COMP 251

Algorithms & Data Structures (Winter 2021)

Algorithm Paradigms – Greedy

School of Computer Science
McGill University

Slides of (Comp321 ,2021), Langer (2014), Kleinberg & Tardos, 2005 & Cormen et al., 2009, Jaehyun Park' slides CS 97SI, Top-coder tutorials, T-414-AFLV Course, Programming Challenges books.

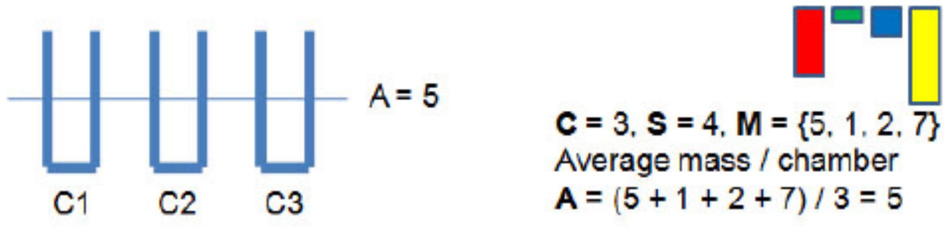
Outline

- Complete Search
- Divide and Conquer.
- Dynamic Programming.
- Greedy.
  - Introduction.
  - Examples.

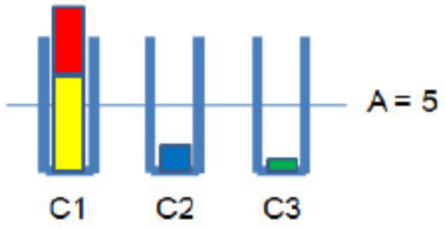
Greedy - Example

Given 1 ≤ C ≤ 5 chambers which can store 0, 1, or 2 specimens,

1 ≤ S ≤ 2C specimens, and M: a list of mass of the S specimens, determine in which chamber we should store each specimen in order to minimize $\text{IMBALANCE} = \sum_{i=1}^{C} |X_i - A|$ i.e. sum of differences between the mass in each chamber w.r.t A. where X_i is the total mass of specimens in chamber $A = (\sum_{j=1}^{S} M_j)/C,$ A is the average of all mass over C chambers



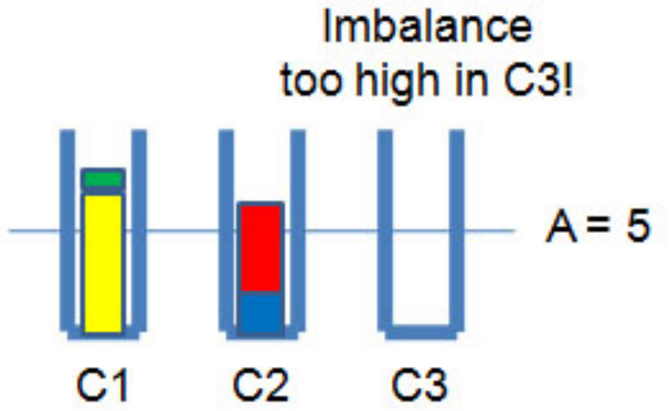
C = 3, **S** = 4, **M** = {5, 1, 2, 7}
Average mass / chamber
A = (5 + 1 + 2 + 7) / 3 = 5



IMBALANCE =
|(7+5)-5| + |2-5| + |1-5| =
7 + 3 + 4 = **14**

Greedy - Example

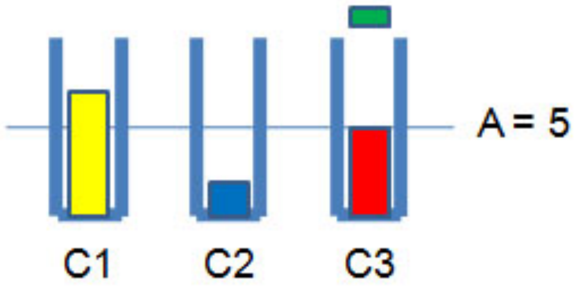
- Observations.
    - If there exists an empty chamber, at least one chamber with 2 specimens must be moved to this empty chamber! Otherwise the empty chambers contribute too much to IMBALANCE!



IMBALANCE =
|(7+1)-5| + |(2+5)-5| + |0-5| =
3 + 2 + 5 = **10**

Greedy - Example

- Observations. If S > C, then S−C specimens must be paired with one other specimen already in some chambers.



