

AcWing 826. 单链表

一、题目描述

实现一个单链表，链表初始为空，支持三种操作：

向链表头插入一个数；删除第 k 个插入的数后面的数；在第 k 个插入的数后插入一个数。现在要对该链表进行 M 次操作，进行完所有操作后，从头到尾输出整个链表。

注意:题目中第 k 个插入的数并不是指当前链表的第 k 个数。例如操作过程中一共插入了 n 个数，则按照插入的时间顺序，这 n 个数依次为：第 1 个插入的数，第 2 个插入的数，...第 n 个插入的数。

输入格式 第一行包含整数 M ，表示操作次数。

接下来 M 行，每行包含一个操作命令，操作命令可能为以下几种：

H x ，表示向链表头插入一个数 x 。**D k** ，表示删除第 k 个插入的数后面的数（当 k 为 0 时，表示删除头结点）。**I k x** ，表示在第 k 个插入的数后面插入一个数 x （此操作中 k 均大于 0）。

输出格式 共一行，将整个链表从头到尾输出。

数据范围 $1 \leq M \leq 100000$ 所有操作保证合法。

输入样例：

```
10
H 9
I 1 1
D 1
D 0
H 6
I 3 6
I 4 5
I 4 5
I 3 4
D 6
```

输出样例：

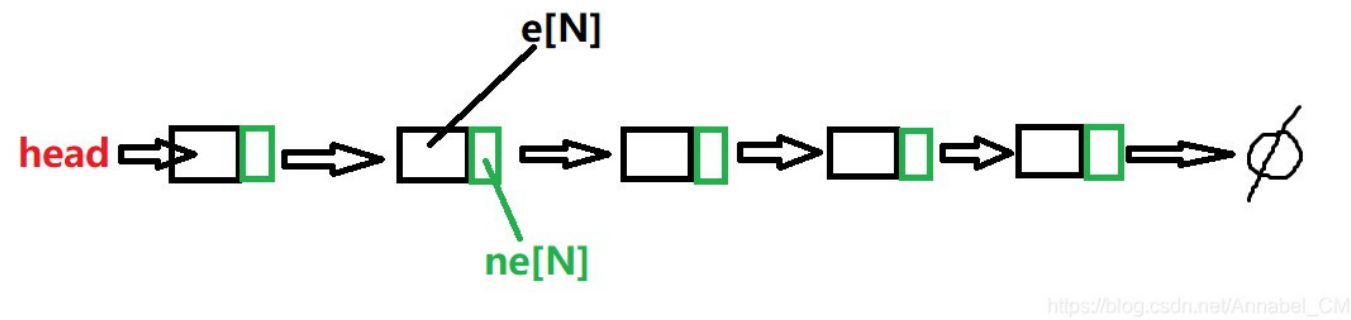
```
6 4 6 5
```

二、链表有什么用，用数组不就可以了吗？

数组是一种支持随机访问，但不支持在任意位置插入或删除元素的数据结构。与之相对应，**链表支持在任意位置插入或删除，但只能按顺序依次访问其中的元素。**

本节主要用数组模拟链表，速度更快，并且为静态存储。 $e[N]$ 存储节点的值， $ne[N]$ 存储该节点下一节点的下标。注意：下标从零开始，第 k 个数对应数组 $e[k - 1]$ ；在删除节点时，要判断该节点是否为头节点，若为头

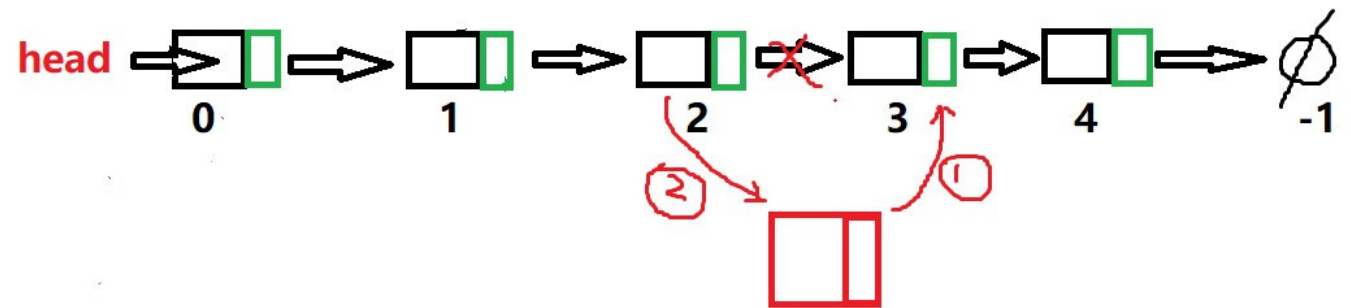
节点，直接 $head$ 指向该节点所指向的位置，即 $head = ne[head]$;



