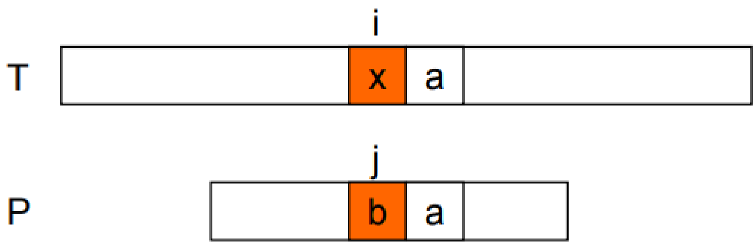
27. Prohledávání řetězců- Boyer-Moore, Rabin-Karp- princip, srovnání

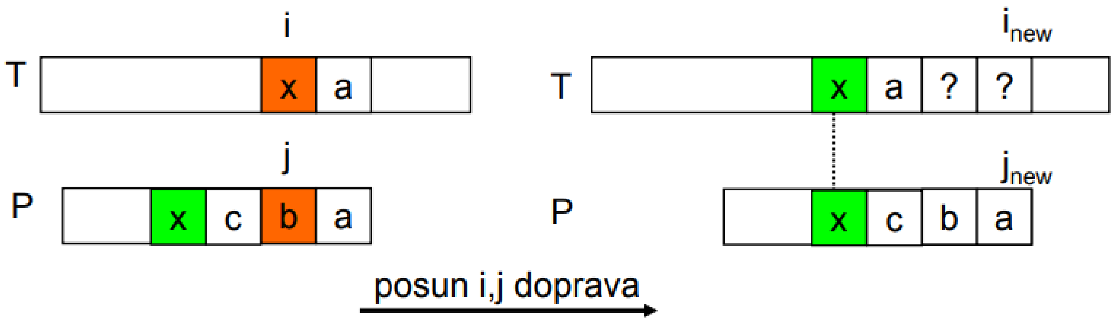
Boyer-Moore

- Základní myšlenka – hledáme zrcadlově.
- začínáme na konci P a postupujeme zpět k začátku T
- V okamžiku neshody můžeme přeskočit celé skupiny znaků, které se neshodují.
- Existují tři situace ve kterých se v okamžiku neshody (P[j] ≠ T[i]) nachází vzor.



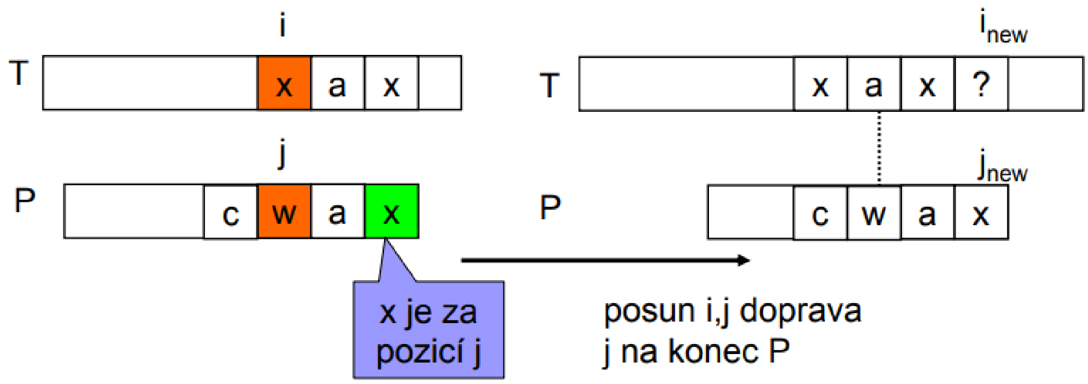
BM – případ první

- Pokud P obsahuje x, pak zkusíme posunout P doprava tak, aby se poslední výskyt x dostal proti x obsaženému v T[i].



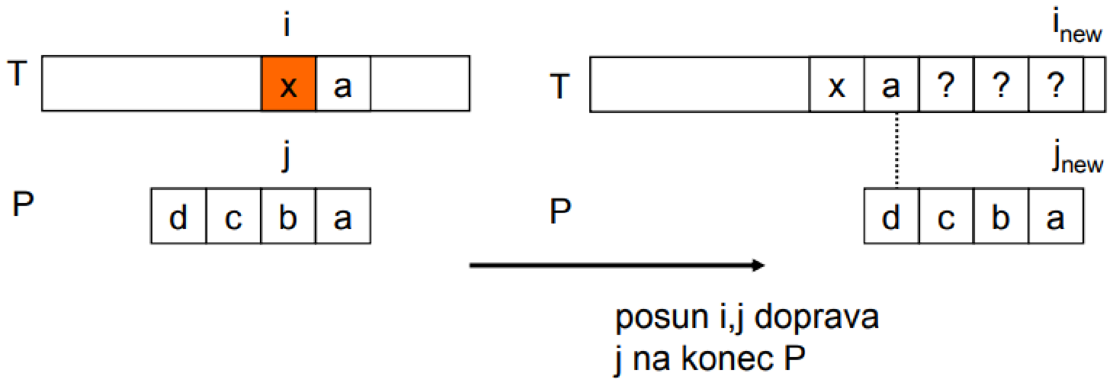
BM – případ druhý

- P obsahuje x, ale posun doprava na poslední výskyt x není možný, pak posuneme P doprava o jeden znak k T[i+1].

