Algorithms

Greedy:
Huffman codes (cont)
Stable matchings

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.

$\Rightarrow$ 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:

Goal: Minimize cost

↳ here, minimize total length of encoded message:

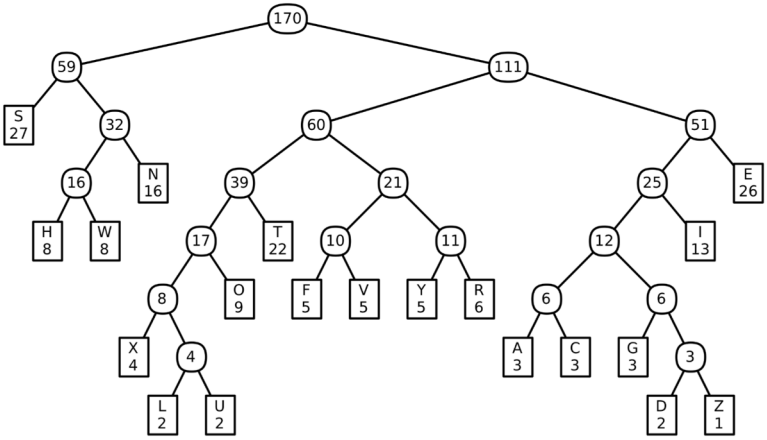
Input: frequency counts $f[1..n]$

Compute:

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

Strategy:
- Pick 2 least common letters ← make them leaves
- "Merge them": remove letters, ← add a new letter with sum of their frequencies
- Recurse!
  ↳ well, a bit imprecise..

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	

Implementation: use priority queue

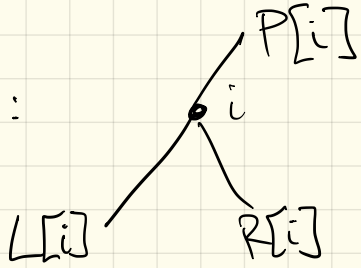
```
BUILDHUFFMAN(f[1..n]):
  for i ← 1 to n
      L[i] ← 0;  R[i] ← 0
      INSERT(i, f[i])

  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 :



So: BANANA

index:	1	2	3	4
letters:	B	A	N	EOM
freq: f:	1	3	2	1

```
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
```

L: 1 2 3 4 5 6 7

R:

P:

Runtime?

```
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
```

Correctness:

1st Lemma: There is an optimal prefix tree where the 2 least common letters are siblings + have largest depth.

pf: Spps not. Then optimal tree T has some depth d, but 2 least common letters are not at that depth.