Journal of Computer and Communications, 2017, 5, 57-62
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Heap Sorting Based on Array Sorting

Haiming Li, Ping Chen, Yong Wang

School of Computer and Information Engineering, Shanghai University of Electric Power, Shanghai, China
Email: zjlhm@189.cn

How to cite this paper: Li, H.M., Chen, P. and Wang, Y. (2017) Heap Sorting Based on Array Sorting. *Journal of Computer and Communications*, **5**, 57-62.
https://doi.org/10.4236/jcc.2017.512006

Received: October 10, 2017
Accepted: October 27, 2017
Published: October 30, 2017

Abstract

A kind of heap sorting method based on array sorting was proposed. Some advantages and disadvantages of it were discussed. It was compared with the traditional method of direct application. In the method, the ordered keywords in the array are put into the heap one by one after building an empty heap. This method needs relatively less space and is fit for ordered sequence.

Keywords

Heap Sort, Array, Bottom-Up, Algorithm

1. Introduction

In the field of computer algorithm design, sorting algorithm is one of the important methods which is used to process date. Heap sort algorithm's time complexity is relatively low [1] [2] [3] [4]. So if we can understand its thought very well and use it flexibly, we will solve many problems in our life.

In computer science, heapsort is a comparison-based sorting algorithm. Heapsort can be thought of as an improved selection sort: like that algorithm, we can quickly locate the elements of the required index by using the characteristics of the array [5] [6]. Heap is a completely two binary tree that in ordered set, satisfies the following properties of the heap that the max key value of the key elements of every node in heaps is no more bigger than it in parent node (just in terms of maximum heap terms). Therefore, the largest element of the heap is stored in the root node (the minimum heap similarly, no longer). When we realize the stack with an array of H[1···N], according to the order of top-down and from left to right in turns, we can store the elements of the heap in the array elements of H[1], H[2], ···, H[n]. In general, the left-son node element of heap element of H[i] is H[2i]; the right-son node element is H[2i + 1]; the parent node element is H[[i2]]; then heap properties can be expressed as: H[[i2]] = H[i], I = 2 ~ n.

Heapsort was invented by J. W. J. Williams in 1964 [7] [8]. This was also the birth of the heap, presented already by Williams as a useful data structure in its own right: in this algorithm, establishing the initial stack by O (n) time; then continually exchanging the top element with the bottom element to reconstruction of the heap. Eventually, all the elements are in good order. In the same year, R. W. Floyd published an improved version that could sort an array in-place, continuing his earlier research into the treesort algorithm. It only needs one record size of auxiliary space to use heap sort. And each record to be sorted only takes up one storage space.

It is easy to find that there is a large amount of calculation in the process of reconstructing the heap, and the efficiency of reconstruction depends on comparing between the number of elements and moving of heap elements. The commonly used reconstruction algorithm Heapify [1] [2] makes the node elements along anyone path down. Each layer needs to be compared for 2 times, and the left and right son node elements compare for 1 time. Then the larger one does with the parent node elements for another time. This process is repeated until the parent element is no less than the son node elements (the maximum heap).

There is an example of depth h: Numbers of keywords comparison are at least $2(h-1)$ times in the selection sort algorithm. When building the heap which has n elements and h depth, the numbers should be less 4n for Formula (1). The depth of a complete two fork tree is $\lfloor \log_2 n \rfloor + 1$. In the process of the heap rebuilt, the Heapad just is invoked $n-1$ times. The total numbers of comparison should be no more than

$$\begin{aligned} t(n) &= \sum_{i=2}^{n-1} \lfloor \log_2 i \rfloor \\ &= 2 \times \left[1 \times 2 + 2 \times 2^2 + 3 \times 2^3 + \cdots + (h-1) \times 2^{h-1} + h \times \left(n - 2^h \right) \right] \\ &= 2nh - 2^{h+2} + 4 \end{aligned} \tag{1}$$

Therefore, the worst case is that we should do $2n\log n + O(n)$ comparison and $n\log n\, O(n)$ times elements moving.

