Special course in Computer Science: Advanced Text Algorithms

Eugen Czeizler
Department of IT, Abo Akademi
http://combio.abo.fi/teaching/textalg/

(slides originally by I. Petre, E. Czeizler, V. Rogojin)

Algorithms on strings

- Very often information is represented as a written text, i.e., a sequence of symbols from some finite alphabet:
  - Newspapers, books, journals
  - Information stored on the hard-drive, or optical disks
  - Genetic information of any living organism.
- Text algorithms occur in many areas of science:
  - Information processing: many text editors and programming languages include tools for processing texts.
  - Data compression
  - Genetics: text algorithms became essential when studying molecular sequences.

Algorithms on strings

- One of the fundamental problems in this field is pattern matching: find occurrences of a given sequence in a text.

- Implementations of algorithms for this problem are used daily for accessing information:
  - internet browsers, e.g., Google search engine,
  - library catalog searching programs
  - internet news readers or online scientific journals that can search articles for topics of interest
  - searching online encyclopedias, dictionaries,
  - querying (large) specialized databases
  - text editors, e.g., the "search" or "replace".

Algorithms on strings

- Pattern matching is often a sub-problem in other algorithmic problems on texts.

- The pattern matching problem is comparable with sorting or with basic arithmetic operations.

Dictionaries and Indexes

- Dictionaries and indexes are very useful tools in sequence searching.

- Indexes are often used in technical books.
  - They contain a list of words (or expressions) with pointers to the parts of the book that contains information related to them.

- Dictionaries are used to speed up the access to entries in static texts.

- Very useful for natural language processing, e.g., spelling checkers.

Dictionaries and Indexes

- The dictionaries can be very large:
  - the lexicon used by the UNIX command spell contains aprox. 70 000 entries

- Quick access to entries in dictionaries is essential to produce efficient parsers.

- Data structures used to achieve quick access:
  - Suffix trees
  - Directed acyclic word graphs
  - Factor automata
  - Suffix arrays

Data Compression

- Question: How to store a huge text?
- Text compression problem:
  - Reduce the size of the representation of the text as much as possible.
  - The original text can be recovered completely.
  - No loss of information is allowed.
- Availability of large storing devices does not decrease the interest in compressing data.
- Text compression is very important in the transfer of data since it reduces the time of the transfer.

Data Compression

- There are general compression methods that can be used for almost any type of data.

- However, the best results are achieved when the compression method is developed specifically for a particular type of data.
    - Example: Using some specific methods, some data received from geostationary satellites was successfully compressed to 7% of its original size without losing any information.

- The compression is very successful if the text has many regularities.
    - Detecting regularities in a text is very important.

Text algorithms in genetics

- Nucleic acids (DNA and RNA) encode fundamental information for the development and survival of an organism.

- They can be viewed as texts over a 4-letter alphabet:
    - For DNA: C (cytosine), G (guanine), A (adenine), and T (thymine)
    - For RNA: C, G, A, and U (uracil)

- Proteins can also be viewed as texts over a 20-letter alphabet (amino acids).

Text algorithms in genetics

- The development of powerful sequencing techniques led to a huge amount of data.

- Powerful and fast text algorithms are required to analyse these data.

The Human Genome Project (HGP)

- An international research project aiming:
  - to determine the sequence of chemical base pairs which make up DNA and
  - to identify and map the approximately 20,000–25,000 genes of the human genome

- There are two general strategies for sequencing whole genomes:
  - Hierarchical Shotgun Sequencing. This method was used in the Human Genome Project
  - Shotgun Sequencing (CELERA). This approach was developed for prokaryotic genomes which are smaller in size and contain less repetitive DNA.

Shotgun Sequencing

- Multiple copies of the genome are broken randomly into smaller pieces (2000 and 10 000 bp long).
- These pieces are replicated in a huge number.
- Each of these pieces is then sequenced separately.
- The original sequence is then reconstructed from the smaller fragments using specific sequence assembly algorithms.

- The shortest common superstring problem: construct the shortest string that contains some given smaller fragments.

