Пирамидальная сортировка

Слёлин А.В.

18 мая, 2024 г.

Описание алгоритма

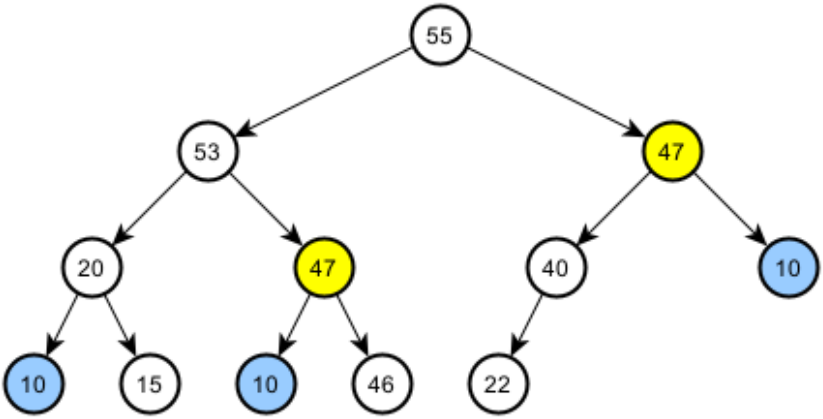
Пусть дан массив $A[0..n-1]$, такой что $n = \text{length}[A]$, причём индексация начинается с нуля для облегчения понимания кода.

Рассмотрим случай когда нам нужно отсортировать массив A в неубывающую последовательность; случай, когда нужно отсортировать в невозрастающую последовательность аналогичен.

Для того, чтобы понять пирамидальную сортировку необходимо хорошо знать такую структуру данных как пирамида.

Пирамида - это структура данных, представляющая собой объект массив, который можно рассматривать как почти полное бинарное дерево.

Каждый узел этого дерева соответствует определённому элементу массива. На всех уровнях, кроме, может быть, последнего, дерево полностью заполнено. В корне дерева находится элемент $A[0]$, а дальше оно строится по следующему принципу: если какому-то узлу соответствует индекс i, то индекс левого дочернего узла - $2i + 1$, а индекс правого дочернего узла - $2i + 2$.



Различают два вида бинарных пирамид: неубывающие и невозрастающие. В пирамидах обоих видов значения, расположенные в узлах, удовлетворяют свойству пирамиды, являющемуся отличительной чертой пирамиды того или иного типа.

Свойство невозрастающих пирамид заключается в том, что каждый отличный от корневого значения узел не превышает значение родительского по отношению к нему узла. Таким образом, в невозрастающей пирамиде самый боль-

шой элемент находится в корне дерева, а значения узлов поддерева, берущего начало в каком-то элементе, не превышает значения самого этого элемента.

Принцип организации неубывающей пирамиды прямо противоположный: наименьший элемент такой пирамиды находится в её корне.

В алгоритме пирамидальной сортировки используется невозрастающая пирамида. Неубывающая пирамида используется в приоритеных очередях.

Базовые процедуры для невозрастающих пирамид (для неубывающих пирамид процедуры аналогичны):

1) Процедура MAX_HEAPIFY выполняется в течение времени $O(\lg n)$ и служит для поддержки свойтва невозрастания пирамиды.

2) Время выполнения процедуры BUILD_MAX_HEAP увеличивается с ростом количества элементов линейно. Эта процедура предназначена для создания невозрастающей пирамиды из неупорядоченного массива.

3) Процедура HEAPSORT выполняется в течение времени $O(n \lg n)$ и сортирует массив без привлечения дополнительной памяти.

Разберём функцию MAX_HEAPIFY.

На вход подаётся массив A и индекс i этого массива, а также размер пирамиды (размер пирамиды может отличаться от размера массива, т.е. мы просто не учитываем информацию из некоторых ячеек массива, под которую всё равно выделена память). При вызове процедуры MAX_HEAPIFY предпологатеся, что бинарные деревья, корнями которого являются элементы $2i + 1$ и $2i + 2$, являются невозрастающими пирамидам, но сам элемент $A[i]$ может быть меньше своих дочерних элементов, нарушая тем самым свойство невозрастающей пирамиды. Функция MAX_HEAPIFY опускает значение элемента $A[i]$ вниз по пирамиде до тех пор, пока поддерево с корнем, отвечающем индексу i, не станет невозрастающей пирамидой. Для этого из двух чисел $A[2i + 1]$, $A[2i + 2]$ мы выбираем наибольшее и меняем его с $A[i]$, далее вызывем процедуру уже для поменявшегося элемента, пока не спустимся до листа.

