Algorithms and datastructures II

Lecture 2: text searching (2/2)

Jan Hubička

Department of Applied Mathematics
Charles University
Prague

Oct 5 2020

String-searching (“searching needle in a haystack”)

string-searching

Given a string ν (“a needle”) and η (“a haystack”) find all occurrences of ν in η.

Some notation:

1. Σ: an alphabet (finite set of characters)
2. Σ^*: the set all finite words in alphabet Σ
3. α, β, . . . : words
4. $|\alpha|$: length of the word α.
5. ϵ: empty word (the only word of length 0)
6. $\alpha\beta$: concatenation of α and β
7. $\alpha[i]$: i-th character of α (starting from 0)
8. $\alpha[i : j]$: subword $\alpha[i]\alpha[i+1]\cdots\alpha[j-1]$
9. $\alpha[: j]$: prefix of α of length j
10. $\alpha[i :]$: a suffix of α
11. $\alpha[:]$: whole word α

Occurrence of ν in η is any index i such that $\eta[i : i + |\nu|] = \nu$

Knuth–Morris–Pratt (KMP) algorithm (1974)

Searching automaton

1. State $0, \ldots, |\nu|$
   (state s corresponds to prefix $\nu[: s]$)
2. Forward edges: $s \to s + 1$
3. Backward edges: pointing from $s > 0$ to j such that $\nu[: j]$ is a proper suffix of $\nu[: s]$

KMPConstruction (ν):

1. $b[0] \leftarrow$undefined, $b[1] \leftarrow 0$, $s \leftarrow 0$.
2. For $i = 2, \ldots, |\nu|$:
3. $\quad s \leftarrow$ Step $(s, \nu[i-1])$.
4. $\quad b[i] \leftarrow s$.

Step (s, c):

1. While $s \neq 0$ and $\nu[s] \neq c$:
2. $\quad s \leftarrow b[s]$.
3. If $\nu[s] = c$: $s \leftarrow s + 1$.
4. Return s

Search (η, automaton for ν):

1. $s \leftarrow 0$.
2. For $i = 0, \ldots, |\eta| - 1$:
3. $\quad s \leftarrow$ Step $(s, \eta[i])$.
4. $\quad$ If $s = |\nu|$: report $i - |\nu| + 1$.

Theorem

Algorithm KMP will finish in time $\Theta(|\eta| + |\nu|)$.

Invariant: The state s corresponds to the longest suffix of $\eta[: i]$ that is a prefix of ν.

Today: Searching multiple needles at once

string-searching

Given a string ν (“a needle”) and η (“a haystack”) find all occurrences of ν in η.

Today: Searching multiple needles at once

string-searching

Given a string ν (“a needle”) and η (“a haystack”) find all occurrences of ν in η.

string-searching with multiple needles

Given strings $\nu_1, \nu_2, \ldots, \nu_n$ (“a needles”) and η (“a haystack”) find all occurrences of $\nu_1, \nu_2, \ldots, \nu_n$ in η.

We expect output in the form $S = \{(i, j) : \eta[i : i + |\nu_j|] = \nu_j\}$.

Today: Searching multiple needles at once

string-searching

Given a string ν ("a needle") and η ("a haystack") find all occurrences of ν in η.

string-searching with multiple needles

Given strings $\nu_1, \nu_2, \ldots, \nu_n$ ("a needles") and η ("a haystack") find all occurrences of $\nu_1, \nu_2, \ldots, \nu_n$ in η.

We expect output in the form $S = \{(i, j) : \eta[i : i + |\nu_j|] = \nu_j\}$.
Time complexity using KMP:

$$O\left(|\eta| \cdot n + \sum_{i=1}^{n} |\nu_i|\right)$$

Today: Searching multiple needles at once

string-searching

Given a string ν (“a needle”) and η (“a haystack”) find all occurrences of ν in η.

string-searching with multiple needles

Given strings $\nu_1, \nu_2, \ldots, \nu_n$ (“a needles”) and η (“a haystack”) find all occurrences of $\nu_1, \nu_2, \ldots, \nu_n$ in η.

