Umelá inteligencia

Dokumentácia na zadanie 2a – klasifikácia

Radoslav Muntág

Cvičiaci: Ing. Jakub Abrahoim

Contents

1. Úloha

Úlohou je vytvoriť algoritmus na klasifikovanie bodov v 2D priestore. Na začiatku máme definované pole bodov v 2D priestore so šírkou a výškou v intervale -5000, 5000. Každý bod má unikátne súradnice (x, y). V tomto priestore je inicializovaných 20 bodov rozdelených po 5 do štyroch skupín podľa ich farby: red (R), green (G), blue (B), purple (P).

R: [-4500, -4400], [-4100, -3000], [-1800, -2400], [-2500, -3400] a [-2000, -1400]

G: [+4500, -4400], [+4100, -3000], [+1800, -2400], [+2500, -3400] a [+2000, -1400]

B: [-4500, +4400], [-4100, +3000], [-1800, +2400], [-2500, +3400] a [-2000, +1400]

P: [+4500, +4400], [+4100, +3000], [+1800, +2400], [+2500, +3400] a [+2000, +1400]

Do daného priestoru je potrebné pridať 40000 bodov (10000 pre každú farbu), kde každý bod má náhodnú pozíciu.

Červené body majú 99% pravdepodobnosť byť generované v $x < +500$, $y < +500$.

Zelené body majú 99% pravdepodobnosť byť generované v $x > -500$, $y < +500$.

Modré body majú 99% pravdepodobnosť byť generované v $x < +500$, $y > -500$.

Fialové body majú 99% pravdepodobnosť byť generované v $x < -500$, $y < -500$.

V prípade zvyšného 1% sú body generované náhodne v celom priestore.

Tieto body sú potom klasifikované pomocou k-NN algoritmu, tá je abstraktne zobrazená ako funkcia classify(x, y, k) kde x a y sú súradnice bodu v 2D priestore a k je predstavuje počet k najbližších susedov. Z k susedov funkcia vráti predpokladanú farbu nového bodu.

Na klasifikáciu použite k hodnoty 1, 3, 7, 15.

2. Návod na používanie

Requirements

pillow==11.0.0

Možné nainštalovať pomocou pip install pillow najlepšie vo virtuálnom prostredí projektu.

Odporúčam PyPy interpreter pre rýchlejšie vygenerovanie väčších vstupov.

Spustenie

Pred spustením je možné nakonfigurovať súbor constants.py

`SIZE = 5000` Veľkosť 2D plochy pre generovanie bodov (spravil som to trochu mätúco a dĺžka strany X a Y je v skutočnosti dvojnásobok premennej SIZE, takže pre hodnotu 5000 je veľkosť priestoru 10 000 * 10 000. x a y je od -5 000 do 5 000)

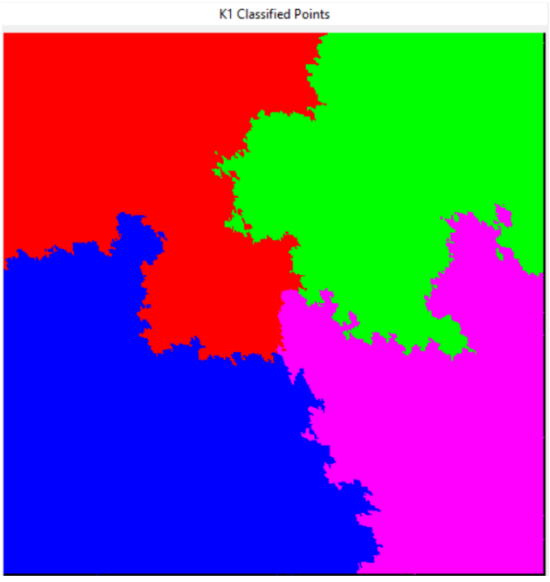
`WIN_SCALE = 0.05` Škálovanie okien pri vykresľovaní bodov. (veľkosť jedného okna je v tomto prípade 500*500, čiže jedna strana je (SIZE*2)*WIN_SCALE)

`GENERATE_BY_SEED = True` Generovanie náhodných hodnôt zo SEED hodnoty.

