Further topics

- Pattern Matching
- Planar Graphs
- Fixed-Parameter Tractable Algorithms

Pattern Matching

- Goal: find all occurrences of a pattern in a text
- Applications: text editing (eg for “find and replace all”), the search for occurrences of a *motif* in biological sequences

String Matching Problem

- **Input**: *text* of length n, that is an array $T[1..n]$, and a *pattern* of length m, that is an array $P[1..m]$; all elements in T and P are from finite alphabet (set) Σ
- **Output**: All valid shifts with which pattern P occurs in text T
- Here
  - Pattern P occurs with *shift* s in text T if $0 \le s \le n\text{-}m$ and $T[s+1\ldots s+m] = P[1..m]$, that is $T[s+j] = P[j]$ for $1 \le j \le m$
  - If P occurs with shift s in T then we call s a valid shift; otherwise we call s an invalid shift

Algorithms solving the String-Matching Problem

- Brute-force algorithm
- Algorithm by Rabin and Karp
- Knuth-Morris-Pratt algorithm
- Algorithm by Boyer and Moore

Brute-force algorithm

NaiveStringMatcher(T,P)

$n \leftarrow$ length[T]

$m \leftarrow$ length[P]

for s from 1 to n-m **do**

if $P[1..m] = T[s..s+m]$ **then**

print(“valid shift” s)

Running time

- $O(nm)$
- Practical performance: bad

Rabin-Karp Algorithm

- Idea: speeds up naive algorithm by the use of hashing
- Worst-case running time does not improve
- Running time improves in practice

Speeding up the brute force algorithm

- Reminder: *hash function*: function that maps every string to a numeric value (*hash value*)
- Note: if two strings are equal then so are their hash values
- Idea:
  - compute the hash value of the pattern
  - look for a substring with the same hash value
- Problem: same hash values will exists for different strings; patterns can be long
- Needed: hash function that yields a good performance in practice

Knuth-Morrison-Pratt Algorithm

- String matching using finite automata (*string matching automata*)