СибГУТИ
Кафедра ВС
Программирование на языке высокого уровня (ПЯВУ), семестр 2
2010-2011 учебный год
Поляков А.Ю., Молдованова О.В.

Лекция №1. Алгоритмы сортировки.

Алгоритмы сортировки

Алгоритм сортировки — это алгоритм для упорядочения элементов в списке. В случае, когда 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.
В терминах комбинаторики отсортированный массив - массив не содержащий инверсий.
Инверсия — пара элементов массива (a_i,a_j), для которых имеет место неравенство $a_i > a_j$ при $i < j$.

Оценка алгоритма сортировки

Единственного эффективнейшего алгоритма сортировки нет, ввиду множества параметров оценки эффективности:

1. Время — основной параметр, характеризующий быстродействие алгоритма. Называется также вычислительной сложностью. Для упорядочения важны ***худшее, среднее и лучшее*** поведение алгоритма в терминах размера списка (n). Для типичного алгоритма хорошее поведение — это O(n log n) и плохое поведение — это $O(n^2)$. Идеальное поведение для упорядочения — O(n). Алгоритмы сортировки, использующие только абстрактную операцию сравнения ключей всегда нуждаются по меньшей мере в O(n log n) сравнениях в среднем;
2. Память — ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных. При оценке используемой памяти не будет учитываться место, которое занимает исходный массив и независящие от входной последовательности затраты, например, на хранение кода программы.
3. Устойчивость (stability) — устойчивая сортировка не меняет взаимного расположения равных элементов.
4. Естественность поведения — эффективность метода при обработке уже упорядоченных, или частично упорядоченных данных. Алгоритм ведёт себя естественно, если учитывает эту характеристику входной последовательности и работает лучше.

Ещё одним важным свойством алгоритма является его сфера применения. Здесь основных типов упорядочения два:

1. Внутренняя сортировка оперирует с массивами, целиком помещающимися в оперативной памяти с произвольным доступом к любой ячейке. Данные обычно упорядочиваются на том же месте, без дополнительных затрат. В современных архитектурах персональных компьютеров широко применяется подкачка и кэширование памяти. Алгоритм сортировки должен хорошо сочетаться с применяемыми алгоритмами кэширования и подкачки.
2. Внешняя сортировка оперирует с запоминающими устройствами большого объёма, но с доступом не произвольным, а последовательным (упорядочение файлов), т.е в данный момент мы 'видим' только один элемент, а затраты на перемотку по сравнению с памятью неоправданно велики. Это накладывает некоторые дополнительные ограничения на алгоритм и приводит к специальным методам упорядочения, обычно использующим дополнительное дисковое пространство. Кроме того, доступ к данным

на носителе производится намного медленнее, чем операции с оперативной памятью.

1. доступ к носителю осуществляется последовательным образом: в каждый момент времени можно считать или записать только элемент, следующий за текущим
2. объём данных не позволяет им разместиться в ОЗУ

Математические обозначения

«O» большое — математическое обозначение для сравнения асимптотического поведения функций. Используются в различных разделах математики. Активнее всего — в математическом анализе, теории чисел и комбинаторике, а также при оценке сложности алгоритмов. В частности, фраза «сложность алгоритма есть O(n!)» означает, что при больших n время работы алгоритма (или общее количество операций) не более чем C*n!, где C — некая положительная константа (обычно в качестве параметра n берут объем входной информации алгоритма).

Список алгоритмов сортировки

В приведенных ниже таблицах n — это количество записей, которые необходимо упорядочить, а k — это количество уникальных ключей.

Алгоритмы устойчивой сортировки

Название	**Сложность**	**Доп. память**	**Примечание**
Сортировка пузырьком (Bubble sort)	$O(n^2)$	-	для каждой пары индексов производится обмен, если элементы расположены не по порядку
Сортировка перемешиванием (Cocktail sort)	$O(n^2)$	-	
Сортировка вставками (Insertion sort)	$O(n^2)$	-	Для каждого элемента определяется его позиция в упорядоченном списке и выполняется вставка
Блочная сортировка (Bucket Sort)	$O(n)$	$O(k)$	
Сортировка подсчетом (Counting sort)	$O(n+k)$	$O(n+k)$	
Сортировка слиянием (Merge sort)	$O(n \log n)$	$O(n)$	Список делится пополам, каждая половина выстраивается отдельно, далее списки объединяются
Двоичное дерево сортировки (Binary tree sort)	$O(n \log n)$	$O(n)$	
Поразрядная сортировка (Radix sort)	$O(nk)$	$O(n)$	Сортировка выполняется сортирует строки буква за буквой (разряд за разрядом)

