Основы организации данных и алгоритмизация (2012)

Лекция 7. Сортировка

Штанюк А.А.

9 декабря 2012 г.

Понятие сортировки

Понятие сортировки

Сортировка - процесс упорядочивания данных по возрастанию (убыванию) их значений

Простейшие методы имеют оценку сложности $O(N^2)$. Если N невелико, то такие алгоритмы вполне могут считаться эффективными, благодаря простоте реализации. Но с ростом числа элементов будет неизбежно возрастать время сортировки, поэтому при больших N предпочтение нужно отдать более "продвинутым"алгоритмам.

Понятие сортировки

К *простейшим* алгоритмам сортировки можно отнести:

- непрактичную сортировку;
- сортировку вставками;
- сортировку выбором;
- пузырьковую сортировку.

К более *продвинутым* можно отнести алгоритмы:

- быструю сортировку;
- сортировку Шелла;
- пирамидальную сортировку;
- сортировку слиянием;
- сортировку с помощью бинарного дерева.

Непрактичная сортировка

Самой простой для запоминания является непрактичная сортировка, которая имеет сложность $O(N^2)$.

```
void UnratioSort(int A[], int N)
{
   int i,j;

   for(i=0;i<N;i++)
     for(j=0;j<N;j++)
       if(A[i]<A[j])
          swap(&A[i],&A[j]);
}
```

Сортировка может считаться непрактичной из-за лишних операций сравнения и обмена (например, **A[0]** с **A[0]**).

Сортировка вставками

Сортировка вставками удобна для сортировки коротких последовательностей элементов. Именно таким образом обычно сортируют карты: держа в левой руке уже упорядоченные карты и взяв правой рукой очередную карту мы вставляем ее в нужное место, сравнивая с уже имеющимися, двигаясь справа налево.

Все элементы условно разделяются на готовую последовательность $A_1 ... A_{i-1}$ и входную $A_i ... A_N$. На каждом шаге, начиная с $i = 2$ и увеличивая i на 1, берем i-й элемент входной последовательности и вставляем его на нужное место в готовую.

Сортировка вставками

Справа - ещё не упорядоченная последовательность.
Слева - упорядоченная.

```
44 \\ 55 12 42 94 18 06 67
44 55 \\ 12 42 94 18 06 67
12 44 55 \\ 42 94 18 06 67
12 42 44 55 \\ 94 18 06 67
12 42 44 55 94 \\ 18 06 67
12 18 42 44 55 94 \\ 06 67
06 12 18 42 44 55 94 \\ 67
06 12 18 42 44 55 67 94 \\
```

Сортировка вставками

```
void InsertionSort(int A[], int N)
{
    int i,j;
    int temp;

    for(i=1;i<N;i++)
    {
        j=i;
        temp=A[i];
        while(j>0 && temp<A[j-1])
        {
            A[j]=A[j-1];
            j--;
        }
        A[j]=temp;
    }
}
```

Сортировка выбором

Предполагается, что N элементов данных хранятся в массиве A. и по этому массиву выполняется $N-1$ проход. В нулевом проходе выбирается наименьший элемент, который затем меняется местами с $A[0]$. После этого неупорядоченными остаются элементы $A[1]...A[N-1]$. В следующем проходе просматривается эта часть массива и наименьший элемент помещается в $A[1]$. И так до конца.

Сортировка выбором

```
void SelectionSort(int A[], int N)
{
   int i,j,index;
   for(i=0;i<N-1;i++)
   {
      index=i;
      for(j=i+1;j<N;j++)
        if(A[j]<A[index])
           index=j;
      swap(&A[i],&A[index]);
     }
}
```

Пузырьковая сортировка

Для сортировки N-элементного массива A методом пузырька требуется до $N - 1$ проходов. В каждом проходе сравниваются соседние элементы, и если, первый из них больше или равен второму, эти элементы меняются местами.

Можно выделить два варианта реализации алгоритма:

1. избыточный;
2. экономный.

Пузырьковая сортировка (избыточный вариант)

```
void BubbleSort(int A[], int N)
{
   int i,j;
   for(i=0;i<N-1;i++)
      for(j=N-1;j>i;j--)
        if(A[j-1]>A[j])
           swap(&A[j-1],&A[j]);
}
```

В избыточном варианте циклы выполняются независимо от начального расположения элементов, что ведёт к лишним операциям сравнения, если массив уже упорядочен.

