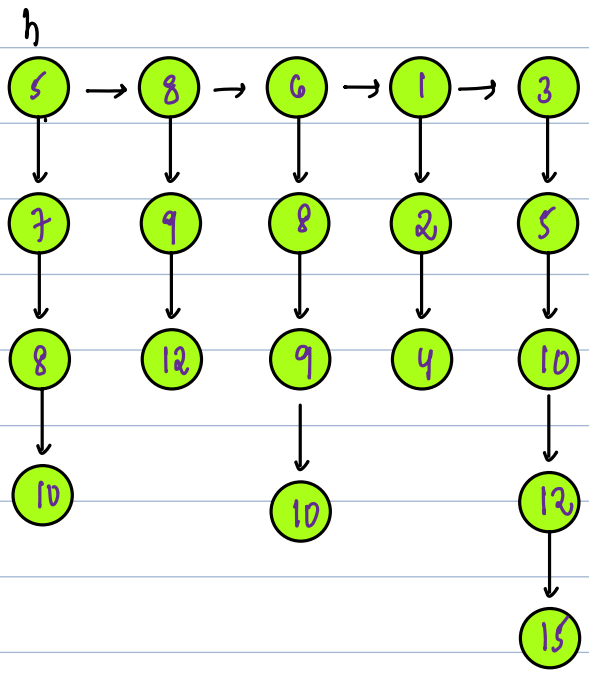
Todays Content

1. Flatten linked list : TODO similar to merge N Lists
2. Clone Linked List

In C++;

```
struct Node{
    int data
    struct Node *nent;
    struct Node *bottom;
    Node(int x){
        data = x;
        nent = NULL;
        bottom = NULL;
    }
}
```

Eg:

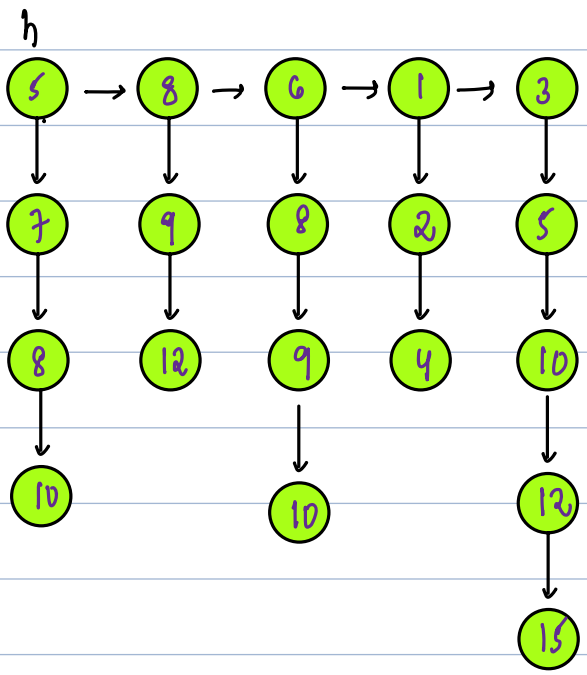


Ass:

```
Node* Flatten(Node *h) {
```

3

DryRun:



Clone Linked List:

Q. Given a linked list, where every node, next & rand,

Next: Holds address of next node of linked list

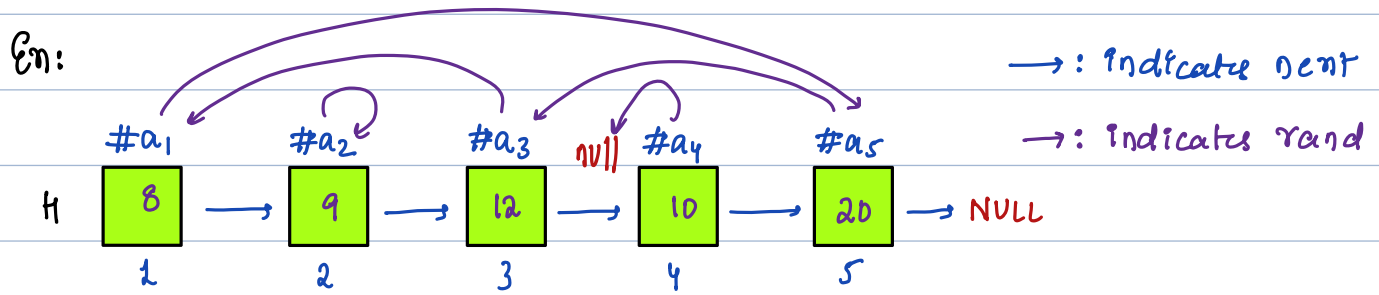
rand: Holds address of some random node of linked list