About the study of the heap sort, there are now many studies analyzing that and putting forward the optimization plan of it based on different views for the heap sort, such as some reference papers like Mr. Wu Shangzhi who published the "heap sorting algorithm improvement and complexity analysis on the heap" in 2002 at the Journal of Northwest Normal University (Natural Science Edition). It improves the traditional sorting algorithm and reduces the complexity of the algorithm.

2. Reference Knowledge

1) Heap: it can be defined as a two binary tree where each node has one key. There are some requirements:

a) The shape of a tree: every layer of the tree is full except the rightmost element on the last floor.

b) Parent advantage (heap's characteristic): the key of each node isn't less (more than in minimum heap) than its child's key (for any leaf node, we think this condition is automatically satisfied).

2) The large and small root heap: the key of root node (also known as the top of the stack) in which heap is the largest of all node keyword and this heap called root pile or maximum heap. Similarly, the minimum keyword root node in which heap is called the small heap or the minimum heap. For example in **Figure 1**.

a) Large root heap sequences: (96, 83, 27, 38, 11, 09)

b) Small heap sequence: (12, 36, 24, 85, 47, 30, 53, 91)

Be careful:

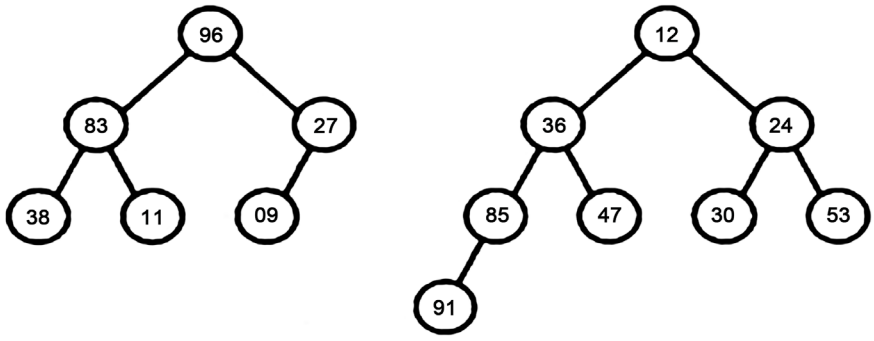
1) Any sub tree in a heap is also a heap.

2) The heap discussed above is actually a two fork heap (Binary Heap). The K fork heap can be defined like that. But it is not studied in this paper.

3) Heap sort is a tree selection sort algorithm. There are some characteristics: in the sorting process, the H[l···N] is regarded as a sequential storage structure with a totally two fork tree. We can choose record of the maximum (or minimum) keyword in the current unsorted state by the relationship between the parent node and child node in two binary tree algorithm (according to the sequence storage structure of the two fork tree). Large root heap (or small root heap) records maximum (or minimum) key, so the heapsort can get the maximum (or minimum) keyword in the unsorted state currently. This process is simpler.

3. A Method of Classical Heap Construction-Bottom-Up Construction Reactor

In the initialization of a completely two forks tree which contains several nodes, the key is placed in the given order, and then heap the tree. The process is as follows: from the last parent node to the root node, checking the key whether meet the requirements. If it doesn't meet the heap's characteristic, we should exchange the position of the biggest key of its child nodes and the key value of the node. We repeat the same process for remaining element until to meet the



(a) (b)

Figure 1. Tree map of a example.

requirements. For the subtree rooted at the current parents node, the algorithm operate trend node of the node with the operation after complete the heaping. After complete the operation of the tree node, the algorithm will be ended.

Description of Bottom-up build heap algorithm:

```
method HeapBottomUp(H[1..n])
//Construct a heap from a known array by a bottom-up algorithm
//Input: a known arrayH[1..n]
//Output: a heapH[1..n]
for i←n/2 downto 1 do
k←i;v←H[k]
heap←false
while not heap and 2*k≤n do
j←2*k
if j<n //There are two children
if H[j]<H[j+1]j←j+1
if v≥H[j]
heap←true
else H[k]←H[j];k←j
H[k]←v
```

4. Array Build Heap

In the first step, the heap is built according to the given order in the classical stack construction method. In the second step, the parents and children nodes are exchanged until to meet the heap's characteristic. Therefore there is some thought. Firstly, we can build the empty heap. Secondly, these key values which given by an array of known sequence stored in the array from large to small (or large) are arranged in the array. Thirdly, we insert the sorted sequence into a heap directly one by one. So you don't have to adjust the key node in the heap.

