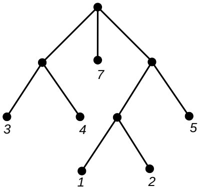
Graph and Network Theory

A rooted tree is called a **weighted tree** if each leaf is assigned to a nonnegative number called a **weight of leaf**.

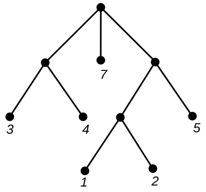


Let T be a rooted tree with t leaves with the weights $w_1, w_2, \dots, w_t$, let $l_1, l_2, \dots, l_t$ denote the levels of corresponding leaves. A **weight of T** is a value

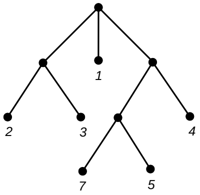
$$W(T) := \sum_{i=1}^{t} w_i l_i$$

So, for T_1 we have

$$\begin{aligned} W(T_1) &= 7 \cdot 1 + 3 \cdot 2 + 4 \cdot 2 + 5 \cdot 2 \\ &\quad + 1 \cdot 3 + 2 \cdot 3 \\ &= 7 + 6 + 8 + 10 + 3 + 6 = 40 \end{aligned}$$



$W(T_1) = 40$



$W(T_2) = 55$

The weight $W(T_1)$ is smaller than $W(T_2)$, because **heavier** leaves are closer to the root.

Let $\{w_1, w_2, ..., w_n\}$ be a list of nonnegative numbers and let T be a rooted tree with n leaves. A tree T is called **minimal (optimal) tree** with the weights $\{w_1, w_2, ..., w_n\}$, if the above weights are assigned to leaves in such a way that the number $W(T)$ is minimal.

Huffman algorithm

Inputs: L — the list of t nodes with the weights $\{w_1, ..., w_t\}$, $(t \geq 2)$
Outputs: optimal tree $T(L)$

HUFFMAN(L)

```
n ← |L|
Q ← L
for i ← 1 to n − 1
    do allocate a new node z
        z.left ← x ← EXTRACT-MIN(L)
        z.right ← y ← EXTRACT-MIN(L)
        w[z] ← w[x] + w[y]
        INSERT(L, z)
```

$$\{1, 3, 4, 6, 9, 13\}$$

Huffman algorithm - example for $L = \{1, 3, 4, 6, 9, 13\}$

- We build lists $T(L')$, which are formed from the list L by removing the two smallest weights and attaching the sum of these weights.

Huffman algorithm - example for $L = \{1, 3, 4, 6, 9, 13\}$

- We build lists $T(L')$, which are formed from the list L by removing the two smallest weights and attaching the sum of these weights.

$$L = \{1, 3, 4, 6, 9, 13\}$$

Huffman algorithm - example for $L = \{1, 3, 4, 6, 9, 13\}$

- We build lists $T(L')$, which are formed from the list L by removing the two smallest weights and attaching the sum of these weights.

$$L = \{1, 3, 4, 6, 9, 13\}$$

$$\left\{\overset{1+3}{4}, 4, 6, 9, 13\right\}$$

- We build lists $T(L')$, which are formed from the list L by removing the two smallest weights and attaching the sum of these weights.

$$
\begin{aligned}
L &= \{1, 3, 4, 6, 9, 13\} \\
& \quad \left\{ \overset{1+3}{\color{red}{4}}, {\color{red}{4}}, 6, 9, 13 \right\} \\
& \quad \left\{ \overset{\color{red}{4+4}}{8}, 6, 9, 13 \right\}
\end{aligned}
$$

- We build lists $T(L')$, which are formed from the list L by removing the two smallest weights and attaching the sum of these weights.

$$
\begin{aligned}
L &= \{1, 3, 4, 6, 9, 13\} \\
& \left\{\overset{1+3}{4}, 4, 6, 9, 13\right\} \\
& \left\{\overset{4+4}{8}, 6, 9, 13\right\} \\
& \left\{\overset{6+8}{14}, 9, 13\right\}
\end{aligned}
$$

Huffman algorithm - example for $L = \{1, 3, 4, 6, 9, 13\}$

- We build lists $T(L')$, which are formed from the list L by removing the two smallest weights and attaching the sum of these weights.

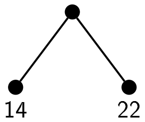
$$
\begin{aligned}
L \quad = \quad & \{1, 3, 4, 6, 9, 13\} \\
& \left\{ \overset{1+3}{4}, 4, 6, 9, 13 \right\} \\
& \left\{ \overset{4+4}{8}, 6, 9, 13 \right\} \\
& \left\{ \overset{6+8}{14}, 9, 13 \right\} \\
& \left\{ 14, \overset{9+13}{22} \right\}
\end{aligned}
$$

- We build lists $T(L')$, which are formed from the list L by removing the two smallest weights and attaching the sum of these weights.

$$
\begin{aligned}
L \quad &= \quad \{1, 3, 4, 6, 9, 13\} \\
&\left\{ \overset{1+3}{4}, 4, 6, 9, 13 \right\} \\
&\left\{ \overset{4+4}{8}, 6, 9, 13 \right\} \\
&\left\{ \overset{6+8}{14}, 9, 13 \right\} \\
&\left\{ 14, \overset{9+13}{22} \right\}
\end{aligned}
$$

- when there are **two** weights in the list, we finish calling the Huffman recursive procedures for individual lists and we build the the first tree

- We build lists $T(L')$, which are formed from the list L by removing the two smallest weights and attaching the sum of these weights.