Create a exact copy of given linked list & return head Node of copy

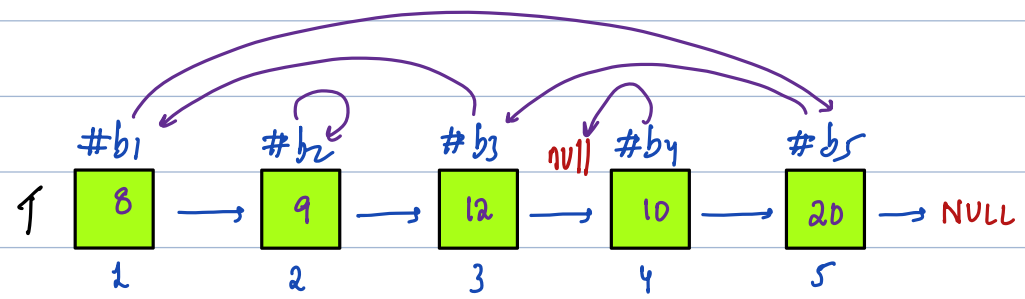
Copy means: Data should be same & overall structure should be same.

```
class Node {
    int data;
    Node next; // pointing to next node
    Node rand; // pointing to any node in linked list
    Node(int n) {
        data = n;
        next = NULL; rand = NULL;
    }
}
```

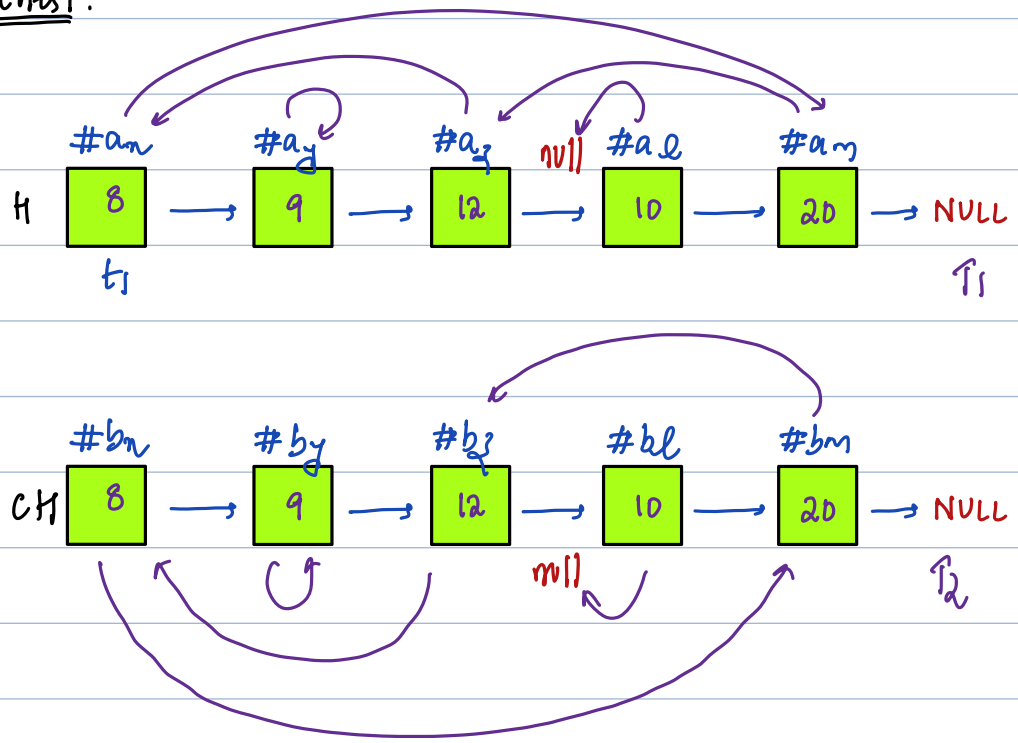
Ex:



