**VI.СОРТИРОВКА ДАННЫХ**

**Общие понятия**

Сортировку следует понимать как процесс перегруппировкой однотипных элементов структуры данных в некотором определенном порядке. Цель сортировки — облегчить последующий поиск, обновление, исключение, включение элементов в структуру данных. На отсортированных данных легче определить имеются ли пропущенные элементы, все ли элементы проверены, легче найти общие элементы двух однотипных структур слить их воедино. Сортировка является важным средством для ускорения работы практически любого алгоритма, в которой требуется частое обращение к определенным элементам структуры данных.

Разработано множество алгоритмов сортировки, однако нет алгоритма, который был бы наилучшим в любом случае. Эффективность алгоритма сортировки может зависеть от ряда факторов, таких, как: число сортируемых элементов; диапазон и распределение значений сортируемых элементов; степень отсортированности элементов; характеристики алгоритма (сложность требования к памяти и т.п.); место размещения элементов (в оперативной памяти или на **ВЗУ).**

В зависимости от последнего фактора все методы сортировки разбивают на два класса: *внутренняя сортировка* (сортировка массивов) и *внешняя сортировка* (сортировка файлов, или сортировка последовательностей).

Сортируемые элементы часто представляют собой записи данных определенной структуры. Каждая запись имеет поле ключа по значению которого осуществляется сортировка, и поля данных При рассмотрении алгоритмов сортировки нас интересует только поле ключа, поэтому другие поля опускаются из рассмотрения.

Метод сортировки называется *устойчивым,* если в процессе сортировки относительное расположение элементов с одинаковыми (равными) ключами не изменяется. Устойчивость сортировки желательна, если речь идет об элементах, уже отсортированных по некоторым другим ключам (свойствам), не влияющим на ключ, по которому сейчас осуществляется сортировка.

**Внутренняя сортировка**

Основными требованиями к алгоритмам сортировки, как и ко всяким алгоритмам, являются требования по памяти и по времени выполнения. Это предполагает, что сортировка элементов массива выполняется на месте, без передачи их в результирующий массив. Хорошей мерой эффективности алгоритма по времени могут быть число необходимых сравнений ключей *С* и число пересылок *М,* которые зависят от числа элементов *п.*

Самыми простыми методами сортировки являются так называемые *прямые методы,* они требуют порядка *О(п2)* сравнений ключей. *Быстрые (улучшенные) методы* сортировки в самом благоприятном случае требуют порядка *O*(*n*\**1оg2n*) сравнений, имеются многочисленные варианты прямых и быстрых методов сортировки.

Прямые методы сортировки, не требующие дополнительной памяти, можно разбить на три категории:

• сортировка методом прямого включения;

• сортировка методом прямого выбора;

• сортировка методом прямого обмена (сортировка методом пузырька).

Быстрые методы сортировки являются улучшением прямых методов сортировки.

**1. Сортировка методом**

**прямого включения**

Имеется последовательность элементов *а0, а1, а2, ...,ап-1.* Эта последовательность мысленно делится на две последовательности: упорядоченную последовательность *а0*, *а1, ..., аi-1* и исходную, неупорядоченную *аi, аi+1*, *..., аn-1.* Очевидно, что первоначально упорядоченная последовательность состоит из единственного элемента *а0*. При каждом шаге, начиная с *i*= 1 и увеличивая *i* каждый раз на единицу, из исходной последовательности извлекается *i*-й элемент *аi* и вставляется в готовую последовательность в нужное место, при необходимости сдвигая элементы готовой последовательности на одну позицию вправо вплоть до *i*-й позиции. Поиск подходящего места осуществляется сравнением аi с элементами *аj, j=i-1,i-2, …,0,* и одновременным обменом местами *аi* и *аj*, если *аi* < *аj*, т.е. сдвигом *аj* на одну позицию вправо. Процесс поиска заканчивается при выполнении одного из условий:

• *аi > = аj,*

*•* достигнут левый конец готовой последовательности.

Этот метод обеспечивает устойчивую сортировку.

Для процедуры сортировки методом прямого включения число сравнений *С* и число пересылок *М* таковы : *Cmin*= n - 1; *Cсредн* = *(п2 + п -* 2)/4; *Сmax* = *(п2 - п)/2 -* 1; *Mсредн* = (*n2*+ *9п -* 10)/4.

