Теория алгоритмов Лк5 - Алгоритмы сортировки, слияния, поиска

Массив *- набор компонентов (обычно одного типа), расположенных в памяти непосредственно друг за другом, доступ к которым осуществляется по индексу (индексам). Массив - структура с произвольным доступом, обычно имеет постоянную длину.*

Связный список *- базовая динамическая структура данных, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка.*

Стек *- структура данных, представляющая собой список элементов, организованных по принципу LIFO (last in — first out, «последним пришёл — первым вышел»).*

Очередь *- структура данных с дисциплиной доступа к элементам «первый пришёл — первый вышел» (FIFO, First In — First Out). Добавление элемента возможно лишь в конец очереди, выборка — только из начала очереди (что принято называть словом dequeue — убрать из очереди), при этом выбранный элемент из очереди удаляется.*

Алгоритмы сортировки

Целочисленный массив (список, файл) с расположенными по неубыванию или по невозрастанию значениями элементов называется упорядоченным.

Алгоритмы сортировки используют там, где речь идет об обработке и хранении больших объемов информации.

Некоторые задачи обработки данных решаются проще, если данные заранее упорядочить.

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке (массиве).

Если элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки.
На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

Формулировка задачи сортировки

Дан массив A из n элементов: $a_1, a_2, \dots, a_n$,
Будем считать, что с каждым элементом a_i (помимо другой информации, не влияющей на сортировку) связан ключ $k_i \in K$. На множестве ключей K задано отношение порядка - линейного (или совершенного) упорядочивания, для которого были бы выполнены следующие условия:

закон трихотомии: для любых $x,y \in K$ либо $x<y$, либо $x>y$,, либо $x=y$;
транзитивность: для любых $x,y,z \in K$ если $x<y$ и $y<z$, то $x<z$.

Задачей сортировки по неубыванию является нахождение перестановки элементов p(1),p(2),…,p(n) с индексами 1,2,…,n, при которой ключи располагаются в порядке неубывания:

$$k_{p(1)} \le k_{p(2)} \le \dots \le k_{p(n)}.$$

В результате работы алгоритма и применения перестановки получается отсортированный массив $a_{p(1)}, a_{p(2)}, \dots a_{p(n)}$.

Аналогично можно определить **сортировку по невозрастанию**.

5.1 Оценка алгоритма сортировки

Алгоритмы сортировки оцениваются по скорости выполнения и эффективности использования памяти.

Временная оценка характеризует быстродействие алгоритма.

Важны худшая, средняя и лучшая оценка алгоритма в терминах мощности входного множества A:

множество A подаётся на вход алгоритму ;

n = |A| - мощность множества (кол-во элементов массива).

Хорошая оценка O(n log n), плохая оценка $O(n^2)$, идеальная оценка O(n).

Алгоритмы сортировки, использующие только абстрактную операцию сравнения ключей, всегда нуждаются по меньшей мере в Ω (n log(n)) сравнениях.

Существует понятие сортирующих сетей. Предполагая, что можно одновременно (например, при параллельном вычислении) проводить несколько сравнений, можно отсортировать n чисел за $O(\log^2(n))$ операций. При этом число n должно быть заранее известно.

5.1 Оценка алгоритма сортировки (продолжение)

Оценка эффективности использования памяти

Ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных.

Как правило, эти алгоритмы требуют O(log(n)) памяти.

При оценке не учитывается место, которое занимает исходный массив и независящие от входной последовательности затраты, например, на хранение кода программы (так как всё это потребляет O(1)).

Алгоритмы сортировки, не потребляющие дополнительной памяти, относят к сортировкам на месте.

5.2 Оптимальность алгоритма сортировки O(n log(n))

Задача сортировки в общем случае предполагает, что единственной обязательно существующей операцией для элементов является сравнение.

Пусть по ходу работы алгоритмом производится k сравнений.

Ответом на сравнение двух элементов a и b может быть один из двух вариантов ($a < b$ или $a > b$).

Значит, всего возможно 2^k вариантов комбинаций ответов на k вопросов.

