《数据结构综合设计》

名称		作业 1 C++&数据结构-1		评分 1		评分 2		总分	
类型		□验证型 ☑设计型 □综合型							
专业班级	卓软 2401	学号	5120246728						
任课教师	杨春明	姓名	华昊朗	时间		2025-06-23			
说明	评分 1 针对教学大纲课程目标 2，满分 10 分；评分 2 针对课程目标 3，满分 5 分。本次作业满分 15 分。								

一、目的

1、通过自定义顺序表类或链栈类等数据结构，深入理解 C++ 中类的封装、继承和多态特性，熟悉构造函数、析构函数和成员函数的实现方法。

2、掌握使用自定义数据结构类构建应用程序的方法，提升数据结构设计与编程实践能力。

3、能够根据程序设计的需求，具体练使用主流开发工具的能力和不断学习新技术的能力。

二、原理

使用单链表类实现两个集合的并运算。围绕节点链接与指针操作，每个节点包含数据域和指向下一节点的指针；链表通过头指针串联节点，尾节点指向 nullptr。集合并运算利用链表的动态增删特性。

三、环境

1、操作系统：Windows 11。

2、集成开发环境：Visual Studio 2022。

3、编程语言：C++

四、内容与步骤

1、定义链表 node

```
typedef struct node
{
    int data;
    struct node* next;
} Node;
```

2、定义类名及所需变量、函数

创建 Chain 类，创建相关变量节点及函数，代码如下：

```
class Chain
{
public:
    Node* head;
    Chain() { head = nullptr; }
    void insert(int value);
    void create(int n);
    bool isExist(int value);
    Chain cal(Chain& other);
    void print();
};
```

3、插尾函数

insert 类，用于对单链表尾部插入一个不重复的新元素，代码如下：

```
void Chain::insert(int value) {
    if (isExist(value)) {
        return;
    }
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = nullptr;

    if (head == nullptr) {
        head = newNode;
    }
    else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
```

```
}
```

4、读取输入并处理

从标准输入读取 n 个整数，并将它们依次插入到链表中并去重，代码如下：

```
void Chain::create(int n) {
    int value;
    for (int i = 0; i < n; i++) {
        cin >> value;
        insert(value);
    }
}
```

5、检查存在性

遍历链表检查给定值是否已存在于链表中，代码如下：

```
bool Chain::isExist(int value) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->data == value) {
            return true;
        }
        temp = temp->next;
    }
    return false;
}
```

6、计算并集并返回结果

计算当前链表与另一个链表的并集，代码如下：

```
Chain Chain::cal(Chain& other) {
    Chain result;
    Node* temp = this->head;
```

```
        while (temp != nullptr) {
            result.insert(temp->data);
            temp = temp->next;
        }
        temp = other.head;
        while (temp != nullptr) {
            result.insert(temp->data);
            temp = temp->next;
        }
        return result;
    }
```

7、输出函数

输出所产生的并集，代码如下：

```
    void Chain::print() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
```

8、主函数

新建两个链表，接受输入，并得出最后结果。

```
    int main()
    {
        Chain setA, setB;
        int n, m;cin >> n >> m;
        setA.create(n);
        setB.create(m);
```

```
    Chain res = setA.cal(setB);
    res.print();
    return 0;
}
```

4、编译与运行程序

在 Visual Studio 2022 中，对编写好的代码进行编译，检查并修正语法错误。编译成功后，运行程序，观察控制台输出结果，验证 “chain” 类的功能是否正确实现。

五、结果

程序运行后，控制台输出如下：

```
//样例 1
0 0
//无输出
```

```
//样例 2
8 8
1 5 9 2 6 5 3 5
8 9 7 9 3 2 3 8
1 5 9 2 6 3 8 7
```

```
//样例 3
5 5
1 1 1 1 1
1 1 1 1 1
1
```

从输出结果可以看出，通过调用类的公共接口函数，成功实现了对两个单链表的求并集操作。程序从测试样例接受多种正常或异常输入，并均能正确实现功能。

六、问题分析与解决

这段代码最容易报错的地方在于对重复数据、零输入、空指针解引用等的正确处理。在测试中，针对空指针解引用问题，我们进行了如下几点操作来规避这种错误：

- 初始化指针为 nullptr；

```
head = nullptr; //  开始时空链表
```

- 操作前检查空指针

```
if (head == nullptr) { //  如果是空链表
    head = newNode;     //  直接设为头节点
}
```

