Algorytmy i struktury danych z językiem Python
Algorytm Aho-Corasick

Angela Czubak

1 Teoria

1.1 Wprowadzenie

Algorytm Aho-Corasick został opracowany przez Alfreda V. Aho oraz Margaret J. Corasick. Jego celem jest znalezienie wzorców $\mathcal{P} = \{P_1, \ldots, P_k\}$ pochodzących z pewnego słownika w tekście.
Cechą charakterystyczną tego algorytmu jest to, że szukanie wystąpień zadanych słów następuje "na raz", dzięki czemu złożoność obliczeniowa tego algorytmu wynosi $O(m + z + n)$, gdzie m - długość tekstu, w którym wyszukujemy, z - liczba wystąpień wzorców w zadanym tekście oraz $n = \sum_{i=1}^{k} |P_i|$ - sumy długości tychże.
Algorytm ten jest stosowany np. w komendzie UNIX-a - `fgrep`.
Idea algorytmu opiera się na drzewach trie i automatach, których tworzenie omówię dalej.

1.2 Drzewo trie

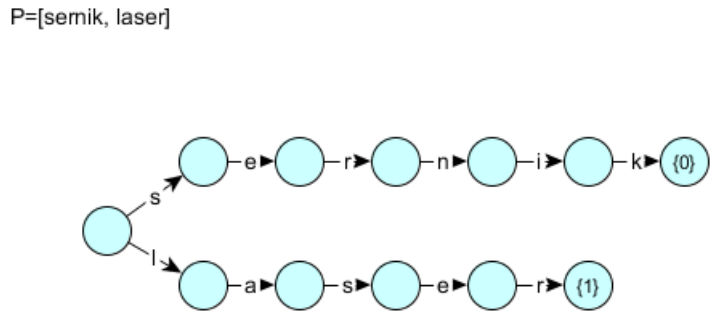
Definicja 1. ***Drzewem trie*** *dla zbioru wzorców $\mathcal{P}$ nazywamy takie ukorzenione drzewo $\mathcal{K}$, że:*

1. *Każda krawędź drzewa $\mathcal{K}$ jest etykietowana jakimś znakiem*
2. *Każde dwie krawędzie wychodzące z jednego wierzchołka mają różne etykiety*

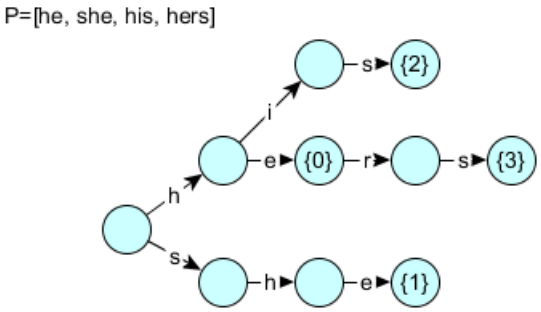
   Definicja 2. ***Etykietą węzła*** *v nazywamy konkatenację etykiet krawędzi znajdujących się na ścieżce z korzenia do v. Oznaczamy ją jako $\mathcal{L}(v)$.*

3. *Dla każdego $P \in \mathcal{P}$ istnieje wierzchołek v taki, że $\mathcal{L}(v) = P$, oraz*
4. *Etykieta $\mathcal{L}(v)$ jakiegokolwiek liścia v jest równa jakiemuś $P \in \mathcal{P}$*

Przykładowe rysunki drzew trie znajdują się na następnej stronie. Numer w wierzchołku oznacza indeks słowa należącego do słownika, które jest etykietą tego wierzchołka.



Rysunek 1: Drzewo trie dla słów $\mathcal{P} = \{sernik, laser\}$



Rysunek 2: Drzewo trie dla słów $\mathcal{P} = \{he, she, his, hers\}$

1.3 Konstrukcja drzewa trie

Jak budować drzewo trie dla $\mathcal{P} = \{P_1, \ldots, P_k\}$? Procedura jest następująca:

1. Rozpocznij od stworzenia korzenia
2. Umieszczaj kolejne wzorce jeden po drugim według poniższych kroków:
   (a) Poczynając od korzenia, podążaj ścieżką etykietowaną kolejnymi znakami wzorca P_i
   (b) Jeśli ścieżka kończy się przed P_i, to dodawaj nowe krawędzie i węzły dla pozostałych znaków P_i
   (c) Umieść identyfikator i wzorca P_i w ostatnim wierzchołku ścieżki

Jak łatwo zauważyć, konstrukcja drzewa zajmuje $O(|P_1| + \cdots + |P_k|) = O(n)$.

1.4 Wyszukiwanie wzorca w drzewie

Wyszukiwanie wzorca P odbywa się następująco:
Tak długo, jak to możliwe, podążaj ścieżką etykietowaną kolejnymi znakami P

1. Jeśli ścieżka prowadzi to wierzchołka z pewnym identyfikatorem, P jest słowem w naszym słowniku $\mathcal{P}$
2. Jeśli ścieżka kończy się przed P, to słowa nie ma słowniku
3. Jeśli ścieżka kończy się w wierzchołku bez identyfikatora, to słowa nie ma w słowniku

Wyszukiwanie zajmuje więc $O(|P|)$.
Naiwnie postępując, moglibyśmy chcieć wyszukiwać wzorce w tekście tak, by dla każdego znaku tekstu próbować iść wzdłuż krawędzi odpowiadającym kolejnym znakom - jeśli po drodze przejdziemy przez wierzchołki z identyfikatorami, to znaleźliśmy słowa im odpowiadające. Gdy już nie ma krawędzi, którą moglibyśmy przejść, zaczynamy wyszukiwanie dla kolejnego znaku tekstu. Jednak takie wyszukiwanie zajęłoby $O(nm)$ czasu, gdzie m - długość tesktu, n - suma długości wzorców.
By przyspieszyć wyszukiwanie wzorców, rozszerzamy drzewo trie do **automatu**.

2 Opis interfejsu

3 Kod źródłowy

4 Testy

5 Podsumowanie

6 Bibliografia

- `http://pl.wikipedia.org/wiki/Algorytm_Aho-Corasick`
- `http://www.cs.uku.fi/~kilpelai/BSA05/lectures/slides04.pdf`
- `https://wiki.python.org/moin/TimeComplexity`