

С/к Введение в современное программирование (v.5.5)

Физфак МГУ. 2006/7 уч. год.

Лекция 23

Сортировки-2 (списки)

Упорядоченность -- фундаментальное свойство

Отличия списков от массивов:

ориентированность на последовательную обработку
затратно прыгать в середину
алгоритмы хорошо масштабируются

дополнительная цель (очень важная):

чтобы те же алгоритмы -- для данных на диске, в и-нете.

наиболее естественно -- создавать упорядоченные "банчи" (bunch) по мере
поступления новых элементов, и по мере накопления банчей выполнять их
слияние.

Но сначала простой случай.

Сортировка вставками для списков

Для коротких списков и для создания первичных банчей в сортировке
слияниями.

Интерфейс приведен к виду, удобному для дальнейшего:

MODULE Kurs2006InsSort;

IMPORT Log := StdLog, Math, In := FVTSysIn;

TYPE

Element = POINTER TO RECORD

next: Element;

n: INTEGER

END;

List = POINTER TO RECORD

h: Element;

END;

Sorter = POINTER TO RECORD (* сортировщик *)

h: Element

END;

PROCEDURE NewSorter (): Sorter;

VAR res: Sorter;

BEGIN

NEW(res); res.h := NIL; RETURN res

END NewSorter;

```

51  PROCEDURE ( s: Sorter ) Put ( e: Element ), NEW;
52      VAR cur, prec: Element;
53  BEGIN
54      ASSERT( ( e # NIL ) & ( e.next = NIL ), 20 );
55      IF s.h = NIL THEN (* граничный случай, особая обработка *)
56          s.h := e
57      ELSE
58          prec := NIL; cur := s.h;
59          WHILE ( cur # NIL ) & ~( e.n <= cur.n ) DO
60              prec := cur; cur := cur.next
61          END;
62          IF cur # NIL THEN
63              IF prec = NIL THEN (* вставка в начало *)
64                  e.next := s.h;
65                  s.h := e
66              ELSE
67                  e.next := prec.next;
68                  prec.next := e
69              END
70          ELSE
71              prec.next := e
72          END
73      END
74  END Put;

```

```

75  PROCEDURE ( s: Sorter ) Eject (): List, NEW;
76      VAR res: List;
77  BEGIN
78      NEW( res );
79      res.h := s.h; s.h := NIL;
80      RETURN res
81  END Eject;

```

```

82  PROCEDURE Do*;
83      VAR s: Sorter; n: INTEGER; e: Element; l: List;
84  BEGIN
85      s := NewSorter();
86      In.Open; In.Int( n );
87      WHILE In.Done DO Log.Int( n );
88          NEW( e ); e.n := n;
89          s.Put( e );
90          In.Int( n )
91      END;
92      l := s.Eject();
93      Log.Ln;
94
95      e := l.h;
96      WHILE e # NIL DO
97          Log.Int( e.n ); e := e.next
98      END;
99      Log.Ln
100  END Do;
101  END ①Kurs2006InsSort.Do 9 4 3 9 9 -23 2435 5 11 -4 0①
102

```

103 **Упр** Ввести поле len: INTEGER; и функцию для списка Length(): INTEGER;

104

105 Сортировка больших списков

106

107 Сортировку больших списков можно было бы выстроить по подобию быстрой
108 сортировки -- но брать опорный элемент из середины списка неестественно --
109 любые "пустые" хождения по списку не хороши. А если последовательность
110 хранится на диске? на другом узле кластера? Нужен немножко другой подход:
111 идти не от больших сегментов к малым.

112

113 Получаем элементы по одному из некоего источника.

114 Получили несколько -- отсортировали.

115 Сохранили подпоследовательность (bunch -- банч).

116 И т.д., пока в источнике не закончатся элементы.

117 Нужно соединить отсортированные банчи в одну последовательность. Можно,
118 например, попарно объединять банчи; понадобится несколько итераций.

119 Не обязательно совсем после. Могли бы отвлекаться на эту работу и в процессе
120 получения элементов из источника.

