Rekonstrukcija nepoznate strukture složenih podataka na primjeru SQLite datoteke

Babić, Lucija

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike u Puli

LUCIJA BABIĆ

REKONSTRUKCIJA NEPOZNATE STRUKTURE SLOŽENIH PODATAKA NA PRIMJERU SQLITE DATOTEKE

Završni rad

Pula, 23.rujna, 2021. godine

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike u Puli

LUCIJA BABIĆ

REKONSTRUKCIJA NEPOZNATE STRUKTURE SLOŽENIH PODATAKA NA PRIMJERU SQLITE DATOTEKE

Završni rad

JMBAG: 0054045963, redovita studentica
Studijski smjer: Informatika

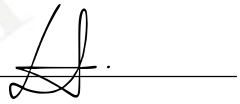
Predmet: Statistika
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc.dr.sc Siniša Miličić

Pula, 23.rujna, 2021. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisana Lucija Babić, 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, Lucija Babić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj Završni rad pod nazivom „Rekonstrukcija nepoznate strukture složenih podataka na primjeru SQLite Datoteke“ 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

1 Uvod

U ovom dokumentu, demonstrirat će se postupak pokušaja iščitavanja podataka iz datoteke koju nije moguće otvoriti za to predviđenim programom. Nažalost, postoji puno uzroka koji mogu proizvesti takve probleme. Recimo da se potkrade neočekivana greška prilikom pohrane podataka. Čini se da su podaci zauvijek izgubljeni, no to ne mora biti slučaj. Oni se vjerojatno i dalje nalaze u moru bajtova.

Iako su jedini tragovi o podacima koje tražimo trivijalni, isti predstavljaju temelj za rastavljanje, detaljno ispitivanje i analizu ključnih koncepata za konačnu rekonstrukciju nepoznate strukture. S obzirom na to da polazimo od stajališta nekoga tko nije upoznat s načinom pohrane podataka u promatranoj datoteci, niti s rukovanjem binarnim datotekama, potraga za podacima predstavlja izniman izazov.

Važno je razmišljati kreativno i metodički. Rješavamo problem stvarajući vlastite metode pomoću postojećih programskih alata. Ako se metode pokažu uspješnim, mogu se primijeniti kao temelj kod rješavanja istovrsnih problema.

No, je li uopće moguće pročitati tzv. nečitljive podatke? Osim što je cilj rekonstruirati podatke iz specifične datoteke, stečeno znanje i iskustvo može se primijeniti kao temelj za razvoj naprednijih algoritama.

2 Alati

Svi alati korišteni za analizu i rekonstrukciju, dio su programskog jezika Python [Van20].

- Numpy - biblioteka za vješto stvaranje i upravljanje višedimenzionalnim poljima. Najpristupačniji način za pohranu velikih nizova bajtova i jednostavno ažuriranje samog niza. [Har+20]
- Matplotlib - biblioteka za grafičko crtanje. Omogućuje objektno orijentirani API za izradu grafičkih prikaza. Koristit ćemo ju kao pomoć pri stvaranju vizualizacija i kao alat za prikaz rezultata. [Hun07]
- Pillow - biblioteka za stvaranje visokorezolucijskih slika iz danih podataka. U kombinaciji s već spomenutim bibliotekama, koristit ćemo Pillow za stvaranje slika vizualizacija i rješenja. [Cla15]
- Struct - modul koji izvodi pretvorbe između Python vrijednosti i C struktura predstavljenih kao objekti bajtova. Spomenuti modul koristit ćemo kao pomoć pri identifikaciji brojeva iz binarnih podataka u datoteci.
- Math - modul koji omogućuje pristup matematičkim funkcijama definiranim standardom C. Koristit ćemo ga kao pomoć prilikom matematičkih izračuna.
- Binascii - modul koji sadrži brojne metode za pretvaranje između binarnih i različitih ASCII kodiranih binarnih prikaza. Koristit ćemo ga za konverziju brojeva
- Functools - modul koji služi za funkcije višeg reda: funkcije koje djeluju na ili vraćaju druge funkcije. Koristit ćemo ih kao pomoć pri izračunu rezultata.

3 Metode

U ovom poglavlju navedene su metode koje će biti korištene prilikom rekonstrukcije nepoznatih struktura.

3.1 Analiza poznate datoteke

Ako nismo sigurni kako SQLite pohranjuje podatke u datoteku, kroz analizu poznate datoteke baze podataka jednostavnije ćemo otkriti kako.

Ovakav način čini se najbrži za utvrđivanje hipoteza o pohrani podataka. Osim navedenog, poznate datoteke možemo koristiti za testiranje valjanosti ostalih metoda navedenih u poglavlju - jer znamo koji rezultat očekivati.

3.2 Analiza hex uređivačem

Hex uređivač, odnosno uređivač bajtova, moćan je alat za pregled i manipulaciju binarnih podataka datoteke. Pruža jednostavno sučelje za pregled čitljivih i odgovarajućih binarnih podataka, odnosno bajtova zapisanih u heksadecimalnom zapisu.

Pomoću hex uređivača dobivamo brzi uvid u promatrane podatke. Koristit ćemo ga kao pomoć u identifikaciji obrazaca, potvrđivanju formata pohrane i mogućih ponavljajućih bajtova koji mogu identificirati nekakav zapis.

3.3 Vizualizacija

Najbitnija promatrana cjelina u datoteci je jedan bajt. Bajt sam po sebi može vraćati nekakvu informaciju. Osim toga, određen skup bajtova može identificirati nešto smisleno u datoteci. Vizualizacijom jednostavnije prikazujemo tražene bajtove, jer sami biramo boje i raspon prikaza. Metodu koristimo kako bi analizirali datoteku, potvrdili nekakvu hipotezu ili uočili smisleni obrazac.

Algoritam za izradu vizualizacije koristi polje bajtova ili nekakvih drugih vrijednosti, i po definiranoj mapi boja izrađuje sliku. Cilj je učiniti prikaz što jednostavnijim, sa što manje elemenata i boja, kako bi lakše naučili iz njega.

```
def image_generator(bytes_stream, file_name, colors, cols = 256):
    cmap, norm = colors
    np_array = np.array(list(bytes_stream))
    rows = int(len(np_array) / cols)

    if(rows):
        data = np.resize(np_array,  rows * cols).reshape(rows, cols)
        im = Image.fromarray(np.uint8(cmap(norm(data))*255))
        im.save(file_name + '.png')
```

3.4 Algoritmi pretrage

Algoritmi pretrage, u suštini, "hodaju" po datoteci tražeći definirane vrijednosti. Možemo ih koristiti za vizualizaciju i za pronalaženje podataka prema definiranim pravilima.

Oni namijenjeni vizualizaciji vraćaju novo polje veličine dane datoteke. To polje sadržava brojeve koji prema definiranoj mapi boja dobivaju svoju svrhu. S druge strane, algoritmi pretrage namijenjeni pronalaženju vrijednosti vraćaju, naravno, tražene podatke. Ovakvi uspješni algoritmi cilj su svakog pokušaja potpune rekonstrukcije nepoznate strukture podataka.

4 Format datoteke SQLite baze podataka

Potpuno stanje SQLite baze podataka obično se nalazi u jednoj datoteci na disku koja se naziva "glavna datoteka baze podataka". Glavna datoteka baze podataka sastoji se od jedne ili više stranica. Sve su stranice dio svojevrsnog stabla zvanog balansirano stablo - b-stablo. Postoje dvije vrste stranica: unutarnje stranice i stranice listovi. Podaci se pohranjuju samo na listovima stranica. Nas zanima najniži level, odnosno, kako su zapisi pohranjeni u datoteci.

Podaci su uvijek u "formatu zapisa". Format zapisa određuje broj stupaca, tip podataka svakog stupca i sadržaj svakog stupca. Format zapisa opsežno koristi cijeli broj promjenjive duljine, skraćeno Varint, 64-bitnih potpisanih cijelih brojeva.

Zapis sadrži zaglavlje i tijelo, tim redoslijedom. Zaglavlje počinje jednim Varint-om koji određuje ukupan broj bajtova u zaglavlju. Nakon veličine, slijedi jedan ili više dodatnih Varint-a, jedan po stupcu. Spomenute dodatne Varint vrijednosti određuju tip podataka svakog stupca, prema sljedećoj tablici:

Serijski tip	Veličina	Značenje
0	0	Vrijednost je NULL
1	1	Vrijednost je 8-bitni dvojni komplement cijelog broja
1	1	Vrijednost je 8-bitni dvojni komplement cijelog broja
2	2	Vrijednost je 16-bitni dvojni komplement cijelog broja
3	3	Vrijednost je 24-bitni dvojni komplement cijelog broja
4	4	Vrijednost je 32-bitni dvojni komplement cijelog broja
5	6	Vrijednost je 48-bitni dvojni komplement cijelog broja
6	8	Vrijednost je 64-bitni dvojni komplement cijelog broja
7	8	Vrijednost je 64-bitni IEEE 754-2008 big endian realni broj
8	0	Vrijednost je 0
9	0	Vrijednost je 1
10, 11	varijabilan	Rezerviran za unutarnju upotrebu
$N \geq 12$, paran	(N-12)/2	Vrijednost je BLOB
$N \geq 13$, neparan	(N-13)/2	Vrijednost je string bez NULL terminator vrijednosti

Tablica 1: Kodovi serijskog tipa formata zapisa [Hip20]

Slijedno tome, tijelo zapisa sadrži tražene podatke, definirane serijskim tipom.

