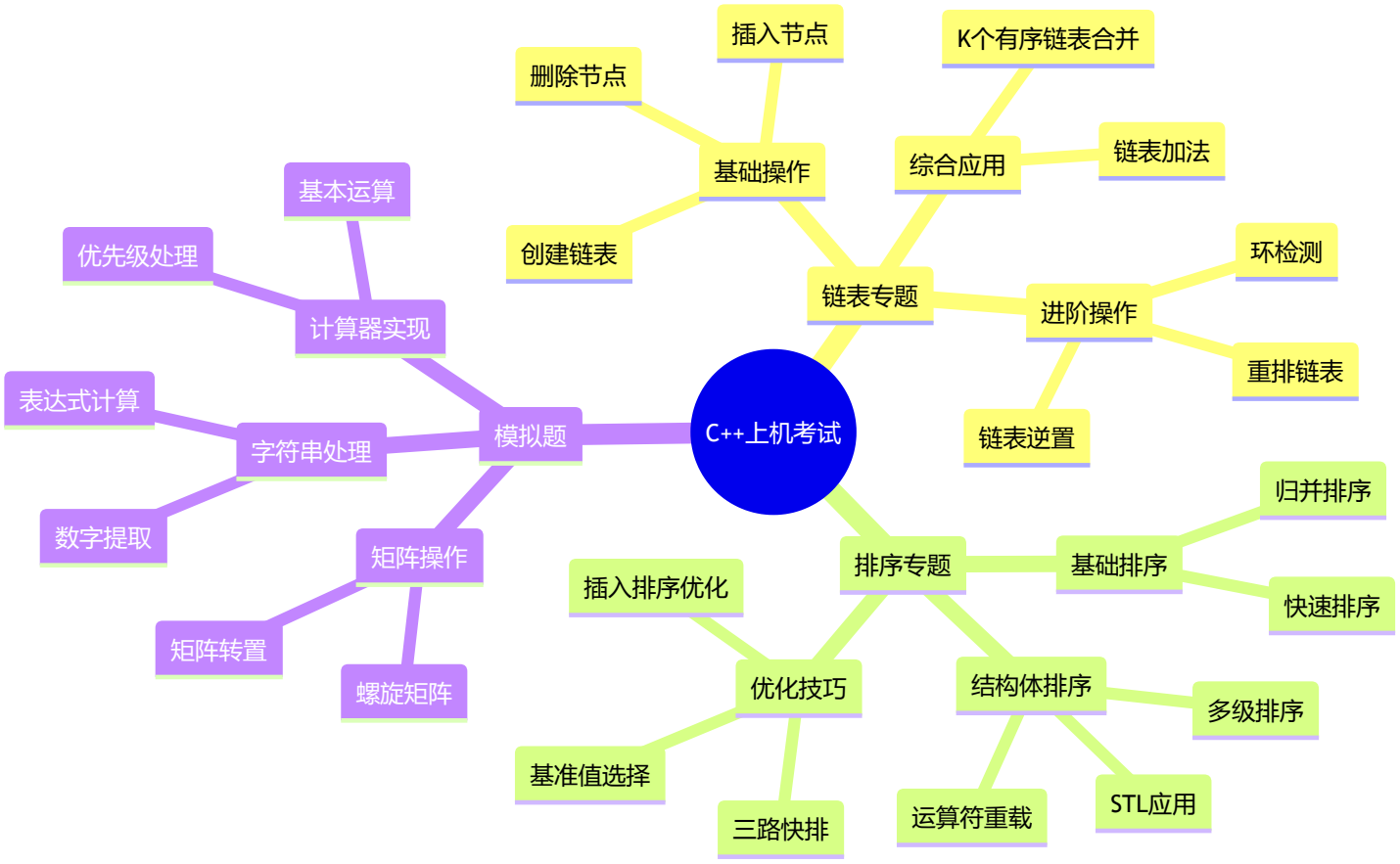
C++ 期中上机复习笔记

目录

知识体系

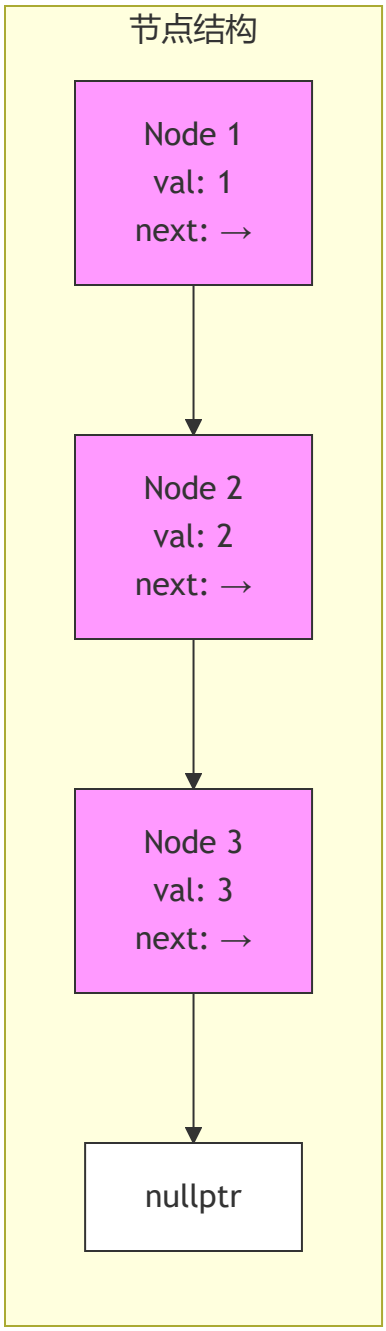


链表专题

基础概念与结构

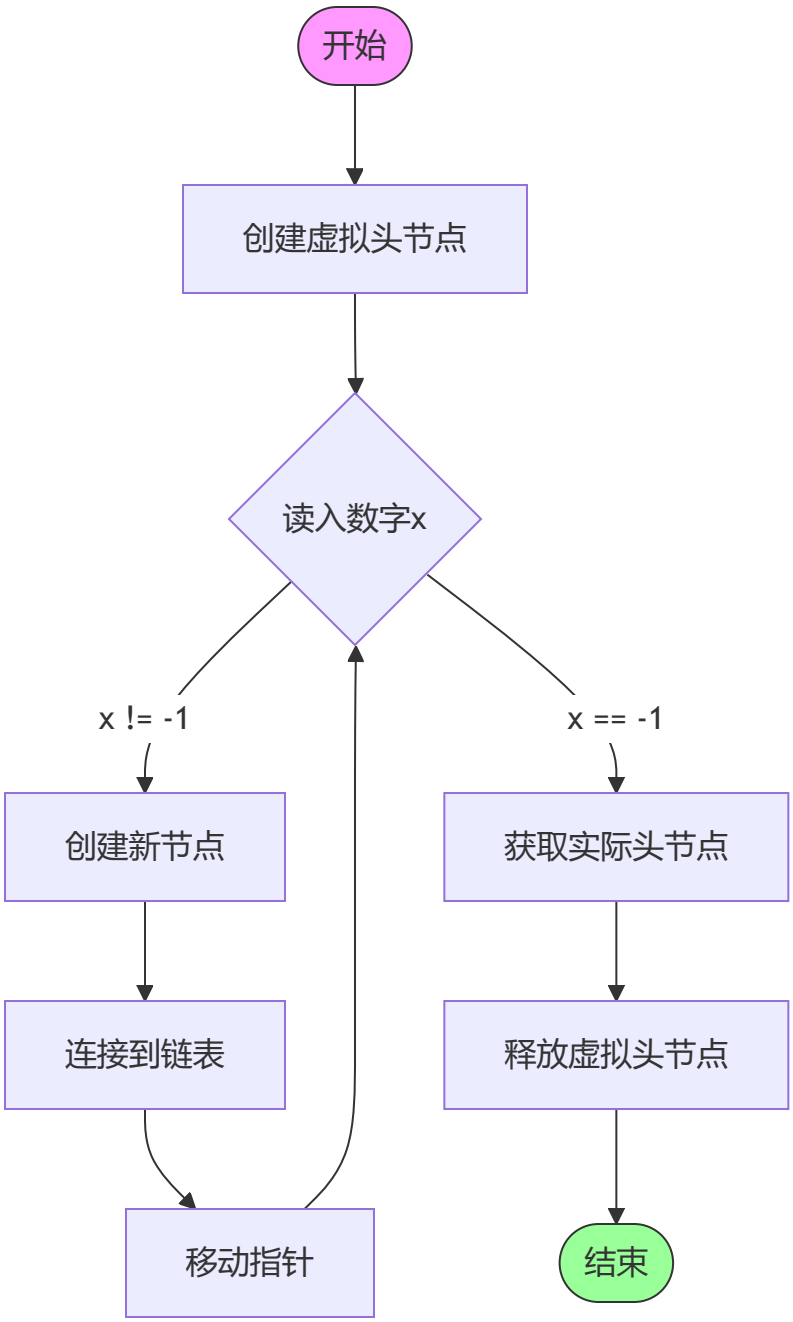
```cpp
struct ListNode {
    int val;           // 节点值
    ListNode* next;    // 指向下一个节点的指针
    ListNode(int x = 0) : val(x), next(nullptr) {}
};
```

内存结构示意图


节点结构
Node 1
val: 1
next: →
Node 2
val: 2
next: →
Node 3
val: 3
next: →
nullptr

链表创建实现

