**浙江智臾科技研发笔试题**

**请用熟悉的语言完成编程：**

**开始时间2022.10.28.23:16**

**结束时间2022.10.29.01:06**

1. 函数topRange接受1个数组v为参数，对v中的每一个元素，返回其为前连续多少个值内的最大值。每个元素的对应的仅仅与其之前的元素相关，与其之后的元素无关。返回一个与v相同长度的向量。请给出算法的复杂度，并且证明它。

示例：

topRange([13.5,13.6,13.4,13.3,13.5,13.9,13.1,20.1,20.2,20.3]

返回结果：[1,2,1,1,3,6,1,8,9,10]

解释：13.5为第一个值，其为连续1个值的最大值；13.6为子数组[13.5,13.6]的最大值，即为连续两个数的最大值；同理，最后的20.3是整个数组的最大值，故其为连续10个数的最大值。

#include <iostream>

#include <vector>

#include <stack>

int main()

{

    std::vector<double> v = {13.5, 13.6, 13.4, 13.3, 13.5, 13.9, 13.1, 20.1, 20.2, 20.3};

    uint32\_t n = v.size();

    std::vector<uint32\_t> maxArrLen(n);

    std::stack<uint32\_t> st;

    for (uint32\_t i = 0; i < n; i++)

    {

        while (!st.empty() && v[st.top()] <= v[i])

            st.pop();

        maxArrLen[i] = st.empty() ? i + 1 : i - st.top();

        st.push(i);

    }

    for (uint32\_t i = 0; i < n; i++)

        std::cout << maxArrLen[i] << " ";

    return 0;

}

算法复杂度为O（n）。因为每个元素最多只进入栈一次，最多只出栈一次，for循环的时间复杂度为O(n)。

1. 小明新买了一台有两条内存条的电脑。这台电脑的在分配内存的时候，总是从可用内存最多的内存条来分配内存，若两条内存条可用内存一样，则从第一条内存条中分配内存。如果两条内存条都没有足够内存了，就会报Out Of Memory(OOM)。

小明刚拿到新电脑，非常兴奋，赶紧写了一个hello world的程序，但是小明很快发现他的程序里有一个内存泄漏。具体地说，他的程序在第i秒的时候会申请i字节的内存。小明想知道他启动程序多久之后会OOM，以及OOM的时候两条内存条各剩下多少内存。你能帮帮他吗？

输入为M1和M2，分别代表第一条内存条和第二条内存条的总内存容量，单位是字节。输出为3个值t, m1和m2。其中t代表电脑会在第t秒报OOM，m1和m2代表OOM的时候第一条内存条和第二条内存条还剩下多少内存。其中1 <= M1 <= 10^18, 1 <= M2 <= 10^18 （提示：这意味着你的算法的复杂度不能超过log(M1+M2)）

示例1：

输入：2, 2

输出：3，1，0

解释：

|  |  |  |
| --- | --- | --- |
| 时间 | 第一条内存条可用内存 | 第二条内存条可用内存 |
| 0s | 2 | 2 |
| 1s | 1 | 2 |
| 2s | 1 | 0 |
| 3s | OOM | OOM |

示例2：

输入：8, 11

输出：6，0，4

解释：

|  |  |  |
| --- | --- | --- |
| 时间 | 第一条内存条可用内存 | 第二条内存条可用内存 |
| 0s | 8 | 11 |
| 1s | 8 | 10 |
| 2s | 8 | 8 |
| 3s | 5 | 8 |
| 4s | 5 | 4 |
| 5s | 0 | 4 |
| 6s | OOM | OOM |

#include <iostream>

#include <cmath>

int main()

{

    unsigned long long M1, M2, m1, m2, t = 0, n;

    std::cin >> M1 >> M2;

    m1 = M1;

    m2 = M2;

    if (m2 > m1)

    {

        unsigned long long out = m2 - m1; // n(1+n)=2out

        n = (unsigned long long)(std::sqrt(1 + 8 \* out) - 1) / 2;

        unsigned long long c = (1 + n) \* n / 2;

        m2 = m2 - c;

        t = n + 1;

        if (c < out && m2 >= n + 1)

        {

            m2 = m2 - (n + 1);

            t++;

        }

    }

    else

    {

        unsigned long long out = m1 - m2;

        n = (unsigned long long)(std::sqrt(1 + 8 \* out) - 1) / 2;

        unsigned long long c = (1 + n) \* n / 2;

        m1 = m1 - c;

        t = n + 1;

    }

    // 接下来轮流地

    // 从m1上申请t、t+2、t+4...t+2n个字节, t(n+1)+(2+2n)n/2=tn+t+n(1+n)=n\*n+(t+1)n+t-m1<=0

    // 从m2上申请t+1、t+3、t+5...、t+2n-1或t+2n+1个字节

    if (m1 >= t)

    {

        n = (std::sqrt((t + 1) \* (t + 1) + 2 \* t + 1 + 4 \* m1 - 4 \* t) - t - 1) / 2;

        m1 = m1 - (n \* n + (t + 1) \* n + t);

        m2 = m2 - (n \* n + (t + 1) \* n + t + n + 1 - (t + 2 \* n + 1)); // Bug：m2是无符号数，不能提前多减一个数再判断是否小于0再加上多减去的数

        if (m2 >= t + 2 \* n + 1)

        {

            m2 -= t + 2 \* n + 1;

            t += 2 \* n + 2;

        }

        else

            t += 2 \* n + 1;

    }

    std::cout << "Answer: " << t << ", " << m1 << ", " << m2 << "\n";

    // 验证程序

    m1 = M1, m2 = M2, t = 0;

    while (true)

    {

        if (m1 >= t && m2 >= t)

        {

            if (m1 < m2)

                m2 -= t;

            else

                m1 -= t;

        }

        else if (m1 >= t)

            m1 -= t;

        else if (m2 >= t)

            m2 -= t;

        else

            break;

        t++;

    }

    std::cout << "GroundTruth: " << t << ", " << m1 << ", " << m2 << "\n";

    return 0;

}

// 1000000000000000000 99999999999999999

// Answer: 1483239697, 646162599, 717131344

// GroundTruth: 1483239697, 646162599, 717131344

只有一些if判断、加减乘除和开根号，时间复杂度可以看成O(1)，如果按二进制位数来算时间复杂度是O(log(M1+M2))。

3、函数cumrank接受1个数组v为参数，对v中的每一个元素，返回其在从第一个元素到此元素为止的向量中的排序。每个元素的排序仅仅与其之前的元素相关，与其之后的元素无关。返回一个与v相同长度的的向量。请给出算法的复杂度。

示例：cumrank([1,3,2,3,4]) 返回结果： [0,1,1,2,4]

#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

inline int lowBit(int x)

{

    return x & -x;

}

int main()

{

    std::vector<int32\_t> v = {1, 3, 2, 3, 4};

    uint32\_t n = v.size();

    std::vector<int32\_t> r(n), a = v, t = v;

    // 离散化

    std::sort(t.begin(), t.end());

    for (int &num : a)

    {

        num = std::lower\_bound(t.begin(), t.end(), num) - t.begin() + 1;

    }

    std::vector<int32\_t> tree(n + 1, 0);

    for (uint32\_t i = 0; i < n; i++)

    {

        int32\_t q = a[i];

        q -= 1; // 不包括本身

        while (q)

        {

            r[i] += tree[q];

            q -= lowBit(q);

        }

        q = a[i];

        while (q < tree.size())

        {

            tree[q] += 1;

            q += lowBit(q);

        }

    }

    for (auto &i : r)

        std::cout << i << " ";

    return 0;

}

时间复杂度为O(nlogn)。