Martin Ziegler

Syllabus

6. String Problems

- Recap on Strings
- Pattern Matching: *Knuth-Morris-Pratt*
- Longest Common Substring
- Edit Distance
- Context-free Parsing: *Cocke-Younger-Kasami*
- *Huffman* Compression

6. String Problems

strings recap

Specification: Fix finite alphabet $\Sigma \neq \emptyset$, often $\{0,1\}$

A string over Σ is a finite sequence $s = (s_0,\ldots s_{n-1}) \in \Sigma^*$,

input/output as array $s[0\ldots n\text{-}1]$.

Terminology: Length $|(s_0,\ldots s_{n-1})| = n$,

concatenation $s \circ t$

prefix = initial segment $(s_0,\ldots s_{n-1})_{<m} = (s_0,\ldots s_{m-1})$ for $m \leq n$.

s prefix of $t \Leftrightarrow \exists u\colon\ t = s \circ u$

s suffix of $t \Leftrightarrow \exists v\colon\ t = v \circ s$

s substring of $t \Leftrightarrow \exists u,v\colon\ t = v \circ s \circ u$

Specification (cont.): Fix finite set $V \neq \emptyset$ disjoint to Σ.

6. String Problems

Pattern Matching

Input: Two strings w and p of lengths $n = |w|$ >> $|p| = m$.

Output: Does w contain p, and where (first, all) ?

arrays $w[0\ldots n\text{-}1]$ and $p[0\ldots m\text{-}1]$

$w =$ ABCXABCDABXABCDABCDABDE

$p =$ ABCDABD

Naïve algorithm:

runtime $O(n \cdot m)$

For k:=0 to n-1

  If $w[k]$=$p[0]$ then

    Compare $w[k+1\ldots k+m-1]$ to $p[1\ldots m-1]$

    If agree, then output k.

Preprocess pattern:

$T[] =$

-1 0 0 0 -1 0 2 0

A B C D A B D

6. String Problems Knuth-Morris-Pratt

Input: Two strings w and p of lengths $n = |w|$ >> $|p| = m$.

Output: Does w contain p, and where (first, all) ?

arrays $w[0\ldots n\text{-}1]$ and $p[0\ldots m\text{-}1]$

$w =$ ABCXABCDABXABCDABCDABDE

$p =$ **ABACABABA**

KMP algorithm:

k:=0; j:=0; While $k<n$ do

If $w[k]=p[j]$ then

k++; j++;

If $j=m$ then output k-j; j:=$T[j]$; endif

else j:=$T[j]$; If $j<0$ then k++; j++; endif

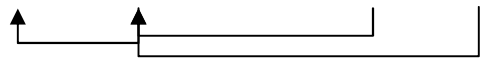
runtime $O(n+m)$

Preprocess pattern:

$T[] =$

-1 0 -1 1 -1 0 -1 3 -1 3

A B A C A B A B A



runtime $O(m)$

6. String Problems

Longest Common Substring

Specification: Fix alphabet Σ **Input:** $v \in \Sigma^n$, $w \in \Sigma^m$

Output: Length/positions of longest common substring?

Example: "ABABC" and "BABCA" share "BABC" as longest common substring

Naïve Algorithm:

Try all possible pairs of initial positions

$i=0,\ldots n\text{-}1$ and $j=0,\ldots m\text{-}1.$

For each compare $v[i,..i+k]$ to $w[j,..]$

runtime $O(n \cdot m \cdot \min(n,m))$

6. String Problems

Longest Common Substring

Specification: Fix alphabet Σ **Input:** $v \in \Sigma^n$, $w \in \Sigma^m$

Output: Length/positions of longest common substring?

Better Algorithm:

Fill integer table $LCS[0\ldots n, 0\ldots m]$,

such that $LCS[i,j]$:= length of

longest common suffix

shared by initial segments

$v[0...i-1]$ and $w[0...j-1]$

runtime $O(n \cdot m)$

$$LCS[0,j] = 0 = LCS[i,0]$$

$$LCS[i+1,j+1] = LCS[i,j]+1 \quad \text{if } v[i]=w[j]$$
$$= 0 \quad \text{if } v[i] \neq w[j]$$

Example:

		A	B	A	B
B	0	1	0	1
A	1	0	2	0
B	0	2	0	3
A	1	0	3	0

6. String Problems

Edit Distance

Specification: Fix alphabet Σ **Input:** $v \in \Sigma^n$, $w \in \Sigma^m$

Output: Min. # symbol insert/delete op.s converting v into w.

