SLOVENSKÁ TECHINCKÁ UNIVERZITA V BRATISLAVE

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

ZADANIE 4

Clustering

UMELÁ INTELIGENCIA

Samuel Bubán

ID: 102879

10.12.2020

ZADANIE:

Kategorizácia neznámeho zadaného datasetu (súradnice bodov na ploche) do skupín, podľa vzdialenosti, pomocou algoritmov: K-means, Agglomerative clustering, Divisive clustering.

RIEŠENIE:

Na vygenerovanie datasetu som využil samostatný program – **Generator.py**. Tento program obsahuje tri funkcie na generovanie bodov.

generate_vertices() – generuje náhodné unikátne body v zadaných hraniciach.

generate_square_clusters() – generuje body podľa zadaných parametrov. Najprv vygeneruje pivotné náhodné body, a okolo týchto bodov následne generuje štvorcové zhluky, podľa maximálneho polomeru štvorca. To znamená, že body sa môžu generovať aj mimo hraníc, čo ale nie je problém, ináč by boli urezané.

generate_fancy_clusters() – generuje body rovnako ako predchádzajúca funkcia. Rozdiel je v tom, že ich generuje v kruhovom zhluku, namiesto v štvorcovom. Tieto body sú najprv vygenerované rovnomerne v kruhu, a následne sú roztiahnuté do priestoru pomocou funkcie x^10.

Algoritmy sú uložené v programe **Algorithms.py**, spolu s pomocnými funkciami na výpočet vzdialenosti, centra zhlukov ...

k-means() - Na začiatku vygeneruje počiatočné centrá zhlukov. Sú generované, ako náhodný bod z datasetu. Toto generovanie som zvolil z toho dôvodu, aby sa predišlo tomu, že by niektoré zhluky zostali prázdne.

Keď sú vygenerované centrá, prepočítajú (kategorizujú) sa všetky body z datasetu k najbližšiemu centru. Následne sa cez parameter **recalculate_clusters** zvolí nové centrum zhlukov. Môže ísť o jednu z dvoch funkcii: **recalculate_clusters_average()** a **recalculate_clusters_medoid()**. Prvá funkcia prepočíta centrá na priemernú hodnotu zo zhluku. Súčet x-ových súradníc a súčet y-ových súradních predelených počtom bodov v zhluku. Druhá funkcia pracuje zložitejšie – pre všetky body v zhluku vypočíta sumu vzdialeností ku všetkým bodom z tohto zhluku. Následne vyberie ten bod, ktorý má najmenšiu vzdialenosť ako nové centrum daného zhluku.

Tento cyklus sa opakuje, až kým nepríde medzi dvoma iteráciami k žiadnej zmene, alebo kým sa dosiahne horná zadaná limitná hranica. Funkcia vráti dve polia – vývoj zhlukov, a centier zhlukov podľa iterácie.

Agglomerative: tento problém som riešil dvoma spôsobmi. Oba spôsoby vracajú iba výsledné usporiadanie bodov, nakoľko celý vývoj by bol aj časovo aj pamäťovo o dosť zložitejší.

agglomerative_brute() – táto funkcia má neskutočne vysokú časovú zložitosť, ale za to má nízku pamäťovú zložitosť. Problémom tejto funkcie je, že všetko musí vždy počítať znovu.

Na začiatku sa všetky body priradia do vlastného zhluku. Potom sa prehľadajú všetky zhluky, a porovnajú sa ich vzdialenosti Nájdu sa tie, ktoré majú najmenšiu vzájomnú vzdialenosť a spoja sa dokopy. Toto sa opakuje až kým nie je dosiahnutý počet zhlukov rovnaký, ako bol zadaný cez parameter.

Odhadovaný čas pre 20 000 bodov – viac ako 20 dní počítania.

Funkcia berie tiež ako parameter funkciu na vypočítanie stredu zhluku, a môže byť centroid alebo medoid.

agglomerative_matrix() – nakoľko predchádzajúce riešenie neprichádzalo do úvahy na zvolený problém, zvolil som aj túto druhú možnosť, ktorá je presne opačná. Časová zložitosť je minimálna, ale zas pamäťová zložitosť je veľmi vysoká.