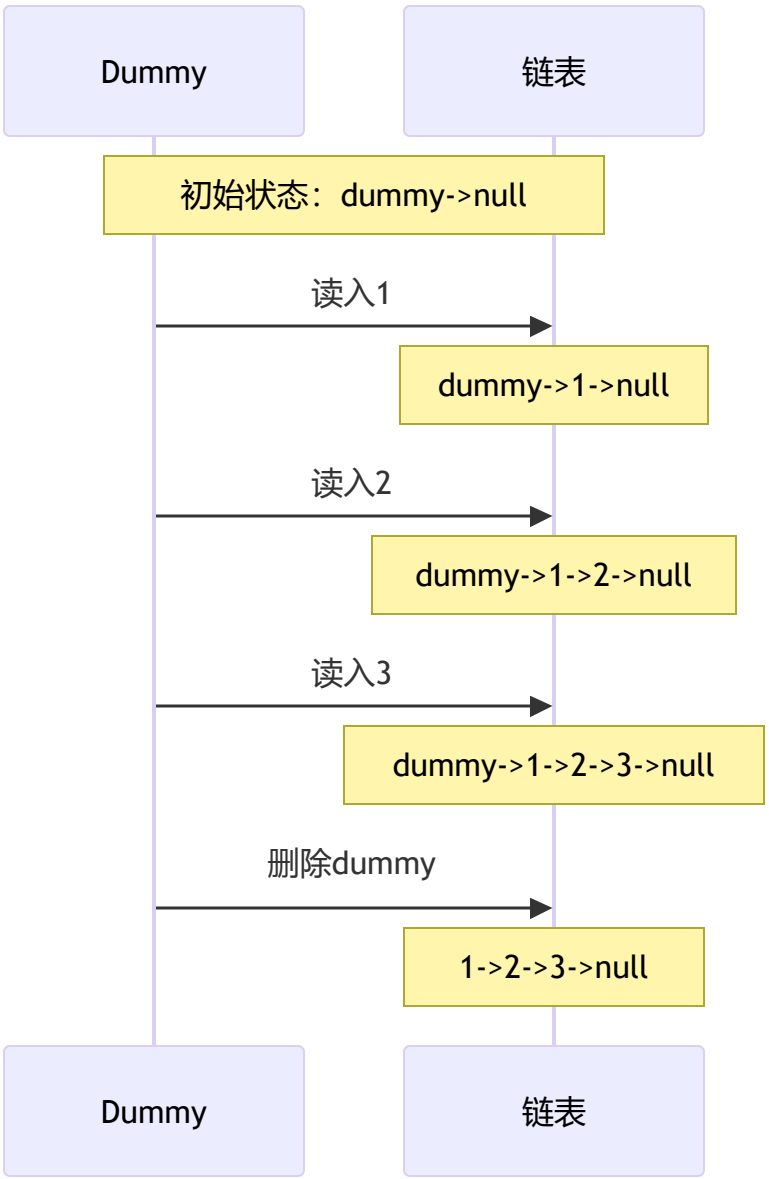
创建流程



代码实现

```
ListNode* createList() {
    ListNode* dummy = new ListNode(0);     // 虚拟头节点
    ListNode* cur = dummy;                 // 当前指针
    int x;

    // 循环读入数据
    while (cin >> x && x != -1) {
        cur->next = new ListNode(x);       // 创建新节点
        cur = cur->next;                   // 移动指针
    }

    // 处理虚拟头节点
    ListNode* head = dummy->next;
    delete dummy;
    return head;
}
```