- 遍历时检查结束条件

```
while (temp != nullptr) { //  遇到空指针就停止
temp = temp->next;     //  移动到下一个
}
```

七、总结

通过实现单链表类及其集合并运算，我获得了以下核心收获：

更深入理解了链表机制：从节点结构到指针操作，掌握了尾插法构建、遍历终止条件等底层逻辑；

边界防御实战经验：空链表场景贯穿所有操作，强化了健壮性编程思维；

数据结构与业务结合：通过 **isExist()**实现插入去重，用两次遍历完成集合并运算，理解算法与数据结构的协同本质；

《数据结构综合设计》

名称		作业 1 C++&数据结构-2		评分 1		评分 2	总分	
类型		☐验证型 ☑设计型 ☐综合型						
专业班级	卓软 2401	学号	5120246728					
任课教师	杨春明	姓名	华昊朗	时间		2025-06-23		
说明	评分 1 针对教学大纲课程目标 2，满分 10 分；评分 2 针对课程目标 3，满分 5 分。本次作业满分 15 分。							

一、目的

1、通过自定义顺序表类或链栈类等数据结构，深入理解 C++ 中类的封装、继承和多态特性，熟悉构造函数、析构函数和成员函数的实现方法。

2、掌握使用自定义数据结构类构建应用程序的方法，提升数据结构设计与编程实践能力。

3、能够根据程序设计的需求，具体练使用主流开发工具的能力和不断学习新技术的能力。

二、原理

应用封装将类的成员变量和成员函数组合成一个整体。使用 private 处理输入，并应用栈的原理进行括号匹配：即遇到左括号则入栈，遇到右括号时，仅在栈非空且栈顶元素和右括号不匹配时输出 YES，否则均输出 NO。

三、环境

1、操作系统：Windows 11。

2、集成开发环境：Visual Studio 2022。

3、编程语言：C++

四、内容与步骤

1、定义 stacks 类用于对字符串进行操作

```
class stacks
{
private:
    string s;
public:
    void setString(string s)
    {
        this->s = s;
    }
```

```
    bool judge()
    {
        stack<char> st;
        for (char c : s)
        {
            if (c == '(' || c == '[' || c == '{')
                st.push(c);
            else if (c == ')' || c == ']' || c == '}' ）
            {
                if (st.empty()) return false;
                char top = st.top();
                st.pop();//出栈
                if ((c == ')' && top != '(') || (c == ']' && top != '[') || (c == '}' && top != '{'))
            return false;
                }
            }
            return st.empty();
    }
    void print()
    {
        if (judge())
            cout << "YES" << endl;
        else
            cout << "NO" << endl;
    }
};
```

2、在主函数中创建对象并进行操作

在 main 函数中，创建 stacks 类的对象，并调用类的公共接口函数进行输入输出、判断操作，代码如下：

```
int main()
{
```

```
    stacks sa;
     cout << "insert string : ";
    string s;cin >> s;
    sa.setString(s);
    sa.print();
    return 0;
}
```

3、编译与运行程序

在 Visual Studio 2022 中，对编写好的代码进行编译，检查并修正语法错误。编译成功后，运行程序，观察控制台输出结果，验证 “stacks” 类的功能是否正确实现。

五、结果

程序运行后，控制台输出如下：

```
//样例 1
insert string: (){}[()[]]
YES
```

```
//样例 2
insert string: a(b[c]d}e
NO
```

```
//样例 3
insert string: ({[()]}
NO
```

从输出结果可以看出，通过调用类的公共接口函数，成功实现了对私有成员变量 s 的合法操作。程序从测试样例接受多种正常或异常输入，并均能在**不超限**的情况下正确实现功能。

六、问题分析与解决

在一般情况下，该程序出现的问题无非有两种。一是上面所提到的超限问题，当长度超过 int 的范围后会出现不可预知的 bug，对此可以使用 long long 进行暂时性调整。但显然，如果用户的输入及其极端，以至于超出了 long long 的处理范围，预计程序不会返回正确结果。二是编码问题，即程序可能无法正确处理不同编码格式的输入。虽然本人环境下，输入