Proposition: This constitutes a metric on Σ^*. runtime $O(n \cdot m)$

Example: "**kitten**" and "**sitting**" have edit distance 5:

itten, sitten, sittn, sittin, sitting

Wagner-Fischer Algorithm: Fill table $d[0\ldots n, 0\ldots m]$

such that $d[i,j]$:= edit distance of $v[0..i\text{-}1]$ and $w[0..j\text{-}1]$

$$d[0,j] = j \qquad d[i+1,j+1] = d[i,j] \quad \text{if } v[i]=w[j]$$

$$d[i,0] = i \qquad = \min\{ d[i,j+1]+1 , d[i+1,j]+1 \} \quad \text{if } v[i]\neq w[j]$$

Variants: Dis/allow (i) replacement, (ii) transposition, (iii) …

Assign positive weights to different operations.

6. String Problems

Grammar

Specification: Fix alphabet Σ, disjoint finite set V of variables and fix a finite set R of rules as well as $S \in V$

Input: $w \in \Sigma^*$. **Output:** Can w be generated from S ?

Example: $V = \{S, X\}$, $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$

three rules $S \to \mathtt{a}XS\mathtt{c}$, $S \to \mathtt{abc}$, $X\mathtt{a} \to \mathtt{a}X$, $X\mathtt{b} \to \mathtt{bb}$

generate precisely the strings $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$, $n \in \mathbb{N}$.

Definition: A rule r is an assignment $x \to y$, where $x, y \in (\Sigma \cup V)^*$ and x contains some variable.

A rule $x \to y$ is context-free, if $x \in V$.

6. String Problems Cocke-Younger-Kasami

Specification: Fix alphabet $\sum$, disjoint finite set V of variables and fix a finite set R of *context-free* rules as well as $S \in V$

Input: $w \in \sum^*$. **Output:** Can w be generated from S ?

Rules in *Chomsky normal form*:

either (i) $X \to YZ$ (one to two variables)

or (ii) $X \to \mathtt{a}$ (one variable to one symbol)

or (iii) $S \to \varepsilon$ (empty string)

[exception only to generate ε..]

Example:
brackets

$S \to (S)\ S$

$S \to \varphi$

Definition: A rule r is an assignment $x \to y$,
where $x,y \in (\sum \cup V)^*$ and x contains some variable.

A rule $x \to y$ is context-free, if $x \in V$.

6. String Problems Cocke-Younger-Kasami

Rules of the form (i) $X \to YZ$ or (ii) $X \to \mathtt{a}$ or (iii) $S \to \varepsilon$

Table $P[s,l,X]$:= "$w_s, \ldots w_{s+l-1}$ can be generated from variable X".

Input: $w \in \sum^n$. **Output:** Can w be generated from S ?

```
Initialize P[..] with false.
For each s = 0 to n-1
  For each rule X → w_s of type (ii)
    P[s,1,X] := true
For l := 2 to n                    // Length of span
  For s := 0 to n-l                // Start of span
    For k := 1 to l-1              // Partition of span
      For each rule of type (i) X → Y Z
        if P[s,k,Y] and P[s+k,l-k,Z]
          then  P[s,l,X] := true
// w can be generated iff P[0,n,S] = true.
```

runtime $O(n^3)$

6. String Problems

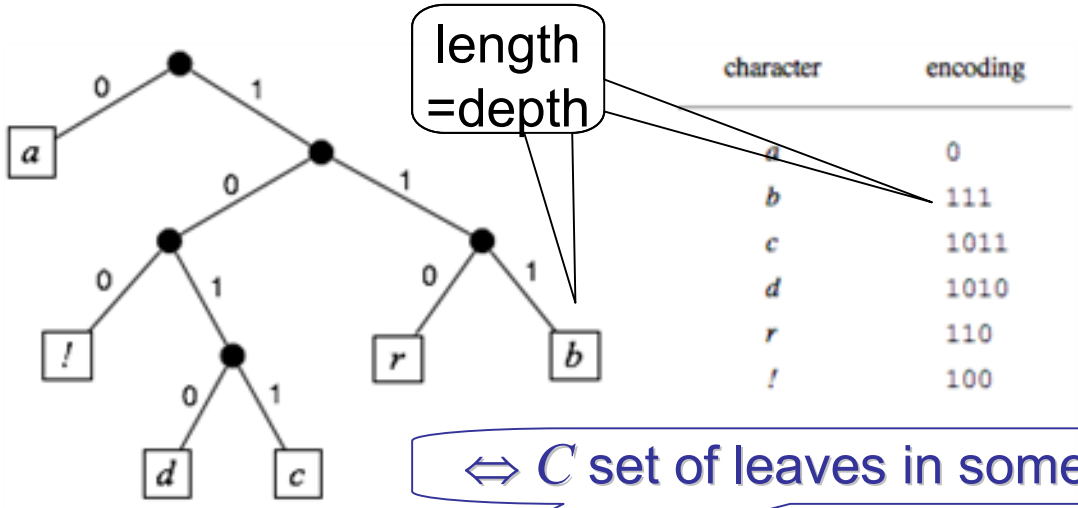
Lossless Compression

Specification: Fix alphabet $\sum$

Input: $w \in \sum^*$ **Output:** "short bit-encoding" of w

1. Determine frequencies f_s of symbols $s \in \Sigma$ in w

"this is an example of a huffman tree"



