Rekonstrukcija nepoznate strukture složenih podataka na primjeru Python pickle

Vareško, Lucia

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike u Puli

LUCIA VAREŠKO

REKONSTRUKCIJA NEPOZNATE STRUKTURE SLOŽENIH PODATAKA NA PRIMJERU PYTHON PICKLE

Završni rad

Pula, 23.rujna, 2021. godine

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike u Puli

LUCIA VAREŠKO

REKONSTRUKCIJA NEPOZNATE STRUKTURE SLOŽENIH PODATAKA NA PRIMJERU PYTHON PICKLE

Završni rad

JMBAG: 0303082397, redovita studentica
Studijski smjer: Informatika

Predmet: Statistika
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc.dr.sc Siniša Miličić

Pula, 23.rujna, 2021. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisana Lucia Vareško, kandidatkinja za prvostupnicu informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

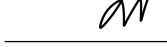
U Puli, 23.09.2021.

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, Lucia Vareško dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj Završni rad pod nazivom „Rekonstrukcija nepoznate strukture složenih podataka na primjeru Python pickle“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 23.09.2021.

Potpis



Sadržaj

Uvod

Polazna točka formiranja teme ovog završnog rada bio je projekt u kojemu sudjelujemo pof. Siniša Miličić, kolegica Lucija Babić i ja. Naime, projekt ima za cilj rekonstruirati podatke jednog nepoznatog podatkovnog modela i migrirati ih u neku relacijsku bazu podataka. Podaci koje je potrebno rekonstruirati dobiveni su iz jednog zastarjelog sustava neke tvrtke. Govorimo o količini od otprilike 12GB podataka koji su spremljeni u nekom stablu datoteka s nečitkom strukturom. Postoji aplikacija koja te podatke čita, ali nema mogućnost izvoza podataka u neki oblik prihvatljiv za provođenje detaljnih statističkih analiza. Planirani pristup ovom projektu je analiza načina na koji su organizirani podaci koje ta aplikacija čita, zatim kreiranje algoritma koji će te podatke uspješno iščitati i dohvatiti te migracija podataka u bazu podataka.

S ciljem postizanja što boljih rezultata te boljeg razumijevanja same problematike ovog projekta odlučeno je napraviti rekonstrukciju nepoznate strukture složenih podataka na primjeru Python pickle. Točnije, rekonstrukcija podataka napravljena je na datoteci koju je profesor izradio na način da je ona neupotrebljiva za klasično otvaranje. Ovim pristupom, na relativno poznatoj strukturi, izgrađen je primjer za ozbiljno nepoznatije strukture poput onih iz prethodno opisanog projekta.

1 Formulacija zadatka

Dobivena je Python pickle datoteka koja je neupotrebljiva za klasično otvaranje. Datoteka se sastoji od sljedećih elemenata:

- niz (x,y)
- koordinata, realnih brojeva
- niz (r,g,b) integer podataka, struktura neke rgb slike
- nekakav informacijski škart

Zadatak je pronaći koordinate i strukturu rgb slike te rekonstruirati ih. Uspješna rekonstrukcija podataka bi značila prikaz prvog niza vrijednosti kao točke u ravnini (scatterplot) i rekonstrukcija slike iz zadanih podataka o bojama, tj. drugog niza vrijednosti.

Osim toga poznato je da se traženi podaci nalaze između informacijskog škarta, tj. da imamo neku količinu informacijskog škarta, zatim imamo korisne podatke, nakon čega se ponovno pojavljuje informacijski škart te na posljetku korisni podaci te informacijski škart.

2 Metode rješavanja

Za uspješnu rekonstrukciju podataka iz prethodno opisane datoteke korištene su različite metode koje su nam većinom generirale korisne podatke, no bilo je i onih koje nisu bile uspješne. Upravo te manje uspješne metode rezultirale su ponovnim promišljanjem i prim-jenjivanjem metoda koje nisu unaprijed bile predviđene.

Glavni fokus je na različitim metodama vizualizacije podataka iz datoteke.Ideja je da nam vizualizacije pomognu u otkrivanju detaljnije strukture same datoteke te da nam ukažu na dijelove u kojima bi se mogli naći korisni podaci.

Za kreiranje vizualizacija korišten je programski jezik Python te njegova biblioteka Pillow, odnosno njezin modul poznat pod nazivom Image [Cla15]. Ovaj modul omogućava kreiranje slika iz polja uz prethodno definiran colormap za čiju je definiciju korištena je biblioteka Matplotlib [Hun07]. Osim toga, za lakše manipuliranje poljima korištena je i biblioteka NumPy [Har+20].

[1]:
```
import numpy as np
import matplotlib.colors
from PIL import Image
```

[2]:
```
colors = {
  1: ["#000000", "#83eca4", "#493ef6", "#83eca4", "#ff4000", "#ffffff"],
  2: ["#ff4000", "#000000", "#83eca4"],
  3: ["#000000", "#493ef6", "#ffffff", "#c900ff", "#ff4000"],
  4: ["#000000", "#83eca4", "#493ef6", "#ff4000", "#ffffff", "#ffcc00",␣
 ↪"#c900ff"]
}
```

[3]:
```
bounds = {
    1: [0,0.99,31,127,128,255,255.5],
    2: [0,0.99,255,255.5],
    3: [0,0.99,1.99,2.99,3.99,4.99],
    4: [0,0.99,1.99,2.99,3.99,4.99,5.99,6.99]}
```

Radi lakše organizacije koda napravljena su dva riječnika, colors i bounds. Prvi riječnik sadrži polja heksadecimalnih kodova boja koje će se koristiti prilikom vizualizacija, a drugi riječnik sadrži intervale koji služe za kreiranje indeksa colormap-a.

Osim toga, definirane su i dvije funkcije koje će biti korištene prilikom generiranja svih vizualizacija.

[4]:
```
def image_settings(bounds, colors):
    cmap = matplotlib.colors.ListedColormap(colors)
    norm = matplotlib.colors.BoundaryNorm(bounds, len(colors))
    return (cmap,norm)
```

Funkcija image_settings prima dva parametra, bounds i colors, na temelju kojih kreira colormap i generira njegove indekse, tj. definira rubove spremnika. Podaci koji spadaju u spremnik preslikavaju se u boju s istim indeksom.

[5]:
```
def image_maker (array, img_name, bounds, colors):
    cmap, norm = image_settings(bounds, colors)
    len_array = len(array)
    rows = int(len_array / 256)
    cols = 256


    data = np.resize(array,  rows * 256).reshape(rows, cols)
    im = Image.fromarray(np.uint8(cmap(norm(data))*255))
    im.save(img_name)
```

Funkcija image_maker prima četiri parametra:

- array - polje iz čijih se vrijednosti kreira slika
- img_name - naziv pod kojim će slika biti pohranjena u memoriji računala
- bounds - parametar koji se proslijeđuje funkciji image_settings
- colors - parametar koji se proslijeđuje funkciji image_settings

Na temelju dobivenih parametara funkcija kreira sliku i pohranjuje ju u internu memoriju računala.

2.1 Vizualizacija na razini bajtova

Vrlo jednostavna i efikasna vizualizacija koja podrazumijeva bojanje odgovarajućih vrijednosti bajta. Iz datoteke čitamo bajt po bajt pri čemu svaki bajt može imati 256 različitih vrijednosti, tj. vrijednosti u rasponu od 0 do 255. Kako bi bolje razumjeli distribuciju bajtova u datoteci, vrijednosti svrstavamo u jednu od 5 klasa. [Che16]

Naziv klase	Raspon vrijednosti	Boja
0xFF	255	Bijela
0x00	0	Crna
Printable characters	32-126	Plava
Control characters	1-31 + 127	Zelena
Extended characters	128-254	Crvena

Tablica 1: Tablica klasa korištenih prilikom vizualizacije na razini bajtova

Tablica prikazuje definirane klase, raspon vrijednosti koji ulazi u te klase te boju kojom se pripadnost klasi obilježava u vizualizaciji. To su zapravo uobičajene vrste klasa ASCII standarda.