【】仍然能返回 YES，但无法排除其他可能的情况。因此推荐在系统中使用 ASCII 编码进行输入。

七、总结

本次作业让我初步了解了封装在 C++的应用。经过了对本任务与其它一些任务的探索，我学会了根据数据类型和任务要求合理选择public和private，因此更深层了解了封装的作用。同时对某些极端情况的 debug 也暴露了部分考虑不周的情况。我将更深入学习面向对象知识，合理运用，提高水平。

《数据结构综合设计》

<table>
<tr><td colspan="2">名称</td><td colspan="2">作业 1 C++&数据结构-3</td><td rowspan="2">评分 1</td><td rowspan="2"></td><td rowspan="2">评分 2</td><td rowspan="2"></td><td rowspan="2">总分</td><td rowspan="2"></td></tr>
<tr><td colspan="2">类型</td><td colspan="2">☐验证型 ☑设计型
☐综合型</td></tr>
<tr><td>专业班级</td><td>卓软 2401</td><td>学号</td><td>5120246728</td><td colspan="3">时间</td><td colspan="3">2025-06-26</td></tr>
<tr><td>任课教师</td><td>杨春明</td><td>姓名</td><td>华昊朗</td><td colspan="3"></td><td colspan="3"></td></tr>
<tr><td>说明</td><td colspan="9">评分 1 针对教学大纲课程目标 2，满分 10 分；评分 2 针对课程目标 3，满分 5 分。本次作业满分 15 分。</td></tr>
</table>

一、目的

1、通过自定义顺序表类或链栈类等数据结构，深入理解 C++ 中类的封装、继承和多态特性，熟悉构造函数、析构函数和成员函数的实现方法。

2、掌握使用自定义数据结构类构建应用程序的方法，提升数据结构设计与编程实践能力。

3、能够根据程序设计的需求，具体练使用主流开发工具的能力和不断学习新技术的能力。

二、原理

引用 STL 的 queue 头对代码进行简化，直接新建两个 queue 变量 qa、qb，使用双队列交替存储行数据。初始 qa 存入首行 1，循环处理每行：从 qa 取出元素输出并计算相邻元素并存入 qb，行末补 1。每行处理完后交换 qa 和 qb，qa 始终存储当前行，qb 预存下一行数据。

三、环境

1、操作系统：Windows 11。

2、集成开发环境：Visual Studio 2022。

3、编程语言：C++

四、内容与步骤

1、定义 queues 类，并封装功能函数

```
class queues
{
    private:
        queue<int>qa, qb;
    public:
        void print(int n)
        {
            qa.push(1);
            for (int i = 0; i <n; i++)
```

```
            {
                int pre = 0;
                while (!qa.empty())     //保证完全遍历 qa
                {
                    int curr = qa.front();
                    qa.pop();
                    cout << curr << " ";
                    qb.push(pre + curr);     //输出 qa 的同时预处理 qb
                    pre = curr;
                }
                qb.push(1);
                cout << endl;
                swap(qa, qb);
            }
        }

        void returns(int n)
        {
            if (n <= 0) cout << "invalid input!!!" << endl;
            else print(n);
        }
    };
```

2、在主函数中创建对象并进行操作

在 main 函数中，创建 queues 类对象 q，并调用类的 returns 公共接口函数进行输入、计算、输出操作，代码如下：

```
int main()
{
    int c;
    cin >> c;
    queues q;
    q.returns(c);
```

```
    return 0;
}
```

3、编译与运行程序

在 Visual Studio 2022 中，对编写好的代码进行编译，检查并修正语法错误。编译成功后，运行程序，输入需要输出的行数，观察输出结果，验证 “queues” 类的功能是否正确实现。

五、结果

程序运行后，控制台输出如下：

```
//样例 1
0
invalid input!!!
```

```
//样例 2
10
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
```

```
//样例 3
-3
invalid input
```

从输出结果可以看出，通过调用类的公共接口函数，基本成功实现了对杨辉三角的正确输出（如果不考虑格式）。

六、问题分析与解决

首先该代码存在一个很明显的问题，即显示的杨辉三角并不标准。由于数字长度等的制约，暂时无法做到将每个数字完全对齐。开发时有尝试着使用空格来补全，但后面冗长的数字告诉我，需要添加的空格数不止受单行制约。因此在权衡下，我选择删除空格。虽然这直接导致了显示变成了现在这鸟样。不过观感确实比先前加了空格好一点。所以……先这样吧。

