Priority Queue
Implementation using Binary Max Heap

Md Awsaf Alam
Department of Computer Science
Bangladesh University of Engineering and Technology

Contents

Abstract

The abstract does not only mention the paper, but is the original paper shrunken to approximately 200 words. It states the purpose, reports the information obtained, gives conclusions, and recommendations. In short, it summarizes the main points of the study adequately and accurately. It provides information from every major section in the body of the report in a dense and compact way. Past tense and active voice is appropriate when describing what was done. If there is any, it includes key statistical detail.

Depending on the format you use, the abstract may come on the title page or at the beginning of the main report.

1 Introduction

A priority queue is a data structure for maintaining a set S of elements, each with an associated value called a key.A (single-ended) priority queue is a data type supporting the following operations on an ordered set of values:

INSERT (S, x) inserts the element x into the set S, which is equivalent to the operation $S = S \cup x$.

MAXIMUM (S) returns the element of S with the largest key.

EXTRACT-MAX (S) removes and returns the element of S with the largest key.

INCREASE-KEY (S, x, k) increases the value of element x's key to the new value k, which is assumed to be at least as large as x's current key value

Obviously, the priority queue can be redefined by substituting operations $MAXIMUM(S)$ and $EXTRACT{-}MAX(S)$ with $MINIMUM(S)$ and $EXTRACT{-}MIN(S)$, respectively. Several structures, some implicitly stored in an array and some using more complex data structures, have been presented for implementing this data type, including max-heaps (or min-heaps).

2 Applications of Priority Queue

2.1 Job Scheduler

Among their other applications, we can use max-priority queues to schedule jobs on a shared computer. The max-priority queue keeps track of the jobs to be performed and their relative priorities. When a job is finished or interrupted, the scheduler selects the highest-priority job from among those pending by calling $EXTRACT - MAX$. The scheduler can add a new job to the queue at any time by calling $INSERT$.

2.2 Event-Driven Simulator

A min-priority queue can be used in an event-driven simulator. The items in the queue are events to be simulated, each with an associated time of occurrence that serves as its key. The events must be simulated in order of their time of occurrence, because the simulation of an event can cause other events to be simulated in the future. The simulation program calls EXTRACT-MIN at each step to choose the next event to simulate. As new events are produced, the simulator inserts them into the min-priority queue by calling $INSERT$.

2.3 Dijkstra's algorithm

For Dijkstra's algorithm, it is always recommended to use priority queue as the required operations (extract minimum and decrease key) match with speciality of priority queue.When the graph is stored in the form of adjacency list or matrix, priority queue can be used to extract minimum efficiently when implementing Dijkstra's algorithm.

3 Implementations of Priority Queue

Priority queue can be implemented using several datastructures. In this section we will look at some datastructures that can be used.

3.1 Array representation (unordered)

Perhaps the simplest priority queue implementation is based on our code for pushdown stacks. The code for insert in the priority queue is the same as for push in the stack. To implement remove the maximum, we can add code like the inner loop of selection sort to exchange the maximum item with the item at the end and then delete that one, as we did with pop() for stacks. Program UnorderedArrayMaxPQ.java implements a priority queue using this approach.

3.1.1 Complexity Analysis of unordered array, a priority queue data structure

1. **Worst Case analysis**

    - $decrease - key \epsilon O(n)$

2. **Amortized Analysis**

    - $decrease - key \epsilon O(n)$

3. **Expected Running Time**

    - $decrease - key \epsilon O(n)$

3.2 Array representation (ordered)

Another approach is to add code for insert to move larger entries one position to the right, thus keeping the entries in the array in order (as in insertion sort). Thus the largest item is always at the end, and the code for remove the maximum in the priority queue is the same as for pop in the stack. Program OrderedArrayMaxPQ.java implements a priority queue using this approach.