Количество перестановок из n элементов равно $n!$.

5.3 Свойства и характеристики алгоритмов сортировки

Свойства сортировки - Устойчивость (stability)

Устойчивая сортировка не меняет взаимного расположения элементов с одинаковыми ключами.

Практически всегда устойчивость может быть достигнута путём удлинения исходных ключей за счёт дополнительной информации об их первоначальном порядке.

Устойчивость не обязательна для правильности сортировки и чаще всего не соблюдается, так как для её обеспечения практически всегда необходимы дополнительная память и время.

5.3 Свойства и характеристики алгоритмов сортировки (продолжение)

Свойства сортировки - Естественность поведения

Естественность поведения - эффективность метода при обработке уже упорядоченных или частично упорядоченных данных.

Алгоритм ведёт себя естественно, если учитывает упорядоченность или частичную упорядоченность данных входной последовательности и работает лучше.

Свойства сортировки - Использование операции сравнения

Алгоритмы, использующие для сортировки сравнение элементов между собой, называются основанными на сравнениях.

Минимальная трудоемкость худшего случая для этих алгоритмов составляет O(n log(n)), но они отличаются гибкостью применения.

5.3 Свойства и характеристики алгоритмов сортировки (продолжение)

Свойства сортировки - Сфера реализации алгоритма

Применительно к типу используемой памяти компьютера есть два основных вида упорядочения – внутренняя и внешняя сортировка.

Внутренняя сортировка

оперирует массивами, целиком помещающимися в RAM.

Данные упорядочиваются на том же месте без дополнительных затрат.

Применяется
- подкачка;
- кэширование памяти.

Алгоритм сортировки должен хорошо сочетаться с применяемыми алгоритмами кэширования и подкачки.

Внешняя сортировка

использует внешние устройства большого объёма с последова-тельным доступом (в каждый момент «виден» только один элемент), а затраты по сравнению с ОП неоправданно велики.

На алгоритм накладываются дополнительные ограничения, используются специальные методы упорядочения (обычно с дополнительным дисковым пространством).

Доступ к носителю осуществляется последовательным образом: в каждый момент времени можно считать или записать только элемент, следующий за текущим.

5.3 Свойства и характеристики алгоритмов сортировки (продолжение)

Классификация алгоритмов сортировки

Алгоритмы сортировки можно классифицировать по различным признакам:

- по устойчивости;
- по поведению;
- по использованию операций сравнения;
- по потребности в дополнительной памяти;
- по потребности в знаниях о структуре данных, выходящих за рамки операции сравнения, и другие.

В приведенной классификации:

n - количество записей, которые необходимо упорядочить;

k - количество уникальных ключей.

Классификация алгоритмов сортировки – Алгоритмы устойчивой сортировки

№ пп	Название	Комментарий	Слож-ность
1	Сортировка выбором (*Selection sort*)	*Поиск наименьшего (наибольшего) элемента и помещение его в начало (в конец) упорядоченного списка.*	$O(n^2)$
2	Сортировка пузырьком (*Bubble sort*)	*Для каждой пары индексов производится обмен, если элементы расположены не по порядку.*	$O(n^2)$
3	Сортировка перемешиванием Шейкер-сортировка (*Cocktail sort*)	*Разновидность пузырьковой сортировки, Массив просматривается поочередно справа налево и слева направо.*	$O(n^2)$
4	Гномья сортировка (*Gnome sort*)	*Схожа с сортировкой пузырьком и сортировкой вставками, но перед вставкой на нужное место происходит серия обменов, как в сортировке пузырьком.*	$O(n^2)$
5	Сортировка вставками (Insertion sort)	*Определяется, где текущий элемент должен находиться в упорядоченном списке, и вставляется туда.*	$O(n^2)$
6	Сортировка слиянием (*Merge sort*)	*Выстраивается первая и вторая половина списка отдельно, а затем объединяются упорядоченные списки. Требуется O(n) дополнительной памяти.*	$O(n \log n)$