[6]:
```
file = open('task.pickle', 'rb')
file_array = list(file.read())
```

[7]:
```
image_maker(file_array[64:], 'vizualizacija_bez_headera_64.png',␣
↪bounds[1], colors[1])
```

Otvaramo datoteku u kojoj su nam pohranjene koordinate i rgb vrijednosti slike, čitamo cijeli njen sadržaj i pohranjujemo ga u varijablu file_array. Za kreiranje vizualizacije pozivamo funkciju image_maker te koristimo intervale i boje iz prethodno opisanih klasa.

2.2 Vizualizacija vrijednosti 0 i 255

Vizualizacija podrazumijeva bojanje bajtova ukoliko je njihova decimalna vrijednost 0 ili 255. Uz pretpostavku da su u datoteci traženi podaci pohranjeni na način da zauzimaju 8 bajtova uz njih će se pojaviti više vrijednosti 0 ili 255 iz razloga što podaci koje tražimo nisu veliki brojevi. Ovakvim pristupom pokušat ćemo odrediti regije u kojima bi se mogli naći podaci, tj. integeri ili floatovi.

[8]:
```
image_maker(file_array, 'vizualizacija_0_255.png', bounds[2], colors[2])
```

Prilikom kreiranja ove vizualizacije koristimo crvenu boju za vrijednost 0, zelenu za vrijednost 255 te crnu za sve ostale vrijednosti.

2.3 Istraživanje pickle datoteke

Za daljnje analize potrebno je razumijeti na koji su način podaci pohranjeni u pickle datoteci. Zanima nas koliko je bajtova rezervirano za pohranu vrijednosti tipa integer i float te kojim se redoslijedom pohranjuju bajtovi u memoriji. Redoslijed pohrane bajtova može biti big endian pri čemu se najznačajniji bajt pohranjuje na najmanjoj memorijskoj adresi ili little endian gdje se najznačajniji bajt pohranjuje na najvećoj memorijskoj adresi. Osim toga, pretpostavljamo da su traženi podaci pohranjeni kao lista tuple-ova pa nas zanima na koji način su pohranjene takve liste.

[9]:
```
import pickle
import struct
import io
```

[10]:
```
buffer = io.BytesIO()
pickle.dump([(0.00001, -100000.2),(1.1, 20.5)], buffer)
buffer.seek(0)
buffer.read()
```

[10]:
```
b'\x80\x04\x95-\x00\x00\x00\x00\x00\x00\x00]\x94(G>\xe4\xf8\xb5\x88\xe3h\xf1G\xc
0\xf8j\x033333\x86\x94G?
 ↪\xf1\x99\x99\x99\x99\x99\x9aG@4\x80\x00\x00\x00\x00\x00\
x86\x94e.'
```

U prvom primjeru pohranjujemo listu koordinata realnih brojeva. Uz pomoć biblioteke io kreiramo jedan buffer koji očekuje objekte slične bajtovima [Van20]. Zatim iz biblioteke Pickle pozivamo funkciju dump koja Python objekte pretvara u stream bajtova koji ubacuje u prethodno kreirani buffer [Van20]. Postavimo se na početak buffera te pročitamo sve zapise iz njega.

Kako bi mogli analizirati dobiveni niz bajtova, pokušat ćemo uz pomoć biblioteke Struct vrijednosti koordinata pretvoriti u niz bajtova. Biblioteka Struct omogućava nam pretvaranje niza bajtova u izvorne vrste podataka u Python-u i obrnuto [Van20].

[11]:
```
struct.pack('dddd',0.00001,-100000.2,1.1,20.5)
```

[11]:
```
b'\xf1h\xe3\x88\xb5\xf8\xe4>3333\x03j\xf8\xc0\x9a\x99\x99\x99\x99\x99\xf1?
 ↪\x00\x
00\x00\x00\x00\x804@'
```

Funkciji pack prosljeđujemo format prema kojemu će se odviti pretvorba u niz bajtova te vrijednosti koje želimo pretvoriti. Format podrazumijeva da se sve vrijednosti tretiraju kao tip podataka float veličine 8 bajtova pri čemu je zadani redoslijed bajtova little endian.

Ako usporedimo dobiveni niz bajtova sa nizom bajtova pohranjenim u buffer, možemo vidjeti da su vrijednosti tamo zapisane obrnutim redoslijedom iz čega zaključujemo da će u našoj datoteci realni brojevi zauzimati 8 bajtova i redoslijed njihove pohrane biti će big endian.

Osim toga, možemo primijetiti da se u bufferu prije bajtova realnih brojeva pojavljuje slovo G i nakon svake (x,y) koordinate pojavljuje se niz bajtova \x86 \x94 koji vjerojatno označava da se radi o tuple-u koji ima dvije vrijednosti.

Postupak ponavljamo kako bi vidjeli što se događa sa cijelim brojevima u rasponu od 0 do 255 koji predstavljaju rgb vrijednosti slike.

[12]:
```
buffer = io.BytesIO()
pickle.dump([(187, 111,  53),(221, 142,  76)], buffer)
buffer.seek(0)
buffer.read()
```

[12]:
```
b'\x80\x04\x95\x15\x00\x00\x00\x00\x00\x00\x00]\x94(K\xbbKoK5\x87\x94K\xddK\x8eK
L\x87\x94e.'
```

[13]:
```
struct.pack('BBBBBB',187,111,53,221,142,76)
```

[13]:
```
b'\xbbo5\xdd\x8eL'
```

Budući da jedan bajt može poprimiti 256 različitih vrijednosti, tj. vrijednosti u rasponu od 0 do 256, pretpostavljamo da će rgb vrijednosti biti pohranjene u jednom bajtu. Funkciji pack prosljeđujemo format koji podrazumijeva da se vrijednosti tretiraju kao tip podataka integer veličine jednog bajta pri čemu je zadani redoslijed pohrane bajtova little endian.

Ako usporedimo dobiveni niz bajtova sa nizom bajtova pohranjenim u buffer, možemo vidjeti da se vrijednosti u potpunosti preklapaju iz čega zaključujemo da će u našoj datoteci rgb vrijednosti zauzimati jedan bajt.

Osim toga, možemo primijetiti da se u bufferu prije bajtova koji predstavljaju rgb vrjednosti pojavljuje slovo K i nakon svake (r,g,b) trojke pojavljuje se niz bajtova \x87 \x94. Budući da se bajt x94 pojavljivao i u prethodnom primjeru, pretpostavljamo da samo bajt x87 označava da se radi o tuple-u koji ima tri vrijednosti.

2.4 Vizualizacija pickle oznaka

Na temelju prethodno navedenih zaključaka napravljena je jedna jednostavna vizualizacija koja nam daje jako dobar uvid u strukturu same datoteke. Vizualizacijom se prikazuju mjesta na kojima se pojavljuju prethodno spomenute oznake.

[14]:
```
def char_finder (file_name):
    file = open(file_name, 'rb')
    file_size = len(list(file.read()))
    file.seek(64)
    bytes_read = file.read(1)
    np_array =  np.array([0] * (file_size - 64))
    i = 0

    while bytes_read:
        if bytes_read == b'\x86':
            np_array[i]= 1 # plava
        elif bytes_read == b'G':
             np_array[i]= 2 # bijela
        elif bytes_read == b'K':
            np_array[i]= 3 # ljubičasta
        elif bytes_read == b'\x87':
            np_array[i]= 4 # crvena

        i += 1
        bytes_read = file.read(1)
```

```
    file.close()
    return np_array
```

Funkcija char_finder traži neku od prethodno spomenutih oznaka na način da se kreće kroz datoteku bajt po bajt te ukoliko pronađe oznaku označava ju posebnom bojom.

Oznaka	Boja
x86	Plava
G	Bijela
K	Ljubičasta
x87	Crvena

Tablica 2: Tablica oznaka i pripadajućih boja za vizualizaciju pickle oznaka

Boje kojima se obilježavaju oznake u vizualizaciji prikazane su u tablici.

[15]:
```
char_array = char_finder('task.pickle')
```

[16]:
```
image_maker(char_array,'pickleChars.png', bounds[3], colors[3])
```

Kako bi napravili vizualizaciju prvo pozivamo funkciju char_finder i prosljeđujemo joj naziv datoteke u kojoj su pohranjeni traženi podaci. Sljdeći korak je generirati samu vizualizaciju pozivanjem funkcije image_maker.