其次，还是那个老生常谈的问题：数据范围。测试数据表明，当给定的行数过大时，会出现负数的异常。虽然还是可以改 long long 来临时补救一下，但这里也懒得改了，毕竟多写一天 C++，这个问题就多存在一天。

七、总结

本次作业让我深入了解了封装在 C++的应用并更加深入理解了 FIFO 原理，以及使用循环防止空间浪费。

《数据结构综合设计》

<table>
<tr><td colspan="2">名称</td><td colspan="2">作业 1 C++&数据结构-4</td><td rowspan="2">评分 1</td><td rowspan="2"></td><td rowspan="2">评分 2</td><td rowspan="2"></td><td rowspan="2">总分</td><td rowspan="2"></td></tr>
<tr><td colspan="2">类型</td><td colspan="2">☐验证型 ☑设计型
☐综合型</td></tr>
<tr><td>专业班级</td><td>卓软 2401</td><td>学号</td><td>5120246728</td><td colspan="3">时间</td><td colspan="3">2025-06-23</td></tr>
<tr><td>任课教师</td><td>杨春明</td><td>姓名</td><td>华昊朗</td><td colspan="3"></td><td colspan="3"></td></tr>
<tr><td>说明</td><td colspan="9">评分 1 针对教学大纲课程目标 2，满分 10 分；评分 2 针对课程目标 3，满分 5 分。本次作业满分 15 分。</td></tr>
</table>

一、目的

1、通过自定义顺序表类或链栈类等数据结构，深入理解 C++ 中类的封装、继承和多态特性，熟悉构造函数、析构函数和成员函数的实现方法。

2、掌握使用自定义数据结构类构建应用程序的方法，提升数据结构设计与编程实践能力。

3、能够根据程序设计的需求，具体练使用主流开发工具的能力和不断学习新技术的能力。

二、原理

应用封装将类的成员变量和成员函数组合成一个整体。使用 private 处理输入，并应用栈的原理进行括号匹配：即遇到左括号则入栈，遇到右括号时，仅在栈非空且栈顶元素和右括号不匹配时输出 YES，否则均输出 NO。

三、环境

1、操作系统：Windows 11。

2、集成开发环境：Visual Studio 2022。

3、编程语言：C++

四、内容与步骤

1、定义 btree 结构体

创建二叉树节点结构，包含数据域和左右子树指针。

```
typedef struct btree
{
    int data;
    struct btree* ltree;
    struct btree* rtree;
} binary_tree;
```

2、创建 tree 类

定义二叉树类，包含根节点、创建方法、WPL 计算方法及 DFS 辅助函数。

```
class tree
{
public:
    btree bt;
    void create(int n);
    void check(int n);

private:
    void dfs(btree* node, int depth, int& wpl);
};
```

3、对 tree 的创建节点操作

通过层序遍历输入构建二叉树，-1 表示空节点。

```
void tree::create(int n)
{
    if (n <= 0) {
        bt.data = 0;
        bt.ltree = NULL;
        bt.rtree = NULL;
        return;
    }

    vector<int> arr(n);
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    if (arr[0] == -1) {
        bt.data = 0;
        bt.ltree = NULL;
        bt.rtree = NULL;
        return;
```

```
}

bt.data = arr[0];
bt.ltree = NULL;
bt.rtree = NULL;

queue<btree*> q;
q.push(&bt);
int i = 1;

while (!q.empty() && i < n) {
    btree* p = q.front();
    q.pop();

    if (i < n) {
        int left_val = arr[i];
        i++;
        if (left_val != -1) {
            btree* lnode = new btree;
            lnode->data = left_val;
            lnode->ltree = NULL;
            lnode->rtree = NULL;
            p->ltree = lnode;
            q.push(lnode);
        }
    }

    if (i < n) {
        int right_val = arr[i];
        i++;
        if (right_val != -1) {
            btree* rnode = new btree;
```

```
                    rnode->data = right_val;
                    rnode->ltree = NULL;
                    rnode->rtree = NULL;
                    p->rtree = rnode;
                    q.push(rnode);
                }
            }
        }
    }
```