Táto funkcia bola veľmi silno a brutálne optimalizovaná, aby zaberala čím menej priestoru, a pracovala čím rýchlejšie.

Prešla si viacerými verziami, nakoľko žiadna nebola dostatočne dobrá na náš problém.

Prvá verzia nebola úplne správna.

PRVÁ VERZIA: Na začiatku sa vytvorí dictionary, teda hash tabuľka zhlukov. Toto je potrebné pre to, aby sa zachovali indexy v tabuľke aj po tom, čo sa niektorý vymaže – aby sa neposunuli. Zároveň sa vytvára aj tabuľka indexov, ktorá udržuje informáciu o tom, ktorý bod sa nachádza v ktorom zhluku.

Ďalej je pamäťovo problematická časť, kedy sa vygenerujú všetky kombinácie, ako by sa mohli spojiť dva body. Tento zoznam je následne zoradený, podľa toho, akú vzdialenosť majú tieto dva body od seba. V kombináciach sa nachádzajú len indexy bodov, aby sa zabezpečila optimalizácia pamäte. Vzdialenosť týchto dvoch bodov sa teda musí vypočítať tak, že sa z pôvodného datasetu vyberú dva body na zadaných indexoch (uložené informácie), a tieto sa porovnajú.

Teraz, keď už sú všetky body zoradené, tak sa začne postupne cez všetky prechádzať. Najprv treba určiť, z ktorého zhluku tieto dva indexy (body) sú. Na to sa použije vytvorená tabuľka s indexami. Ak sú tieto dva body v rozdielnych zhlukoch, tak sa zhluky zlúčia. Všetky body zo zhluku, v ktorom sa nachádza menej bodov majú aktualizovaný index, že v ktorom zhluku sa nachádzajú, a následne je tento zhluk vymazaný zo zoznamu.

Tu je využité to, že sú to všetko hash tabuľky, a teda tieto operácie majú časovú zložitosť len O(1), teda hit or miss.

Cyklus sa znovu opakuje, až kým nie je počet zhlukov rovnaký ako požadovaný počet. Následne sa vypočítajú centrá zhlukov podľa zadanej funkcie, a toto sa vráti ako výstupná hodnota.

Táto funkcia má veľmi vysokú pamäťovú náročnosť, a pre 20 000 bodov vyžaduje približne 15 GB. Moja odhadovaná pamäťová zložitosť pre tento cyklus bola rovnaká, čo sa mi aj približne potvrdilo pri sledovaní využitia RAM a disku v pc. Časová zložitosť je výborná, a aj keď nemá dostatok pamäte (časť dát bude uložená na disk), tak je schopná vypočítať dataset o veľkosti 20 000 za približne 10 minút.

DRUHÁ VERZIA: Nakoľko predchádzajúce riešenie nebolo správne, musel som to celé prerobiť. Chcel som zachovať rýchlosť funkcie, preto som použil dvojitý dictionary, ktorý slúžil na ukladanie clusterov, a ešte druhý dictionary na centrá. Dictionary som využil preto, aby sa nemenili počas behu indexy clusteru, aj keď by sa zlúčil, a niektorý sa vymazal.

Problém ale bol, že to zaberalo neskutočne veľa pamäte, a aj časovo to bolo slabé – trvalo by to asi 10 dní na celkové vypočítanie.

TRETIA VERZIA: Preto som sa rozhodol zmeniť dictionary na polia. Musel som trochu pomeniť indexáciu, ale toto urýchlilo program, že by sa požadované riešenie vypočítalo asi za 3 dni. Stále málo.

ŠTVRTÁ VERZIA: V pokuse o urýchlenie som zmenil svoju vlastnú funkciu na nájdenie minima na už preddefinovanú min(), spolu v kombinácii s funkciou index() na nájdenie daného indexu v poli. Riešenie sa urýchlilo asi 6-násobne, ale stále by trvalo asi pol dňa, čo si stále nemôžem dovoliť.

PIATA VERZIA: Na odporúčanie kolegov som skúsil druhý Python interpreter PyPy. Celé riešenie sa zrýhlilo 6-násobne, a teda teraz trvá agglomerative clustering asi hodinu.

divisive() – toto je asi najjednoduchšia funkcia. Na začiatku sa vytvorí prvotný zhluk bodov pomocou funkcie k-means, a rozdelí sa na polovicu.