2.5 Vizualizacija integera i floatova

Budući da su tipovi podataka koje tražimo poznati, možemo napraviti vizualizaciju iz koje će biti vidljivo u kojim se dijelovima datoteke nalaze traženi tipovi podataka i na taj ćemo način suziti polje pretrage. Za identifikaciju suvislih integera ili floatova koristimo biblioteku Struct koja omogućava pretvaranje niza bajtova u izvorne vrste podataka u Python-u i obrnuto.

Drugim riječima, krećemo se kroz datoteku i čitamo nizove bajtova određene veličine te ih uz pomoć alata struct.unpack pokušamo pretvoriti u integere ili floatove. Suvisli integer biti će nesuvisli float i obrnuto.

[17]:
```
def int_and_float_finder (file_name):
    file = open(file_name, 'rb')
    file_size = len(list(file.read()))
    file.seek(64)
    bytes_read = file.read(1)
    np_array =  np.array([0] * (file_size - 64))

    while bytes_read:
        if(bytes_read == b'\x86'):
            file.seek(-19,1)
            bytes_read = file.read(1)

            if bytes_read == b'G' : # float 8-bytes
                for x in range(2):
                    j = file.tell() - 65
                    val = struct.unpack('>d', file.read(8))

                    if abs(val[0]) < 10:
                        np_array[j:j+8] = 1  # zelena
                    elif abs(val[0]) < 100:
                        np_array[j:j+8] = 2  # plava
                    elif abs(val[0]) < 1000:
                        np_array[j:j+8] = 3  # crvena
                    elif abs(val[0]) < 10000:
                        np_array[j:j+8] = 4  # bijela
                    elif abs(val[0]) < 100000:
                        np_array[j:j+8] = 5 # zuta
                    file.seek(1, 1)
            else:
```

```
                file.seek(18,1)
        if bytes_read == b'K':  # int 1-byte
            file.seek(1, 1)
            val = struct.unpack('<B', file.read(1))
            if 0 < val[0] < 256:
                np_array[file.tell() - 66] = 6  # ljubičasta
                file.seek(-2,1)

        bytes_read = file.read(1)

    file.close()
    return np_array
```

Funkcija int_and_float_finder traži suvisle vrijednosti tipa integer ili float. U trenutku kada funkcija pročita oznaku za tuple koji ima dvije vrijednosti potrebno je vratiti se natrag za 19 bajtova, ako je na tom mjestu oznaka G pronađena je jedna koordinata (x,y). Funkcija zatim čita 8 bajtova, pretvara ih u float te provjerava upada li dobivena vrjednost između 0 i 100000, ukoliko upada označava ju određenom bojom. Postupak se ponavlja još jednom.

Ako funkcija naiđe na oznaku K, to znači da je pronađen integer veličine jednog bajta. Provjerava se upada li taj integer između 0 i 256 te ukoliko upada pronašli smo jednu vrijednost iz niza (r,g,b) koju označavamo određenom bojom.

[18]:
```
float_int_array = int_and_float_finder('task.pickle')
```

[19]:
```
image_maker(float_int_array,'vizualizacija_float_int.png', bounds[4],␣
↪colors[4])
```

Kako bi napravili vizualizaciju prvo pozivamo funkciju int_and_float_finder i prosljeđujemo joj naziv datoteke u kojoj su pohranjeni traženi podaci. Sljdeći korak je generirati samu vizualizaciju pozivanjem funkcije image_maker.

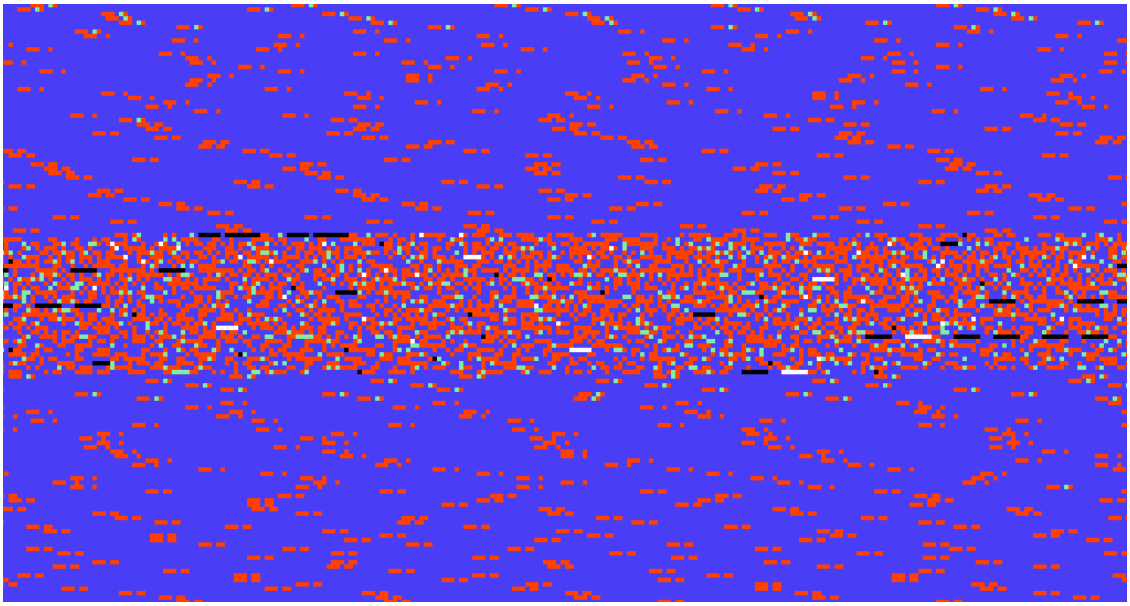
3 Analiza vizualizacija

Prethodno opisane metode rezultirale su raznim vizualizacijama koje je potrebno analizirati kako bi dobili informacije potrebne za uspješnu rekonstrukciju podataka iz datoteke. Ukupan broj dobivenih vizualizacija je 4, no one neće biti prikazane u potpunosti. Prikazat će se samo bitni dijelovi svake pojedine vizualizacije iz kojih možemo izvući korisne informacije.

3.1 Vizualizacija na razini bajtova

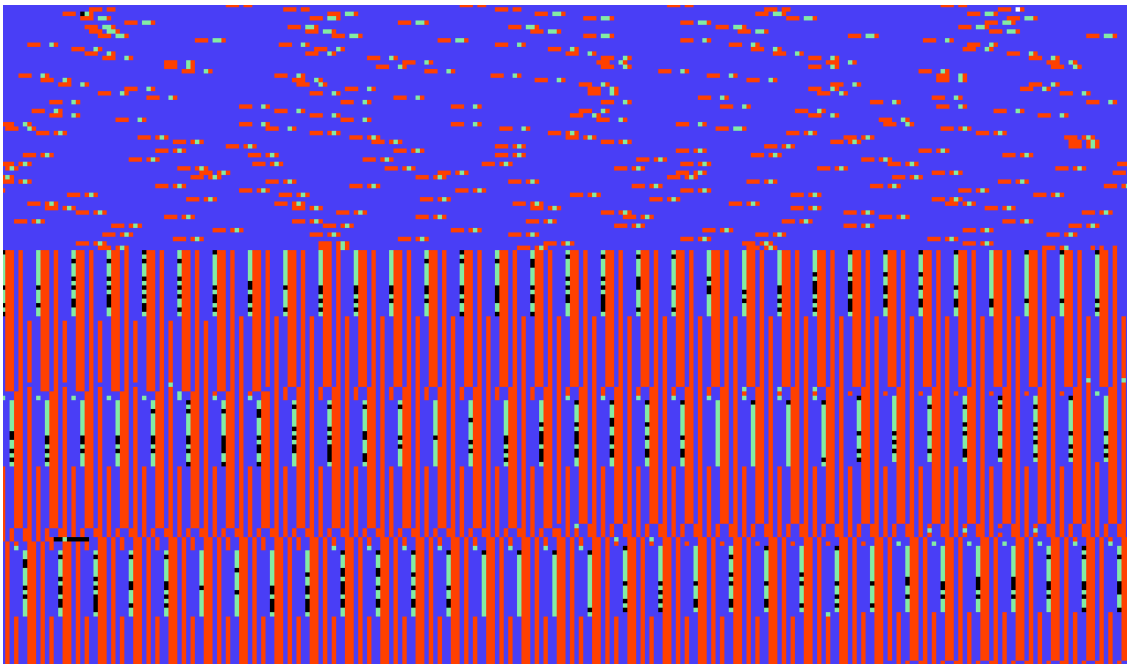
Vizualizacija na razini bajtova potvrđuje činjenicu da su traženi podaci okruženi informacijskim škartom, tj. nekakvim stringovima. Iz vizualizacije također možemo vidjeti da je datoteka uredno strukturirana, odnosno da ima nekakav pravilni ritam.

