

# Approximate Boyer-Moore

Mateusz KACZMAREK

## I Algorytm aproksymacyjnego dopasowania wzorca względem odległości edycyjnej - podejście na podstawie algorytmu Boyer'a-Moore'a.

---

**Problem 1:**  $k$  - przybliżone wyszukiwanie wzorca w tekście względem odległości edycyjnej

---

**Wejście:** Słowa  $t, p \in \mathcal{A}^+$  ( $|t| = n, |p| = m$ )

**Wyjście:** Zbiór liczb  $S = \{j : \text{istnieje podśłowo } T \text{ zakończone na } t_j \text{ o odległości edycyjnej od } p \text{ równej co najwyżej } k\}$ .

---

### I.1 Algorytm programowania dynamicznego.

Podstawowym algorytmem rozwiązującym zadany wyżej problem jest następujący algorytm programowania dynamicznego wyliczający tablicę  $D(i, j)$  oznaczającą minimalną odległość edycyjną między słowami  $p_1 \dots p_i$  oraz dowolnym podśłowem  $T$  zakończonym na  $t_j$ :

$$D(0, j) = 0, 0 \leq j \leq n$$
$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 \\ D(i-1, j-1) + 1 & \text{if } p_i = t_j \text{ then } 0 \text{ else } 1 \\ D(i, j-1) + 1 \end{cases}$$

Rozwiązaniem są wszystkie  $j$  takie, że  $D(m, j) \leq k$ . Algorytm ten działa w czasie  $O(mn)$ .

### I.2 Idea algorytmu ABM (Approximate Boyer-Moore)

Algorytm oparty na pomysłach analogicznych do algorytmu Boyer'a-Moore'a będzie działał w dwóch głównych fazach: *skanowania* i *sprawdzania*. W fazie skanowania przechodzimy przez dane słowo  $t$  i zaznaczamy pewne komórki pierwszego rzędu tablicy  $D$  (tj komórki postaci  $D(0, j)$ ). Po zakończeniu fazy skanowania rozpoczynamy fazę sprawdzania, czyli dla każdej zaznaczonej komórki  $D$  będziemy obliczali przekątną tablicy  $D$  poprzez zwykłe programowanie dynamiczne. Kiedykolwiek nasza procedura dynamiczna będzie odnosić się do niewyliczonej komórki możemy przyjąć, że jej wartość wynosi  $\infty$ .

Faza sprawdzania wydaje się dość intuicyjna dlatego skupmy się na fazie skanowania.

Faza skanowania powtarza w pętli dwie czynności. *Zaznacza* (mark) i *przesuwa* (shift) indeks który skanujemy. Operacja przesuwania jest analogiczna do operacji przesuwania w algorytmie *Boyer'a – Moore'a*. Operacja zaznaczania wykonywana jest wtedy gdy obecny indeks wymaga dokładniejszego sprawdzenia przez wyliczenie tablicy  $D$ .

### 1.3 Faza skanowania algorytmu ABM

**Definicja 1.1.** Mówimy, że dla każdego  $D(i, j)$  istnieje **łuk minimalizujący** z  $D(i-1, j)$  do  $D(i, j)$  jeśli  $D(i, j) = D(i-1, j) + 1$ , z  $D(i, j-1)$  do  $D(i, j)$  jeśli  $D(i, j) = D(i, j-1) + 1$ , oraz z  $D(i-1, j-1)$  do  $D(i, j)$  jeśli  $D(i, j) = D(i-1, j-1)$  gdy  $p_i = t_j$  albo  $D(i, j) = D(i-1, j-1) + 1$  gdy  $p_i \neq t_j$ .

**Definicja 1.2.** Dla każdego  $1 \leq h \leq n$ , **minimalizującą ścieżką** nazywamy dowolną ścieżkę złożoną z minimalizujących łuków zaczynającą w  $D(0, j)$  w pierwszym rzędzie tablicy  $D$  do komórki  $D(m, h)$  znajdującej się w ostatnim rzędzie. Minimalizującą ścieżkę nazywamy **dobrą** gdy prowadzi do  $D(m, h) \leq k$ .

**Lemat 1.3.** Komórki dowolnej dobrej ścieżki minimalizującej zawarte są wśród  $\leq k + 1$  kolejnych przekątnych tablicy  $D$ .

**Definicja 1.4.** Dla  $i = 1, \dots, m$  przez  $k$ -**otoczenie** znaku  $p_i$  ze wzorca  $p$  oznaczmy słowo  $C = p_{i-k} \dots p_{i+k}$ , gdzie  $p_j = \epsilon$  dla  $j < 1$  oraz  $j > m$ .

**Lemat 1.5.** Jeśli dobra ścieżka minimalizująca przechodzi przez pewne komórki przekątnej  $h$  tablicy  $D$ , wtedy dla co najwyżej  $k$  indeksów  $i$ ,  $1 \leq i \leq m$ , znak  $t_{h+i}$  nie występuje w  $k$ -otoczeniu  $C_i$ .