Cyklus: vyberie sa zhluk bodov s najväčšou odchylkou a vykoná sa na ňom k-means. Toto sa opakuje, až kým nie je počet zhlukov taký, ako sa požaduje.

Funkcia tiež vráti vývoj zhlukov a ich centier podľa iterácie, ale nevracia čiastočné hodnoty (výstup z k-means), iba ich poslednú iteráciu.

Centrá bodov sú znovu počítané podľa zadanej funkcie (centroid, medoid).

GUI:

Na vizualizáciu výsledného riešenia som použil knižnicu tkinter. Uložené je to v programe **gui.py**. Najprv je možné si zvoliť všetky parametre – hranice, počet zhlukov, počet bodov, rozptyl, typ generátoru, algoritmus a aj funkciu na výpočet centra.

Správnosť vstupu je kontrolovaná pomocou funkcie z programu inputcheck.py.

Následne sa spustí zvolený algoritmus, a keď bude dostupný výsledok, otvorí sa plocha. Veľkosť plochy je silno prepočítaná, aby nebola veľká, a aby sa tam zmestili akurát body. Čaká na vstup, teda enter v konzole, a potom sa vykreslí ďalšia iterácia.

Program podporuje ľubovoľnú veľkosť a počet bodov (sú obmedzené zadanými hranicami, ale tie by sa dali zmeniť).

TESTOVANIE:

Na testovanie je samostatný program: **test.py**.

Na porovnávanie algoritmov som použil priemer ich priemerných vzdialenosť všetkých bodov od centier zhluku, a čas trvania algoritmu. Opakoval som každú kombináciu 10x, na získanie priemerných hodnôt, pričom dataset zostával medzi algoritmami rovnaký, nakoľko niektoré funkcie vyberajú počiatočné hodnoty náhodne.

Testoval som to na počet zhlukov 5, 10 a 20, pre počet bodov od 550 do 1500.

Ako sa aj dalo čakať, k-means má dosť nepresné výsledky, nakoľko berie náhodné hodnoty na začiatku. Aglomeratívne zhlukovanie má výsledky dobré, a po optimalizácii s maticou aj lepší čas, ale vyššie pamäťové nároky. Preto najlepšie vyšlo asi divizívne zhlukovanie, ktoré má aj výborné výsledky aj výbornú časovú zložitosť.

Pre počítanie s medoidmi rastie časová zložitosť kvadraticky, pre aglomeratívne brute-force rastie dokonca až kubicky.

Grafy sa nachádzajúv excel dokumente.

ZHODNOTENIE:

Osobne si myslím, že toto posledné zadanie za veľa nestálo, a bolo veľmi nevyvážené. Mnoho z kolegov mali svoju časť (a-čko) hotovú v priebehu hodiny až dvoch, a zvyšok času trávili na dolaďovaní už doslova blbostí – krajšie gui, nevykresľovanie okrajov, vyfarbovanie pozadia ...

Základ celého zadania je na úrovni strednej školy. Ešte v treťom ročníku na gymnáziu sme robili obdobné úlohy, kedy sa generovali body, a na základe nejakých podmienok sa im nastavila farba... Samozrejme sme to nerobili pre 40 000 (respektíve 20 000) bodov, ale asi 100.

