exam

Exam

2009_code_lastK

【2009 统考真题】已知一个带有表头结点的单链表，结点结构为

data	link

假设该链表只给出了头指针 list。在不改变链表的前提下，请设计一个尽可能高效的算法，查找链表中倒数第 k 个位置上的结点（k 为正整数）。若查找成功，算法输出该结点的 data 域的值，并返回 1；否则，只返回 0。要求：

1）描述算法的基本设计思想。

2）描述算法的详细实现步骤。

3）根据设计思想和实现步骤，采用程序设计语言描述算法（使用 C、C++或 Java 语言实现），关键之处请给出简要注释。

```
#include <iostream>

struct Node {
    int data;
    Node* link;
};

void create(Node* p, int a[], int pos, int length) {
    if (pos < length) {
        p->data = a[pos];
        p->link = new Node;
        p->link->data = 0; // 用0作为结束标志
        create(p->link, a, pos + 1, length);
    }
}

int getLastK(Node* p, int k) {
    Node* p_fast = p, * p_slow = p;
    while (--k && p_fast->link != NULL) p_fast = p_fast->link;
    p_fast = p_fast->link;
    while (p_fast->data) {
        p_slow = p_slow->link;
        p_fast = p_fast->link;
    }
    return p_slow->data;
```

```
}

int main() {
    Node p;
    int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    create(&p, a, 0, 10);
    int ret = getLastK(&p, 7);
    cout << "倒数第7个元素是: " << ret << endl;
    return 0;
}
```

提示：算法程序题，如果能够写出数据结构类型定义、正确的算法思想都会至少给一半以上分数，如果能用伪代码写出自然更好，比较复杂的地方可以直接用文字表达。

若所给出的算法采用一遍扫描方式就能得到正确结果，可给满分 15 分；若采用两遍或多遍扫描才能得到正确结果的，最高分 10 分。若采用递归算法得到正确结果的，最高给 10 分；若实现算法的空间复杂度过高（使用了大小与 k 有关的辅助数组），但结果正确，最高给 10 分。

太复杂了，不要那么模块化，因为我是考试写的，因此，只需要必要的数据结构定义，然后一个 main 汉书就可以了，确保语法 c++ 标准，结构用结构体，不要清理内存，假设是已经有了数据结构

2010_code_circle_leftmove

【2010 统考真题】设将 n（$n>1$）个整数存放到一维数组 R 中。设计一个在时间和空间两方面都尽可能高效的算法。将 R 中保存的序列循环左移 p（$0<p<n$）个位置，即将 R 中的数据由$(X_0, X_1, \cdots, X_{n-1})$变换为$(X_p, X_{p+1}, \cdots, X_{n-1}, X_0, X_1, \cdots, X_{p-1})$。要求：

1）给出算法的基本设计思想。

2）根据设计思想，采用 C 或 C++或 Java 语言描述算法，关键之处给出注释。

3）说明你所设计算法的时间复杂度和空间复杂度。

```
#include <iostream>
#include <vector>

void reverse(std::vector<int>& a, int start, int end) {
    while (start < end) {
        std::swap(a[start], a[end]);
        ++start;
        --end;
    }
}
```

```
void move(std::vector<int>& a, int p) {
    reverse(a, 0, p - 1);
    reverse(a, p, a.size() - 1);
    reverse(a, 0, a.size() - 1);
}

int main() {
    std::vector<int> a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    move(a, 7);

    for (int num : a) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

2011_code_double_divide

```
#include <iostream>
#include <vector>

int getMiddle(const std::vector<int>& a, int start, int end) {
    return a[(start + end) / 2];
}

int getDoubleMid(const std::vector<int>& s1, const std::vector<int>&
s2) {
    int length = s1.size();
    int i = 0, log_i = 1;
    while (log_i * 2 < length) { i++; log_i *= 2; }
    int start_a = 0, start_b = 0, end_a = length - 1, end_b = length
- 1;

    while (i-- && start_a < end_a && start_b < end_b) {
        int mid_a = getMiddle(s1, start_a, end_a);
        int mid_b = getMiddle(s2, start_b, end_b);
        if (mid_a < mid_b) {
            start_a = (start_a + end_a) / 2 + 1;
            end_b = (start_b + end_b) / 2 - 1;
        } else {
            start_b = (start_b + end_b) / 2 + 1;
            end_a = (start_a + end_a) / 2 - 1;
        }
```

