ALGORITHMS ANS DATA STRUCTURES II

Lecture 7
Greedy Algorithms,
File compression,
Huffman codes.

Lecturer: K. Markov
markov@u-aizu.ac.jp

1/27

GREEDY ALGORITHMS

- Algorithms for optimization problems typically go through a sequence of steps, with a set of choices at each step.
- A **greedy algorithm** always makes the choice that looks best at the moment. That is, it makes a **locally optimal** choice in the hope that this choice will lead to a globally optimal solution.

2/27

GREEDY ALGORITHMS

- Examples of greedy algorithms:
  - **Prim's** algorithm (for the minimum spanning tree problem).
  - **Kruskal's** algorithm (for the minimum spanning tree problem).
  - **Dijkstra's** algorithm (for the single source shortest path problem).

3/27

GREEDY ALGORITHMS

- A greedy algorithm obtains an optimal solution to a problem by making a **sequence** of choices.
- At each decision point, the algorithm makes a choice that **seems best** at the moment.
- This strategy **does not always** produce an optimal solution.

4/27

FILE COMPRESSION

- **Objective:** conserving of space (the memory space needed for storing the file)
- **Method:** use the redundancy (save space by exploiting the fact that most files have a relatively low "information content")

5/27

RUN-LENGTH ENCODING

- The simplest type of redundancy in a file is long runs (sequences) of repeated characters (no digits are allowed) e.g.:
  AAAABBBAABBBBBCCCCCCCCDABCBAAABBBBCCCD
- Above string can be encoded as :
  4A3BAA5B8CDABCB3A4B3CD (every digit denotes the number of repetitions for the character that follows it)

6/27

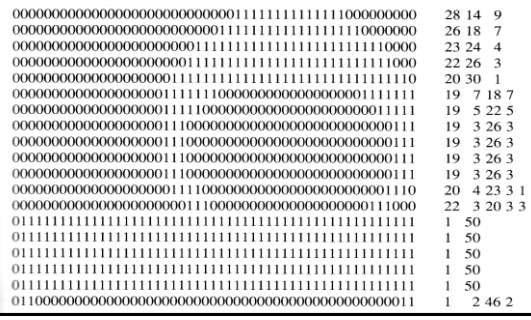
RUN-LENGTH ENCODING

- If we want to use the same character set for encoding the lengths we need an "escape character" (**Q**) which signals the beginning of a (count, character) pair
- **AAAABBBAABBBBBCCCCCCCCDABCBAAA BBBBCCCD** can be encoded as: **QDABBBAAQEBQHCDABCBAAAQDBCCCD**
- Repetitions: **D**=4,**E**=5,**F**=6,**G**=7,**H**=8… **QQ** is used for encoding a single **Q**

7/27

RUN-LENGTH ENCODING

- Bitmap encoding example



VARIABLE-LENGTH ENCODING

- Assume that the 26 letters **A, B, … Z** are assigned the 5-bits binary codes as follows:

  1. **A : 00001**, 2. **B : 00010**,
  3. **C : 00011**, …. , 26.**Z : 11010**.

- Given **ABRACADABRA**, using the 5-bits binary representation, it is encoded as the following:

9/27

VARIABLE-LENGTH ENCODING

ABRACADABRA:

00001 00010 10010 00001 00011 00001
00100 00001 00010 10010 00001

- To decode this message, read off 5 bits at a time and convert according to the above assignment.
- In the sequence, **D** appears once and **A** appears 5 times, while they are both assigned 5 bits in the binary codes.

10/27

VARIABLE-LENGTH ENCODING

- We may give an assignment like **A : 0, B : 1, R : 01, C : 10, D : 11**, and **ABRACADABRA** is encoded as
  0␣1␣01␣0␣10␣0␣11␣0␣1␣01␣0
- This uses **15** bits compared with the **55** bits before. But, this code depends on the blanks to delimit the characters. Without the blanks, the string **010101001101010** could be decoded as **RRRARBRRA** or as several other strings.

11/27

VARIABLE-LENGTH ENCODING

- How to encode the string compactly without using the delimiters?
- If the code of a character is **not** the prefix of any code of other characters, then delimiters are not needed.
  - **0**, **00**, **001**, and **0011** are prefix of **00110**.
  - In **A : 0, B : 1, R : 01, C : 10, D : 11**, the code of **A** is a prefix of **R**, the code of **B** is a prefix of **C** and **D**.

12/27

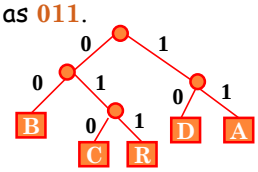
VARIABLE-LENGTH ENCODING

- A binary tree with M leaves can be used to encode any message with M different characters (one leaf holds one character).
- The code for each character is determined by the path from the root to the leaf that holds that character, with 0 for go 'left' and 1 for go 'right'.