Neposredno prije samog zapisa, postoji jedinstveni identifikator tablice - identifikator retka. Kao i ostale veličine, on je Varint što znači da je varijabilne dužine. Samo u slučaju ako je primarni ključ tablice definiran kao cijeli broj, on postaje ekvivalent identifikatoru retka. Odnosno, primarni ključ se ne nalazi u zapisu nego neposredno prije njega, a u zaglavlju je naznačen serijskim tipom nula. Naravno, u svim ostalim slučajevima primarni ključ iščitava se iz tijela s odgovarajućim serijskim tipom.

Postoji još jedna veličina koja identificira redak u tablici - veličina zapisa. Spomenuti broj nalazi se neposredno prije identifikatora retka i daje dužinu zapisa, naravno kao Varint. Postoje iznimke od ovakvih struktura zapisa, recimo u slučaju kada je zapis reda prevelik i potrebno je "prelit" dio zapisa na sljedeću stranicu, no za nas to sigurno neće biti slučaj.

4.1 Cijeli broj promjenjive duljine - Varint

Kompresija Baze-128 poznata je pod mnogim imenima - VB (promjenjivi bajt), VByte, Varint, VInt, EncInt. Cijeli broj promjenjive duljine, odnosno Varint, kodiranje je 64-bitnih cijelih brojeva bez znakova (unsigned) u zapis veličine između 1 i 9 bajtova. Koristi se zbog optimizacije memorije, jer manje (i češće) vrijednosti koriste manje bajtova i zauzimaju manje prostora od većih (i rjeđih) vrijednosti. Primjer kodiranja cijelog broja u cijeli broj promjenjive duljine prikazan je tablicom.

Broj	1000
Binarni zapis	00000011 11101000
Zapis rascijepljen na 7 bita	0000111 1101000
Zapis s nadodanim vodećim bitovima	10000111 01101000
Heksadecimalni zapis	8768

Tablica 2: Primjer pretvorbe cijelog broja varijabilne duljine

Kodiranje podrazumijeva oktet (osmobitni bajt) gdje je najznačajniji bit, također općenito poznat kao znakovni bit, rezerviran za označavanje slijedi li drugi oktet. Postupak stvaranja Varint zapisa započinje osnovnim, 256 baznim, binarnim zapisom broja. Zatim je potrebno konvertirati zapis u onaj na bazi 128, odnosno ''rascijepiti" bajtove na 7-bitne cjeline. Ono što preostaje je nadodati im najznačajniji bit. Zadnja cjelina, ili jedina cjelina ako je broj manji od 128, ima nulu kao vodeći bit.

Sve ostale cjeline imaju jedinicu kao vodeći bit, što znači da nakon njih slijedi još cjelina, odnosno, još barem jedan bajt. [Wan+17]

Sljedeći algoritam pretvara niz bajtova u ispravan zapis u dekadskom obliku. Algoritam podrazumijeva ulaz niza bajtova koji su ispravan Varint, neispravan unos neće dati ispravan izlaz.

```
def get_varint(byte_stream):
    bytes_string = ''.join(list(map(lambda x: f'{x:08b}'[1:], byte_stream)))
    return int(bytes_string, base=2)
```

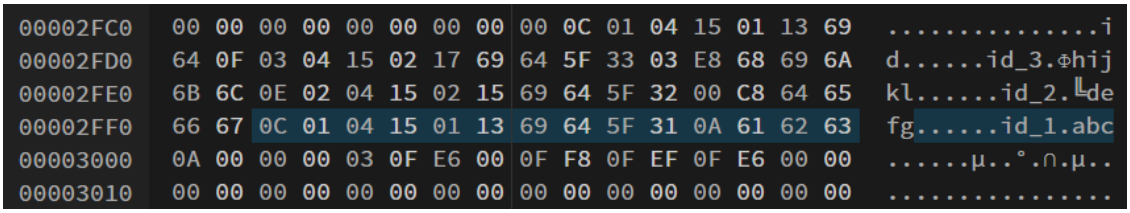
4.2 Primjer zapisa

Recimo da imamo definiranu sljedeću tablicu u SQLite bazi podataka:

	id	num	str
1	id_1	10	abc
2	id_2	200	defg
3	id_3	1000	hijkl

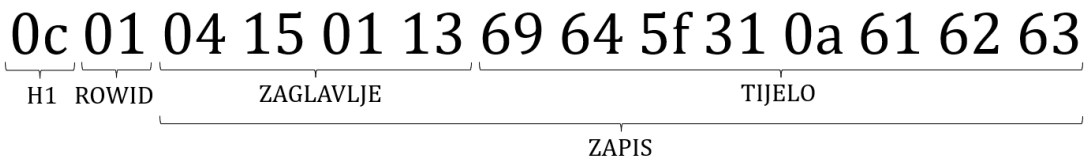
Tablica 3: Primjer zapisa u bazi podataka

Tablica ima primarni ključ "id" tipa TEXT, te stupce "num" i "str" redom tipa INTEGER i TEXT. Kako bi lakše razumjeli zapis, sadržaj datoteke u kojoj je spremljena navedena tablica prikazan je u hex uređivaču na slici 1.



Slika 1: Prikaz primjera zapisa podataka u hex uređivaču

Na slici, naglašen dio je onaj koji identificira prvi redak u bazi podataka. Slijedno tome, vidljivo je kako sekvenca redova nije sadržana u glavnoj datoteci baze podataka, ali sekvenca stupaca jest. Analizirajmo naglašeni dio prema informacijama o pohrani podataka.



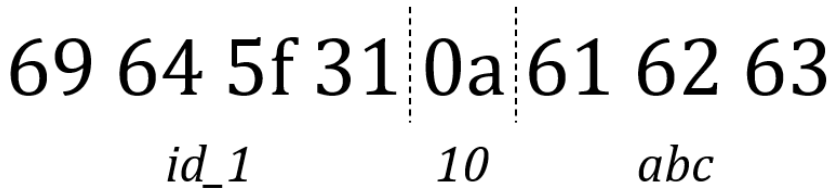
Slika 2: Opis zapisa podataka u SQLite datoteci

Prva vrijednost, H1, označava dužinu zapisa. Iz navedenog vidimo da je dužina zapisa dvanaest, što iz slike 2 možemo potvrditi. Nakon H1, slijedi identifikator retka. S obzirom na to da nemamo cjelobrojni primarni ključ, SQLite ga automatski dodjeljuje. Zatim imamo sam zapis. U zaglavlju zapisa, prva vrijednost označava samu dužinu zapisa, i vidljivo je da dužina stvarno jest četiri. Zatim slijede tri broja serijskog tipa, jedan kao identifikator svakog stupca. Nakon usporedbe tablice serijskih tipova s danim brojevima, zaključujemo sljedeće:

Redni broj zapisa	Serijski tip	Značenje
1. zapis	15	Vrijednost je tekst duljine 4 bajta
2. zapis	01	Vrijednost je 8-bitni dvojni komplement cijelog broja
3. zapis	13	Vrijednost je tekst duljine 3 bajta

Tablica 4: Analiza zaglavlja primjera zapisa

Navedeno odgovara očekivanom. Bitno je napomenuti da su tekstovi pohranjeni bez nul terminator vrijednosti, a kodiranje teksta zabilježeno je u zaglavlju stranice. Pretpostavimo da je navedeni tekst čitljiv pomoću ASCII tablice, s obzirom da ne tražimo nikakve posebne znakove. Sada možemo promotriti tijelo zapisa. U tijelu se nalaze podaci koje tražimo. Razdvojimo li bajtove na spomenute cjeline i dekodiramo svaku cjelinu, dobivamo sljedeće:



Slika 3: Analiza tijela primjera zapisa

5 Uvod u problem

U ovom poglavlju, predstavit će se datoteka iz koje je potrebno iščitati podatke, odnosno rekonstruirati nepoznate strukture. Također, prikazat će se i prva analiza datoteke metodom vizualizacije.

5.1 Zadatak

Imamo datoteku koju nije moguće otvoriti klasičnim otvaranjem. Znamo da datoteka sadrži sljedeće:

- niz (x,y) koordinata, realnih brojeva
- niz (r,g,b) integer podataka i primarnog ključa koji osigurava redoslijed vrijednosti piksela neke RGB slike
- nekakav informacijski škart

Zadatak je uspješno pročitati tražene vrijednosti. Odnosno, koordinate prikazati kao točke u scatterplot-u, a sliku rekonstruirati iz dobivenih podataka.

