Функции качества в этом случае определяются как

$$L_1(f) = C_1(f) + jQ_1(f),$$
$$L_2(f) = C_2(f) + jQ_2(f).$$

где $C_1(f), C_2(f)$ синфазные и $Q_1(f), Q_2(f)$ квадратурные составляющие функций качества.

Для их нахождения можно воспользоваться следующими соотношениями

$$C_1(f) = \int_0^T L_1(t) \cdot \cos(2\pi ft)dt, \quad Q_1(f) = \int_0^T L_1(t) \cdot \sin(2\pi ft)dt;$$
$$C_2(f) = \int_0^T L_2(t) \cdot \cos(2\pi ft)dt, \quad Q_2(f) = \int_0^T L_2(t) \cdot \sin(2\pi ft)dt.$$

Также взаимная спектральная плотность есть обычно функция комплексная

$$\hat{G}_{12}(f) = \hat{C}_{12}(f) + j\hat{Q}_{12}(f),$$

где

$$\hat{C}_{12}(f) = \frac{2}{T}\left[C_1(f)C_2(f) + Q_1(f)Q_2(f)\right],$$
$$\hat{Q}_{12}(f) = \frac{2}{T}\left[C_1(f)Q_2(f) - Q_1(f)C_2(f)\right].$$

В итоге фазовый спектр запишется в виде

$$\hat{\varphi}_{12} = \operatorname{arc}tg\frac{\hat{Q}_{12}(f)}{\hat{C}_{12}(f)}.$$

Таким образом, можно оценить взаимные фазовые спектры, не прибегая к восстановлению сигналов, что дает преимущество при оценивании спектров в зонах повышенной интерференции.

Предложенный способ оценивания взаимных фазовых спектров отличается незначительно от классического метода вычисления взаимных фазовых спектров. Что позволяет в дальнейшем использовать его в алгоритме прогноза коллекторских свойств пород при сильной интерференции анализируемых сигналов.

ЛИТЕРАТУРА

1. Разработка методики прогноза нефтегазоносности продуктивных комплексов на базе комплексного использования энергетических и фазочастотных характеристик отраженных волн в условиях Томской области. – Отчет. – Томск, 2000, – 159с.

СРАВНЕНИЕ СЛОЖНОСТЬ АЛГОРИТМОВ ВСТАВКОЙ И БЫСТРОЙ СОРТИРОВКИ

Чан Тхюи Зунг

(г. Томск, Томский политехнический университет)

COMPARISON OF COMPLEXITY OF SORTING ALGORITHM QUICKSORT AND INSERTION SORT

Tran Thuy Dung

(s.Tomsk, Tomsk Polytechnik University)

A sorting algorithm is an algorithm that puts elements of a list in a certain order. The most used orders are numerical order and lexicographical order. Efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) which require input data to be in sorted lists; it is also often useful for canonicalizing data and for producing human readable output. Sorting algorithms are prevalent in introductory computer science classes, where the abundance of algorithms for the problem provides a gentle introduction to a variety of core algorithm concepts, such as big O notation, divide and conquer algorithms, data structures such as heaps and binary trees, randomized algorithms, best, worst and average case analysis, time-space tradeoffs, and upper and lower bounds.

Введение Выбор структуры обрабатываемых данных оказывает значительное влияние на алгоритмы решения задачи. Задача сортировки данных может быть решена при помощи различных алгоритмов. Сортировка - это достаточно часто выполняемая фундаментальная операция. Есть много способов для сортировки данных, но здесь нас интересует только два алгоритма: сортировка по методу вставки (сортировка вставками) и сортировка на основе разбиения (алгоритм быстрой сортировки). В этой статье проводится анализ двух алгоритмов сортировки и их сравнение, а также оценка их сложности.

Вычислительная сложность алгоритма это попросту оценка количества операций, требуемых для достижения результата, в зависимости от количества обрабатываемых элементов. Чаще всего встречаются такие виды вычислительной сложности: константная, логарифмическая, линейная, квадратичная, экспоненциальная и факториальная. Каждая следующая - «хуже» предыдущей.

