Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математических методов прогнозирования

Колмагоров Евгений Игоревич

Отчёт о проделанной лабораторной работе №1.

Москва, 2020

Оглавление

1 Ведение

В данной работе рассматривается решение задачи об оптимальном выборе столицы. Проводится оптимизация взвешенного манхэттенского расстояния на примере городов Российской Федерации. Все вычислительные эксперименты были проведены на языке программирования C++11, программная реализация доступна по следующему адресу: `www.github.com/kolmX/weighted_median.cpp`

2 Постановка задачи

Имеется текстовый файл, в котором содержится таблица городов Российской Федерации, где для каждого города представлена следующая информация:

1. Название города
2. Долгота
3. Широта
4. Численность населения (тыс. человек)

Поставлен вопрос о том, какой выбрать город из всего списка, чтобы следующий функционал принимал наименьшее значение:

$$F(p) = \sum_{i=1}^{n} W_i Dist(p, p_i) \to min \tag{1}$$

Где $p_i, i = 1, ..., n$ - города, а $\{W_1, ..., W_n\}$ - численность населения, в качестве метрики выбрано манхэттенское расстояние между городами, с неравным вкладом широты и долготы:

$$Dist(p, q) = 2 * |p.x - q.x| + |p.y - q.y| \tag{2}$$

Где X-овая координата - долгота, а Y-овая - широта. Географические координаты представлены в двух форматах:

1. Градусы и минуты. Пример: $38°58'$;
2. Градусы в виде десятичных дробей. Пример: 42.71.

3 Описание алгоритма

Первым делом для решения поставленной задачи необходимо привести все данные к одному виду, для этого было выбрано второе представление географических

координат во второй форме. Соответственно минуты из первого формата преобразуются к долям градуса по формуле:

$$Degrees = 0, \frac{Minutes \times 100}{60} \tag{3}$$

3.1 Алгоритм с квадратичной временной сложностью

Далее для вычисления оптимальной столицы можно попробовать воспользоваться простым переборным алгоритмом по всем городам с временной сложностью $O(n^2)$.

Algorithm 1: Псевдокод алгоритма со сложностью $O(n^2)$

```
Input  : City array of tuples: (x, y, population);
Output: City name with min F value;
Min_W ← +inf;
CityName ← NULL;
foreach fcity in Cities do
    W ← 0;
    foreach scity in Cities do
        W ← W + scity.population*( 2*|fcity.x - scity.x| + |fcity.y - scity.y|)
    end
    if W < Min_W then
        Min_W ← W;
        CityName ← fcity;
    end
end
```

3.2 Алгоритм с временем работы nlog(n)

Но время работы алгоритма можно улучшить до оценки $O(n\ log_2(n))$. Достаточно заметить следующее - если значения долготы и широты будут упорядоченны по возрастанию т.е. $x_1 \leq x_2 \leq ... \leq x_{n-1} \leq x_n$, тогда следующее выражение:

$$f(x_j) = \sum_{i=1}^{n} W_i * |x_i - x_j| \tag{4}$$

Может быть переписано в следующем виде:

$$f(x_j) = \sum_{i=1}^{j} -W_i * (x_i - x_j) + \sum_{i=j+1}^{n} W_i * (x_i - x_j) =$$

$$= \sum_{i=j+1}^{n} W_i * x_i - \sum_{i=1}^{j} W_i * x_i + x_j * (\sum_{i=1}^{j} W_i - \sum_{i=j+1}^{n} W_i)$$

Таким образом, если иметь предпосчитанные значения первого слагаемого и второго, а также часть третьего члена, которая заключена в скобках, то вычисление $f(x_j)$ может быть выполнено за константное время, а расчёт всех значений $x_j,\ j = 1, ..., n$ - за линейное. Для вычисления первого и второго слагаемого достаточно выделить дополнительный массив A размера n и один раз пройтись по всему массиву городов, вычисляя кумулятивную сумму и записывая её в выделенный массив.

$$A[i+1] = A[i] + x_i * W_i$$

Тогда первое слагаемое может быть получено через обращение к j-му элементу массива A, а второе - через A[j] - A[n]. Соответственно для расчёта выражения стоящего в скобках у третьего слагаемого, воспользуемся той же самой идей, только теперь - простая кумулятивная сумма по весам (без умножения на x_i).

Таким образом, проделав данные вычисления отдельно для долготы и широты городов и, выполнив суммирование весов широты и долготы получим вес каждого города, для ответа на поставленную задачу достаточно взять минимум среди всех весов. Самая затратная процедура в рассматриваемом подходе - сортировка значений широты и долготы. При решении данной задачи была использована сортировка слиянием с временной сложностью $O(n\ log_2 n)$.