5.2 Analiza cijele datoteke

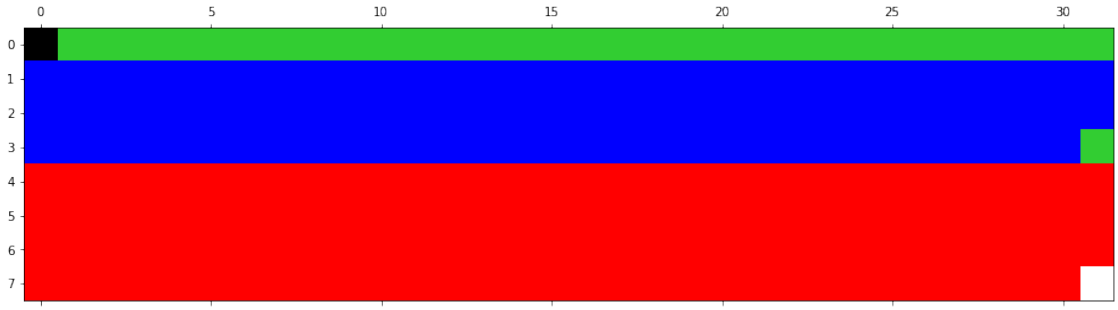
Za početak, korisno je vizualizirati datoteku po bajtovima, odnosno, grupacijama smislenih ASCII vrijednosti. Iako znamo da podaci nisi kodirani po ASCII, takav prikaz nam može biti od pomoći, ako se vodimo pretpostavkom da informacijski škart vjerojatno čine riječi. Ukratko, tražimo obrazac, segregaciju i/ili bilo kakav koristan indikator sadržaja datoteke. Grupacija po bojama:

Kategorija	Vrijednost/i	Boja
Najmanja moguća vrijednost	0	crna
Najveća moguća vrijednost	255	bijela
Ispisivi znakovi	[32-126]	plava
Kontrolni znakovi	[1-31] i 127	zelena
Prošireni kodovi	[128-254]	crvena

Tablica 5: Grupacije ASCII vrijednosti [Che16]

Prema tablici, napravit ćemo odgovarajuću mapu boja.

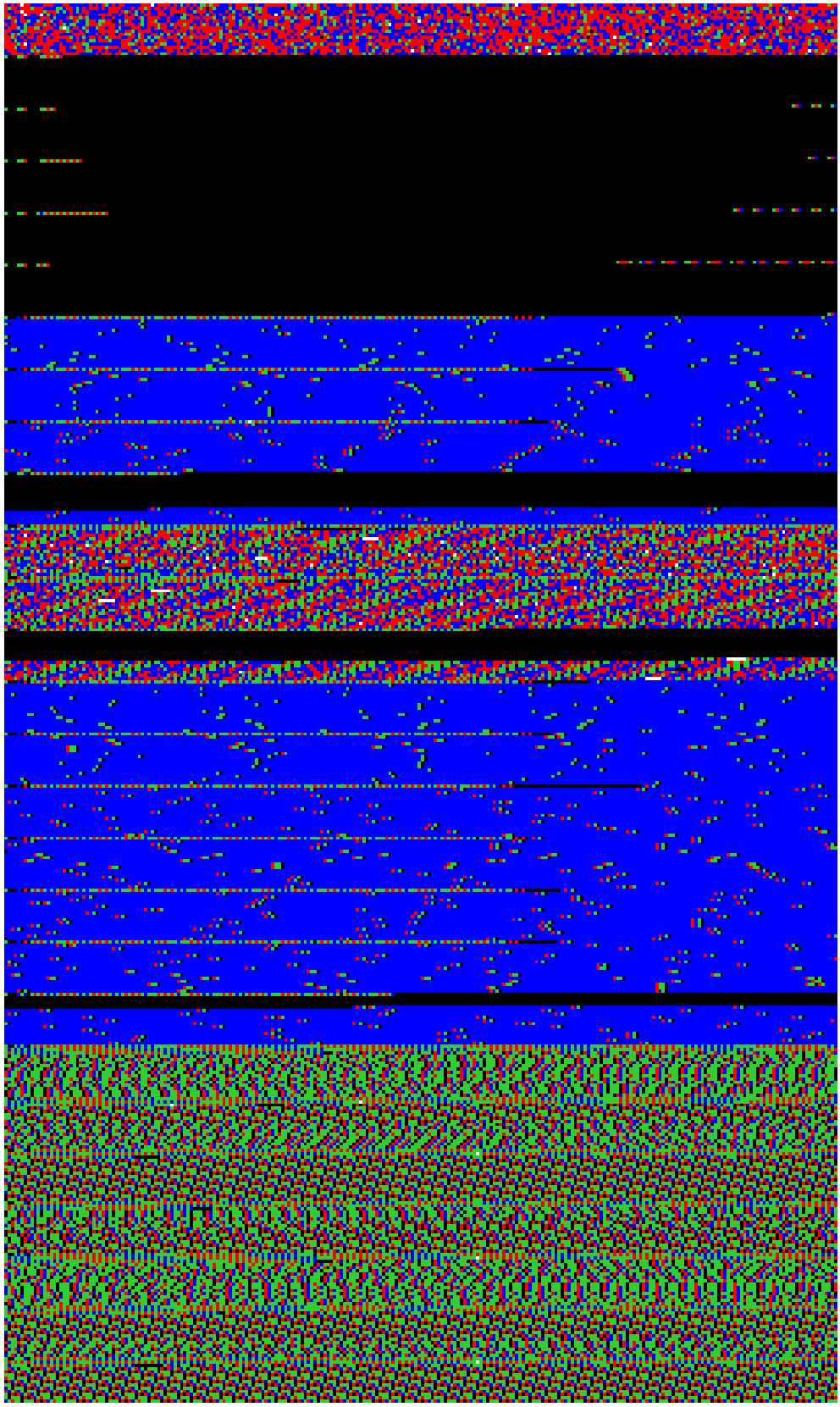
```
def ascii_colors():
    bounds = [0, 0.99, 31, 127, 128, 255, 256]
    colors = ["black", "limegreen", "blue", "limegreen", "red", "white"]
    cmap = matplotlib.colors.ListedColormap(colors)
    norm = matplotlib.colors.BoundaryNorm(bounds, len(colors))
    return (cmap, norm)

#test
arr = np.arange(256)
cmap, norm = ascii_colors()
fig = plt.figure()
plt.matshow(arr.reshape(8,32), cmap=cmap, norm=norm)
plt.show()
```



Korištenjem algoritma za izradu slike i izrađene mape boja, napravit ćemo vizualizaciju cijele datoteke.

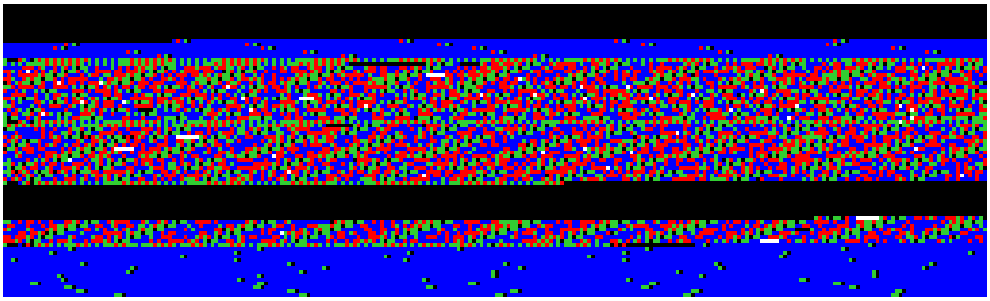
```
with open('zadatak.db', 'rb') as f:
    file = f.read()
    image_generator(file, 'fullImage', ascii_colors())
```

Na slici 4 vidljiv je rezultat vizualizacije. Dobivena slika prevelika je za prikaz, stoga je prikazana skraćene visine. Tražimo skupove realnih brojeva i RGB vrijednosti. Iz dobivene slike jednostavno je identificirati informacijski škart, s obzirom na to da gotovo svako kodiranje zadržava vrijednosti od 32 do 126 kao ASCII ispisive znakove. Vidljivo je da informacijski škart, odnosno pretežno plavi blokovi, sigurno dijele nekakve smislene strukture. Smislene strukture su one koje nisu čisti šum, kao na primjer šareni blok na početku slike - takvi blokovi su moguće kriptirani ili jednostavno besmisleni. Besmisleni su naravno i blokovi samih nula, odnosno crni blokovi na slici. Pretpostavka je da su to strukture rezervirane za buduće moguće ažuriranje tablice. Uglavnom, smisleni blokovi moraju imati nekakav uzorak koji odgovara traženim vrijednostima.