Рассмотрим функцию BUILD_MAX_HEAP.

С помощью процедуры MAX_HEAPIFY можно преобразовать массив $A[0..n-1]$ в невозрастающую пирамиду в направлении снизу вверх. Мы начинаем «проталкивать» элементы вниз до образования пирамиды необходимого типа. Мы начинаем не с последнего элемента, а с того, у которого индекс $i = \lfloor n/2 \rfloor$, так как не имеет смысла вызывать процедуру MAX_HEAPIFY для листьев, так как у них нет детей. Так как все дети находятся на последне уровне, они занимают ровно половину от всех элементов.

Идём мы в обратном порядке потому, что это обеспечивает корректность работы алгоритма.

Рассмотрим функцию HEAPSORT.

Работа алгоритма пирамидальной сортировки начинается с вызова BUILD_MAX_HEAP, с помощью которой из входного массива $A[0..n-1]$ создаётся невозрастающая пирамида.

Мы начинаем перебирать все элементы, кроме первого, с конца. Текущий элемент обмениваем с первым. Так как мы идём с конца, то в корне будет наибольшее число, теперь уменьшим размер пирамиды на один. Просеим элемент, который стоит первый. Теперь на вершине снова максимум из пирамиды, но без того элемента, который мы отрезали от пирамиды. Таким образом, придется сделать $n-1$ итераций, чтобы отсортировать пирамиду.

Пример

Пусть массив $A = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14]$. Сначала выполним функцию BUILD_HEAP, где i - означает индекс элемента, который мы будем «просеивать»(выделено красным цветом), синим выделен правый дочерний элемент, а оранжевым левый дочерний элемент. Зелёным цветов выделены изменения в массиве A. Нумерация в итерации означает номер вызова HEAPIFY (при любых изменениях она рекурсивно вызывается вновь).

i = 5:

1)

$$A_{in} = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14];$$

$$A_{out} = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14];$$

i = 4:

1)

$$A_{in} = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14];$$

$$A = [8, 3, 12, 16, 14, 4, 1, 9, 7, 10];$$

2)

$$A = [8, 3, 12, 16, 14, 4, 1, 9, 7, 10];$$

$$A_{out} = [8, 3, 12, 16, 14, 4, 1, 9, 7, 10];$$

i = 3:

1)

$$A_{in} = [8, 3, 12, 16, 14, 4, 1, 9, 7, 10];$$

$$A_{out} = [8, 3, 12, 16, 14, 4, 1, 9, 7, 10];$$

i = 2:

1)

$$A_{in} = [8, 3, 12, 16, 14, 4, 1, 9, 7, 10];$$

$$A_{out} = [8, 3, 12, 16, 14, 4, 1, 9, 7, 10];$$

i = 1:

1)

$$A_{in} = [8, 3, 12, 16, 14, 4, 1, 9, 7, 10];$$

$$A = [8, 16, 12, 3, 14, 4, 1, 9, 7, 10];$$

2)

$$A = [8, 16, 12, 3, 14, 4, 1, 9, 7, 10];$$

$$A = [8, 16, 12, 9, 14, 4, 1, 3, 7, 10];$$

3)

$$A = [8, 16, 12, 9, 14, 4, 1, 3, 7, 10];$$

$$A_{out} = [8, 16, 12, 9, 14, 4, 1, 3, 7, 10];$$

i = 0:

1)

$$A_{in} = [8, 16, 12, 9, 14, 4, 1, 3, 7, 10];$$

$$A = [16, 8, 12, 9, 14, 4, 1, 3, 7, 10];$$

2)

$$A = [16, 8, 12, 9, 14, 4, 1, 3, 7, 10];$$

$$A = [16, 14, 12, 9, 8, 4, 1, 3, 7, 10];$$

3)

$$A = [16, 14, 12, 9, 8, 4, 1, 3, 7, 10];$$

$$A = [16, 14, 12, 9, 10, 4, 1, 3, 7, 8];$$

4)

$$A = [16, 14, 12, 9, 10, 4, 1, 3, 7, 8];$$

$$A_{out} = [16, 14, 12, 9, 10, 4, 1, 3, 7, 8].$$

Теперь из массив A мы сделали невозрастающую пирамиду *heap*. Теперь начинаем запускать HEAPSORT для полученного A, где i - означает номер итерации, *heap* означает текущую пирамиду, красным выделен первый, а синим последний элементы (их мы будем переставлять). Зелёным обозначены любые изменения как в пирамиде, так и в массиве. Цифры в итерациях также обозначают номер вызова функции HEAPIFY для просеивания (обозначения всё те же). Жёлтым обозначены уже отсортированные элементы, которые не входят в пирамиду.

i = 9:

$$heap_{in} = [16, 14, 12, 9, 10, 4, 1, 3, 7, 8];$$

$$heap = [8, 14, 12, 9, 10, 4, 1, 3, 7, 16];$$

$$A = [8, 14, 12, 9, 10, 4, 1, 3, 7, 16];$$

1)

$$heap = [8, 14, 12, 9, 10, 4, 1, 3, 7];$$

$$heap = [14, 8, 12, 9, 10, 4, 1, 3, 7];$$

$$A = [14, 8, 12, 9, 10, 4, 1, 3, 7, 16];$$

2)

$$heap = [14, 8, 12, 9, 10, 4, 1, 3, 7];$$

$$heap = [14, 10, 12, 9, 8, 4, 1, 3, 7];$$

$$A = [14, 10, 12, 9, 8, 4, 1, 3, 7, 16];$$

3)

$$heap = [14, 10, 12, 9, 8, 4, 1, 3, 7];$$

$$heap_{out} = [14, 10, 12, 9, 8, 4, 1, 3, 7];$$

$$A = [14, 10, 12, 9, 8, 4, 1, 3, 7, 16];$$

i = 8:

$$heap_{in} = [14, 10, 12, 9, 8, 4, 1, 3, 7];$$

$$heap = [7, 10, 12, 9, 8, 4, 1, 3, 14];$$

$$A = [7, 10, 12, 9, 8, 4, 1, 3, 14, 16];$$

1)

$$heap = [7, 10, 12, 9, 8, 4, 1, 3];$$

$$heap = [12, 10, 7, 9, 8, 4, 1, 3];$$

$$A = [12, 10, 7, 9, 8, 4, 1, 3, 14, 16];$$

2)

$$heap = [12, 10, 7, 9, 8, 4, 1, 3];$$

$$heap_{out} = [12, 10, 7, 9, 8, 4, 1, 3];$$

$$A = [12, 10, 7, 9, 8, 4, 1, 3, 14, 16];$$

i = 7:

$$heap_{in} = [12, 10, 7, 9, 8, 4, 1, 3];$$

$$heap = [3, 10, 7, 9, 8, 4, 1, 12];$$

$$A = [3, 10, 7, 9, 8, 4, 1, 12, 14, 16];$$

1)

$$heap = [3, 10, 7, 9, 8, 4, 1];$$

$$heap = [10, 3, 7, 9, 8, 4, 1];$$

$$A = [10, 3, 7, 9, 8, 4, 1, 12, 14, 16];$$

2)

$$heap = [10, 3, 7, 9, 8, 4, 1];$$

$$heap = [10, 9, 7, 3, 8, 4, 1];$$

$$A = [10, 9, 7, 3, 8, 4, 1, 12, 14, 16];$$

3)

$$heap = [10, 9, 7, 3, 8, 4, 1];$$

$$heap_{out} = [10, 9, 7, 3, 8, 4, 1];$$

$$A = [10, 9, 7, 3, 8, 4, 1, 12, 14, 16];$$

i = 6:

$$heap_{in} = [10, 9, 7, 3, 8, 4, 1];$$

$$heap = [1, 9, 7, 3, 8, 4, 10];$$

$$A = [1, 9, 7, 3, 8, 4, 10, 12, 14, 16];$$

1)

$$heap = [1, 9, 7, 3, 8, 4];$$

$$heap = [9, 1, 7, 3, 8, 4];$$

$$A = [9, 1, 7, 3, 8, 4, 10, 12, 14, 16];$$

2)

$$heap = [9, 1, 7, 3, 8, 4];$$

$$heap = [9, 8, 7, 3, 1, 4];$$

$$A = [9, 8, 7, 3, 1, 4, 10, 12, 14, 16];$$

3)

$$heap = [9, 8, 7, 3, 1, 4];$$

$$heap_{out} = [9, 8, 7, 3, 1, 4];$$

$$A = [9, 8, 7, 3, 1, 4, 10, 12, 14, 16];$$

i = 5:

$$heap_{in} = [9, 8, 7, 3, 1, 4];$$

$$heap = [4, 8, 7, 3, 1, 9];$$

$$A = [4, 8, 7, 3, 1, 9, 10, 12, 14, 16];$$

1)

$$heap = [4, 8, 7, 3, 1];$$

$$heap = [8, 4, 7, 3, 1];$$

$$A = [8, 4, 7, 3, 1, 9, 10, 12, 14, 16];$$

2)

$$heap = [8, 4, 7, 3, 1];$$

$$heap_{out} = [8, 4, 7, 3, 1];$$

$$A = [8, 4, 7, 3, 1, 9, 10, 12, 14, 16];$$

i = 4:

$$heap_{in} = [8, 4, 7, 3, 1];$$

$$heap = [1, 4, 7, 3, 8];$$

$$heap = [1, 4, 7, 3, 8, 9, 10, 12, 14, 16];$$

1)

$$heap = [1, 4, 7, 3];$$

$$heap = [7, 4, 1, 3];$$

$$A = [7, 4, 1, 3, 8, 9, 10, 12, 14, 16];$$

2)

$$heap = [7, 4, 1, 3];$$

$$heap_{out} = [7, 4, 1, 3];$$

$$A = [7, 4, 1, 3, 8, 9, 10, 12, 14, 16];$$

i = 3:

$$heap_{in} = [7, 4, 1, 3];$$

$$heap = [3, 4, 1, 7];$$

$$A = [3, 4, 1, 7, 8, 9, 10, 12, 14, 16];$$

1)

$$heap = [3, 4, 1];$$

$$heap = [4, 3, 1];$$

$$A = [4, 3, 1, 7, 8, 9, 10, 12, 14, 16];$$

2)

$$heap = [4, 3, 1];$$

$$heap_{out} = [4, 3, 1];$$

$$A = [4, 3, 1, 7, 8, 9, 10, 12, 14, 16];$$

i = 2:

$$heap_{in} = [4, 3, 1];$$

$$heap = [1, 3, 4];$$

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 14, 16];$$

1)

$$heap = [1, 3];$$

$$heap = [3, 1];$$

$$A = [3, 1, 4, 7, 8, 9, 10, 12, 14, 16];$$

2)

$$heap = [3, 1];$$

$$heap_{out} = [3, 1];$$

$$A = [3, 1, 4, 7, 8, 9, 10, 12, 14, 16];$$

i = 1:

$$heap = [3, 1];$$

$$heap = [1, 3];$$

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 14, 16];$$

1)

$$heap = [1];$$

$$heap = [1];$$

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 14, 16];$$

i = 0:

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 14, 16].$$

Массив A отсортирован. Для получения неубывающей последовательности следует искать не наибольший элемент между i, $2i + 1$ и $2i + 2$, а наименьший; таким образом, мы построим неубывающую пирамиду.