If we already assign 3 specimens to 3 chambers, the 4th specimen and beyond must be paired…

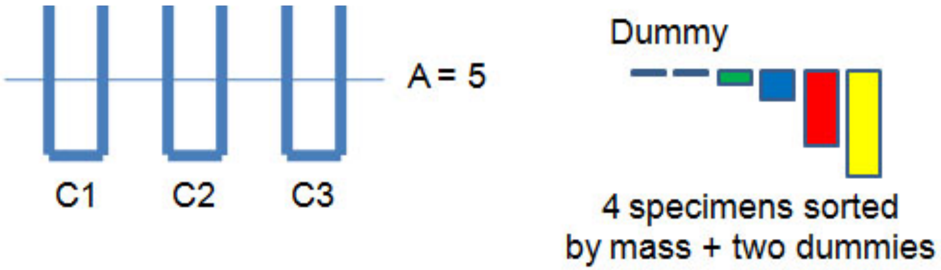
IMBALANCE =
|7-5| + |2-5| + |(5+1)-5| =
2 + 3 + 1 = 6

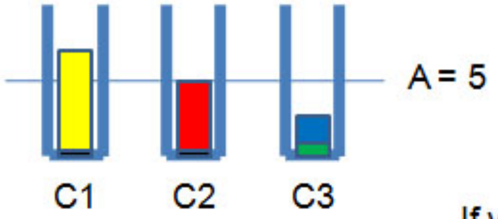
Greedy - Example

- Observations. If $S < 2C$, add dummy $2C-S$ specimens with mass 0.
  - For example, $C = 3$, $S = 4$, $M = \{5, 1, 2, 7\} \rightarrow C = 3$, $S = 6, M = \{5, 1, 2, 7, 0, 0\}$.
- Then, sort these specimens based on their mass such that $M_1 \leq M_2 \leq . . . \leq M_{2C-1} \leq M_{2C}$
  - For example, $M = \{5, 1, 2, 7, 0, 0\} \rightarrow \{0, 0, 1, 2, 5, 7\}$.

Greedy - Example

- By adding dummy specimens and then sorting them, a greedy strategy 'appears'. We can now:
  - Pair the specimens with masses M_1&M_{2C} and put them in chamber 1, then
  - Pair the specimens with masses M_2&M_{2C-1} and put them in chamber 2, and so on





IMBALANCE =
|(0+7)-5| + |(0+5)-5| + |(1+2)-5| =
2 + 0 + 2 = **4 (OPTIMAL)**

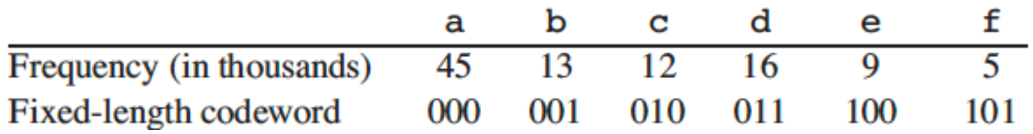
If you swap any two specimens from two different chambers, you will always have worse/equal solution

Greedy – Example – Data Compression

- We’ve seen how greedy algorithms can be used to commit to certain parts of a solution, based entirely on relatively short-sighted considerations. We will study now a problem on which a greedy rule is used, essentially, to shrink the size of the problem instance, so that an equivalent smaller problem can then be solved by recursion.
- Given a string X, efficiently encode X into a smaller string Y (Saves memory and/or bandwidth).
  - Since computers ultimately operate on sequences of bits, one needs encoding schemes that take text written in richer alphabets (such as the alphabets underpinning human languages) and converts this text into long strings of bits.

Greedy – Example – Data Compression

- Given a string X, efficiently encode X into a smaller string Y.
- A data file of 100,000 characters contains only the characters a–f, with the frequencies indicated.



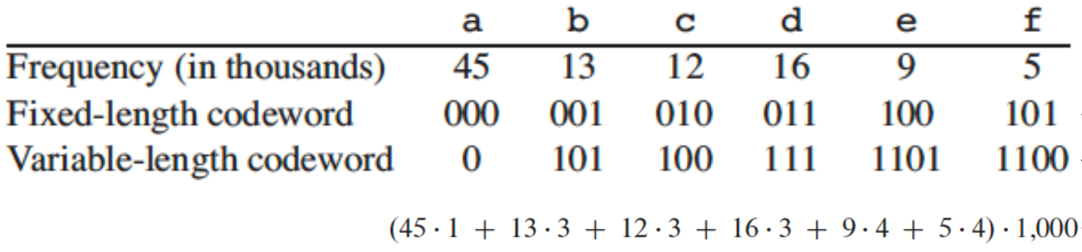
	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101

$\longrightarrow$ 300000 bits

- The simplest way to do this would be to use a fixed number of bits for each symbol in the alphabet, and then just concatenate the bit strings for each symbol to form the text.
  - The letters in most human alphabets do not get used equally frequently.
  - English: the letters e, t, a, o, i, and n get used much more frequently than q, j, x, and z (by more than an order of magnitude)

