Question 1)

(a)

Algorithm NodesDepth(T, n)

Input Tree T with n nodes

Output Depth of all the nodes in T

D 🡨 empty iterator of size n

Nodes 🡨 T.positions()

while Nodes.hasNext()

count 🡨 0

temp 🡨 Nodes.next()

if temp = T.root()

D.add(0)

do

count 🡨 count + 1

temp 🡨 parent(temp)

while (parent(temp) != T.root())

return D

//Time complexity O(n2) due to double looping.

(b)

Algorithm countFullNodes(t)

Input binary search tree with root t

Output the number of full nodes

count 🡨 0

D 🡨 new empty queue

Q.enqueue(t)

while !(Q.isEmpty())

temp 🡨 Q.dequeue()

if (temp.hasRight()) AND (temp.hasLeft())

count 🡨 count + 1

if temp.hasLeft() then

Q.enqueue(left(temp))

if temp.hasLeft() then

Q.enqueue(right(temp))

return count

Question 2)

* 8 nodes exist at depth 3
* add numbers from left to right from the bottom up in order of generation.
* connect the 4 bottom subtrees

A close up of a mans face

Description automatically generated

//Final Tree

10

19

24

27

39

35

16

20

15

7

19

12

5

3

11

//Heap order satisfied. Add more values and restore the heap order at every level until all elements are in the tree.

//Add More Values

24

19

10

7

35

39

16

20

3

5

15

11

12

19

39

19

24

15

10

7

5

35

39

19

16

20

12

//Restore Heap Order

10

7

24

15

19

5

19

35

16

20

12

Performing removeMin(6) operations:

n = 1

* remove 3 from the root node and replace with last node (24)
* swap 5 and 24 to restore heap order
* swap 7 and 24 to restore heap order
* swap 10 and 24 to restore heap order

n = 2 //(removeMin() called once again)

* swap 5 at the head with the current last node (19)
* swap 19 and it’s min. child 7 to restore heap order
* swap 19 and new min. child 10 to restore heap order

n = 3 //(removeMin() again)

* remove last node 24 and replace 7 in the head
* swap 24 and 10 to restore order
* swap 24 and 15 to restore

n = 4 (removeMin() called)

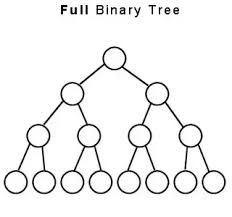
* last node 27 goes to root node
* swap 27 with 11 to restore order
* swap 27 with 11 to restore order
* swap 27 with 16 to restore order

n = 5 //removeMin()

* last node 39 replaces root node
* swap 39 with 12 to restore heap order
* swap 39 with new min. child 16 to restore order
* finally, swap 39 with new min. child 20 to restore order

n = 6 //(last call of removeMin())

* last node 35 replaces root node
* 35 swapped with right child 15
* 35 swapped with new left child 19



15

19

16

20

19

24

35

39

27

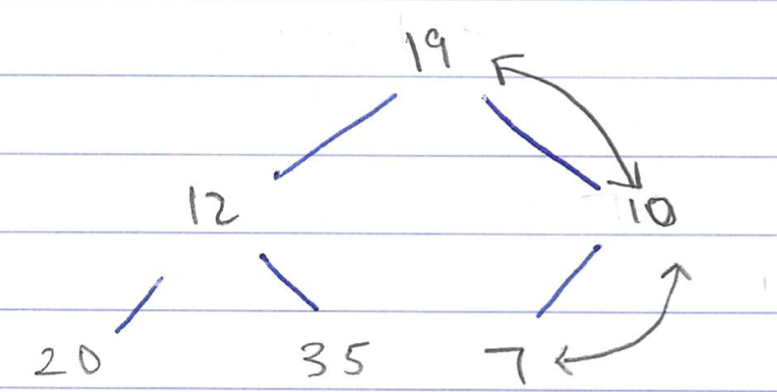
A picture containing photo, sitting, bunch, clock

Description automatically generatedA close up of a wire fence

Description automatically generatedA picture containing sitting, bunch, group, white

Description automatically generated(b)

1. Restore order then add elements 24 & 15
2. restore order and add element 5
3. restore heap order and add elements 39 & 16



1. Add elements from left to right

A close up of a map

Description automatically generatedA picture containing map, bunch, drawing, group

Description automatically generatedA picture containing outdoor, bunch, drawing, group

Description automatically generated

//Sorry for the written images. I ran out of time.-

1. restore order and get final tree
2. restore order and add 3, 11, 27 (final elements)
3. restore order and add 19

Question 3)

|  |  |  |  |
| --- | --- | --- | --- |
| Entry k | h(k) | Entry k | h(k) |
| **32** | 6 | **202** | 7 |
| **147** | 4 | **91** | 0 |
| **265** | 5 | **94** | 3 |
| **195** | 0 | **162** | 6 |
| **207** | 12 | **75** | 10 |
| **180** | 11 | **37** | 11 |
| **21** | 8 | **77** | 12 |
| **16** | 3 | **81** | 3 |
| **189** | 7 | **48** | 9 |

i)

195 - 91

|  |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |

207 - 77

180 - 37

75

48

21

189 - 202

32 - 162

147

265

16 - 94 - 81

ii) A total of 7 collisions occurred and were handled by adding colliding elements to a linked list, within the same index.

