Data Structures
Quiz 3 Review

Cain Susko

Queen's University
School of Computing

March 14, 2022

Priority Queue

we should know the following things to do with Priority Queue:

- priority queue vs. BST
- Properties - ie. for a complete binary tree, what is the priority level
- Operations and time complexity
- Implementations
  - max heap, min heap (array based, linked list, tree)
  - check, put (add, insert), pop (delete)
  - heapify
  - heapsort
- Applications
  - use priority queue for task scheduling
  - heapsort

Operations

a priority queue has three operations:

i insert a new item into a queue

ii get he value of the highest priority item

iii remove the highest priority item from the gueue

Additionally, when we implement a priority queue, we must define what is the priority of each possible item in the queue (normally based on it's value)

Heaps

There are 2 types of heaps that we can use (which implement the priority queue)

- maxheap
- minheap

maxheap has the following advantages:

- Quickli insert items into the heap
- quickly extract the largest item from the heap

minheap has the following advantages

- quickly insert a new item into the heap
- quickly extract the smallest value from the heap

Implementation

When implementing a heap, we find the best was is by using an array. We do this like so:

1. the root of the heap goes in `array[0]`
2. if the data for the node appears in `array[i]`, it's children, if they exist, are in the locations:
   - left child: `array[2i+1]`
   - right child: `array[2i+2]`
3. if the data for a non-root node is `array[i]`, then its parent is always at `arra[(i-1)/2]` (using integer division)

Extracting from the Heap

Adding Node to Heap

Complexity the complexity of the Heap, when both inserting and extracting is

$$O(\log_2(n))$$

But, if you cannot find a given element in a heap, it takes

$$O(n)$$

time. In order to avoid this case we can implement efficient heapsort.

1. If the len(heap) == 0 (it's an empty tree), return error.
2. Otherwise, heap[0] holds the biggest value. Remember it for later.
3. If the len(heap) == 1 (that was the only node) then delete the only value and return the saved value.
4. Copy the value from the right-most, bottom-most node to the root node: heap[0] = heap[-1]
5. Delete the right-most node in the bottom-most row: del heap[-1]
6. Repeatedly swap the just-moved value with the larger of its two children:
   Starting with i=0, compare and swap:
   heap[i] with heap[2*i+1] and heap[2*i+2]
7. Return the saved value to the user.

	heap
0	12
1	7
2	10
3	3
4	2
5	8
6	4
7	2
8	1
9	
10	
11	
12	
13	
...	

Heapsort

Heapsort is done in 2 steps:

1. convert input array into maxheap
2. reheapify the array:

Complexity the time complexity for transfering a list into a maxheap is:

$$O(n)$$

and the time complexity for extracting a element is :

$$O(\log n) * n$$

1. Insert a new node in the bottom-most, left-most open slot:
   heap.append(value)

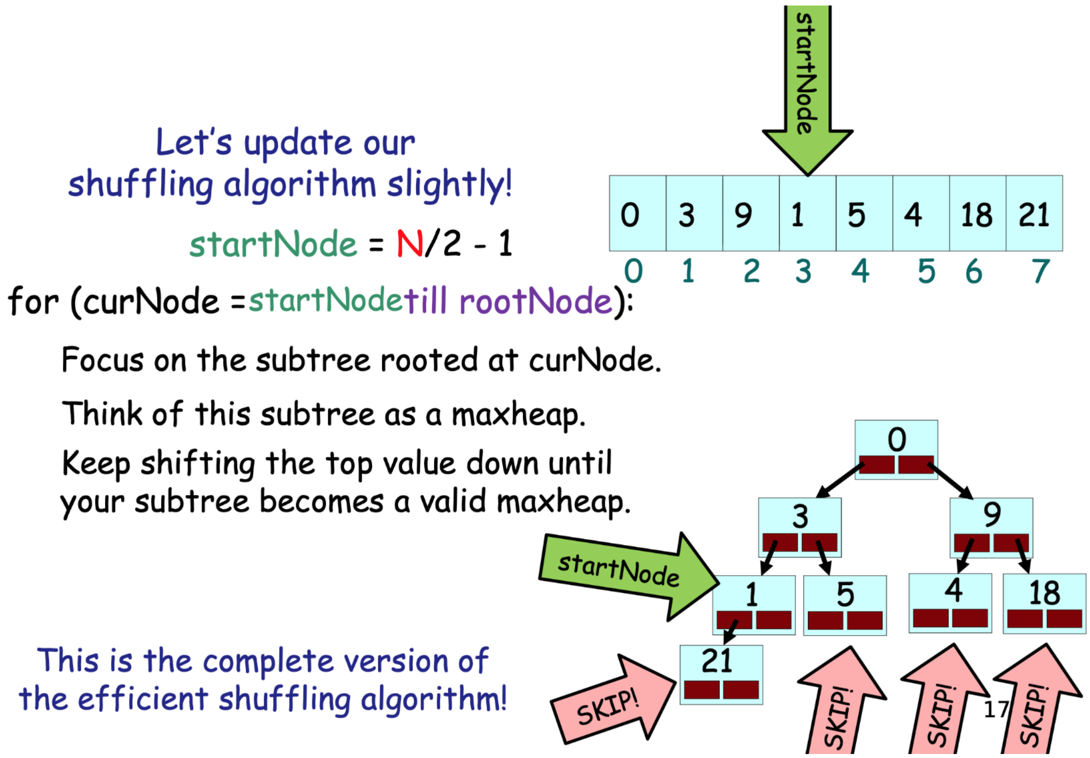
2. Compare the new value heap[-1] with its parent's value: heap[(len(heap)-1 -1)/2]