3.2.1 Complexity Analysis of unordered array, a priority queue data structure

1. **Worst Case analysis**
   - $decrease-key \ \epsilon \ O(n)$
2. **Amortized Analysis**
   - $decrease-key \ \epsilon \ O(n)$
3. **Expected Running Time**
   - $decrease-key \ \epsilon \ O(n)$

All of the elementary implementations just discussed have the property that either the insert or the remove the maximum operation takes linear time in the worst case. Finding an implementation where both operations are guaranteed to be fast is a more interesting task, and it is the main subject of this section.

3.3 Fibonacci Heap

The Fibonacci Heap is an interesting data structure with slightly better amortized asymptotic complexity than an k-ary heap. The primary motivation behind the Fibonacci Heap is the O(1) amortized reduce key operation: this allows a lower asymptotic complexity to be derived for Dijkstra's Algorithm.

3.3.1 Complexity Analysis of unordered array, a priority queue data structure

1. **Worst Case analysis**
   - $decrease-key \ \epsilon \ O(n)$
2. **Amortized Analysis**
   - $decrease-key \ \epsilon \ O(n)$
3. **Expected Running Time**
   - $decrease-key \ \epsilon \ O(n)$

4 Binary Max Heap

4.1 Definition

The binary heap is a data structure that can efficiently support the basic priority-queue operations. In a binary heap, the items are stored in an array such that each key is guaranteed to be larger than (or equal to) the keys at two other specific positions. In turn, each of those keys must be larger than two more keys, and so forth. This ordering is easy to see if we view the keys as being in a binary tree structure with edges from each key to the two keys known to be smaller.

4.2 Heap Property

- The key in each node is larger than (or equal to) the keys in that nodes two children
- The largest key in a heap-ordered binary tree is found at the root.

It is particularly convenient, however, to use a complete binary tree like the one below.

We represent complete binary trees sequentially within an array by putting the nodes with level order, with the root at position 1, its children at positions 2 and 3, their children in positions 4, 5, 6 and 7, and so on.

In a heap, the parent of the node in position k is in position k/2; and, conversely, the two children of the node in position k are in positions 2k and 2k + 1. We can travel up and down by doing simple arithmetic on array indices: to move up the tree from a[k] we set k to k/2; to move down the tree we set k to 2*k or 2*k+1.

5 Max Heap Operations

We will now discuss how to implement the operations of a max-priority queue. The procedure HEAP-MAXIMUM implements the M AXIMUM operation in $O(1)$ time.

– Copy algorithms from cormen page 163

H EAP -M AXIMUM .A/ 1 return A The procedure H EAP -E XTRACT -M AX implements the E XTRACT -M AX opera- tion. It is similar to the for loop body (lines 3–5) of the H EAPSORT procedure. H EAP -E XTRACT -M AX .A/ 1 if A:heap-size ¡ 1 2 error “heap underflow” 3 max D AŒ1 4 AŒ1 D AŒA:heap-size 5 A:heap-size D A:heap-size 1 6 M AX -H EAPIFY .A; 1/ 7 return max The running time of H EAP -E XTRACT -M AX is O.lg n/, since it performs only a constant amount of work on top of the O.lg n/ time for M AX -H EAPIFY . The procedure H EAP -I NCREASE -K EY implements the I NCREASE -K EY opera- tion. An index i into the array identifies the priority-queue element whose key we wish to increase. The procedure first updates the key of element AŒi to its new value. Because increasing the key of AŒi might violate the max-heap property, the procedure then, in a manner reminiscent of the insertion loop (lines 5–7) of I NSERTION -S ORT from Section 2.1, traverses a simple path from this node toward the root to find a proper place for the newly increased key. As H EAP -I NCREASE - K EY traverses this path, it repeatedly compares an element to its parent, exchang- ing their keys and continuing if the element’s key is larger, and terminating if the el- ement’s

	Growth Media				
Strain	1	2	3	4	5
GDS1002	0.962	0.821	0.356	0.682	0.801
NWN652	0.981	0.891	0.527	0.574	0.984
PPD234	0.915	0.936	0.491	0.276	0.965
JSB126	0.828	0.827	0.528	0.518	0.926
JSB724	0.916	0.933	0.482	0.644	0.937
Average Rate	0.920	0.882	0.477	0.539	0.923

