Priority queue

From Wikipedia, the free encyclopedia

In computer science, a **priority queue** is an abstract data type which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.

While priority queues are often implemented with heaps, they are conceptually distinct from heaps. A priority queue is an abstract concept like "a list" or "a map"; just as a list can be implemented with a linked list or an array, a priority queue can be implemented with a heap or a variety of other methods such as an unordered array.

Contents

Operations [edit]

A priority queue must at least support the following operations:

- *insert_with_priority*: add an element to the queue with an associated priority.
- *pull_highest_priority_element*: remove the element from the queue that has the *highest priority*, and return it.

  This is also known as "*pop_element(Off)*", "*get_maximum_element*" or "*get_front(most)_element*". Some conventions reverse the order of priorities, considering lower values to be higher priority, so this may also be known as "*get_minimum_element*", and is often referred to as "*get-min*" in the literature. This may instead be specified as separate "*peek_at_highest_priority_element*" and "*delete_element*" functions, which can be combined to produce "*pull_highest_priority_element*".

In addition, *peek* (in this context often called *find-max* or *find-min*), which returns the highest-priority element but does not modify the queue, is very frequently implemented, and nearly always executes in *O*(1) time. This operation and its *O*(1) performance is crucial to many applications of priority queues.

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Български
Català
Čeština
Deutsch
Ελληνικά
Español
فارسی
Français
한국어
Italiano
עברית
Lietuvių
日本語
Polski
Русский
Српски / srpski
Suomi
Svenska
ไทย
Українська
中文

Edit links

More advanced implementations may support more complicated operations, such as *pull_lowest_priority_element*, inspecting the first few highest- or lowest-priority elements, clearing the queue, clearing subsets of the queue, performing a batch insert, merging two or more queues into one, incrementing priority of any element, etc.

Similarity to queues [edit]

One can imagine a priority queue as a modified queue, but when one would get the next element off the queue, the highest-priority element is retrieved first.

Stacks and queues may be modeled as particular kinds of priority queues. As a reminder, here is how stacks and queues behave:

- *stack* – elements are pulled in last-in first-out-order (e.g., a stack of papers)
- *queue* – elements are pulled in first-in first-out-order (e.g., a line in a cafeteria)

In a stack, the priority of each inserted element is monotonically increasing; thus, the last element inserted is always the first retrieved. In a queue, the priority of each inserted element is monotonically decreasing; thus, the first element inserted is always the first retrieved.

Implementation [edit]

Naive implementations [edit]

There are a variety of simple, usually inefficient, ways to implement a priority queue. They provide an analogy to help one understand what a priority queue is. For instance, one can keep all the elements in an unsorted list. Whenever the highest-priority element is requested, search through all elements for the one with the highest priority. (In big *O* notation: *O*(1) insertion time, *O*(*n*) pull time due to search.)

Usual implementation [edit]

To improve performance, priority queues typically use a heap as their backbone, giving *O*(log *n*) performance for inserts and removals, and *O*(*n*) to build initially. Variants of the basic heap data structure such as pairing heaps or Fibonacci heaps can provide better bounds for some operations.[1]

Alternatively, when a self-balancing binary search tree is used, insertion and removal also take *O*(log *n*) time, although building trees from existing sequences of elements takes *O*(*n* log *n*) time; this is typical where one might already have access to these data structures, such as with third-party or standard libraries.

Note that from a computational-complexity standpoint, priority queues are congruent to sorting algorithms. See the next section for how efficient sorting algorithms can create efficient priority queues.

Specialized heaps [edit]

There are several specialized heap data structures that either supply additional operations or outperform heap-based implementations for specific types of keys, specifically integer keys.

- When the set of keys is {1, 2, ..., *C*}, and only *insert*, *find-min* and *extract-min* are needed, a *bounded height priority queue* can be constructed as an array of C linked lists plus a pointer top, initially C. Inserting an item with key k appends the item to the k'th, and updates $\mathrm{top} \leftarrow \min(\mathrm{top}, k)$, both in constant time. *Extract-min* deletes and returns one item from the list with index top, then increments top if needed until it again points to a non-empty list; this takes $O(C)$ time in the worst case. These queues are useful for sorting the vertices of a graph by their degree.[2]:374
- For the set of keys {1, 2, ..., *C*}, a van Emde Boas tree would support the *minimum*, *maximum*, *insert*, *delete*, *search*, *extract-min*, *extract-max*, *predecessor* and *successor* operations in *O*(log log *C*) time, but has a space cost for small queues of about $O(2^{m/2})$, where *m* is the number of bits in the priority value.[3]
- The Fusion tree algorithm by Fredman and Willard implements the *minimum* operation in *O*(1) time and *insert* and *extract-min* operations in $O(\sqrt{\log n})$ time however it is stated by the author that, "Our algorithms have theoretical interest only; The constant factors involved in the execution times preclude practicality.".[4]

For applications that do many "peek" operations for every "extract-min" operation, the time complexity for peek actions can be reduced to *O*(1) in all tree and heap implementations by caching the highest priority element after every insertion and removal. For insertion, this adds at most a constant cost, since the newly inserted element is compared only to the previously cached minimum element. For deletion, this at most adds an additional "peek" cost, which is typically cheaper than the deletion cost, so overall time complexity is not

significantly impacted.

