Algorytmy tekstowe

Maciej Wisniewski wtorek 13.15 A

Laboratorium 2

Zadanie 4

Boyer-Moore algorithm

Bad characters table

```
def compute_bad_character_table(pattern: str) -> dict:

    if not pattern: return {} # edge case

    d = {} # mapa przechowujaca najbardziej na prawo wystapienie znaku w patternie
    n = len(pattern)

    for i in range(n-1,-1,-1): # od koncowego elementu
        if pattern[i] not in d: d[pattern[i]] = i # jesli wczesniej(po prawej) nie wystapil to dodajemy go do slownika

    return d
```

Good sufix table

```
def compute_good_suffix_table(pattern: str) -> list[int]:

    if not patern: return []
    n = len(pattern)
    shift = [1] * (n + 1) # przygotowanie tablicy szfitów do algorytmu

    if all(c == pattern[0] for c in pattern): # czy wszystkie sa identyczne
        for i in range(1, n + 1):
            if i == 1: shift[i-1] = 1 # jesli na i - 1 shiftujemy o 1
            else: shift[i-1] = i - 1 # niepoprawne wczesniej, szift o i - 1
        shift[n] = n  # pelny match, sziftujemy o n

    return shift
```

Booyer Moore pattern match

```
def boyer_moore_pattern_match(text: str, pattern: str) -> list[int]:

    if not pattern or not text: return [] # czy dane sa dobre do obliczen

    bad_char = compute_bad_character_table(pattern) # zliczamy bad_char_table
    good_suffix = compute_good_suffix_table(pattern) # zliczamy good_suf_table

    m = len(pattern)
    n = len(text)
    positions = [] # tablia zwracanych pozycji matchy

    i = 0
    while i <= n - m:
        j = m - 1
       # porwnujemy i przesuwamy matcha po kolei w lewo
        while j >= 0 and pattern[j] == text[i + j]: j -= 1

        if j < 0: # znaleziono match
            positions.append(i)
            i += 1
        else: # nie pasuje na j-tej
            # j - bad_char.get(text[i + j], -1) -->  przesuwamy zeby pasowal z niepasującym znakiem tak aby pokrywal
            # sie z jej ostatnim wystapieniem w paternie
            i += max(1, j - bad_char.get(text[i + j], -1))
            # teoretycznie mozna to uwzglednia good_sufix ale on nie optymalizuje
            # duzo, testy przechodza bez

    return positions
```