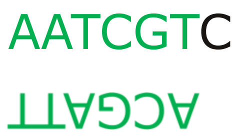
Text algorithms in genetics

- Once the large sequence is obtained the next question is whether it resembles another sequence which is already in some molecular database.

- Edit distance problem:
  - How different are two given sequences?
  - High sequence similarity often indicates significant functional or structural similarity.
  - It aligns the two sequences and computes the minimal number of edit operations through which one string can be transformed into the other.

Text algorithms in genetics

- Text algorithms can also be useful when analysing molecular sequences.
  - Determine regularities in a molecular sequence.
  - Determine which parts of the molecules are coding.
  - Determine the 3D-fold of a molecule
    - Nucleotides bind pair-wise (A-T and C-G)
    - Thus in a folded molecule we should have approximate palindromic sequences, e.g., AATCGTCACGATT



  - Approximate search for regularities such as repetitions or palindromes, where certain types of errors are allowed

Efficiency of algorithms

- Depending on the complexity measure used, we have several different notions of efficiency.

- Complexity measures:
  - Sequential time
  - Memory space

- Evaluating precisely the complexity of an algorithm is a difficult problem.

Efficiency of algorithms

- The big O notation:
  - It states what are the important terms of the complexity expression.
  - It allows its users to simplify functions in order to concentrate on their growth rates.

- For 2 functions f and g, we write $f(n) \in O(g(n))$ if *f* is bounded above by *g* (up to constant factor) asymptotically, i.e., $|f(n)|<C\,|g(n)|$ for all $n>N$ for two constants N and C.

- Example: Let $f(n)=6n^4-2n^3+5$. Then $f(n) \in O(n^4)$ (i.e., $g(n)=n^4$) since we can chose $N=1$ and $C=13$ such that:
$|6n^4-2n^3+5|<|6n^4+2n^3+5|<6n^4+2n^4+5n^4=13n^4=13|n^4|$.

Efficiency of algorithms

- Big Theta notation: We write $f(n) \in \Theta(g(n))$ if the function *f* is bounded both above and below by *g* asymptotically, i.e., there exists constants k_1, k_2, and N such that for all $n>N$ $|k_1 g(n)|<|f(n)|<|k_2 g(n)|$

- Informally, the Big *O* notation is sometimes "abused" to describe an asymptotic tight bound where using Big Theta notation might be more.

- Example: For $f(n)=7n^3+n^2+5$ we can write
  - $f(n)=O(n^3)$, i.e., f(n) grows asymptotically no faster than n^3
  - $f(n)=\Theta(n^3)$, i.e., f(n) grows asymptotically as fast as n^3

Efficiency of algorithms

- For sequential machines we have several types of computations:
  - Off-line algorithms
  - On-line algorithms
  - Real-time

Efficiency of algorithms

- Off-line algorithms:
  - the whole input can be stored in the memory before the computation actually starts.
  - we are interested only in the end result.
  - time complexity = the total time from the moment the computation starts until the end result.

Efficiency of algorithms

- On-line algorithms:
  - portions of the input data are processed (read, edited, deleted, etc) at each step.
  - we expect an intermediary result at the end of each step, and only after that the algorithm can continue processing the input.
  - we are interested in the total time T(n) necessary to obtain the first n-th intermediary results.

- Example:
  - Given a string w as input, after reading each character, we want to print 1 if the text contains a particular pattern as a suffix and 0 otherwise.
  - The output is a string over the alphabet {0,1}
  - The algorithm has to give an output value before reading the next character of the input.

Efficiency of algorithms

- Real-time algorithms:
  - “Optimal” on-line algorithms: the time between reading two consecutive input characters (i.e. the time necessary for computing the last output) is bounded by a constant.
  - Many on-line algorithms are in fact real-time algorithms.

