CS2040S Semester 1 2023/2024
Data Structures and Algorithms

Tutorial+Lab 04
Priority Queue

For Week 06

Document is last modified on: September 20, 2023

1 Introduction and Objective

This tutorial marks the end of the first $\frac{1}{3}$ of CS2040/C/S: Basic Java/C++, basic analysis of algorithms (worst case time complexity only), various sorting algorithms, and various linear Data Structures (DSes): Linked List/Stack/Queue/Doubly Linked List/Deque.

This tutorial marks the start of the next $\frac{1}{3}$ of CS2040/C/S: Various non-linear DSes. Today, we will discuss the Priority Queue (PQ) ADT with its Binary Heap implementation (use `https://visualgo.net/en/heap` to help you answer some questions in this tutorial).

2 Tutorial 04 Questions

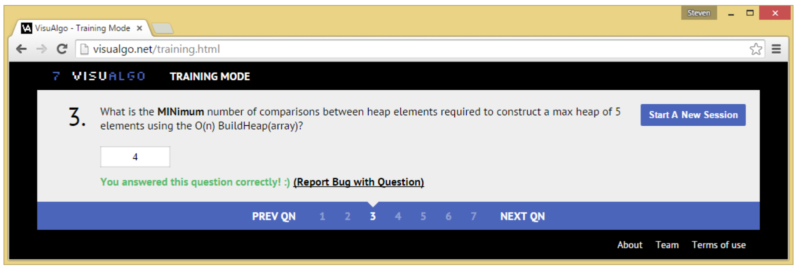
Basic Binary Heap

Q1). Quick check: Let's review all basic operations of Binary Heap that are currently available in VisuAlgo (use the Exploration mode of `http://visualgo.net/en/heap`). During the tutorial session, the tutor will randomize the Binary Heap structure, ask student to compare Binary Tree versus (1-based) Compact Array mode, `Insert(random-integer)` (you can try inserting duplicates, it is now allowed), perform `ExtractMax()` operations (once, K-times (i.e., partial sort), or N-times (i.e., `HeapSort()`)), the $O(N \log N)$ or the $O(N)$ `Create(from-a-random-array)`, `UpdateKey(i, newv)` and `Delete(i)`.

This part is open ended, up to the tutor. Focus on how Binary Heap can be used as a **Priority** Queue. Do not be long winded here. Perhaps focus more on the four new ones: Changing Mode, Partial Sort, Update Key (if index known), Delete Key (if index known).

Q2). What is the minimum and maximum number of comparisons between Binary Heap elements required to construct a Binary (Max) Heap of arbitrary n elements using the $O(n)$ `Create(array)`?

Note that this question has been integrated in VisuAlgo Online Quiz, so it may appear in future Online Quizzes :).



3. What is the **MINimum** number of comparisons between heap elements required to construct a max heap of 5 elements using the O(n) BuildHeap(array)?

4

You answered this question correctly! :) **(Report Bug with Question)**

Figure 1: Now automated :)

Ans: Let's use an example for $n = 8$. The min and max answers are 7 and 11 respectively.

The case for minimum happens when the array to be converted into a Binary (Max) Heap already satisfies the Max Heap property (e.g., try input array {8,7,6,5,4,3,2,1} in VisuAlgo). The structure of an 8 vertices heap is shown below.

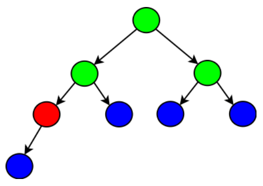


Figure 2: Minimum case, n=8

In this case except for the last internal vertex (the red vertex), which only does 1 comparison, the rest of the internal vertices (green vertices) do exactly 2 comparisons (with its left and right child) so the # of comparisons = 1+3*2 = 7.

The case for maximum happens where we have to call `ShiftDown` at each internal node all the way to the deepest leaf (note: contrary to intuition, try input array {1,2,3,4,5,6,7,8} in VisuAlgo does **not** really produce the maximum number of comparison as 1 will be shifted down to 8 then to 5 only, try input array {1,2,3,5,4,6,7,8} in VisuAlgo where 1 will be shifted down to 8, then 5, then 2. The structure of an 8 vertices heap is shown below and can be generated as follows: {5,6,2,7,4,3,1,8} (do you see the pattern on how to generate this test case?).
The red node requires 1 comparison
The purple node requires 2 comparisons
The green node requires 2+1 (2 comparisons at level 1, 1 comparison one level down at level 2)