Greedy – Example – Data Compression

- Given a string X, efficiently encode X into a smaller string Y.
- A data file of 100,000 characters contains only the characters a–f, with the frequencies indicated.



	a	b	c	d	e	f	
Frequency (in thousands)	45	13	12	16	9	5	
Fixed-length codeword	000	001	010	011	100	101	$\longrightarrow$ 300000 bits
Variable-length codeword	0	101	100	111	1101	1100	$\longrightarrow$ 224000 bits

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1{,}000 \longleftarrow 224000 \text{ bits}$$

Greedy – Example – Data Compression

- Given a string X, efficiently encode X into a smaller string Y.
- A data file of 100,000 characters contains only the characters a–f, with the frequencies indicated.

	a	b	c	d	e	f	
Frequency (in thousands)	45	13	12	16	9	5	
Fixed-length codeword	000	001	010	011	100	101	$\rightarrow$ 300000 bits
Variable-length codeword	0	101	100	111	1101	1100	$\rightarrow$ 224000 bits

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1{,}000 \leftarrow 224000 \text{ bits}$$

- To give frequent characters short codewords and infrequent characters long codewords.
- To consider here only codes in which no codeword is also a prefix of some other codeword.

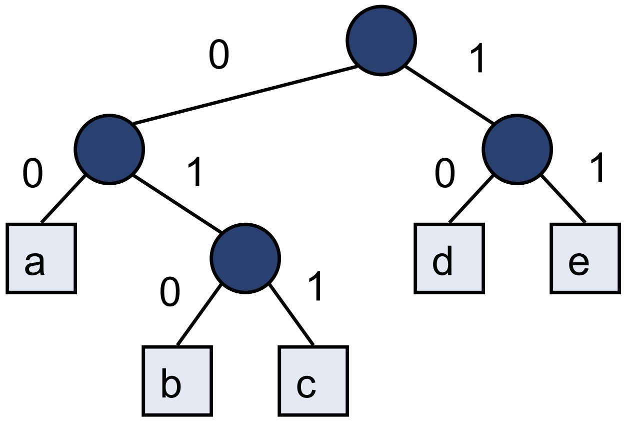
Greedy – Example – Data Compression

- A good approach: **Huffman encoding**
  - Compute frequency f(c) for each character c.
  - Encode high-frequency characters with short code words
  - No code word is a prefix for another code
  - Use an optimal encoding tree to determine the code word.
    - Any prefix-free binary code can be visualized as a binary tree with the encoded characters stored at the leaves.

Greedy – Huffman encoding

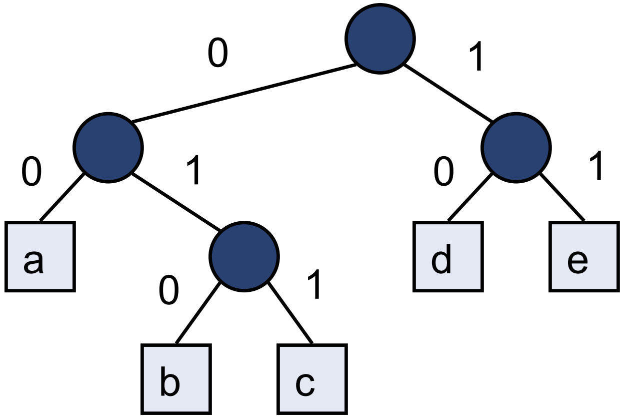
- A **code** is a mapping of each character of an alphabet to a binary code-word
- A **prefix code** is a binary code such that no code-word is the prefix of another code-word
- An **encoding tree** represents a prefix code
  - Each external node (leaf) stores a character
  - The code word of a character is given by the path from the root to the external node storing the character (0 for a left child and 1 for a right child)