Course Structure

1. Pattern matching algorithms
   - This is the most studied problem in algorithms on texts
   - Implementations of algorithms for this problem are used daily for accessing information:
     - use Google search engine
     - use the "search" or "replace" commands in text editors
     - make a data base query
   - There are several algorithms solving this problem efficiently, i.e., linear time complexity
     - Knuth-Morris-Pratt algorithm
     - Boyer-Moore algorithm

Course Structure

2. Suffix trees

- They are data structures for storing the suffixes of a text.
- Any factor of a string can be extended to a suffix of the text.
- By storing efficiently the suffixes of a text, we get direct access to all the factors of that text.
- These data structures are very useful
  - For solving the pattern-matching problem in linear time.
  - For constructing indexes for a text.
  - They can be used as search machines in pattern matching problems.
  - They are used in algorithms for the membership problem.

Course Structure

2. Suffix trees

- Ukkonen’s algorithm
  - Builds the suffix tree for a given text in linear time.
- Applications of suffix trees
  - The longest common factor problem
  - The longest repeated factor inside a given text

TUCS

Course Structure

3. Alignments

- Alignment = a lining up of the characters of two strings, that allows mismatches as well as matches, and allows characters of one string to be placed opposite spaces in the other string.
  - `ACTCGCCTGATGGG`
  - `ACAC CCACAT  G`
- This is one of the methods used to compare strings.
- Alignments are based on notions of distance and similarity between strings.
- Several distances can be defined on strings: prefix distance, suffix distance, factor distance, Hamming distance, edit (alignment) distance.
- The longest common subsequence of two strings is a typical example of an alignment problem.
  - This problem occurs very often when comparing biological sequences.

Course Structure

3. Alignments:

- Topics we will discuss here:
  - The edit distance and the edit graph.
  - The computation of an optimal alignment.
  - The longest common subsequence:
    - is a typical example of an alignment problem.
    - This problem occurs very often when comparing biological sequences.
    - This notion is used also for file comparison, e.g., the command diff in UNIX
  - Alignments with gaps.

- The techniques and algorithms that we will describe here are used very often in molecular biology when comparing sequences of nucleic acids (DNA or RNA) or of amino acids (proteins).

Course Structure

4. Approximate pattern matching

- This topic is central in computational molecular biology.
- Approximate search for a fixed string u in a text w: find all occurrences of factors of the text w that "approximate" the string u.
- There are several notions of approximation on strings: jokers (a special symbol that stands for any character from the alphabet), differences, mismatches.
- We will look for all factors v of the text w which are at distance at most k from u.
  - We consider here two distances: the edit distance and the Hamming distance.

Course Structure

5. Symmetries and repetitions in texts

- Algorithms dealing with regularities in strings.
- Regularity: a similarity between two factors of a text.
- Examples of regularities:
  - Two factors are identical: repetitions (abbabb, abbcdcacabb)
  - The two factors are symmetric: palindromes (abbbba, abbcbba)

Course Structure

6. Constant-space algorithms

- Very interesting are those algorithms that are efficient regarding two measures of complexity:
  - linear-time complexity
  - constant space

- There are several such algorithms for pattern matching
  - They are based on periodicity properties of the text (and pattern)
  - MaxSuffix matching algorithm: a time-space optimal extension of the Morris-Pratt algorithm
  - Galil-Seiferas algorithm
  - An algorithm for testing the cyclic equality of words.

Course Structure

7. Text compression techniques

- Provide reduced representations of the data.
- There should be no loss of information, i.e., the texts can be recovered from their compressed representations.
- The goals of data compression algorithms are very practical:
  - Reduce the memory space necessary to store the information.
  - Accelerate data transmission in telecommunication.

TUCS

Course Structure

7. Text compression techniques

- We will discuss data compression techniques based on substitutions.
  - these methods are general, so they can be applied on various types of data.
- Data compression techniques aim at eliminating redundancies, repetitions or other regularities in the considered text.

- We will discuss the Lempel-Ziv compression algorithm and Huffman statistical coding.

Course structure

- Course given during October 30 – December 18, every Tuesday 15-17 and Thursday 10-12 in room 115A, Agora

- 14 lectures, including several exercise sessions (completing exercises will gain few point in the exam);
- Final exam 20.12.2018 (and other
  - Course webpage: http://combio.abo.fi/teaching/textalg/
  - Lecture slides
  - Announcements
  - Literature

Notations and Definitions

- An alphabet A = a finite set of symbols.
- The elements of A are called letters or characters.
- Examples:
  - The latin alphabet
  - The binary set {0,1}
  - The 4-letter alphabet of nucleotides {A,T,C,G}
- A text (or word) w = a finite sequence of letters from a given alphabet A.
  - The empty word is denoted by ε.
- A language L over the alphabet A= a set of words over A.