Пузырьковая сортировка (экономный вариант)

```
void BubbleSort2(int A[], int N)
{
   int i,j,lastIndex;
   i=N-1;
   while(i>0)
   {
     lastIndex=0;
     for(j=0;j<i;j++)
        if(A[j+1]<A[j])
        {
           swap(&A[j+1],&A[j]);
           lastIndex=j;
        }
        i=lastIndex;
   }
}
```

Сравнение простейших методов сортировки

К положительным качествам простейших методов сортировки относят

- экономное расходование памяти
- простота реализации

Главным недостатком сортировок методом *вставок*, *выбора*, *пузырька* является низкая производительность

Количество операций		
Метод сортировки	*Тип операции*	*Количество*
Selection	сравнение обмен	$N^2/2$ N
Insertion	сравнение обмен	$N^2/4$ $N^2/4$
Bubble	сравнение обмен	$N^2/2$ $N^2/2$

Быстрая сортировка

QuickSort - широко известный алгоритм сортировки, разработанный английским учёным Чарльзом Хоаром. Один из быстрых известных универсальных алгоритмов сортировки массивов, являющийся также, наиболее распространённым.

В стандартную библиотеку C/C++ входит функция qsort.

Краткое описание работы:

- выбрать элемент, называемый опорным.
- сравнить все остальные элементы с опорным, на основании сравнения разбить множество на три — «меньшие опорного», «равные» и «большие», расположить их в порядке меньшие-равные-большие.
- повторить рекурсивно для «меньших» и «больших».

Быстрая сортировка

Алгоритм быстрой сортировки является одним из самых быстрых: среднее время работы близко к $O(NlogN)$. Кроме того, он не требует дополнительной памяти, за исключением расходов на стек вызовов.

Быстрая сортировка использует рекурсию. Рассмотрим работу алгоритма применительно к участку массива $A[p..r]$.

1. Элементы массива A переставляются так, чтобы любой из элементов $A[p], ...A[q]$ был не больше любого из элементов $A[q+1], ..., A[r]$, где q - некоторое число в интервале $p \leq q < r$. Эта операция называется *разделением (partition)*.
2. Процедура сортировки рекурсивно вызывается для массивов $A[p..q]$ и $A[q+1...r]$.

После этого исходный массив $A[p..r]$ отсортирован.

Быстрая сортировка

Реализация:

```
void QuickSort(int A[], int p,
                        int r)
{
    int i,j;
    int x;
    i=p;
    j=r;
    x=A[(i+j)/2];
    do
    {
        while(A[i]<x) i++;
        while(A[j]>x) j--;
        if(i<=j)
        {
            swap(&A[i],&A[j]);
            i++;
            j--;
        }
    }
    while(i<=j);
    if(j>p)
        QuickSort(A,p,j);
    if(i<r)
        QuickSort(A,i,r);
}
```

Быстрая сортировка

Функция **qsort** из стандартной библиотеки (заголовочный файл **stdio.h**).

```
#include <stdlib.h>

void qsort(void *base, size_t nmemb, size_t size,
           int(*compar)(const void *, const void *));
```

- **base** - адрес первого элемента массива.
- **nmemb** - количество элементов
- **size** - размер элемента
- **compar** - функция сравнения двух элементов

Быстрая сортировка

Пример использования **qsort** для сортировки числового массива:

```
#include <stdlib.h>
#include <stdio.h>
int compare(const void *a, const void *b)
{
   if(*(int*)a > *(int*)b)
      return 1;
   else
      return -1;
}
int main()
{
   int arr[10]={34,89,56,21,67,34,78,84,90,3},i;
   qsort(arr,10,sizeof(int),compare);
   for(i=0;i<10;i++)
      printf("%d ",arr[i]);
   return 0;
}
```

Быстрая сортировка

Рассмотрим три случая, определяющие эффективность быстрой сортировки

Лучший случай. Самый лучший случай — если в каждой итерации каждый из подмассивов делился бы на два равных по величине массива. В результате количество сравнений, делаемых быстрой сортировкой, было бы равно значению рекурсивного выражения $C_N = 2C_{N/2} + N$. Это дало бы наименьшее время сортировки.

Среднее. Даёт в среднем $O(NlogN)$ обменов при упорядочении n элементов. В реальности именно такая ситуация обычно имеет место при случайном порядке элементов и выборе опорного элемента из середины массива либо случайно. $2C_{N/2}$ покрывает расходы по сортировке двух полученных подмассивов; N — это стоимость обработки каждого элемента, используя один или другой указатель. Известно также, что примерное значение этого выражения равно $C_N = NlgN$.

