Zadanie "Proof" 1

Jerzy Szyjut, nr albumu: 193064

21.01.2023r.

1 Treść zadania

Przez $\sum$ będziemy rozumieli skończony alfabet złożony z liter (dowolnych symboli), np. $\sum = \{a, b, c\}$ lub $\sum = \{0, 1\}$. Przez *słowo* nad alfabetem $\sum$ będziemy rozumieli dowolny skończony ciąg elementów z alfabetu, np. $aaba$ lub 001100. Zbiór wszystkich skończonych słów nad alfabetem oznaczymy przez $\sum^+$. Wprowadzimy tzw. *słowo puste* oznaczane przez ϵ ($\epsilon \notin \sum$) oraz oznaczenie $\sum^* = \sum^+ \cup \{\epsilon\}$. Dla dwóch słów $u, v \in \sum^*$ przez uv będziemy rozumieli słowo powstałe przez *konkatenację* (sklejenie) słów u oraz v. Zdefinujemy słowo ϵ przez własności $\epsilon w = w\epsilon = w$, dla dowolnego $w \in \sum^*$. Długością słowa w, ozn. $|w|$, będziemy nazywali ilość kolejnych liter w słowie, np. $|aabab| = 5$. Niech $u, w \in \sum^*$, $u = u_1...u_k$ oraz $w = w_1...w_n$, gdzie $k = |u|$ oraz $n = |w|$. Słowo u nazwiemy ***podciągiem*** słowa w jeżeli $u_i = w_{j_i}$ dla $i \in \{1, ..., k\}$, gdzie $1 \leq j_1 < j_2 < ... < j_k \leq n$, np. $abbc$ jest podciągiem $xyababac$. Wprowadzimy oznaczenie $u \sqsubset w$, o ile u jest podsłowem w.

Zdefiniujemy problem znajdowania najdłuższych wspólnych podciągów dla danego zbioru słów. Niech $u, w \in \sum^*$, przez $nwp(u, w)$ będziemy rozumieli $max\{|v| : v \sqsubset u \wedge v \sqsubset w\}$, czyli długość najdłuższego wspólnego podciągu słów u oraz w. Analogicznie możemy zdefiniować $nwp(w_1, ..., w_k)$ dla zbioru słów $\{w_1, ..., w_k\}$.

Niech $L \subset \sum^+$. Problem NWP zdefiniujemy jako problem znalezienia najdłuższego wspólnego podciągu słów ze zbioru L (wersja optymalizacyjna). Wersja decyzyjna ma na wejściu dodatkowo liczbę $d \geq 0$ i pytamy czy $nwp(L) \geq d$. Wprowadzimy teraz wersję sparametryzowaną liczbami całkowitymi $p, q, r \geq 0$. Przez $NWP(p, q, r)$ będziemy rozumieli problem znajdowania najdłuższego wspólnego podciągu wszystkich słów ze zbioru $L \subset \sum^+$, gdzie $|L| \leq p$, $|\sum| \leq q$ oraz r oznacza ograniczenie na długość zwartych jednoliterowych ciągów w słowach ze zbioru L. Np. zapisując słowo $aabbabcccdaa$ w postaci skompresowanej z krotnościami $a^2b^2abc^3da^2$ najdłuższy ciąg jednoliterowy to c^3 i ciąg ten należy do instancji dla $r = 3$, ale nie dla $r = 2$. W oznaczeniach przyjmiemy symbol "$\cdot$"oznaczający brak wybranej restrykcji.

2 $NWP(2, \cdot, \cdot) \in P$

Algorytm wielomianowy dla problemu $NWP(2, \cdot, \cdot)$ w języku *Python* [1]

```
def nwp(u, w):
    n = len(u)
    k = len(w)

    macierz = [[0] * (k + 1) for _ in range(n + 1)]

    for i in range(n + 1):
        macierz[i][0] = 0
    for j in range(k + 1):
        macierz[0][j] = 0

    for i in range(1, n + 1):
        for j in range(1, k + 1):
            if u[i - 1] == w[j - 1]:
                macierz[i][j] = macierz[i - 1][j - 1] + 1
            else:
                macierz[i][j] = max(macierz[i - 1][j], macierz[i][j - 1]
                                     )

    podciag = []
    i, j = n, k
    while i > 0 and j > 0:
        if u[i - 1] == w[j - 1]:
            podciag.append(u[i - 1])
            i -= 1
            j -= 1
        elif macierz[i - 1][j] > macierz[i][j - 1]:
            i -= 1
        else:
            j -= 1

    return "".join(reversed(podciag))
```

Powyższy algorytm jest implementacją algorytmu dynamicznego dla problemu $NWP(2, \cdot, \cdot)$. Generuje on tablicę dwuwymiarową, gdzie w wierszach znajdują się litery słowa u, a w kolumnach litery słowa w. W komórkach tablicy znajdują się długości najdłuższych wspólnych podciągów słów u oraz w. Algorytm przechodzi po tablicy od lewej do prawej, od góry do dołu. Jeżeli litery w słowach u oraz w są takie same, to w komórce znajdującej się w tym samym wierszu i kolumnie co litery, które są takie same, zwiększamy wartość o 1. Jeżeli litery są różne, to w komórce znajdującej się w tym samym wierszu i kolumnie co litery, które są różne, wpisujemy wartość większą z dwóch wartości znajdujących się w komórkach powyżej i po lewej stronie od komórki, w której się znajdujemy. Po przejściu całej tablicy, w ostatniej komórce znajduje się długość najdłuższego wspólnego podciągu słów u oraz w.

3 Status problemu $NWP(\cdot, 2, 1)$

Algorytm dla problemu $NWP(\cdot, 2, 1)$ w języku *Python* wykonujący α-redukcję do problemu $NWP(2, \cdot, \cdot)$

```
def nwp_2_1(*sekwencje):
    dlugosc_najkrotszej_sekwencji = len(sekwencje[0])
    for sekwencja in sekwencje:
        if len(sekwencja) < dlugosc_najkrotszej_sekwencji:
            dlugosc_najkrotszej_sekwencji = sekwencja

    if dlugosc_najkrotszej_sekwencji == 0:
        return ""

    najkrotsze_sekwencje = set()
    for sekwencja in sekwencje:
        if len(sekwencja) == dlugosc_najkrotszej_sekwencji:
            najkrotsze_sekwencje.add(sekwencja)

    if len(najkrotsze_sekwencje) == 1:
        return najkrotsze_sekwencje.pop()

    if len(najkrotsze_sekwencje) == 2:
        return nwp(*najkrotsze_sekwencje)
```

Zakładając alfabet $\sum = \{0, 1\}, |\sum| = 2 = q$ i $r = 1$ każde słowo musi być ciągiem zaczynającym się od 0 lub 1 i nie może zawierać dwóch takich samych liter obok siebie. Niech n oznacza długość najkrótszego słowa ze zbioru L, $min\{|w| : w \in L\}$. $NWP(\cdot, 2, 1) = n$ kiedy najkrótsze słowo jest jedno lub najkrótsze słowa są takie same. $NWP(\cdot, 2, 1) = n - 1$, kiedy najkrótsze słowa się różnią, ponieważ najkrótsze słowa mogą się różnić pierwszym znakiem ciągu.

Bibliografia

[1] Thomas H Cormen i in. *Wprowadzenie do algorytmów*. Wydawnictwo Naukowe PWN, 2012. ISBN: 978-83-01-16911-4.