Crib for 4M26 Algorithms and Data Structures 2023/2024

Version: POK/3*

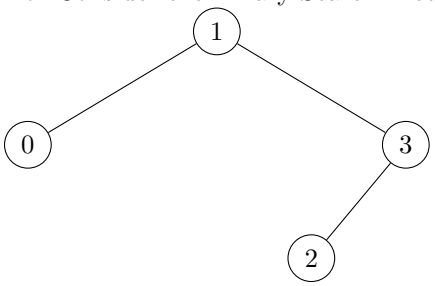
Question 1

a)

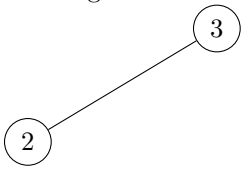
In `shift_nodes()`, we need to change
`elif old = old.parent.left:`
to
`elif old == old.parent.left:`
In `delete()`, we need to change
`succ = self.maximum(u.left)`
to
`succ = self.minimum(u.right)`
In `delete()`, we need to change
`elif not u.right:  self.shift_nodes(u, u.right)`
to
`elif not u.right:  self.shift_nodes(u, u.left)`
In `delete()`, we need to change
`self.shift_nodes(succ, succ.left)`
to
`self.shift_nodes(succ, succ.right)`
In `Node.__init__()`, we need to change
`self.left, self.right = right, left`
to
`self.left, self.right = left, right`

b)

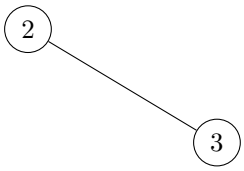
No. Consider the Binary Search Tree depicted below:



Deleting the node with key 0 then the node with key 1 produces:



Deleting the node with key 1 then the node with key 0 produces:



c)

- Perform an inorder traversal of the Binary Search Tree, returning the result as a list.
- Select the element in the kth position in the list.

d)

- Perform an inorder traversal of the Binary Search Tree, returning the result as a list of keys.
- Sort this list, and identify the two keys whose positions differ from the unsorted list.
- Perform a second traversal, swapping the two keys identified in the previous step.

e)

- Worst case complexity: $\Theta(n)$, example sequence: $k_n = n$
- Best case complexity: $\Theta(\log n)$. Example sequence: $k_n = \frac{2n+1}{2^{\lfloor \log n \rfloor}}$

Question 2

a)

Note: *full reference code for this question can be found here.*

In `partition()`, we need to change
`A[i], A[j] = A[i], A[j]`
to
`A[i], A[j] = A[j], A[i]`
In `partition()`, we need to change
`for j in range(high):`
to
`for j in range(low, high):`
In `partition()`, we need to change
`return pivot`
to
`return i`
In `quicksort()`, we need to change
`if low > high:`
to
`if low < high:`
In `quicksort()`, we need to change
`pivot = partition(A, high, low)`
to
`pivot = partition(A, low, high)`

b)

- The array in which every element is equal will induce worst-case $\Theta(n^2)$ comparisons.
- The array in ascending order will also induce worst-case $\Theta(n^2)$ comparisons.
- The array in descending order will also induce worst-case $\Theta(n^2)$ comparisons.

c)

- Sort the array using quicksort, heapsort (or some other algorithm with $O(n \log n)$ expected behaviour).
- Swap the elements in positions 1 and 2, 3 and 4, 5 and 6 etc.

d)

1.
   - Sort the list using quicksort.
   - Iterate once over the list, keeping track of the greatest difference seen so far between consecutive elements.
   - Report the maximum difference observed.
2.
   - Find the min and max of L, then create $n-1$ equal size buckets that span $[min, max]$
   - Distribute element into corresponding buckets, tracking the min and max in each bucket.
   - Loop over buckets and report biggest difference between max of one and min of next.

Question 3

a)(i)

It is possible to assign an arbitrary number of objects membership in B as even if the bit array has all bits set, a test for object membership will return true.

a)(ii)

It is not possible to remove object memership in B as unsetting a bit in B may affect other objects' memberships and there is no way to know whether this is the case in a standard Blooom filter. (An extension called a *Counting Bloom filter* makes this a possibility by replacing each bit with a small counter).

a)(iii)

Just like a hash table, objects may hash to the same value. Thus, different objects may hash to the same bits and thus there is the possibility of false positives.

b)

```
def add(self, item):
        for i in range(self.k):
            h = hash(item) % self.n
            self.bit_array[h] = True

def check(self, item):
```

```
for i in range(self.k):
    h = hash(item) % self.n
    if self.bit_array[h] == False:
        return False
return True
```

c)(i)

The probability of a bit being set is $\frac{1}{n}$ and hence the probability of it being unset is $1 - \frac{1}{n}$. The probability of it not being set by any of the k hash functions is then $\left(1 - \frac{1}{n}\right)^k$.
If we have assigned m memberships to objects, then the probability of a particular bit still being unset is $\left(1 - \frac{1}{n}\right)^{km}$. Hence, the probability that this bit is set is thus $1 - \left(1 - \frac{1}{n}\right)^{km}$.
The probability of a false positive means we compute k hashes for some object not assigned membership in B, which means all the corresponding k bits in B are already set. The probability p of this happening is the probability of a particular bit being set k times, hence $p = \left(1 - \left(1 - \frac{1}{n}\right)^{km}\right)^k$.

c)(ii)

Substitute the parameter values into the expression obtained in c(i) to find the probability of a false positive (≈ 0.025). The maximum nominal cost is five hashes and five lookups.

Question 4

a)

Find the pivot point, convert points to polar form, sort by angle in relation to the pivot point, push the three initial points to the stack, and then traverse the remaining points; while the test succeeds pop the stack, and thereafter push the active point. The stack will contain the points belonging to the convex hull. Full marks are awarded for explanations that describe each step and at least list the correct final values of the stack.

b)

If several points have the same angle in relation to the pivot point p_0, then a single point must be chosen. This can be done by calculating distances. Since all points have the same angle in relation to p_0, they lie on a ray and it is not necessary to use the L_2 metric for this purpose. For example, an L_0 or L_∞ metric is faster to calculate, primarily because they avoid a square root operation.

c)

The time complexity is $O(n \log n)$. The initial sorting by angle is $O(n \log n)$. The loop is linear time (not $O(n^2)$), as each point is only considered at most two times. Hence the time complexity of Graham's scan is dominated by the initial sort.

d)

Any monotonic function in $[0, \pi]$ will suffice as only the relative order matters. For this reason, simply using the slope of the line (or the cosine from the dot product) is sufficient.