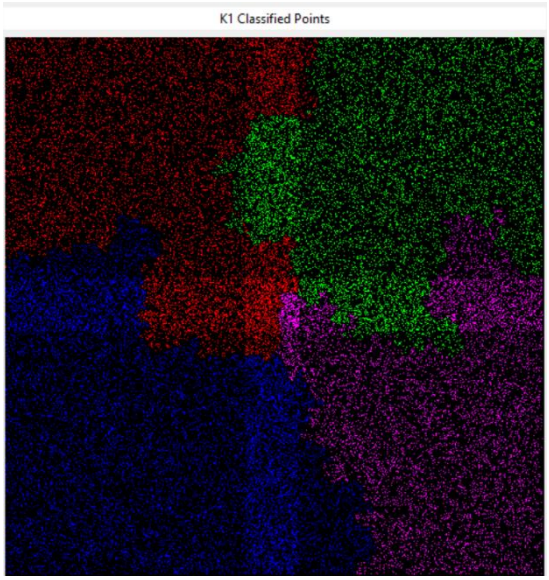
`SEED = 11` SEED môže byť akýkoľvek integer. Pri rovnakom Seede bude vždy rovnaký output.

`FILL_EMPTY_SPACE = False` Ak je True, prázdne miesta vo výstupe pre k hodnoty sú dodatočne klasifikované a priradí sa im farba. (silno ovplyvňuje rýchlosť programu)

Pre hodnotu True:



Pre hodnotu False:



```
POINTS_TO_GENERATE = 10000
```
Počet bodov na vygenerovanie pre **každú farbu**. (pre hodnotu 10 000 sa vygeneruje dokopy 40 000 bodov, nepočítajúc začiatočných 20)

```
K_Values = {
    1: True,
    3: True,
    7: True,
    15: True
}
```
Pre ktoré K hodnoty sa má vygenerovať výstup. Vo výsledku sa vygenerujú 1 + 4 okná pre všetky hodnoty True. 1 okno sú originálne vygenerované farby. 1 okno pre každú k hodnotu. (meniť len boolean hodnoty)

Na spustenie programu sa spúšťa main.py súbor

Výstup

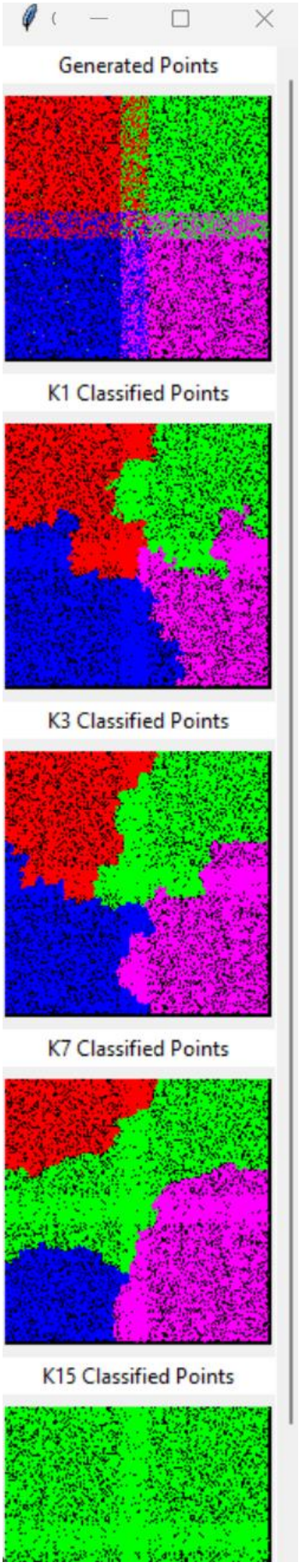
```
K1 accuracy is: 75.8475 %
K3 accuracy is: 77.345 %
K7 accuracy is: 65.25999999999999 %
K15 accuracy is: 25.0525 %
```

Po spustení sa vypíše percentuálna presnosť pre K hodnoty. Presnosť je počítaná z originálne vygenerovaných farieb bodov ako:

(počet bodov s rovnakou farbou ako originál / celkový počet bodov) * 100

Pri hodnotách okolo 25% majú skoro všetky body len jednu farbu.(generované pre SEED = 11)

Výstup vizualizácie je predošle spomenutých 5 plôch podsebnou v jednom okne. Na pravo je slajder pre posunutie plôch.