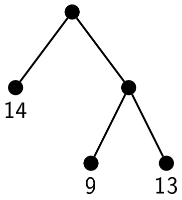


$$\begin{aligned} L &= \{1, 3, 4, 6, 9, 13\} \\ &\left\{\overset{1+3}{4}, 4, 6, 9, 13\right\} \\ &\left\{\overset{4+4}{8}, 6, 9, 13\right\} \\ &\left\{\overset{6+8}{14}, 9, 13\right\} \\ &\left\{14, \overset{9+13}{22}\right\} \end{aligned}$$

- when there are **two** weights in the list, we finish calling the Huffman recursive procedures for individual lists and we build the the first tree

- We build lists $T(L')$, which are formed from the list L by removing the two smallest weights and attaching the sum of these weights.

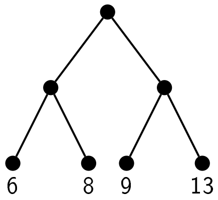
$$
\begin{aligned}
L &= \{1,3,4,6,9,13\} \\
&\left\{\overset{1+3}{4}, 4, 6, 9, 13\right\} \\
&\left\{\overset{4+4}{8}, 6, 9, 13\right\} \\
&\left\{\overset{6+8}{14}, 9, 13\right\} \\
&\left\{14, \overset{9+13}{22}\right\}
\end{aligned}
$$



- when there are **two** weights in the list, we finish calling the Huffman recursive procedures for individual lists and we build the the first tree

- We build lists $T(L')$, which are formed from the list L by removing the two smallest weights and attaching the sum of these weights.

$$
\begin{aligned}
L \quad &= \quad \{1, 3, 4, 6, 9, 13\} \\
&\quad \left\{\overset{1+3}{4}, 4, 6, 9, 13\right\} \\
&\quad \left\{\overset{4+4}{8}, 6, 9, 13\right\} \\
&\quad \left\{\overset{6+8}{14}, 9, 13\right\} \\
&\quad \left\{14, \overset{9+13}{22}\right\}
\end{aligned}
$$

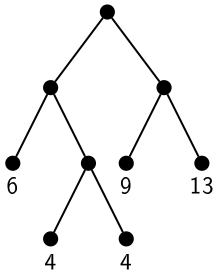


- when there are **two** weights in the list, we finish calling the Huffman recursive procedures for individual lists and we build the the first tree

Huffman algorithm - example for $L = \{1, 3, 4, 6, 9, 13\}$

- We build lists $T(L')$, which are formed from the list L by removing the two smallest weights and attaching the sum of these weights.

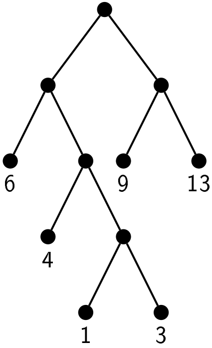
$$
\begin{aligned}
L \quad = \quad & \{1, 3, 4, 6, 9, 13\} \\
& \left\{ \overset{1+3}{4}, 4, 6, 9, 13 \right\} \\
& \left\{ \overset{4+4}{8}, 6, 9, 13 \right\} \\
& \left\{ \overset{6+8}{14}, 9, 13 \right\} \\
& \left\{ 14, \overset{9+13}{22} \right\}
\end{aligned}
$$



- when there are **two** weights in the list, we finish calling the Huffman recursive procedures for individual lists and we build the the first tree

- We build lists $T(L')$, which are formed from the list L by removing the two smallest weights and attaching the sum of these weights.

$$
\begin{aligned}
L \quad = \quad & \{1, 3, 4, 6, 9, 13\} \\
& \left\{ \overset{1+3}{4}, 4, 6, 9, 13 \right\} \\
& \left\{ \overset{4+4}{8}, 6, 9, 13 \right\} \\
& \left\{ \overset{6+8}{14}, 9, 13 \right\} \\
& \left\{ 14, \overset{9+13}{22} \right\}
\end{aligned}
$$



- when there are **two** weights in the list, we finish calling the Huffman recursive procedures for individual lists and we build the the first tree

How to code the word **ABRAKADABRA** using binary codes (each letter has its unique binary code)?

- Suppose first that we use codes with of the same length for each letter.
- We have 5 different letters (A,B,R,C,D), so we need 5 different binary codes.
- Thus the minimal length of the code will be 3 (for 2−bits binary codes we have only 2^2 different codes).
- Consequently the length of the word ABRACADABRA will be $3 \times 11 = 33$.

