Trouble with Heaps

Recently I posted a question here on stackoverflow Time complexity issues with multimap and I got some great answers which referred me to the use of heaps, which oddly I hadn't used at all before now. I created a new program re-written with the use of a minheap and a maxheap. It worked great in that it was a lot faster than any of the other programs I had implemented for this problem. The only problem was it was throwing out some wrong answers occasionally. I went back and did a lot of debugging. I realized that the problem was in the organization of my heap. It wasn't being sorted and distributed like I thought it would be with the use push_heap and the pop_heap with the comparison operations. Also, when I would try running the program on Visual Studio, I would end up seeing a lot of the assertion errors being thrown out there. I tried reading up more on heaps and their methods at cplusplus.com and cppreference.com. I think I maybe am not understanding something correctly and therefore am running into further problems.

first thing that is confusing me is push_heap. The way I understand it is this: push_heap has two arguments and by **default** it pushes the ***least*** value to position last-1. It only does this if the first argument is less than the second argument, otherwise it stays the same. it basically maintains the order of a normal heap. The third optional argument is a comparison operator, which can be used as greater() which then pushes the greater element to the last-1 position.

What doesn't make sense is if I have a dynamic insertion or deletion of numbers going on in a vector, I have issues keeping this order. If I wanted the vector to be in ascending order I would use the greater operation to keep pushing the heap so that the values would be ascending. But it is confusing when you first look at the push_heap method because it appears a lot like some of the other algorithm functions which perform in a ***range*** of numbers like for instance:

```
std::unique (myvector.begin(), myvector.end(), myfunction);
```

which push_heap does not do. it does not do this comparison operation on **all** the numbers in the *range* of the vector, which I didn't understand at first.

After finding that push_heap wasn't really keeping my vector sorted, and I had to keep my vector sorted in order to use a binary search. I used sort_heap, but that slowed down the program to where it wasn't fast enough.

Also, I find that sometimes push_heap will throw an invalid heap error in strange occasions.

like for example:

```
   push_heap(v.begin(), v.end(), greater<int>());
```

with a vector of **755, 98, 55, 22**

you would see after the push_heap:

```
      22, 98, 55, 755
```

but let's say that you had **22, 98, 55 ,755**

usually it will just move on without performing any push because of the false return on comparison. this is to be expected.

but sometimes I will try push_heap on:

887, 52, 44, 22

and it will say

```
       'invalid heap'
```

or if I try: **22, 52, 44, 887**, instead of just returning false and moving on it will break with

```
 'invalid heap'
```

this also happens sometimes with pop_heap.

Why am I getting invalid heap? is it because all heaps have to be in descending order?

EDIT: I found this on cplusplus.com, which I guess answers one question:

```
 The element with the highest value is always pointed by first. The order of the other elements depends on the particular
implementation, but it is consistent throughout all heap-related functions of this header.
```

c++ vector heap time-complexity stl-algorithm edited Mar 25 '13 at 18:09

asked Mar 25 '13 at 17:36
user1066524
153 4 19

I'd say "forget about `push_heap`, and use `std::priority_queue`. priority queue works the same way as the heap stuff, but is simpler to use and understand. – Mooing Duck Mar 25 '13 at 18:25

priority_queue would be great, except for my problem I need to be able to remove one item, which priority_queue won't let you do. but other than that, priority_queue is a decent choice. – user1066524 Mar 25 '13 at 18:31

`std::priority_queue` has `pop` and `top` for looking at and removing the "greatest" element. If that's not what you want, then none of the heap stuff is what you want either, since they work on the *exact* same concepts. – Mooing Duck Mar 25 '13 at 19:28

1 Answer

> ... push_heap has two arguments and by default it pushes the least value to position last-1. It only does this if the first argument is less than the second argument, otherwise it stays the same.

Nope. If your storage is a vector `v`, and is currently a heap (as created with `make_heap`), you should call

```
v.push_back(new_item);
push_heap(v.begin(), v.end());
```

to add a new item. See for example here or here.

Consider that `push_heap` really takes the range `[begin, end-1)` (which is required to fulfil the heap invariant already) and the appended element at `end-1` (which may not), and moves up the last element until the heap invariant is restored for all of `[begin, end)`. The algorithm is explained here.

> After finding that push_heap wasn't really keeping my vector sorted ...

Heaps *are not sorted*. They have an ordering constraint (the heap property) which is *specifically and deliberately weaker* than being fully sorted.

If you want to perform a binary search, you need a fully-sorted container, and converting your heap to one using `sort_heap` each time is both slow and destructive: your container isn't a heap any more after you call this, and you can't use it as one.

Now, about your edit: heaps *do not* have to be in descending order. A max-heap is in descending order (with the largest element at the front), and a min-heap is in ascending order (with the smallest element at the front).

The *default* in the standard library is to build a min-heap, using `operator<` for comparison. To build a max-heap instead, you just pass `std::greater<int>` or whatever for the (optional) final argument.

edited Mar 25 '13 at 18:17

answered Mar 25 '13 at 17:51
Useless
24.2k 2 24 59

When you say heap sorted, do you mean that I called make_heap before doing a push_heap, or do you mean actually having to use the heap_sort function? – user1066524 Mar 25 '13 at 17:53

`make_heap` turns your vector into a heap, and it needs to be a heap for `push_heap` to work. If you `heap_sort` it, then the heap invariant is broken. Read the last link (to wikipedia) if you don't understand what a heap is supposed to look like. – Useless Mar 25 '13 at 17:55

1 yes, `push_heap` moves the last element (which you just added to the back, so it may violate the heap property) up until the heap property is restored. So, it jumps around in your heap until it's valid again. – Useless Mar 25 '13 at 18:08

3 @user1066524: In a max-heap, the first element will be the greatest, but the least is not necessarily the last. – Dave S Mar 25 '13 at 18:16

1 @user1066524 Like I said, the "elements in between" are in the correct order *for a heap*, because they satisfy the heap property. They *are not sorted*. And for getting max & min, you're almost there but I think

you have it backwards. – Useless Mar 25 '13 at 18:19