```
    }
    return std::min(s1[start_a], s2[start_b]);
}

int main() {
    std::vector<int> s1 = {1, 2, 3, 8, 9};
    std::vector<int> s2 = {4, 5, 6, 7, 10};
    int ret = getDoubleMid(s1, s2);
    std::cout << "Result: " << ret << std::endl;
    return 0;
}
```

2012_code_doublelink

```
#include <iostream>

struct Node {
    char data;
    Node* link;
};

void create(Node* p, const std::string& a, int pos) {
    if (pos < a.length()) {
        p->data = a[pos];
        p->link = new Node;
        p->link->data = '\0';
        create(p->link, a, pos + 1);
    }
}

Node* getSameNode(Node* a, Node* b) {
    int a_length = 0, b_length = 0;
    Node* a_p = a->link, * b_p = b->link;
    while (a_p->data != '\0') { a_length++; a_p = a_p->link; }
    while (b_p->data != '\0') { b_length++; b_p = b_p->link; }

    Node* a_scan = a->link, * b_scan = b->link;
    while (a_length < b_length) { b_scan = b_scan->link; b_length--;
}
    while (b_length < a_length) { a_scan = a_scan->link; a_length--;
}

    while (a_length-- > 0) {
        if (a_scan == b_scan) return a_scan;
        a_scan = a_scan->link;
```

```
        b_scan = b_scan->link;
    }
    return nullptr;
}

int main() {
    Node a, b;
    create(&a, "loading", 0);
    create(&b, "being", 0);

    Node* tmp = b.link->link->link;
    b.link->link = a.link->link->link->link;
    delete tmp;

    Node a_with_head, b_with_head;
    a_with_head.link = &a;
    b_with_head.link = &b;
    Node* same = getSameNode(&a_with_head, &b_with_head);

    if (same) {
        std::cout << "Same node found: " << same->data << std::endl;
    } else {
        std::cout << "No same node found" << std::endl;
    }

    return 0;
}
```

2013_code_mian_ele

```
#include <iostream>
#include <vector>

bool majority(const std::vector<int>& A) {
    int start = 0, count = 0;
    for (int j = 0; j < A.size(); j++) {
        if (A[start] == A[j]) count++;
        else count--;
        if (count == 0 && j < A.size() - 1) {
            start = ++j;
            count = 1;
        }
    }

    int mainElement = A[start];
```

```
    int countElement = 0;
    for (int num : A) {
        if (num == mainElement) countElement++;
    }
    return countElement > A.size() / 2;
}

int main() {
    std::vector<int> A = {1, 1, 0, 0, 0, 1, 1, 1, 0};
    bool isMajority = majority(A);
    if (isMajority) {
        std::cout << "有主元素" << std::endl;
    } else {
        std::cout << "没有主元素" << std::endl;
    }
    return 0;
}
```

2014_code_getwpl

```
#include <iostream>
#include <vector>

struct TreeNode {
    int weight;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int w) : weight(w), left(nullptr), right(nullptr) {}
};

void create(TreeNode* node, const std::vector<int>& c, int pos) {
    if (pos < c.size()) {
        node->weight = c[pos];
        if (pos * 2 + 1 < c.size()) {
            node->left = new TreeNode(0);
            create(node->left, c, pos * 2 + 1);
        }
        if (pos * 2 + 2 < c.size()) {
            node->right = new TreeNode(0);
            create(node->right, c, pos * 2 + 2);
        }
    }
}

int getWPL(TreeNode* root, int depth) {
```

```
    if (root == nullptr) return 0;
    if (root->left == nullptr && root->right == nullptr) {
        return depth * root->weight;
    }
    return getWPL(root->left, depth + 1) + getWPL(root->right, depth
+ 1);
}

int main() {
    std::vector<int> c = {54, 4654, 7565, 234, 436546, 876876, 353,
757, 2, 345, 23, 445, 1, 235, 0, 6346};
    TreeNode* root = new TreeNode(0);
    create(root, c, 0);
    int wpl = getWPL(root, 0);
    std::cout << "WPL: " << wpl << std::endl;
    return 0;
}
```