Код сортировки

Листинг 1: HEAPIFY

```
1       public static void HEAPIFY(int[] A, int i, int heap_size,
            boolean reverse) {
2           int l = (i << 1) + 1, r = l + 1, swapped = i;
3
4           if (l < heap_size && (reverse ? A[l] < A[swapped] : A[l]
                > A[swapped])) swapped = l;
5           if (r < heap_size && (reverse ? A[r] < A[swapped] : A[r]
                > A[swapped])) swapped = r;
6
7           if (swapped != i) {
8               int tmp = A[i];
9               A[i] = A[swapped];
10              A[swapped] = tmp;
11
12              HEAPIFY(A, swapped, heap_size, reverse);
13          }
14      }
```

Листинг 2: BUILD_HEAP

```
1       public static void BUILD_HEAP(int[] A, int heap_size,
            boolean reverse) {
2           for (int i = A.length / 2; i >= 0; --i)
3               HEAPIFY(A, i, heap_size, reverse);
4       }
```

Листинг 3: HEAPSORT

```
1       public static void HEAPSORT(int[] A, boolean reverse) {
2           int heap_size = A.length;
3           BUILD_HEAP(A, heap_size, reverse);
4           for (int i = A.length - 1; i >= 1; --i) {
5               int tmp = A[0];
6               A[0] = A[i];
7               A[i] = tmp;
8
9               HEAPIFY(A, 0, --heap_size, reverse);
10          }
11      }
```

Исходный код

Представим весь код Main.java для тестирования алгоритма.

Листинг 4: Main

```
1  public class Main {
2      public static void HEAPIFY(int[] A, int i, int heap_size,
           boolean reverse) {
3          int l = (i << 1) + 1, r = l + 1, swapped = i;
4
5          if (l < heap_size && (reverse ? A[l] < A[swapped] : A[l]
               > A[swapped])) swapped = l;
6          if (r < heap_size && (reverse ? A[r] < A[swapped] : A[r]
               > A[swapped])) swapped = r;
7
8          if (swapped != i) {
9              int tmp = A[i];
10             A[i] = A[swapped];
11             A[swapped] = tmp;
12
13             HEAPIFY(A, swapped, heap_size, reverse);
14         }
15     }
16     public static void BUILD_HEAP(int[] A, int heap_size,
           boolean reverse) {
17         for (int i = A.length / 2; i >= 0; --i)
18             HEAPIFY(A, i, heap_size, reverse);
19     }
20
21     public static void HEAPSORT(int[] A, boolean reverse) {
22         int heap_size = A.length;
23         BUILD_HEAP(A, heap_size, reverse);
24         for (int i = A.length - 1; i >= 1; --i) {
25             int tmp = A[0];
26             A[0] = A[i];
27             A[i] = tmp;
28
29             HEAPIFY(A, 0, --heap_size, reverse);
30         }
31     }
32     public static void PRINT(int[] A) {
33         for (int i = 0; i < A.length; ++i)
34             System.out.print(A[i] + " ");
35
36         System.out.println();
37     }
38
39     public static void main(String[] args) {
40         int[] A = {8, 3, 12, 16, 10, 4, 1, 9, 7, 14};
41
```

```
42            PRINT(A);
43            HEAPSORT(A, false);
44            PRINT(A);
45        }
46    }
```

Сложность

Во-первых, заметим, что мы не используем дополнительную память не считая некоторых переменных, поэтому можно сразу сказать, что пространственная сложность алгоритма пирамидальной сортировки равна $O(1)$.

В лучшем случае (отстортированный массив), среднем случае (неотсортированный массив), худшем случае (отсортированный в обратную сторону массив) мы выполняем функцию BUILD_HEAP, которая выполняется за время $O(n)$, а в методе сортировки HEAPSORT мы линейно проходимся по массиву A и для каждой итерации вызываем HEAPIFY, которая выполняется за время $O(\lg n)$. Поэтому временная сложность алгоритма пирамидальной сортировки в лучшем, среднем и худшем случае равна $\Omega(n \lg n)$, $\Theta(n \lg n)$ и $O(n \lg n)$ соответственно.

Временная сложность			Пространственная сложность
Худший	Средний	Лучший	Худший
$O(n \lg n)$	$\Theta(n \lg n)$	$\Omega(n \lg n)$	$O(1)$

Таблица 1: **Резюме**

Комментарии

Пирамидальная сортировка хороший алгоритм сортировки, который не использует дополнительной памяти и работает асимптотически стабильно для любых входных последовательностей. Его можно использовать в проектах.