Slika 4: Vizualizacija po ASCII grupama vrijednosti

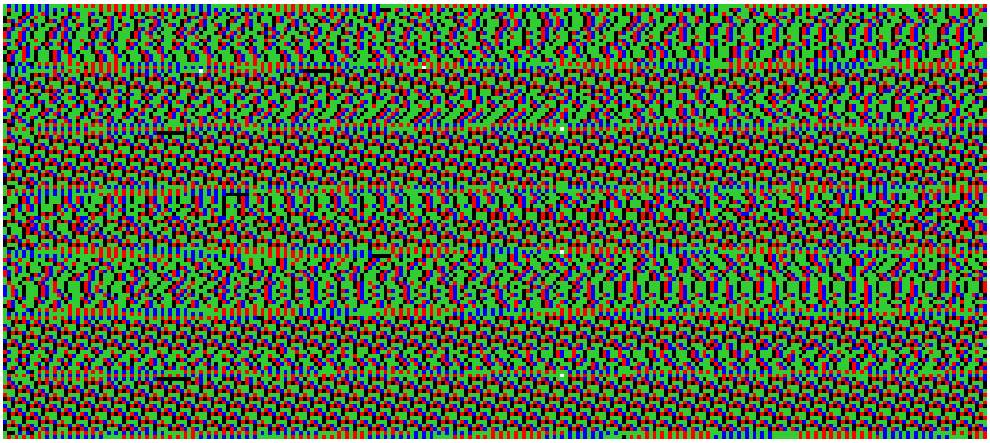
Pretpostavimo da sljedeći blok sadržava koordinate:



Slika 5: Vizualizacija po ASCII grupama vrijednosti - koordinate

Iako na prvu djeluje kao šum, ovaj blok itekako ima strukturu. S obzirom da koordinate trebaju činiti prikaz nečega smislenog u koordinatnom sustavu, vrijednosti bi trebale biti dosta slične. To je moguće samo pretpostaviti iz slike prema grupaciji ASCII vrijednosti, s obzirom na to da je obrazac gotovo jasno vidljiv.

Ispod drugog plavog bloka sve do kraja slike vidljiva je struktura. Pretpostavimo da spomenuti blok sve do kraja sadržava RGB vrijednosti.



Slika 6: Vizualizacija po ASCII grupama vrijednosti - RGB vrijednosti

Gotovo sigurno je da se ovdje nalaze RGB vrijednosti slike. S obzirom na to da govorimo o pikselima, takve vrijednosti ne samo da su slične, nego se u velikim količinama i iste.

Ako su gornje tvrdnje ispravne, uspješno smo identificirali mjesto čitljivih podataka. Sljedeći korak je pokušaj iščitavanja i sama rekonstrukcija.

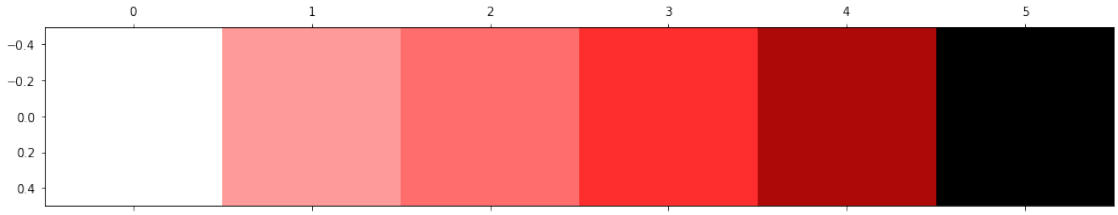
6 Identifikacija i rekonstrukcija

U ovom poglavlju, koristit ćemo spomenute alate i metode s ciljem ispravne rekonstrukcije nepoznatih podataka.

6.1 Identifikacija koordinata

Potrebno je identificirati realne brojeve. Znamo da SQLite pohranjuje realne brojeve kao 64-bitni big endian realan broj. Koristeći Pythonov paket struct, identifikacija će biti jednostavna. Prije sakupljanja samih podataka, koristit ćemo metodu vizualizacije kako bi potvrdili navedenu tvrdnju. Pretpostavimo da, hodajući kroz datoteku, postoje preklapajuće vrijednosti - s obzirom na to da je zapis realnog broja dug čak 8 bajta. Navedeno moramo uzeti u obzir. Definirajmo niz boja kojima ćemo prikazati stupanj preklapanja.

```
def floating_points_colors():
    bounds = [0, 0.99, 2, 3, 4, 5, 6]
    colors = ["#ffffff", "#ff9a9a", "#ff6d6d", "#ff2e2e", "#ae0909", "#000000"]
    cmap = matplotlib.colors.ListedColormap(colors)
    norm = matplotlib.colors.BoundaryNorm(bounds, len(colors))
    return (cmap, norm)

#test
arr = np.arange(6)
cmap, norm = floating_points_colors()
fig = plt.figure()
plt.matshow(np.expand_dims(arr, axis=0), cmap=cmap, norm=norm)
plt.show()
```



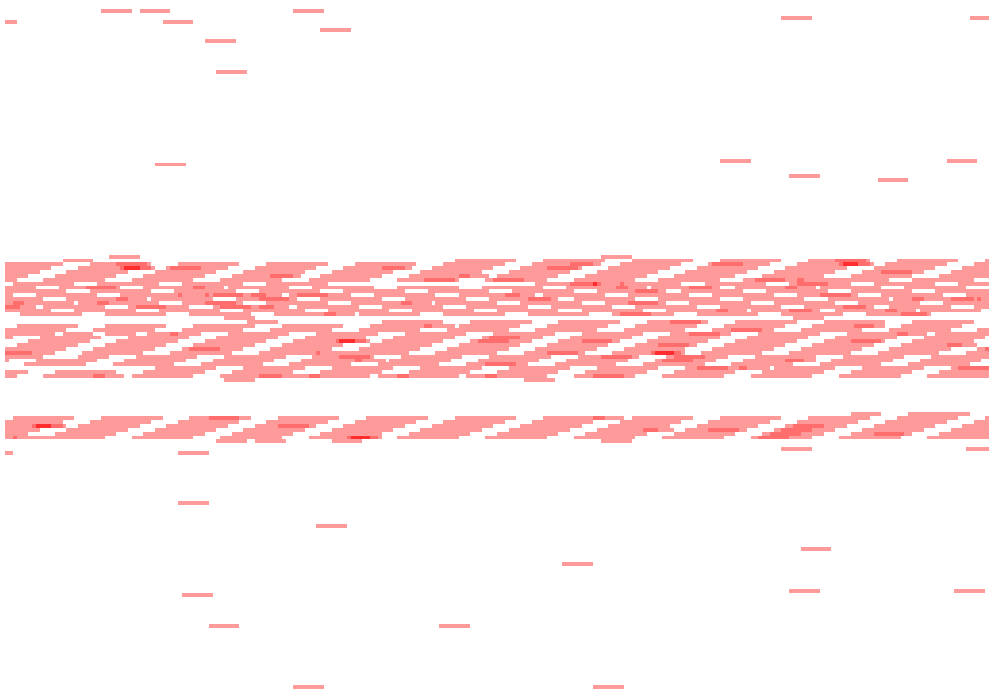
Pretpostavimo da je smisleni realan broj onaj apsolutne vrijednosti između 0.000001 i milijun. Slijedno tome, koordinate su dva smislena realna broja.

```
def is_valid_float(num):
    return 1e-6 <= abs(num) <= 1e6

def is_valid_pair(coordinates):
    return is_valid_float(coordinates[0]) and is_valid_float(coordinates[1])
```

Sljedeći algoritam pretrage pronalazi smislene realne brojeve. Prije pretrage definirano je polje

nula, jednake veličine kao sama datoteka. Kada algoritam pronađe smisleni broj, na njegovom mjestu u novom polju poveća zapise za jedan. Ako je neposredno nakon njega još jedan smisleni cijeli broj, na njegovo mjestu opet povećava zapise za jedan, čime postaje vidljivo spomenuto preklapanje.

```
def floating_points_array(file_name):
    file = open(file_name, 'rb')

    float_length = 8
    np_final_array = [0] * len(file.read())
    file.seek(0)

    bytes_read = file.read(float_length)


    while len(bytes_read) == float_length:
        var = struct.unpack('>d', bytes_read)
        if is_valid_float(var[0]):
            index = file.tell() - float_length
            for i in range(float_length):
                np_final_array[index + i] += 1

        file.seek(- (float_length - 1), 1)
        bytes_read = file.read(float_length)

    return np_final_array

floating_points = floating_points_array('zadatak.db')
image_generator(floating_points, 'floatingPoints', floating_points_colors())
```



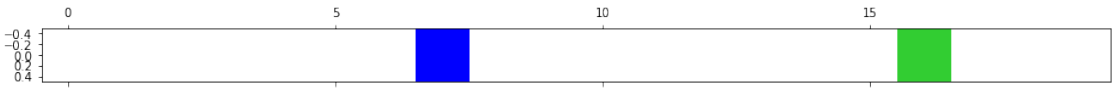
Slika 7: Vizualizacija realnih brojeva

Na slici 7 prikazan je isječak za koji pretpostavljamo da sadrži koordinate. Iako vidimo puno preklapanja, postoji puno nizova od točno šesnaest bajtova. Ti nizovi su u neposrednoj blizini, stoga je gotovo sigurno da se ovdje nalaze tražene koordinate.

Sljedeće možemo vizualizirati smisleni niz od dva realna broja, koji se nalaze neposredno jedan pokraj drugog. Osim toga, prema tablici serijskog tipa, dvije sedmice identificiraju redak od dva realna broja neposredno prije njihovog zapisa. Tražimo takvu strukturu - dvije sedmice i dva smislena realna broja. Kako bi potvrdili navedenu tvrdnju, ponovno ćemo koristiti metodu vizualizacije.