**Definicja 1.6.** Kolumnę  $j$ ,  $h+1 \leq j \leq h+m$  tablicy  $D$  nazywamy **złą** jeśli  $t_j$  nie należy do  $k$ -otoczenia  $C_{j-h}$ .

**Obserwacja 1.7.** Z lematu 1.5 wynika, że dla danego  $t_j$  złych kolumn jest co najwyżej  $k$  o ile

Na podstawie powyższej obserwacji i lematu możemy skonstruować następującą procedurę zaznaczania. Dla przekątnej  $h$ , dla  $i = m, m-1, \dots, k+1$  lub dopóki nie znaleźliśmy  $k+1$  złych kolumn sprawdź czy  $t_{h+i}$  należy do  $C_i$ . Jeśli znaleźliśmy  $\leq k$  złych kolumn wtedy zaznaczmy komórki  $D(0, h-k), \dots, D(0, h+k)$ .

Aby szybko odpowiadać na to czy kolumna jest zła możemy obliczyć tablicę  $BAD(i, a)$  dla  $1 \leq i \leq m$ ,  $a \in \mathcal{A}$  taką, że

$$Bad(i, a) = \text{true, wtw gdy } a \text{ nie należy do } k\text{-otoczenia } C_i.$$

Taką tablicę dla wzorca  $p$  możemy obliczyć w czasie  $O((|\mathcal{A}| + k)m)$ .

Po zbadaniu przekątnej  $h$  możemy ustalić przesunięcie  $d$  tak aby rozpocząć sprawdzanie od przekątnej  $d$ . Oczywiście jest, że minimalnie  $d$  może wynosić  $k+1$  i niczego nie pominiemy, jednak aby zrobić to lepiej, możemy skorzystać z podejścia analogicznego do algorytmu *Boyer'a – Moore'a*. Skorzystajmy

tutaj z tablicy przesunięć  $BM_k[i, a] = \min\{s : s = m \text{ lub } (1 \leq s < m \text{ oraz } p_{i-s} = a)\}$  wymiaru  $(k + 1) \times |\mathcal{A}|$ . Poniższy algorytm oblicza ją w czasie  $O(m + k|\mathcal{A}|)$ :

Algorytm 1: Obliczanie tablicy przesunięć Boyer'a-Moore'a

```
def boyer_moore_multi_dim_shift(A, p, m, k):
    ready = {a: m + 1 for a in A}
    BM_k = {(i, a): m for a in A for i in range(m, m - k - 1, -1)}
    for i in range(m - 1, 0, -1):
        for j in range(ready[p[i]] - 1, max(i, m - k) - 1, -1):
            BM_k[(j, p[i])] = j - i
            ready[p[i]] = max(i, m - k)
    return BM_k
```

Cała faza skanowania opisana jest w następującym kodzie:

Algorytm 2: Faza skanowania algorytmu ABM

```
j = m
while j <= n + k:
    r, i, = k, m
    bad = 0          # counts bad indexes
    d = m            # initial value of shift
    while i > k >= bad:
        if i >= m - k:
            d = min(d, BM_k[(i, t[r])])
        if BAD[(i, t[r])]:
            bad = bad + 1
        i, r = i - 1, r - 1

    if bad <= k:
        mark D(0, j-m-k), ..., D(0, j-m+k)
    j = j + max(k + 1, d)
```

## 1.4 Złożoność.

Obliczanie tablic  $BAD$  i  $BM_k$  przy założeniu, że  $k < m$  zajmuje  $O((k + |\mathcal{A}|)m)$ . Cały algorytm skanowania w najgorszym przypadku zajmuje  $O(\frac{mn}{k})$ . Autorzy pracy [https://www.researchgate.net/publication/220616992\\_Approximate\\_Boyer-Moore\\_String\\_Matching](https://www.researchgate.net/publication/220616992_Approximate_Boyer-Moore_String_Matching) udowodnili (dość nietrywialnie) następujące twierdzenie:

**Twierdzenie 1.8.** Dla  $2k + 1 < |\mathcal{A}|$  oczekiwana czas trwania Algorytmu 2 zajmuje  $O(\frac{|\mathcal{A}|}{|\mathcal{A}| - 2k} \cdot kn \cdot (\frac{k}{|\mathcal{A}| + 2k^2} + \frac{1}{m}))$ .

Fazę sprawdzania można zaimplementować tak by działała w czasie  $O(kn)$ , co sprawia, że cały algorytm  $ABM$  rozwiązuje problem  $k$ -przybliżonego dopasowania względem odległości edycyjnej w oczekiwanym czasie  $O(kn \cdot (\frac{1}{m-k} + \frac{k}{|cA|}))$