2. domača naloga: Stiskanje podatkov

Tokrat se bomo posvetili problemu brezizgubnega stiskanja podatkov [1]. Izdelali bomo lasten program za stiskanje datotek, ki bo deloval na podoben način kot vam zagotovo znane aplikacije (WinZip, WinRAR, gzip, 7-Zip, compress, bzip2, pkzip, ...).

Poznamo veliko postopkov, kako učinkovito zapisati nek niz znakov. Na predavanjih ste jih spoznali nekaj, ki temeljijo predvsem na informacijski teoriji: Shannonovo, Shannon-Fanojevo, aritmetično in Huffmanovo kodiranje. Ogledali ste si tudi tri postopke, ki kodirajo na podlagi slovarja (RLE, LZ77 in LZW) in algoritem DEFLATE, ki v grobem združuje LZ77 in Huffmanovo kodiranje. Obstaja seveda še veliko drugih pristopov (Rice, Golomb, Burrows-Wheeler, LZMA, ...). Pri tej nalogi se bomo osredotočili na postopek LZW.

Algoritem LZW

Algoritem Lempel-Ziv-Welch [2] opisuje brezizgubni postopek za stiskanje podatkov (avtorji: Abraham Lempel, Jacob Ziv in Terry Welch). Algoritem je preprost za implementacijo, hiter in razmeroma dobro stisne podatke. Uporablja se npr. v datotekah GIF, TIFF in PDF ter v aplikacijah *compress* in *uncompress*, ki sta del operacijskega sistema FreeBSD.

LZW - kodiranje

```
definiraj začetni slovar znakov z in kodnih zamenjav k
N = ''    # prazen niz
ponavljaj, dokler so znaki na vhodu:
    preberi naslednji znak z
    če je N+z v slovarju:
        N = N+z
    drugače:
        izpiši kodno zamenjavo k niza N
        dodaj N+z v slovar
        N = z
izpiši kodno zamenjavo k niza N
```

LZW - dekodiranje

```
definiraj začetni slovar znakov z in kodnih zamenjav k
preberi kodno zamenjavo k
v slovarju poišči niz N, ki ustreza k
izpiši N
K = N
ponavljaj, dokler so znaki na vhodu:
    preberi naslednjo kodno zamenjavo k
    če je k v slovarju:
        v slovarju poišči pripadajoči niz N
    sicer:
        N = K+K[0]
    izpiši N
    v slovar dodaj K+N[0]
    K = N
```

Osnovna naloga

Napišite funkcijo z imenom `naloga2` v programskem jeziku Python. Funkcija mora implementirati kodiranje in dekodiranje sporočil na osnovi algoritma LZW. Dogovor je, da začetni slovar vsebuje vse 8-bitne vrednosti (0-255), torej ima 256 vnosov. Kodne zamenjave v slovarju se začnejo z 0. Največje število vnosov v slovar je omejeno na $2^{12} = 4096$, kar pomeni, da bi bile vse kodne zamenjave v praksi zapisane z 12 biti. Kot vhodni argument funkcija sprejme seznam znakov `vhod`, ki predstavlja vhodno sporočilo, in celo število `nacin`, ki pove ali vhod kodiramo (0) ali ga dekodiramo (1). Izhodna argumenta funkcije sta dva: zakodirano/odkodirano sporočilo `izhod` in kompresijsko razmerje `R`. Glede na vhodni argument `nacin` si oglejmo, kaj predstavljata:

- Če `nacin = 0`: seznam 8-bitnih znakov `vhod` zakodiramo v `izhod`, ki je seznam ustreznih kodnih zamenjav. Kompresijsko razmerje `R` se izračuna kot $\frac{|\text{izhod}|*12}{|vhod|*8}$, kjer je $|x|$ dolžina seznama x.
- Če `nacin = 1`: `vhod` vsebuje zakodirano sporočilo, t.j. kodne zamenjave, za katere predpostavljamo, da so 12-bitne. Odkodiramo ga v `izhod`, ki je seznam znakov, za katere se delamo, da so 8-bitni. Kompresijsko razmerje `R` se izračuna kot $\frac{|\text{vhod}|*12}{|izhod|*8}$, kjer je $|x|$ dolžina seznama x.