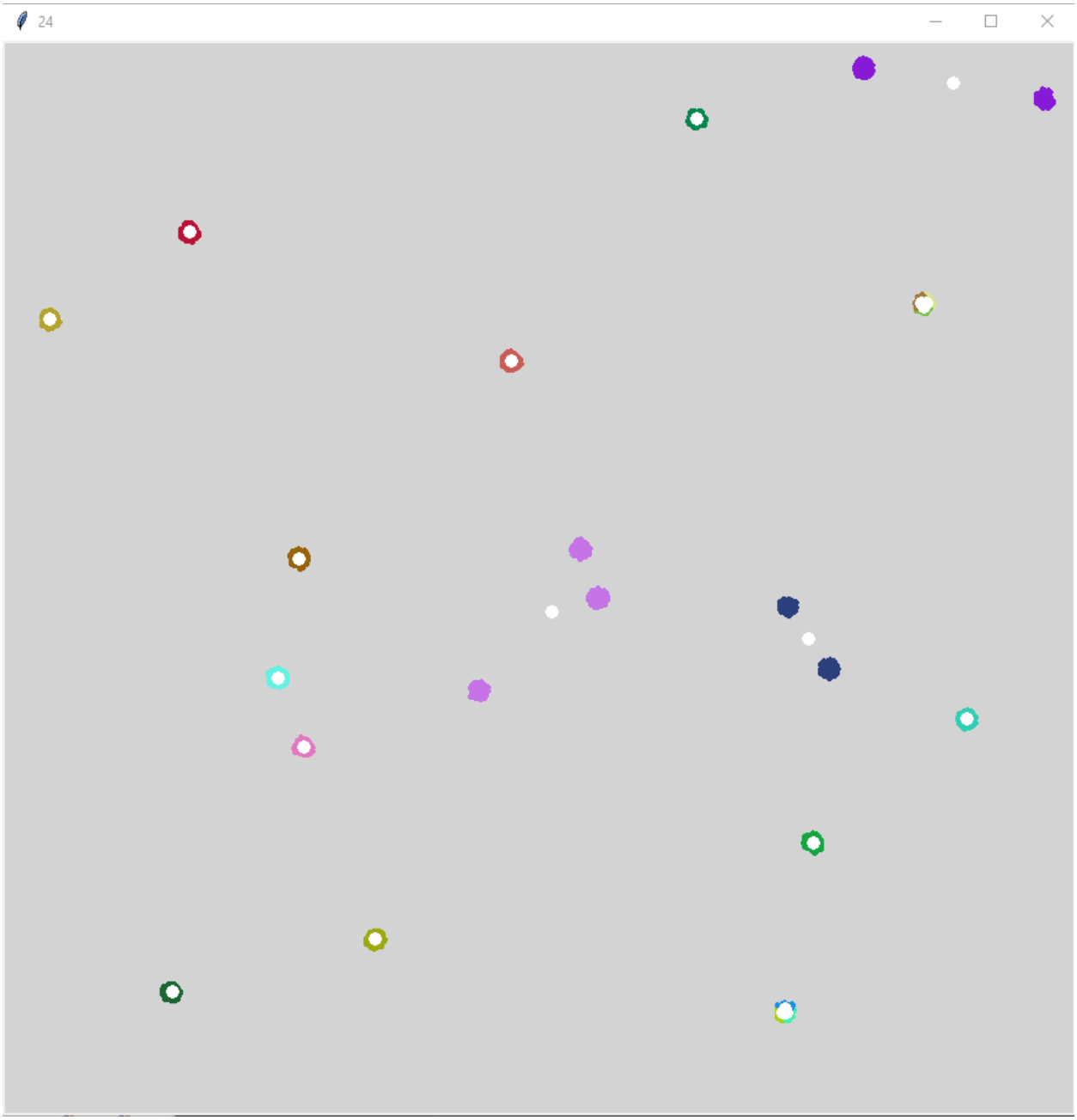
Na druhú stranu je zadanie b, ktoré obsahuje 3-4 algoritmy. K-means a divisive clustering by nebol problém, nakódené v rovnakom časovom rozsahu ako zadanie a-čka, ale agglomerative clustering bol úplne škaredý algoritmus na riešenie. Namiesto toho, aby som rozmýšľal nad tým, ako algoritmus funguje, alebo niečo podobné, som viac ako polovicu času strávil tým, že som ho musel optimalizovať, lebo by mi riešenie neprešlo na zadanej vzorke, ktorá je znovu neprimerane obrovská. Musel som si prejsť cez mnoho rôznych kombinácii štruktúr, a pri každej vyhodnocovať ich efektivity, (dict of arrays, array of dicts, dict of dicts, dict of dicts with another dict ...)

Pre aglomeratívne zhlukovanie nie je žiadnou podmienkou na splnenie zadania vykonávať hocijaké optimalizácie, ale bez optimalizácii nie je ŽIADNA šanca, že by algoritmus prešiel brute-force riešením v normálnom čase. Keďže treba robiť testovanie, za normálny čas považujem maximálne tak 10 minút – aby sa dali spraviť testy, ktoré budú deň vkuse bežať, a postupne dajú nejaké výsledky na porovnanie.