Monotone priority queues are specialized queues that are optimized for the case where no item is ever inserted that has a lower priority (in the case of min-heap) than any item previously extracted. This restriction is met by several practical applications of priority queues.

Equivalence of priority queues and sorting algorithms [edit]

Using a priority queue to sort [edit]

The semantics of priority queues naturally suggest a sorting method: insert all the elements to be sorted into a priority queue, and sequentially remove them; they will come out in sorted order. This is actually the procedure used by several sorting algorithms, once the layer of abstraction provided by the priority queue is removed. This sorting method is equivalent to the following sorting algorithms:

Name	Priority Queue Implementation	Best	Average	Worst
Heapsort	Heap			
Smoothsort	Leonardo Heap	n		
Selection sort	Unordered Array	n^2	n^2	n^2
Insertion Sort	Ordered Array	n	n^2	n^2
Tree sort	self-balancing binary search tree			

Using a sorting algorithm to make a priority queue [edit]

A sorting algorithm can also be used to implement a priority queue. Specifically, Thorup says:[5]

> We present a general deterministic linear space reduction from priority queues to sorting implying that if we can sort up to n keys in $S(n)$ time per key, then there is a priority queue supporting *delete* and *insert* in $O(S(n))$ time and *find-min* in constant time.

That is, if there is a sorting algorithm which can sort in $O(S)$ time per key, where S is some function of n and word size,[6] then one can use the given procedure to create a priority queue where pulling the highest-priority element is $O(1)$ time, and inserting new elements (and deleting elements) is $O(S)$ time. For example, if one has an $O(n \log \log n)$ sort algorithm, one can create a priority queue with $O(1)$ pulling and $O(\log \log n)$ insertion.

Libraries [edit]

A priority queue is often considered to be a "container data structure".

The Standard Template Library (STL), and the C++ 1998 standard, specifies `priority_queue` as one of the STL container adaptor class templates. However, it does not spec how two elements with same priority should be served, and indeed, common implementations will not return them according to their order in the queue. It implements a max-priority-queue, and has three parameters: a comparison object for sorting such as a functor (defaults to less<T> if unspecified), the underlying container for storing the data structures (defaults to std::vector<T>), and two iterators to the beginning and end of a sequence. Unlike actual STL containers, it does not allow iteration of its elements (it strictly adheres to its abstract data type definition). STL also has utility functions for manipulating another random-access container as a binary max-heap. The Boost (C++ libraries) also have an implementation in the library heap.

Python's heapq module implements a binary min-heap on top of a list.

Java's library contains a `PriorityQueue` class, which implements a min-priority-queue.

Go's library contains a container/heap module, which implements a min-heap on top of any compatible data structure.

The Standard PHP Library extension contains the class SplPriorityQueue.

Apple's Core Foundation framework contains a CFBinaryHeap structure, which implements a min-heap.

Applications [edit]

Bandwidth management [edit]

Priority queuing can be used to manage limited resources such as bandwidth on a transmission line from a network router. In the event of outgoing traffic queuing due to insufficient bandwidth, all other queues can be halted to send the traffic from the highest priority queue upon arrival. This ensures that the prioritized traffic (such as real-time traffic, e.g. an RTP stream of a VoIP connection) is forwarded with the least delay and the least likelihood of being rejected due to a queue reaching its maximum capacity. All other traffic can be handled when the highest priority queue is empty. Another approach used is to send disproportionately more traffic from higher priority queues.

Many modern protocols for local area networks also include the concept of priority queues at the media access control (MAC) sub-layer to ensure that high-priority applications (such as VoIP or IPTV) experience lower latency than other applications which can be served with best effort service. Examples include IEEE 802.11e (an amendment to IEEE 802.11 which provides quality of service) and ITU-T G.hn (a standard for high-speed local area network using existing home wiring (power lines, phone lines and coaxial cables).