00	010	011	10	11
a	b	c	d	e



Greedy – Huffman encoding

00	010	011	10	11
a	b	c	d	e



Initial string: X = acda

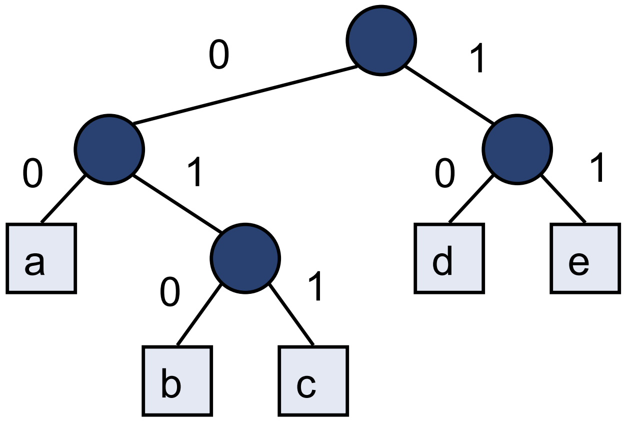
Encoded string: Y = 00 011 10 00

Greedy – Huffman encoding

- An **encoding tree** represents a prefix code
  - The **tree** is a full binary tree.
    - Every nonleaf node has two children
  - Each external node (leaf) stores a character.
    - The number of leaves is |alphabet| (one for each letter of the alphabet)
    - The number of internal nodes is |alphabet| - 1
  - The code word of a character is given by the path from the root to the external node storing the character (0 for a left child and 1 for a right child)
    - We can compute the number of bits required.

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

00	010	011	10	11
a	b	c	d	e



Greedy – Huffman encoding

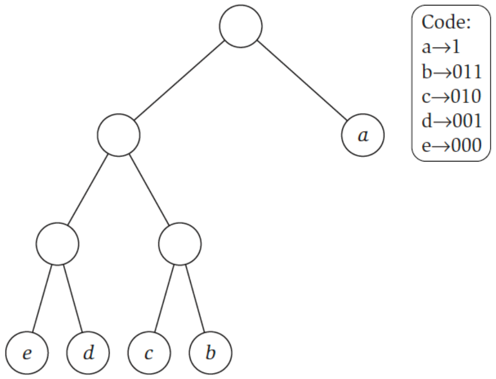
$$\sum_{i=1}^{n} f[i] \cdot depth(i)$$

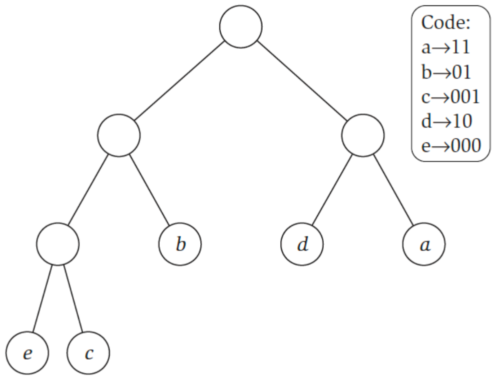
- This is exactly the same cost function we considered for optimizing binary search trees, but the optimization problem is different, because code trees are not required to keep the keys in any particular order.
- The search for an optimal prefix code can be viewed as the search for a binary tree T, together with a labeling of the leaves of T, that minimizes the number of bits used (length of the path).

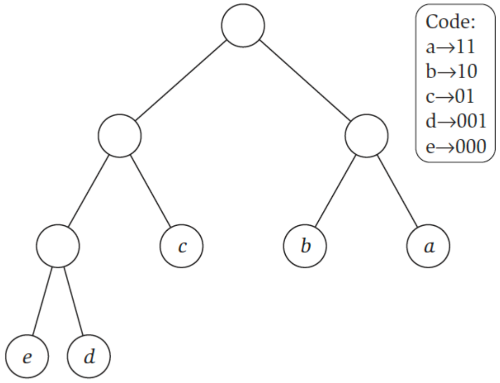
Greedy – Huffman encoding

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

- The search for an optimal prefix code can be viewed as the search for a binary tree T, together with a labeling of the leaves of T, that minimizes the number of bits used (length of the path).