3. If the new value is greater than its parent's value, then swap them. (we don't need to care other nodes, why?)

4. Repeat steps 2-3 until the new value rises to its proper place or we reach the top of the array.

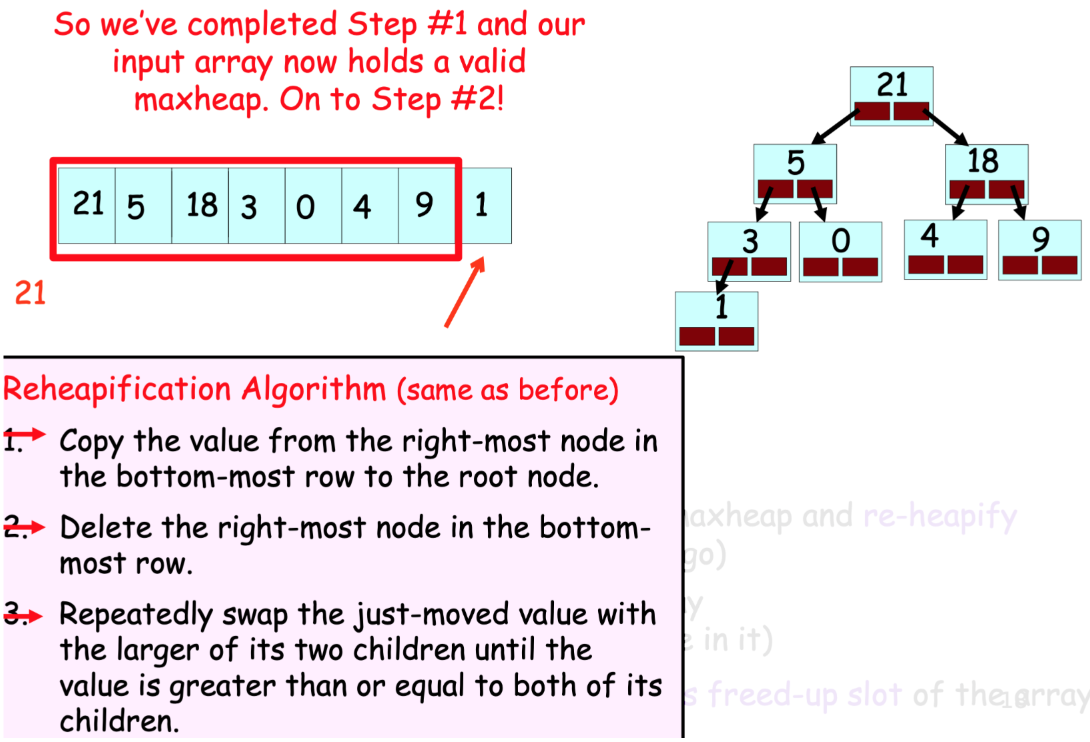
0	10
1	7
2	8
3	3
4	
	…

Hash Tables



Types

There are 2 types of hash tables:



Chaining

Chaining is a way of searching for an item:

1) Concepts:
   - closed hash table, open hash table, open addressing, linear probing, quadratic probing, double hashing, load factor, rehash, hash function.

2) Operations (status after operations) and time complexity
   - Hash table (search, insert, delete)
   - Solve collisions (separate chaining, open address)
   - Average steps for different hash tables (open addressing vs. chaining).

3) Implementation (writing code)
   - Closed hash table and open hash table (insertion/search/deletion).

4) Understand how to design a good hash function. Identify whether a given hash function is good or bad.

There are many schemes for dealing with collisions, and today we'll learn two of the most popular…

Closed Hash Table   Open Hash Table

Closed Hash Table: all elements are stored in the hash table itself

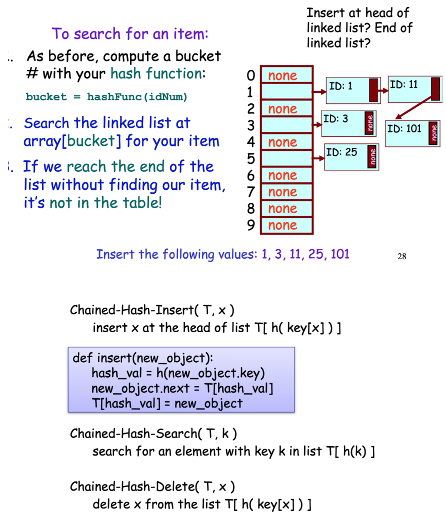
Closed Hashing = Open Addressing

Open Hashing = Separate chaining

Open addressing: Put everything into the table, but not necessarily into cell h(k).

Implementation

The following is a possible implementation of a hash table in python.



Collisions

collisions are when the hash table probes into a bucket that is already full, there are 2 ways we can avoid that:

- quadratic probing - using a quadratic function to dictate the sequence in which buckets are checked
- double hasing - using 2 hash functions in order to add more randomness to the initial probing and avoid collisions.