121 В любом случае -- **слияние** двух уже отсортированных последовательностей.
122 (Больше -- обсудим отдельно.)

123

124 **NB** Последовательность из одного элемента -- автоматически отсортирована.

125 Поэтому теоретически достаточно знать алгоритм слияния и уметь

126 организовывать сортировку из него.

127

128

129 Фундаментальный алгоритм: слияние 130 последовательностей (списков)

131

132 Два упорядоченных списка. Превратить в один упорядоченный.

133 **NB** Мы говорили, что 98% всех циклов -- полный просмотр и схема поиска (или
134 их незначит. вариации, где мы "по пути" что-то еще делаем).

135 Остальные 2% -- это как раз слияние. Могут быть и другие циклы (см. книжки
136 Э.Дейкстры и Д.Гриса). Но очень редко. Обычно можно изменить алгоритмы и
137 свести всё к этим трем типам.

138 Когда удобней выполнять сливание в промежутках между получением новых
139 элементов:

140 1) медленный доступ к очередному элементу -- согласовать размер банча с
141 буферизацией источника

142 2) алгебраическая сортировка -- кол-во элементов часто сильно уменьшается.

143 Разумно обеспечить как можно больше сокращений в памяти, прежде чем
144 отправить новый банч на хранение.

145

146 TYPE

147 Element = POINTER TO RECORD

148 next: Element;

149 n: INTEGER;

150 END;

151

152 PROCEDURE Merge (e0, e1: Element; OUT first: Element);

153 VAR t, last: Element;

154

155

PROCEDURE Append (t: Element);

156 BEGIN

157 IF first = NIL THEN

158 first := t

159 ELSE

160 last.next := t

161 END;

162 last := t

163 END Append;

164

165 BEGIN

166 (* основной цикл: *)

167 WHILE (e0 # NIL) & (e1 # NIL) DO

168 IF e0.n < e1.n THEN

169 t := e0; e0 := e0.next;

170 ELSE

171 t := e1; e1 := e1.next;

172 END;

173 Append(t)

174 END;

175

176 (* завершение: *)

177 WHILE e0 # NIL DO

178 t := e0; e0 := e0.next;

179 Append(t)

180 END;

181 WHILE e1 # NIL DO

182 t := e1; e1 := e1.next;

183 Append(t)

184 END;

185 ASSERT((e0 = NIL) & (e1 = NIL), 60);

186 ASSERT((last = NIL) OR (last.next = NIL), 61);

187 END Merge;

188

189 Можно обобщить на слияние >2 последовательностей -- их тогда нужно
190 упорядочить по первому элементу;

191 отправлять в список-результат младший элемент младшей

192 последовательности;

193 после выполнении e := e.next для младшей последовательности

194 "передвинуть" ее на правильное место в последовательности

195 последовательностей, чтобы сохранить их упорядоченность по первому

196 элементу.

197

198 **NB*** Можно использовать более сложные алгоритмы для поддержки

199 упорядоченности последовательностей -- например, пирамиду (~каскад

200 бинарных слияний). Но это гораздо сложнее, причем имеет смысл только для

201 большого числа последовательностей, а в таких случаях пирамида имеет

202 тенденцию вырождаться в линейную структуру. Поддержка сбалансированности

203 пирамиды делает алгоритмы еще сложнее.

204 Каскад бинарных слияний удобно реализовать через конкретную реализацию

205 некоего абстрактного Source'a (sr.Next(): Element), являющийся "сливателем"