Классификация алгоритмов сортировки – Алгоритмы устойчивой сортировки

№ пп	Название	Комментарий	Слож-ность
7	Сортировка с помощью двоичного дерева (*Tree sort*).	*Строится двоичное дерево поиска по ключам массива с последующей сборкой результирующего массива при обходе узлов дерева в требуемом порядке следования ключей. Является оптимальной при непосредственном чтении с потока (например, из файла, сокета или консоли). Требует O(n) дополнительной памяти.*	$O(n \log n)$
8	Timsort (*Timsort*)	*Комбинация алгоритмов сортировки вставками и сортировки слиянием. Требуется O(n) дополнительной памяти.*	$O(n \log n)$
9	Сортировка подсчётом (*Counting sort*)	*Используется диапазон чисел сортируемого массива (списка) для подсчёта совпадающих элементов. Применение целесообразно, когда сортируемые числа имеют (или их можно отобразить в) диапазон возможных значений, который достаточно мал по сравнению с сортируемым множеством, например, миллион натуральных чисел меньших 1000. Требуется O(n+k) дополнительной памяти.*	$O(n+k)$
10	Блочная, корзинная сортировка (*Basket sort*)	*Требуется знание о природе сортируемых данных, выходящее за рамки функций «переставить» и «сравнить». Требуется O(k) дополнительной памяти.*	$O(n)$

Классификация алгоритмов сортировки – Алгоритмы неустойчивой сортировки

№ пп	Название	Комментарий	Сложность
11	Сортировка Шелла (*Shell sort*).	*Улучшение сортировки вставками. Сравнение элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами.*	$O(n \log^2 n)$
12	Сортировка расчёской (*Comb sort*)	*Улучшение сортировки пузырьком, конкурирует с алгоритмами, подобными быстрой сортировке. Основная идея — устранить черепах (маленькие значения в конце списка), которые крайне замедляют сортировку пузырьком (кролики, большие значения в начале списка, не представляют проблемы для сортировки пузырьком).*	$O(n \log n)$
13	Пирамидальная сортировка, сортировка кучи, (*Heap Sort*)	*Список превращается в кучу, наибольший элемент добавляется в конец списка.*	$O(n \log n)$
14	Плавная сортировка (Smoothsort)	*Разновидность пирамидальной сортировки, сложность приближается к O(n), если входные данные частично отсортированы.*	$O(n \log n)$

Классификация алгоритмов сортировки – Алгоритмы неустойчивой сортировки

15	Быстрая сортировка (*Quick Sort*)	*Быстрейший из известных для упорядочения больших случайных списков; с разбиением исходного набора данных на две половины так, что любой элемент первой половины упорядочен относительно любого элемента второй половины; затем алгоритм применяется рекурсивно к каждой половине. При использовании O(n) дополнительной памяти, можно сделать сортировку устойчивой.*	в варианте с минимальными затратами памяти $O(n \log n)$ — среднее время, $O(n^2)$ — худший случай
16	Интроспективная сортировка (*Intro Sort*)	*Сочетание быстрой и пирамидальной сортировки. Пирамидальная сортировка применяется в случае, если глубина рекурсии превышает log n.*	$O(n \log n)$
17	Терпеливая сортировка (*Patience Sort*)	*В наихудшем случае требует дополнительно O(n) памяти, находит самую длинную увеличивающуюся подпоследовательность.*	$O(n \log n)$

Классификация алгоритмов сортировки – Алгоритмы неустойчивой сортировки

18	Сортировка по частям, блуждающая сортировка (*Stooge Sort*)	*Рекурсивный алгоритм сортировки:* *1) если значение элемента в конце списка меньше, чем значение элемента в начале, то поменять их местами;* *2) если есть 3 или более элементов в текущем подмножестве списка, то:* • *рекурсивно вызвать Stooge sort для первых 2/3 списка* • *рекурсивно вызвать Stooge sort для последних 2/3 списка* • *рекурсивно вызвать Stooge sort для первых 2/3 списка снова.*	временная сложность $O(n^{\log_{1,5} 3}) \approx O(n^{2.71})$
19	Поразрядная сортировка, лексикографическая сортировка, цифровая сортировка (*Radix Sort*)	*Существует два варианта: least significant digit (LSD) и most significant digit (MSD). При LSD сортировке, сначала сортируются младшие разряды, затем старшие. При MSD сортировке - наоборот. Требуется O(k) дополнительной памяти.*	$O(n\ k)$