Prototip funkcije:

```
def naloga2(vhod: list, nacin: int) -> tuple[list, float]:
    """
    Izvedemo kodiranje ali dekodiranje z algoritmom LZW.
    Zacetni slovar vsebuje vse 8-bitne vrednosti (0-255).
    Najvecja dolzina slovarja je 4096.

    Parameters
    ----------
    vhod : list
        Seznam vhodnih znakov: bodisi znaki abecede
        (ko kodiramo) bodisi kodne zamenjave
        (ko dekodiramo).
    nacin : int
        Stevilo, ki doloca nacin delovanja:
            0: kodiramo ali
            1: dekodiramo.

    Returns
    -------
    (izhod, R) : tuple[list, float]
        izhod : list
            Ce je nacin = 0: "izhod" je kodiran "vhod"
            Ce je nacin = 1: "izhod" je dekodiran "vhod"
        R : float
            Kompresijsko razmerje
    """

    izhod = []
    R = float('nan')
    return (izhod, R)
```

Testni primeri

Na učilnici se nahaja arhiv `tis-naloga2.zip`, ki vsebuje tri testne primere sporočil, za katere imate podane vhode in izhode. Primeri so podani v obliki datotek `.json`. Priloženo imate tudi funkcijo `test_naloga2`, ki jo lahko uporabite za preverjanje pravilnosti rezultatov, ki jih vrača vaša funkcija. Pri testiranju vaše funkcije upoštevajte, da je rezultat pravilen, če se `izhod` popolnoma ujema z rešitvami; dovoljeno odstopanje za `R` je 10^{-6}. Izvajanje vaše funkcije je časovno omejeno na 30 sekund.

Namigi

Python ima že vgrajeno podatkovno strukturo slovar (*dictionary*), ki vam bo prišla zelo prav pri reševanju naloge.

Dodatna naloga - tekmovanje

Osnovno nalogo, ki sloni na algoritmu LZW nadgradite tako, da bo z njo mogoče dejansko stisniti in nazaj razširiti poljubno datoteko. Implementirajte funkcijo `naloga2_tekma`, ki bo vsebovala tri argumente: pot do vhodne datoteke, ki jo želimo stisniti ali razširiti; pot do izhodne datoteke, v katero se bo zapisal rezultat; in način delovanja - stiskanje (0) ali razširjanje (1). Pričakovan rezultat stiskanja je datoteka, ki je manjša od vhodne; pričakovan rezultat razširjanja pa je datoteka, identična izvorni. Cilj je torej implementacija lastnega programa za stiskanje po vzoru programov 7-Zip, WinRAR, WinZip itd. Algoritmu za stiskanje na osnovi LZW lahko dodate poljubne nadgradnje, da izboljšate kompresijsko razmerje. Nekaj predlogov:

- dinamično prilagajanje slovarja glede na vsebino datoteke pri algoritmu LZW,
- dinamično prilagajanje dolžine kodnih zamenjav pri algoritmu LZW (brez odvečnih bitov v zapisu),
- učinkovit način zapisa glave datoteke,
- ideje iz algoritma DEFLATE,
- predprocesiranje s transformacijami Burrows-Wheller, RLE ali MTF,
- kombinacija s Huffmanovim algoritmom za stiskanje.

Prototip funkcije:

```
def naloga2_tekma(dat_vhod: str, dat_izhod: str, nacin: int) -> float:
    """
    Izvedemo kodiranje ali dekodiranje datoteke z algoritmom LZW
    in morebitnimi izboljsavami.

    Parameters
    ----------
    dat_vhod : str
        Pot do vhodne datoteke

    dat_izhod : str
        Pot do izhodne datoteke;
        ce datoteka se ne obstaja, jo ustvari; ce obstaja, jo povozi.

    nacin : int
        Nacin delovanja: kodiramo (0) ali dekodiramo (1).

    Returns
    -------
    R : float
        Kompresijsko razmerje
    """

    R = float('nan')
    return R
```

Prototip funkcije `naloga2_tekma` in nabor testnih datotek sta na voljo na spletni učilnici v arhivu `tis-naloga2-tekma.zip`. Na podlagi tega testnega nabora bomo priredili tekmovanje v stiskanju, kjer si lahko prislužite dodatne točke pri končni oceni predmeta. Vašo rešitev za tekmovanje oddajte preko posebnega obrazca na spletni učilnici (*Tekmovanje: oddaja rešitve*). Oddane naloge bomo razvrstili glede na povprečno kompresijsko razmerje, ki je v tem primeru definirano kot razmerje med velikostjo vhodne in izhodne datoteke v bajtih. Dodatne točke h končni oceni bomo dodelili po naslednjem kriteriju:

1. mesto: +10 %,
2. mesto: +8 %,
3. mesto: +5 %.

Ostali udeleženci tekmovanja dobite +3 % h končni oceni pri predmetu. Upoštevali bomo samo tiste oddane naloge, pri katerih je povprečno kompresijsko razmerje manjše od 1 (povprečna velikost stisnjenih datotek je manjša od povprečne velikosti izvornih datotek) in znajo datoteko stisniti in tudi pravilno razširiti. **Tisti, ki boste oddali izboljšano različico, morate prav tako oddati tudi rešitev osnovne naloge.**

Literatura

[1] D.G. Luenberger: Information Science, Princeton University, pogl. 4, 2006.

[2] T. A. Welch: A Technique for High-Performance Data Compression. Computer, zv. 17, št. 6, str. 8-19, 1984.