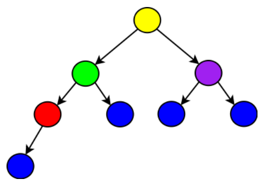


Figure 3: Maximum case, n=8

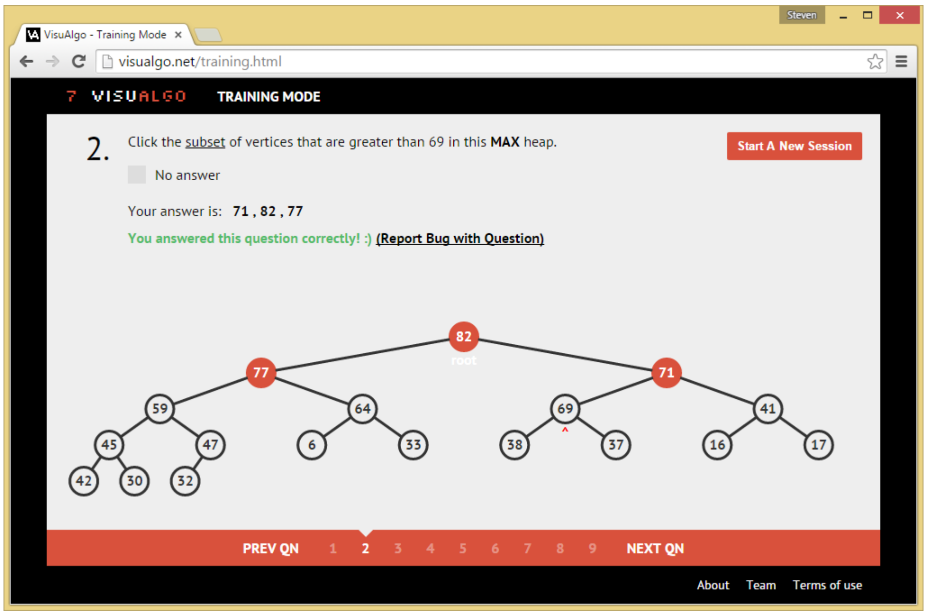
Finally, the yellow node requires 2+2+1 (trace the longest path) comparisons
Total = 1+2+(2+1)+(2+2+1) = 11.

We have a bigger test case of this pattern (with integer values) in VisuAlgo (/heap). Click 'Create(A) - O(N)', 'Worst Case: Diagonal Entry' test case for a more general form of this test case.

More Binary Heap

Q3). Give an algorithm to find all vertices that have value $> x$ in a Binary max heap of size n. Your algorithm must run in $O(k)$ time where k is the number of vertices in the output.
Key lesson: This is a new algorithm analysis type for most of you as the time complexity of the algorithm does not depends on the input size n but rather the output size k :O...

Note that this question has also been integrated in VisuAlgo Online Quiz, so it may appear in future Online Quizzes :).



2. Click the subset of vertices that are greater than 69 in this **MAX** heap.

No answer

Your answer is: **71 , 82 , 77**

You answered this question correctly! :) (Report Bug with Question)

Figure 4: Also automated :)

Ans: The key insight is that the answers are at the top of the max heap... So we can perform a pre-order traversal (tutor will elaborate the concept of pre-order traversal as this may be new for many students; such graph traversal algorithms will only be discussed later) of the max heap starting from the root. At each vertex, check if vertex key is > x. If yes, output vertex and continue traversal. Otherwise terminate traversal on subtree rooted at current vertex (as none of the vertex in this subtree will contribute to the answers), return to parent and continue traversal.

Algorithm 1 findVerticesBiggerThanX(vertex, x)

```
if (vertex.key $>$ x) then
    output(vertex.key)
    findVerticesBiggerThanX(vertex.left, x)
    findVerticesBiggerThanX(vertex.right, x)
end if
```

Analysis of time bound required:

The traversal terminates when it encounters that a vertex's key $\leq x$. In the worst case, it encounters $k * 2$ number of such vertices. That is, each of the left and right child of a valid vertex (vertex with key $> x$) are invalid. It will not process any invalid vertex other than those $k * 2$ vertices. It will process all k valid vertices, since there cannot be any valid vertices in the subtrees not traversed (due to the heap property). Thus the traversal encounters $O(k + 2 * k) = O(k)$ number of vertices in order to output the k valid vertices.

Q4). Show an easy way to convert a Binary Max Heap of a set integers (as shown in VisuAlgo `https://visualgo.net/en/heap`) into a Binary Min Heap (of the 'same' set of integers) without changing the underlying data structure at all. Hint: modify the data.

Ans: Simple, just insert the **negation** of those integers into another empty Binary Max Heap. For example: 5 (max), 4, 3, 2, 1 will now looks like this if negated: -1 (max), -2, -3, -4, -5.

For Java, the default of Java PriorityQueue class is a min heap. We can convert it into a max heap for numbers by inserting their negatives.

Q5). The *second* largest element in a max heap with more than two elements (to simplify this question, you can assume that all elements are unique) is always one of the children of the root. Is this true? If yes, show a simple proof. Otherwise, show a counter example.

Note that this kind of (simple) proof may appear in future CS2040/C/S written tests, so please refresh your CS1231 (if you have taken that module) or just concentrate on how the tutor will answer this kind of question.

Yes it is true. This can be proven easily by proof of contradiction. Suppose the second largest element is not one of the children of the root. Then, it could be the root, but this cannot be since root is the largest element by definition, thus contradiction. Or it has a parent that is not the root :O. There will be a violation to max heap property (there is nothing between largest and 2nd largest element). Contradiction. So, the second element must always be one of the children of the root.