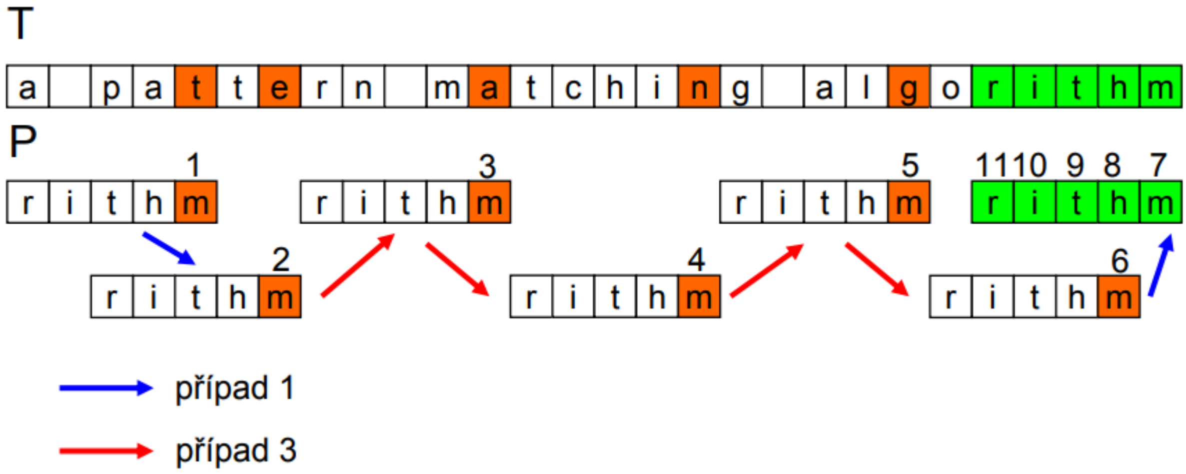


BM – případ třetí

- Pokud P neobsahuje x, posuneme P tak aby bylo P[0] zarovnáno s T[i+1].
- Nejlepší případ – skok o celý vzor.



Boyer Moore - příklad



BM – preprocesing

- Stejně jako u KMP i zde určujeme posuny předem analýzou vzoru
- Pro preprocesing používáme zobrazení všech znaků použité abecedy A do množiny celých čísel
- Funkce Prep()
- Pro libovolný znak x z A volíme Prep(x) jako:
    - Největší index i pro který platí, že P[i] == x, nebo
    - -1 pokud žádný takový index v P neexistuje

BM – příklad fce Prep()

- A = {a,b,c,d}
- P = abacab

i	0	1	2	3	4	5
P[k]	a	b	a	c	a	b

x	a	b	c	d
Prep(x)	4	5	3	-1

Zhodnocení

- Rychlý pro velkou abecedu, pomalý pro malou (podobně jako přirozené vyhledávání)
- Pro stejné abecedy bude ale BM rychlejší než přirozené

Rabin-Karp algoritmus

- Založen na použití hašování
- Vypočteme hash pro vzor P (délky M) a pro každý podřetězec řetězce T délky M
- Procházíme řetězcem T ale místo jednotlivých znaků porovnáváme hash každého podřetězce a vzoru
- V případě shody provedeme test podřetězce a vzor znaku po znaku – ochrana pro kolizi hashe

Jakou zvolit hashovací funkci?

- Klíčová volby pro efektivitu celého algoritmu
- Podřetězec se posouvá po znaku => části podřetězců jsou shodné
- Potřebujeme funkci, která umožní vypočítat hash následujícího podřetězce s využitím již vypočtených hashů

Hašovací funkce

- Zvolíme prvočíslo q – určuje velikost hašovacího prostoru
- Zvolíme základ d, může být roven velikosti |A| nebo libovolnému většímu číslu, ideálně $d = 2^x$
- Čím větší q, tím menší pravděpodobnost kolize, d jako mocnina dvou umožňuje efektivní implementaci násobení.

Hašovací fce

- Řetězec P[1:m]

$$S_m(P) = \sum_{i=1}^{m} d^{m-i} P[i] \bmod q = P[m] + dP[m-1] + \ldots + d^{m-2}P[2] + d^{m-1}P[1] \bmod q$$

- Příklad:

  A = {0,1,2,3,4,5,6,7,8,9}

  q = 13

  d = 10

  S_4(0815) = (0 . 1000 + 8 . 100 + 1 . 10 + 5) mod 13 =
  = 815 mod 13 = 9

Hašovací fce

- Pro efektivní výpočet využíváme Hornerovo schéma (polynom je v monotické formě)

$$S_m(P) \equiv \sum_{i=1}^{m} d^{m-i} P[i] \equiv d\left(\sum_{i=1}^{m-1} d^{m-i-1} P[i]\right) + P[m] \equiv dS_{m-1}(P[1..m-1]) + P[m] \pmod{q}$$

- Příklad:

  A = {0,1,2,3,4,5,6,7,8,9}, q = 13, d = 10

  S_4(0815) = ((((**0**·10+**8**)·10)+**1**)·10)+**5** mod 13 =
  ((((**8** ·10)+**1**)·10)+**5** mod 13 = (3 ·10)+**5** mod 13 = 9

Rabin-Karp zhodnocení

- Přes stejnou složitost běží algoritmus pomaleji než KMP na stejných datech (díky režii výpočtů)
- Algoritmus lze ovšem snadno rozšířit na vyhledávání více slov (vzorů) najednou
- Algoritmus se efektivně uplatňuje například v detekci plagiátů