创建过程示例

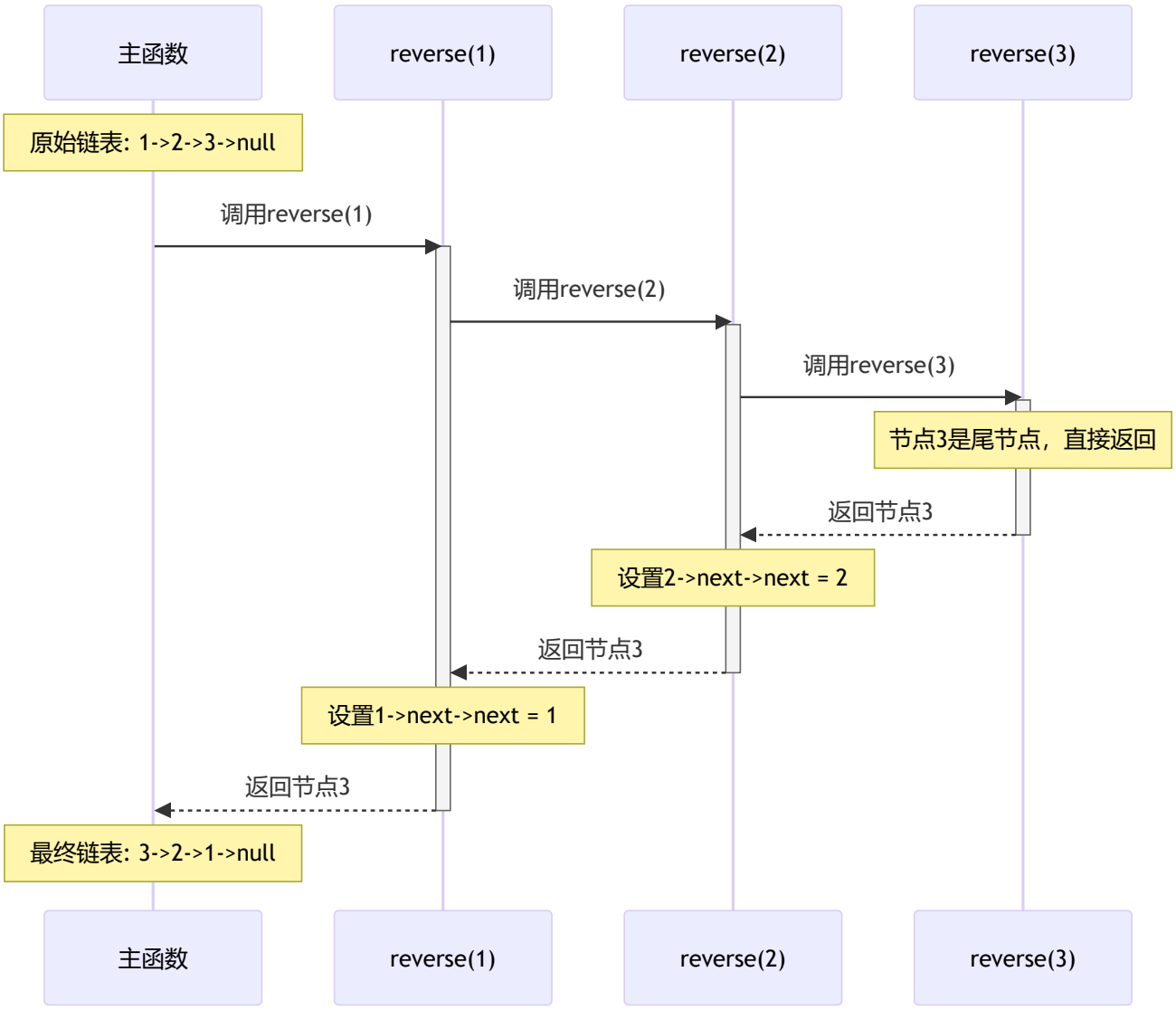


链表逆置

问题分析

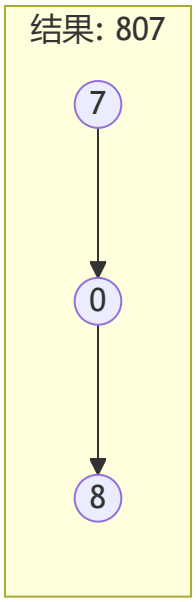
将链表从 `1->2->3->null` 转变为 `3->2->1->null`

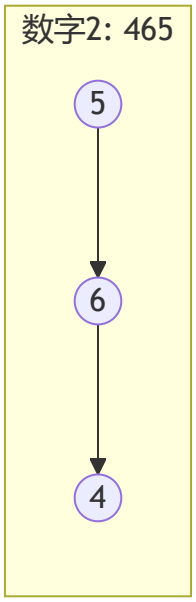
递归法实现过程

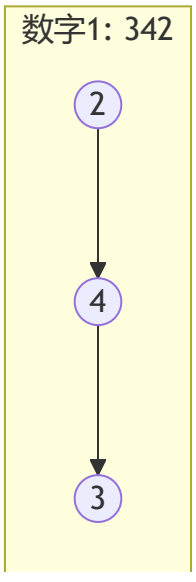

主函数
reverse(1)
reverse(2)
reverse(3)
原始链表: 1->2->3->null
调用reverse(1)
调用reverse(2)
调用reverse(3)
节点3是尾节点，直接返回
返回节点3
设置2->next->next = 2
返回节点3
设置1->next->next = 1
返回节点3
最终链表: 3->2->1->null
主函数
reverse(1)
reverse(2)
reverse(3)