Create Copy: { New copy won't come under extra space }



Approaches:



Ideal: TC: $O(N + N*\{N+N\}) = O(N^2)$ SC: $O(1)$

1. Create a copy of linked lists without random links. # Say head is CH

2. Node *T1 = H, *T2 = CH;
   while(T1 != null){
   Node *t1 = T1->random;
   if(t1 != null){

Can be optimize with HM < Node*, pos > in H

Iterate on H & calculate position of node t1 = p

Iterate on CH & calculate node at position p1 = t2

Can be optimized with HM < pos, Node* > in CH

   T2->rand = t2;
   }
   T1 = T1->next;
   T2 = T2->next;
   }

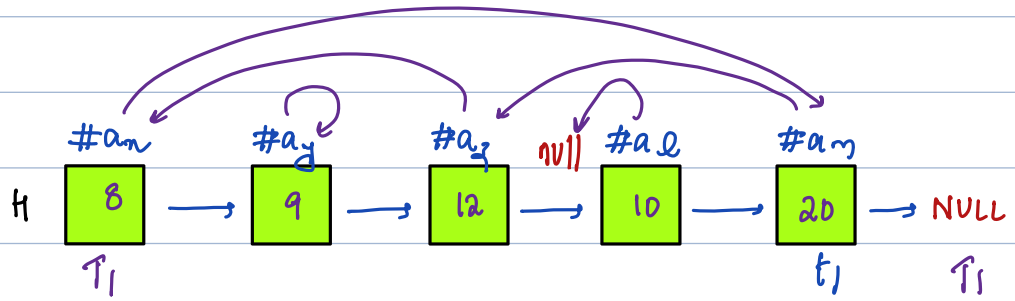
Idea2: TC: $O(N + N + N) = O(N)$ SC: $O(N + N) = O(N)$

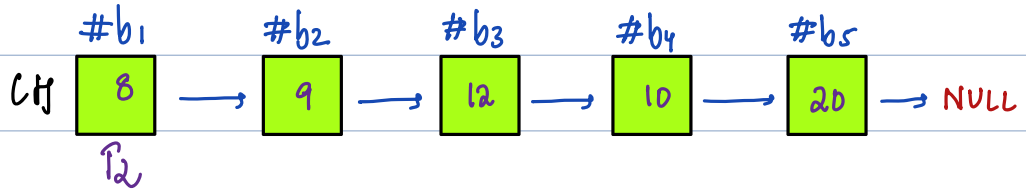
1. Create a copy of linked lists without random links. # Say head is H

2. Optimize with 2 HashMaps

HM1: For every node at H store it's <Node*, pos> #TODO Create

HM2: For every node at CH store it's <pos, Node*> #TODO Create





HM1

<an: 1>
<ay: 2>
<az: 3>
<al: 4>
<am: 5>

Copy random pointers

```
Node *T1 = H, *T2 = CH;
while(T1 != null){
    Node *t1 = T1->rand;
    if(t1 != null){
        Node *t1 = T1->rand;
        int p = HM1[t1]; #5
        Node *t2 = HM2[p]; #b5
        T2->rand = t2;
    }
    T1 = T1->next;
    T2 = T2->next;
}
```

t1 → p p → t2

Given t1 → t2 ?

