Algorithms

Greedy (pt 2)

Recap

- HW3 - due
- HW1 - So close! Come Wed...
- HW4: oral grading next Monday + Tuesday
  Sign-up for a spot on Wednesday in class!
  (Please have a group member here!)

Dynamic Programing vs Greedy

Dyn. pro: try all possibilities
↳ but intelligently!

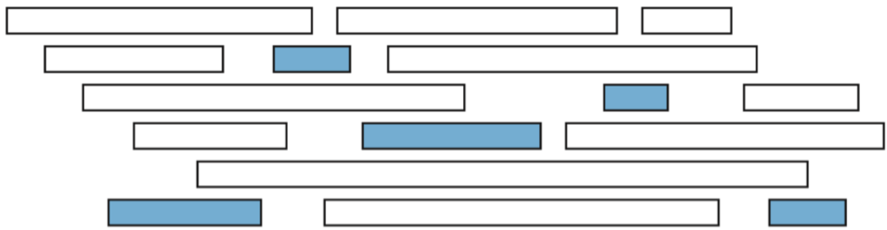
In greedy algorithms, we avoid building all possibilities.

How?
- Some part of the problem's structure lets us pick a local "best" and have it lead to a global best.

But - be careful!

Problem: Interval Scheduling

Given a set of events (intervals with a start + end time), select as many as possible so that no 2 chosen will overlap.



A maximal conflict-free schedule for a set of classes.

Key intuition:

If it finishes as early as possible, we can fit more things in!

So - strategy:

Pseudocode

```
GreedySchedule(S[1..n], F[1..n]):
    sort F and permute S to match
    count ← 1
    X[count] ← 1
    for i ← 2 to n
        if S[i] > F[X[count]]
            count ← count + 1
            X[count] ← i
    return X[1..count]
```

Runtime:

Correctnes:

Why does this work?

Note: No longer trying all possibilities or relying on optimal substructure!

So we need to be very careful on our proofs.

(Clearly, intuition can be wrong!)

Lemma: We may assume the optimal schedule includes the class that finishes first.

pf: By contradiction:

Thm: The greedy schedule is optimal.

pf: Suppose not.

Then $\exists$ an optimal schedule that has more intervals than the greedy one.

Consider first time they differ:

greedy:

$\langle g_1, g_2, g_3, \ldots, g_\ell \rangle$

optimal:

pf cont

Overall greedy strategy:

- Assume optimal is different than greedy
- Find the "first" place they differ.
- Argue that we can exchange the two without making optimal worse.

⟹ there is no "first place" where they must differ, so greedy in fact is an optimal solution.

Another example in notes: storing the most files on a tape

Intuition:

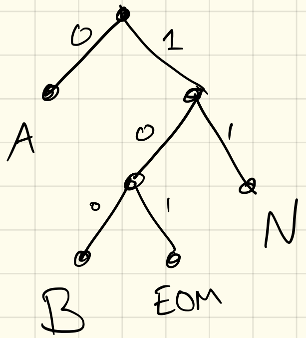
Huffman Codes - the idea:

We would like to transmit info using as few bits as possible.

What does ASCII do?

How can we do better?

Prefix-free codes



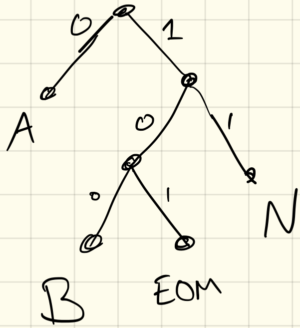
An unambiguous way to send information when we have characters not of a fixed length.

Key: No letter's code will be the prefix of another.

Encode: BAN

Decode:

1000110110101



Goal: Minimize cost

↳ here, minimize total length of encoded message:

Input: frequency counts $f[1..n]$

Compute:

$$\text{cost}(T) = \sum_{i=1}^{n} f[i] \cdot \text{depth}(i)$$

To do this, we'll need to use the array f:

This sentence contains three a's, three c's, two d's, twenty-six e's, five f's, three g's, eight h's, thirteen i's, two l's, sixteen n's, nine o's, six r's, twenty-seven s's, twenty-two t's, two u's, five v's, eight w's, four x's, five y's, and only one z.

If we ignore punctuation & spaces (just to keep it simple),

we get:

A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1

Which letters should be deeper (or shallower)?

(ie: How to be greedy?)

Huffman's alg:

Take the two least frequent characters.

Merge them into one letter, which becomes a new "leaf":

A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1

↓

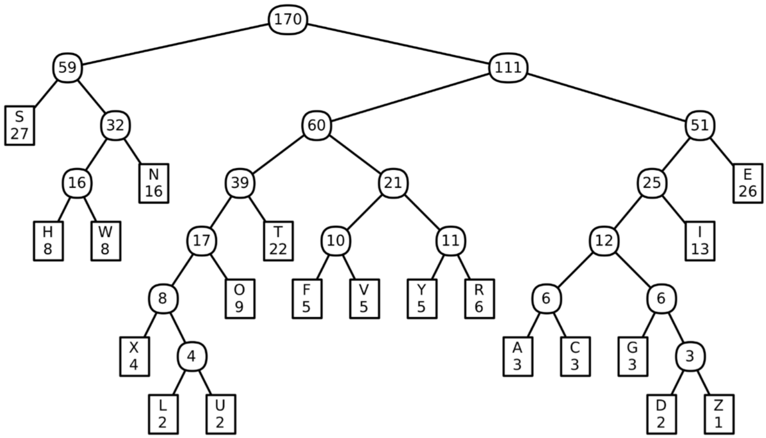
A	C	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	DZ
3	3	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	3

Example (cont):

A	C	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	DZ
3	3	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	3

The tree:

In the end, get a tree with letters at the leaves:



A Huffman code for Lee Sallows' self-descriptive sentence; the numbers are frequencies for merged characters

A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1

If we use this code, the encoded message starts like this:

1001	0100	1101	00	00	111	011	1001	111	011	110001	111	110001	10001	011	1001	110000	…
T	H	I	S	S	E	N	T	E	N	C	E	C	O	N	T	A	

How many bits?

char.	A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
freq.	3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1
depth	6	6	7	3	5	6	4	4	7	3	4	4	2	4	7	5	4	6	5	7
total	18	18	14	78	25	18	32	52	14	48	36	24	54	88	14	25	32	24	25	7

Total is $\sum f[i] \cdot depth(i)$

$= 646$ bits here

How would ASCII do on these 170 letters

Thm: Huffman codes are optimal: they use the fewest # of bits possible.

Pf: Greedy — so how to start?

Lemma: Let x & y be 2 least common characters.

There is an optimal tree in which x & y are siblings and have largest depth.

pf: Spps not:

Pf: (of thm that Huffman codes are optimal)

Induction on the # of characters

Base case:

Implementation: use priority queue

```
BuildHuffman(f[1..n]):
    for i ← 1 to n
        L[i] ← 0; R[i] ← 0
        Insert(i, f[i])

    for i ← n to 2n − 1
        x ← ExtractMin()
        y ← ExtractMin()
        f[i] ← f[x] + f[y]
        L[i] ← x; R[i] ← y
        P[x] ← i; P[y] ← i
        Insert(i, f[i])

    P[2n − 1] ← 0
```

3 arrays: L, R, P
to encode the tree

node i :