Podsjetimo se, plava boja označava klasu Printable characters, zelena boja označava klasu Control characters i crvena boja označava klasu Extended characters. Vrijednosti 0 i 255 označene su crnom i bijelom bojom.



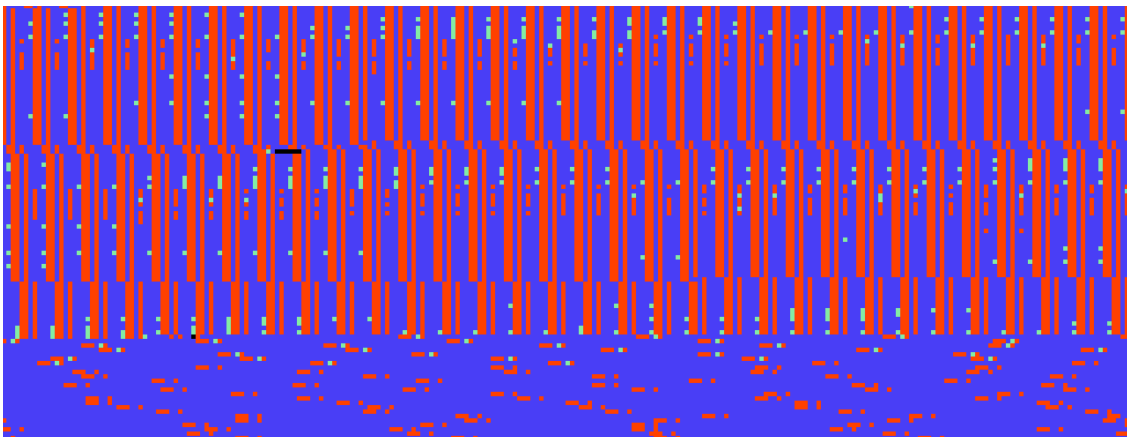
Slika 1: Vizualizacija na razini bajtova - prva struktura

Slika 1 prikazuje prvi dio datoteke sa nekakvom strukturom okruženom informacijskim škartom, tj. stringovima koji su prikazani plavom bojom. Strukturirani dio bi mogao sadržavati koordinate, tj. vrijednsoti tipa float, ali ne možemo to sa sigurnošću utvrditi.



Slika 2: Vizualizacija na razini bajtova - početak druge strukture

Sa sigurnošću možemo reći da Slika 2 na početku ima stringove nakon kojih slijedi jedan jako dugačak strukturirani dio koji vrlo vjerojatno sadrži rgb vrijednosti slike koju je potrebno rekonstruirati. To nam sugerira gotovo pravilno ponavljanje zelene, crvene i plave boje. Pravilno ponavljanje remeti crni niz koji vrlo vjerojatno predstavlja kraj nekakvog zapisa u datoteci.

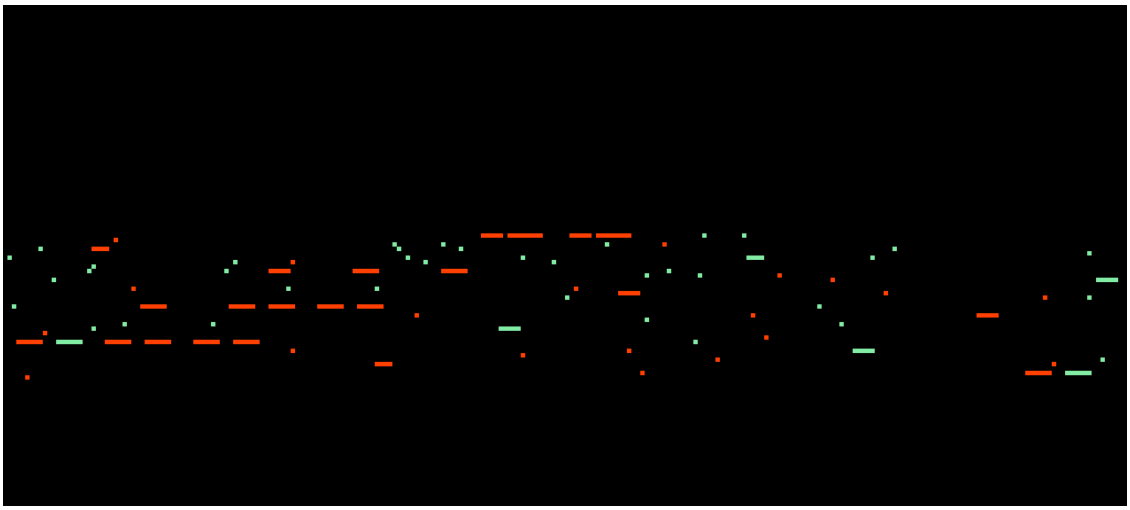


Slika 3: Vizualizacija na razini bajtova - kraj druge strukture

Slika 3 prikazuje kraj datoteke gdje vidimo da se nakon pravilnog ponavljanja opet pojavljuju stringovi.

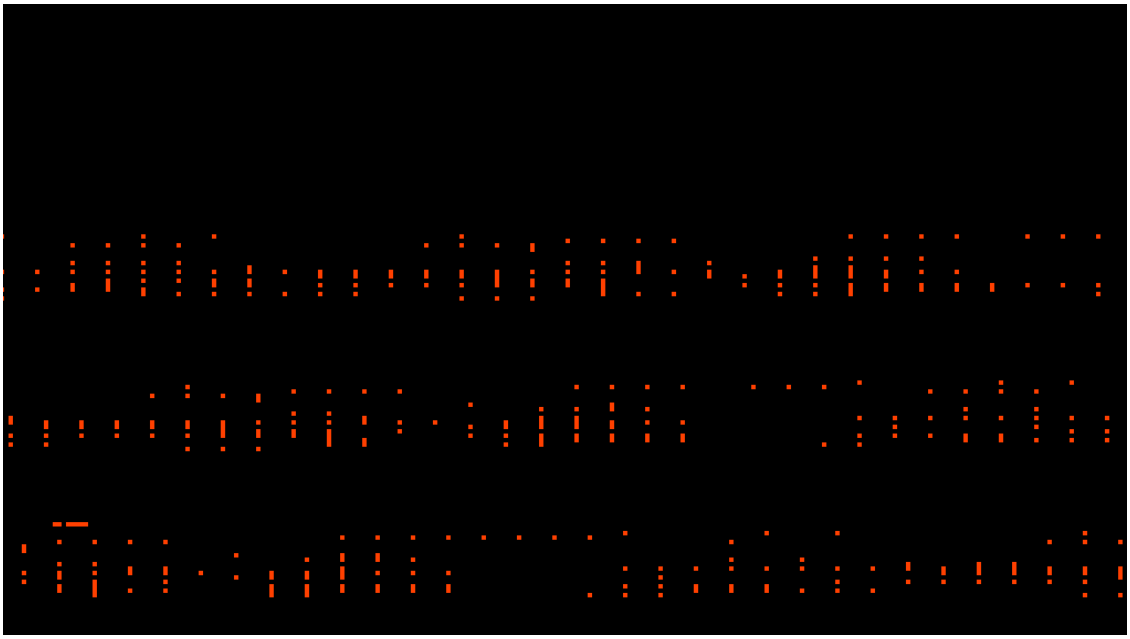
3.2 Vizualizacija vrijednosti 0 i 255

Cilj ove vizualizacije bio je pronaći mjesta u dadoteci na kojima se pojavljuju vrijednosti tipa integer i float, no ova nam ne govori puno. Nismo pronašli velik broj uzastopnih ponavljanja vrijednosti 0 ili 255 pa zaključujemo da naša pretpostavka ne vrijedi za datoteku sa nastavkom pickle, no to ne znači da za neku drugu vrstu datoteke ova metoda ne bi generirala vrijedne informacije.