HM2

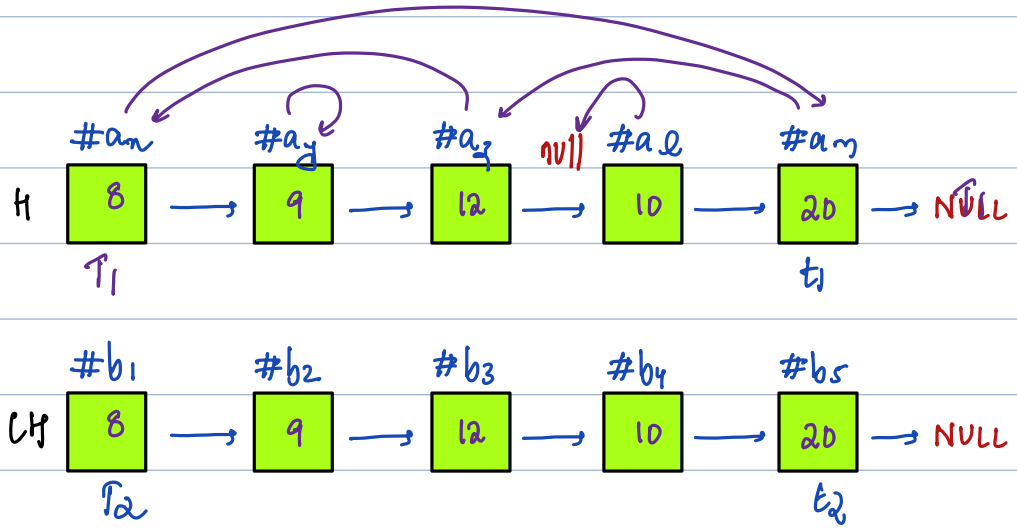
<1: b1>
<2: b2>
<3: b3>
<4: b4>
<5: b5>

Idea3: TC: $O(N + N + N) = O(N)$ SC: $O(N) = O(N)$

1. Create a copy of linked lists without random links. # Say head is CH

2. Optimize using hashmap

HM: For every node in H, it's respective position node in CH
< Node*, Node* > : TODO Code



HM:

a1	b1
a3	b3
a2	b2
a4	b4
a5	b5

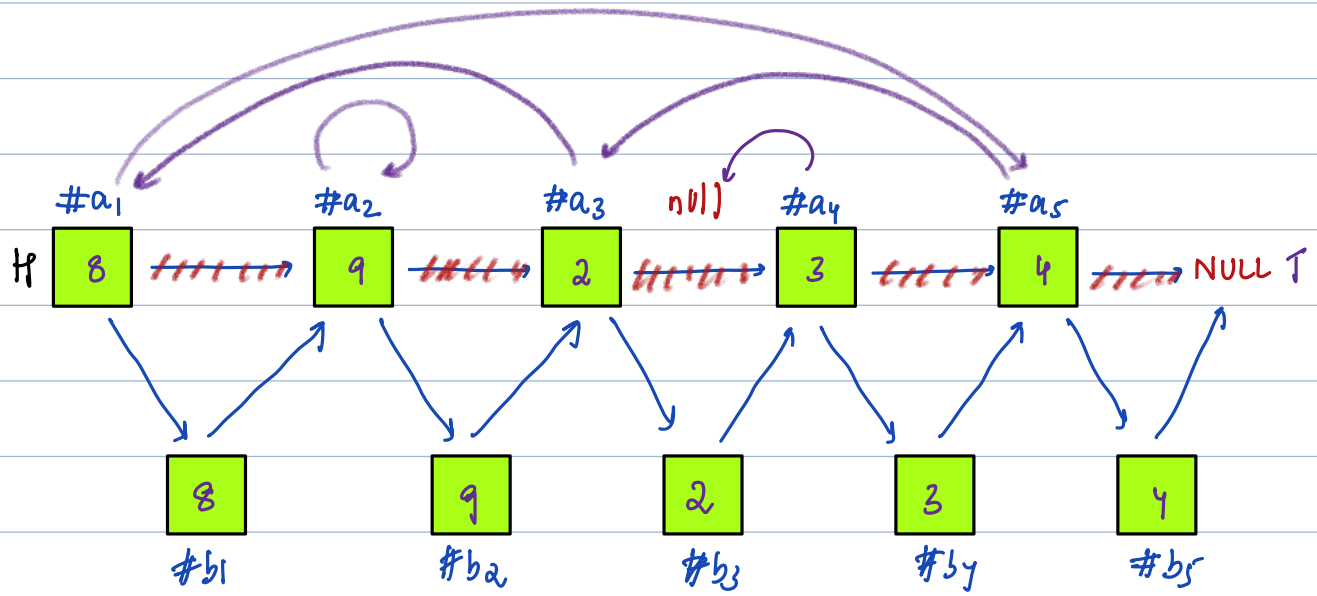
Copy random pointers

```
Node *T1 = H, *T2 = CH;
while(T1 != nullptr){
    Node *t1 = T1->random;
    if(t1->rand != nullptr){
        Node *t2 = HM[t1];
        T2->rand = t2;
    }
    T1 = T1->next;
    T2 = T2->next;
}
```