Быстрая сортировка

Худший случай. Худшим случаем будет такой, при котором на каждом этапе массив будет разделяться на вырожденный подмассив из одного опорного элемента и на подмассив из всех остальных элементов. Такое может произойти, если в качестве опорного на каждом этапе будет выбран элемент либо наименьший, либо наибольший из всех обрабатываемых. Худший случай даёт $O(N^2)$ обменов. Хуже то, что в таком случае глубина рекурсии при выполнении алгоритма достигнет N, что будет означать n-кратное сохранение адреса возврата и локальных переменных процедуры разделения массивов. Для больших значений n худший случай может привести к исчерпанию памяти во время работы алгоритма.

Быстрая сортировка

Достоинства:

- Один из самых быстродействующих (на практике) из алгоритмов внутренней сортировки общего назначения.
- Прост в реализации.
- Требует лишь $O(logN)$ дополнительной памяти для своей работы.
- Хорошо сочетается с механизмами кэширования и виртуальной памяти.

Быстрая сортировка

Недостатки:

- Сильно деградирует по скорости (до $O(N^2)$) при неудачных выборах опорных элементов, что может случиться при неудачных входных данных. Этого можно избежать, выбирая опорный элемент случайно, а не фиксированным образом.
- Наивная реализация алгоритма может привести к ошибке переполнения стека, так как ей может потребоваться сделать $O(N)$ вложенных рекурсивных вызовов. В улучшенных реализациях, в которых рекурсивный вызов происходит только для сортировки бо´льшей из двух частей массива, глубина рекурсии гарантированно не превысит $O(logN)$.

Сортировка Шелла

Сортировка Шелла относится к числу быстродействующих и является расширением сортировки методом вставок.

При сортировке Шелла сначала сравниваются и сортируются между собой ключи, отстоящие один от другого на некотором расстоянии d. 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 (но она не предпочтительна, так как все равно остается медленной) каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, при сортировке Шелла же это число может быть больше).

Последовательность шагов задается некоторой формулой, например $d_n = 3d_{n-1} + 1$. В этом случае получаем следующие шаги:

d=1,4,13,40,121,364,1093,3280,9841,..

Сортировка Шелла

Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ:

- отсутствие потребности в памяти под стек
- отсутствие деградации при неудачных наборах данных — **qsort** легко деградирует до $O(N^2)$, что хуже, чем худшее гарантированное время для сортировки Шелла

Часто оказывается, что сортировка Шелла есть самый лучший способ сортировки до, примерно, 1000 элементов.

При удачном выборе последовательности шагов можно увеличить скорость сортировки в 3-5 раз!

Сортировка Шелла

Пусть дан список $A = (32, 95, 16, 82, 24, 66, 35, 19, 75, 54, 40, 43, 93, 68)$ и выполняется его сортировка методом Шелла, а в качестве значений d выбраны $5, 3, 1$.

На первом шаге сортируются подсписки A, составленные из всех элементов A, различающихся на 5 позиций, то есть подсписки $A_{5,1} = (32, 66, 40)$, $A_{5,2} = (95, 35, 43)$, $A_{5,3} = (16, 19, 93)$,$A_{5,4} = (82, 75, 68)$, $A_{5,5} = (24, 54)$.

В полученном списке на втором шаге вновь сортируются подсписки из отстоящих на 3 позиции элементов.

Процесс завершается обычной сортировкой вставками получившегося списка.

Исходный массив	32	95	16	82	24	66	35	19	75	54	40	43	93	68	
После сортировки с шагом 5	32	35	16	68	24	40	43	19	75	54	66	95	93	82	6 обменов
После сортировки с шагом 3	32	19	16	43	24	40	54	35	75	68	66	95	93	82	5 обменов
После сортировки с шагом 1	16	19	24	32	35	40	43	54	66	68	75	82	93	95	15 обменов

Сортировка Шелла

```
/* Пример из книги Герберта Шилдта */
void shell(char *items, int count)
{
  register int i, j, gap, k;
  char x, a[5];

  a[0]=9; a[1]=5; a[2]=3; a[3]=2; a[4]=1;

  for(k=0; k < 5; k++) {
    gap = a[k];
    for(i=gap; i < count; ++i) {
      x = items[i];
      for(j=i-gap; (x < items[j]) && (j >= 0); j=j-gap)
        items[j+gap] = items[j];
      items[j+gap] = x;
    }
  }
}
```

Сортировка Шелла

Один из лучших вариантов выбора шагов d предложил Р. Серджвик

$$inc[s] = \begin{cases} 9 * 2^s + 9 * 2^{s/2} + 1, \text{если} s - \text{нечетно} \\ 8 * 2^s + 6 * 2^{(s+1)/2} + 1, \text{если} s - \text{четно} \end{cases}$$

При использовании таких приращений среднее количество операций: $O(N^{7/6})$, в худшем случае - порядка $O(N^{4/3})$.