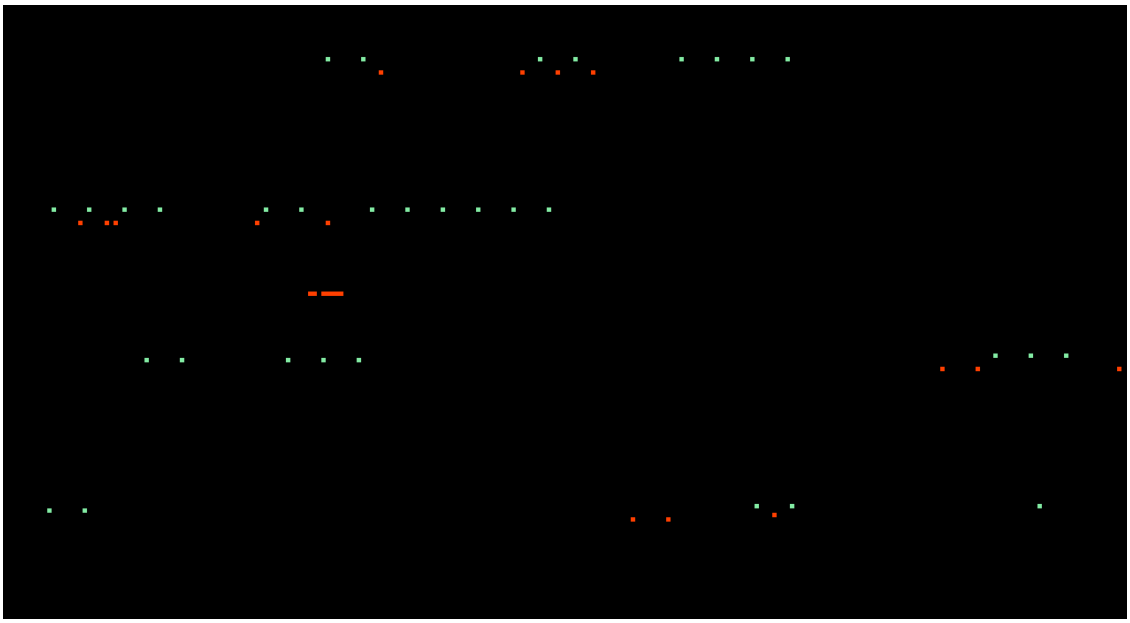


Slika 4: Vizualizacija vrijednosti 0 i 255 - početak datoteke

Slika 4 prikazuje prvi dio datoteke za koji smo iz prethodne vizualizacije pretpostavili da sadrži koordinate. Možemo vidjeti neki broj ponavljanja vrijednosti 0 i 255, ali i dalje nedovoljan.



Slika 5: Vizualizacija vrijednosti 0 i 255 - središnji dio datoteke



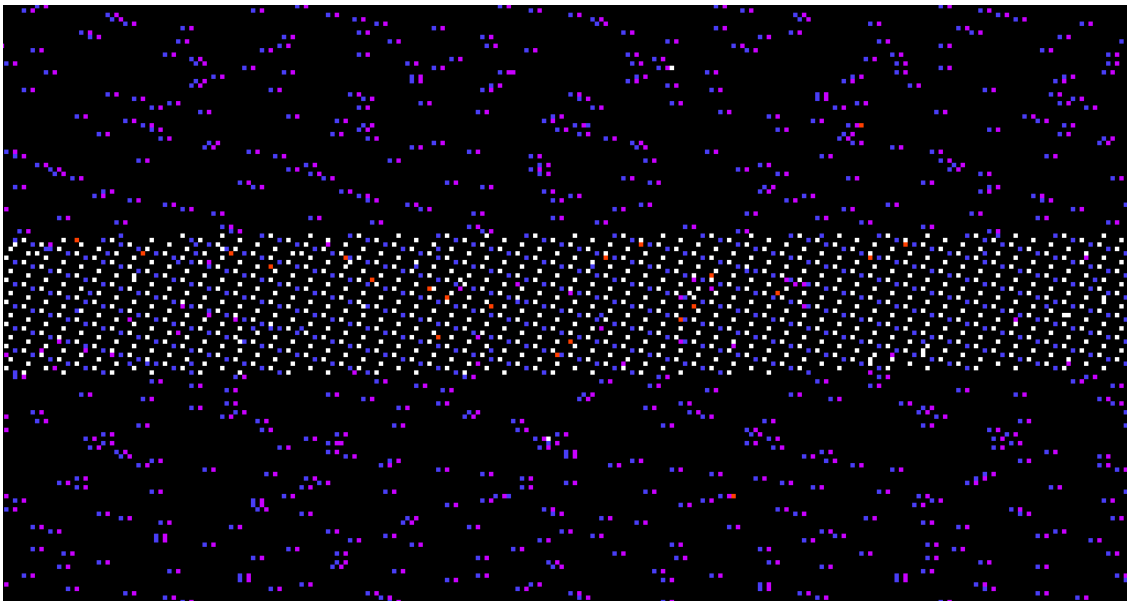
Slika 6: Vizualizacija vrijednosti 0 i 255 - kraj datoteke

Slika 5 i 6 prikazuju središnji i krajnji dio datoteke te iz njih možemo vidjeti da nema puno ponavljanja vrijednosti 0 i 255. Većina je vizualizacije zapravo crne boje pa dolazimo do zaključka da ova metoda nije baš uspješna na našem primjeru. Neuspjeh ove vizualizacije bio je poticaj za istraživanje načina pohrane u pickle datoteci.

3.3 Vizualizacija pickle oznaka

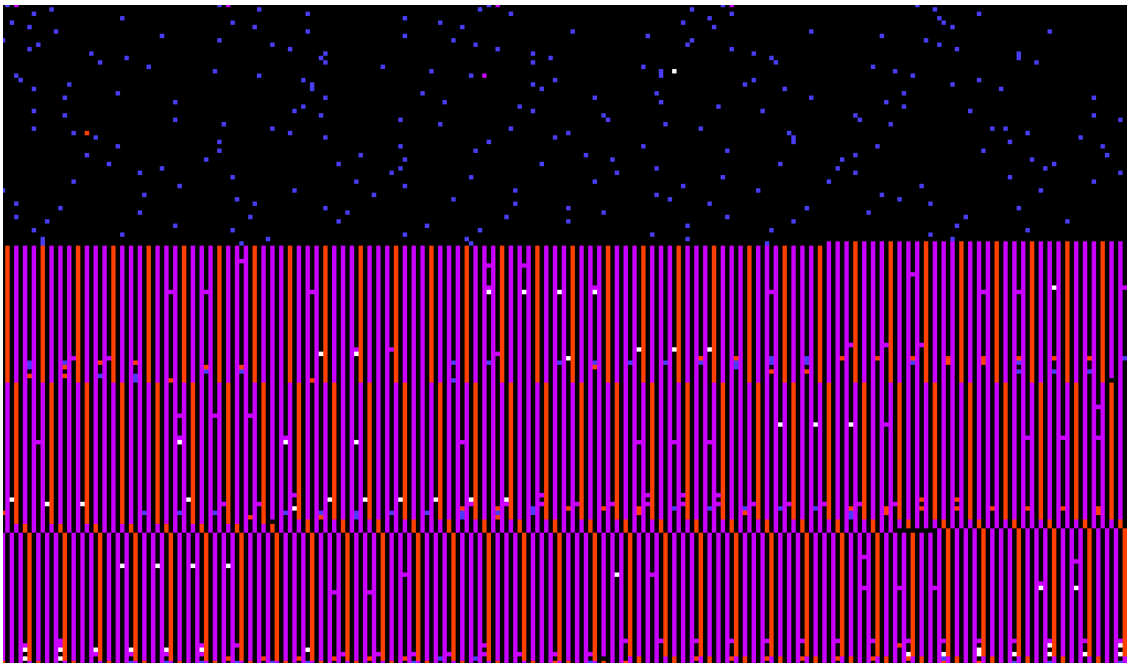
Istraživanjem načina pohrane u pickle datoteci zaključeno je da postoje oznake za vrijednosti tipa float i integer te za tauple koji ima dvije ili tri vrijednosti.

