Лекция 4. Простые алгоритмы внутренней сортировки

**Алгоритм сортировки** - это алгоритм для упорядочения элементов в некоторой структуре данных по росту или убыванию. В случае наличия элементов с одинаковыми значениями, в благоустроенной последовательности они располагаются рядом друг с другом в любом порядке. Однако иногда бывает полезно придерживаться первоначального порядка элементов с одинаковыми значениями.

В зависимости от того, над какой структурой данных осуществляется сортировка, бывает сортировки массива, связанного списка, дерева, графа, таблицы.

В случае, когда элемент структуры данных имеет несколько полей, вводится понятие ключ сортировки.

**Ключ сортировки** - это поле структуры данных, по значению которого определяется порядок элементов (то есть по которым происходит сортировка). На практике в качестве ключа часто выступает число, а в других полях хранятся какие-либо данные, никак не влияют на работу алгоритма.

Простой алгоритм сортировки можно разбить на 3 части:

- Сравнения, определяет упорядоченность пары элементов;

- Перестановка, что меняет местами эту пару элементов;

- Повтор первых двух действий до тех пор, пока все элементы структуры данных не будут упорядочены.

Алгоритмы сортировки имеют большое практическое значение. Их можно встретить там, где происходит обработка и хранение больших объемов информации. Они используются также в криптографии, кодировке, решении математических задач и др.

Ни одна другая проблема не породила такого количества различных решений, как задача сортировки. Универсального, лучшего алгоритма сортировки не существует. Однако, имея приблизительные характеристики входных данных, можно подобрать метод, работающий оптимальным образом. Для этого необходимо знать параметры, по которым будет осуществляться оценка алгоритмов.

**Параметры алгоритмов сортировки:**

- **Время сортировки** - основной параметр, характеризующий быстродействие алгоритма.

- **Память** - один из параметров, характеризующееся тем, что ряд алгоритмов сортировки требуют выделения дополнительной памяти под временное хранение данных. При оценке используемой памяти не будет учитываться место, которое занимает исходный массив данных и независимые от входной последовательности расходы, например, на хранение кода программы.

- **Устойчивость** - это параметр, отвечающий за то, что сортировка не меняет взаимного расположения равных элементов. Например, если алфавитный список группы сортируется по оценкам, то стабильный метод создает список в котором фамилии студентов с одинаковыми оценками будут упорядочены по алфавиту, а нестабильный метод создаст список в котором, возможно, исходный порядок будет нарушен.

- **Естественность** **поведения** - параметр, указывающий на эффективность метода при обработке уже отсортированных или частично отсортированных данных. Алгоритм ведет себя естественно, если учитывает эту характеристику входной последовательности и работает лучше.

- **Использование** **операции** **сравнения**. Алгоритмы, используемые для сортировки сравнения элементов между собой, называются основанными на сравнениях. Минимальная трудоемкость худшего случая для этих алгоритмов составляет O(n log n), но они отличаются гибкостью применения. Для специальных случаев (типов данных) существуют более эффективные алгоритмы.

**Классификация алгоритмов сортировки**

Все алгоритмы сортировки можно классифицировать по различным признакам, например, по устойчивости, по особенностям функционирования, за использованием операций сравнения, по необходимости в дополнительной памяти, по структуре данных, по ширине области применения, по необходимости в знаниях о структуре данных, выходящих за рамки операций сравнения ключей и тому подобное.

Рассмотрим более подробно классификацию алгоритмов сортировки за использованием памяти. В этом случае основные типы сортировки делятся на:

**- Внутреннее сортировки -** это алгоритм сортировки, в процессе упорядочения данных использует только оперативную память (ОЗУ) компьютера. То есть оперативной памяти достаточно для размещения в ней сортируемого массива данных с произвольным доступом к любой ячейки и собственно для выполнения алгоритма. Внутреннее сортировка применяется во всех случаях, за исключением однопроходной считывания данных и однопроходной записи отсортированных данных. В зависимости от конкретного алгоритма и его реализации данные могут сортироваться в той же области памяти, или использовать дополнительную оперативную память.

**- Внешнее сортировки -** это алгоритм сортировки, при проведении упорядочивание данных использует внешнюю память, как правило, жесткие диски. Внешнее сортировки разработанное для обработки больших структур данных, которые не помещаются в оперативную память. Обращение к различным носителям накладывает некоторые дополнительные ограничения на данный алгоритм: доступ к носителю осуществляется последовательным образом, то есть в каждый момент времени можно считать или записать только элемент, следующий за текущим; объем данных не позволяет им разместиться в ОЗУ.

Внутреннее сортировка является базовым для любого алгоритма внешней сортировки - отдельные части массива данных сортируются в оперативной памяти и с помощью специального алгоритма соединяются в один массив, упорядоченный по ключу.