Generated Points
K1 Classified Points
K3 Classified Points
K7 Classified Points
K15 Classified Points

3. Štruktúra programu

Rozdelenie súborov

V projekte sú súbory:

main.py – inicializovanie začiatočných 20 bodov a spúšťanie všetkých častí programu

constants.py – konfigurovateľné konštanty

point.py – reprezentácia jedného bodu a jeho generovanie

algorithms.py – vlastné implementácie k-NN algoritmu a KD-tree dátovej štruktúry/algoritmu + využitie knižnice heapq pre priority Queue

visualization.py – využitie knižnice tkinter a pillow pre generovanie K plôch a hlavného okna

Reprezentácia bodov – class Point

Nová inštancia bodu príma len farbu, s ktorou má byť bod vygenerovaný. Farba má tvar enum, kde hodnotou tohto enumu je tuple reprezentujúci tri RGB farby v intervale 0-255 (príklad: RED = (255, 0, 0)).

Jeho pozícia je potom generovaná podľa tejto farby pomocou metódy determine_position(color: Color). Bod ma 1% pravdepodobnosť nadobudnúť náhodnú pozíciu v celom priestore, inač je jeho pozícia generovaná podľa zadania.

Po vygenerovaní bodu, môže byť jeho pozícia natvrdo prepísaná pomocou metódy hard_set_position(x: int, y :int)

Inštancia bodu udržiava 5 farieb ako jej atribút.

Originálne priradená farba:

self.color

Priradená farba classify funkciou pre každú hodnotu K:

self.k1_color

self.k3_color

self.k7_color

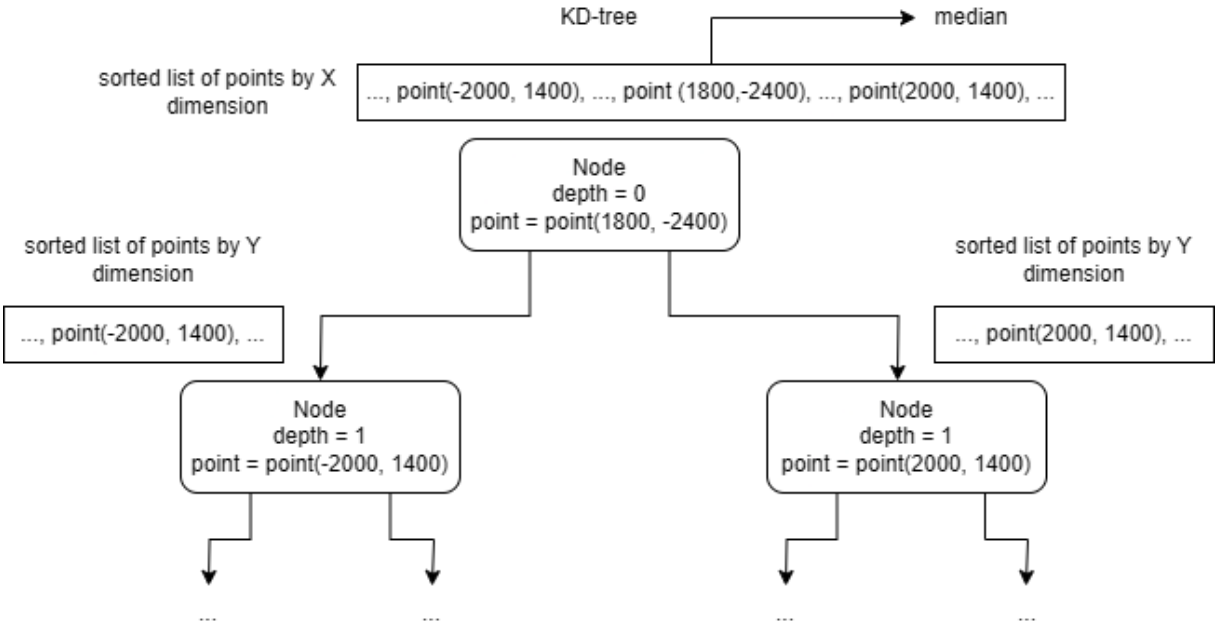
self.k15_color

Algoritmy

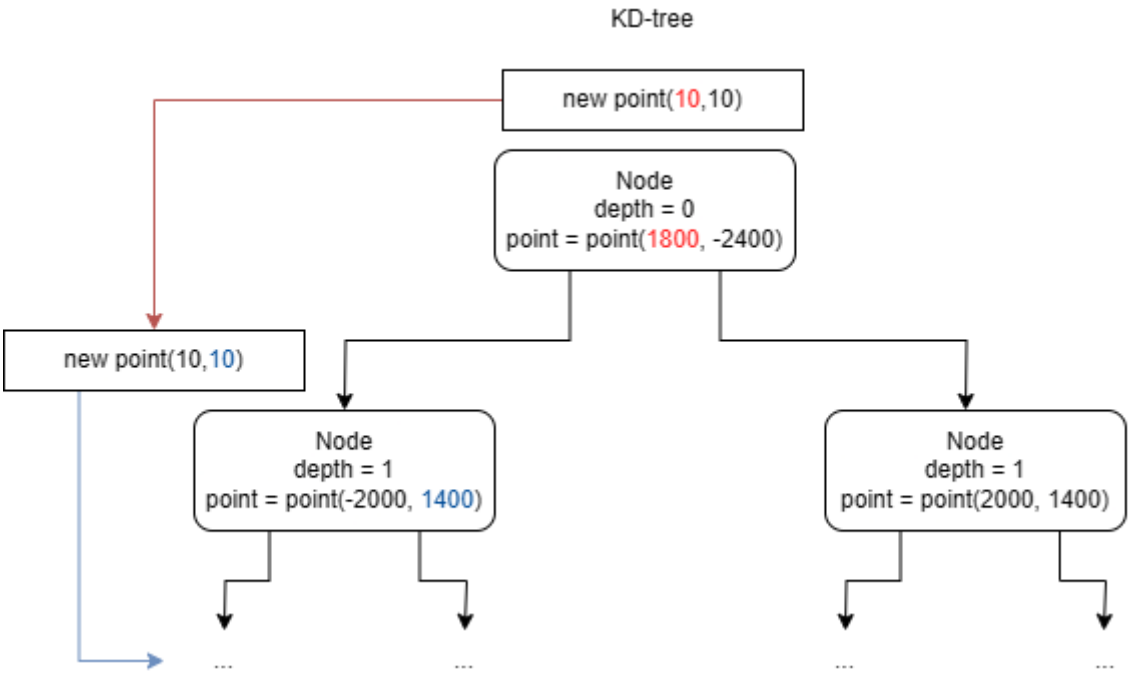
V programe sú 2 vlastne implementované algoritmy k-NN a KD-tree. Tieto dva algoritmy sú svojím spôsobom skombinované do jedného pre lepšiu optimalizáciu. Myšlienka za tým je, že keď sa program rekurzívne vnára do KD-stromu, aby vložil nový bod, tak pri vynáraní sa rovno klasifikuje a predídeme zbytočnému dvojitému vnáraniu a vynáraniu pre každý bod.

KD-tree

Strom sa inicializuje pomocou listu 20 začiatočných bodov. Prvý node v strome má hĺbku 0. Získame bod, ktorý reprezentuje tento node tak že zoradíme list podľa x dimenzie pozície bodov (dimenzia je určená ako hĺbka % 2, kde 0 je x a 1 je y) a získame index mediánu bodu v liste. Rekurzívne potom do ľavej vetvy inicializujeme strom s bodov, ktoré majú x pozíciu menšiu ako medián a do pravej inštanciu stromu so zvyšku listu.