Prisjetimo se, plava boja označava tuple koji sadrži dvije vrijednosti, bijela boja označava vrijednost tipa float veličine 8 bajta, ljubičasta označava vrijednost tipa integer veličine jednog bajta i crvena boje označava tuple koji sadrži tri vrijednosti.



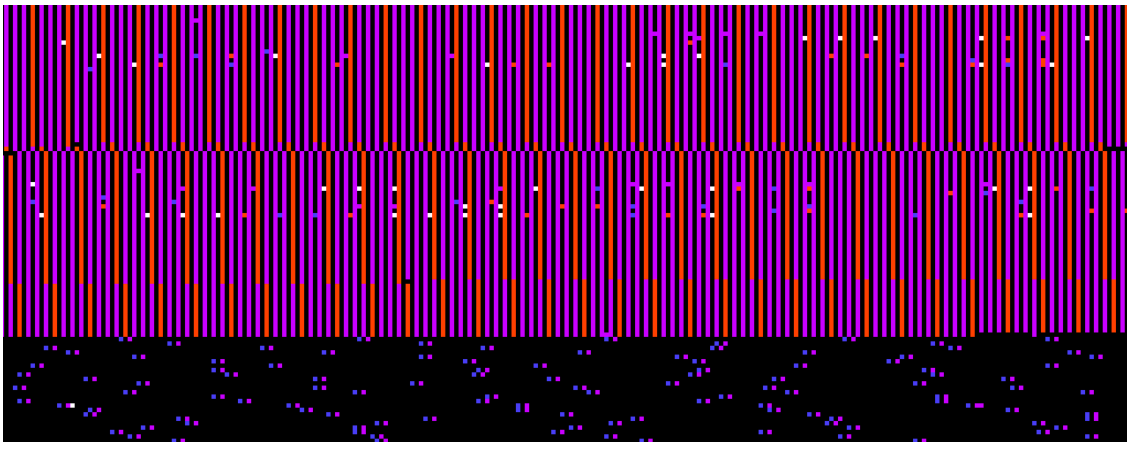
Slika 7: Vizualizacija pickle oznaka - prva struktura

Slika 7 prikazuje prvi dio datoteke u kojemu možemo vidjeti puno plave boje, no to ne znači da na svim tim mjestima imamo tuple koji sadrži dvije vrijednosti. Točnost identificiranja koordinata omogućit će nam i činjenica da se ispred 8-bajtne vrijednsoti tipa float pojavljuje bijela boja. Dakle, tražimo uzorak koji sadrži bijelu boju, 8 piksela crne boje pa ponovno bijelu boju i 8 piksela crne boje nakon čega slijedi plava boja kao oznaka za tuple. Možemo vidjeti da u središnjem dijelu slike imamo puno takvih ponavljanja pa sada sa sigurnošću možemo potvrditi da se radi o koordinatama.



Slika 8: Vizualizacija pickle oznaka - početak druge strukture

Slika 8 prikazuje isti dio datoteke kao i Slika 2 te na taj način potvrđuje prethodnu pretpostavku da u tom dijelu datoteke započinje niz rgb vrijednosti slike koju je potrebno rekonstruirati. Vidljivo je da imamo uzorak koji ima po 3 oznake za integere veličine jednog bajta označene ljubičastom bojom, između kojih se nalaze same vrijednsoti trenutno crne boje i na kraju imamo oznaku za tuple koji sadrži tri vrijednosti označenu crvenom bojom.



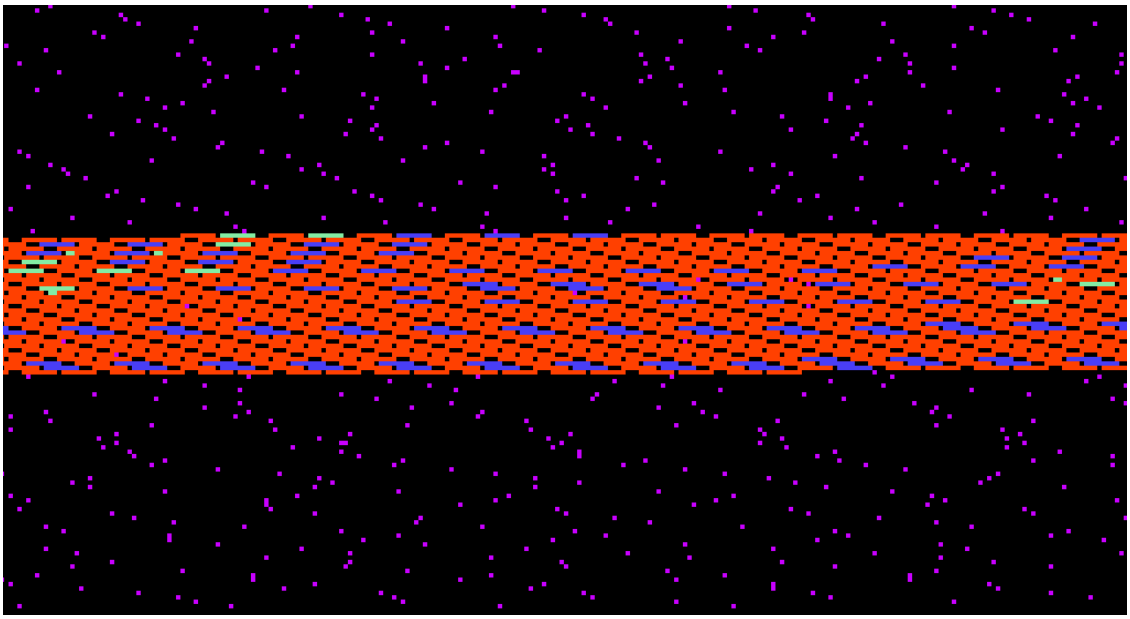
Slika 9: Vizualizacija pickle oznaka - kraj druge strukture

Slika 9 prikazuje kraj datoteke, tj. kraj niza rgb vrijednosti slike.

3.4 Vizualizacija integera i floatova

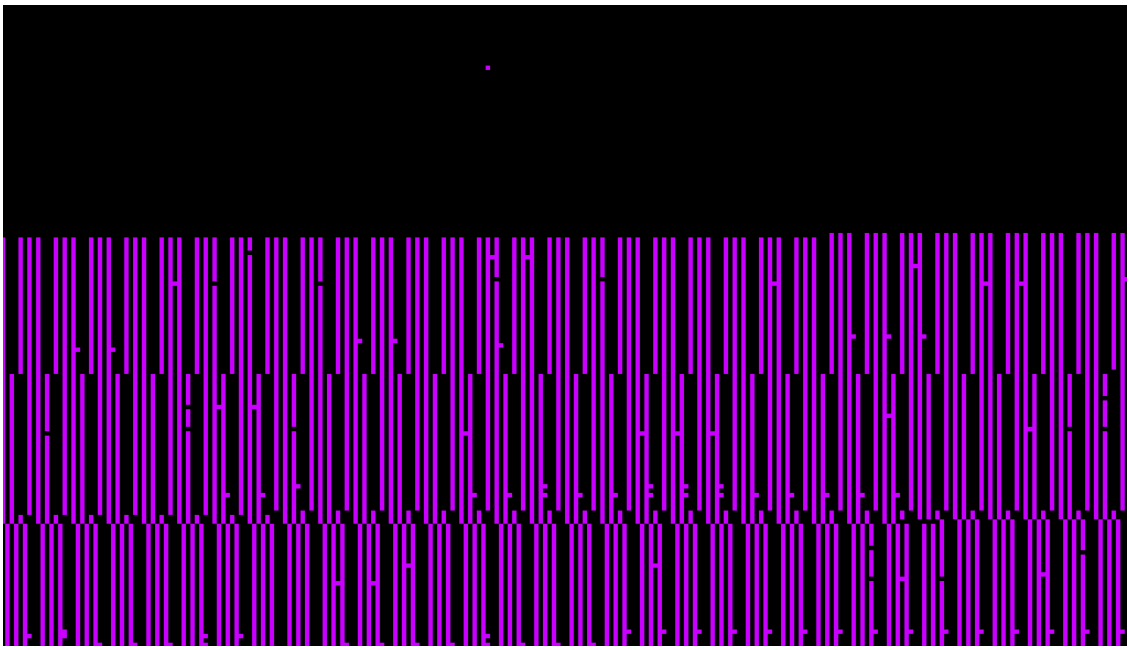
Posljednja vizualizacija je ujedno i posljednja potvrda prije početka ekstrakcije i rekonstrukcije traženih vrijednosti. Dodatno su potvrđene regije u kojima se nalazi niz koordinata i niz rgb vrijednosti slike te da su oni u potpunosti okruženi stringovima.