Алгоритмы неустойчивой сортировки

Название	**Сложность (худш)**	**Сложность (средн)**	**Примечание**
Сортировка выбором (Selection sort)	$O(n^2)$	-	поиск наименьшего или наибольшего элемента и помещения его в начало или конец упорядоченного списка
Сортировка Шелла (Shell sort)	*O(n log n)*	-	попытка улучшить сортировку вставками
Сортировка расческой (Comb Sort)	*O(n log n)*	-	
Пирамидальная сортировка (Heapsort)	*O(n log n)*	*O(n log n)*	Список преобразуется в "кучу". Наибольший элемент из кучи добавляется в отсортированный список.
Плавная сортировка (Smooth sort)	*O(n log n)*	-	
Бытрая сортировка (Quicksort)	$O(n^2)$	*O(n log n)*	Считается самым быстрым из известных для упорядочения больших случайных списков; Исходный набор занных разбиенивается на две половины так, что любой элемент первой половины упорядочен относительно любого элемента второй половины; затем алгоритм применяется рекурсивно к каждой половине
Introsort	*O(n log n + k)*	-	требует дополнительно O(n + k) памяти, также находит самую длинную увеличивающуюся подпоследовательность

Сортировка выбором (selection sort)
'Очевидный' алгоритм сортировки - перебором находится наименьший элемент, он меняется местами с элементом, стоящим на нулевом месте, затем находится наименьший среди оставшихся и меняется местами с элементом, стоящим на первом месте. Цикл заканчивается когда будут выбраны все элементы:

Этот алгоритм отличается небольшим числом перестановок (n-1) (операций перемещения данных в три раза больше - 3*(n-1)), число сравнений велико - n*(n-1)/2.

Пузырьковая сортировка (bubble sort)
Пузырьковая сортировка основана на методе перестановок. В процессе работы очередной элемент сравнивается с соседним и при необходимости выполняется их перестановка. Элементы массива при этом уподобляются всплывающим пузырькам в сосуде с водой. Число сравнений всегда N*(N-1)/2, число перестановок в худшем случае такое же (3*N*(N-1)/2 перемещений данных).

Сортировка перемешиванием (Cocktail sort)
Разновидность пузырьковой сортировки. Отличается тем, что просмотры элементов выполняются один за другим в противоположных направлениях, при этом большие элементы стремятся к концу массива, а маленькие — к началу.

Лучший случай для этой сортировки — отсортированный массив (O(n)), худший — отсортированный в обратном порядке ($O(n^2)$).

***Сортировка вставками (insertion sor*t)
Сортируемый масив просматривается в порядке возрастания номеров и каждый элемент вставляется в уже просмотренную часть массива так, чтобы сохранить порядок. На каждом шаге сортировки часть массива уже упорядочена, поэтому для поиска места вставки можно использовать метод половинного деления:

Число сравнений значительно меньше, чем при сортировке отбором (<n log(n)), Число перемещений данных в худшем случае n*(n+3)/2.

Сортировка Шелла (Shell sort)
При сортировке Шелла сначала выполняется сортировка элементов, отстоящих один от другого на некотором расстоянии d. В качестве алгоритма сортировки могут использоваться **сортировка вставками, выбором и пузырьковая сортировка**. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочиванием элементов при d = 1 (то есть, обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места. В простых методах сортировки вставками или пузырьком каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, при сортировке Шелла же это число может быть больше.

Расстояния между сравниваемыми элементами могут изменяться по-разному. Обязательным является лишь то, что последний шаг должен равняться единице. Например, хорошие результаты дает последовательность шагов 9, 5, 3, 2, 1, которая использована в показанном выше примере. Следует избегать последовательностей степени двойки, которые, как показывают сложные математические выкладки, снижают эффективность алгоритма сортировки. /Однако, при использовании таких последовательностей шагов между сравниваемыми элементами эта сортировка будет по-прежнему работать правильно/.

Пирамидальная сортировка (heap sort)
Этот алгоритм почти также эффективен, как сортировка слиянием. К тому же не требует дополнительной памяти.

Алгоритм основан на том, что сортируемому массиву может быть поставлено в соответствие двоичное дерево, каждый узел которого соответствует одному элементу массива с некоторым номером k и этот узел содержит ссылки на два других узла, соответствующих элементам с номерами 2*k+1 и 2*k+2. Корень дерева соответствует элементу с номером 0 (ноль).

Дальнейшие действия следующие - массив переупорядочивается так, чтобы для любого k выполнялись неравенства:

```
Buff[k]>=Buff[2*k+1]
Buff[k]>=Buff[2*k+2]
```

Затем массив еще раз переупорядочивается, уже по возрастанию номеров. Каждый из этих шагов требует ~n*log(n) операций.