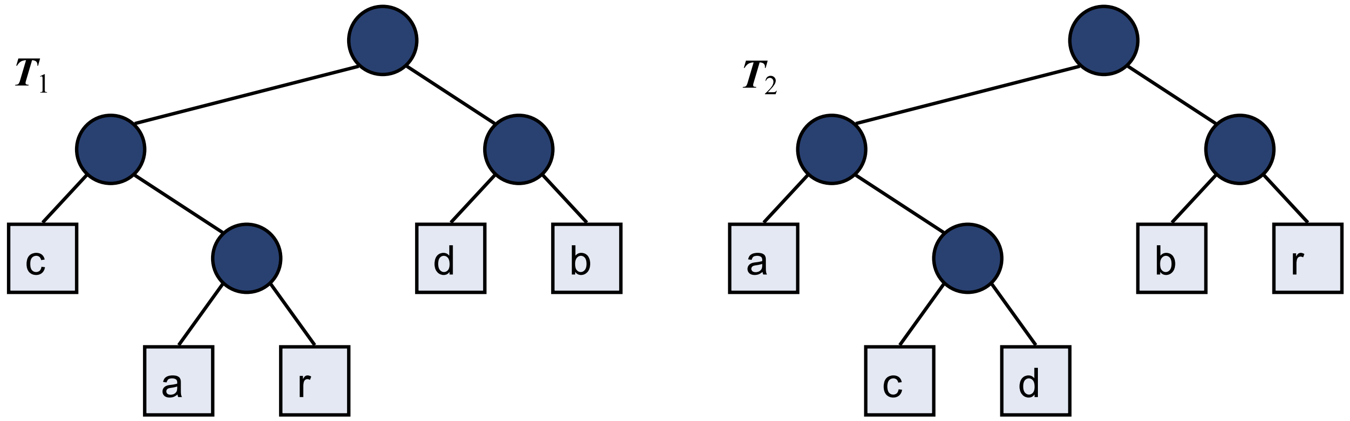




Greedy – Huffman encoding

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

- Example
  - X = abracadabra
  - T_1 encodes X into 29 bits, T_2 encodes X into 24 bits



Greedy – Huffman encoding

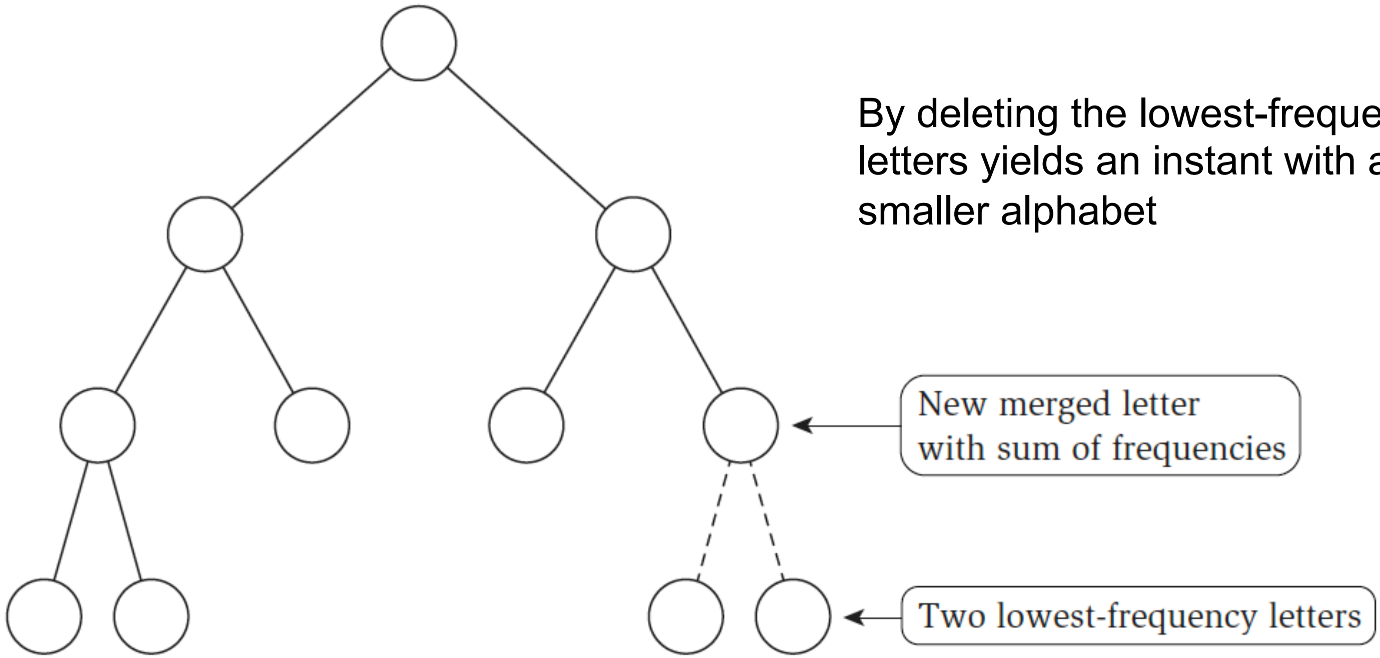
$$\sum_{i=1}^{n} f[i] \cdot depth(i)$$

- The search for an optimal prefix code can be viewed as the search for a binary tree T, together with a labeling of the leaves of T, that minimizes the number of bits used (length of the path).
  - In 1951, as a PhD student at MIT, David Huffman developed the following greedy algorithm to produce such an optimal code

HUFFMAN: Merge the two least frequent letters and recurse.

Huffman's Algorithm

- There is an optimal prefix code, with corresponding tree T*, in which the two lowest-frequency letters are assigned to leaves that are siblings in T* and have the largest depth of any leaf.