$\Leftrightarrow C$ set of leaves in some bin.tree

Variable length code, need delimiters—or better:

$C \subseteq \Gamma^*$ is *prefix-free* if $v,w \in C$ and $v \blacktriangleleft w \Rightarrow v=w$.

Char	Freq	Co
space	7	11
a	4	01
e	4	00
f	3	11
h	2	10
i	2	10
m	2	01
n	2	00
s	2	10
t	2	01
l	1	11
o	1	00
p	1	10

6. String Problems

Prefix Codes

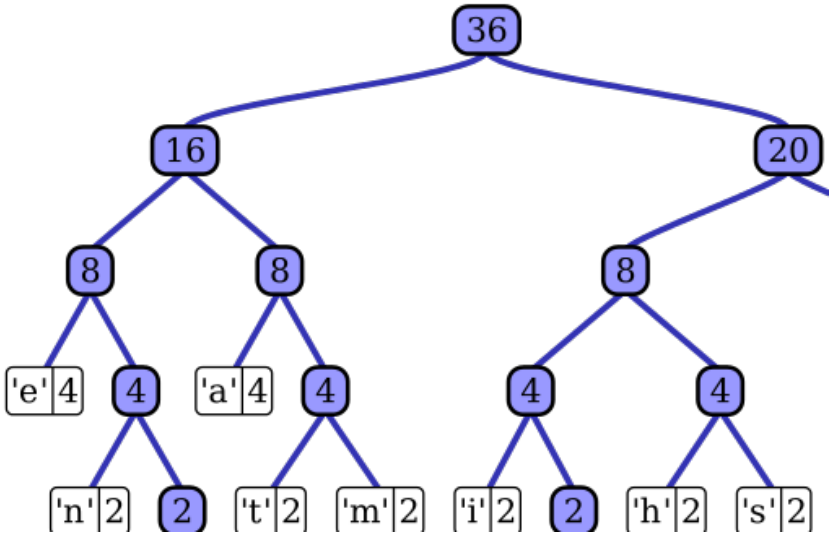
Specification: Fix alphabet $\sum$

Input: $w \in \sum^*$ **Output:** "short bit-encoding" of w

1. Determine frequencies f_s of symbols $s \in \Sigma$ in w

2. Choose prefix-free $C \subseteq \{0,1\}^*$ / binary tree T

3. Assign to each $s \in \Sigma$ a unique $c_s \in C$ / leaf l_s of T such as to minimize weighted length $\sum_{s \in \Sigma} d(l_s) \cdot f_s$



length =depth

Char	Freq	Co
space	7	11
a	4	01
e	4	00
f	3	11
h	2	10
i	2	10
m	2	01
n	2	00
s	2	10
t	2	01
l	1	11
o	1	00
p	1	10

6. String Problems

Huffman Tree

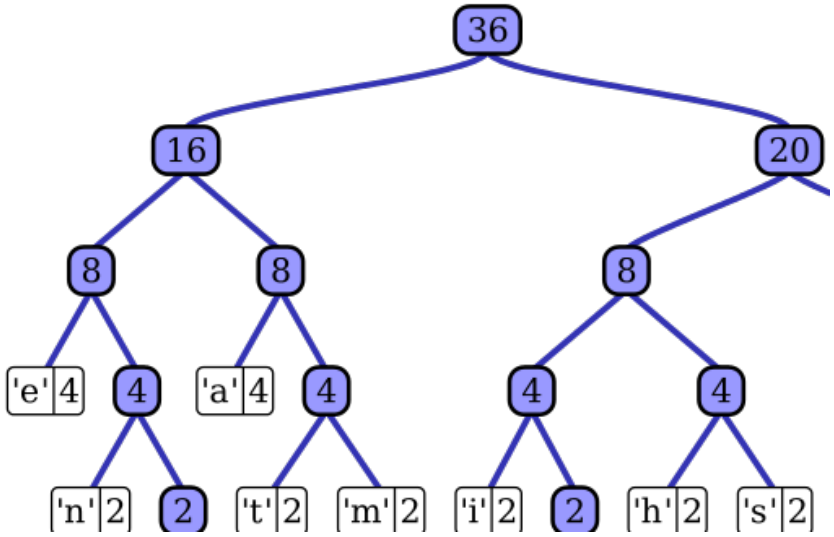
Specification: Fix alphabet Σ

Input: $w \in \Sigma^*$ **Output:** "short bit-encoding" of w

1. Determine frequencies f_s of symbols $s \in \Sigma$ in w

2. Choose prefix-free $C \subseteq \{0,1\}^*$ / binary tree T

3. Assign to each $s \in \Sigma$ a unique $c_s \in C$ / leaf l_s of T such as to minimize weighted length $\sum_{s \in \Sigma} \mathrm{d}(l_s) \cdot f_s$



