СОДЕРЖАНИЕ

РУБЕЖНЫЙ КОНТРОЛЬ

Постановка задачи

Оптимизировать код кластеризации методом k-means, не задействуя массивов, кроме массива расстояний от одной точки до каждого центроида и массива номеров кластеров длиной M, где M - количество трёхмерных точек на входе алгоритма (точки для работы можно нарандомить).

Решение

Кластеризация методом k-средних состоит из нескольких этапов:

а) Инициализировать центры кластеров;

б) для каждой точки найти минимальное расстояние до центроида;

в) установить принадлежность точки соответствующему кластеру;

г) Вычислить новые центроиды, как среднее всех точек кластера.

Шаги 3 и 4 повторяются до тех пор, пока не будет достигнуто условие остановки, в моём решении это достижение предела количества итераций.

В листинге 1 приведена функция поиска ближайшего центроида для каждой точки и присваивание точки кластеру.

Листинг 1 — Функция поиска ближайшего центроида

```
1     void closest_center(point* points, int npoints, point* centers, int k,
          int* clusters)
2     {
3         for (int i = 0; i < npoints; i++)
4         {
5             float min = 150 * 150 * 150;
6             int cluster = 0;
7             for (int j = 0; j < k; j++)
8             {
9                 float d = distance(points[i], centers[j]);
10                if (d < min)
11                {
12                    cluster = j;
13                    min = d;
14                }
```

```
15              }
16              clusters[i] = cluster;
17          }
18      }
```

В листинге 2 представлена функция кластеризации точек по кластерам k, для возможности дальнейшего сравнения алгоритмов центры передаются инициализированными.

Листинг 2 — Не оптимизированная функция кластеризации

```
1  int* clustering(point* points, int npoints, int k, point* centers)
2  {
3      int* clusters = new int[npoints];
4
5      for (int iter = 0; iter < ITERATIONS; iter++)
6      {
7          closest_center(points, npoints, centers, k, clusters);
8
9          for (int i = 0; i < k; i++)
10         {
11             centers[i].x = 0;
12             centers[i].y = 0;
13             centers[i].z = 0;
14         }
15
16         for (int i = 0; i < npoints; i++)
17         {
18             centers[clusters[i]].x = centers[clusters[i]].x + points[i].x;
19             centers[clusters[i]].y = centers[clusters[i]].y + points[i].y;
20             centers[clusters[i]].z = centers[clusters[i]].z + points[i].z;
21         }
22
23         for (int i = 0; i < k; i++)
24         {
25             int cl = 0;
26             for (int j = 0; j < npoints; j++)
27             if (clusters[j] == i)
28                 cl = cl + 1;
29
30             if (cl)
31             {
32                 centers[i].x = centers[i].x / float(cl);
33                 centers[i].y = centers[i].y / float(cl);
34                 centers[i].z = centers[i].z / float(cl);
35             }
```

```
36          }
37      }
38
39      return clusters;
40  }
```

В листинге 3 представлена оптимизированная функция кластеризации.

Листинг 3 — Оптимирированная функция кластеризации

```
1  int *clustering_opt(point *points, int npoints, int k, point *centers)
2  {
3      int *clusters = new int[npoints];
4
5      for (int iter = 0; iter < ITERATIONS; iter++)
6      {
7          closest_center(points, npoints, centers, k, clusters);
8
9          for (int i = 0; i < k; i++)
10         {
11             centers[i].x = 0;
12             centers[i].y = 0;
13             centers[i].z = 0;
14
15             int cl = 0;
16             for (int j = 0; j < npoints; j++)
17             {
18                 if (clusters[j] == i)
19                 {
20                     cl++;
21                     centers[i].x += points[j].x;
22                     centers[i].y += points[j].y;
23                     centers[i].z += points[j].z;
24                 }
25             }
26             if (cl)
27             {
28                 centers[i].x /= float(cl);
29                 centers[i].y /= float(cl);
30                 centers[i].z /= float(cl);
31             }
32         }
33     }
34
35     return clusters;
36 }
```

С целью оптимизации были приняты такие меры:

1. где это возможно операторы формата $a = a + b$ заменены на $a+ = b$ или аналогичные;

2. объединены циклы, связанные с вычислением новых центроидов, а именно:

1) цикл зануления координат центроида;

2) цикл суммирования значений центроида;

3) цикл подсчёта размерности кластера;

3. слияние циклов позволило избавится от сложной индексации в суммировании координат.

Рассчитаем трудоёмкости

Расчёт будем производить только для изменённой части.