New merged letter with sum of frequencies
Two lowest-frequency letters

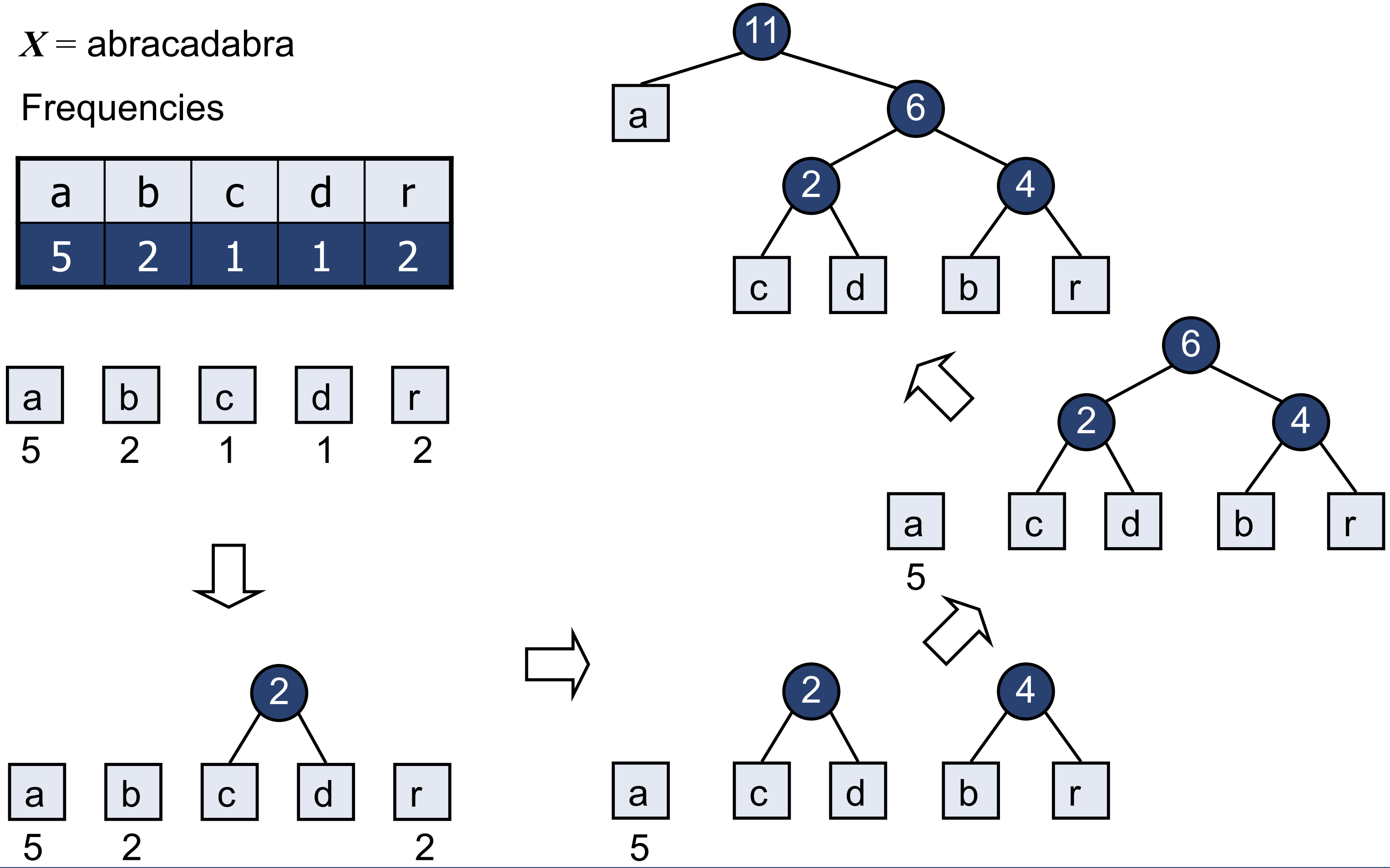
By deleting the lowest-frequency letters yields an instant with a smaller alphabet

Merge the two least frequent letters and recurse

X = abracadabra

Frequencies

a	b	c	d	r
5	2	1	1	2



Greedy – Example – Data Compression

- Suppose we want to encode the following helpfully self-descriptive sentence.

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.

discovered by Lee Sallows

- let's ignore the forty-four spaces, nineteen apostrophes, nineteen commas, three hyphens, and only one period.

```
THISSENTENCECONTAINSTHREEASTHREECSTWODSTWENTYSIXESFIVEFST
HREEGSEIGHTHSTHIRTEENISTWOLSSIXTEENNSNINEOSSIXRSTWENTYSEV
ENSSTWENTYTWOTSTWOUSFIVEVSEIGHTWSFOURXSFIVEYSANDONLYONEZ
```

- An the frequency table.