206 двух других Source'ов.

```

207 <На практике оказалось оптимальным отказаться от всяких пирамид в пользу
208 простой линейной структуры и слияния небольшого числа
209 последовательностей.>
210
211 Вот модуль для сотрировки бинарными слияниями, приспособленный для
212 мощного развития:
213
214 MODULE Kurs2006MergeSort;
215   IMPORT Log := StdLog, In := FVTsysIn, Math;
216
217   TYPE
218     List = POINTER TO RECORD (* при желании можно абстрагировать *)
219       h: Element; (* head -- голова списка *)
220       len: INTEGER; (* длина списка *)
221     END;
222
223     Element = POINTER TO RECORD (* при желании можно абстрагировать *)
224       next: Element;
225       n: INTEGER;
226     END;
227
228     Source = POINTER TO ABSTRACT RECORD END;
229     StdSource = POINTER TO RECORD ( Source )
230       h: Element;
231     END;
232
233     Writer = POINTER TO ABSTRACT RECORD END;
234     StdWriter = POINTER TO RECORD ( Writer )
235       first, last: Element;
236       len: INTEGER;
237     END;
238
239     Sorter = POINTER TO RECORD ( Writer )
240       wr: Writer; (* для создания первичных банчей длины 1; можно
241 подставлять сортировщик вставками *)
242       n: INTEGER; (* кол-во эл-тов, положенных в wr *)
243       threshold: INTEGER; (* размер первичных банчей *)
244       bunches: ARRAY 32 OF List; (* i-й банч имеет длину либо 0, либо 2^i *)
245       aux: Writer; (* для слияния пары банчей -- именно подходящей
246 подстановкой aux можно все превратить в алгебраическую сортировку! либо
247 обеспечить хранение результатов на диске ... *)
248     END;
249
250     VAR nil: Element;
251
252     PROCEDURE ( wr: Writer ) Put ( e: Element ), NEW, ABSTRACT;
253     PROCEDURE ( wr: Writer ) Eject (): List, NEW, ABSTRACT; (* список,
254 представленный первым элементом *)
255
256     PROCEDURE ( wr: StdWriter ) Init, NEW;
257     BEGIN wr.first := nil; wr.last := nil; wr.len := 0;
258     END Init;
259

```

```

260 PROCEDURE NewStdWriter (): Writer;
261   VAR res: StdWriter;
262 BEGIN
263   NEW( res ); res.Init; RETURN res
264 END NewStdWriter;
265
266 PROCEDURE ( wr: StdWriter ) Put ( e: Element );
267 BEGIN
268   ASSERT( ( wr.first = nil ) = ( wr.last = nil ), 20 );
269   ASSERT( ( e # NIL ) & ( e # nil ) & ( e.next = NIL ), 21 );
270   IF wr.first = nil THEN
271     wr.first := e;
272     wr.last := e;
273   ELSE
274     wr.last.next := e;
275     wr.last := e;
276   END;
277   INC( wr.len );
278 END Put;
279
280 PROCEDURE ( wr: StdWriter ) Eject (): List;
281   VAR res: List;
282 BEGIN
283   ASSERT( ( wr.first = nil ) = ( wr.last = nil ), 20 );
284   IF wr.last # nil THEN
285     wr.last.next := nil
286   END;
287   NEW( res ); res.h := wr.first; res.len := wr.len;
288   wr.Init;
289   ASSERT( ( res.h = NIL ) OR ( res.h.next # NIL ), 60 );
290   RETURN res
291 END Eject;
292
293 PROCEDURE ( l: List ) Length (): INTEGER, NEW;
294 BEGIN RETURN l.len
295 END Length;
296
297 PROCEDURE ( l: List ) NewSource (): Source, NEW;
298   VAR res: StdSource;
299 BEGIN
300   NEW( res ); res.h := l.h;
301   l.h := nil; l.len := 0;
302   RETURN res
303 END NewSource;
304
305