Классификация алгоритмов сортировки – Непрактичные алгоритмы сортировки

№ пп	Название	Комментарий	Слож-ность
20	Случайная сортировка, сортировка ружья, обезьянья сортировка (*Bogo Sort*)	*Массив произвольно перемешивается, проверяется порядок.*	в среднем $O(n \cdot n!)$
21	Сортировка перестановкой	*Для каждой пары осуществляется проверка верного порядка и генерируются всевозможные перестановки исходного массива.*	худшее время $O(n \cdot n!)$
22	Глупая сортировка (*Stupid Sort*)	*Рекурсивная версия требует дополнительно $O(n^2)$ памяти*	$O(n^3)$
23	Bead Sort	*Требуется специализированное аппаратное обеспечение*	$O(n^2)$ или $O(\sqrt{n})$
24	Блинная сортировка (*Pancake Sort*)	*Требуется специализированное аппаратное обеспечение*	$O(n)$

Классификация алгоритмов сортировки – Алгоритмы, не основанные на сравнениях

Блочная сортировка, корзинная сортировка, (Basket Sort)
Лексикографическая, поразрядная сортировка (Radix Sort)
Сортировка подсчётом (Counting Sort)

Прочие алгоритмы сортировки

№ пп	Название	Комментарий
28	Топологическая сортировка	*Упорядочивание вершин бесконтурного ориентированного графа согласно частичному порядку, заданному ребрами орграфа на множестве его вершин (Алгоритм Кана, Алгоритм Тарьяна)*
29	Внешняя сортировка	*Сортировка данных, расположенных на периферийных устройствах и не вмещающихся в оперативную память*

5.4 Эффективные алгоритмы сортировки

Алгоритм быстрой сортировки

- используется гораздо чаще любого другого;
- прост в реализации;
- хорошо работает на различных видах входных данных;
- во многих случаях требует меньше затрат ресурсов по сравнению с другими методами сортировки;
- принадлежит к категории обменных сортировок (требует небольшого вспомогательного стека);
- на выполнение сортировки N элементов в среднем затрачивается время, пропорциональное Nlog(N).

Недостаток: алгоритм неустойчив, для его выполнения в наихудшем случае требуется N^2 операций.

5.4 Эффективные алгоритмы сортировки

Алгоритм быстрой сортировки (продолжение)

Функционирует по принципу "разделяй и властвуй".
Делит сортируемый массив на две части, затем сортирует эти части независимо друг от друга.
Точное положение точки деления зависит от исходного порядка элементов в исходном массиве.

Массив разбивается на две части при соблюдении условий:

- элемент a[i] для некоторого i занимает свою окончательную позицию в массиве;
- ни один из элементов a[i],...,a[i-l] не превышает a[i];
- ни один из элементов a[i+l],..., a[r] не является меньшим a[i].

Полная сортировка достигается путем деления массива на подмассивы с последующим применением к ним этих же методов.

Процесс разбиения всегда помещает, по меньшей мере, один из элементов в окончательную позицию.

Алгоритм рекурсивный, обеспечивает правильную сортировку.

5.4 Эффективные алгоритмы сортировки

Алгоритм быстрой сортировки - оценка сложности

Операция разделения массива на две части относительно опорного элемента занимает время O(n).

Все операции разделения, выполняемые на одной глубине рекурсии, обрабатывают разные части исходного массива, размер которого постоянен, суммарно на каждом уровне рекурсии потребуется O(n) операций.

Общая сложность алгоритма определяется количеством разделений, то есть глубиной рекурсии.

Глубина рекурсии, в свою очередь, зависит от сочетания входных данных и способа определения опорного элемента.

5.4 Эффективные алгоритмы сортировки

Алгоритм быстрой сортировки - оценка сложности

Лучший случай

При каждой операции разделения массив делится на две почти одинаковые части.