#Ideay: No Extra Space

Step1: For one nodes H:

Insert a new node, copy of T, betwn T & T->next

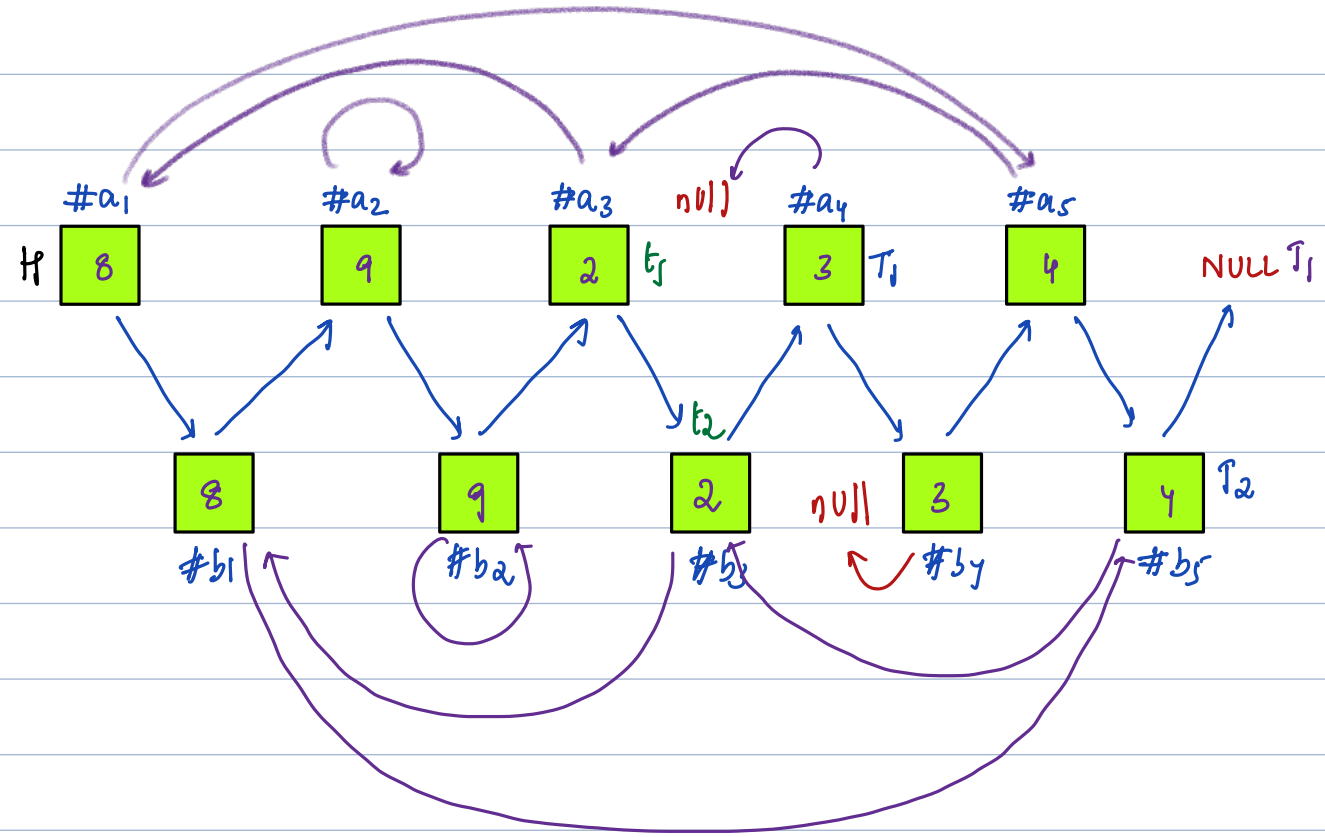


Step1:

```
Node *T = H;
while(T != nullptr){
    Node* nn = new Node(T->data);
    nn->next = T->next;
    T->next = nn;
    T = nn->next;
}
```