```

```

306   PROCEDURE ( sr: Source ) Next (): Element, NEW, ABSTRACT; (* res = NIL =
307   конец послед-сти *)
308
309   PROCEDURE ( sr: StdSource ) Next (): Element;
310     VAR res: Element;
311   BEGIN
312     IF sr.h = nil THEN
313       res := NIL;
314     ELSE
315       res := sr.h; sr.h := res.next; res.next := NIL
316     END;
317     RETURN res
318   END Next;
319
320
321   PROCEDURE Merge ( s0, s1: Source; wr: Writer );
322     VAR e0, e1, t: Element;
323   BEGIN
324     e0 := s0.Next(); e1 := s1.Next();
325
326     (* основной цикл: *)
327     WHILE ( e0 # NIL ) & ( e1 # NIL ) DO
328       IF e0.n < e1.n THEN
329         t := e0; e0 := s0.Next();
330       ELSE
331         t := e1; e1 := s1.Next();
332       END;
333       wr.Put( t );
334     END;
335
336     (* завершение: *)
337     WHILE e0 # NIL DO
338       t := e0; e0 := s0.Next();
339       wr.Put( t );
340     END;
341     WHILE e1 # NIL DO
342       t := e1; e1 := s1.Next();
343       wr.Put( t );
344     END;
345   END Merge;
346
347   (* реализуем сортировку не в виде процедуры, а сразу в виде объекта-
348   сортировщика -- ведь с самого начала подразумевали последовательное
349   добавление элементов по одному: *)
350

```

```

351   PROCEDURE NewSorter (): Sorter; (* здесь можно задавать сортировщик
352   первичных банчей и т.п. *)
353     VAR res: Sorter; i: INTEGER;
354   BEGIN
355     NEW( res );
356     res.wr := NewStdWriter(); res.threshold := 1;
357     res.n := 0;
358     FOR i := 0 TO LEN( res.bunches ) - 1 DO
359       res.bunches[ i ] := res.wr.Eject();
360       ASSERT( res.bunches[ i ].Length() = 0, 100 );
361     END;
362     res.aux := NewStdWriter();
363     RETURN res
364   END NewSorter;
365
366   PROCEDURE ( s: Sorter ) Put ( e: Element );
367     VAR bunch: List; i: INTEGER; s0, s1: Source;
368   BEGIN
369     ASSERT( ( e # NIL ) & ( e.next = NIL ), 30 );
370     s.wr.Put( e ); INC( s.n );
371
372     (* обработка накопленного банча: *)
373     IF s.n = s.threshold THEN
374       bunch := s.wr.Eject(); s.n := 0; (* готовы делать новые банчи *)
375       ASSERT( bunch.Length() # 0, 105 );
376       (* bunch -- новый банч, надо его "вливать" в имеющуюся структуру: *)
377
378       (* ищем для него пустую ячейку, по пути сливаем с ним всё, что
379       проходим: *)
380       i := 0;
381       WHILE ( i < LEN( s.bunches ) ) & ~( s.bunches[ i ].Length() = 0 ) DO
382         ASSERT( bunch.Length() # 0, 103 );
383
384         s0 := bunch.NewSource(); ASSERT( bunch.Length() = 0, 100 );
385         s1 := s.bunches[ i ].NewSource(); ASSERT( s.bunches[ i ].Length()
386         = 0, 101 );
387         Merge( s0, s1, s.aux );
388         bunch := s.aux.Eject(); (* aux готов к повторному использованию *)
389
390         INC( i )
391       END;
392       IF i < LEN( s.bunches ) THEN
393         ASSERT( bunch.Length() # 0, 104 );
394         s.bunches[ i ] := bunch;
395       ELSE
396         HALT( 126 ); (* на случай будущих переделок на 64-битную
397       машину *)
398       END;
399     END;
400   END Put;
401