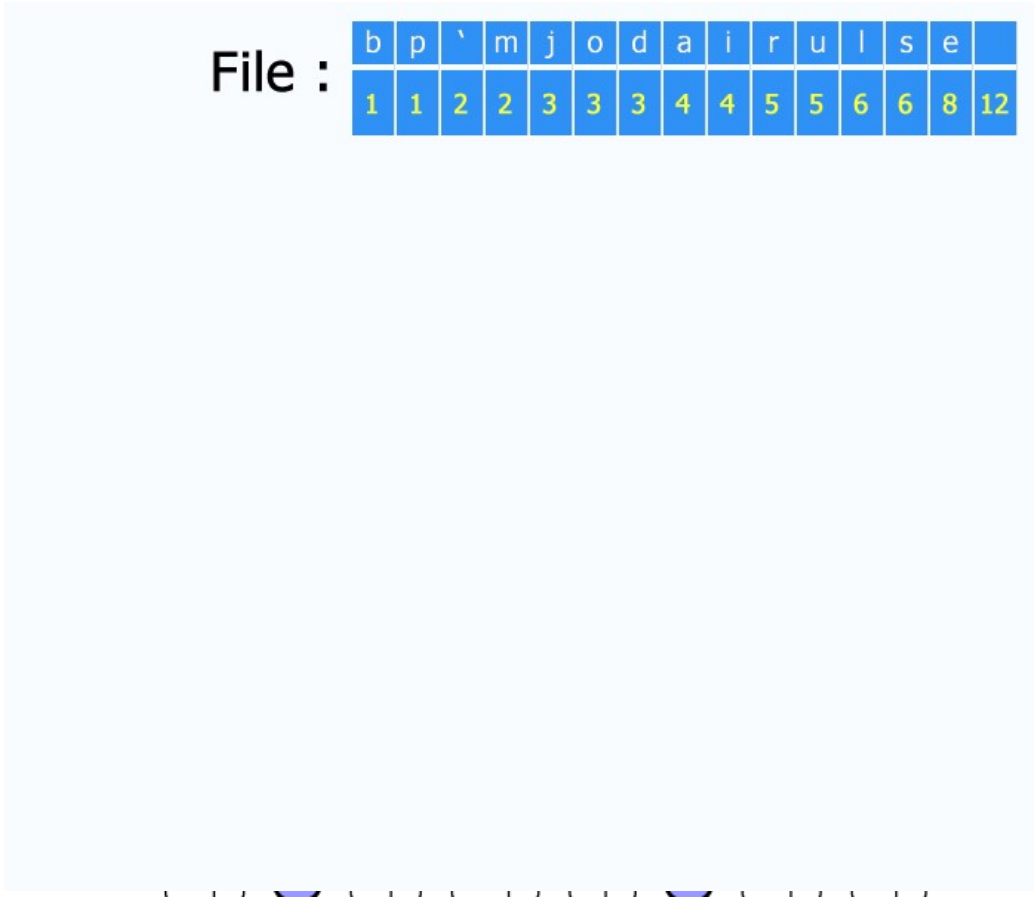
Idea: *Frequent* symbols s (=large f_s) should receive *small* depth $\mathrm{d}(l_s)$, rare ones can „afford“ large depth

6. String Problems

Huffman Tree

Specification: Fix alphabet $\sum$

Input: $w \in \sum^*$ **Output:** "short bit-encoding" of w



Extract two symbols $s,t \in \sum^*$ with least frequencies f_s, f_t.

Combine them to a tree with leaves s,t and root $s \circ t$ of frequency $f_{st} := f_s + f_t$.

Repeat.

Idea: *Frequent* symbols s (=large f_s) should receive *small* depth $d(l_s)$,

rare ones can „afford“ large depth

Recap

6. String Problems

- Recap on Strings
- Pattern Matching: *Knuth-Morris-Pratt*
- Longest Common Substring
- Edit Distance
- Context-free Parsing: *Cocke-Younger-Kasami*
- *Huffman* Compression