Следует отметить, что внутреннее сортировка значительно эффективнее внешнего, так как на обращение к оперативной памяти затрачивается гораздо меньше времени, чем к носителям.

Одна из самых распространенных классификаций методов сортировки

приведена на рисунке 1.1.

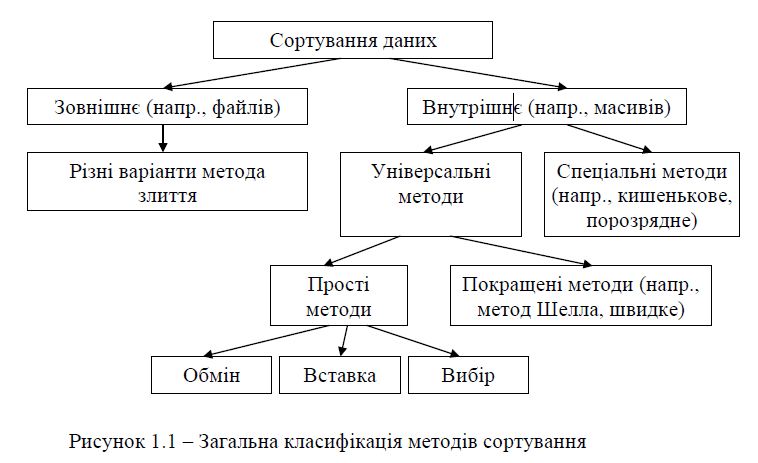


Рисунок 1.1 - Общая классификация методов сортировки

**Классификация по времени выполнения:**

*за время O(n2)*

• сортировка выбором

• сортировка вставкой

• сортировка обменом

*за время O(n log n)*

• пирамидальное сортировка

• быстрая сортировка

• сортировка слиянием

*За время O(n) с использованием дополнительной информации об элементах*

• сортировка подсчетом

• сортировка по разрядам

• блочная сортировка

*за время O(n log2 n)*

• сортировка слиянием модифицированное

• сортировка Шелла

**Алгоритмы стабильного упорядочивания:**

*• сортировка выбором*

*• сортировка вставкой*

*• сортировка обменом*

*• сортировка слиянием*

*• сортировка подсчетом*

*• сортировка по разрядам*

• блочная сортировка

• сортировка слиянием модифицированное

**Простые алгоритмы внутренней сортировки**

Чтобы сконцентрироваться на алгоритмических вопросах, мы будем работать с алгоритмами, которые сортируют массивы целых чисел. Эти алгоритмы легко адаптировать для сортировки записей.

В основном программы сортировки работают с записями двумя способами: либо они сравнивают и сортируют только ключи, либо передвигают записи целиком. Большинство алгоритмов, которые мы изучим можно применять, посредством их переформулированные в терминах этих двух операций, для произвольных записей.

Если записи достаточно большие, то обычно стараются избежать их передвижения с помощью "косвенного сортировки": при этом сами записи не сортируются, а вместо этого сортируется массив указателей (индексов), так, что первый указатель указывает на самый маленький элемент и так далее. Ключи могут храниться либо с записями (если они большие), либо с указателями (если они маленькие).

**Сортировка Выбором**

Один из самых простых методов сортировки работает следующим образом:

1) находим наименьший элемент в массиве;

2) меняем его местами с элементом, который находится на первом месте;

3) повторяем процесс со второй позиции в файле и найденный наименьший элемент обмениваем со вторым элементом и так далее пока весь массив НЕ будет отсортирован.

По мере продвижения указателя слева направо через массив, элементы слева от указателя находятся уже в своей конечной позиции (и их уже не будут больше передвигать), поэтому массив становится полностью отсортированным к тому моменту, когда указатель достигает правого края.

Этот метод называется сортировкой выбором поскольку он работает циклически, выбирая наименьший из оставшихся элементов.

Псевдокод процедуры сортировки выбором:

Процедура Сортировка выбором (a: массив, N: длина массива)

переменные i, j, min, t: цели

начало

для и от 1 до N-1 // N-размер массива

начало

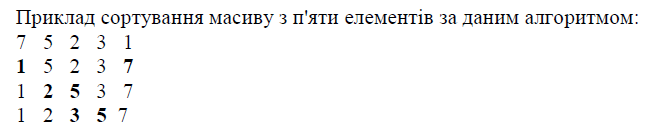
min = i

для j от i + 1 до N // цикл нахождения минимального элемента

если a [j] <a [min] тогда

min = j t = a [min] // замена элементов a [min] = a [i] a [i] = t конец

конец



Этот метод - один из самых простых, и он работает очень хорошо для небольших структур данных. Его "внутренний цикл" состоит из сравнения a [i] <a [min] (плюс код необходимый для увеличения j и проверки того, что он не превысил N), что вряд ли еще упростить.