Recimo da su sedmice u polju koje vraća algoritam pretrage prikazane kao sam broj sedam, a nizovi od dva realna broja prikazani nizovima šesnaestica. Definiramo boje za željeni prikaz:

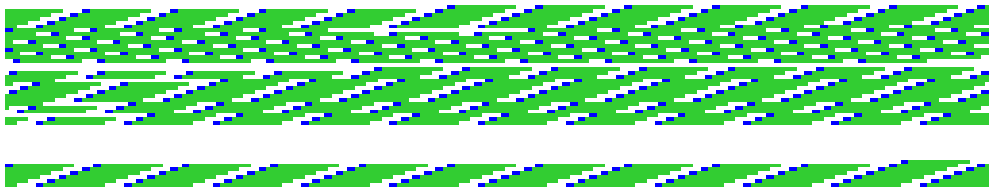
```
def identify_coordinates_colors():
    bounds = [0, 6.99, 8, 16, 17, 255]
    colors = ["white", "blue", "white", "limegreen", "white"]
    cmap = matplotlib.colors.ListedColormap(colors)
    norm = matplotlib.colors.BoundaryNorm(bounds, len(colors))
    return (cmap, norm)

#test
arr = np.arange(20)
cmap, norm = identify_coordinates_colors()
fig = plt.figure()
plt.matshow(np.expand_dims(arr, axis=0), cmap=cmap, norm=norm)
plt.show()
```



Algoritam pretrage traži niz od dvije sedmice. Ako nakon dvije sedmice dolaze dvije smislene koordinate, struktura je pronađena i potrebno ju je pohraniti u polje točno na istom mjestu na kojem se nalazi u datoteci.

```
def identify_coordinates_array(file_name):
    file = open(file_name, 'rb')

    #init array
    np_final_array = [0] * len(file.read())

    file.seek(0)

    two_floats_length = 8 * 2
    serial_nums_length = 2
    required_chunk_length = two_floats_length + serial_nums_length
```

```
    bytes_read = file.read(required_chunk_length)

    while len(bytes_read) == required_chunk_length:
        if list(bytes_read[0:serial_nums_length]) == [7, 7]:
            var = struct.unpack('>dd', bytes_read[serial_nums_length:
 ↪required_chunk_length])
            if is_valid_pair(var):
                index = file.tell() - required_chunk_length
                np_final_array[index:index + required_chunk_length] = [7] *␣
 ↪serial_nums_length + [16] * two_floats_length

        file.seek(- (required_chunk_length - 1), 1)
        bytes_read = file.read(required_chunk_length)

    file.close()

    return np_final_array

final_array = identify_coordinates_array('zadatak.db')
image_generator(final_array, 'coordinatesIdentify',␣
 ↪identify_coordinates_colors())
```

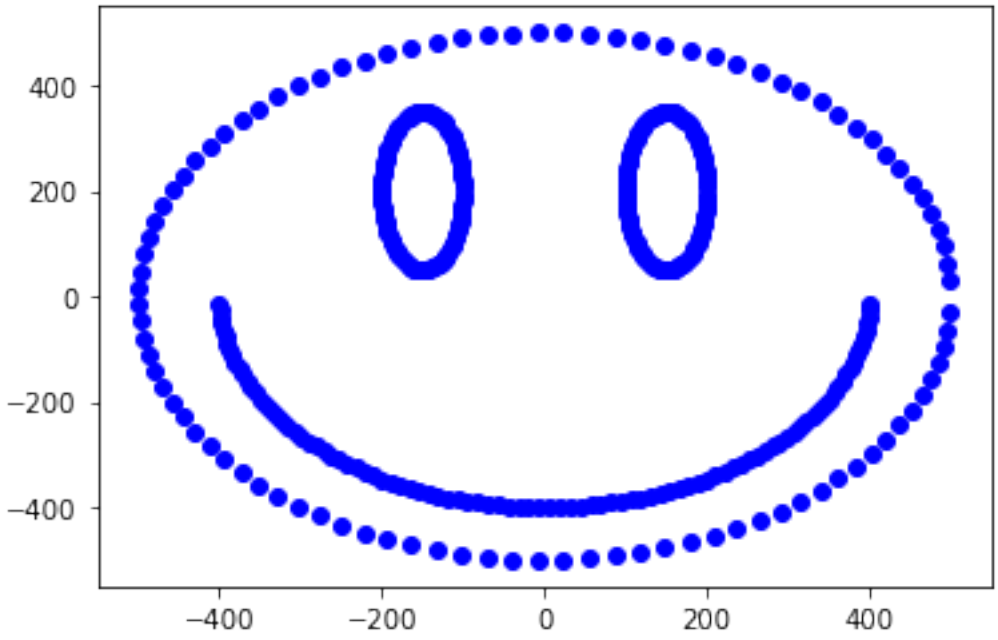


Slika 8: Vizualizacija smislenog para koordinatnih točaka

Rezultat je obećavajuć. Očekivano je da su sve vrijednosti koordinata barem donekle prostorno blizu. To potvrđujemo činjenicom da nigdje drugdje u datoteci ne postoji ovakva struktura. Zaključak je da se možemo voditi algoritmom, koji će sada identificirati i vratiti koordinate.

```
def identify_coordinates(file_name):
    file = open(file_name, 'rb')

    two_floats_length = 8 * 2
    serial_nums_length = 2
    required_chunk_length = two_floats_length + serial_nums_length

    coordinates = []
    bytes_read = file.read(required_chunk_length)

    while len(bytes_read) == required_chunk_length:
        if list(bytes_read[0:serial_nums_length]) == [7, 7]:
            var = struct.unpack('>dd', bytes_read[serial_nums_length:
 ↪required_chunk_length])
            if is_valid_pair(var):
                coordinates.append(var)

        file.seek(- (required_chunk_length - 1), 1)
```

```
        bytes_read = file.read(required_chunk_length)

    file.close()
    return coordinates

coordinates = identify_coordinates('zadatak.db')
plt.scatter(*zip(*coordinates), color='blue')
```



Slika 9: Gotovo potpuna rekonstrukcija koordinatnog sustava

S obzirom na to da rezultat daje smisleni prikaz, rekonstrukciju možemo smatrati uspješnom.

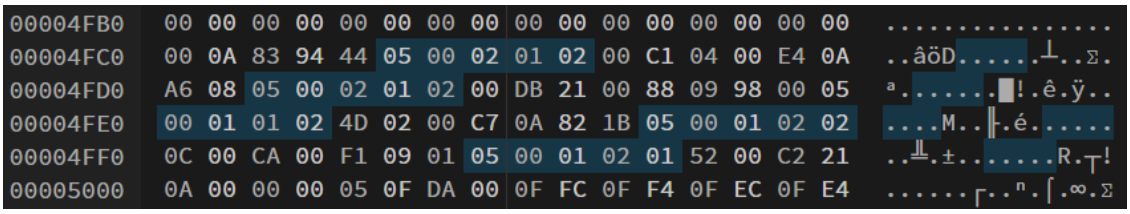
6.2 Identifikacija RGB vrijednosti

Ovaj zadatak djeluje složenije nego prijašnji. Potrebno je identificirati redove i njihov primarni ključ. Možemo koristiti metodu analize poznate datoteke kako bi pretpostavili način pohrane takve strukture. Pohranimo nasumično odabrane cijele brojeve koji mogu biti valjana RGB vrijednost.

	id	r	g	b
1	1	82	194	33
2	283	12	202	241
3	3072	77	2	199
4	4872	219	33	136
5	51780	193	4	228

Tablica 6: Primjer zapisa RGB vrijednosti u bazi podataka

Tablica ima primarni ključ "id" tipa INTEGER, te stupce r, g, i b isto tipa INTEGER. S obzirom na uspješnost koja je slijedila nakon identifikacije zaglavlja zapisa u prijašnjem poglavlju, rješavanje ovog problema započeti ćemo tako. Pomoću hex uređivača tražimo zaglavlja retka RGB vrijednosti.



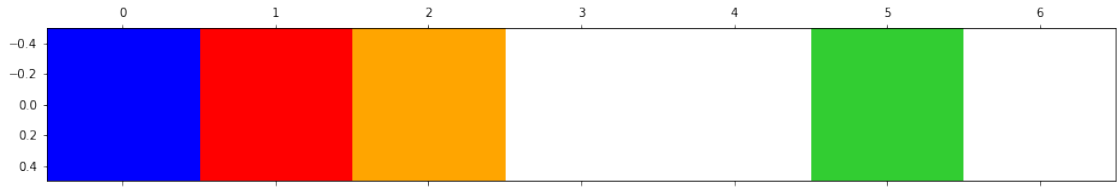
Slika 10: Prikaz obrasca zaglavlja RGB vrijednosti hex uređivaču

Sa slike 10 vidljivo je da je zaglavlje uvijek veličine pet. Nakon toga, vidljivo je da je primarni ključ identificiran kao INTEGER, jer je naznačen serijskim brojem nula. Time znamo da je primarni ključ kodiran kao Varint pohranjen ispred samog zapisa. Osim toga, vidimo ponavljajući niz dužine tri koji se sastoji od jedinica i dvojki. Spomenuto ima smisla jer su to serijski tipovi 8-bitnih i 16-bitnih cijelih brojeva, a tražimo RGB vrijednosti - brojeve između 0 i 255.

