CS213/293 Data Structure and Algorithms 2023

Lecture 13: Data compression

Instructor: Ashutosh Gupta

IITB India

Compile date: 2023-10-17

Data compression

You must have used Zip, which reduces the space used by a file.

How does Zip work?

Fixed-length vs. Variable-length encoding

- Fixed-length encoding. Example: An 8-bit ASCII code encodes each character in a text file.
- Variable-length encoding: each character is given a different bit length encoding.
- We may save space by assigning fewer bits to the characters that occur more often.
- We may have to assign some characters more than 8-bit representation.

Example: Variable-length encoding

Example 13.1

Consider text: "agra"

- In a text file, the text will take 32 bits of space.
  - 01100001011001110111001001100001
- There are only three characters. Let us use encoding, $a =$ "0", $g =$ "10", and $r =$ "11". The text needs six bits.
  - 010110

Exercise 13.1

Are the six bits sufficient?

Commentary: If the encoding depends on the text content, we also need to record the encoding along with the text.

Example: decoding variable-length encoding

Example 13.2

Consider encoding $a =$ "0", $g =$ "10", and $r =$ "11" and the following encoding of a text.

$$101100001110$$

The text is "*graaaarg*".

We scan the encoding from the left. As soon as a match is found, we start matching the next symbol.

Example: decoding bad variable-length encoding

Example 13.3

Consider encoding $a =$ "0", $g =$ "01", *and* $r =$ "11" *and the following encoding of a text.*

$$0111000011001$$

We cannot tell if the text starts with a "g" or an "a".

Prefix condition: Encoding of a character cannot be a prefix of encoding of another character.

Encoding trie

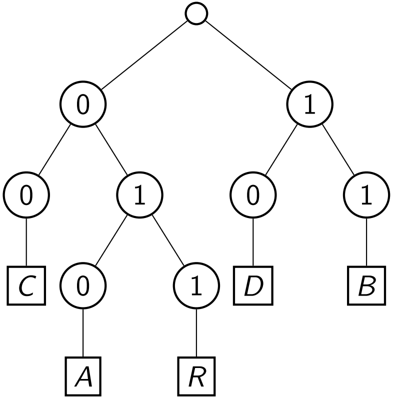
Definition 13.1

An encoding trie *is a binary trie that has the following properties.*

- *Each terminating leaf is labeled with an encoded character.*
- *The left child of a node is labeled* 0 *and the right child of a node is labeled* 1

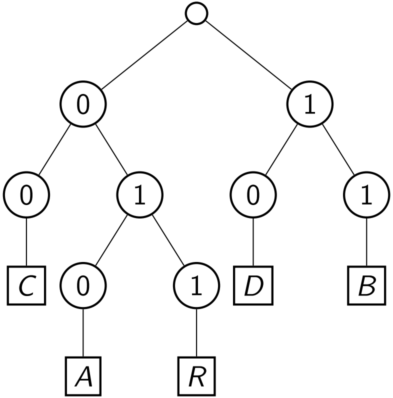
Exercise 13.2

Show: An encoding trie ensures that the prefix condition is not violated.



Character encoding/codewords:
$C = 00$, $A = 010$, $R = 011$,
$D = 10$, and $B = 11$.

Example: Decoding from a Trie



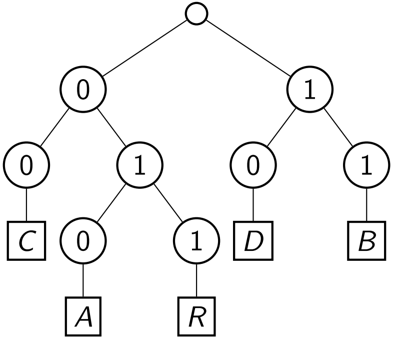
Encoding: 0101101101000010100101101101 0

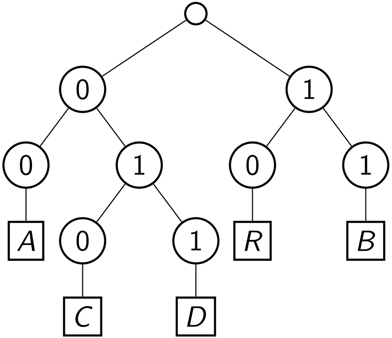
Text: ABRACADABRA

Encoding length

Example 13.4

Let us encode ABRACADABRA using the following two tries.



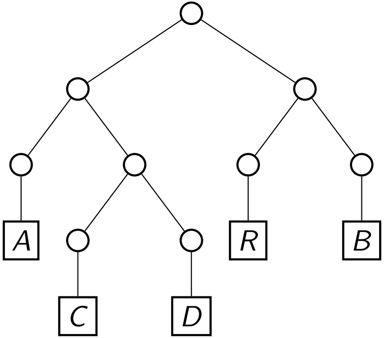


Encoding:(29 bits)
01011011010 0001010 01011011010

Encoding:(24 bits)
00111000 01000011 00111000

Drawing with tries without labels

Since we know the label of an internal node by observing that a node is a left or right child, we will not write the labels.