Podsjetimo se, ljubičasta boja označava integere veličine jednog bajta koji ulaze u raspon vrijednosti od 0 do 255 (uključujući i 255). Vrijednosti tipa float veličine 8 bajtova označavaju četiri boje: zelena, crvena, bijela i plava. Zelena boja označava vrijednosti koje ulaze u raspon od 0 do 10, crvena one koje ulaze u raspon od 10 do 1000, bijela one koje ulaze u raspon od 1000 do 10000 i plava one koje ulaze u raspon od 10000 do 100000.



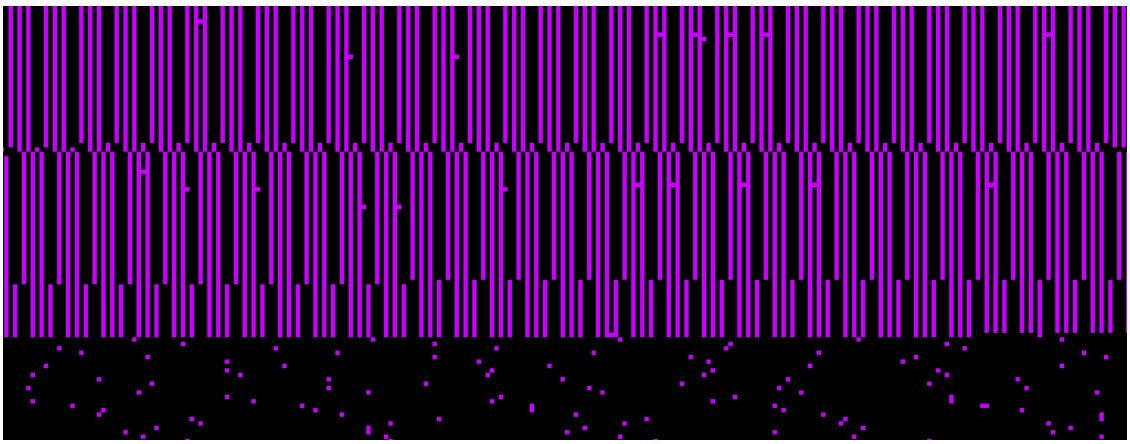
Slika 10: Vizualizacija integera i floatova - prva struktura

Na slici 10 vidimo ponavljanje boja koje označavaju vrijednosti tipa float te ljubičaste boje koja označava integere veličine jednog bajta, no nemamo tri uzastopna ponavljanja pa ih ne uzimamo u obzir kao kandidate za rgb vrijednosti slike.



Slika 11: Vizualizacija integera i floatova - početak druge strukture

Slika 11 lijepo prikazuje ponavljanje tri uzastopna integera uz pomak koji stvara prethodno spomenuti niz vrijednosti koji vjerojatno označava početak novog zapisa u datoteci.



Slika 12: Vizualizacija integera i floatova - kraj druge strukture

Slika 12 prikazuje kraj datoteke i kraj niza rgb vrijednosti slike, nakon kojeg se ponovno pojavljuju nekakvi integeri no njih ne uzimamo u obzir kao vrijednosti rgb slike jer nisu uzastopni.

4 Ekstrakcija i rekonstrukcija podataka

Iz metoda vizualizacije saznajemo informacije važne za ekstrakciju podataka iz datoteke. Informacije su specifične za datoteke sa nastavkom pickle, ali to ne znači da se metode ne bi mogle koristiti na drugim vrstama datoteka.

U nastavku slijedi ekstrakcija kooordinata i njihov prikaz u ravnini te ekstrakcija rgb vrijednosti slike te njezina rekonstrukcija.

4.1 Ekstrakcija koordinata i njihov prikaz u ravnini

Poznato je da pickle relne brojeve pohranjuje kao 8-bajtni big endian float ispred kojeg se nalazi slovo G te da tuple koji ima dvije vrijednosti označava bajtom x86. Sljedeći je zadatak identificirati takav uzorak te pohraniti pronađene vrijednosti u polje kako bi ih mogli prikazati u ravnini.

[20]:
```
def extract_floats (file_name):
    file = open(file_name, 'rb')
    file_size = len(list(file.read()))
    file.seek(64)
    bytes_read = file.read(1)
    f_array = []

    while bytes_read:
        if bytes_read == b'\x86':
            file.seek(-19,1)
            bytes_read = file.read(1)

            if bytes_read == b'G': # float 8-bytes
                for x in range(2):
                    val = struct.unpack('>d', file.read(8))
```

```
                    if abs(val[0]) < 10:
                        f_array.append(val)  # zelena
                    elif abs(val[0]) < 100:
                        f_array.append(val)  # plava
                    elif abs(val[0]) < 1000:
                        f_array.append(val)  # crvena
                    elif abs(val[0]) < 10000:
                        f_array.append(val)  # bijela
                    elif abs(val[0]) < 100000:
                        f_array.append(val)
                    file.seek(1, 1)
            else:
                file.seek(18,1)

        bytes_read = file.read(1)

    file.close()
    return f_array
```

Funkcija extract_floats radi na sličnom principu kao funkcija ind_and_float_finder. Dakle, traži suvisle vrijednosti tipa float. U trenutku kada pročita oznaku za tuple koji ima dvije vrijednosti potrebno je vratiti se natrag za 19 bajtova, ako je na tom mjestu oznaka G pronađena je jedna koordinata (x,y). Funkcija zatim čita 8 bajtova, pretvara ih u float uz pomoć funkcije unpack te provjerava upada li dobivena vrjednost između 0 i 100000, ukoliko upada ubacuje ju u listu f_array. Postupak se ponavlja još jednom.

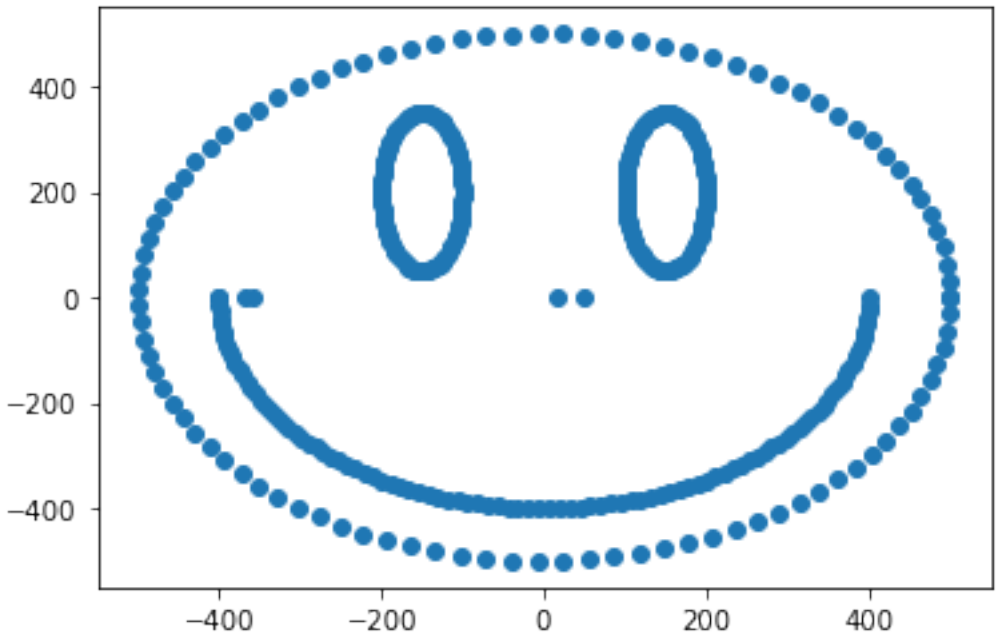
Na kraju dobivamo listu svih vrijednosti koordinata koje je još potrebno prikazati u ravnini.

