Linked List Random Node

Given a singly linked list, return a random node's value from the linked list. Each node must have the **same probability** of being chosen.

Follow up:
What if the linked list is extremely large and its length is unknown to you? Could you solve this efficiently without using extra space?

Example:

```
// Init a singly linked list [1,2,3].
ListNode head = new ListNode(1);
head.next = new ListNode(2);
head.next.next = new ListNode(3);
Solution solution = new Solution(head);

// getRandom() should return either 1, 2, or 3 randomly. Each element should have e
qual probability of returning.
solution.getRandom();
```

Solution 1

Problem:

- Choose `k` entries from `n` numbers. Make sure each number is selected with the probability of `k/n`

Basic idea:

- Choose `1, 2, 3, ..., k` first and put them into the reservoir.
- For `k+1`, pick it with a probability of `k/(k+1)`, and randomly replace a number in the reservoir.
- For `k+i`, pick it with a probability of `k/(k+i)`, and randomly replace a number in the reservoir.
- Repeat until `k+i` reaches `n`

Proof:

- For `k+i`, the probability that it is selected and will replace a number in the reservoir is `k/(k+i)`
- For a number in the reservoir before (let's say `X`), the probability that it keeps staying in the reservoir is
  - `P(X was in the reservoir last time)` × `P(X is not replaced by k+i)`
  - = `P(X was in the reservoir last time)` × (`1` - `P(k+i is selected and replaces X)`)
  - = `k/(k+i-1)` × (`1` - `k/(k+i)` × `1/k`)
  - = `k/(k+i)`
- When `k+i` reaches `n`, the probability of each number staying in the reservoir is `k/n`

Example

- Choose `3` numbers from `[111, 222, 333, 444]`. Make sure each number is selected with a probability of `3/4`
- First, choose `[111, 222, 333]` as the initial reservior
- Then choose `444` with a probability of `3/4`
- For `111`, it stays with a probability of
  - `P(444 is not selected)` + `P(444 is selected but it replaces 222 or 333)`
  - = `1/4` + `3/4` * `2/3`
  - = `3/4`
- The same case with `222` and `333`
- Now all the numbers have the probability of `3/4` to be picked

This Problem <Linked List Random Node>

- This problem is the sp case where `k=1`

P.S. Thanks for @WKVictor for pointing out my mistake!

written by WTIFS original link here

Solution 2

```
import java.util.*;
public class Solution {

    /** @param head The linked list's head. Note that the head is guanranteed to
be not null, so it contains at least one node. */
    ListNode head = null;
    Random randomGenerator = null;
    public Solution(ListNode head) {
        this.head = head;
        this.randomGenerator = new Random();

    }

    /** Returns a random node's value. */
    public int getRandom() {
        ListNode result = null;
        ListNode current = head;

        for(int n = 1; current!=null; n++) {
            if (randomGenerator.nextInt(n) == 0) {
                result = current;
            }
            current = current.next;
        }

        return result.val;

    }
}

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(head);
 * int param_1 = obj.getRandom();
 */
```

written by abi93k original link here

Solution 3

according to the wiki https://en.wikipedia.org/wiki/Reservoir_sampling
here is sudo code for k size reservoir:

```
/*
  S has items to sample, R will contain the result
*/
ReservoirSample(S[1..n], R[1..k])
  // fill the reservoir array
  for i = 1 to k
      R[i] := S[i]

  // replace elements with gradually decreasing probability
  for i = k+1 to n
    j := random(1, i)   // important: inclusive range
    if j <= k
        R[j] := S[i]
```

you need to remember the range [0, i] should be inclusive.

```
class Solution {
private:
    ListNode* head;
public:
    /** @param head The linked list's head. Note that the head is guanranteed to
be not null, so it contains at least one node. */
    Solution(ListNode* head) {
        this->head = head;
    }

    /** Returns a random node's value. */
    int getRandom() {
        int res = head->val;
        ListNode* node = head->next;
        int i = 2;
        while(node){
            int j = rand()%i;
            if(j==0)
                res = node->val;
            i++;
            node = node->next;
        }
        return res;
    }
};
```

written by primbo original link here