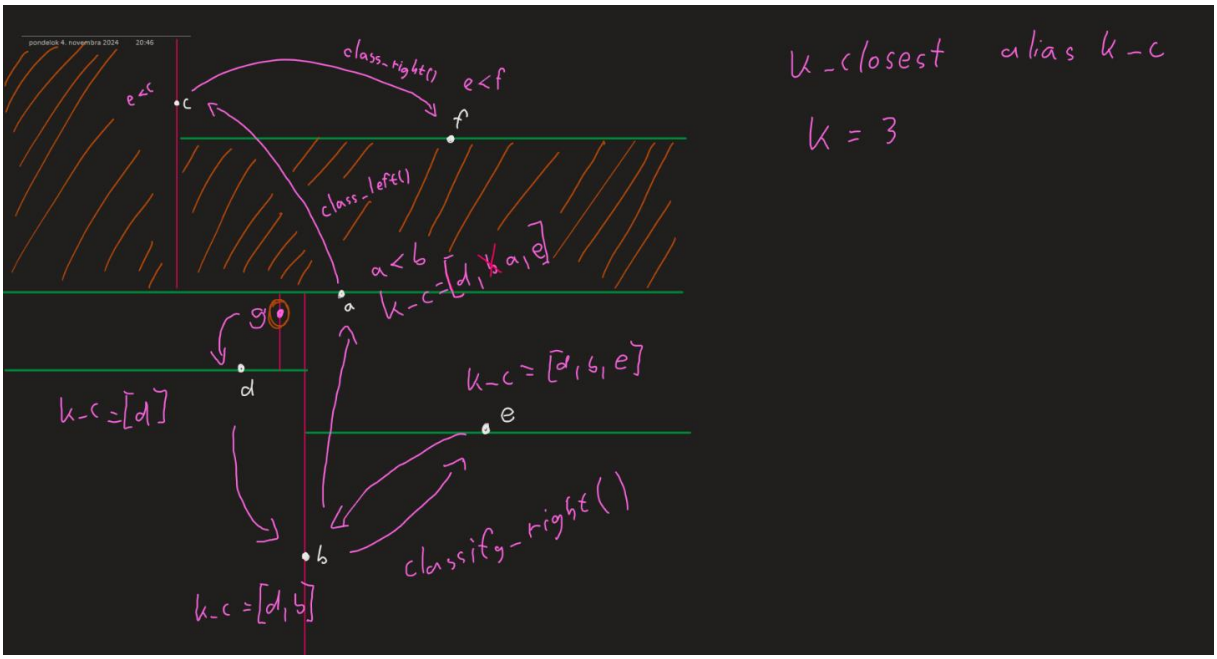
Pri vkladaní nového bodu do stromu sa hľadá leaf node, ktorý má hodnotu None. Toto dosiahne pomocou rekurzívneho volania metódy insert_point(point: Point, k: int). Najprv zavoláme metódu insert_point() na root node stromu s bodom, ktorý chcem vložiť do stromu. V metóde sa porovná dimenzia pozície, podľa hĺbky, nového bodu s bodom v node. Ak je menšia zavola sa inser_point na novde vľavo, inač sa zavolá v node vpravo. Tento cyklus sa opakuje, kým sa nenájde leaf node.



Vynáranie z KD-tree a hľadanie k-NN

Tu začína problém. Po nájdení miesta pre bod na leaf node, začne vynáranie z metódy insert_point(). Metóda insert_point() vracia heap queue list k_closest, ten sa postupne napĺňa k najbližšími bodmi. V k_closest sú tuple[distance_from_point, point].

Pri každom vynorení rekurzívne prehľadávame ostatné vetvy (ak sa vynoríme zprava prehľadávame pravú a naopak) pomocou metód classify_left(point: Point, k_closest: list[tuple[float,Point]]) a classify_right(point: Point, k_closest: list[tuple[float,Point]])). Tieto metódy sú takmer identické, ako z názvu vyplíva jedna prehľadáva ľavú vetvu a druha pravú. V každej sa rekurzívnie voľajú obidve znova ak k_closest ešte nebol naplnený alebo ak vzdialenosť vetvy po dimenzionálnej dĺžke nie je väčšia ako najväčšia vzdialenosť v naplnenom k_closest, v opačnom prípade sa prehľadáva len strana priľahlá k nášmu bodu (tu sa ešte môžu nachádzať bližšie body).