Notations and Definitions

- The length of a word w, denoted by |w|= the number of letters occurring in w, counting also repetitions.
  - Example: |a|=1, |aba|=3, |ε|=0

- The i-th letter of a word w is denoted by w[i]

- The concatenation of two words u and v is the word uv obtained by writing the letters of u followed by the letters of v.

- The reverse of a word w=w[1]w[2]..w[n] is the word w^r=w[n]..w[2]w[1]
  - Example: w=coffee w^r=eeffoc

Notations and Definitions

- A word u is a prefix (resp. suffix) of a word w if we can write w=uv (resp. w=vu) for some word v.
  - Example: u=cof is a prefix of w=coffee

    u=ee is a suffix of w=coffee

- A factor of length n of a word w is a sequence of n consecutive letters w[i]w[i+1]...w[i+n-1] for some i>0.

- A sequence of letters w[i]w[i+1]..w[j] is usually denoted w[i..j].
  - If i>j then w[i..j]=ε.

Notations and Definitions

- We say that u occurs in w (or there is an occurrence of u in w) if u is a factor of w.

- We say that an occurrence of u starts at left position i on w if u=w[i..i+|u|-1]

- We can also consider the right position i+|u|-1 at which this occurrence ends in w

- Example:

i	1	2	3	4	5	6	7	8	9
w[i]	b	a	b	a	a	b	a	b	a
left positions of aba		2			5		7		
right positions of aba				4			7		9

Notations and Definitions

- A word u is a subsequence (subword) of a word v if it can be obtained from v by removing 0 or more letters (not necessarily adjacent).
  - If v=v[1]v[2]..v[n], then $u=v[i_1]v[i_2]..v[i_k]$ for an increasing sequence of indexes $1<i_1<i_2<..<i_k<n$.
  - Example: abc is a subsequence of abbac: abbac or abbac

Notations and Definitions

- A period of a word w is an integer $0<p<|w|$ such that $w[i]=w[i+p]$ for all $i\in\{1, .., |w|-p\}$.
  - Sometimes, we say that the word w[1..p] is a period of the word w.
  - Example:
    - abcabc has period 3,
    - abababa has periods 2,4, and 6

- Properties of periodicities of words are among the main theoretical tools in string-matching algorithms.

Periodicity properties

- An integer $p>0$ is a period of a word w whenever one of the following statements is true:

1. The word w is a factor of a word t^k with $|t|=p$
2. The word w can be written as $w=xy=yz$ with $|x|=|z|=p$
3. There exists two words u and v and an integer k such that $w=(uv)^k u$ and $|uv|=p$

Notations and Definitions

- We denote by period(w) the smallest period of w.

- A word w is called periodic if period(w)≤|w|/2.
    - Examples: $abab, abcabcab$ are periodic words

      aba is not a periodic word

- A word w is called primitive if it cannot be written as a power of another word, i.e., there is no word u such that $w=u^k$ for some $k\geq 2$.
    - Examples of primitive words: abc, abb
    - Examples of words which are not primitive: $abab=(ab)^2$, a^3