链表加法实现

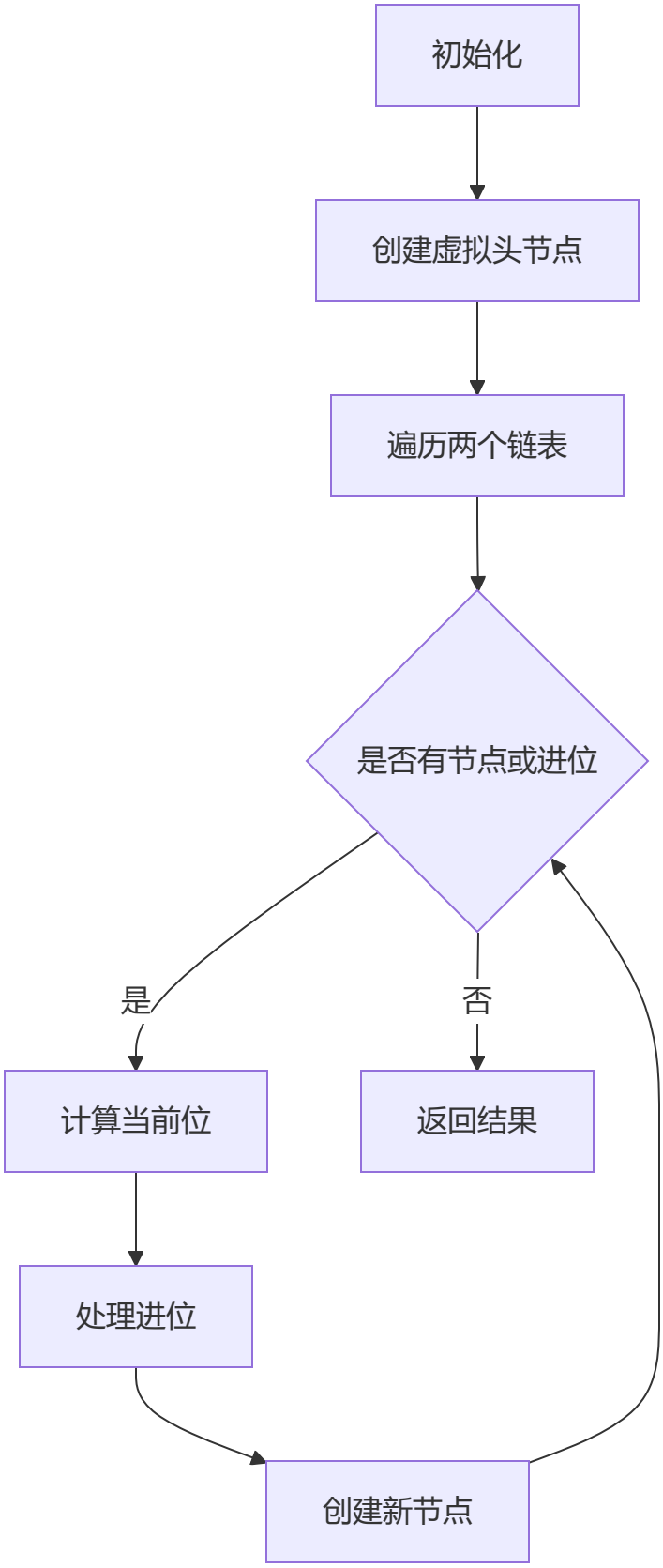
问题示例


结果: 807
7
0
8


数字2: 465
5
6
4

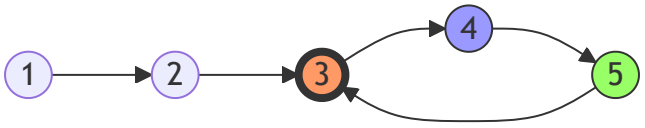

数字1: 342
2
4
3

计算流程

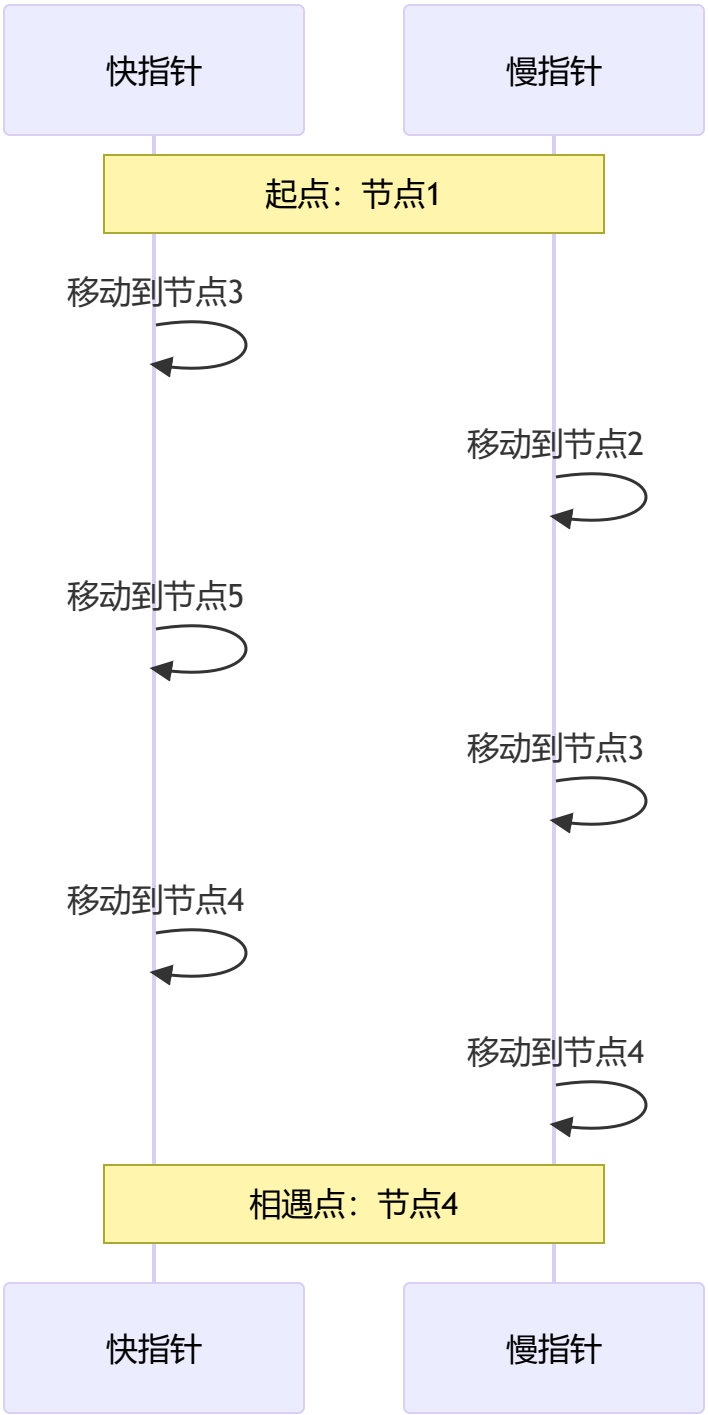


环形链表检测

Floyd 判圈算法原理



检测过程分析



排序专题

结构体排序

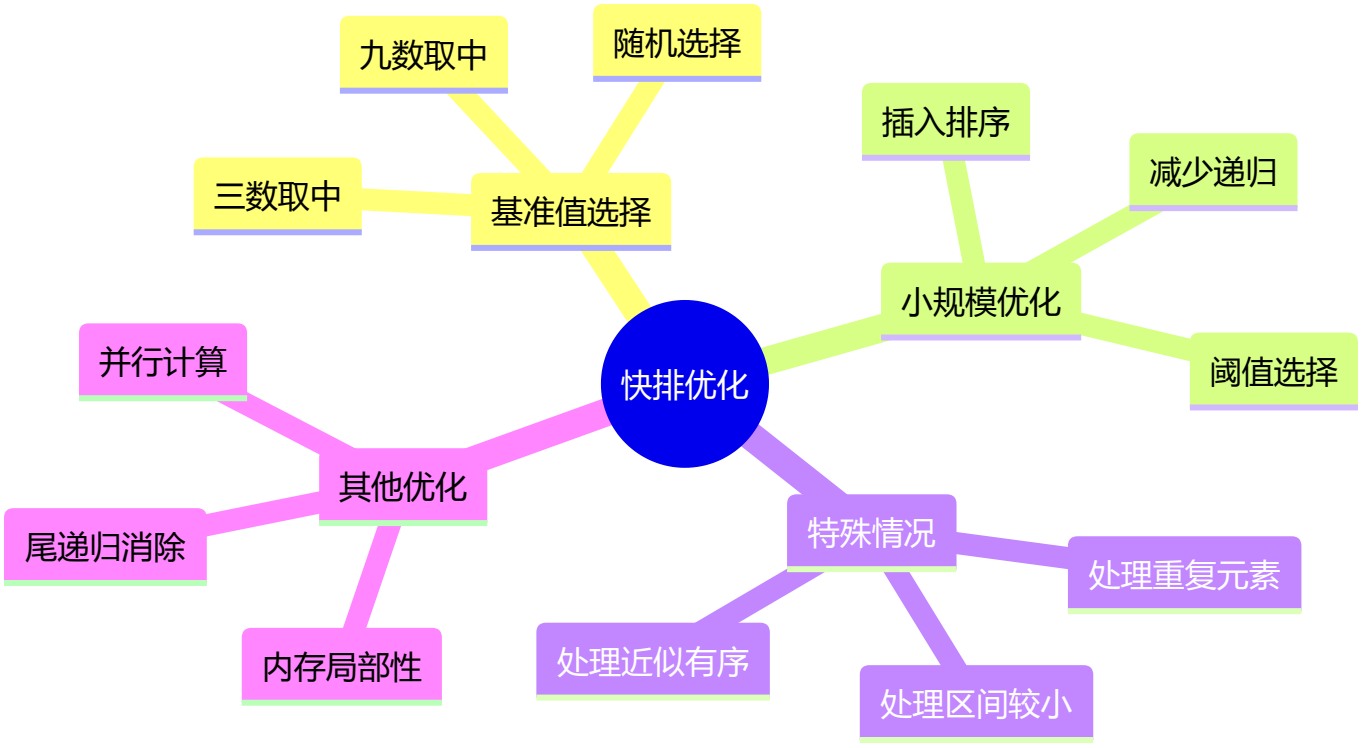
系统架构设计

排序规则实现

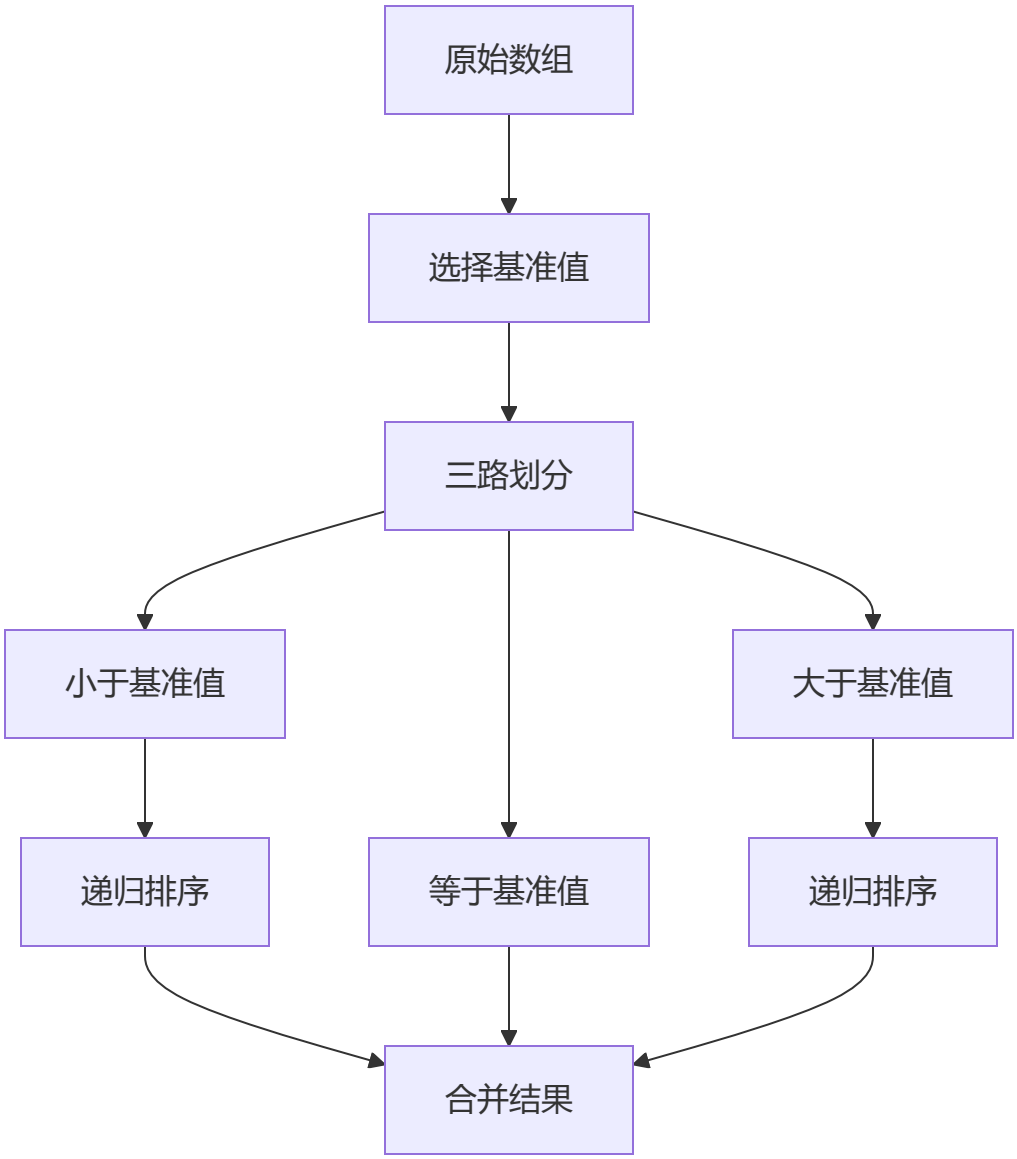
```
class ProductSorter {
public:
    // 按折后价格和库存排序
    static bool byPriceAndStock(const Product& a, const Product& b) {
        double priceA = a.getDiscountPrice();
        double priceB = b.getDiscountPrice();
        if (abs(priceA - priceB) > 1e-6) return priceA < priceB;
        return a.stock > b.stock;
    }

    // 按过期日期和价格排序
    static bool byExpiryAndPrice(const Product& a, const Product& b) {
        if (a.expiry < b.expiry) return true;
        if (b.expiry < a.expiry) return false;
        return a.getDiscountPrice() < b.getDiscountPrice();
    }
};
```