Tu je vizualizácia vynárania po pridaní bodu **g** kde bod **a** je root. k_closest s k = 3 po dokončení by bolo [**d**, **a**, **e**]. Oranžovo vyznačené časti sú časti (nie všetky), kde by napríklad bolo ešte treba pozerať či tam náhodou nie je ešte nejaký bod bližší ako bod **e**. Toto je z dôvodu že aj keď vzdialenosť od **g** po **c** je väčšia ako po **e**, vzdialenosť po x dimenzií k **c** od **g** je menšia ako vzdialenosť od **g** po **e** a mohol by sa tam nachádzať nejaký bližší bod. V prípade **g** po **f**, dajme tomu že vzdialenosť po dimenzií y je od **g** po **f** už je väčšia ako od **g** po **e** (aj keď to z obrázku nevyplýva), tak stále by sme museli prehľadať priľahlú stranu od **f** ku **g**. lebo tu sa stále môže nachádzať bod bližší ku **g** ako je bod **e**.

Classify

K je vždy nastavené na 15, aby mohol program z neho po vynorení mohol vyberať všetky k hodnoty (1,3,7,15). A to je všetko čo spraví. Urobí *„histogram“* z k-tych farieb bodov pre danú k hodnotu a vyberie najčastejšiu. Ptom vybere najčastejšie vyskytujúcu sa farbu a priradí ju bodu pre danú k_color.