Максимальная глубина рекурсии, при которой размеры обрабатываемых подмассивов достигнут 1, равна log2(n).

Количество сравнений: $C_n=2\cdot C_{n/2}+n$,

Общая сложность алгоритма $O(n\cdot log2\ (n))$.

5.4 Эффективные алгоритмы сортировки

Алгоритм быстрой сортировки - оценка сложности

Средний случай - при случайном распределении входных данных оценивают вероятностно.

При любом фиксированном соотношении между левой и правой частями разделения сложность алгоритма = O(n·log(n)).

«Удачное» разделение (длина подмвссива > 25%, <= 75%) даёт глубину рекурсии <= log3/4(n).

Вероятность удачи =0,5.

Для получения k удачных разделений в среднем потребуется 2k рекурсивных вызовов, чтобы опорный элемент k раз оказался среди центральных 50% массива.

В среднем глубина рекурсии <= 2·log3/4(n), сложность = O(log(n)).

На каждом уровне рекурсии выполняется <= O(n) операций, средняя сложность = O(n·log(n)).

5.4 Эффективные алгоритмы сортировки

Алгоритм быстрой сортировки - оценка сложности

Худший случай

Каждое разделение даёт два подмассива размерами 1 и n-1, если в качестве опорного на каждом этапе будет выбран либо наименьший, либо наибольший элемент из всех обрабатываемых.

При каждом рекурсивном вызове больший массив будет на 1 короче, чем в предыдущий раз.

Потребуется n-1 операций разделения,
общее время работы $\Sigma ni=0(n-i)=O(n^2)$ операций.

Глубина рекурсии при выполнении алгоритма достигнет n с n-кратным сохранением адреса возврата и локальных переменных процедуры разделения массивов.

Для больших значений n худший случай может привести к исчерпанию памяти (переполнению стека) во время работы программы.

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм сортировки вставками

Суть сортировки вставками

1. Берется один из элементов массива.
2. Находится позиция для вставки.
3. Рассматриваемый элемент вставляется.

Массив из одного элемента считается отсортированным.

Сортировка вставками наиболее эффективна:

* когда массив уже частично отсортирован;
* когда элементов массива не много.

Если элементов меньше 10, то данный алгоритм является лучшим.

В быстрой сортировке алгоритм сортировки вставками используется как вспомогательный.

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм сортировки вставками - оценкасложности

Алгоритм имеет сложность $O(n^2)$:

* на практике приблизительное количество перестановок вычисляется по формуле $O(n^2/2)$;
* количество сравнений вычисляется по формуле $n \cdot (n-1)/2$.

Время выполнения алгоритма зависит от

* входных данных: чем большее множество нужно отсортировать, тем большее время выполняется сортировка;
* исходной упорядоченности массива:
  ** лучший случай - отсортированный массив;
  ** худший случай - массив, отсортированный в порядке, обратном нужному.

Временная сложность алгоритма при худшем варианте - $O(n^2)$.

Алгоритм можно ускорить при помощи бинарного поиска для нахождения места текущему элементу в отсортированной части.

Проблема с долгим сдвигом массива вправо решается при помощи смены указателей.

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм сортировки вставками – достоинства и недостатки

Достоинства :

- эффективен на небольших наборах данных, на наборах данных до десятка элементов может оказаться лучшим;
- эффективен на наборах данных, которые уже частично отсортированы;
- обладает устойчивостью (не меняет порядок элементов, которые уже отсортированы);
- может сортировать список по мере его получения;
- использует O(1) временной памяти, включая стек.
- может работать значительно быстрее за счёт бинарного поиска (использующего дробление массива на половины).

Недостаток: высокая вычислительная сложность алгоритма $O(n^2)$ (при использовании стандартного алгоритма).

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм сортировки слиянием

Является дополнением быстрой сортировки –
состоит из двух рекурсивных вызовов с последующей процедурой слияния.