```

```

402 PROCEDURE ( s: Sorter ) Eject (): List;
403   VAR bunch: List; i: INTEGER; s0, s1: Source;
404 BEGIN
405   (* сначала нужно собрать все банчи -- полный проход по массиву: *)
406
407   bunch := s.wr.Eject(); s.n := 0;
408   i := 0;
409   WHILE i < LEN( s.bunches ) DO
410     IF s.bunches[ i ].Length() # 0 THEN
411       (* сей фрагмент повторяется: *)
412       s0 := bunch.NewSource(); ASSERT( bunch.Length() = 0, 100 );
413       s1 := s.bunches[ i ].NewSource(); ASSERT( s.bunches[ i ].Length()
414 = 0, 101 );
415       Merge( s0, s1, s.aux );
416       bunch := s.aux.Eject();
417     ELSE
418       (* нечего делать *)
419     END;
420     INC( i )
421   END;
422   RETURN bunch
423 END Eject;
424
425
426 PROCEDURE Do*;
427   VAR e: Element; s: Sorter; checksum, n: INTEGER; l: List; sr: Source;
428 BEGIN
429   Log.String('start:'); Log.Ln;
430
431   s := NewSorter();
432   In.Open; ASSERT( In.Done ); In.Int( n ); checksum := 0;
433   WHILE In.Done DO Log.Int( n ); INC( checksum, n );
434     NEW( e ); e.n := n;
435     s.Put( e );
436     In.Int( n )
437   END;
438   l := s.Eject();
439   Log.Ln;
440
441   Log.String('sorted:'); Log.Ln;
442   sr := l.NewSource(); e := sr.Next(); ASSERT( e # NIL );
443   WHILE e # NIL DO Log.Int( e.n ); DEC( checksum, e.n );
444     e := sr.Next()
445   END;
446   Log.Ln; ASSERT( checksum = 0 );
447   Log.String('checksum (должна быть 0):'); Log.Int( checksum ); Log.Ln;
448 END Do;
449
450 BEGIN NEW( nil ); nil.next := nil;
451 END !Kurs2006MergeSort.Do 9 0 0 -1 8 7 4 9 2345 -4 -4 -4 2 -8 124 -34 !
452
453 compiling "Kurs2006MergeSort" 1932 4
454 Kurs2006MergeSort unloaded

```

```

455 start:
456 9 0 0 -1 8 7 4 9 2345 -4 -4 -4 2 -8 124 -34
457 sorted:
458 -34 -8 -4 -4 -4 -1 0 0 2 4 7 8 9 9 124 2345
459 checksum (должна быть 0): 0
460
461 NB Проверки e0.n < e1.n могли бы выглядеть как e0.LessThan( e1 ) -- тогда
462 процедура стала бы весьма универсальной.
463
464 Упр Сделать еще один конкретный Source, но осуществляющий 2/4/n-кратное
465 слияние задаваемых в фабричной процедуре Source'ов.
466
467 Упр Модифицировать для алгебраической сортировки (приведение подобных).
468 Пусть Element содержит еще поле c: INTEGER -- "коэффициент". Пусть
469 элементы e0, e1 с равными n должны заменяться одним новым с тем же n, но с
470 равным e0.c + e1.c. Окончательная последовательность не должна содержать
471 элементов с нулевым c.
472
473 а) модификацией алгоритма слияния
474 б) введением возможности задавать соотв. Source и Writer, и предусмотреть
475 возможность задания таких конкретных Source и Writer, которые служили бы
476 "фильтром" для первоначальных таким образом, что модифицированный Source
477 отфильтровывает нулевые элементы, а модифицированный Writer распознает
478 подряд идущие элементы с одинаковым n и осуществляет сложение
479 "коэффициентов".
480
481
482
483 Упр Обдумать реализацию модуля, предоставляющего средства работы с
484 полиномами двух переменных (скажем, x, y) с коэффициентами типа REAL.
485 Реализовать сложение, вычитание, умножение, дифференцирование по x и y,
486 вычисление значения в заданной точке, интегрирование по x, y.
487
488
489
490 Представим себе систему для работы с последовательностями данных (эксп.
491 события; алгебраич. выражения).
492 Основной модуль -- содержит интерфейсные ABSTRACT-типы Element, Sequence,
493 Source, Writer, Sorter ...
494 а также предоставляет (через фабричный объект dir: Directory) разумную
495 реализацию (для последовательностей, реализованных простыми списками в
496 памяти) + "строительные блоки" (процедуры слияний и т.п.), реализованные
497 полностью в терминах ABSTRACT-интерфейсов.
498
499 В отдельном модуле модуле -- можно дать более навороченные реализации
500 вместе с соотв. фабричным объектом. При этом достаточно реализовать
501 ввод/вывод (на диск, в и-нет), т.к. "сложные" алгоритмы сортировок (слияния)
502 совершенно не будут зависеть от того, откуда приходят и куда уходят данные.

```