The specific procession is that: first, constructing an empty two binary tree. Second, all the key value of nodes that removed are stored in an array. Then, these key are ranked with quick sorting according to the order from large to small (maximum heap).Besides, the one at the head is the parent node and the one at the back is the child node. Next, the values are sorted in turn into two binary tree according to the order of the top-down.

Description of array build heap algorithm:

```
method sort Byarray(H[1..n])
//A heap is constructed by array sorting
//Input: a known arrayH[1..n]
//Output: a heapH[1..n]
for i←n-1 downto 1 do
max←i
for j←n downto i+2 do
if H[j]>H[max] max←j
```

swap H[i] and H[max]
for i←1 downto n do
heap←H[i]

5. Algorithm Comparison between Classical Heap Construction Method and Array Heap Build Method

1) Time complexity:
Classical algorithm:
Time needed to build a heap: O(n)
The time needed to take the top element and adjustment: O(nlog(2^n))
Several elements need to repeat this action several times, and the time each action required is related to height. However, height can be considered invariable by rough calculations, so that the time complexity is: nlog(2^n)。
Worst case: Mean complexity
Best case: O(1)
Array algorithm:
In an ideal situation, every time the array to be sorted will be divided into two parts as long as each other, and it needs log(2,n) times division. In the worst case that an array has ordered or roughly ordered, only one element of each partitioning can be divided every time. The lower bound of time complexity is O(nlog(2,n) and the worst case is O(n^2). In actually, the average value of time complexity is O(nlog(2,n).
2) Space complexity:
Classical algorithm: O(1)
Array algorithm:
Best case: O(log(2,n)
Worst case: O(n)
3) Stability:
The stability of two methods is caused by the disagreement of its relationship between father and son nodes with its subscript rules. But the stability can be promised if exchanging their conditions which are comprised. Besides, the two elements which have the same keyword may be belong to different parent nodes, so stability is not confirmed.

6. Conclusion

This paper presents a heap sorting algorithm based on array and finishes the comparison with the traditional method of direct application. In the larger array sequence, the heap sorting algorithm is applied directly. Its time complexity is as same as quick sort and merging sort. It can run with less storage space, so it's fit for ordered sequence sorting. The application of principle reflected that the algorithm is especially suitable for the realization of priority queue.

Acknowledgements

This work is supported by Shanghai University of Electric Power Smart Grid

Collaborative Innovation Project (A-0009-17-002-05) and Shanghai Science and Technology Innovation Fund for Small and Medium Enterprises (1601H1E2600).

References

[1] Pazy, A. (1983) Semigroups of Linear Operators and Applications to Partial Differential Equations. Springer-Verlag, Berlin. https://doi.org/10.1007/978-1-4612-5561-1

[2] Huo, H.W. (2002) Research of Fast Sorting Algorithm. *Microelectronic and Computers*, **19**, 6-9.

[3] Wu, S.Z. (2002) The Analysis of Improved Heap Sorting Algorithm and Its Complexity. *Journal of Northwest Normal University* (*Natural Science Edition*), **38**, 24-26.

[4] Liu, M.Q. (2012) Study of Sorting Algorithm Time Complexity. *Software Tribune*, **11**, 35-37.

[5] Cook, C. and Kim, D. (1980) Best Sorting Algorithm for Nearly Sorted Lists. *CACM*, **23**, 620-626. https://doi.org/10.1145/359024.359026

[6] Mehlhorn, K. (1984) Data Structures and Algorithms. Vol. 1, Springer-Verlag, Berlin.

[7] Schaffer, R. and Sedgewick, R. (1991) The Analysis of Heapsort. Technical Report CS-TR-330-91, Princeton University, Princeton, NJ.

[8] Hayward, R. and Mcdiarmid, C. (1991) Average Case Analysis of Heap Building by Repeated Insertion. *Journal of Algorithms*, **12**, 126-153. https://doi.org/10.1016/0196-6774(91)90027-V