We expect output in the form $S = \{(i, j) : \eta[i : i + |\nu_j|] = \nu_j\}$.
Time complexity using KMP:

$$O\left(|\eta| \cdot n + \sum_{i=1}^{n} |\nu_i|\right)$$

We seek for:

$$O\left(|\eta| + \sum_{i=1}^{n} |\nu_i| + |S|\right)$$

Aho-Corasick (1975)

Aho-Corasick: The automaton

Searching automaton (Knuth–Morris–Pratt)

1. States: $0, \ldots, |\nu|$
2. Forward edges: $s \to s+1$
3. Backward edges: pointing from $s > 0$ to j such that $\nu[: j]$ is a proper suffix of $\nu[: s]$

Searching automaton (Aho–Corasick)

- States: $\{\alpha : \exists_i\ \alpha$ is a prefix of $\eta_i\}$.
- Forward edges: $\{(\alpha, \beta) : \exists_{x \in \Sigma} \beta = \alpha x\}$.
- Backward edges: $\{(\alpha, \beta) : \beta$ is the longest proper suffix of α that is a state$\}$.

Aho-Corasick: How to output the locations?

1. Output whenever a leaf state is reached

Aho-Corasick: How to output the locations?

1. Output whenever a leaf state is reached (not working)

Aho-Corasick: How to output the locations?

1. Output whenever a leaf state is reached (not working)
2. Follow back edges and output all needles on the path

Aho-Corasick: How to output the locations?

1. Output whenever a leaf state is reached (not working)
2. Follow back edges and output all needles on the path (slow)

Aho-Corasick: How to output the locations?

1. Output whenever a leaf state is reached (not working)
2. Follow back edges and output all needles on the path (slow)
3. Precompute lists of needles to output when given state is reached

Aho-Corasick: How to output the locations?

1. Output whenever a leaf state is reached (not working)
2. Follow back edges and output all needles on the path (slow)
3. Precompute lists of needles to output when given state is reached (too large automaton)

Aho-Corasick: How to output the locations?

1. Output whenever a leaf state is reached (not working)
2. Follow back edges and output all needles on the path (slow)
3. Precompute lists of needles to output when given state is reached (too large automaton)
4. Output edges pointing from state α to nearest output state β reachable by back edges.

Representation of the searching automaton

1. States: 1, 2, . . . , N.

Aho-Corasick: How to output the locations?

1. Output whenever a leaf state is reached (not working)
2. Follow back edges and output all needles on the path (slow)
3. Precompute lists of needles to output when given state is reached (too large automaton)
4. Output edges pointing from state α to nearest output state β reachable by back edges.

Representation of the searching automaton

1. States: 1, 2, . . . , N.
2. Needles: Needle[s].

Aho-Corasick: How to output the locations?

1. Output whenever a leaf state is reached (not working)
2. Follow back edges and output all needles on the path (slow)
3. Precompute lists of needles to output when given state is reached (too large automaton)
4. Output edges pointing from state α to nearest output state β reachable by back edges.

Representation of the searching automaton

1. States: 1, 2, . . . , N.
2. Needles: Needle[*s*].
3. Forward edges: Forward[*s*][*c*].
   (*s* is an index of a state, *c* is an incoming character.)

Aho-Corasick: How to output the locations?

1. Output whenever a leaf state is reached (not working)
2. Follow back edges and output all needles on the path (slow)
3. Precompute lists of needles to output when given state is reached (too large automaton)
4. Output edges pointing from state α to nearest output state β reachable by back edges.

Representation of the searching automaton

1. States: 1, 2, . . . , N.
2. Needles: Needle[*s*].
3. Forward edges: Forward[*s*][*c*].
   (*s* is an index of a state, *c* is an incoming character.)
4. Backward edges: Back[*s*].

Aho-Corasick: How to output the locations?

1. Output whenever a leaf state is reached (not working)
2. Follow back edges and output all needles on the path (slow)
3. Precompute lists of needles to output when given state is reached (too large automaton)
4. Output edges pointing from state α to nearest output state β reachable by back edges.

Representation of the searching automaton

1. States: 1, 2, . . . , N.
2. Needles: Needle[s].
3. Forward edges: Forward[s][c].
   (s is an index of a state, c is an incoming character.)
4. Backward edges: Back[s].
5. Output edges: Output[s].

Aho–Corasick: search loop

Representation of the searching automaton

1. States: 1, 2, . . . , N.
2. Needles: Needle[s].
3. Forward edges: Forward[s][c].
   (s is an index of a state, c is an incoming character.)
4. Backward edges: Back[s].
5. Output edges: Output[s].