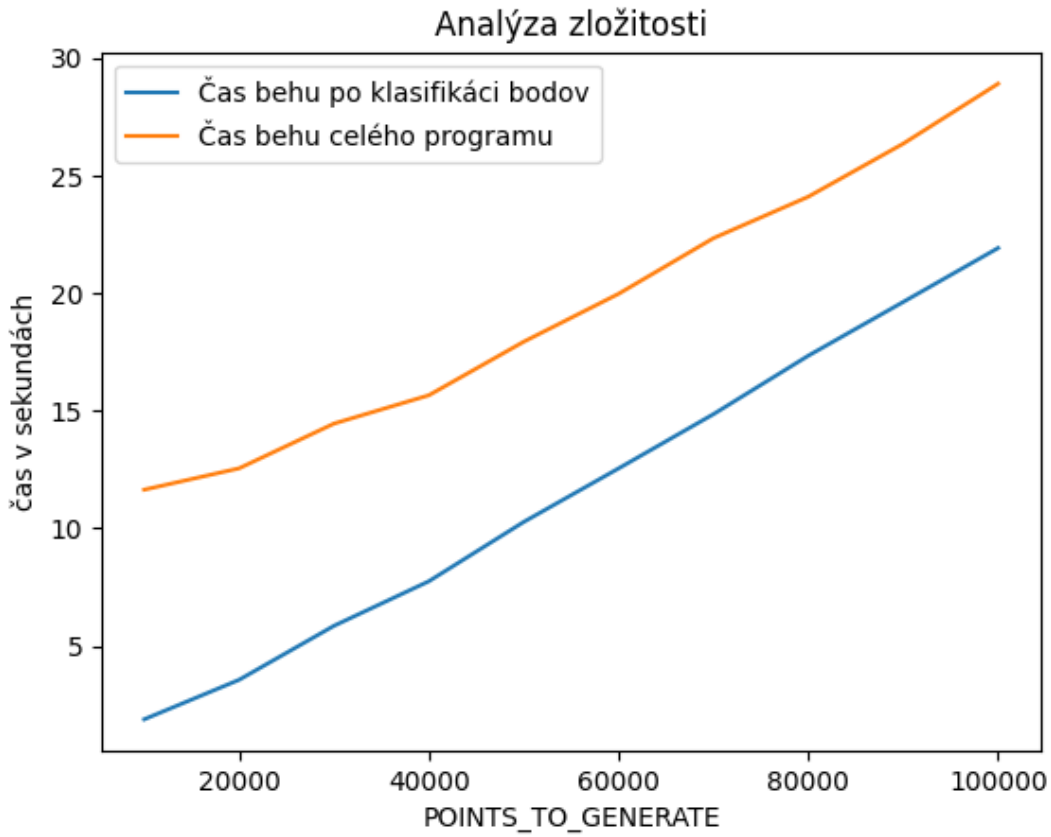
4. Vizualizácia

Vo vizualizácii sa nachádza ešte vyplňovanie prázdnych miest, ak to bolo predtým nakonfigurované. Tu sa nájdu body, ktoré sú prázdne a pridajú sa na klasifikáciu. Toto môže odzrkadliť na finálnom čase.

Potom sa body len postupne vykreslia.

5. Analýza zložitosti programu

Graf reprezentuje časovú zložitosť programu. Skutočný počet bodov je POINTS_TO_GENERATE * 4. Napríklad pre vstup 100 000 je v skutočnosti počet klasifikovaných bodov 400 000.



Modrá predstavuje čas potrebný skoro len pre k-NN algoritmus a KD-tree. Oranžová čiara je čas behu celého programu aj s dodatočným klasifikovaním pre prázdny priestor.

Je zaujímavé vidieť že pre originálnych 4*10 000 bodov je čas pre len pre klasifikáciu len 1.86 sekundy zatiaľ čo pre celý program 11.6 sekundy. Tento rozdiel sa však zmenší s väčším vstupom a pre 4*100 000 bodov to už je 21.92 sekundy ku 28.9 sekundy.

Program však bežal pomocou pypy. V normálnom pythone narástol čas pre 4*100 000 bodov na 71.62 sekundy len pre klasifikáciu a na 82.33 sekúnd pre celý program.

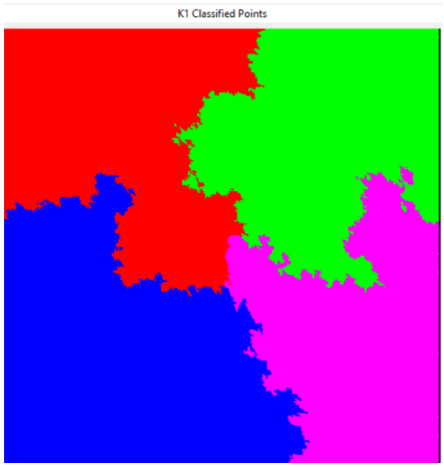
6 Záver

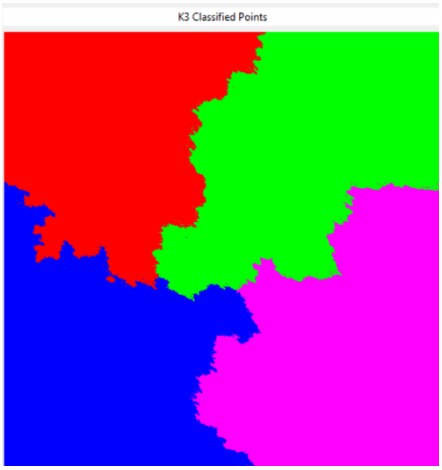
Na záver vyhodnotím niekoľko pokusov.