Algorithm 2: Псевдокод алгоритма со сложностью $O(nlog_2n)$

```
Input  : City array of tuples: (x, y, population);
Output: City name with min F value;
CityName ← NULL;
SortedX ← Sort(Cities.X) ;                          // Сортировка долготы
SortedY ← Sort(Cities.Y) ;                          // Сортировка широты
foreach Arr in {SortedX, SortedY} do
    CUM_W ← [0];
    CUM_XW ← [0];
    for i ← 1 to Arr.size do
        CUM_W[ i ] += CUM_W[ i - 1 ] + Arr[ i ].W;
        CUM_XW[ i ] += CUM_XW[ i - 1 ] + Arr[ i ].W * Arr[ i ].X;
    end
    Arr←computeWeights(Arr, CUM_W, CUM_XW);          // Вычисление
     значений f(x_j),  j = 1, .., n
end
United←unite(SortedX, SortedY); ;          // Объединение весов широты и
 долготы
CityName, Min_W←findMin(United) ; // Поиск минимального элемента в
 массиве
```

4 Результаты

Ниже приведены 10 городов, имеющие наименьший вес F(p), с различной фильтрацией по численности населения.

	Город	Широта	Долгота	Население	Вес
0	Муром	55.57	42.00	107	2596535.25
1	Арзамас	55.38	43.87	104	2596543.75
2	Дзержинск	56.27	43.40	229	2611063.25
3	Нижний Новгород	56.30	44.00	1252	2613153.75
4	Ковров	56.37	41.35	136	2619562.00
5	Владимир	56.15	40.42	357	2620151.00
6	Саранск	54.18	45.17	321	2628994.50
7	Рязань	54.60	39.70	539	2629968.50
8	Орехово-Зуево	55.82	38.98	118	2644575.25
9	Иваново	57.02	40.98	405	2649288.25

Рис. 1: Оптимальное расположение столицы без применения фильтрации по численности населения.

	Город	Широта	Долгота	Население	Вес
0	Владимир	56.15	40.42	357	2047068.25
1	Рязань	54.60	39.70	539	2053027.12
2	Нижний Новгород	56.30	44.00	1252	2058733.75
3	Иваново	57.02	40.98	405	2074542.75
4	Саранск	54.18	45.17	321	2080256.62
5	Москва	55.75	37.62	12678	2094893.00
6	Мытищи	55.92	37.73	236	2095213.62
7	Тамбов	52.72	41.42	292	2097894.75
8	Пенза	53.20	45.00	520	2102428.00
9	Ярославль	57.62	39.85	608	2103170.00

Рис. 2: Оптимальное расположение столицы с численность населения большем, чем 235 000.

	Город	Широта	Долгота	Население	Вес
0	Москва	55.75	37.62	12678	788310.00
1	Нижний Новгород	56.30	44.00	1252	867004.44
2	Воронеж	51.72	39.27	1058	892945.94
3	Казань	55.78	49.17	1257	975139.50
4	Ростов-на-Дону	47.23	39.70	1138	1031724.25
5	Волгоград	48.72	44.48	1009	1053040.00
6	Самара	53.23	50.17	1157	1057701.25
7	Санкт-Петербург	59.93	30.32	5398	1208388.25
8	Уфа	54.82	56.07	1129	1209285.25
9	Пермь	58.00	56.23	1055	1241277.12

Рис. 3: Оптимальное расположение столицы среди городов-миллионников.

5 Программная реализация

Весь код написан на языке программирования C++ 11-ой версии, в качестве компилятора был взят GNU G++ компилятор. Был реализован второй метод с вычислительной сложностью $O(nlog(n))$, объём написанного кода 240 строк. Чтение происходит из стандартного потока ввода, запись производится в стандартный поток вывода. Входная кодировка используется utf-8 (Исходный файл был переведён из кодировки cp1251 в utf-8). В качестве аргументов программа может принимать одно значение - порог по численности населения, по которому производить отсечение. Условно весь код можно поделить на две части - обработка текстового файла, содержащего исходные данные, и сам поиск оптимальной столицы.

Пример кода компиляции программы:

```
g++ weighted_median.cpp -Wall -o weighted_median
```

Пример кода запуска программы с отсечением по порогу 235:

```
./weighted_median 235  < cities-utf8.txt  > result235.txt
```

6 Выводы

В ходе лабораторной работы было рассмотрено два подхода с различной временной сложностью, программно был реализован метод, имеющий наименьшую вычислительную сложность. Стоит отметит, что алгоритм может быть немного усовершенствован за счёт использования другого подхода к сортировкам - как правило большинство городов удалены друг от друга на расстояние бОльшем чем 1 градус, то можно было бы воспользоваться CountSort с количеством бинов 360 для долготы и 180 бинов для широты с последующей сортировкой (к примеру MergeSort или QuickSort) внутри бинов .

7 Используемая литература

[1] Кормен Т.Х. и др. Алгоритмы: построение и анализ, 3-е изд., Москва, «И. Д. Вильямс», 2016. – 1328 с.