Linked list 4
Friday, 3 May 2019 10:00

Q1) 3 → 4 → 7 → 2 → 6 → 7 → 6 → -3 → 8 → 7 → 10

Ans: h1 7 → 6 → 4 → 2 → 7 → 6 → 10 → 15

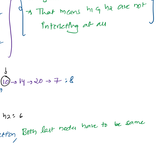
h2 3 → 7 → 10

Sol1: Goto to h1 last Node.
last Node → next = h2
We found a cycle now
We can go a step further 'f'
TC: O(N+M) SC: O(1)
cycle, there is nothing but Intersection

Issue: [They are not intersected to each other]
h1: 3 → 5 → 2 → 7 → 11
h2: 8 → 7 → 6 → 11 → 15 → 20
→ Cycle is not formed
→ That means h1 & h2 are not intersecting at all

Sol2: h1: 3 → 2 → 6 → 7 → 8 → 7 → 20 → 7 : 8
h2: 8 → 7

→ lengths of h1 : 8 , lengths of h2 : 6
If appear there is an intersection. Both first nodes have to be same



→ From node onwards we differ

h1: 3 → 2 → 6 → 7 → 8 → 7 → 20 → 7
h2: 8 → 7
→ Intersection point

Note: If at all there is no intersection h1 & h2 it will point NULL

h1: 5 → 2 → 4 → 6 → 10 → 9 → 6 → 7 → 8 → 7 → 4 → 19 : 12
h2: 6 → 7 → 7 → 8 → 7 → 4 → 19 : 12

h1	h2
12	7
11	6
10	5
9	4
6	3
7	2
6	1

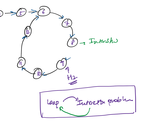
→ They are same
→ From node they cannot be moved to any node to the left
→ No if extra node are there (12-7)

Idea:
l1 = len(h1) l2 = len(h2) → O(N+M)
diff = abs(l1 - l2)
if(l1 > l2){
we move h1 by diff
}
else{
we move h2 by diff
}
while(h1 != NULL && h2 != NULL)
if(h1 == h2) & return h1
else{ h1 = h1 → next
h2 = h2 → next }
return NULL

→ No need

O(max(M, N))

3) Loop:



1) Find Intersection & slow & prev
2) h2 = Intersection → next
Intersection → next = NULL
The above problem became finding Intersection & LinkedList

Node* MergeRec(Node *h1, Node *h2){
if(h1 == NULL) return h2;
if(h2 == NULL) return h1;
if(h1 → data < h2 → data){
h1 → data = MergeRec(h1 → data, h2)
return h1;
}
else{
h2 → data = MergeRec(h1, h2 → data)
return h2;
}
}
TC: O(N+M)
SC: O(N+M)

Recursion

Ans: Given 2 sorted linked lists merge & return headnode & new LinkedList

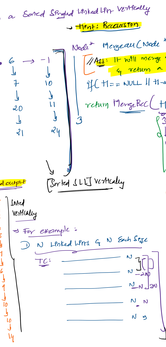
h1: 2 → 4 → 9 → 11 → 15
h2: 5 → 6 → 8 → 14 → 19 → 20
2 → 4 → 5 → 6 → 8 → 9 → 11 → 14 → 15 → 19 → 20

→ Merging

H3: Given head, sort it vertically

→ Hint: Recursion

Ans) Return a Sorted Singly linked list vertically



Node* MergeAll(Node *h3){
// Will merge all N linked lists & return a sorted node & Sorted LL (vertically)
if(h3 == NULL || h3 → next == NULL) return h3
return MergeRec(h3, MergeAll(h3 → next))
}

Node* MergeAll(Node *h3){
if(h3 == NULL || h3 → next == NULL) return h3
Node *mid = Mid(h3, NULL);
Node *h2 = mid → next
mid → next = NULL
return mergeRec(MergeAll(h3), MergeAll(h2))
}

How many nodes are there N^2
$K = N^2$
$T(K) = 2T(K/2) + K$
$T(K) = K \log K$; $K = N^2$
$T(\ ) = N^2 \log N^2$
$= 2N^2 \log N$
$= O(N^2 \log N)$

Ans: H1 3 5 7 9 11 13 5 10 15 25 2 11 20 28 6 7 9 21 -1 10 11 14

H2 3 5 7 9 11 13 2 3 4 6 3 10 19

No every 6 new nodes

Final output

Merge Rec

Merge Rec

Sorted LL (vertically)

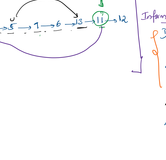
Idea: vertically
→ For example:
→ N linked lists & N each size

TC:

Time taken
$1 \times 2 = 2N$
$2 \times (1 \times 2) = 2N$
$3 \times (1 \times 2 \times 3) = 4N$
$4 \times (1 \times 2 \times 3 \times 4) = 2N$

Total iterations : $N(1+2+\dots N) = \frac{N}{2}$
$= N(N+1)/2 \approx N^2$
$= O(N^2)$

9)



Create a clone & return New head:

→ Sol 1 →
SC: O(N) SC: O(1)

Sol 1: 2 HashMaps
TC: O(N) SC: O(N)

Idea:
10 → 10 random (2): distance to a from start : 3
5 → random 13: distance to 13 from start : 7
11 → 2: distance to 2 from start : 3

Information needed
1) For a given node in H1 we need its distance from start
2) For a distance from start we need its node in H2

→ Iterate on H1 & construct <tmap1>
→ Iterate on H2 construct <tmap2>

// Say we have tmap1 & tmap2
// Say the also created, we need to find random nodes

tmap1 <Node*, dist> tmap2 <dist, Node*>
tmap <Node*, Node*>
→ H1 address
→ H2 address
Use comparator for H1 address & Ordered Map

while(H1 != NULL){
Node* tmp = H1 → random
int d = tmap1[tmp]
// getting dist from node from start
// using d we need to get node in H2
Node* tmp2 = tmap2[d];
H2 → random = tmp2
// getting node at a distance from a start
H1 = H1 → next
H2 = H2 → next
}

Sol 4 :

Step: Insert node to behind | Insert node to behind

Step 1

while(h1 != NULL){
h1 → next → random = h1 → random → next
h1 = h1 → next → next
}
h1, h2 = h1 = next
while(h1 != NULL){
h2 → next = h2 → next
h1 → next = h1 → next → next
h2 → next = h1 → next
h1 = h2 → next
}

Finally finished look to behind

Note: [Time to change that to go behind NULL]

Added condition reason below

Example:

A : 3 → 5 → 10 → 11

Step 1: Insert nodes to Behind

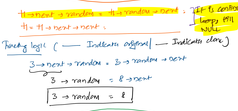
Step 2: Insert random

h1 → random = h1 → random → random : If h1 continue
h1 = h1 → next → next ;

Tracing logic (—— Indicate assign | — Indicate clone)

3 → next → random = 3 → random → next
3 → random = 8 → next
3 → random = 8

Step: 3



h1 → next = h2 → next
h1 = h1 → next
h2 → next = h1 → next
h2 = h2 → next

Tracing code:
h1 = 3 h2 = 3
After 1st iteration
h1 = 8 h2 = 8
After 2nd iteration
h1 = 10, h2 = 3
After 3rd iteration
h1 = 11, h2 = 11

So 4th iteration:
h1 → next = h2 → next
h1 → next = h1 → next
NULL
h1 = h1 → next
h1 → next = NULL
h2 → next = h2 → next
h2 → next = NULL

Because h2 is now at end we still get error.
If (h1 == NULL) stop below line
This line has to be added to above code