Достоинство: *сортирует массив, состоящий из N элементов, за время, пропорциональное NlogN, независимо от характера входных данных.*

Недостаток: *прямолинейные реализации этого алгоритма требуют дополнительного пространства памяти, пропорционального N.*

Свойства:

- *устойчивая сортировка,*
- *обычно осуществляет последовательный доступ к данным (один элемент за другим).*

Применяется к связным спискам, для которых из всех методов доступа применим только метод последовательного доступа.

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм сортировки слиянием

Алгоритм использует базовую абстрактную операцию – **слияние.**

Двухпутевое слияние

Имеем два непересекающихся упорядоченных массива целых чисел A [0],...,A[N-l] и B[0],...,B[M-1].
Требуется исходные массивы слить в третий массив C[0],...,C[N+M-l].

Для массива C последовательно выбирают наименьший оставшийся элемент из A и B.

Область применения слияния
В среде обработки данных необходимо поддерживать крупный (упорядоченный) массив данных, в который непрерывно поступают новые элементы. Новые элементы группируются в пакеты, которые затем добавляются в главный (намного больший) массив, после чего выполняется очередная сортировка всего массива.

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм сортировки слиянием

Алгоритм использует базовую абстрактную операцию – **слияние.**

Абстрактное обменное слияние

Имеем два возрастающих массива.

1. Исходные массивы копируются во вспомогательный массив, при этом второй массив в обратном порядке непосредственно следует за первым.

2. На выход перемещают левый или правый элемент в зависимости от того, какой из них меньше.

3. Наибольший ключ служит служебной меткой для другого массива, независимо от того, в каком массиве этот ключ находится.

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм сортировки слиянием

Нисходящая сортировка слиянием имеет основой рекурсивную процедуру сортировки.

1. Массив a[1],..., a[r] делится на две части a[1],...,a[m] и a[m+1],...,a[r].
2. Полученные части сортируют независимо друг от друга (через рекурсивные вызовы).
3. Выполняют слияние полученных упорядоченных подмассивов в отсортированный результирующий массив.
4. Функция может потребовать использования вспомогательного массива, достаточно большого, чтобы принять копию входного массива.

```
template <class Item>
void mergesort (Item a[] , int 1, int r)
{ if (r <= 1) return;
int m = (r+l)/2;
mergesort(a, 1, m) ;
mergesort(a, m+1, r) ;
merge(a, 1, m, r);
}
```

5.4 Эффективные алгоритмы сортировки (продолжение)

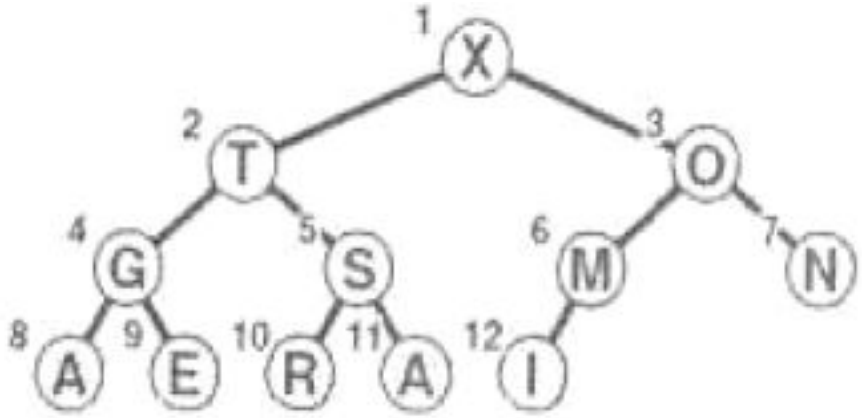
Алгоритм пирамидальной сортировки

Пирамидальная структура данных

Дерево называется пирамидально упорядоченным, если ключ в каждом его узле больше или равен ключам всех потомков этого узла (если таковые имеются).

Удобно пользоваться полным бинарным деревом (complete binary tree).

В верхней части помещают корневой узел, а затем спускаясь вниз и перемещаясь слева направо, подсоединяют к каждому конкретному узлу предыдущего уровня два узла текущего уровня до тех пор, пока не будут помещены все N узлов.