Usually a limitation (policer) is set to limit the bandwidth that traffic from the highest priority queue can take, in order to prevent high priority packets from choking off all other traffic. This limit is usually never reached due to high level control instances such as the Cisco Callmanager, which can be programmed to inhibit calls which would exceed the programmed bandwidth limit.

Discrete event simulation [edit]

Another use of a priority queue is to manage the events in a discrete event simulation. The events are added to the queue with their simulation time used as the priority. The execution of the simulation proceeds by repeatedly pulling the top of the queue and executing the event thereon.

See also: Scheduling (computing), queueing theory

Dijkstra's algorithm [edit]

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, although one also needs the ability to alter the priority of a particular vertex in the priority queue efficiently.

Huffman coding [edit]

Huffman coding requires one to repeatedly obtain the two lowest-frequency trees. A priority queue is one method of doing this.

Best-first search algorithms [edit]

Best-first search algorithms, like the A* search algorithm, find the shortest path between two vertices or nodes of a weighted graph, trying out the most promising routes first. A priority queue (also known as the *fringe*) is used to keep track of unexplored routes; the one for which the estimate (a lower bound in the case of A*) of the total path length is smallest is given highest priority. If memory limitations make best-first search impractical, variants like the SMA* algorithm can be used instead, with a double-ended priority queue to allow removal of low-priority items.

ROAM triangulation algorithm [edit]

The Real-time Optimally Adapting Meshes (ROAM) algorithm computes a dynamically changing triangulation of a terrain. It works by splitting triangles where more detail is needed and merging them where less detail is needed. The algorithm assigns each triangle in the terrain a priority, usually related to the error decrease if that triangle would be split. The algorithm uses two priority queues, one for triangles that can be split and another for triangles that can be merged. In each step the triangle from the split queue with the highest priority is split, or the triangle from the merge queue with the lowest priority is merged with its neighbours.

Prim's algorithm for minimum spanning tree [edit]

Using min heap priority queue in Prim's algorithm to find the minimum spanning tree of a connected and undirected graph, one can achieve a good running time. This min heap priority queue uses the min heap data structure which supports operations such as *insert*, *minimum*, *extract-min*, *decrease-key*.[7] In this implementation, the weight of the edges is used to decide the priority of the vertices. Lower the weight, higher the priority and higher the weight, lower the priority.[8]

See also [edit]

- Batch queue
- Command queue
- Job scheduler

References [edit]

1. ^ Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Chapter 20: Fibonacci Heaps, pp.476–497. Third edition p518.
2. ^ Skiena, Steven (2010). *The Algorithm Design Manual* (2nd ed.). Springer Science+Business Media. ISBN 1-849-96720-2.
3. ^ P. van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, pages 75-84. IEEE Computer Society, 1975.
4. ^ Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 48(3):533-551, 1994
5. ^ Thorup, Mikkel (2007). "Equivalence between priority queues and sorting". *Journal of the ACM* **54** (6). doi:10.1145/1314690.1314692.
6. ^ http://courses.csail.mit.edu/6.851/spring07/scribe/lec17.pdf
7. ^ Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein (2009). *INTRODUCTION TO ALGORITHMS* **3**. MIT Press. p. 634. ISBN 978-81-203-4007-7. "In order to implement Prim's algorithm efficiently, we need a fast way to select a new edge to add to the tree formed by the edges in A. In the pseudo-code"
8. ^ "Prim's Algorithm". Geek for Geeks. Retrieved 12 September 2014.

Further reading [edit]

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 6.5: Priority queues, pp. 138–142.

External links [edit]

- C++ reference for `std::priority_queue`
- Descriptions by Lee Killough
- PQlib - Open source Priority Queue library for C
- libpqueue is a generic priority queue (heap) implementation (in C) used by the Apache HTTP Server project.
- Survey of known priority queue structures by Stefan Xenos
- UC Berkeley - Computer Science 61B - Lecture 24: Priority Queues (video) - introduction to priority queues using binary heap

v · t · e	Data structures
Types	Collection · Container
Abstract	Associative array · Double-ended priority queue · Double-ended queue · List · Map · Multimap · **Priority queue** · Queue · Set (multiset) · Disjoint Sets · Stack
Arrays	Bit array · Circular buffer · Dynamic array · Hash table · Hashed array tree · Sparse array
Linked	Association list · Linked list · Skip list · Unrolled linked list · XOR linked list
Trees	B-tree · Binary search tree (AA · AVL · red-black · self-balancing · splay) · Heap (binary · binomial · Fibonacci) · R-tree (R* · R+ · Hilbert) · Trie (Hash tree)
Graphs	Binary decision diagram · Directed acyclic graph · Directed acyclic word graph
List of data structures	

Categories: Priority queues | Abstract data types