Step2: For all nodes of copy:

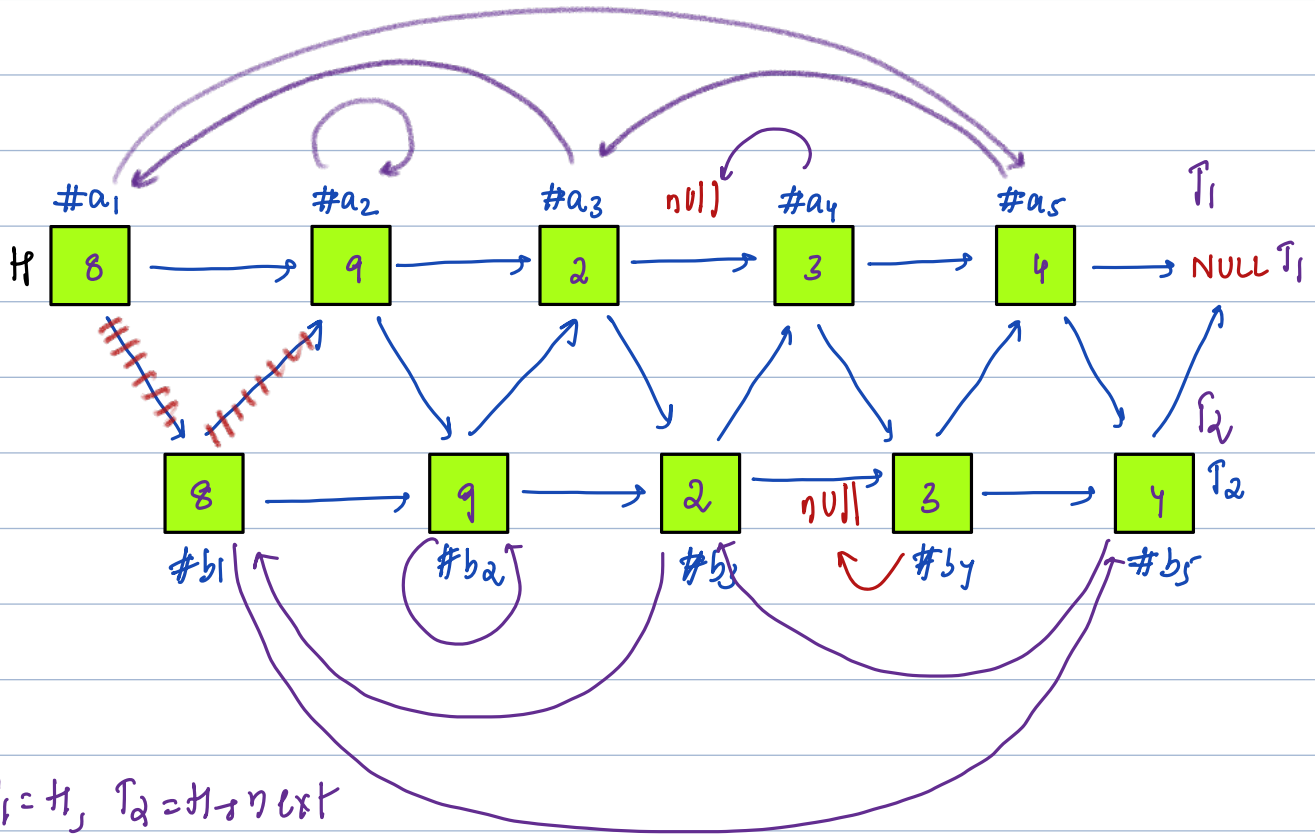
Arrange random Links:




```
Node *T1 = H
Node *T2 = H->next;
while (T1 != nullptr) {
    Node *t1 = T1->rand;
    if (t1 != nullptr) {
        Node *t2 = t1->next;
        T2->rand = t2;
    }
    T1 = T2->next;
    if (T1 != nullptr) {
        T2 = T1->next;
    }
}
```