Question 4)

For better performance, a load factor below 1 is ideal. Increasing N to 15, it will reduce the load factor and reduce collisions. The statement has validity, although none of the two load factors are below 1.

Question 5)

open addressing hash table

N 🡪 double hashing

h(k) = [(k mod 19) + j ((7-k) mod 7] mod 19

from j = 0, 1, 2, …, 18

i)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Indexes** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** |
| Entries |  | 39 |  | 29 | 42 |  |  |  |  | 29 | 48 |  | 12 |  |  |  | 35/35 |  | 18 |

ii) longest cluster is 2

iii) 2 collisions occurred

iv) load factor = n/N = **9/19**

Question 6)

A close up of a map

Description automatically generatedi)

A close up of a map

Description automatically generatedii)

A close up of a map

Description automatically generated

iii)

A close up of a map

Description automatically generated

iv)

//Sorry for the written images. I ran out of time.

Programming Question Pseudocodes)

Algorithm setThreshold(Threshold)

allEntries 🡨 CVR.entrySet()

if n < Threshold then

get(vin) 🡨 getSeq(VIN)

put(VIN, ACClist) 🡨 putSeq(VIN, ACClist)

remove(VIN) 🡨 removeSeq(VIN)

else if n >= Threshold then

get(VIN) 🡨 getHT(VIN)

put(VIN, ACClist) 🡨 putHT(VIN, ACClist)

remove(VIN) 🡨 removeHT(VIN)

Algorithm getSeq(VIN)

allEntries 🡨 CVR.position() //Iterable collection of positions

while allEntrieshasNext() do

vehicRecs 🡨 allEnrties.next() //Sequence of the entries via linked list

if vehicRecs.element().getKey() = VIN

return vehicRecs.element().getValue()

return null;

Algorithm removeSeq(VIN)

allEntries 🡨 CVR.positoins()

while allEntries.hasNext() do

vehicRecs 🡨 allEntries.next()

if vehicRecs.element().getKey() = VIN

temp 🡨 vehicRecs.element().getValue()

allEntries.element().remove(temp)

n 🡨 n-1

return temp

return null;

Algorithm addSeq(VIN, ACClist)

allentries 🡨 CVR.positions()

while allEntries.hasNext() do

vehicRecs 🡨 allEntries.next()

if vehicRecs.element().getKey() = VIN

temp 🡨 vehicRecs.element().getValue()

allEntries.set(temp, (VIN, Acclist))

return temp

allEntries.addLast(VIN, Acclist)

n 🡨 n + 1

return null

Algorithm getHT(VIN)

i 🡨 h(VIN) //VIN into has function

travCount 🡨 0

do

cell 🡨 A[i] //accepts first full array

if cell = null then //empty cell: return null

return null

else if cell.getKey() = VIN //value matches: return element

return cell.getValue()

else //collision: perform double hashing

travcount 🡨 tranvCount + 1

i 🡨 (i + travCount \* d(VIN)) MOD N

while(travCount < N)

return null

Algorithm removeHT(VIN)

//N has some value such that n/N < 0.5

i 🡨 h(VIN)

travCount 🡨 0

do

cell 🡨 A[i]

if cell = null then

return null

else if cell.getKey() = VIN

temp🡨 cell.getValue()

cell.setValue(‘AVAILABLE’) //Replace with String to keep spositions intact

return temp

else

travcount 🡨 travCount + 1

i 🡨 (i + travCount \* d(VIN)) MOD N

while (travCount < N)

return null

Algorithm addHT(VIN, ACClist)

i 🡨 h(VIN)

travCount 🡨 0

do

cell 🡨 A[i]

if (cell = null) OR (cell = ‘AVAILABLE’)

cell 🡨 (VIN, ACClist) //String to consider removal method

return null //put method returns previous element

else if cell.getKey() = VIN

temp 🡨 cell.getValue()

cell.setValue(ACClist)

return temp

else

travCount 🡨 travCount + 1

i 🡨 (i + travCount \* d(VIN)) MOD N

while (travCount < N)

return null

HashCode

//VIN is 10 -17 characters alphanumeric characters; use polynomial accumulation

Hcode(VIN) = a0 + z(a1) + z2(a2) + z3(a3) + … + z16(a16) //where an = character at position n of VIN

comp(y) = (ay + b) MOD N //where a, b > 0 s.t (a MOD N) != 0

h(VIN) = comp(Hcode(VIN)) //This final function h incorporates both Hcode and compression to map the entries VIN

Algorithm allKeys()

//based on radix sort

VIN 🡨 CVR.keySet() //Iterble collection of VIN

for I 🡨 length -1 to 0 do

stableSort(allVINs, Ci) //Stable bucket sorting algorithm, where Ci is the ith character and allVINs is an iterable collection of all VINs

return allVINs

Algortihm prevAccids(VIN)

//Dates stored in the form YYMMDD to facilitate lexicographic ordering

S 🡨 getValue(VIN)

D 🡨 an empty double-ended queue

for i 🡨 7 to 0 do

stableSort(S, Ci)

while S.hasNext() do

D.addFirst(S.next())

while !(D.isEmpty)

S.add(D.removeFirst())

return D.removeFirst()

Algorithm setKeyLength(Length)

allKeys 🡨 CVR.key

while allKeys.hasNext() do

key 🡨 allKeys().next()

key.length() 🡨 Length

Algorithm getValues(Key)

vehicRecs 🡨 get(key)

return vehicRecs