

Select a Random Node from a Singly Linked List

Given a singly linked list, select a random node from linked list (the probability of picking a node should be $1/N$ if there are N nodes in list). You are given a random number generator.

Below is a Simple Solution

- 1) Count number of nodes by traversing the list.
- 2) Traverse the list again and select every node with probability $1/N$. The selection can be done by generating a random number from 0 to $N-i$ for i 'th node, and selecting the i 'th node only if generated number is equal to 0 (or any other fixed number from 0 to $N-i$).

We get uniform probabilities with above schemes.

```
i = 1, probability of selecting first node = 1/N
i = 2, probability of selecting second node =
    [probability that first node is not selected] *
    [probability that second node is selected]
    = ((N-1)/N) * 1/(N-1)
    = 1/N
```

Similarly, probabilities of other selecting other nodes is $1/N$

The above solution requires two traversals of linked list.

How to select a random node with only one traversal allowed?

The idea is to use [Reservoir Sampling](#). Following are the steps. This is a simpler version of [Reservoir Sampling](#) as we need to select only one key instead of k keys.

```
(1) Initialize result as first node
    result = head->key
(2) Initialize n = 2
(3) Now one by one consider all nodes from 2nd node onward.
    (3.a) Generate a random number from 0 to n-1.
           Let the generated random number is j.
    (3.b) If j is equal to 0 (we could choose other fixed number
           between 0 to n-1), then replace result with current node.
    (3.c) n = n+1
    (3.d) current = current->next
```

Below is the implementation of above algorithm.

C

```
/* C program to randomly select a node from a singly
   linked list */
#include<stdio.h>
#include<stdlib.h>
#include <time.h>

/* Link list node */
struct node
{
    int key;
    struct node* next;
};

// A reservoir sampling based function to print a
// random node from a linked list
void printRandom(struct node *head)
{
    // IF list is empty
    if (head == NULL)
        return;
```

```

// Use a different seed value so that we don't get
// same result each time we run this program
srand(time(NULL));

// Initialize result as first node
int result = head->key;

// Iterate from the (k+1)th element to nth element
struct node *current = head;
int n;
for (n=2; current!=NULL; n++)
{
    // change result with probability 1/n
    if (rand() % n == 0)
        result = current->key;

    // Move to next node
    current = current->next;
}

printf("Randomly selected key is %d\n", result);
}

/* BELOW FUNCTIONS ARE JUST UTILITY TO TEST */

/* A utility function to create a new node */
struct node *newNode(int new_key)
{
    /* allocate node */
    struct node* new_node =
        (struct node*) malloc(sizeof(struct node));

    /* put in the key */
    new_node->key = new_key;
    new_node->next = NULL;

    return new_node;
}

/* A utility function to insert a node at the beginning
of linked list */
void push(struct node** head_ref, int new_key)
{
    /* allocate node */
    struct node* new_node = new node;

    /* put in the key */
    new_node->key = new_key;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

// Driver program to test above functions
int main()
{
    struct node *head = NULL;
    push(&head, 5);
    push(&head, 20);
    push(&head, 4);
    push(&head, 3);
    push(&head, 30);

    printRandom(head);

    return 0;
}

```

```
}
```

Java

```
// Java program to select a random node from singly linked list

import java.util.*;

// Linked List Class
class LinkedList {

    static Node head; // head of list

    /* Node Class */
    static class Node {

        int data;
        Node next;

        // Constructor to create a new node
        Node(int d) {
            data = d;
            next = null;
        }
    }

    // A reservoir sampling based function to print a
    // random node from a linked list
    void printrandom(Node node) {

        // If list is empty
        if (node == null) {
            return;
        }

        // Use a different seed value so that we don't get
        // same result each time we run this program
        Math.abs(UUID.randomUUID().getMostSignificantBits());

        // Initialize result as first node
        int result = node.data;

        // Iterate from the (k+1)th element to nth element
        Node current = node;
        int n;
        for (n = 2; current != null; n++) {

            // change result with probability 1/n
            if (Math.random() % n == 0) {
                result = current.data;
            }

            // Move to next node
            current = current.next;
        }

        System.out.println("Randomly selected key is " + result);
    }

    // Driver program to test above functions
    public static void main(String[] args) {

        LinkedList list = new LinkedList();
        list.head = new Node(5);
        list.head.next = new Node(20);
        list.head.next.next = new Node(4);
        list.head.next.next.next = new Node(3);
        list.head.next.next.next.next = new Node(30);

        list.printrandom(head);
    }
}
```

```

    }
}

// This code has been contributed by Mayank Jaiswal

```

Python

```

# Python program to randomly select a node from singly
# linked list

import random

# Node class
class Node:

    # Constructor to initialize the node object
    def __init__(self, data):
        self.data= data
        self.next = None

class LinkedList:

    # Function to initialize head
    def __init__(self):
        self.head = None

    # A reservoir sampling based function to print a
    # random node from a linkd list
    def printRandom(self):

        # If list is empty
        if self.head is None:
            return

        # Use a different seed value so that we don't get
        # same result each time we run this program
        random.seed()

        # Initialize result as first node
        result = self.head.data

        # Iterate from the (k+1)th element nth element
        current = self.head
        n = 2
        while(current is not None):

            # change result with probability 1/n
            if (random.randrange(n) == 0 ):
                result = current.data

            # Move to next node
            current = current.next
            n += 1

        print "Randomly selected key is %d" %(result)

    # Function to insert a new node at the beginning
    def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node

    # Utility function to print the linked LinkedList
    def printList(self):
        temp = self.head
        while(temp):
            print temp.data,
            temp = temp.next

```

```
# Driver program to test above function
l1ist = LinkedList()
l1ist.push(5)
l1ist.push(20)
l1ist.push(4)
l1ist.push(3)
l1ist.push(30)
l1ist.printRandom()

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Note that the above program is based on outcome of a random function and may produce different output.

How does this work?

Let there be total N nodes in list. It is easier to understand from last node.

The probability that last node is result simply $1/N$ [For last or N'th node, we generate a random number between 0 to N-1 and make last node as result if the generated number is 0 (or any other fixed number)]

The probability that second last node is result should also be $1/N$.

```
The probability that the second last node is result
    = [Probability that the second last node replaces result] X
      [Probability that the last node doesn't replace the result]
    = [1 / (N-1)] * [(N-1)/N]
    = 1/N
```

Similarly we can show probability for 3rd last node and other nodes.