快速排序优化

优化策略概览



三路快排实现



模拟题专题

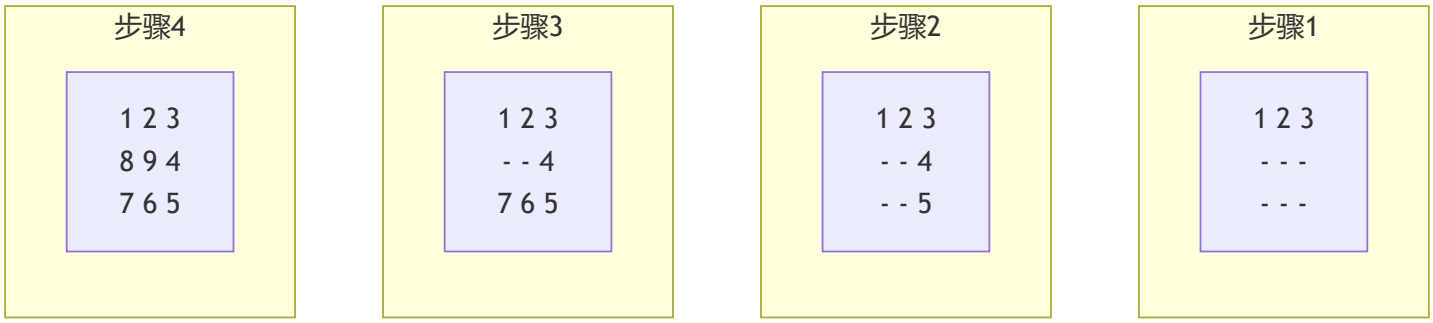
螺旋矩阵生成

代码实现

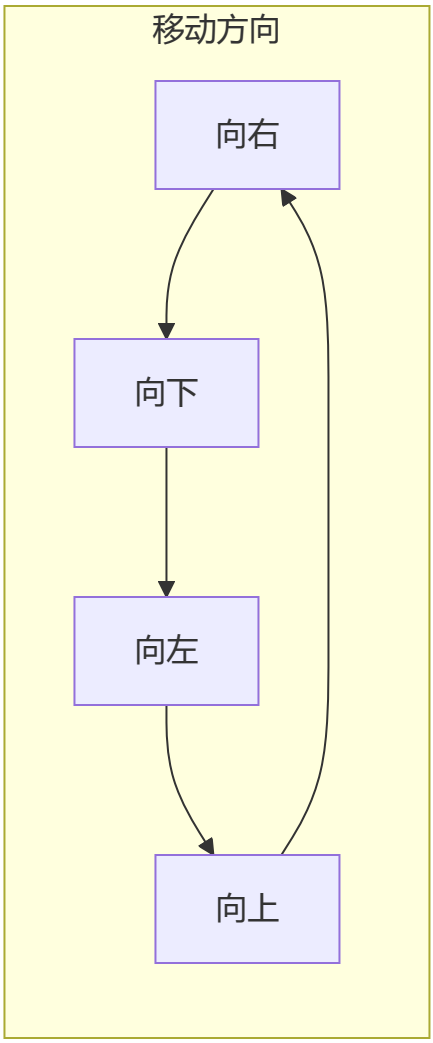
```
vector<vector<int>> generateMatrix(int n) {
    vector<vector<int>> matrix(n, vector<int>(n, 0));  // 初始化n*n矩阵
    int left = 0, right = n - 1;                       // 左右边界
    int top = 0, bottom = n - 1;                       // 上下边界
    int num = 1;                                       // 填充的数字

    while (num <= n * n) {
        // 从左到右填充
        for (int i = left; i <= right; i++) {
            matrix[top][i] = num++;
        }
        top++;

        // 从上到下填充
        for (int i = top; i <= bottom; i++) {
            matrix[i][right] = num++;
        }
        right--;

        // 从右到左填充
        for (int i = right; i >= left; i--) {
            matrix[bottom][i] = num++;
        }
        bottom--;

        // 从下到上填充
        for (int i = bottom; i >= top; i--) {
            matrix[i][left] = num++;
        }
        left++;
    }

    return matrix;
}

// 打印矩阵的辅助函数
```