Более того, несмотря на то, что этот метод очевидно является методом "грубой силы", он имеет очень важное применение: поскольку каждый элемент передвигается не более чем раз, то он очень хорош для больших записей с маленькими ключами.

**Характеристика алгоритма:**

**Структура данных:** Массив

**Быстродействие:** О(n²)

**Пространство:** О(n), O(1)

**Оптимальность:** не практичный

**Сортировка вставками**

Сортировка вставкой - это метод который почти настолько же прост, что и сортировка выбором, но гораздо более гибкий. Большинство людей при сортировке колоды игральных карт, используют метод, похожий на алгоритм сортировки включением. Суть алгоритма: берем один элемент и вставляем его в нужное место среди тех, которые мы уже обработали (тем самым оставляя их отсортированными). алгоритм:

1) слева направо проходим массив, сравнивая соседние элементы, пока не найдем элемент, который расположен не в порядке сортировки;

2) обмениваем этот элемент с элементами, расположенными слева от него, пока он не займет нужную позицию;

3) повторяем первые два действия пока массива не будет отсортировано.

**Псевдокод процедуры сортировки вставкой:**

Процедура Сортировка вставкой (a: массив, N: длина массива)

Переменные i, j, t: цели

начало

для i от 2 до N

начало

t = a [i]

j = i-1

пока j> 0 и t <a [j]

начало

a [j + 1] = a [j]

j = j-1

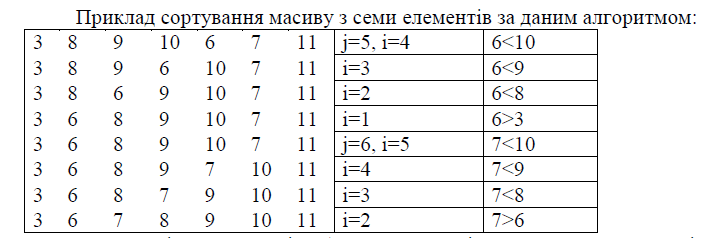
конец

a [j + 1] = t // для ускорения операция передвинута в верхней цикл

конец

конец

Пример сортировки массива из семи элементов по данному алгоритму:



Также, как и в сортировке выбором, в процессе сортировки элементы слева от указателя i находятся уже в отсортированном порядке, но они не обязательно находятся в своей последней позиции, поскольку их еще могут передвинуть справа чтобы вставить более маленькие элементы, встреченные позже. Однако массив становится полностью отсортированным, когда указатель достигает правого края. Данный алгоритм также простой в реализации и эффективен для небольших массивов. Есть стабильным алгоритмом.

Эффективен при сортировке массивов, данные в которых уже неплохо отсортированы: производительность равна O (n + d), где d - количество инверсий.

**Характеристика алгоритма:**

**Структура данных:** Массив

**Быстродействие:** О (n2), для лучшего случае O (n + d)

**Пространство:** О (n), O (1)

**Оптимальный:** Преимущественно нет

**Пузырьковая сортировка (сортировка простыми обменами)**

Алгоритм работает следующим образом - в массиве сравниваются два соседних элемента. Если один из элементов, не соответствует критерию сортировки (есть больше, или же, наоборот, меньше своего соседа), то эти два элемента меняются местами. Проход по списку продолжается до тех пор, пока данные не будут отсортированными. Алгоритм получил свое название от того, что процесс сортировки по ним напоминает поведение пузырьков воздуха в резервуаре с водой. Поскольку для работы с элементами массива использовать лишь сравнения, это сортировка на основе сравнений.

Процедура Пузырьковая сортировка (a: массив, N: длина массива)

Переменные i, j, t: цели

начало

для i от 1 до N для j от 1 до N-1 если a [j]> а [j + 1] тогда

начало

t = а [j]

а [j] = a [j + 1]

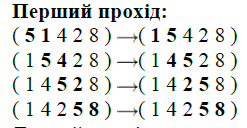
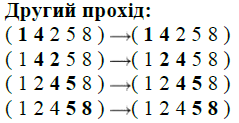
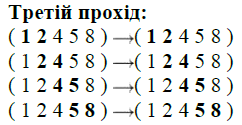
а [j + 1] = t

конец

конец

Пример реализации шаг за шагом

Возьмем массив чисел "1 5 4 2 8", и с помощью данного алгоритма, отсортируем его от маленького до большого значения элемента. На каждом шагу, элементы, выделенные жирным шрифтом, будут сравниваться.



Теперь наш массив полностью отсортирован, однако, алгоритм этого еще не знает. Ему нужен еще один "пустой" проход, во время которого от не поменяют местами ни одного элемента (если реализовать соответствующий флаг, иначе проходов будет столько сколько элементов в массиве - по классической реализацией).

Наконец, массив отсортировано, и алгоритм может прекратить свою работу.