Aho–Corasick: search loop

Representation of the searching automaton

1. States: 1, 2, . . . , N.
2. Needles: Needle[s].
3. Forward edges: Forward[s][c].
   (s is an index of a state, c is an incoming character.)
4. Backward edges: Back[s].
5. Output edges: Output[s].

Step (s,c):

1. While Forward[s][c] $= \emptyset$ and $s \neq$ root: $s \leftarrow$Back[s].
2. If Forward[s][c] $\neq \emptyset$: $s \leftarrow$Forward[s][c].
3. Return s.

Aho–Corasick: search loop

Representation of the searching automaton

1. States: 1, 2, . . . , N.
2. Needles: Needle[s].
3. Forward edges: Forward[s][c].
   (s is an index of a state, c is an incoming character.)
4. Backward edges: Back[s].
5. Output edges: Output[s].

Step (s,c):

1. While Forward[s][c] $= \emptyset$ and $s \neq$ root: $s \leftarrow$Back[s].
2. If Forward[s][c] $\neq \emptyset$: $s \leftarrow$Forward[s][c].
3. Return s.

Search (η, automaton)

1. $s \leftarrow$root.
2. For characters c in η do:
3. $\quad s \leftarrow$ Step (s, c).
4. $\quad j \leftarrow s$.
5. $\quad$While $j \neq \emptyset$:
6. $\quad\quad$If Needle(j) $\neq \emptyset$: Report occurence of needle.
7. $\quad\quad j \leftarrow$ *Output*(j).

Aho–Corasick: search loop

Representation of the searching automaton

1. States: 1, 2, . . . , N.
2. Needles: Needle[s].
3. Forward edges: Forward[s][c].
   (s is an index of a state, c is an incoming character.)
4. Backward edges: Back[s].
5. Output edges: Output[s].

Step (s,c):

1. While Forward[s][c] $= \emptyset$ and $s \neq$ root: $s \leftarrow$Back[s].
2. If Forward[s][c] $\neq \emptyset$: $s \leftarrow$Forward[s][c].
3. Return s.

Search (η, automaton)

1. $s \leftarrow$root.
2. For characters c in η do:
3. $\quad s \leftarrow$ Step (s, c).
4. $\quad j \leftarrow s$.
5. $\quad$While $j \neq \emptyset$:
6. $\qquad$If Needle(j) $\neq \emptyset$: Report occurence of needle.
7. $\qquad j \leftarrow$ *Output*(j).

Invariant: The state s corresponds to the longest suffix of $\eta[:i]$ that is a prefix of some needle.

Lemma

Search runtime is $\Theta(|\eta| + |S|)$ *.*

Aho–Corasick: automaton construction

Construct ($\nu_1, \ldots, \nu_n$)

1. Initialize a trie with root r.
   (sets Forward and Needle arrays)
2. Insert words $\nu_1, \ldots, \nu_n$ to the trie
3. Back(r) $\leftarrow \emptyset$, Output(r) $\leftarrow \emptyset$.
4. Create a queue Q and insert all sons of r.
5. For every son s of r: Back(s) $\leftarrow r$, Output(s) $\leftarrow \emptyset$
6. While $Q \neq \emptyset$:
7. Dequeue i from Q.
8. For every son s of i:
9. $b \leftarrow$ Step (Back[i], letter on edge (i, s)).
10. Back[s] $\leftarrow b$.
11. If Needle[b] $\neq \emptyset$: Output[s] $\leftarrow b$
    else Output[s] $\leftarrow$ Output[b].
12. Append s to Q.

Representation of the searching automaton

1. States: 1, 2, . . . , N.
2. Needles: Needle[s].
3. Forward edges: Forward[s][c].
   (s is an index of a state, c is an incoming character.)
4. Backward edges: Back[s].
5. Output edges: Output[s].

Aho–Corasick: automaton construction

Construct ($\nu_1, \ldots, \nu_n$)

1. Initialize a trie with root r.
   (sets Forward and Needle arrays)
2. Insert words $\nu_1, \ldots, \nu_n$ to the trie
3. Back(r) $\leftarrow \emptyset$, Output(r) $\leftarrow \emptyset$.
4. Create a queue Q and insert all sons of r.
5. For every son s of r: Back(s) $\leftarrow r$, Output(s) $\leftarrow \emptyset$
6. While $Q \neq \emptyset$:
7. Dequeue i from Q.
8. For every son s of i:
9. $b \leftarrow$ Step (Back[i], letter on edge (i, s)).
10. Back[s] $\leftarrow b$.
11. If Needle[b] $\neq \emptyset$: Output[s] $\leftarrow b$
    else Output[s] $\leftarrow$ Output[b].
12. Append s to Q.