Исходный массив:

27 412 71 81 59 14 273 87

Отладочная печать по шагам сортировки:

i=1 27 412 71 81 59 14 273 87

i=2 27 71 412 81 59 14 273 87

i=3 27 71 81 412 59 14 273 87

i=4 27 59 71 81 412 14 273 87

i=5 14 27 59 71 81 412 273 87

i=6 14 27 59 71 81 273 412 87

i=7 14 27 59 71 81 87 273 412

Отсортированный массив:

14 27 59 71 81 87 273 412.

Алгоритм с прямыми включениями можно легко улучшить с учетом того, что готовая последовательность, куда надо вставить новый элемент, уже упорядочена. Тогда место включения ищется методом двоичного (бинарного) поиска. Такой улучшенный алгоритм сортировки называется *методом с бинарными включениями (методом прямого включения с делением пополам).*

Среднее число сравнений здесь *С= O(n1оg2n),* а среднее число сдвигов *М* = *О(п2).*

**2. Сортировка методом**

**прямого выбора**

Алгоритм сортировки заключается в следующем:

1. В исходной последовательности из *п* элементов отыскивается элемент с наименьшим ключом.

2. Он меняется местами с первым элементом.

3. В оставшейся последовательности из *(п -* 1) элементов отыскивается минимальный элемент и меняется местами со вторым элементом и т.д., пока не останется один, самый болыиой эле-мент.

Число сравнений *С = (п2 - п),* число перестановок: *Мсредн*= n(ln *п + g),* где g= 0.577216 — константа Эйлера.

Отсюда алгоритм с прямым выбором несколько предпочтительней алгоритма прямого включения. Однако если ключи вначале почти упорядочены, то прямое включение будет несколько быстрее.

**3. Сортировка методом**

**прямого обмена**

**(сортировка методом пузырька)**

Алгоритм прямого обмена основывается на сравнении и смене мест для пары соседних элементов, в результате одного прохода через последовательность по направлению с ее конца к началу самый меньший (легкий) элемент оказывается в начале последовательности (или самый большой элемент в конце последовательности при обратном направлении). При втором проходе следующий меньший элемент сдвинется к началу последовательности и т.д., пока все элементы не будут упорядочены.

Алгоритм можно улучшить с учетом того, что если в очередном проходе не было обмена, то последовательность упорядочена. Число сравнений в прямом обменном алгоритме

*Cmin*= n; *Сmax* = *(п2 - п)/2,* число перестановок *Mсредн* = 3(*n2-n*)/4.

Исходный массив:

27 412 71 81 59 14 273 87.

Отладочная печать по шагам сортировки:

i=0 14 27 412 71 81 59 87 273

i = 1 14 27 59 412 71 81 87 273

i = 2 14 27 59 71 412 81 87 273

i= 3 14 27 59 71 81 412 87 273

i*=*414 27 59 71 81 87 412 273

i=5 14 27 59 71 81 87 273 412

Отсортированный массив:

14 27 59 71 81 87 273 412.

**4. Шейкерная сортировка**

Особенностью пузырьковой сортировки является то, что легкий пузырек всплывает сразу, за один проход от конца последовательности к ее началу, где бы он ни находился, в то время как тяжелый пузырек тонет очень медленно, на одну позицию за один проход. Если бы изменить направление просмотра, то тяжелый элемент опустился бы вниз за один проход. Это свойство использовано в алгоритме шейкерной сортировки, в котором чередуются направления последовательных просмотров.

Шейкерная сортировка хороша тогда, когда элементы последовательности почти упорядочены.

**Быстрые (улучшенные)**

**методы сортировки**

Все прямые методы сортировки фактически передвигают каждый элемент на всяком элементарном шаге на одну позицию. Поэтому они требуют порядка *О(п2)* таких шагов. Отсюда следует, что в основу любых улучшений должен быть положен принцип перемещения элементов на каждом шаге на возможно большие расстояния.

**1. Метод Шелла**

Метод Шелла является улучшением метода сортировки с помощью прямого включения и основан на сортировке посредством включений с уменьшающимися расстояниями. Сначала отдельно группируются и сортируются методом прямых включений элементы, отстоящие друг от друга на некотором расстоянии *h1,* затем на расстоянии *h2< h1* и т.д., последнее расстояние должно быть равно 1. Таким образом, если для сортировки будет использовано *t* расстояний, то *ht*=1, *hi+1<hi*. Желательно, чтобы расстояния обеспечивали бы взаимодействие различных цепочек как

можно чаще.