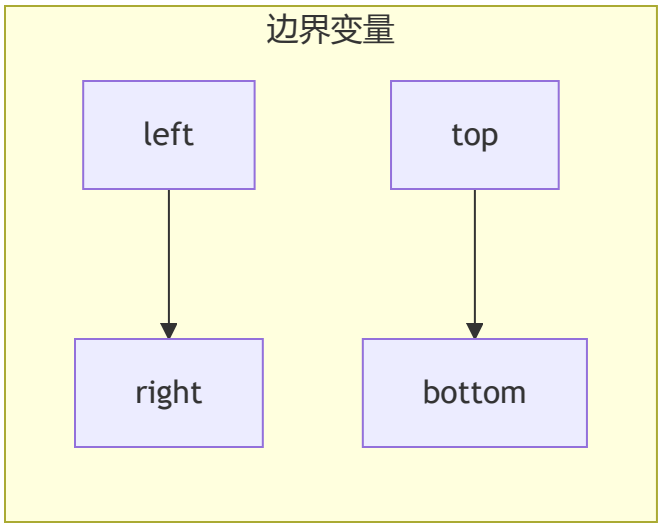
```cpp
void printMatrix(const vector<vector<int>>& matrix) {
    for (const auto& row : matrix) {
        for (int val : row) {
            cout << val << "\t";
        }
        cout << endl;
    }
}

// 使用示例
int main() {
    int n = 3;
    auto matrix = generateMatrix(n);
    printMatrix(matrix);
    return 0;
}
```

填充过程示意



边界控制





字符串计算器

代码实现

```
class Calculator {
private:
    stack<int> nums;
    stack<char> ops;

    // 判断运算符优先级
    int priority(char op) {
        if (op == '*' || op == '/') return 2;
        if (op == '+' || op == '-') return 1;
        return 0;
    }

    // 执行运算
    void calculate() {
        int b = nums.top(); nums.pop();
        int a = nums.top(); nums.pop();
        char op = ops.top(); ops.pop();

        int result = 0;
        switch (op) {
            case '+': result = a + b; break;
            case '-': result = a - b; break;
            case '*': result = a * b; break;
            case '/': result = a / b; break;
        }
        nums.push(result);
    }

public:
    int evaluate(string s) {
        for (int i = 0; i < s.length(); i++) {
            if (isspace(s[i])) continue;

            if (isdigit(s[i])) {
                int num = 0;
                while (i < s.length() && isdigit(s[i])) {
                    num = num * 10 + (s[i] - '0');
                    i++;
                }
```

```
                i--;
                nums.push(num);
            } else {
                while (!ops.empty() && priority(ops.top()) >= priority(s[i])) {
                    calculate();
                }
                ops.push(s[i]);
            }
        }

        while (!ops.empty()) {
            calculate();
        }

        return nums.top();
    }
};

// 使用示例
int main() {
    Calculator calc;
    string expr = "3 + 5 * 2";
    cout << "Expression: " << expr << endl;
    cout << "Result: " << calc.evaluate(expr) << endl;
    return 0;
}
```

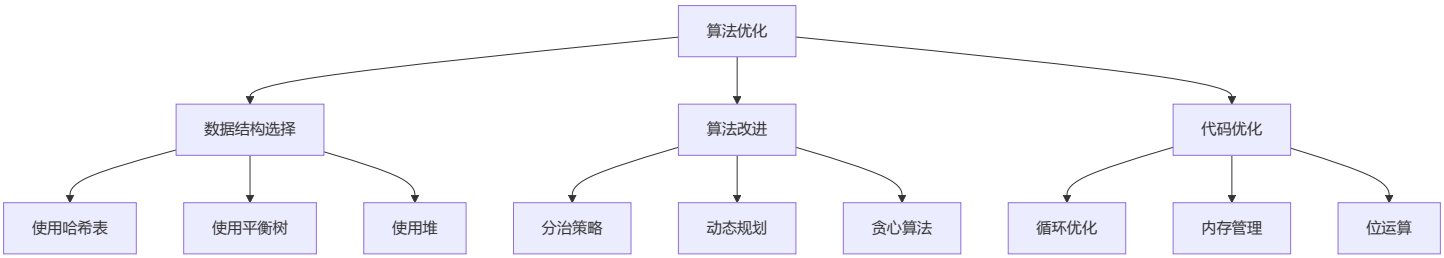
计算过程分析

1. 输入表达式 "3 + 5 * 2"
2. 数字栈和运算符栈的变化：
   - 读取 3：nums=[3]
   - 读取+：ops=[+]
   - 读取 5：nums=[3,5]
   - 读取*：ops=[+,*]（*优先级高于+）
   - 读取 2：nums=[3,5,2]
3. 开始计算：
   - 先计算 5*2：nums=[3,10], ops=[+]
   - 最后计算 3+10：nums=[13]
4. 返回结果 13