Не оптимизированная функция:

$$
\begin{aligned}
& f = 2 + k * (2 + 6) + 2 + n * (2 + 12 + 3 + 3 + 3) + 2+ \\
& k * (2 + 1 + 2 + n * (2 + 2 + \begin{cases} 2 \\ 0 \end{cases}) + 1 + \begin{cases} 6 + 3 + 3 \\ 0 \end{cases}) = \\
& = 6 + 8k + 23n + k * (6 + \begin{cases} 9 \\ 0 \end{cases} + n * (4 + \begin{cases} 2 \\ 0 \end{cases})) \\
& = kn(4 + \begin{cases} 2 \\ 0 \end{cases}) + 23n + k(14 + \begin{cases} 9 \\ 0 \end{cases}) + 6
\end{aligned}
\tag{0.1}
$$

Оптимизированная:

$$f = 2 + k * (2 + 6 + 1 + 2 + n * (2 + 2 + \begin{cases} 1 + 6 + 3 \\ 0 \end{cases}) + 1 + \begin{cases} 3 + 3 \\ 0 \end{cases}) =$$

$$= 2 + k * (12 + n * (4 + \begin{cases} 9 \\ 0 \end{cases}) + \begin{cases} 6 \\ 0 \end{cases}) =$$

$$= kn(4 + \begin{cases} 9 \\ 0 \end{cases}) + k * (12 + \begin{cases} 6 \\ 0 \end{cases}) + 2 \tag{0.2}$$

Тестирование

Тестирование проводилось на случайно генерируемых массивах точек, координаты которых лежат в пределах от -100 до 100 по каждой размерности. Результатом работы программы является: массив номеров кластеров – номер кластера указывает к какому кластеру принадлежит i-ая точка. И множество центроидов. Тесты проводились для 20 точек и 3 кластеров, 30 точек и 7 кластеров, 35 точек и 10 кластеров.

На рисунках 0.1-0.3 представлены примеры работы программы.

```
Cluster numbers: 1 0 0 1 0 2 2 1 2 2 1 0 1 0 0 2 0 2 0 1
Centroid#0:   6.750 -65.000  22.125
Centroid#1: -80.500 -12.500   8.000
Centroid#2:  38.000  52.167  14.667
Cluster numbers: 1 0 0 1 0 2 2 1 2 2 1 0 1 0 0 2 0 2 0 1
Centroid#0:   6.750 -65.000  22.125
Centroid#1: -80.500 -12.500   8.000
Centroid#2:  38.000  52.167  14.667
```

Рисунок 0.1 — Пример работы

```
Cluster numbers unoptimized: 6 0 5 6 3 0 0 3 6 6 2 4 2 3 3 2 1 6 0 1 3 2 0 1 6 5 6 5 5 6
Centroid#0:  65.800  53.200 -29.200
Centroid#1: -77.000  66.000  22.000
Centroid#2: -18.000  59.000 -50.750
Centroid#3:  20.000  67.600  34.200
Centroid#4: -93.000 -42.000   9.000
Centroid#5:  10.750  -3.250 -49.000
Centroid#6:  18.750 -54.250  41.625
Cluster numbers optimized: 6 0 5 6 3 0 0 3 6 6 2 4 2 3 3 2 1 6 0 1 3 2 0 1 6 5 6 5 5 6
Centroid#0:  65.800  53.200 -29.200
Centroid#1: -77.000  66.000  22.000
Centroid#2: -18.000  59.000 -50.750
Centroid#3:  20.000  67.600  34.200
Centroid#4: -93.000 -42.000   9.000
Centroid#5:  10.750  -3.250 -49.000
Centroid#6:  18.750 -54.250  41.625
```

Рисунок 0.2 — Пример работы

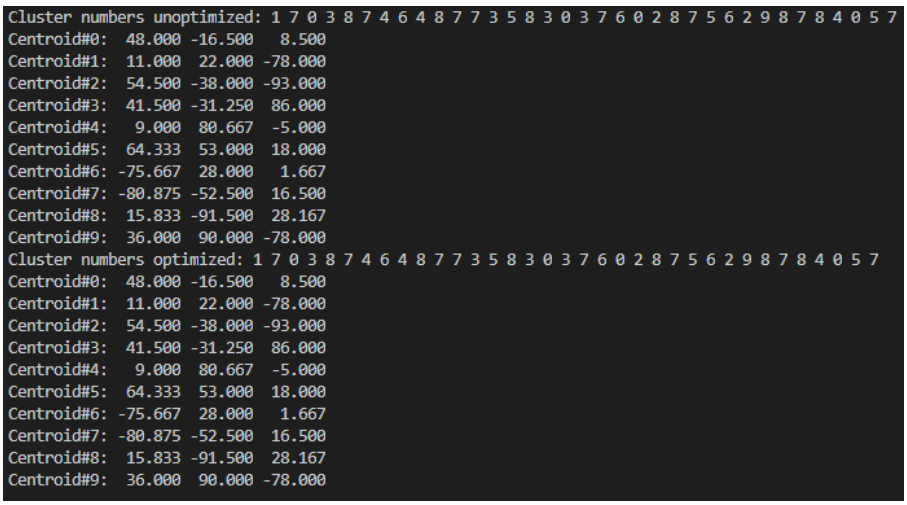

Рисунок 0.3 — Пример работы

Все тесты пройдены успешно.

Вывод

Был оптимизирован код алгоритма кластеризации методом k-средних.