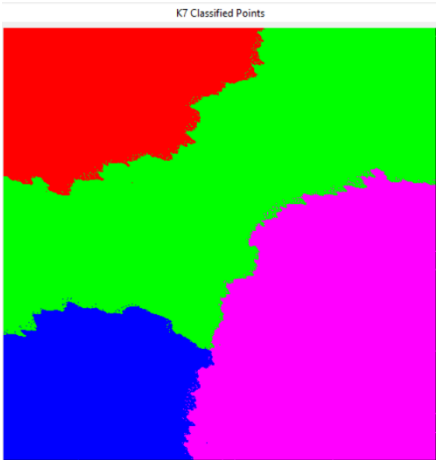
Seed 11

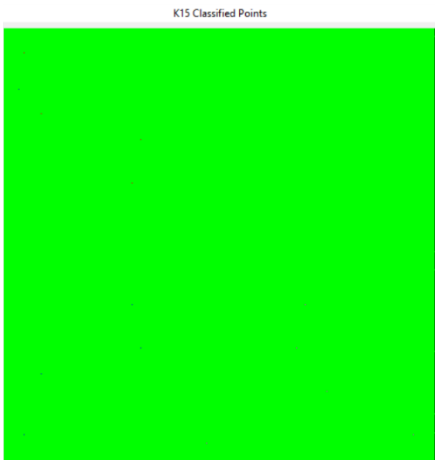
Pre seed 11 som našiel výstup k15, ktorí je celý zelený. Zatiaľ čo k1 je celkom rovnomerne rozložená.

```
K1 accuracy is: 75.8475 %
K3 accuracy is: 77.345 %
K7 accuracy is: 65.25999999999999 %
K15 accuracy is: 25.0525 %
```





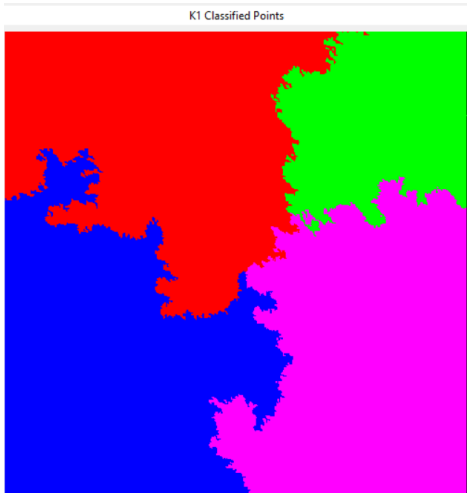


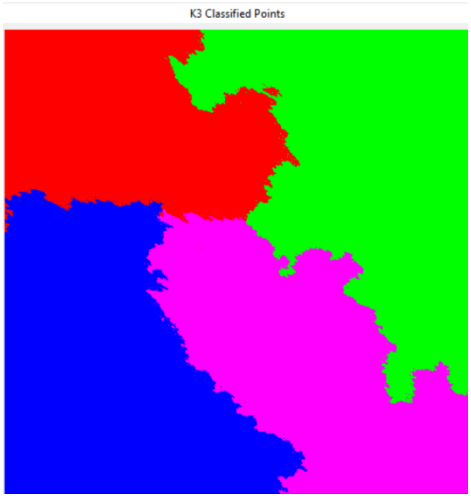


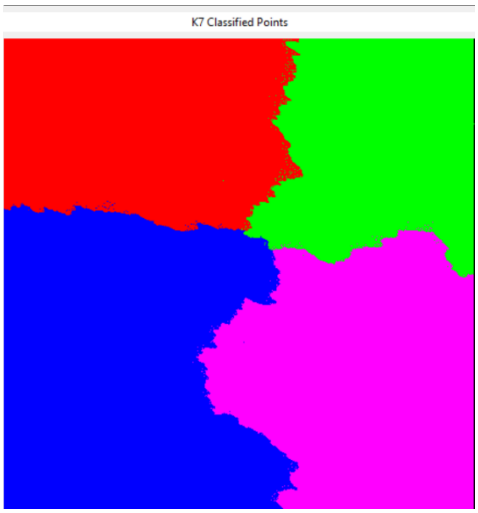
Seed 24

Pre seed 24 som odpozoroval, že k7 má väčšiu presnosť ako všetky ostatné.

```
K1 accuracy is: 73.32 %
K3 accuracy is: 71.295 %
K7 accuracy is: 76.63250000000001 %
K15 accuracy is: 42.1625 %
```







SEED 19

V seed 19 pre k15 prežili všetky farby.

```
K1 accuracy is: 76.125 %
K3 accuracy is: 73.1175 %
K7 accuracy is: 73.985 %
K15 accuracy is: 60.64000000000001 %
```