Ako je pohrana u datoteci ista, očekujemo takve nizove. Prije pretrage možemo koristiti metodu

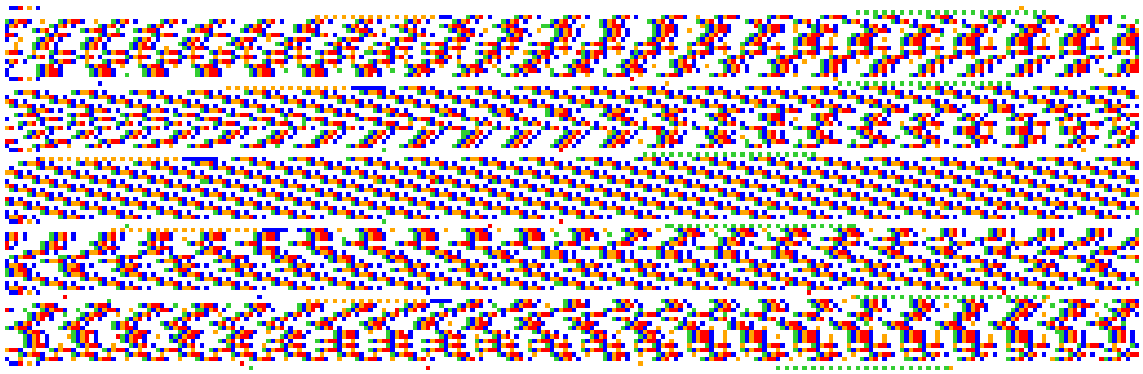
vizualizacije kako bi se uvjerili u gornju tvrdnju. Prikažimo broj pet zelenom, nulu plavom a jedinice i dvojke toplim bojama - crvenom i narančastom.

[14]:
```
def rgb_header_colors():
    bounds = [0, 0.99, 2, 3, 5, 6, 255]
    colors = ["blue", "red", "orange", "white", "limegreen", "white"]
    cmap = matplotlib.colors.ListedColormap(colors)
    norm = matplotlib.colors.BoundaryNorm(bounds, len(colors))
    return (cmap, norm)

#test
arr = np.arange(7)
cmap, norm = rgb_header_colors()
fig = plt.figure()
plt.matshow(np.expand_dims(arr, axis=0), cmap=cmap, norm=norm)
plt.show()
```



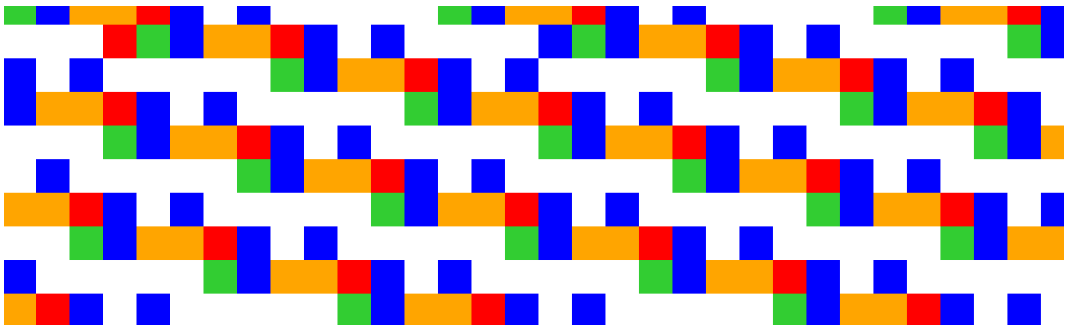
S obzirom na to da želimo istaknuti bajtove određenih vrijednosti, dovoljno je samo napraviti sliku pomoću kreirane mape boja.

[16]:
```
with open('zadatak.db', 'rb') as f:
    file = f.read()
    image_generator(file, 'rgbHeaderImage', rgb_header_colors())
```



Slika 11: Vizualizacija zaglavlja RGB vrijednosti

Slika 11 potvrđuje tvrdnju o pohrani RGB vrijednosti. Dobiven je smisleni obrazac, a poredak bajtova odgovara očekivanom. Promotrimo sljedeći isječak:



Slika 12: Detalji vizualizacije zaglavlja RGB vrijednosti

Zaglavlja periodično započinju zelenom bojom, koju slijedi plava boja - to je očekivani niz petice i nule. Slijede tri zapisa cijelog broja. Ako promatramo tijelo zapisa, ponavljajuća pozicija nula ima smisla - jer pohranjujemo brojeve do 255, stoga u 16-bitom zapisu cijelog broja prvi bajt mora biti jednak nuli.

Sada kada smo sigurni o načinu pohrane traženih vrijednosti, trebamo pronaći cijelu strukturu, odnosno primarni ključ i tijelo zapisa. I primarni ključ i tijelo zapisa su varijabilne veličine, stoga identifikacija mora ići postepeno - počevši od niza redom brojeva 5 i 0, koji identificiraju vjerojatan početak zaglavlja. Algoritam pretrage funkcionira na sljedeći način:

1. Pronađi niz (5, 0)
2. Ako su sljedeće tri vrijednosti jedinice ili dvojke, pronađeno je zaglavlje
3. Definiraj formate zapisa cijelih brojeva
4. Definiraj dužinu tijela
5. Pročitaj RGB vrijednosti
6. Definiraj veličinu h1, odnosno veličinu zapisa
7. Idi u nazad dok ne pronađeš h1
8. Pročitaj primarni ključ nakon vrijednosti h1 i pohrani ga kao ključ u rječniku koji referencira pronađenu RGB vrijednost

```
def rgb_finder(file_name):
    file = open(file_name, 'rb')
    file_length = len(file.read())
    file.seek(0)

    final_dict = {}
    arr = []

    for index in range(file_length):
        file.seek(index)
        bytes_read = file.read(2)
        if list(bytes_read) == [5, 0]:
            bytes_read = file.read(3)
            if all(x == 1 or x == 2 for x in bytes_read):
                struct_string = functools.reduce(lambda a, b: a + b,␣
 ↪list(map(lambda x: 'B' if x == 1 else 'H', bytes_read)))
                bytes_length = functools.reduce(lambda a, b: a + 1 if b == 'B'␣
 ↪else a + 2, struct_string, 0)
                bytes_read = file.read(bytes_length)
                rgb = struct.unpack('>' + struct_string, bytes_read)

                value_of_h1 = bytes_length + 5
                byte_to_find = struct.pack('>B', value_of_h1)

                file.seek(- (value_of_h1 + 1), 1)
                for i in range(11):
                    var = file.read(1 + i)
                    if struct.pack('>B', var[0]) == byte_to_find and i > 0:
                        final_dict[get_varint(var[1:])] = list(rgb)
                        break
                    file.seek(- (2 + i), 1)
    return final_dict
```

Jedino što preostaje jest pohraniti vrijednosti u novo polje, pravilnim redoslijedom primarnih ključeva. Pretpostavimo da ćemo imati nedostajuće vrijednosti, odnosno ne identificirane piksele. Ti pikseli bit će obojani crnom bojom, odnosno RGB(0, 0, 0). Osim toga, pretpostavimo da redni brojevi, odnosno redoslijed piksela, vjerojatno započinje nulom.

```
def get_rgb_array(rgb_dict):
    arr = []
    for key in range(0, max(rgb_dict.keys()) + 1):
        arr.append(rgb_dict.get(key,[0, 0, 0]))

    cols = int(math.sqrt(len(arr)))
    rows = int(len(arr) / cols)

    return np.array(arr, dtype=np.uint8)[:- (len(arr) - rows * cols)].
 ↪reshape(rows, cols, 3)

im = Image.fromarray(get_rgb_array(rgb_finder('zadatak.db')))
im.save('rgbImage' + '.png')
```

Slika 13: Prvi rezultat identifikacije RGB vrijednosti - nedostajući pikseli, neispravan pomak

Vidimo da slika nije potpuna, sigurno smo nešto previdjeli. Osim nedostajućih piksela, slika ima neobičan pomak. Moguće je da postoji fazni pomak, ili je krivo definiran početak slike.

Ne nedostaje mnogo piksela, stoga algoritam uglavnom funkcionira. Moguće da postoji drugačiji način pohrane cijelih brojeva. Prema tablici serijskog tipa, vidljivo je kako postoji način pohrane jedinice i nule koji optimizira memoriju.

U zaglavlju zapisa, nula može biti identificirana brojem osam, a jedinica brojem devet. Time ti zapisi u tijelu ne postoje i ne zauzimaju memoriju. Previdjeli smo da se takav način optimizacije

često koristi, ali ima smisla da su spomenuti brojevi česti u bazama podataka i ovakva naizgled neobična optimizacija može uvelike biti od koristi.