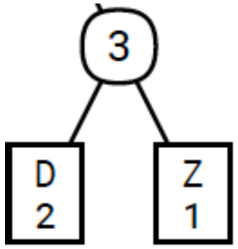
A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1

Greedy – Example – Data Compression

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

- Huffman's algorithm picks out the two least frequent letters, breaking ties arbitrarily—in this case, say, Z and D—and merges them together into a single new character DZ with frequency 3. This new character becomes an internal node in the code tree we are constructing, with Z and D as its children; it doesn't matter which child is which. The algorithm then recursively constructs a Huffman code for the new frequency table

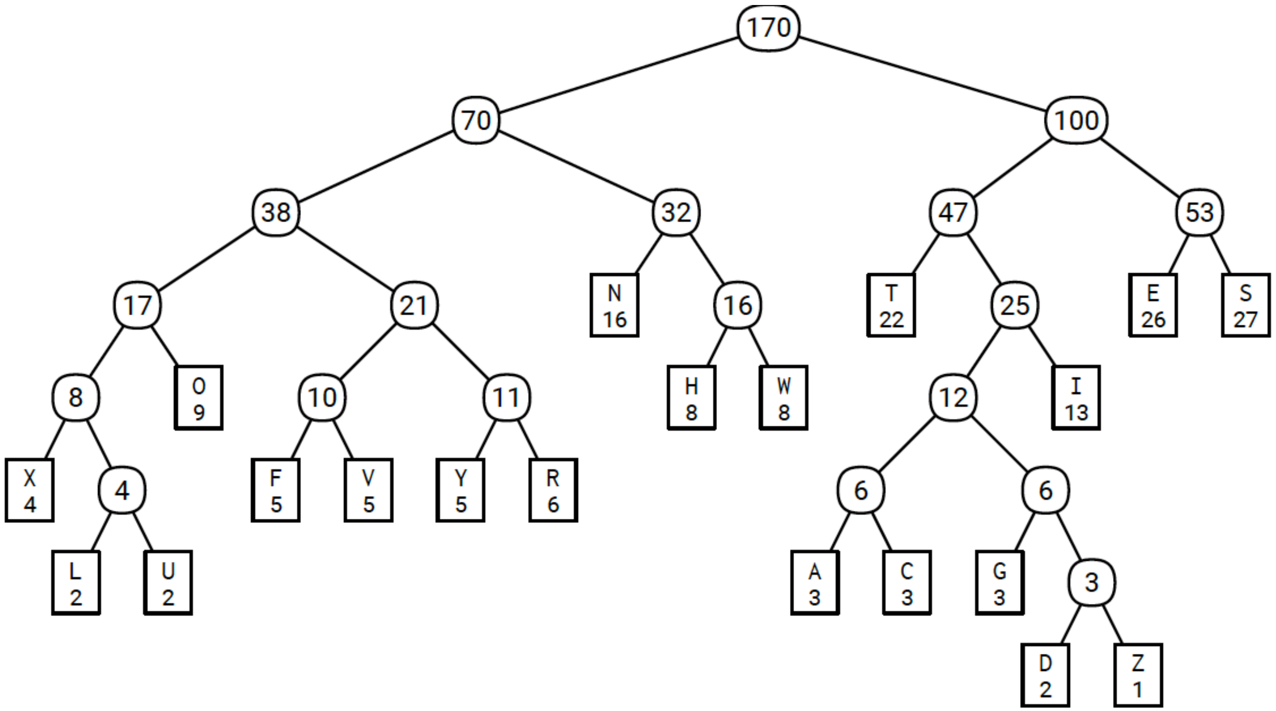
A	C	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	DZ
3	3	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	3