Имеет смысл использовать такие последовательности (записаны в обратном порядке):

1) 1, 4, 13, 40, 121, ...,т.e *hk-1* = 3*hk*+1*, ht=* 1, *t* = [log3n]- 1

2) 1,3,7, 15,31, ..., т.е. *hk-1* =2 *hk* + 1, *ht* = 1, *t=* [log2*n*] - 1. В последнем случае для сортировки *п* элементов методом Шелла затраты пропорциональны *n*1,2 , что значительно лучше *n*2, но хуже *n*\*log2 *n.*

3) часто используют *h1= п/2, h2* =*n*/4, ..., и т.д.

На каждом шаге методом включения сортируются группы *a1 ,a1+h , a1+2h,…., a1+kh ,1+kh< = n; a2,a2+h* ….Так как начальный шаг является наибольшим, то число элементов массива в начальной группе является наименьшим, например, если *h* = n/2, то группируются только два элемента, При втором проходе шаг уменьшается, а число элементов в группе увеличивается, и при *h* = 1 группа включает всю последовательность целиком.

Рассмотрим процесс сортировки последовательности из n=8 целых чисел :

5, 6, 2, 4, 8, 3, 1, 7.

При *h*1=n/2=4 получим четыре группы по два элемента в каждой: 5, 8; 6, 3; 2, 1; 4, 7.

Сортируем каждую последовательность, в результате получим: 5, 8; 3, 6; 1, 2; 4, 7.

Затем группируем первые элементы с первыми, вторые – со вторыми, получаем последовательность 5, 3, 1, 4, 8, 6, 2, 7.

Принимаем *h2* = *h*1/2=2. Получим две последовательности по 4 элемента в каждой, объединяя первый, третий, пятый и седьмой элементы, и второй, четвертый, шестой и восьмой:

5, 1, 8, 2; 3, 4, 6, 7. Сортируем каждую последовательность, в результате получим:

1, 2, 5, 8; 3, 4, 6, 7. Группируем первые элементы с первыми, вторые – со вторыми, третьи – с третьими, четвертые – с четвертыми: 1, 3, 2, 4, 5, 6, 8, 7. Данная последовательность уже частично отсортирована.

И, наконец, *h3* = *h2*/2=1. Сортируем данную последовательность и получаем:

1, 2, 3, 4, 5, 6, 7, 8.

### 2.Сортировка с разделением (быстрая сортировка)

Методика «быстрой сортировки» взята из повседневного опыта. Чтобы отсортировать большую стопку алфавитных карточек по именам, можно разбить ее на две меньшие стопки относительно какой-нибудь буквы, например К. Все имена, меньшие или равные К идут в одну стопку, а остальные – в другую. Затем каждая стопка снова делится на две и т.д.

В алгоритме быстрой сортировки применяется метод разбиения с определением центрального элемента. Исходный массив разделяется на две части - элементы большие центрального и меньшие. То же самое делается с обеими полученными частями массива и т.д., пока каждая часть не будет содержать только один элемент.

Принцип сортировки:

1. Выбирается центральный элемент массива. Все элементы массива просматриваются поочередно слева направо и справа налево. При движении слева направо ищем элемент A[scanUp], который будет больше или равен центральному и запоминаем его позицию. При движении справа налево ищем элемент A[scanDown], который будет меньше или равен центральному, и также запоминаем его позицию. Найденные элементы меняем местами и продолжаем поиск пока встречные индексы scanUp и scanDown не пересекутся. После выполнения первого этапа элементы исходного массива окажутся разделенными на две части относительно центрального значения.
2. На втором этапе повторяются действия первого этапа для правой и левой частей массива в отдельности.
3. На третьем этапе все повторяется для четырех частей массива. И т.д.

Для обработки подсписков используется рекурсия.

Пример:

Пусть дан массив, состоящий из 10 целых чисел:800, 150, 300, 600, 550, 650, 400, 350, 450, 700

Массив простирается от индекса low=0 до индекса high=9. Его середина приходится на индекс mid=4. Первым центральным элементом является A[mid]=550.

Далее см. отсканированные файлы page1.jpg , page2.jpg