Ažurirajmo algoritam pretrage, koji sada uključuje osmice i devetke kao serijske tipove koji mogu biti definirani u zaglavlju.

```
def rgb_finder2(file_name):
    file = open(file_name, 'rb')
    file_length = len(file.read())
    file.seek(0)

    final_dict = {}
    arr = []

    for index in range(file_length):
        file.seek(index)
        bytes_read = file.read(2)
        if list(bytes_read) == [5, 0]:
            bytes_read = file.read(3)
            if all(x in (1, 2, 8, 9) for x in bytes_read):
                data_types = list(bytes_read)
                filtered_list = list(filter(lambda y: y in [1, 2], data_types))
                length_of_h1 = 5
                if len(filtered_list):
                    struct_string = "".join(list(map(lambda x: 'B' if x == 1␣
 ↪else 'H', filtered_list)))
                    bytes_length = functools.reduce(lambda a, b: a + 1 if b ==␣
 ↪'B' else a + 2 if b == 'H' else a + 0, struct_string, 0)
                    bytes_read = file.read(bytes_length)
                    unpacked_values = list(struct.unpack('>' + struct_string,␣
 ↪bytes_read))
                    length_of_h1 += bytes_length

                rgb = list(map(lambda x: 0 if x == 8 else 1 if x == 9 else␣
 ↪unpacked_values.pop(0), data_types))

                byte_to_find = struct.pack('>B', length_of_h1)

                file.seek(- (length_of_h1 + 1), 1)
                for i in range(11):
                    var = file.read(1 + i)
                    if struct.pack('>B', var[0]) == byte_to_find and i > 0:
                        final_dict[get_varint(var[1:])] = list(rgb)
                        break
                    file.seek(- (2 + i), 1)

    return final_dict
```

```
im = Image.fromarray(get_rgb_array(rgb_finder2('zadatak.db')))
im.save('rgbImage2' + '.png')
```

Slika 14: Drugi rezultat identifikacije RGB vrijednosti - potpuni pikseli, neispravan pomak

Gornja tvrdnja je bila točna i očito se memorijska optimizacija uvelike koristi u SQLite datotekama. Gotovo je sigurno da imamo sve piksele, samo je pomak još neispravan.

S obzirom na to da imamo definirane sve piksele, sigurno je i onaj s prvim rednim brojem pohranjen u rječniku. Potrebna je samo izmjena algoritma za stvaranje konačnog polja RGB vrijednosti, tako da popunjavanje kreće on najmanjeg rednog broja, a ne od nule.

```
def get_rgb_array(rgb_dict):
    arr = []
    for key in range(min(rgb_dict.keys()), max(rgb_dict.keys()) + 1):
        arr.append(rgb_dict.get(key,[0, 0, 0]))
```

```
    cols = int(math.sqrt(len(arr)))
    rows = int(len(arr) / cols)

    return np.array(arr, dtype=np.uint8).reshape(rows, cols, 3)

im = Image.fromarray(get_rgb_array(rgb_finder2('zadatak.db')))
im.save('rgbImage3' + '.png')
```

Slika 15: Treći rezultat identifikacije RGB vrijednosti - potpuna rekonstrukcija

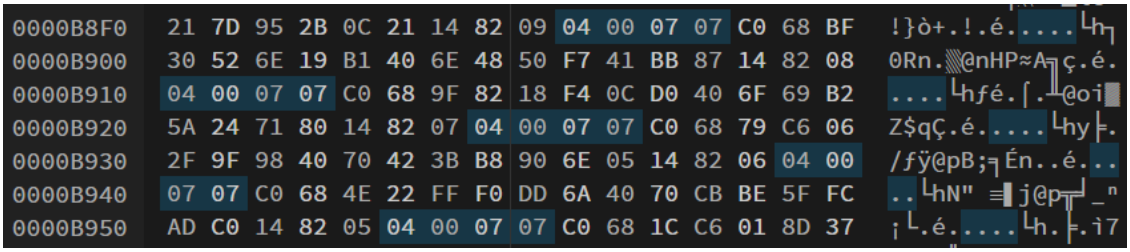
Slika je potpuna, smislene širine i početka. Rekonstrukciju možemo smatrati u potpunosti uspješnom.

6.3 Identifikacija koordinata - nadopuna

Nakon uspješne identifikacije slike, vidljivo je koliko zaglavlje igra veliku ulogu u uspješnoj rekonstrukciji složenih struktura. Ako dovoljno dobro pretpostavimo sadržaj zaglavlja, vjerojatno imamo ispravno rješenje.

Postoji mogućnost da neke brojeve nismo pronašli jer nisu pohranjeni kao realni broj. Prema SQLite dokumentaciji: "Ako je tip stupca REAL i taj stupac sadrži vrijednost koja se može pretvoriti u cijeli broj bez gubitka informacija (ako vrijednost ne sadrži razlomački dio i nije prevelika da bi se mogla prikazati kao cijeli broj), tada stupac može biti pohranjen u zapisu kao cijeli broj. SQLite će pretvoriti vrijednost natrag u realan broj kada ga izvadi iz zapisa." [Hip20]

Recimo da je gornja tvrdnja vrlo moguća, s obzirom na to da vrijednost koja nam vidno nedostaje izgleda kao da ima y koordinatu jednaku nuli. Napravimo brzu analizu u hex uređivaču s ciljem pronalaska čestog zaglavlja zapisa.

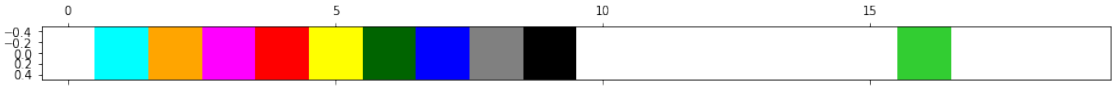


Slika 16: Prikaz obrasca zaglavlja koordinata u hex uređivaču

Pretpostavka je bila da ćemo pronaći nekakav uzorak uz niz od dva broja sedam, s obzirom na to da se takav slijed pokazao uspješnim. Osim što vidimo da je dužina zaglavlja četiri, sada smo sigurni da je primarni ključ ujedno i identifikator retka jer je identificiran serijskim tipom nula. No, mi ne tražimo primarni ključ, ali ovaj podatak nam je vrlo važan, jer sad znamo da će četvorku uvijek slijediti broj nula.

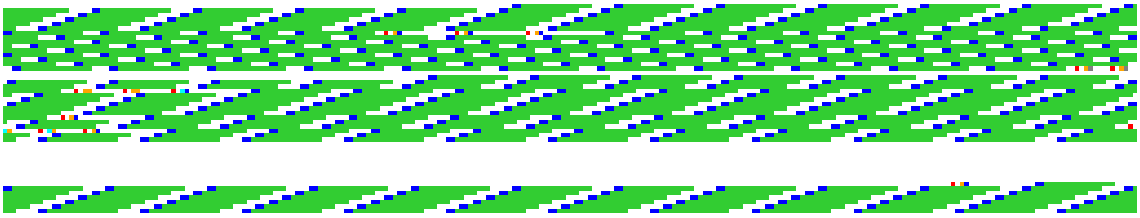
Koristit ćemo metodu vizualizacije kako bi potvrdili navedenu tvrdnju. Recimo da uzmemo polje definirano za prijašnju vizualizaciju, i definiramo boje za ostale moguće serijske tipove cijelog broja. Cijeli broj može biti identificiran serijskim tipom između jedan i šest. Osim toga, gotovo je sigurno da su brojevi nula i jedan pohranjeni u optimiziranom obliku serijskog tipa osam i devet, kao i u primjeru identifikacije RGB vrijednosti. Odredimo nasumičnu mapu boja za tražene brojeve, zadržavajući prijašnju definiciju boja za brojeve sedam i šesnaest.

```
def identify_missing_coordinate_header_colors():
    bounds = [0, 0.99, 2, 3, 4, 5, 6, 7, 8, 9, 10, 16, 17, 255]
    colors = ["white","cyan", "orange", "magenta", "red", "yellow", "darkgreen",␣
 ↪"blue", "gray", "black", "white", "limegreen", "white"]
    cmap = matplotlib.colors.ListedColormap(colors)
    norm = matplotlib.colors.BoundaryNorm(bounds, len(colors))
    return (cmap, norm)

#test
arr = np.arange(20)
cmap, norm = identify_missing_coordinate_header_colors()
fig = plt.figure()
plt.matshow(np.expand_dims(arr, axis=0), cmap=cmap, norm=norm)
plt.show()
```