- 1、将链表初始化， $head$ 指向-1。
- 2、将 x 插到头结点。

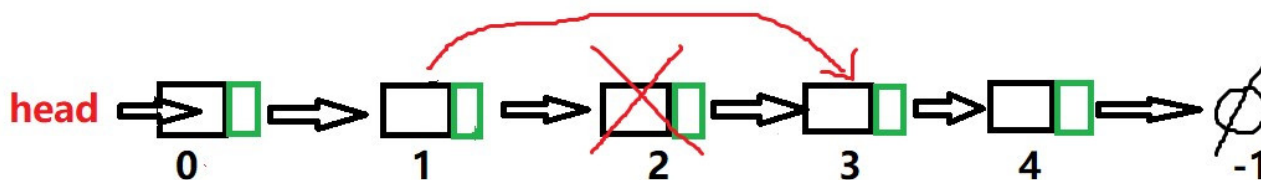


- 3、在第 k 个数后面插入 x



例如：将图中第三个数后面加上 x ，由于下标是从零开始，应为 $ne[k - 1]$;
首先将 x 所在下标指向原来第 k 个数所指向的位置
然后将第 k 个数的指针指向 x 所在位置

https://blog.csdn.net/Annabel_CM

4、删除第 k 个节点后面的数

例如将第2个节点后面的数删除

直接将第2个节点的指针指向其下一个节点所指向的位置

https://blog.csdn.net/Annabel_CM

三、C++ 代码

```
#include <bits/stdc++.h>

using namespace std;
const int N = 100010;

// 链表头: hh, 空链表时, hh =  $\phi$  用数字-1来表示 $\phi$ ; 非空链表时, hh指向链表的第一个结点
// 链表结点: ① 位置: idx ② 值: e[idx] , ③ 下一个结点的位置 : ne[idx]
// idx: 下一个可用结点编号
int e[N], ne[N], idx, hh = -1;

void add_to_head(int x) {
    e[idx] = x, ne[idx] = hh, hh = idx++;
}

void add(int k, int x) {
    e[idx] = x, ne[idx] = ne[k], ne[k] = idx++;
}

void remove(int k) {
    ne[k] = ne[ne[k]];
}

int main() {
    int m;
    cin >> m;
    while (m--) {
        int k, x;
        char op;
        cin >> op;
        if (op == 'H') { // 表示向链表头插入一个数 x
            cin >> x;
            add_to_head(x);
        } else if (op == 'D') { // 表示删除第 k 个插入的数后面的数
            cin >> k;
            if (k == 0)
                hh = ne[hh];
            else
                remove(k-1);
        }
    }
}
```

```

        remove(k - 1);
    } else { // 表示在第 k 个插入的数后面插入一个数 x
        cin >> k >> x;
        add(k - 1, x);
    }
}
for (int i = hh; ~i; i = ne[i]) printf("%d ", e[i]);
return 0;
}

```

链式前向星代码解析

```

#include <bits/stdc++.h>

using namespace std;

const int N = 10010, M = N << 1; // 最大顶点数,最大边数

/*
链式前向星:保存的是每一个顶点的所有出边

h[u] 表示点u为出发点的第一条边的编号
e[i] 表示第i条边的指向终点,有时也写作to[i]
ne[i] 表示第i条边的下一条边的编号
idx 记录边的数量,表现在代码中就是一个计数器
*/
int e[M], h[N], idx, w[M], ne[M];

// 添加一条从 a 到 b 的边,边权为c
void add(int a, int b, int c = 0) {
    // e[idx]=b      第idx号边,是指向点b的
    // ne[idx]=h[a]   第idx号边, 它的下一条边是原来h[a]指向的边
    // w[idx]=c       第idx号边, 权值是c
    // h[a]=idx++     以a点为出发点的所有边组成的链表中, 第一个是idx号边,头插法
    e[idx] = b, ne[idx] = h[a], w[idx] = c, h[a] = idx++;
}

int main() {
    // 初始化
    memset(h, -1, sizeof h);

    int n = 5; // 顶点数
    // 添加边
    add(0, 1);
    add(0, 2);
    add(1, 3);
    add(2, 3);
    add(3, 4);

    // 遍历从每个顶点出发的边
    for (int u = 0; u < n; u++) {

```

```
    cout << "从 " << u << "号节点引出的边: " << endl;

    for (int i = h[u]; ~i; i = ne[i]) { // 枚举从点u出发的每条边
        int v = e[i];
        // u 到点 v 的边
        cout << u << " -> " << v << endl;
    }
    cout << endl;
}

return 0;
}
```