Сортировка Шелла (реализация)

```
int increment(long inc[], long size) {
  int p1, p2, p3, s;

  p1 = p2 = p3 = 1;
  s = -1;
  do {
    if (++s % 2) {
      inc[s] = 8*p1 - 6*p2 + 1;
    } else {
      inc[s] = 9*p1 - 9*p3 + 1;
      p2 *= 2;
      p3 *= 2;
    }
 p1 *= 2;
  } while(3*inc[s] < size);
  return s > 0 ? --s : 0;
}
```

Сортировка Шелла (реализация)

```
void shellSort(int a[], long size) {
  long inc, i, j, seq[40];
  int s;

  // вычисление последовательности приращений
  s = increment(seq, size);
  while (s >= 0) {
 // сортировка вставками с инкрементами inc[]
 inc = seq[s--];

    for (i = inc; i < size; i++) {
      int temp = a[i];
      for (j = i-inc; (j >= 0) && (a[j] > temp); j -= inc)
        a[j+inc] = a[j];
      a[j+inc] = temp;
    }
  }
}
```

Вопросы для самоконтроля

- Что такое сортировка? Подсказка
- Какова наихудшая оценка сложности для алгоритмов сортировки? Подсказка
- Какие методы сортировки относятся к простейшим? К продвинутым? Подсказка
- В чём суть нерациональной сортировки? Подсказка
- На чем основана сортировка вставками? Подсказка
- Для каких последовательностей элементов она больше всего подходит? Подсказка
- Как реализовать сортировку вставками? Подсказка
- Как работает сортировка выбором? Подсказка
- Как реализуется сортировка выбором? Подсказка
- Как работает пузырьковая сортировка? Подсказка

Вопросы для самоконтроля

- Чем плох избыточный вариант пузырьковой сортировки? Подсказка
- Как реализован экономный вариант пузырьковой сортировки? Подсказка
- По каким критериям можно сравнивать алгоритмы сортировки? Подсказка
- Что такое быстрая сортировка (quick sort)? Подсказка
- На чём основана работа алгоритма быстрой сортировки? Подсказка
- Какой прототип имеет функция qsort? Подсказка
- От чего зависит эффективность быстрой сортировки? Подсказка
- Перечислите достоинства алгоритма быстрой сортировки. Подсказка
- Перечислите недостатки алгоритма быстрой сортировки. Подсказка

Глоссарий I

Библиографический список I

Кормен Т.,Лейзерсон Ч., Ривест Р.
Алгоритмы: построение и анализ
МЦНМО, Москва, 2000

Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ.
2-е изд. — М.: «Вильямс», 2006

Википедия
http://ru.wikipedia.org/wiki/Алгоритм

Википедия
http://ru.wikipedia.org/wiki/Список_алгоритмов

Серджвик Р.
Фундаментальные алгоритмы на C++. Части 1-4
Diasoft,2001

Библиографический список II

Седжвик Р.
Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск
СПб.: ДиаСофтЮП, 2003

Седжвик Р.
Фундаментальные алгоритмы на C. Алгоритмы на графах
СПб.: ДиаСофтЮП, 2003

Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы.
Издательский дом «Вильямс», 2000

Кнут Д.
Искусство программирования, том 1. Основные алгоритмы
3-е изд. — М.: «Вильямс», 2006

Кнут Д.
Искусство программирования, том 2. Получисленные методы
3-е изд. — М.: «Вильямс», 2007

Библиографический список III

Кнут Д.
Искусство программирования, том 3. Сортировка и поиск
2-е изд. — М.: «Вильямс», 2007

Кнут Д.
Искусство программирования, том 4, выпуск 3. Генерация всех сочетаний и разбиений
М.: «Вильямс», 2007

Кнут Д.
Искусство программирования, том 4, выпуск 4. Генерация всех деревьев. История комбинаторной генерации
М.: «Вильямс», 2007

Керниган Б., Ритчи Д.
Язык программирования Си
М.: Финансы и статистика,

Подбельский В.В., Фомин С.С.
Программирование на языке С.
М.: Финансы и статистика, 2003

Библиографический список IV

Подбельский В.В.
Практикум по программированию на языке Си.
М.: Финансы и статистика, 2004

Шилдт Г.
Полный справочник по C.
М.: Вильямс, 2002.

Дейтел Х., Дейтел П.
Как программировать на C.