Commentary: We can assign any bit to a node as long as the sibling will use a different bit.

Topic 13.1

Optimal compression

Optimal compression

Different tries will result in different compression levels.

Design principle: We encode a character that occurs more often with fewer bits.

frequency

Definition 13.2

The frequency f_c of a character c in a text T is the number of times c occurs in T.

Example 13.5

The frequencies of the characters in ABRACADABRA are as follows.

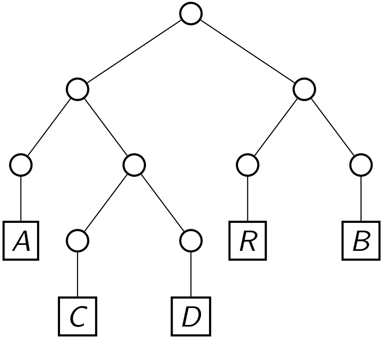
- $f_A = 5$
- $f_B = 2$
- $f_R = 2$
- $f_C = 1$
- $f_D = 1$

Characters encoding length

Definition 13.3

The encoding length l_c *of a character* c *in a trie is the number of bits needed to encode* c*.*

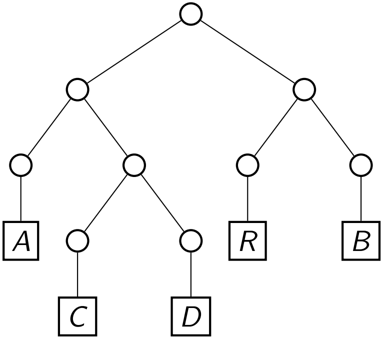
Example 13.6



In the left trie, the encoding length of the characters are as follows.

- $l_A = 2$
- $l_B = 2$
- $l_R = 2$
- $l_C = 3$
- $l_D = 3$

Weighted path length == number of encoded bits



The total number of bits needed to store a text is

$$\sum_{c \in Leaves} f_c l_c.$$

Example 13.7

The number of bits needed for ABRACADABRA using the left trie is the following sum.

$$f_A * l_A + f_C * l_C + f_D * l_D + f_R * l_R + f_B * l_B$$

$$= 5 * 2 + 1 * 3 + 1 * 3 + 2 * 2 + 2 * 2 = 24$$

Is this the best trie for compression? How can we find the best trie?

Huffman encoding

Algorithm 13.1: $\textsc{Huffman}$(Integers $f_{c_1}, ..., f_{c_k}$)

for $i \in [1, k]$ **do**

  $N := \textsc{CreateNode}(c_k, Null, Null)$;

  $T_i := \textsc{CreateNode}(f_{c_k}, N, Null)$;

return $BuildTree(T_1, ..., T_k)$

Algorithm 13.2: $\textsc{BuildTree}$(Nodes $T_1, ..., T_k$)

if $k == 1$ **then**

  return T_1

Find T_i and T_j such that $T_i.value$ and $T_j.value$ are minimum;

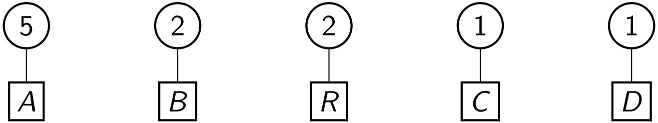
$T_i := \textsc{CreateNode}(T_i.value + T_j.value, T_i, T_j)$;

return $BuildTree(T_1, ..., T_{j-1}, T_{j+1}, ..., T_k)$

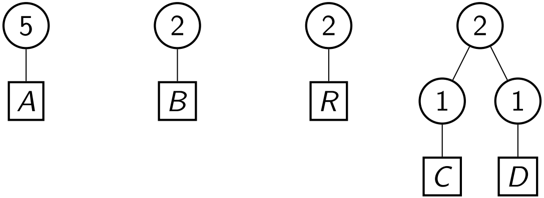
Example: Huffman encoding

Example 13.8

After initialization.

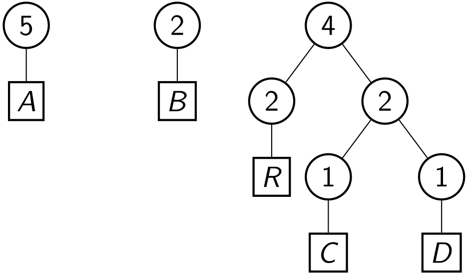


We choose nodes labeled with 1 to join and create a larger tree.