13/27

VARIABLE-LENGTH ENCODING

- **Example:** The following binary tree encodes **A** as **11**, **B** as **00**, **C** as **010**, **D** as **10**, and **R** as **011**.



ABRACADABRA is encoded as

11000111101011101100011 11

14/27

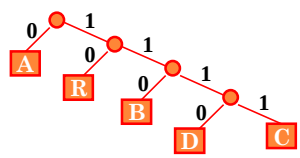
VARIABLE-LENGTH ENCODING

- The binary tree representation **guarantees** that no character code is a prefix of another, so the string is uniquely decodable from the tree.
- **Decoding:** Starting at the root, proceed down the tree according to the bits of the message: each time a leaf is encountered, output the character at that leaf and restart at the root.

15/27

VARIABLE-LENGTH ENCODING

- The following is another binary tree for encoding **ABRACADABRA**.



16/27

VARIABLE-LENGTH ENCODING

- **Question:** which binary tree is the best one to use (gives the shortest binary sequence for a given string of characters)?
- D. Huffman gave an elegant method (called **Huffman encoding**) to compute a binary tree which leads to a bit string of minimal length for any given string of characters.

17/27

HUFFMAN ENCODING

- **Algorithm:** Let **S** be the set of characters that appear in a given string.

  Step 1: Count the frequency of each character of **S** in the given string. Consider **S** as a set of roots of binary trees, each tree has one node.

  Step 2: Find two roots with the minimum frequencies and generate a new node as the root of the two roots. The new root has the frequency of the sum of frequencies of its two children. Delete the two children from **S** and add the new root to **S**.

  Step 3: If $|S| \neq 1$, go to **Step 2**.

18/27

HUFFMAN ENCODING

- Given the string **ABRACADABRA** we have:

S = {**A, B, C, D, R**},

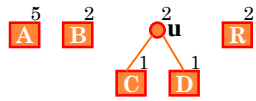
w(**A**) = 5, w(**B**) = 2, w(**C**) = 1,

w(**D**) = 1, w(**R**) = 2.

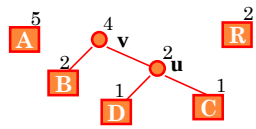
19/27

HUFFMAN ENCODING

S = {**A, B, u, R**},



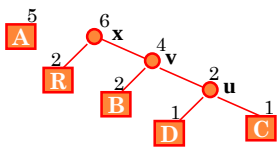
S = {**A, v, R**},



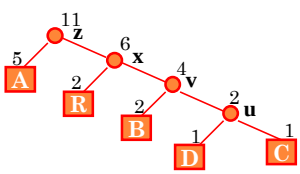
20/27

HUFFMAN ENCODING

S = {**A, x**},



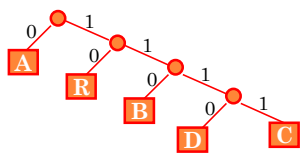
S = {**z**},



21/27

HUFFMAN ENCODING

- Encoding tree:



- We assign the binary codes to the characters of S as: **A : 0**, **B : 110**, **C : 1111**, **D : 1110**, and **R : 10**.

22/27

HUFFMAN ENCODING

- Algorithm implementation using minimum priority queue:

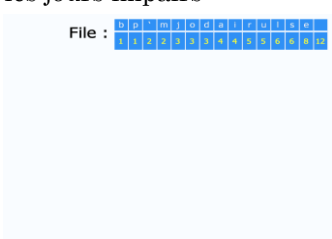
```
def HUFFMAN (S)
// Input: S – set of n letters with attribute freq giving their
// frequency.  It is also the key for Q.
   n = |S|
   Q = MIN-PRIORITY-QUEUE (S)
   for i = 1 to n -1
      allocate new node z
      z.left = x = EXTRACT-MIN (Q)
      z.right = y = EXTRACT-MIN (Q)
      z.freq = x.freq + y.freq
      INSERT (Q, z)
   return EXTRACT-MIN (Q)
```

23/27

HUFFMAN ENCODING

- Animated example:

"j'aime aller sur le bord de l'eau les jeudis ou les jours impairs"



24/27

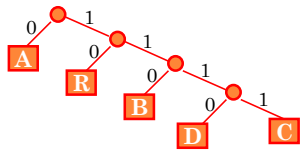
HUFFMAN ENCODING

- **Decoding**:
  - Starting with the first bit in the stream, one then uses successive bits from the stream to determine whether to go left or right in the decoding tree.
  - When we reach a leaf (character) of the tree we place that character to the output (decoded) stream.
  - Note: Compressed file must contain the table of codes.

25/27

HUFFMAN ENCODING

- **Decoding example**:



0110100111101110110100
A B RA C A D A B RA

26/27

THAT'S ALL FOR TODAY!

27/27