Table 1: Some impressive numbers

key is smaller, since the max-heap property now holds. (See Exercise 6.5-5 for a precise loop invariant.) H EAP -I NCREASE -K EY .A; i; key/ 1 if key ¡ AŒi 2 error "new key is smaller than current key" 3 AŒi D key 4 while i ¿ 1 and AŒP ARENT .i/ ¡ AŒi 5 exchange AŒi with AŒP ARENT .i/ 6 i D P ARENT .i/ Figure 6.5 shows an example of a H EAP -I NCREASE -K EY operation. The running time of H EAP -I NCREASE -K EY on an n-element heap is O.lg n/, since the path traced from the node updated in line 3 to the root has length O.lg n/. The procedure M AX -H EAP -I NSERT implements the I NSERT operation. It takes as an input the key of the new element to be inserted into max-heap A. The proce- dure first expands the max-heap by adding to the tree a new leaf whose key is 1. Then it calls H EAP -I NCREASE -K EY to set the key of this new node to its correct value and maintain the max-heap property. M AX -H EAP -I NSERT .A; key/ 1 A:heap-size D A:heap-size C 1 2 AŒA:heap-size D 1 3 H EAP -I NCREASE -K EY .A; A:heap-size; key/ The running time of M AX -H EAP -I NSERT on an n-element heap is O.lg n/. In summary, a heap can support any priority-queue operation on a set of size n in O.lg n/ time.

6 Analysis and Interpretation

We shal analyze the running time using Max Heap to implement priority queue *hedging*

A reference to Table 1.

7 Conclusions and Recommendations

Conclusion shows what knowledge comes out of the report. As you draw a conclusion, you need to explain it in terms of the preceding discussion. You are expected to repeat the most important ideas you have presented, without copying. Adding a table/chart summarizing the results of your findings might be helpful for the reader to clearly see the most optimum solution(s).

It is likely that you will briefly describe the comparative effectiveness and suitability of your proposed solutions. Your description will logically recycle language used in your assessing criteria (section **??**): “Solution A proved to be the most cost effective of the alternatives” or “Solution B, though a viable option in other contexts, was shown to lack adaptability”. Do not have detailed analysis or lengthy discussions in this section, as this should have been completed in section X.

As for recommendations, you need to explain what actions the report calls for. These recommendations should be honest, logical and practical. You may suggest that one, a combination, all or none of your proposed solutions should be implemented in order to address your specific problem. You could also urge others to research the issue further, propose a plan of action or simply admit that the problem is either insoluble or has a low priority in its present state.

The recommendations should be clearly connected to the results of the report, and they should be explicitly presented. Your audience should not have to guess at what you intend to say.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] D. Horowitz, *End of Time*. New York, NY, USA: Encounter Books, 2005. [E-book] Available: ebrary, `http://site.ebrary.com/lib/sait/Doc?id=10080005`. Accessed on: Oct. 8, 2008.

[3] D. Castelvecchi, “Nanoparticles Conspire with Free Radicals” *Science News*, vol.174, no. 6, p. 9, September 13, 2008. [Full Text]. Available: Proquest, `http://proquest.umi.com/pqdweb?index=52&did=1557231641&SrchMode=1&sid=3&Fmt=3&VInst=PROD&VType=PQD&RQT=309&VName=PQD&TS=1229451226&clientId=533`. Accessed on: Aug. 3, 2014.

[4] J. Lach, “SBFS: Steganography based file system,” in *Proceedings of the 2008 1st International Conference on Information Technology, IT 2008, 19-21 May 2008, Gdansk, Poland.* Available: IEEE Xplore, `http://www.ieee.org`. [Accessed: 10 Sept. 2010].

[5] “A ‘layman’s’ explanation of Ultra Narrow Band technology,” Oct. 3, 2003. [Online]. Available: `http://www.vmsk.org/Layman.pdf`. [Accessed: Dec. 3, 2003].