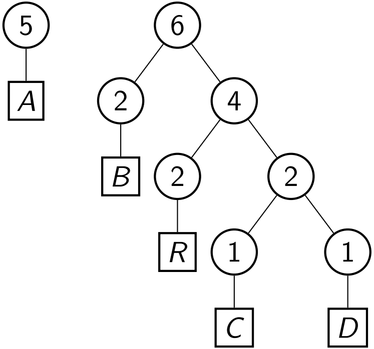


Example: Huffman encoding(2)

After the next recursive step

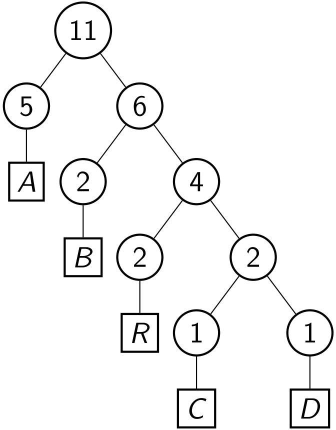


After another recursive step:

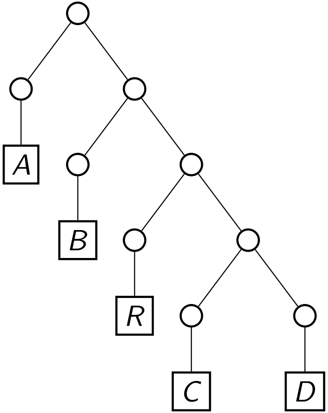


Example: Huffman encoding(3)

After the final recursive step:



We scrub the frequency labels.



Exercise 13.3

How many bits do we need to encode ABRACADABRA?

Topic 13.2

Proof of optimality of Huffman encoding

Minimum weighted path length

Definition 13.4

Given frequencies $f_{c_1}, ..., f_{c_k}$, minimum weighted path length $MWPL(f_{c_1}, ..., f_{c_k})$ is the weighted path length for the encoding trie for which the sum is minimum.

We say a trie is a witness of $MWPL(f_{c_1}, ..., f_{c_k})$ if it has the $c_1, \dots ,c_k$ are the leaves and produces encoding of length $MWPL(f_{c_1}, ..., f_{c_k})$ for a text with frequencies $f_{c_1}, ..., f_{c_k}$

Commentary: The definition of MWPL does not mention the trie. It is the property of occurrence rate distribution

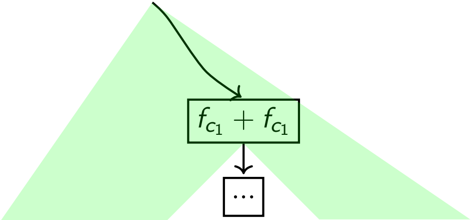
A recursive relation

Theorem 13.1

$MWPL(f_{c_1}, ..., f_{c_k}) \leq f_{c_1} + f_{c_2} + MWPL(f_{c_1} + f_{c_2}, f_{c_3}, ..., f_{c_k})$

Proof.

Let trie T be a witness of $MWPL(f_{c_1} + f_{c_2}, f_{c_3}, ..., f_{c_k})$ containing a node labeled with $f_{c_1} + f_{c_2}$ with a terminal child.

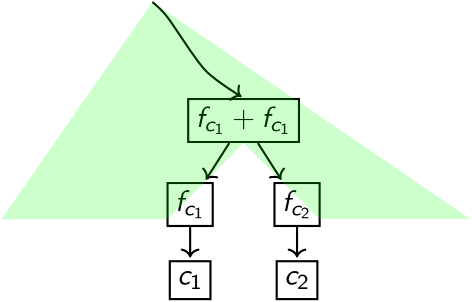


...

A recursive relation(2)

Proof(contd.)

We construct a trie for frequencies $f_{c_1}, ..., f_{c_k}$ such that the weighted path length of the trie is $f_{c_1} + f_{c_2} + MWPL(f_{c_1} + f_{c_2}, f_{c_3}, ..., f_{c_k})$.



Therefore, $MWPL(f_{c_1}, ..., f_{c_k})$ must be less than equal to the above expression. $\square$

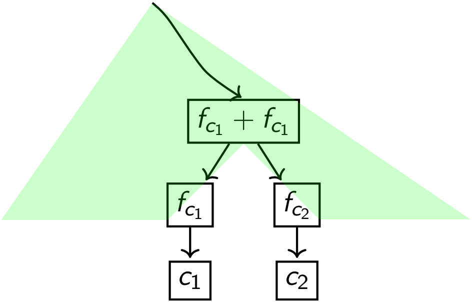
Reverse recursive relation

Theorem 13.2

If f_{c_1} and f_{c_2} are the minimum two, $MWPL(f_{c_1}, ..., f_{c_k}) = f_{c_1} + f_{c_2} + MWPL(f_{c_1} + f_{c_2}, f_{c_3}, ..., f_{c_k})$.