<table>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="3">A</td><td colspan="3">B</td><td colspan="3">R</td><td colspan="3">A</td><td colspan="3">C</td><td colspan="3">A</td><td colspan="3">D</td><td colspan="3">A</td><td colspan="3">B</td><td colspan="3">R</td><td colspan="3">A</td></tr>
</table>

Coding problem

- Too long code!
- Better idea?
- Use code of variable length! More frequent letter — shorter code!
- How to know where one code ends and where next starts when they have variable lengths?
- Solution: Prefix codes!

Prefix coding is a kind of coding of the set of words which has the *"prefix property"* i.e. there is no whole code word in the system that is a prefix (initial segment) of any other code word in the system.

Example

With binary coding the following codes

01001 1001

have the **prefix property**

01001 010

do not have the **prefix property**.

Code the word **ABRACADABRA**

Hint 1: Note that prefix codes can be generated using binary three. Codes will be connected with leaves. When you go down from the root to the left and you turn left — you have 1, otherwise — 0.

Hint 2: The length of the code is the level of the leaf.

Let n be a number of vertices in a tree.

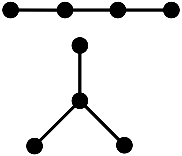
$n = 2$

1 tree

$n = 3$

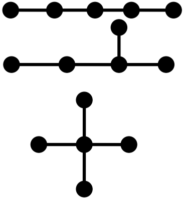
1 tree

$n = 4$



2 trees

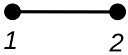
$n = 5$



3 trees

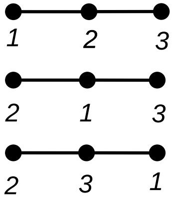
Let n be the number of vertices of a tree.

$n = 2$



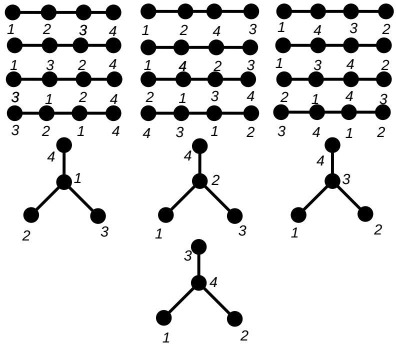
1 tree

$n = 3$



3 tree

$n = 4$



16 tree

This theorem was fromulated by Arthur Caley in XIXth century. The proof uses Prüfer's sequences (1918).

Theorem

For every positive integer n, the number of trees with n labeled vertices is n^{n-2}.

This theorem was fromulated by Arthur Caley in XIXth century. The proof uses Prüfer's sequences (1918).

Theorem

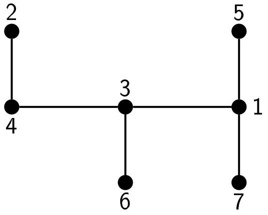
For every positive integer n, the number of trees with n labeled vertices is n^{n-2}.

Proof. We shall show bijective correspondence between the set of labeled trees with n vertices and all sequences $(a_1, a_2, \dots, a_{n-2})$, where a_i is a natural number and $1 \leq a_i \leq n$. Since the number of such sequences is n^{n-2} the proof is completed. ■

Inputs: a labeled tree T with n vertices
Output : Prüfer's code in the form of sequence $(a_1, a_2, ..., a_{n-2})$, uniquely assigned to the tree T.

```
PRÜFER'SCODE(T)
if (n = 2)
    then
        return the empty code
    else
        i ← 0
        T_0 ← T
        while (T_i ≠ K_2)
            do
                v the smallest number assigned to the leaf in tree T_i
                a_{i+1} ← a vertex incident to v in tree T_i
                T_{i+1} ← T_i − {v}
                i ← i + 1
```

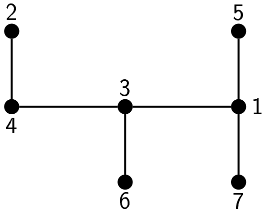
For the tree



Prüfer's code is of the form

$$(4, 3, 1, 3, 1)$$

For the tree



Prüfer's code is of the form

$$(4, 3, 1, 3, 1)$$

- Prüfer's code does not contain any leaf label .
- Each vertex v of the T tree appears in the Prüfer's code exactly $deg(v) - 1$ times.

Prüfer's algorithm

For every Prüfer's sequence it is possible to assign uniquely a tree.

Inputs:A Prüfer's sequence $d = (a_1, a_2, ..., a_{n-2})$
Outputs: A labeled tree T with n vertices.

PRÜFER'S(d)*method*

draw the set of n vertices and put the labels $v_1, v_2, ..., v_n$
$S \leftarrow \{v_1, v_2, ..., v_n\}$
$i \leftarrow 0$
$d_0 \leftarrow d$
$S_0 \leftarrow S$
while $(d_i \neq \emptyset)$
  do
    $j \leftarrow min\{k : v_k \in S_i \text{ } oraz \text{ } k \notin d_i\}$
    draw the edge $\{v_j, v_{a_{i+1}}\}$
    $d_{i+1} \leftarrow d_i - \{a_{i+1}\}$
    $S_{i+1} \leftarrow S_i - \{v_j\}$
    $i \leftarrow i + 1$
draw the edge between others points of he set S_i

Thank you for your attention!!!