[21]:
```
floats = extract_floats('task.pickle')
```

[22]:
```
x = []
y = []
i = 1


for el in floats:
    if i % 2 == 0:
        y.append(el)
    else:
        x.append(el)
    i += 1
```

[23]:
```
import matplotlib
plt.scatter(x, y)
plt.show()
```



Za prikaz koordinata u ravnini koristimo biblioteku Matplotlib, tj. njezinu metodu scatter koja se koristi za crtanje raspršenog grafa. Metoda prima dva polja koja predstavljaju x i y vrijednosti koordinate pa tako naše originalno polje dijelimo na dva nova. Na kraju graf prikazujemo uz pomoć metode show.

Možemo vidjeti da se na grafu pojavljuje nekoliko koordinata koje nisu bile predviđene. Razlog tome je što funkcija extract_floats prolazi kroz cijelu datoteku pa je vjerojatno u nekom drugom dijelu datoteke također pronašla neki tuple čije su vrijednosti suvisle i upadaju između 0 i 100000. Pogreška bi vjerojatno bila minimalna kada bi se fokusirali samo na dio datoteke u kojem umamo gusto raspoređene vrijednosti tipa float.

4.2 Ekstrakcija rgb vrijednosti i rekonstrukcije slike

Poznato je da pickle cijele brojeve između 0 i 256 pohranjuje u jedan bajt kao big endian integer ispred kojeg se nalazi slovo K te da tuple koji ima tri vrijednosti označava bajtom x87. Sljedeći je zadatak identificirati takav uzorak te pohraniti pronađene vrijednosti u polje kako bi iz njih mogli rekonstruirati sliku.

[24]:
```
def extract_integers (file_name):
    file = open(file_name, 'rb')
    file_size = len(list(file.read()))
    file.seek(64)
    size = file_size - 64
    pixel_array = []

    while size >= 8:
        val = struct.unpack('BBBBBBBB', file.read(8))

        if val[0] == 75 and val[2] == 75 and val[4] == 75 and val[6] ==␣
↪135 and val[7] == 148:
            for x in range(1,6,2):
                pixel_array.append(val[x])
```

```
        file.seek(-7, 1)
        size = size - 1

    file.close()
    return np.array(pixel_array)
```

Funkcija extract_integers traži suvisle vrijednosti tipa integer. Kreće se kroz datoteku i čita 8 bajtova koje uz pomoć funkcije pack pretvara integere. Zatim provjerava nalazi li se u dobivenom polju traženi uzorak, tj. niz vrijednosti koje definirauju da imamo tuple od tri vrijednosti. Ukoliko imamo željeni uzorak, 3 vrijednosti koje predstavljaju niz vrijednosti (r,g,b) pohranjuju se u polje pixel_array. Vrlo je bitno da uhvatimo vrijednosti u onom redoslijedu u kojem su zapisane kako ne bi poremetili strukturu slike.

[25]:
```
pixels = extract_integers('task.pickle')
```

Pozivamo fukciju extract_integers koja nam vraća jednodimenzionalno polje rgb vrijednosti.

[26]:
```
pixels = np.reshape(pixels, (len(pixels)//3, 3))
```

Uz pomoć funkcije reshape grupiramo vrijednosti tako da dobijemo trojke koje predstavljaju jedan pixel. Kako bi mogli rekonstruirati sliku moramo saznati i njezine dimenzije te ponovno preoblikovati naše polje.

[27]:
```
width = int(np.sqrt([len(pixels)]))
height = int(len(pixels)/width)
```

[28]:
```
pixels = np.reshape(pixels[(len(pixels) - width*height):], (width,␣
↪height, 3))
```

Ako za širinu uzmemo 1079, tj. cijeli dio kvadratnog korijena od ukupnog broja piksela dobivamo da je visina slike 1080. Primjetimo da nam je algorizam za pronalazak integera dohvatio nešto više vrijednosti od ukupnog broja piksela koji mogu činiti sliku ovih dimenzija. Višak piksela pokušat ćemo ukloniti sa početka polja iz razloga što nam je algoritam hvatao integere od početka datoteke te je vrlo vjerojatno na samom početku uhvatio neke koji nisu dio same slike.

[29]:
```
img = Image.fromarray(np.uint8(pixels) , 'RGB')
img.show()
```

Slika 13: Slika rekonstruirana iz niza rgb vrijednosti

[30]:
```
img.save('rekonstruuirana_slika.png')
```

Za rekonstrukciju slike koristimo biblioteku Pillow, tj. njezinu metodu fromarray koja prima jedno polje i kreira objekt slike iz njega. Možemo vidjeti da je slika u potpunosti rekonstruirana pa ju možemo pohraniti u internoj memoriji računala uz pomoć metode save.

Zaključak

Generalni pristup rekonstrukciji podataka nepoznate strukture obuhvaća nekoliko koraka. Prvi korak podrazumijeva analizu načina na koji su organizirani podaci uz pomoć različitih metoda. Drugi korak je kreiranje algoritma koji će podatke uspješno dohvatiti i pročitati i posljednji korak je njihova pohrana u neki razuman oblik.

Različite metode vizualizacije predstavljaju vrlo jednostavan i efikasan izvor informacija o strukturi neke datoteke i samom načinu pohrane podataka. Osim toga, vrlo je važno eksperimentirati sa različitim metodama te bilježiti njihove rezultate bili oni uspješni ili neuspješni. Neuspješne su metode i dalje važne iz razloga što nam npr. mogu smanjiti raspoložive mogućnosti pohrane i usmjerit nas ka drugim pretpostavkama i metodama.

Ovaj proces rekonstrukcije nepoznate strukture složenih podataka može poslužiti kao pojednostavljen primjer za ozbiljno nepoznatije i složenije strukture sa kojima se možemo susresti.

Literatura

[Che16] Angela Chen. *Visualizing Binaries for Low-level File-analysis*. Wolfram. 2016. URL: https://community.wolfram.com/groups/-/m/t/887456.

[Cla15] Alex Clark. *Pillow (PIL Fork) Documentation*. 2015. URL: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf.

[Har+20] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[Hun07] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[Van20] Guido Van Rossum. *The Python Library Reference, release 3.8.12*. Python Software Foundation, 2020.

Popis slika

Popis tablica

Sažetak

Cilj ovog završnog rada bio je kreirati primjer rekonstrukcije nepoznate strukture složenih podataka koji bi poslužio kao primjer za ozbiljno nepoznatije i složenije strukture sa kojima se možemo susresti. Uz pomoć programskog jezika Python napravljena je rekonstrukcija podataka iz jedne Python pickle datoteke koja je neupotrebljiva za klasično otvaranje. Zadatak je bio pronaći niz (x,y) koordinata realnih brojeva te niz (r,g,b) vrijednosti od kojih se sastoji neka slika. Korištene su različite metode za dobivanje rezultata sa naglaskom na metodama vizualizacije datoteke.

Ključne riječi: rekonstrukcija podataka, nepoznata struktura, složeni podaci, pickle datoteka, metode vizualizacije

The aim of this final paper was to create an example of the reconstruction of an unknown structure of complex data that would serve as an example for seriously unknown and more complex structures that we may encounter. With the help of the Python programming language, data was reconstructed from a Python pickle file, which was unusable for classic opening. The task was to find the sequence of (x,y) coordinates of real numbers and the sequence of (r,g,b) values that make up an image. Different methods were used to obtain results with emphasis on file visualization methods.

Keywords: data reconstruction, unknown structure, complex data, pickle file, visualization methods