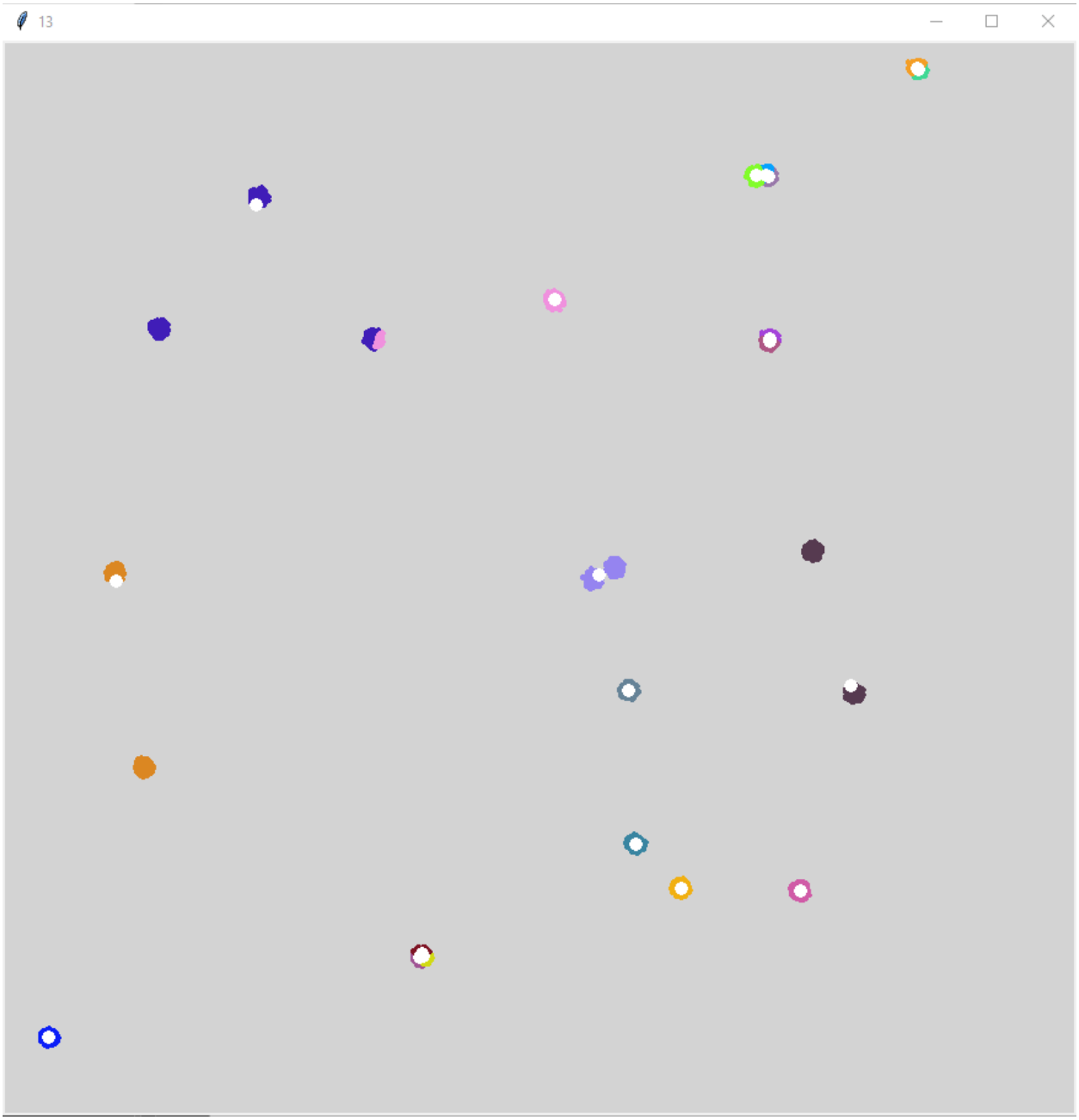
Na porovnanie – aglomeratívne zhlukovanie cez brute-force bez hocijakých optimalizácii by trvalo na datasete o veľkosti 40 000 prvkov niekoľko týždňov, až mesiacov (áno, robil som si test), a pre 20 000 prvkov by to bolo niekoľko dní.