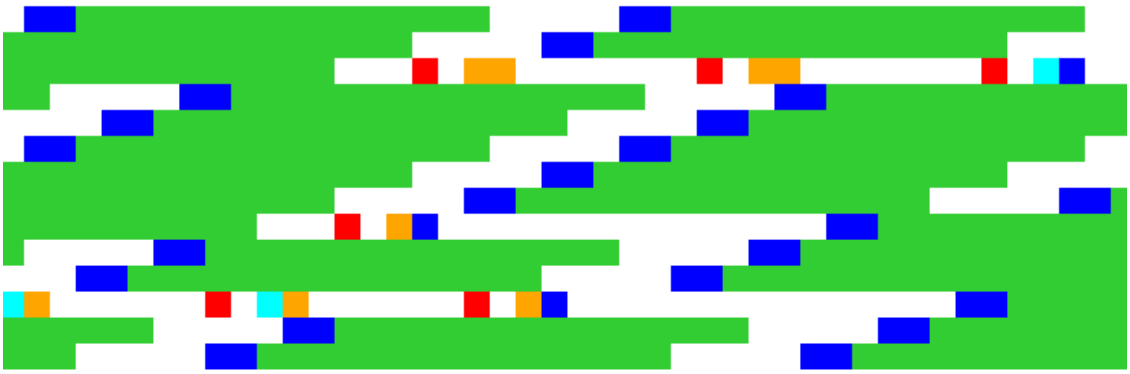
Algoritam pretrage traži niz koji se redom sastoji od broja četiri i nula. Navedeno nam identificira siguran početak zaglavlja. Ako sljedeća dva zapisa odgovaraju brojevima u nizu između jedan i devet, pronašli smo odgovarajuće zaglavlje koje želimo vizualizirati.

```
def identify_missing_coordinate_header_array(file_name, coordinates_array):

    file = open(file_name, 'rb')

    header_length = 4
    bytes_read = file.read(header_length)
    bytes_to_find = np.arange(1, 10)

    while len(bytes_read) == header_length:
        if list(bytes_read)[0 : 2] == [header_length, 0]:
            if all(x in bytes_to_find for x in list(bytes_read)[2:
 ↪header_length]):
                index = file.tell() - header_length
                if not sum(coordinates_array[ index : index + header_length]):
                    coordinates_array[ index : index + header_length ] =␣
 ↪list(bytes_read)

        file.seek(- (header_length - 1), 1)
        bytes_read = file.read(header_length)

    file.close()

    return coordinates_array

coordinates = identify_coordinates_array('zadatak.db')
final_array = identify_missing_coordinate_header_array('zadatak.db', coordinates)
image_generator(final_array, 'missingCoordinatesIdentify',␣
 ↪identify_missing_coordinate_header_colors())
```



Slika 17: Vizualizacija nedostajućih koordinata

Vidljivo je kako vjerojatno postoji više od jedne neidentificirane koordinate. Svako novoidentificirano zaglavlje savršeno popunjava prijašnji prazan prostor. Uzmimo za primjer sljedeći isječak:



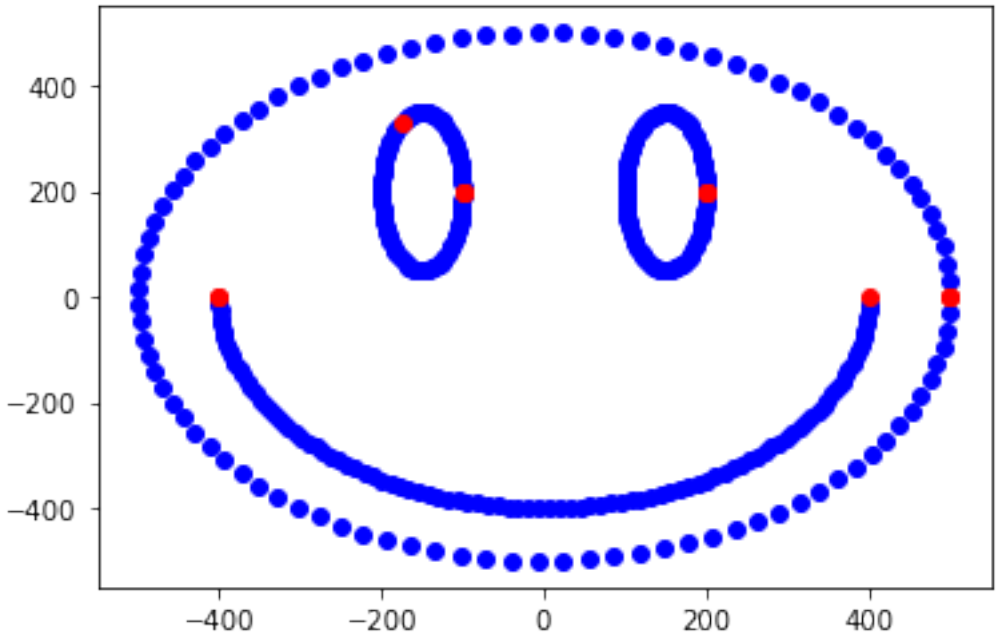
Slika 18: Detalj vizualizacije nedostajućih koordinata

Kao što je i očekivano, neke koordinatne vrijednosti pohranjene su kao cijeli brojevi. U istaknutom isječku, jasno je da postoji više načina pohrane cijelog broja, ovisno o veličini cijelog broja. Promatrajući svaki isječak s novo-identificiranim zaglavljima u potpunosti, zaključak je da su cijeli brojevi pohranjeni u 8-bitnom ili 16-bitnom obliku, i pohrana broja nule optimizirana je na način da ju identificira broj osam. Potrebno je napraviti algoritam vodeći se navedenim zaključcima.

Algoritam pretrage traži početak zaglavlja koji odgovara nizu od redom broja četiri i nule, ali izostavlja zaglavlja koja sadrže niz od dvije sedmice - jer tražimo samo neidentificirane koordinate. Ako ostatak zaglavlja sadrži barem jedan broj iz niza (1, 2, 7, 8), pronašli smo novi zapis.

```
serial_nums_dict = {
    1: { 'size': 1,
         'char_format': 'b'
    },
    2:{ 'size': 2,
```

```
        'char_format': 'h'
    },
    7:{ 'size': 8,
        'char_format': 'd'
    },
}

def identify_missing_coordinates(file_name):
    file = open(file_name, 'rb')

    header_length = 4
    bytes_read = file.read(header_length)
    bytes_to_find = [1, 2, 7, 8]

    coordinates_array= []

    while len(bytes_read) == header_length:
        if list(bytes_read)[0 : 2] == [header_length, 0] and not␣
 ↪list(bytes_read) == [4, 0, 7, 7] :
            if all(x in bytes_to_find for x in list(bytes_read)[2:
 ↪header_length]):
                data_types = list(bytes_read)[2:header_length]
                filtered_list = list(filter(lambda y: y != 8, data_types))

                struct_string = ''.join(list(map(lambda x:␣
 ↪serial_nums_dict[x]['char_format'], filtered_list)))
                bytes_length = functools.reduce(lambda a, b: a +␣
 ↪serial_nums_dict[b]['size'], filtered_list, 0)
                bytes_read = file.read(bytes_length)
                unpacked_values = list(struct.unpack('>' + struct_string,␣
 ↪bytes_read))

                coordinates_array.append(
                    list(map(lambda x: 0 if x == 8 else unpacked_values.pop(0),␣
 ↪data_types))
                )


        else:
            file.seek(- (header_length - 1), 1)
        bytes_read = file.read(header_length)

    file.close()

    return coordinates_array

coordinates = identify_coordinates('zadatak.db')
coordinates_missing = identify_missing_coordinates('zadatak.db')
plt.scatter(*zip(*coordinates), color='blue')
plt.scatter(*zip(*coordinates_missing), color='red')
```



Slika 19: Potpuna rekonstrukcija koordinatnog sustava

Jasno je da su pronađene sve koordinatne vrijednosti, i rekonstrukcija se napokon može smatrati u potpunosti uspješnom.

7 Zaključak

Pomoću korištenih prilagođenih metoda i alata, uspješnih i neuspješnih pokušaja - rekonstrukcija nepoznatih struktura složenih podataka napokon je završena. Metodičkim postupcima, pronašli smo obrazac koji funkcionira nad jednim problemom, postepeno ga nadograđivali i naposljetku došli do potpunih rješenja oba problema.

S obzirom na to da se radilo na datoteci baze podataka, najbitniji identifikator lokacije traženih podataka ubrzo se uspostavilo da je samo zaglavlje. Zaglavlja imaju toliko specifičan poredak, da ako su uspješno identificirana - sigurno nakon njih slijede traženi podaci. Uz tu pretpostavku, identifikacija podataka u drugim bazama podataka ne bi trebala biti veliki izazov, naravno ako dovoljno točno pretpostavimo format zapisa.

Ono što nismo uzimali kao opciju su zaglavlja samih stranica zapisa. Ta zaglavlja sadrže informacije o pohrani podataka u stranice i, ako su zaglavlja čitljiva, donose veliku vrijednost u potrazi - pogotovo tekstualnih podataka.

Slijedno tome, sljedeći pametniji korak bio bi izrada algoritma strojnog učenja, koji sam identificira ono što smo mi tražili ”ručno”, i pametnim treningom, nauči uspješno rekonstruirati svaku datoteku koja mu se nađe na putu.

Literatura

[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.

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

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

[Wan+17] Jianguo Wang et al. "An Experimental Study of Bitmap Compression vs. Inverted List Compression". In: *Proceedings of the 2017 ACM International Conference on Management of Data*. SIGMOD '17. Chicago, Illinois, USA: Association for Computing Machinery, 2017, pp. 993–1008. ISBN: 9781450341974. DOI: 10.1145/3035918.3064007. URL: https://doi.org/10.1145/3035918.3064007.

[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.

[Hip20] Richard D Hipp. *SQLite*. Version 3.31.1. 2020. URL: https://www.sqlite.org/index.html.

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

Popis slika

Popis tablica