To make things more interesting, tutor can vary the statement, e.g., third largest, smallest, second smallest, etc.

More ADT PQ Operations

Q6). There are two Binary Heap data structure features that are not available (yet) in C++ STL `std::priority_queue`, Python `heapq`, and Java `PriorityQueue`: `Increase/Decrease/UpdateKey(old_v, new_v)` and `DeleteKey(v)` where `v` is not necessarily the max element. These two operations are not yet included in VisuAlgo (the hidden slide `https://visualgo.net/en/heap?slide=3-1`) although the version where index i is known, is now included in VisuAlgo (see `https://visualgo.net/en/heap?slide=9-2` and `https://visualgo.net/en/heap?slide=9-3`).

Q6a). Given `https://www.comp.nus.edu.sg/~stevenha/cs2040c/demos/BinaryHeapDemo.cpp` (that is a Binary Max Heap), what should we modify/add so that we can implement `DecreaseKey(old_v, new_lower_v)` – notice that we never IncreaseKey?

How to get where `old-v` is located in Binary Heap quickly, i.e., faster than $O(n)$?
Answer: Looping through index $i \in [1..n]$ will cost us $O(n)$. Self study Hash Table topic :O, to be revisited during next Tut05+Tut06, to do this in $O(1)$. This is the hidden answer of `https://visualgo.net/en/heap?slide=9-4`.

Will `DecreaseKey` operation affect the Complete Binary Tree (compact array) property?
Answer: No, we only change a value to another (lower) value. The Complete Binary Tree structure is unaffected.

Suppose `old-v` is located at index $i \in [1..n]$ in Binary Heap. Suppose we have changed `old-v` to `new_lower_v`. Should we do anything else?
Answer: Yes, because we need to check whether we now violate Max Heap property downwards. We call `shift_down(i)` to address this.

Note to tutor: Just give general ideas without showing any C++/Python/Java code on how `DecreaseKey` works. Just tell students that this single additional ADT Priority Queue operation `DecreaseKey` will be heavily used in Dijkstra's algorithm for SSSP problem later on (Week 12-13). Note: No need to go through lazy update idea. PS: `IncreaseKey` is the opposite. `UpdateKey` is the generic form of this operation.

Q6b). Given `https://www.comp.nus.edu.sg/~stevenha/cs2040c/demos/BinaryHeapDemo.cpp` (that is a Binary Max Heap), what should we modify/add so that we can implement `DeleteKey(v)` where `v` is not necessarily the max element?

We now know (from earlier discussion) that we can get the position of v (not necessarily at the root) still in $O(1)$ with help of Hash Table.

This 'delete at arbitrary position' has four known possible ways:

1. Ask last item to replace v at index $i = position[v]$ then call **both** `shift_up(i)` and `shift_down(i)` (only one is true and will be executed – think about it),
2. Use the (reverse of) previously discussed method: `IncreaseKey(position[v], +inf)`, then `ExtractMax()`,

3. Use lazy deletion technique (quite advanced, FYI only, not to be discussed on the spot), or
4. Or... tell them that there is a Shocking Stuff (title of hidden `https://visualgo.net/en/heap?slide=9-9`) that will be revealed on Week 08.

After all these are discussed, one of the task in PS4 later should be 'trivial' again...

Hands-on 4

TA will run the second half of this session with a few to do list:

- Very quick review of Java PriorityQueue Class
- Do a sample speed run of VisuAlgo online quiz that are applicable so far, e.g., `https://visualgo.net/training?diff=Medium&n=5&tl=5&module=heap`,
- No hands-on today, deferred to next session.

1-2 slides for Java API demonstrations, just show a few techniques or refer students to relevant references, e.g., `https://github.com/stevenhalim/cpbook-code/blob/master/ch2/nonlineards/priority_queue.java`.
Show off in 5m :), again (on another set of random questions).

Problem Set 3

We will end the tutorial with **another round of algorithmic** discussion of PS3 that will due soon.

Give a bit more details for PS3 A (/sim) subtask 2+3 (the presence of '[' and a very long list with large n). To handle the insertion at **the current cursor** position (as '[' can cause the cursor/insertion point to moves to the beginning of the list), it is best to use Java ListIterator (`https://docs.oracle.com/javase/8/docs/api/java/util/ListIterator.html`) that can do this in $O(1)$ per insertion. However, many students will find that they can still get by (not TLE) with DLLinstance.add(currentcursorposition, nextchar) operation... Discuss this with students that this can still be $O(n)$ per insertion (imagine a very long DLL of size $\frac{n}{2}$, then one '[', then another $\frac{n}{2}$ insertions of other characters, this is $O(n/2 \times 1 + (n/2) \times (n/2-1)/2) = O(n^2)$ overall...). One day, Prof Halim will create /sim2 that has (much) stronger test cases than this (/sim).

A bit more details for PS4 B (/janeeyre). It is classified as an easy PQ simulation once students can handle the I/O part.

Midterm Quiz Preparation

TAs have the model answers of the recent past papers (past three years). These files will not be officially given to the students. TAs will spend some time during this session to give some Midterm Quiz preparation tips.