VÝSLEDKY:

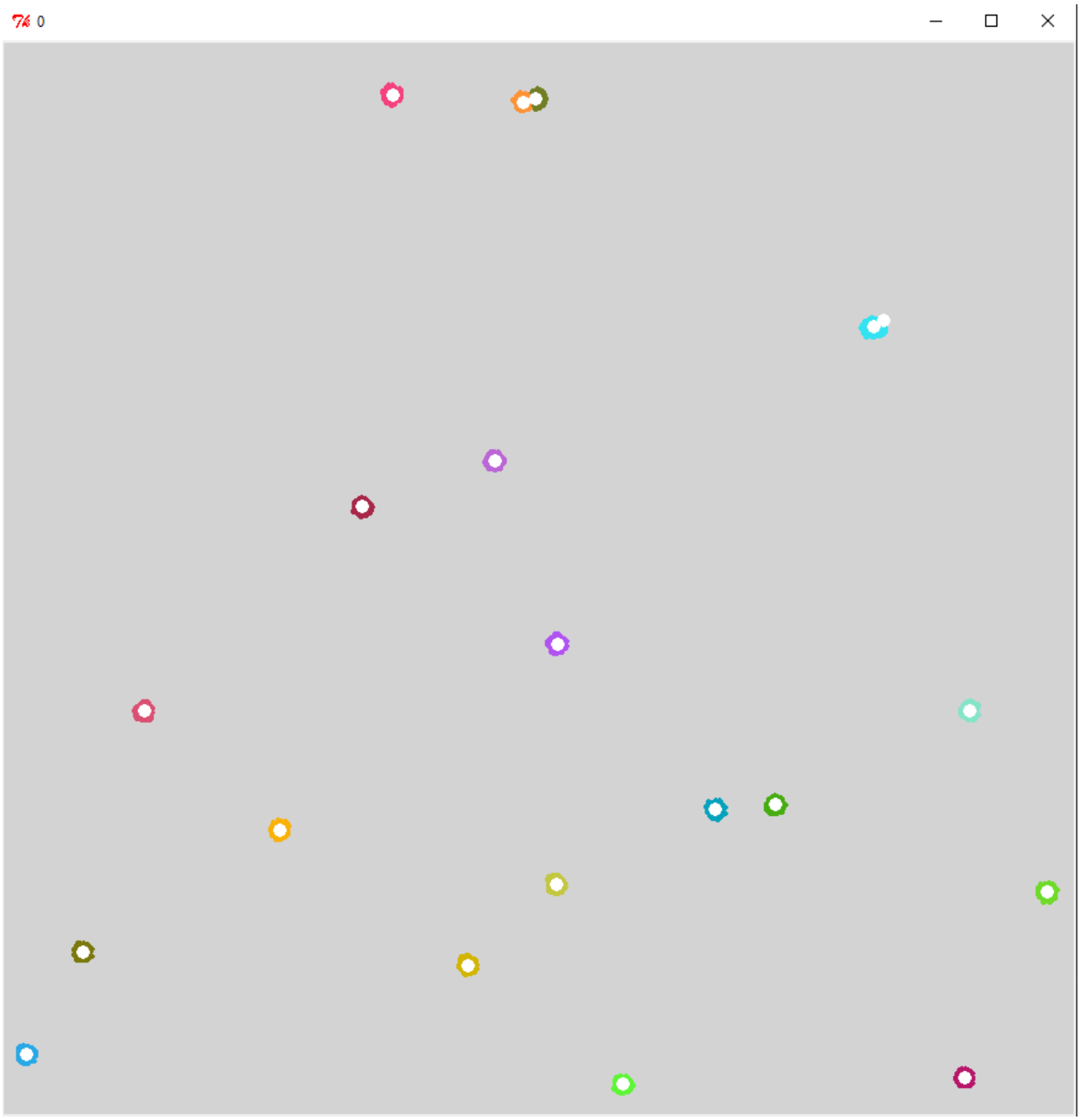
K-means centroid:



K-means medoid:



Agglomerative matrix centroid:



Divisive centroid: