Сортировка
Внешняя сортировка слиянием

Сортировка последовательностей

Простые слияния

Алгоритмы, которые мы рассмотрели ранее, могут применяться только в тех случаях, когда объём сортируемых данных таков, что они полностью помещаются в оперативную память. Но в жизни часто встречаются задачи, когда надо сортировать данные, хранящиеся на внешних устройствах, и объём этих данных таков, что полностью загрузить их в оперативную память нет никакой возможности. В этом случае будем считать, что данные представлены в виде файла, для которого характерно, что в каждый момент времени непосредственно доступен только один элемент. Очевидно, что с таким ограничением нам не удастся использовать ни один из рассмотренных ранее алгоритмов сортировки.

Самый важный метод внешней сортировки[1] - сортировка слиянием. Слиянием будем называть объединение нескольких упорядоченных последовательностей в одну, тоже упорядоченную. Слияние – гораздо более простая операция, чем сортировка, и эту процедуру используют качестве вспомогательной в более сложных процедурах сортировки последовательностей. Первый способ, который мы рассмотрим, называется простой сортировкой слиянием, и состоит он в следующем:

1. $k = 1$
2. Разобьём последовательность a на две половины (b и c).
3. Выполним слияние частей b и c, комбинируя по k элементов из b и c в упорядоченные последовательности длиной $2k$.
4. Назовём получившуюся последовательность a. Увеличим значение k в 2 раза. Повторим шаги $2 - 3$, пока вся последовательность не окажется упорядоченной.

Например, рассмотрим последовательность $a = \{1,100,79,56,205,3,21,65\}$. На шаге 2 мы получим две подпоследовательности: $b = \{1,100,79,56\}$ и $c = \{205,3,21,65\}$. Слияние одиночных элементов в упорядоченные пары даст новую последовательность $a = \{1,205,3,100,21,79,56,65\}$. Снова разбивая последовательность по середине и выполняя слияние упорядоченных пар, получаем: $a = \{1,21,79,205,3,56,65,100\}$. И, наконец, третья операция разделения даст желаемый результат: $a = \{1,3,21,56,65,79,100,205\}$.

Операция, которая требует однократного прохода по всему набору данных, называется фазой, а наименьшая процедура, из повторных вызовов которой состоит сортировка, называется проходом. В рассмотренном примере сортировка состояла из трёх проходов, каждый из которых состоял из фазы разбиения и фазы слияния. Чтобы выполнить сортировку, здесь нужны три ленты, поэтому процедура называется трёхленточным слиянием.

На самом деле фазы разбиения не дают вклада в сортировку в том смысле, что элементы там не переставляются. В этом отношении они непродуктивны, хотя и составляют половину всех операций

[1] Внешней называется сортировка, когда исходные данные хранятся на внешнем носителе.

копирования. Их можно вообще устранить, если объединить фазы разбиения и слияния. Вместо записи в единственную последовательность результат слияния сразу распределяется на две ленты, которые будут служить источником исходных данных для следующего прохода. В отличие от вышеописанной двухфазной сортировки слиянием, такой метод называется методом сбалансированных слияний (или однофазной сортировкой слиянием). Очевидно, что он более эффективен, так как операций копирования нужно вдвое меньше, но плата за это – необходимость использовать четвёртую ленту.

Мы детально разберём процедуру слияния, но сначала будем представлять данные с помощью массивов, только просматривать их будем строго последовательно. Вместо двух последовательностей можно использовать единственный массив, если считать, что оба его конца равноправны. Вместо того чтобы брать элементы для слияния из двух файлов, можно брать элементы с двух концов массива-источника. После слияния элементы отправляются в массив-приёмник с одного или другого конца, причём переключение происходит после каждой упорядоченной пары, получающейся в результате слияния в первом проходе, после каждой упорядоченной четвёрки на втором проходе и т.д., так что будут равномерно заполняться обе последовательности , представленные двумя концами единственного массива-приёмника. После каждого прохода два массива меняются ролями, источник становится приёмником и наоборот.

Дальнейшее упрощение программы получается, если объединить два концептуально разных массива в единственный массив двойного размера. Индексы i, j будут обозначать два элемента из массива-источника, а k, l – две позиции в массиве-приёмнике. Исходные данные – это, конечно, элементы $a_0 \dots a_{N-1}$. Очевидно, нужна переменна up, которая будет управлять направлением потока данных. up будет означать, что в текущем проходе элементы $a_0 \dots a_{N-1}$ пересылаются вверх (в переменные $a_N \dots a_{2N-1}$), а значение $\overline{up}$ будет указывать, что $a_N \dots a_{2N-1}$ пересылаются вниз в $a_0 \dots a_{N-1}$. Значение up переключается перед каждым проходом. Наконец, для обозначения длины сливаемых последовательностей вводится переменная p. Сначала её значение равно 1, а затем оно удваивается перед каждым следующим проходом. Чтобы немного упростить первую версию алгоритма, предположим, что N всегда является степенью двойки. Тогда первый вариант простой сортировки слияниями приобретает следующий вид:

Листинг 1

```
procedure StraightMerge;
var
i,j,k,l,p: integer;
up: boolean;
begin
  up:=true;
  p:=1;
  REPEAT
    инициализировать индексные переменные
    if up then then
    begin
      i:=0;
      j:=N-1;
      k:=N;
      l:=2*N-1;
    end
    else
    begin
      k:=0;
```

```
      l:=n-1;
      i:=N;
      j:=2*N-1;
    end;
    выполнить слияние p-наборов из i- и j-источников в k- и l-приёмники
    up:=not up;
    p:=2*p;
  UNTIL p=n;
end;
```

На следующем шаге разработки мы должны уточнить некоторые инструкции. Очевидно, что проход слияния, обрабатывающий N элементов, сам является серией слияний подпоследовательностей из p элементов (p-наборов). После каждого такого частичного слияния приёмником для подпоследовательности становится попеременно то верхний, то нижний конец массива-приёмника, чтобы обеспечить равномерное распределение в обе принимающие последовательности. Если элементы после слияния направляются в нижний конец массива-приёмника, то индексом приёмником является k, и k увеличивается после каждой пересылки элемента. Если они пересылаются в верхний конец массива-приёмника, то индексом-приёмником является l, и его значение уменьшается после каждой пересылки. Чтобы упростить получающийся программный код для слияния, k всегда будет обозначать индекс-приёмник, значения переменных k и l будут обмениваться после каждого слияния -наборов, а переменная h, принимающая значения $+1/-1$, будет всегда обозначать приращение для k. Эти решения приводят к следующему уточнению:

Листинг 2

```
h:=1;
m:=N; {m - число сливаемых элементов}
REPEAT
  q:=p; r:=p; m:=m-2*p;
  слить q элементов из i-источника с r элементами из j источника
  индекс-приёмник равен k, увеличить k на h
  h:=-h;
  обменять значения k и l
UNTIL m=0;
```

На следующем шаге уточнения нужно конкретизировать операцию слияния. Здесь нужно помнить, что остаток той последовательности, которая осталась непустой после слияния, должен быть присоединён к выходной последовательности простым копированием.

Листинг 3

```
while (q>0) and (r>0) do
begin
  if a[i]<a[j] then
  begin
    переслать элемент из i-источника в k-приёмник
    продвинуть i и k
    q:=q-1;
  end
  else
  begin
    переслать элемент из j-источника в k-приёмник
    продвинуть j и k
    r:=r-1;
  end;
end;
```

```
скопировать остаток i-последовательности
скопировать остаток j-последовательности
```

Уточнение операций копирования остатков даст практически полную программу. Прежде чем выписать её, избавимся от ограничения, что N является степенью двойки. Нетрудно понять, что справиться с такой более общей ситуацией проще всего, если как можно дольше действовать старым способом. В данном примере это означает, что нужно продолжать сливать -наборы до тех пор, пока остатки последовательностей-источниковне станут короче p. Это повлияет только на операторы, в которых устанавливаются значения длины сливаемых последовательностей q и r:

Листинг 4

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

Нужно заменить на приведённые ниже операторы (заметим, что m обозначает полное число элементов в двух последовательностях-источниках, которые ещё предстоит слить):

Листинг 5

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

Кроме того, чтобы обеспечить завершение программы, нужно заменить условие $p = N$, которое управляет внешним циклом, на $p \geq N$. После этих изменений весь алгоритм можно выразить в виде процедуры, работающей с глобальным массивом из $2N$ элементов:

Листинг 6

```
procedure StraightMerge;
var
i,j,k,l,t: integer;
h,m,p,q,r: integer;
up: boolean;
begin
  up:=true;
  p:=1;
  REPEAT
    h:=1;
    m:=N;
    if up then
    begin
      i:=0;
      j:=N-1;
      k:=N;
      L:=2*N-1;
    end
    else
    begin
      k:=0;
      L:=N-1;
      i:=N;
```

```
      j:=2*N-1;
    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]<a[j] 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:=0 to N-1 do
      a[i]:=a[i+N];
end;
```

Анализ простой сортировки слияниями

Поскольку p удваивается на каждом проходе, а сортировка прекращается, как только $p > N$, то будет выполнено $\log(N)$ проходов. По определению, на каждом проходе копируется ровно N элементов, а значит общее число пересылок равно $N * \log(N)$. Число сравнений ключей даже меньше, так как при копировании остатков не требуется никаких сравнений. Однако, поскольку сортировка слиянием обычно используется при работе с внешними устройствами хранения данных, вычислительные затраты на пересылки нередко превосходят затраты на сравнения на несколько порядков, поэтому анализ числа сравнений не имеет практического интереса.

Видно, что сортировка слияниями ведёт себя неплохо даже в сравнении с эффективными методами сортировки, но основной её недостаток – необходимость иметь достаточно памяти для хранения $2N$ элементов. По этой причине сортировку слияниями редко применяют для данных, размещённых в ОЗУ. Если говорить о реальном поведении сортировки слияниями, то она ведёт себя лучше пирамидальной, но хуже быстрой сортировки.