Step3 : for all nodes in H & CH:

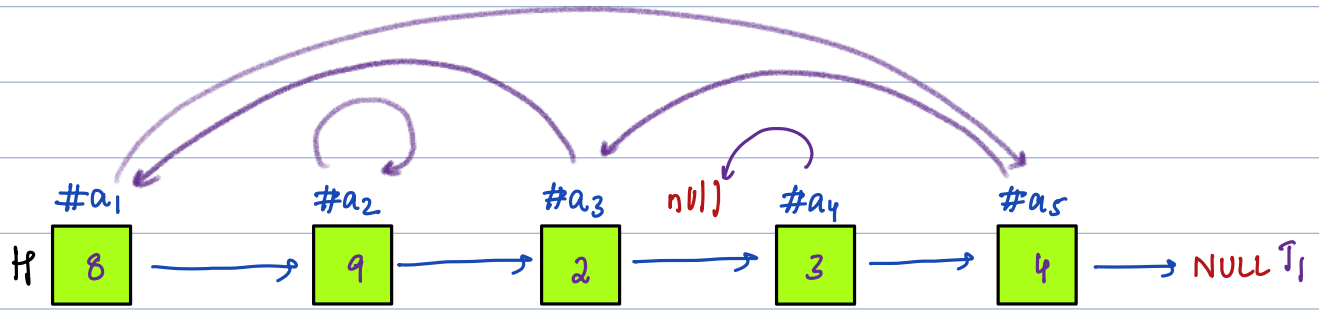
Arrange next links

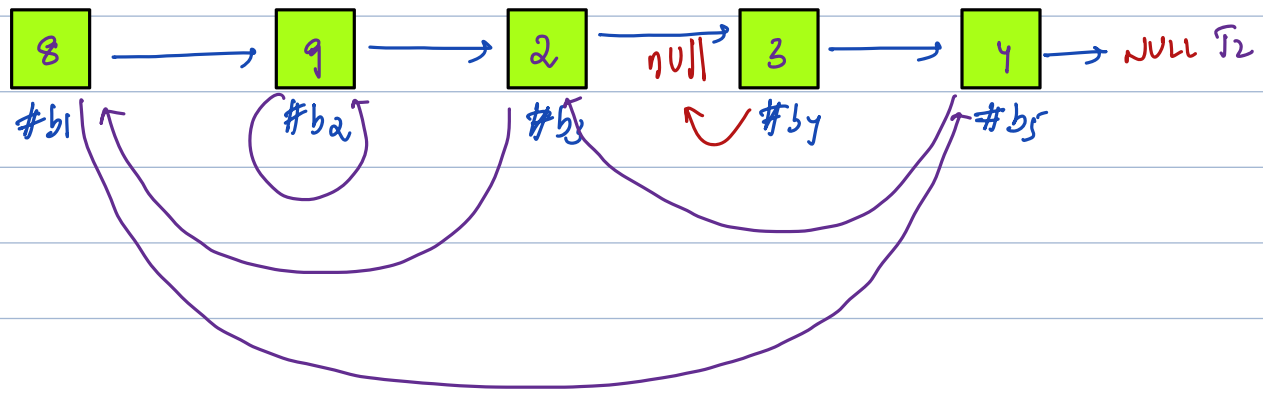


T1 = H, T2 = H->next

Node *CH = H->next; #fixed node of copy linked list

```
while( T1 != nullptr){
    T1->next = T2->next;
    T1 = T1->next;
    if( T1 != nullptr){
        T2->next = T1->next;
    }
    T2 = T2->next;
}
return CH;
```


#a1
#a2
#a3
null
#a4
#a5
8
9
2
3
4
NULL


8
9
2
null
3
4
NULL
#b1
#b2
#b3
#b4
#b5