5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм пирамидальной сортировки

Сортирующее дерево - совокупность узлов с ключами, образующими полное пирамидально упорядоченное бинарное дерево, представленное в виде массива.

Полные деревья предоставляют возможность задействовать компактное представление в виде массива, в котором легко переходить с некоторого узла к его родителю или к его предкам без необходимости поддержки явных связей.

Родителя узла, находящегося в позиции i, необходимо искать в позиции $\lfloor i/2 \rfloor$, и, соответственно, два потомка узла в позиции i находятся в позициях 2i и 2i+1.
Прохождение по такому дереву выполняется проще, чем в связном представлении, где могут понадобиться связи дерева, принадлежащие каждому ключу, чтобы иметь возможность перемещаться вверх и вниз по дереву (каждый элемент будет иметь один указатель на родителя и один указатель на каждого потомка).

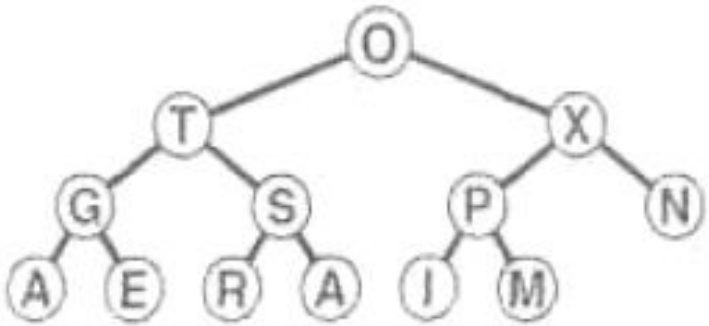
5.4 Эффективные алгоритмы сортировки (продолжение)

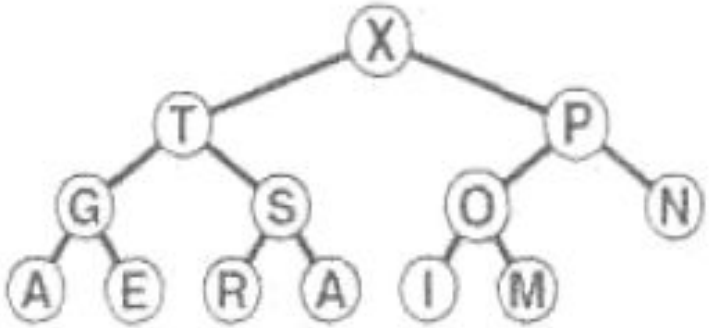
Алгоритм пирамидальной сортировки

При нарушении пирамидальности сортирующего дерева ключ какого-либо узла становится меньше, чем один или оба ключа его потомков.

Нисходящее восстановление пирамидальности сортирующего дерева

1. Заменить узел на больший из его двух потомков.
2. При нарушении свойств сортирующего дерева на узле-потомке устранить это нарушение аналогичным путем.
3. Выполнить продвижение вниз по дереву до тех пор, пока не будет достигнут узел, оба потомка которого меньше его самого, либо нижний уровень дерева.





В примере потомки узла сортирующего дерева в позиции k занимают в нем позиции 2k и 2k+1.

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм пирамидальной сортировки

Пирамидальная сортировка (heapsort)

1. Строится сортирующее дерево сверху вниз без использования вспомогательной памяти.
2. Из дерева многократно удаляется наибольший элемент.

Достоинства алгоритма:

- сортировка N элементов выполняется за время, пропорциональное NlogN независимо от природы входного потока данных;
- возможно выполнения сортировки без использования вспомогательной памяти.
- не бывает входных данных, вызывающих наихудший случай;
- не использует дополнительное пространство памяти (в отличие от сортировки слиянием).

Недостаток: работает медленнее быстрой сортировки, т.к. внутренний цикл алгоритма выполняет больше базовых операций (операций сравнения).

Сортирующие деревья можно использовать для выборки k максимальных элементов из N элементов, особенно в случаях, когда k мало. После того, как k элементов будут отобраны из вершины сортирующего дерева, выполнение алгоритма прекращается.

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм поразрядной сортировки MSD

Ключи, используемые для определения порядка следования записей в файлах, могут иметь сложную природу.