Representation of the searching automaton

1. States: 1, 2, . . . , N.
2. Needles: Needle[s].
3. Forward edges: Forward[s][c].
   (s is an index of a state, c is an incoming character.)
4. Backward edges: Back[s].
5. Output edges: Output[s].

Lemma

Runtime of Construction is $O\left(\sum_{i=1}^{n} \nu_i\right)$.

Aho–Corasick: automaton construction

Construct ($\nu_1, \ldots, \nu_n$)

1. Initialize a trie with root r.
   (sets Forward and Needle arrays)
2. Insert words $\nu_1, \ldots, \nu_n$ to the trie
3. Back(r) $\leftarrow \emptyset$, Output(r) $\leftarrow \emptyset$.
4. Create a queue Q and insert all sons of r.
5. For every son s of r: Back(s) $\leftarrow r$, Output(s) $\leftarrow \emptyset$
6. While $Q \neq \emptyset$:
7.   Dequeue i from Q.
8.   For every son s of i:
9. $b \leftarrow$ Step (Back[i], letter on edge (i, s)).
10.   Back[s] $\leftarrow b$.
11.   If Needle[b] $\neq \emptyset$: Output[s] $\leftarrow b$
            else Output[s] $\leftarrow$ Output[b].
12.   Append s to Q.

Representation of the searching automaton

1. States: 1, 2, . . . , N.
2. Needles: Needle[s].
3. Forward edges: Forward[s][c].
   (s is an index of a state, c is an incoming character.)
4. Backward edges: Back[s].
5. Output edges: Output[s].

Lemma

Runtime of Construction is $O\left(\sum_{i=1}^{n} \nu_i\right)$.

Theorem

Aho–Corasick algorithm will find all occurences of needles in η *and will terminate in time* $O\left(|\eta| + \sum_{i=1}^{n} |\nu_i| + |S|\right)$.

Network flow

Network flow

Network flow

Definition (Network)

Network is an 4-tuple $N = (V, E, s, t, c)$ where

1. (V, E) is a directed graph,
2. $s \in V$ is a source vertex,
3. $t \in V$ is a sink vertex,
4. $c : E \to \mathbb{R}_0^+$ is a function assigning every edge a capacity.

- $f^+(v) = \sum_{u,(u,v)\in E} f(u, v)$ (flow into a vertex)
- $f^-(v) = \sum_{u,(v,u)\in E} f(v, u)$ (flow out of a vertex)
- $f^\Delta(v) = f^+(v) - f^-(v)$ (surplus)

Here $f : E \to \mathbb{R}_0^+$

Network flow

Definition (Network)

Network is an 4-tuple $N = (V, E, s, t, c)$ where

1. (V, E) is a directed graph,
2. $s \in V$ is a source vertex,
3. $t \in V$ is a sink vertex,
4. $c : E \to \mathbb{R}_0^+$ is a function assigning every edge a capacity.

- $f^+(v) = \sum_{u,(u,v)\in E} f(u, v)$ (flow into a vertex)
- $f^-(v) = \sum_{u,(v,u)\in E} f(v, u)$ (flow out of a vertex)
- $f^\Delta(v) = f^+(v) - f^-(v)$ (surplus)

Here $f : E \to \mathbb{R}_0^+$

Definition (Flow)

Function $f : E \to \mathbb{R}_0^+$ is flow if it satisfies

1. Capacity constraint: $(\forall_{e\in E}) : f(e) \leq c(e)$
2. Conservation of flows (Kirchoff's law): $(\forall_{v\in V\setminus\{s,t\}}) : f^\Delta = 0$

Value of the flow: $|f| = f^\Delta(t)$.

Network flow

Lemma

For every flow it holds that: $f^{\Delta}(s) + f^{\Delta}(t) = 0$.

Network flow

Lemma

For every flow it holds that: $f^{\Delta}(s) + f^{\Delta}(t) = 0$.

$$\sum_{v \in V} f^{\Delta}(v) = 0$$

Naive algorithm

Network flow problem

Given network $N = (V, E, s, t, c)$ find flow f maximizing $|f|$.

Naive approach: Start with 0 flow and keep improving as long as there is path from source to sink that can be improved.