补充内容：实战技巧与优化方法

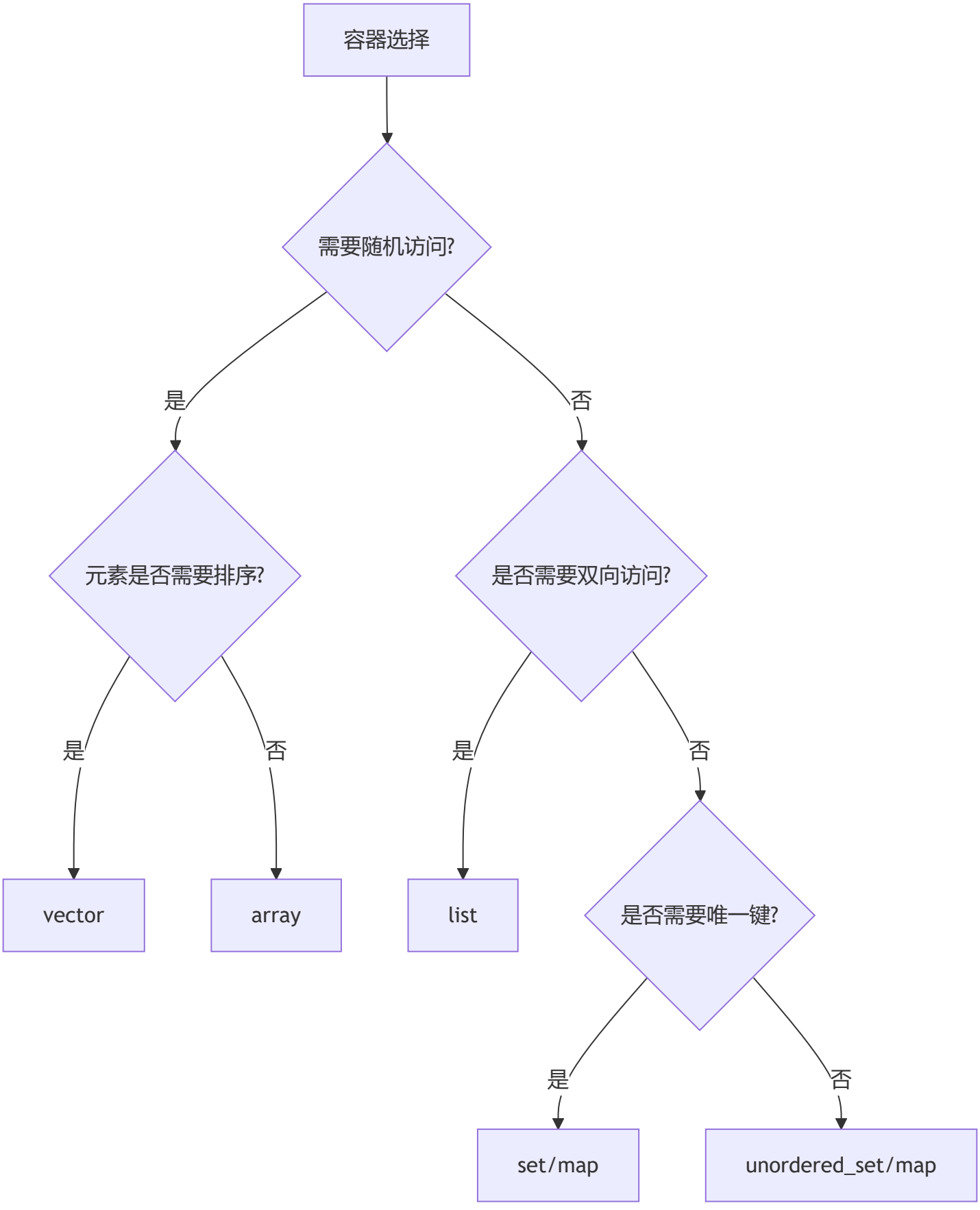
算法性能优化技巧

时间复杂度优化



代码实现优化实例

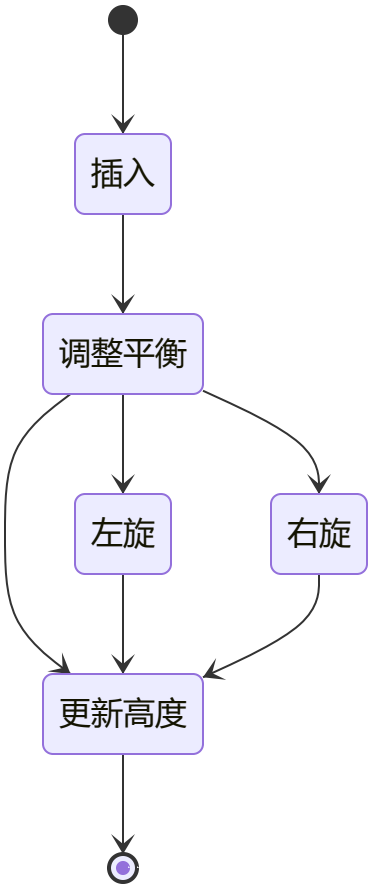
1. STL 容器选择



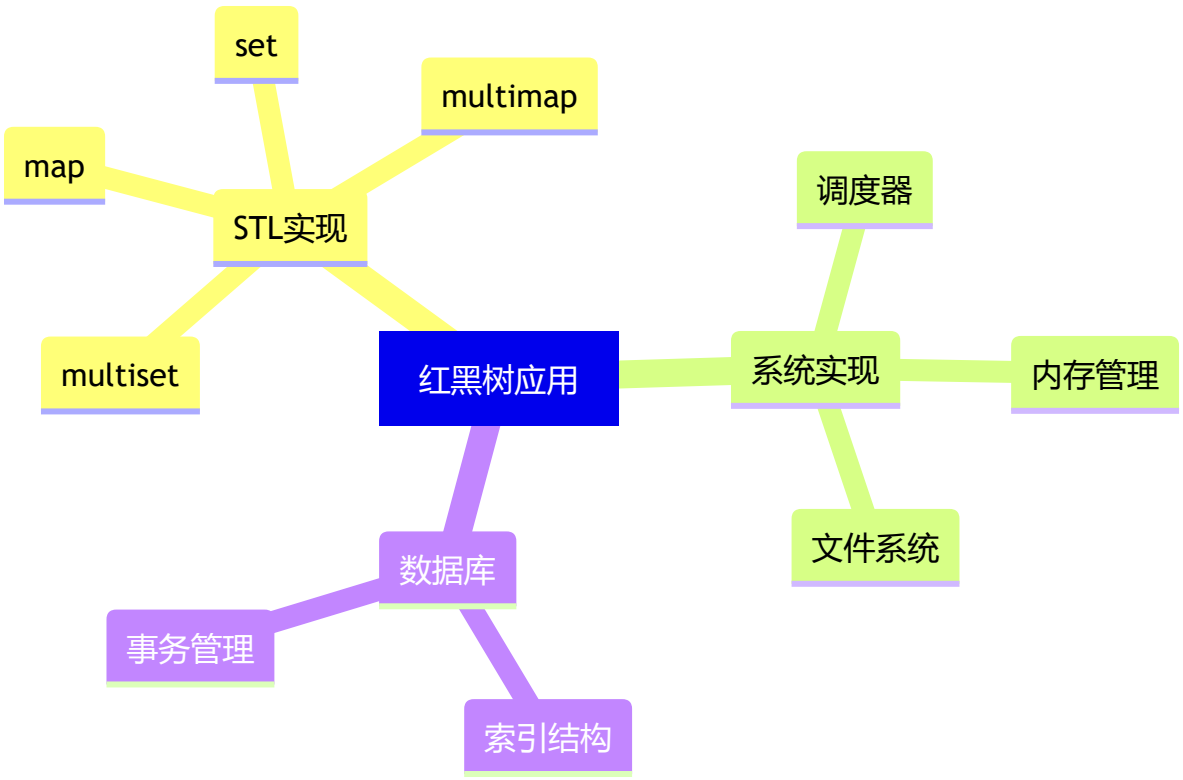
高级数据结构应用

平衡二叉树实现

```
template<typename T>
struct AVLNode {
    T data;
    int height;
    AVLNode<T>* left;
    AVLNode<T>* right;

    AVLNode(const T& d) :
        data(d), height(1),
        left(nullptr), right(nullptr) {}
};
```



红黑树应用场景



考试答题技巧