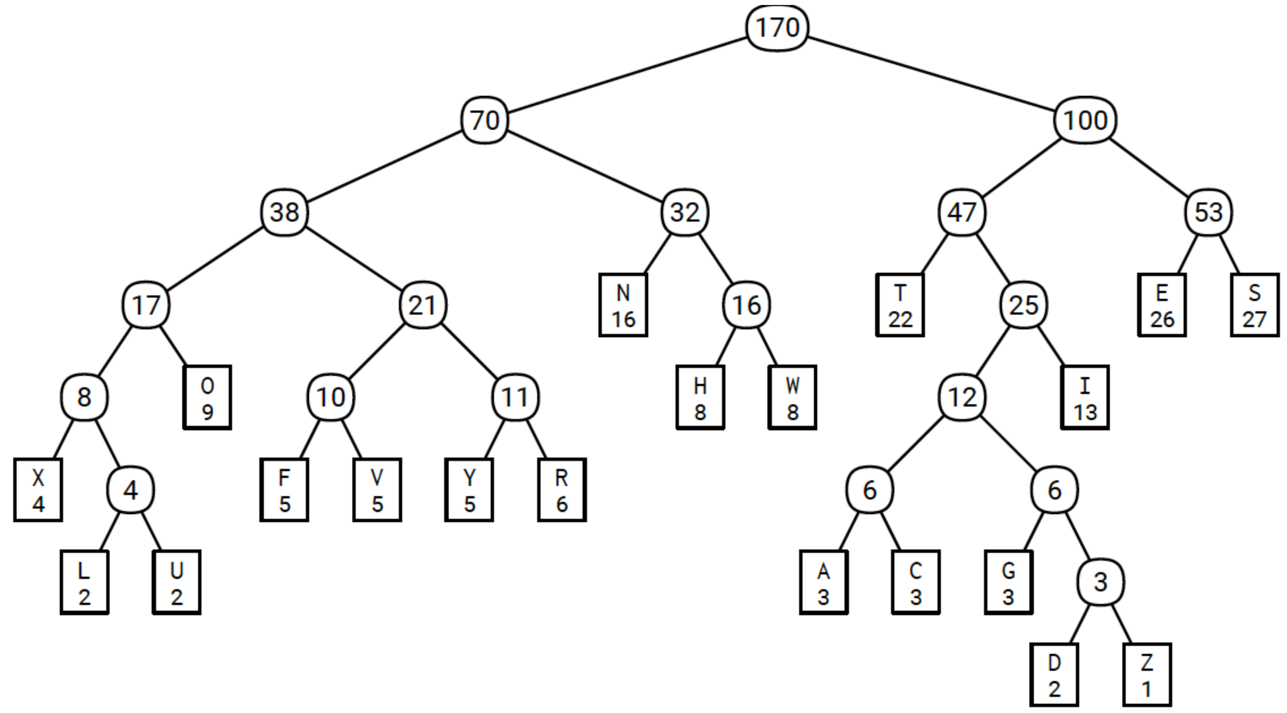
Greedy – Example – Data Compression

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

- After 19 merges, all 20 letters have been merged together.



Greedy – Example – Data Compression



100	0110	1011	111	111	110	010	100	110	010	101001	110	101001	0001	010	100	…
T	H	I	S	S	E	N	T	E	N	C	E	C	O	N	T	

Greedy – Example – Data Compression

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	6	3	4	5	3	3	6	5	4	5	5	7
total	18	18	14	78	25	18	32	52	12	48	36	30	81	66	12	25	32	20	25	7

- Altogether, the encoded message is 649 bits long.
  - Different Huffman codes encode the same characters differently, possibly with code words of different length, but the overall length of the encoded message is the same for every Huffman code: 649 bits.

Huffman's Algorithm

```
BUILDHUFFMAN($f[1..n]$):
  for $i \leftarrow 1$ to $n$
      $L[i] \leftarrow 0$;  $R[i] \leftarrow 0$          <<inserting leaves>>
      INSERT($i, f[i]$)
  for $i \leftarrow n$ to $2n-1$
      $x \leftarrow$ EXTRACTMIN( )        ⟨⟨find two rarest symbols⟩⟩
      $y \leftarrow$ EXTRACTMIN( )
      $f[i] \leftarrow f[x] + f[y]$       ⟨⟨merge into a new symbol⟩⟩
      INSERT($i, f[i]$)
      $L[i] \leftarrow x$;  $P[x] \leftarrow i$        ⟨⟨update tree pointers⟩⟩
      $R[i] \leftarrow y$;  $P[y] \leftarrow i$
  $P[2n-1] \leftarrow 0$
```

2n – 1 INSERT

2n – 2 EXTRACTMIN

If using a binary heap, each of these O(n) operations requires O(logn)

Huffman's Algorithm

- Given a string X, Huffman's algorithm construct a prefix code the minimizes the size of the encoding of X
- It runs in time $O(d + n \log n)$, where d is the size of X and n is the number of distinct characters of X
- A heap-based priority queue is used as an auxiliary structure

```
BuildHuffman($f[1..n]$):
  for $i \leftarrow 1$ to $n$
      $L[i] \leftarrow 0$;  $R[i] \leftarrow 0$
      Insert($i, f[i]$)
  for $i \leftarrow n$ to $2n-1$
      $x \leftarrow$ ExtractMin( )
      $y \leftarrow$ ExtractMin( )
      $f[i] \leftarrow f[x] + f[y]$
      Insert($i, f[i]$)
      $L[i] \leftarrow x$;  $P[x] \leftarrow i$
      $R[i] \leftarrow y$;  $P[y] \leftarrow i$
  $P[2n-1] \leftarrow 0$
```