Чулюков В.А.

МЕТОДЫ РАЗРАБОТКИ ПРОГРАММ (АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ)

ЧАСТЬ 3

СОРТИРОВКИ

Воронеж - 2015

Лекция № 3.

ТЕМА: С**ортировка последовательных файлов**

Основные вопросы, рассматриваемые на лекции:

1. Особенности сортировки последовательных файлов.
2. Прямое слияние.

К сожалению, алгоритмы сортировки, рассмотренные в предыдущих лекциях, неприменимы, если сортируемые данные не помещаются в оперативной памяти, а, например, расположены на внешнем запоминающем устройстве с последовательным доступом, таком, как магнитная лента. В этом случае мы описываем данные как (последовательный) файл, который характеризуется тем, что в каждый момент имеется непосредственный доступ к одному и только одному элементу. Это — строгое ограничение по сравнению с возможностями, которые дает массив, и поэтому здесь приходится применять другие методы сортировки. Основной метод — это сортировка *слиянием*. Слияние означает объединение двух (или более) упорядоченных последовательностей в одну упорядоченную последовательность при помощи циклического выбора элементов, доступных в данный момент. Слияние — намного более простая операция, чем сортировка; она используется в качестве вспомогательной в более сложном процессе последовательной сортировки. Один из методов сортировки слиянием называется *простым слиянием* и состоит в следующем:

1. Последовательность *a* разбивается на две половины *b* и *c*.
2. Последовательности *b* и *c* сливаются при помощи объединения отдельных элементов в упорядоченные пары.
3. Полученной последовательности присваивается имя *a*, и повторяются шаги 1 и 2; на этот раз упорядоченные пары сливаются в упорядоченные четверки.
4. Предыдущие шаги повторяются; четверки сливаются в восьмерки, и весь процесс продолжается до тех пор, пока не будет упорядочена вся последовательность, ведь длины сливаемых последовательностей каждый раз удваиваются.

В качестве примера рассмотрим последовательность

44 55 12 42 94 18 06 67

На первом шаге разбиение дает последовательности

44 55 12 42

94 18 06 67

Слияние отдельных компонент (которые являются упорядоченными последовательностями длины 1) в упорядоченные пары дает

44 94 ' 18 55 ' 06 12 ' 42 67

Повое разбиение пополам и слияние упорядоченных пар дают

06 12 44 94 ' 18 42 55 67

Третье разбиение и слияние приводят, наконец, к нужному результату:

06 12 18 42 44 55 67 94

Операция, которая однократно обрабатывает все множество данных, называется *фазой*, а наименьший подпроцесс, который, повторяясь, образует процесс сортировки, называется *проходом* или *этапом*. В приведенном выше примере сортировка производится за три прохода, каждый проход состоит из фазы разбиения и фазы слияния. Для выполнения сортировки требуются три магнитные ленты, поэтому процесс называется *трехленточным слиянием*.

Собственно говоря, фазы разбиения не относятся к сортировке, поскольку они никак не переставляют элементы; в каком-то смысле они непродуктивны, хотя и составляют половину всех операций переписи. Их можно удалить, объединив фазы разбиения и слияния. Вместо того чтобы сливать элементы в одну последовательность, результат слияния сразу распределяют на две ленты, которые на следующем проходе будут входными. В отличие от двухфазного слияния этот метод называется *однофазным* или *сбалансированным слиянием*. Оно имеет явные преимущества, так как требует вдвое меньше операций переписи, но это достигается ценой использования четвертой ленты.

Разберем программу слияния подробно; предположим сначала, что данные расположены в виде массива, который, однако, можно просматривать только *строго последовательно*.

Вместо двух файлов можно легко использовать один массив, если рассматривать его как последовательность с двумя концами. Вместо того чтобы сливать элементы из двух исходных файлов, мы можем брать их с двух концов массива. Таким образом, общий вид объединенной фазы слияния-разбиения можно изобразить, как показано на рис. 3.1.

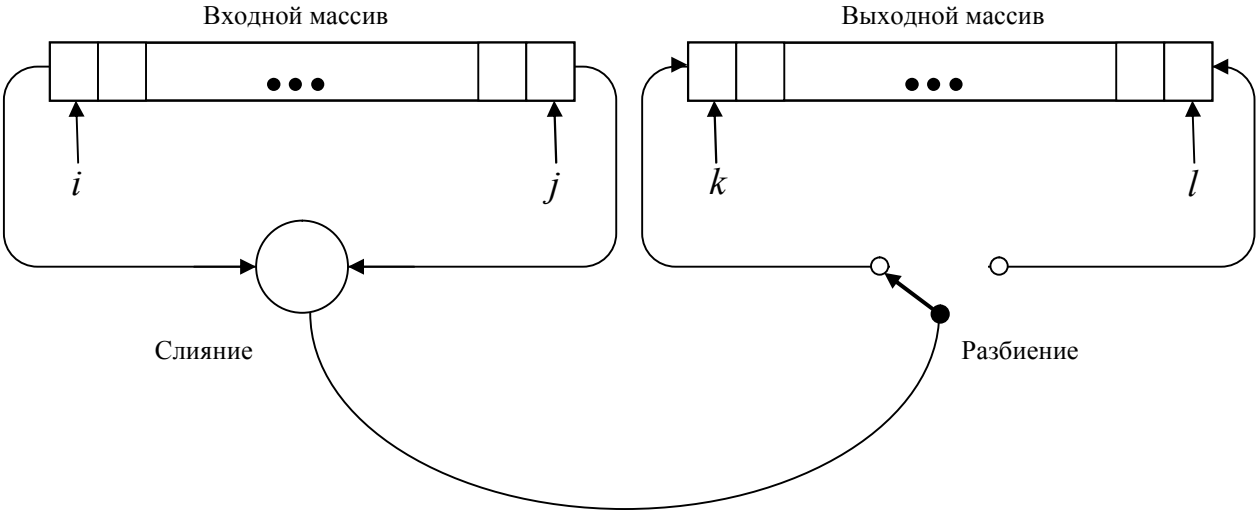

Рис. 3.1. Сортировка двух массивов методом слияния

Направление пересылки сливаемых элементов меняется (переключается) после каждой упорядоченной пары на первом проходе, после каждой упорядоченной

четверки на втором проходе и т. д.; таким образом равномерно заполняются две выходные последовательности, представленные двумя концами одного массива (выходного). После каждого прохода два массива меняются ролями: входной становится выходным и наоборот. Программу можно еще больше упростить, объединив два концептуально различных массива в один массив двойной длины. Итак, данные будут представлены следующим образом:

a: **array**[l. .2**n*] **of** *item*

Пусть индексы *i* и *j* указывают два исходных элемента, тогда как *k* и *l* обозначают два места пересылки (см. рис. 3.1). Исходные данные — это, разумеется, элементы a_1, ..., a_n. Очевидно, что нужна булевская переменная *up* для указания направления пересылки данных; *up = true* будет означать, что на текущем проходе компоненты a_1, ..., a_n будут пересылаться «вверх» — в переменные a_{n+1}, ..., a_{2n}, тогда как *up — false* будет указывать, что a_{n+1}, ..., a_{2n} должны переписываться «вниз» —в a_1, ..., a_n. Значение *up* строго чередуется между двумя последовательными проходами. И наконец, вводится переменная *p* для обозначения длины сливаемых подпоследовательностей (*p*-наборов). Ее начальное значение равно 1, и оно удваивается перед каждым очередным проходом. Для простоты мы будем считать, что *n* — всегда степень двойки. Итак, первая версия программы простого слияния имеет такой вид:

```
procedure mergesort;
    var i,j,k,l: index;
    up: Boolean; p: integer;
    begin up := true; p := 1;
     repeat {инициация индексов}
            if up then
                    begin i := 1 ; j := n; k := n + 1; l := 2*n
           end else
           begin k :=1; l:=n; i := n+1; j := 2*n end;
           «слияние p-наборов последовательностей i и j
           в последовательности k и l»;
            up := - up; p := 2*p
           until p = n
     end
```

На следующем этапе мы уточняем действие, описанное на естественном языке (внутри кавычек). Ясно, что этот проход, обрабатывающий *n* элементов, состоит из последовательных слияний *p*-наборов. После каждого отдельного слияния направление пересылки переключается из нижнего в верхний конец выходного массива или наоборот, чтобы обеспечить одинаковое распределение в обоих направлениях. Если сливаемые элементы посылаются в нижний конец массива, то индексом пересылки служит *k* и *k* увеличивается на 1 после каждой пересылки элемента. Если же они пересылаются в верхний конец массива, то индексом пересылки является *l* и *l* после каждой пересылки уменьшается на 1. Чтобы упростить операцию слияния, мы будем считать, что место пересылки всегда обозначается через *k*, и будем менять местами значения *k* и *l* после слияния каждого *p*-набора, а приращение индекса обозначим через *h*, где *h* равно либо 1, либо -1. Уточнив таким образом «конструкцию», мы получаем

```
h := 1; m := n; {m-номера сливаемых элементов }
 repeat q : = p; r : = p; m:=  m-2*p;
«слияние q элементов из i  и r элементов из j,
индекс засылки есть k с приращением h»;
h := -h;
обмен значениями k и l
until m = 0
```

На следующем этапе уточнения нужно сформулировать саму операцию слияния. Здесь следует учесть, что остаток подпоследовательности, которая остается непустой после слияния, добавляется к выходной последовательности при помощи простого копирования.

```
while (q<>0) and  (r<>0) do
begin {выбор элемента из i или j}
     if a[i] .key < a[j] .key then
          begin «пересылка элемента из i в k,
               увеличение i и k»;  q := q-1
    end else
          begin «пересылка элемента из j в k,
               увеличение j и k» ; r :=r - 1
          end
end;
«копирование остатка последовательности i»;
«копирование остатка последовательности j»
```

После уточнения операций копирования остатков программа будет ясна во всех деталях. Перед тем как записать ее полностью, мы хотим устранить ограничение, в соответствии с которым n должно быть степенью двойки. На какую часть алгоритма это повлияет? Легко убедиться в том, что в более общей ситуации лучше всего использовать прежний метод до тех пор, пока это возможно. В данном случае это означает, что мы продолжаем слияние p-наборов, пока длина остатков входных последовательностей не станет меньше p. Это влияет только на ту часть, где определяются значения q и r — длины последовательностей, которые предстоит слить. Вместо трех операторов

```
q := p;   r:= p;   m:= m - 2 * p
```

используются следующие четыре оператора, и, как может убедиться читатель, здесь эффективно применяется описанная выше стратегия; заметим, что m обозначает общее число элементов в двух входных последовательностях, которые осталось слить:

```
if  m >= p then q := p else q := m;   m := m - q;
if  m >= p then r := p else r := m;   m := m - r;
```

И наконец, чтобы обеспечить окончание работы программы, нужно заменить условие $p = n$, управляющее внешним циклом, на $p >= n$. После этих модификаций мы

можем попытаться описать весь алгоритм в виде законченной программы (см. программу 3.1).

```
procedure mergesort;
 var i,j,k,l,t: index;
    h,m,p,q,r:integer; up:boolean;
 begin up:=true; p:=1;
  repeat h:=1; m:=n;
    if up then
    begin i:=1; j:=n; k:=n+1; l:=2*n;
    end else
    begin k:=1; l:=n; i:=n+1; j:=2*n
    end;
    repeat
      if m>=p then q:=p else q:=m; m:=m-q;
      if m>=p then r:=p else r:=m; m:=m-r;
      while (q<>0) and (r<>0) do
      begin
        if a[i].key < a[j] . key then
        begin a[k]:=a[i]; k:=k+h; i:=i+1; q:=q-1;
        end else
        begin a[k]:=a[j]; k:=k+h; j:=j-1; r:=r-1
        end
      end;
      while r<>0 do
        begin a[k]:=a[j]; k:=k+h; j:=j-1; r:=r-1
        end;
      while q<>0 do
        begin a[k]:=a[i]; k:=k+h; i:=i+1; q:=q-1
        end;
      h:=-h; t:=k; k:=l; l:=t
    until m=0;
    up:= not up; p:=2*p
    until p>=n;
    if  not up then
    for i:=1 to n do a[i]:=a[i+n]
 end;
```

Программа 3.1.

Анализ сортировки слиянием. Поскольку на каждом проходе p удваивается и сортировка заканчивается, как только $p>=n$, она требует $[\log_2 n]$ проходов. По определению при каждом проходе все множество из n элементов копируется ровно один раз. Следовательно, общее число пересылок равно

$$M = n\,[\log_2 n]$$

Число C сравнений по ключу еще меньше, чем M, так как при копировании остатка последовательности сравнения не производятся. Но, поскольку сортировка слиянием обычно применяется при работе с внешними запоминающими устройствами, стоимость операций пересылки часто на несколько порядков превышает стоимость сравнений. Поэтому подробный анализ числа сравнений не представляет особого практического интереса.

Алгоритм сортировки слиянием выдерживает сравнение даже с усовершенствованными методами сортировки, которые обсуждались в предыдущей лекции. Но затраты на управление индексами довольно высоки, кроме того, существенным недостатком является использование памяти размером $2n$ элементов. Поэтому сортировка слиянием редко применяется при работе с массивами, т. е. данными, расположенными в оперативной памяти.