Часто нет необходимости в обработке ключей в полном объеме на каждом этапе.

В сортировочных алгоритмах переходят от абстрактных операций сравнения ключей к абстракциям, в условиях которых ключи разделяются на последовательности порций фиксированных размеров или байтов.

Двоичные числа - последовательности битов,
строки - последовательности символов,
десятичные числа — последовательности цифр,

и многие другие типы ключей можно рассматривать под таким же углом зрения.

Методы сортировки, построенные на обработке чисел по одной порции за раз, называются поразрядными (radix) методами сортировки.

Эти методы не только выполняют сравнение ключей, они обрабатывают и сравнивают соответствующие части ключей.

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм поразрядной сортировки MSD

Алгоритмы поразрядной сортировки рассматривают ключи как числа, представленные в системе счисления с основанием R при различных значениях R (основание системы счисления), и работают с отдельными цифрами чисел.

Для различных приложений подходят различные основания системы счисления R.

Для ключей, в состав которых входят строки символов, используют R = 128 или R = 256, приравнивая основание системы счисления к размеру байта.

Для создания ключей можно рассматривать фактически все, что может быть представлено в цифровом компьютере как двоичное число.

Поэтому можно использовать различные ключи и выполнять поразрядную сортировку для упорядочения ключей-двоичных чисел.

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм поразрядной сортировки MSD

Условия для поразрядной сортировки:

- компьютеры в общем случае ориентированы на обработку групп битов, называемых машинными словами, которые в свою очередь часто объединяются в небольшие фрагменты, называемые байтами;
- ключи сортировки обычно также организуются в последовательности байтов,
- короткие последовательности байтов могут также служить индексами массивов или машинными адресами.

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм поразрядной сортировки MSD

Условия для поразрядной сортировки:

- компьютеры в общем случае ориентированы на обработку групп битов, называемых машинными словами, которые в свою очередь часто объединяются в небольшие фрагменты, называемые байтами;
- ключи сортировки обычно также организуются в последовательности байтов,
- короткие последовательности байтов могут также служить индексами массивов или машинными адресами.

Байт - последовательность битов фиксированной длины.
Строка - последовательность байтов переменной длины.
Слово - последовательность байтов фиксированной длины.

В зависимости от контекста ключом в поразрядной сортировке может быть слово или строка.
Некоторые алгоритмы поразрядной сортировки используют ключи с фиксированной длиной (слова), другие - ключи с переменной длиной (строки).

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм поразрядной сортировки MSD

Алгоритмы поразрядной сортировки анализируют значение цифр в ключах в направлении слева направо. Первыми обрабатываются наиболее значащие цифры.

Такие методы в общем случае называются поразрядной сортировкой MSD (most significant digit radix sort — поразрядная сортировка сначала по старшей цифре).

В поразрядной сортировке MSD анализируется минимальный объем информации, необходимый для выполнения сортировки.

Поразрядная сортировка MSD обобщает понятие быстрой сортировки, поскольку она выполняется за счет разделения сортируемого массива в соответствии со старшими цифрами ключей, после чего тот же метод применяется к подмассивам в режиме рекурсии.

В условиях, когда в качестве основания системы счисления u1074 выбрана 2, поразрядная сортировка MSD реализуется тем же способом, что и быстрая сортировка.

5.4 Эффективные алгоритмы сортировки (продолжение)

Алгоритм поразрядной сортировки MSD

Поразрядная сортировка по младшим разрядам

Исходный массив	872	488	912	7	60	575	943	987	353	537
1-й разряд	60	872	912	943	353	575	7	987	537	488
2-й разряд	7	912	537	353	943	60	872	575	987	488
3-й разряд	7	60	353	488	577	575	872	912	943	957

Поразрядная сортировка по старшим разрядам

Исходный массив	7	268	987	2	260	537	943	982	353	535
3-й разряд	7	2	268	260	353	537	535	987	943	982
2-й разряд	7	2	268	260	353	537	535	943	987	982
1-й разряд	2	7	260	268	353	535	537	912	982	987