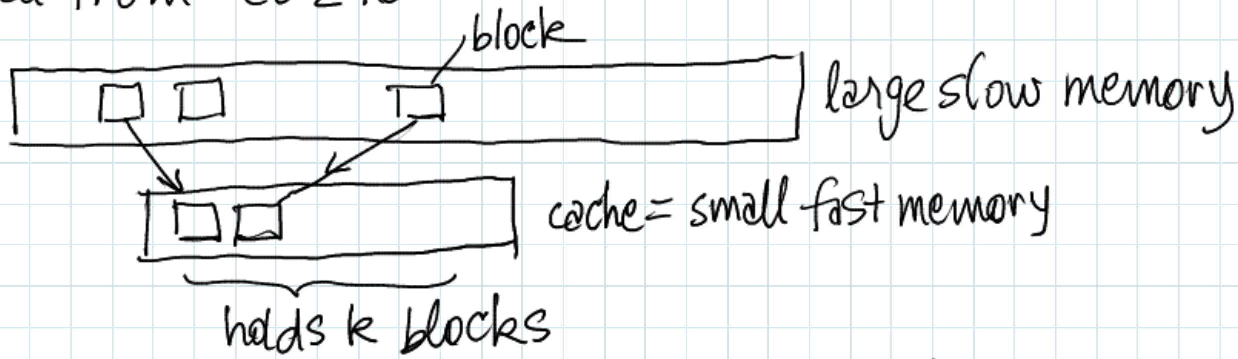
and avg. code length does not go up.

So greedy solution has smallest avg. code length.

Invented by David Huffman as PhD student at MIT. Also known for mathematics of origami.

Another example of greedy algorithm: optimal caching

Recall from CS 240



When the cache is full and we want a new block, we must <u>evict</u> some block. Which one?

Goal: minimize the total number of evictions over the sequence of block requests. (the future sequence is hidden)

e.g. requests 1, 2, 3, 1 with $k=2$

request	cache	cost
1	1 $\emptyset$	
2	1 2	
3	3 2	1
1	1 2	1

total = 2

versus same

3	1 3	1
1	1 3	0

total = 1

Block eviction strategies

- Least Recently Used (LRU) evict the block that was last accessed longest ago
- Least Frequently Used (LFU) evict the block that's been accessed least often.
- First In, First Out (FIFO) just use a queue

In fact LRU is better. Better than LFU in a theoretical sense (competitive analysis). Better than FIFO in practice (though equal in comp. analysis)

To compare some strategy to optimum (with knowledge of the future) we need to compute the opt.

Use this greedy strategy

- evict the block that is next accessed furthest in the future.

EX. Prove that this works - use exchange proof. (It's tricky.)

LRU pretends that the future looks like the past (reversed)

Knapsack Problem

You're going on a 5 day canoing trip to Algonquin Park. You want to pack your knapsack to maximize value and minimize weight.

Given n items, item i has weight w_i and value v_i.
Weight limit of knapsack is W. Put items in knapsack, sum of weights $\leq W$, maximize sum of values.

[Notation: find $S \subseteq \{1, \dots, n\}$, $\Sigma \{w_i : i \in S\} \leq W$ and maximize $\Sigma \{v_i : i \in S\}$]

Two versions of the problem:

- 0-1 knapsack. Items are indivisible (tent, flashlight)
- fractional knapsack. Can use fractions of items (oatmeal, cheese)

Question: Do you think it's ever good to use fractions?

We'll see a dynamic programming algorithm for 0-1 knapsack, but (in some sense) the alg. is not efficient and the problem is hard.

Today: a greedy algorithm for the fractional knapsack

Example:

$W = 6$

i	v_i	w_i	v_i/w_i
1	12	4	3
2	7	3	$2\frac{1}{3}$
3	6	3	2

Note: it makes sense to order items by <u>value per weight</u>.

For the 0-1 case, greedy gives item 1, value 12 (nothing else fits) but taking items 2 and 3 gives value 13

For fractional case, greedy takes item 1, leaving weight of 2 free, so take 2/3 of item 2. Value: $12 + \frac{2}{3} \cdot 7$.

Greedy Algorithm

x_i – weight of item i that we take

free-W $\leftarrow$ W

for $i = 1 \dots n$ (items ordered by v_i / w_i)

  $x_i \leftarrow \min \{ w_i, \text{free-W} \}$

  free-W $\leftarrow$ free-W $- x_i$

Note that the solution will look like:

$$
\begin{array}{lcccc}
\text{item} & 1 \; 2 \; \cdots & j & j{+}1 \; \cdots \; n \\
x_i & \underbrace{\begin{matrix} x_1 & x_2 & \cdots \\ \| & \| & \\ w_1 & w_2 & \cdots \end{matrix}}_{\text{use all of items } 1 \cdots j-1} & x_j & \underbrace{0 \; \cdots \; 0}_{\text{use none of items } j+1 \cdots n}
\end{array}
$$

$\uparrow$ use fraction of item j $0 < x_j < w_j$

Final weight: $\sum x_i = W$ (if $\sum w_i \geq W$)

Final value: $\sum \left(\frac{v_i}{w_i} \right) x_i$

Running time $O(n \log n)$ to sort by v_i / w_i

<u>Claim</u> The greedy alg. gives the opt. sol'n to the fractional knapsack problem.

<u>Proof</u> greedy solution $x_1 \; x_2 \cdots x_{k-1} \; x_k \cdots x_\ell \quad x_n$

$$\begin{array}{cccccccc} \text{greedy solution} & x_1 & x_2 & \cdots & x_{k-1} & x_k & \cdots x_\ell & x_n \\ & \shortparallel & \shortparallel & & \shortparallel & \neq \vee & \wedge & \\ \text{opt. solution} & y_1 & y_2 & \cdots & y_{k-1} & y_k & \cdot\; y_\ell & y_n \end{array}$$

Suppose y is an opt. sol'n that matches x on max # indices.

If $x = y$ done. Let k = first index where $x_k \neq y_k$.

Then $x_k > y_k$ since greedy maximizes x_k.

Since $\Sigma y = \Sigma x = W$, there is a later index $\ell > k$ with $y_\ell > x_\ell$

Exchange weight Δ of item ℓ for equal weight of item k, in opt. sol'n

$y_k' \leftarrow y_k + \Delta$

$y_\ell' \leftarrow y_\ell - \Delta$

choose Δ so $x_k = y_k'$ or $x_\ell = y_\ell'$

so $\Delta \leftarrow \min \{ y_\ell - x_\ell , x_k - y_k \}$ So $\Delta > 0$

$y_\ell - x_\ell$: amount we can move from ℓ; $x_k - y_k$: amount we can add to k.

Change in value

$$\Delta\left(\frac{v_k}{w_k}\right) - \Delta\left(\frac{v_\ell}{w_\ell}\right) = \Delta\left(\frac{v_k}{w_k} - \frac{v_\ell}{w_\ell}\right)$$

This is non-neg. because $\frac{v_k}{w_k} \geq \frac{v_\ell}{w_\ell}$ (we sorted this way)

But y was an opt. sol'n, so this can't be better.

Therefore it's a new opt. sol'n that matches x on one more index. Contra to choice of y.

We will see more greedy algs. for graph problems.