Сортировка вставками Данная сортировка используется игроками в карты. Элементы массива разделяются на последовательность-приемник ($a_0 \ldots a_{i-1}$) и последовательность источник ($a_i \ldots a_n$). Суть алгоритма состоит в следующем. На каждом шаге, начиная с i=1, в последовательности источнике берется один элемент и ставиться в правильную позицию в последовательности приемнике. Количество элементов в последовательности приемнике увеличивается, а в последовательности источнике уменьшается. Сортировка вставками требует фиксированного числа проходов. На *(n-1)* проходах вставляются элементы от *a[1]* до *a[n-1]*. На i-ом проходе вставки производятся в подсписок *a[0]– a[i]* и требуют в среднем (i/2) сравнений. Общее число сравнений равно :

$$1/2 + 2/2 + 3/2 + \ldots + (n-2)/2 + (n-1)/2 = n(n-1)/4$$

Сложность алгоритма измеряется числом сравнений и равна $O(n^2)$. Наилучший случай - когда исходный список уже отсортирован. Тогда на i-ом проходе вставка производится в точке *a[i]*, а общее число сравнений равно *(n-1)*, т.е. сложность составляет *O(n)*. Наихудший случай возникает, когда список отсортирован по убыванию. Тогда каждая вставка происходит в точке *a[0]* и требует i сравнений. Общее число сравнений равно *(n(n-1)/2)*, т.е. сложность составляет $O(n^2)$.

Быстрая сортировка Общая идея алгоритма состоит в разделении входного набора данных относительно опорного элемента, выбираемого случайным образом. Все элементы, которые меньше опорного перемещаются в нижнюю часть массива, большие остаются на своих местах. В результате выполнения этих действий весь массив должен разделиться на три группы, следующие друг за другом: меньше опорного, равная опорному и больше опорного. Для «больших» и «меньших» отрезков рекурсивно выполняется та же последовательность действий.

При первом сканировании производится *(n-1)* сравнений. В результате создаются два подсписка размером *(n/2)*. На следующей фазе обработка каждого подсписка требует примерно *(n/2)* сравнений. Общее число сравнений на этой фазе равно *(2*(n/2) = n)*. На

следующей фазе обрабатываются четыре подсписка, что требует *(4*(n/4))* сравнений, и т.д. В конце концов процесс разбиения прекращается после *k* проходов, когда получившиеся подсписки содержат по одному элементу. Общее число сравнений приблизительно равно

$$n + 2(n/2) + 4(n/4) + ... + n(n/n) = n + n + ... + n = n * k = n * \log_2 n$$

Для списка общего вида вычислительная сложность «быстрой» сортировки равна $(O(n*\log_2 n))$. Идеальный случай, фактически возникает тогда, когда массив уже отсортирован по возрастанию. Тогда центральный элемент попадает точно в середину каждого подсписка.

Если массив отсортирован по убыванию, то на первом проходе центральный элемент обнаруживается на середине списка и меняется местами с каждым элементом как в первом, так и во втором подсписке. Результирующий список почти отсортирован, алгоритм имеет сложность порядка $O(n \log_2 n)$. Наихудшим сценарием для «быстрой» сортировки будет тот, при котором центральный элемент все время попадает в одноэлементный подсписок, а все прочие элементы остаются во втором подсписке. Это происходит тогда, когда центральным всегда является наименьший элемент. Рассмотрим последовательность чисел,например : 1, 2,9,3,8,6.... На первом проходе производится *n* сравнений, а больший подсписок содержит *(n-1)* элементов. На следующем проходе этот подсписок требует *(n-1)* сравнений и дает подсписок из *(n-2)* элементов и т.д. Общее число сравнений равно:

$$n + n-1 + n-2 + ... + 2 = (n(n+1)/2) - 1$$

Сложность худшего случая равна $O(n_2)$, т.е. не лучше, чем для сортировок вставками и выбором. Однако этот случай является патологическим и маловероятен на практике.

Мы сравнили алгоритмы сортировки, испытав их на массивах, содержащих некоторые целые числа, соответственно. Результаты испытаний показаны на рисунке 1.

Элементы	Сортировка вставками(Количество перестановок)	Сортировка быстрая (Количество перестановок)
4891	6082823	15424
7793	15090009	26103
6369	10134015	20983
9983	24836414	34118
9706	23790774	33709
5540	7590232	17776
2840	2024251	8554
3526	3099037	10875
6617	10824211	21844

Рис. 1. Сравнение сортировок.

В общем случае Быстрая сортировка является самым быстрым алгоритмом. Благодаря своей эффективности, равной $O(n \log_2 n)$, он явно превосходит любой алгоритм порядка $O(n^2)$.

ЛИТЕРАТУРА

1. http://www.algolist.net/Algorithms/Sorting/Quicksort.
2. http://en.wikipedia.org/wiki/Sorting_algorithm.
3. http://www.cprogramming.com/tutorial/computersciencetheory/sortcomp.html.