Быстрая сортировка (quick sort)
Алгоритм основан на разделениии сортируемого массива на части и перестановке элементов таким образом, чтобы элементы в левой части массива не превосходили элементов в правой. На каждом шаге выбирается 'средний' элемент массива, в результате ряда перестановок он занимает свое свое законное место, затем каждая из частей упорядочивается.

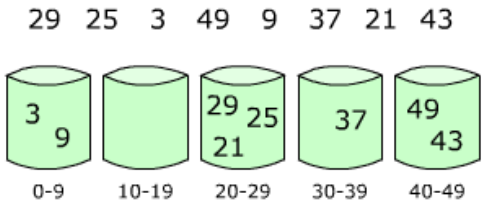
Если 'средние' элементы находится не слишком далеко от своих законных мест, все хорошо - оценка времени выполнения ~n*Log(n), но можно так упорядочить элементы исходного массива, что 'быстрая' сортировка потребует ~n*n операций и, что еще хуже, N фреймов стека. Следующая функция заполняет массив самым неблагоприятным для qSort способом:

```
void Fill(int N; int @Buff)
  if N&gt;0 then
    dec  N;
    Fill(N,@Buff);
    int  M=(N+1)/2;
    Buff[N]=Buff[M];
    Buff[M]=N;
  end
end
```

В более сложных реализациях алгоритма 'средний' элемент выбирается как средний из трех или случайным образом.

Блочная сортировка (Bucket sort)

Способ сортировки, в которой сортируемые элементы делятся на конечное число отдельных блоков (корзин). Каждый блок затем сортируется отдельно, либо рекурсивно тем же методом, либо иным другим. Затем элементы помещаются обратно в массив. Этот тип сортировки может обладать линейным временем исполнения.



Сортировка слиянием (merge sort)

'Divide-and-conquer' - 'разделяй и властвуй' - исходный массив делится пополам, каждая половинка упорядочивается, затем производится слияние. Для сортировки половинок используется та же функция, что и для сортировки всего массива.

Это быстродействующий алгоритм (~n log(n) сравнений и перемещений данных), но для его работы необходим дополнительный массив, длина которого равна половине длины сортируемого массива.

Поразрядная сортировка

Применение поразрядной сортировки имеет одно ограничение: перед началом сортировки необходимо знать

length - максимальное количество разрядов в сортируемых величинах (например, при сортировке слов необходимо знать максимальное количество букв в слове),

range - количество возможных значений одного разряда (при сортировке слов - количество букв в алфавите).

Количество проходов равно числу length. Пример:

Рассмотрим последовательность: 39, 47, 54, 59, 34, 41, 32 (length = 2, range = 10)

1. Создаем пустые списки, количество которых равно числу range.

2. Распределяем исходные числа по этим спискам в зависимости от величины младшего разряда (по возрастанию).

41
32
54, 34
47
59, 39

(Необходимо создать 10 списков, но некоторые из них оказались пустыми)

3. Собираем числа в той последовательности, в которой они находятся после распределения по спискам: 41, 32, 54, 34, 47, 59, 39

4. Повторяем пункты 2 и 3 для всех более старших разрядов поочередно.

Для двузначных чисел мы должны сделать еще один проход. Распределение по спискам будет выглядеть так:

32, 34, 39

41, 47

54, 59

Объединив числа в последовательность, получим отсортированный массив.

Сортировка подсчетом (counting sort)
Неуниверсальный алгоритм, использующий дополнительную память и выполняющий сортировку за время, пропорциональное числу элементов массива. Для его реализации необходимо взаимно однозначно сопоставить каждому возможному значению элемента массива целое неотрицательное число. при котором подсчитывается число одинаковых элементов. Алгоритм выгодно применять, когда в массиве много элементов, но все они достаточно малы. Идея сортировки указана в её названии — нужно подсчитывать число элементов, а затем выстраивать массив. Пусть, к примеру, имеется массив **A** из миллиона натуральных чисел, меньших 100. Тогда можно создать вспомогательный массив **B** из 99 (1..99) элементов, «пробежать» весь исходный массив и вычислять частоту встречаемости каждого элемента — то есть если **A[i]=a**, то **B[a]** следует увеличить на единицу. Затем «пробежать» счетчиком **i** массив **B**, записывая в массив **A** число **i B[i]** раз.

Пусть n — длина исходного массива, k — максимальное число в массиве. Тогда временная сложность алгоритма равна O(n+k), а ёмкостная — O(n+k).