Proof.

There is a witness of $MWPL(f_{c_1}, ..., f_{c_k})$ where the parents of c_1 and c_2 are siblings.(Why?)

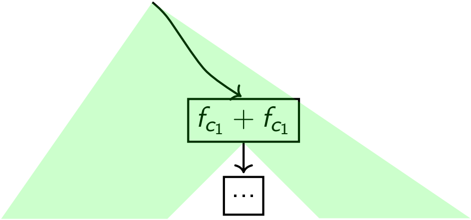


Commentary: Explaining why: Show that smallest frequency symbol can always be moved to last level to improve weighted path length. There must be a sibling at the last level. The second last frequency symbol can also be moved to the sibling to improve the weighted path length.

Reverse recursive relation(2)

Proof(contd.)

We construct a tree for frequencies $f_{c_1} + f_{c_2}, f_{c_3}, ..., f_{c_k}$ such that the weighted path length of the tree is $MWPL(f_{c_1}, ..., f_{c_k}) - f_{c_1} - f_{c_2}$.



Therefore, $MWPL(f_{c_1}, ..., f_{c_k}) - f_{c_1} - f_{c_2} \geq MWPL(f_{c_1} + f_{c_2}, f_{c_3}, ..., f_{c_k})$.

Due to the previous theorem, $MWPL(f_{c_1}, ..., f_{c_k}) = f_{c_1} + f_{c_2} + MWPL(f_{c_1} + f_{c_2}, f_{c_3}, ..., f_{c_k})$. □

Correctness of BuildTree

Commentary: In the proof, the induction step is non-trivial to understand.

Theorem 13.3

Huffman$(f_{c_1}, ..., f_{c_k})$ *always returns a tree that is a witness of* $MWPL(f_{c_1}, ..., f_{c_k})$.

Proof.

We prove this inductively.

In the call $Encode(T_1, .., T_k)$, we assume T_i is a witness of the respective $MWPL$. (For which frequencies?)

Base case:

Trivial. There is a single tree and we return the tree.

Induction step:

Since we are updating trees by combining trees with minimum weight, we have the following due to the previous theorem.

$$\underbrace{MWPL(T_1.value, ..., T_k.value)}_{\text{We will have the witness of the frequencies due to the construction.}} = T_i.value + T_j.value + \underbrace{MWPL(T_i.value + T_j.value, ...)}_{\text{witness returned due to the induction hypothesis}}$$

Practical Huffman

When we compress a file, we do not compute the frequencies for the entire file in one go.

- We compute the encoding trie of a block of bytes.
- we check if the data allows compression, if it does not we do not compress the block
- If the block is small, we use a precomputed encoding trie is used.

Exercise 13.4

How many bits are needed per character for 8 characters if frequencies are all equal?

Topic 13.3

Handling repetitions (LZ77)

Repeated string

In LZ77, if a string is repeated within the sliding window on the input stream, the repeated occurrence is replaced by a reference, which is a pair of the offset and length of the string.

The references are viewed as yet another symbols on the input stream.

Example 13.9

Before encoding ABRACADABRA using a trie, the string will be transformed to

$$ABRACAD[7,4].$$

We run Huffman on the above string.

Multiple repetitions

Example 13.10

Consider the following input text of 16 characters.

$$abababababababab$$

We will transformed the text as follows.

$$ab[2,14]$$

Topic 13.4

Deflate

DEFLATE

In addition to encoding trie, the Linux utility gzip uses the LZ77 algorithm for compression.

The combined algorithm is called DEFLATE, which compresses a file in blocks. Each block may be compresses in one of three modes.

- No compression
- Dynamically computed Huffman coding
- Fixed encoding

To compress multiple file, we need tar utility concatenate them in one file.

gzip output file format

gzip implements DEFLATE, which is a combination LZ77 and Huffman encoding.

Topic 13.5

Tutorial problems

Single-bit Huffman code

Exercise 13.5

a. In an Huffman code instance, show that if there is a character with frequency greater than $\frac{2}{5}$ *then there is a codeword of length 1.*

b. Show that if all frequencies are less than $\frac{1}{3}$ *then there is no codeword of length 1.*

Predictable text

Exercise 13.6

Suppose that there is a source that has three characters a,b,c. The output of the source cycles in the order of a,b,c followed by a again, and so on. In other words, if the last output was a b, then the next output will either be a b or a c. Each letter is equally probable. Is the Huffman code the best possible encoding? Are there any other possibilities? What would be the pros and cons of this?

Compute Huffman code tree

Exercise 13.7

Given the following frequencies, compute the Huffman code tree.

a	20
d	7
g	8
j	4
b	6
e	25
h	8
k	2
c	6
f	1
i	12
l	1

End of Lecture 13