Notations and Definitions

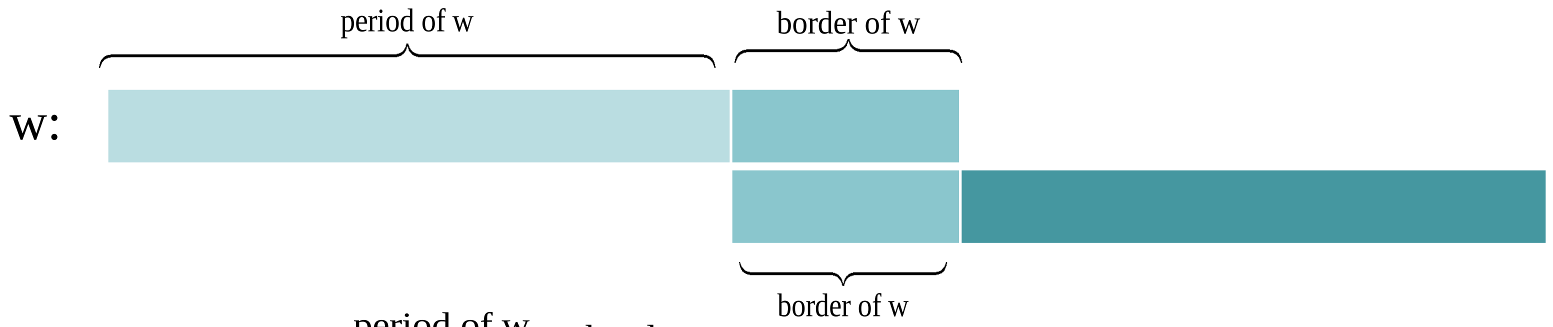
- A word u is a border of a word v if u is both a prefix and a suffix of v.
  - Example: abaaaab

- Unbordered words are a special class of primitive words.

- If we have at least 2 letters in the alphabet, any word w can be transformed into an unbordered word:
  - Let $u=wab^{|w|}$, where a is the first letter of w and b≠a.
  - Then no prefix of length smaller than |w| of u can be a suffix of u.
  - The same is true for prefixes of length larger than |w|.

Notations and Definitions

- There is a strong connection between periods and borders of a word.



- Example: $\overbrace{ababbbbb}^{\text{period of w}}\overset{\text{border}}{aba}$

$$aba\,bbbbbaba$$

- If u is a border of w, then |w|-|u| is a period of w.

Notations and Definitions

- Just as in the case of periods, there are words which have several borders.

- Example: ababab

$ab\,abab$ $\quad$ $abab\,ab$

$abab\,ab$ $\quad$ $ab\,abab$

- Denote by Border(w) the longest nontrivial border of a word w, i.e., Border(w)≠w.

Periodicity Lemma

- Periodicity Lemma: Let p and q be two periods of a word w. If p+q<|w|, then gcd(p,q) is also a period of w.

- Strong periodicity lemma (Fine and Wilf 1965): If p and q are periods of a word w such that p+q-gcd(p,q)≤|w|, then gcd(p,q) is a period of w. Moreover, the bound p+q-gcd(p,q) is strict.

- Example for the strictness of the bound:
  - The word abaababaaba has periods p=5 and q=8:
    - (abaab)(abaab)a
    - (abaababa)aba
  - |abaababaaba|=11=5+8-gcd(5,8)-1 (=p+q-gcd(p,q)-1)
  - It does not have gcd(5,8)=1 as a period

Periodicity properties

- Fibonacci words satisfy numerous properties related to periodicity and repetitions.

- They are defined inductively:

$$F_1 = b, \;\; F_2 = a, \;\; F_n = F_{n-1}F_{n-2}, \text{ for } n \geq 2$$

Periodicity properties

- F_1=b
- F_2=a
- F_3=ab
- F_4=aba
- F_5=abaab
- F_6=abaababa
- F_7=abaababaabaab
- F_8=abaababaabaababaababa

- Any F_i is a prefix of F_{i+1}
- After cutting the last 2 letters of F_i, we obtain a palindrome word, i.e., a word which is identical with its reverse image.
- The lengths of these words are the Fibonacci numbers
- These words contain many periodicities but none with an exponent larger that 4

Cyclic Shifts

- A cyclic shift of a word w is a circular permutation of w, i.e., any word uv if we can write w=vu.

- Example: *abbaba*

  bbabaa

  babaab

  abaabb

  baabba

  aabbab

TUCS

Primitive words

- Lemma: A primitive word w has exactly p=|w| cyclic shifts.

- Corollary: A word w of length |w|=n having p as a period has
    - exactly p different cyclic shifts if p/n
    - n different cyclic shifts otherwise.