4、深度遍历整棵树

递归遍历二叉树，累加叶子节点的带权路径长度（节点值×深度）。

```
void tree::dfs(btree* node, int depth, int& wpl)
{
    if (node == NULL) {
        return;
    }
    if (node->ltree == NULL && node->rtree == NULL) {
        wpl += node->data * depth;
        return;
    }
    dfs(node->ltree, depth + 1, wpl);
    dfs(node->rtree, depth + 1, wpl);
}
```

5、初始化并进行计算

初始化 WPL 计算，调用 DFS 遍历并输出最终带权路径长度。

```
void tree::check(int n)
{
    int wpl = 0;
    dfs(&bt, 0, wpl);
    cout << wpl << endl;
}
```

6、主函数

```
int main()
{
    tree t;
    int n;
    cin >> n;
    t.create(n);
    t.check(0);
    return 0;
}
```

3、编译与运行程序

在 Visual Studio 2022 中，对编写好的代码进行编译，检查并修正语法错误。编译成功后，运行程序，观察控制台输出结果，验证功能是否正确实现。

五、结果

程序运行后，控制台输出如下：

```
//样例 1
0
0
```

```
//样例 2
1
5
0
```

```
//样例 3
7
1 2 -1 3 -1 -1 -1
6
```

从输出结果可以看出，通过调用类的公共接口函数，基本实现了预期功能。

六、问题分析与解决

对于本项目，存在以下几个极易出现的问题，以及我们做出的对策。

1、空树或无效输入处理不完善，我们采用显式初始化根节点并检查边界条件。

```
void tree::create(int n) {
    if (n <= 0 || (n >= 1 && arr[0] == -1)) {
        bt.data = 0;
        bt.ltree = bt.rtree = nullptr;
        return;
    }
}
```

2、指针未初始化，我们选择封装节点创建函数以及析构函数释放

```
btree* create_node(int val) {
    btree* node = new btree;
    node->data = val;
    node->ltree = node->rtree = nullptr;
    return node;
}
~tree() {
    delete_tree(&bt);
}
private:
void delete_tree(btree* node) {
    if (!node) return;
    delete_tree(node->ltree);
    delete_tree(node->rtree);
    delete node;
}
```

3、叶子节点误判，对此我们选择严格判断叶子节点并支持了负值累加

```
void tree::dfs(btree* node, int depth, int& wpl) {
    if (!node) return;
    if (!node->ltree && !node->rtree) {
        wpl += node->data * depth;
        return;
    }
```

```
        dfs(node->ltree, depth + 1, wpl);
        dfs(node->rtree, depth + 1, wpl);
    }
```

七、总结

本次任务在让我加深对面向对象理解的同时，认识到逻辑、内存管理等方面的重要性。通过设计针对性的测试用例，我学会了如何系统地验证程序的正确性。最终完成的 WPL 计算器能够稳定处理各种边界输入，这让我对数据结构实现有了更扎实的掌握。

《数据结构综合设计》

<table>
<tr><td colspan="2">名称</td><td colspan="2">作业 1 C++&数据结构-5</td><td rowspan="3">评分 1</td><td rowspan="3"></td><td rowspan="3">评分 2</td><td rowspan="3"></td><td rowspan="3">总分</td><td rowspan="3"></td></tr>
<tr><td colspan="2">类型</td><td colspan="2">☐验证型 ☑设计型
☐综合型</td></tr>
<tr><td>专业班级</td><td>卓软 2401</td><td>学号</td><td>5120246728</td></tr>
<tr><td>任课教师</td><td>杨春明</td><td>姓名</td><td>华昊朗</td><td colspan="3">时间</td><td colspan="3">2025-06-23</td></tr>
<tr><td>说明</td><td colspan="9">评分 1 针对教学大纲课程目标 2，满分 10 分；评分 2 针对课程目标 3，满分 5 分。本次作业满分 15 分。</td></tr>
</table>

一、目的

1、通过自定义顺序表类或链栈类等数据结构，深入理解 C++ 中类的封装、继承和多态特性，熟悉构造函数、析构函数和成员函数的实现方法。

2、掌握使用自定义数据结构类构建应用程序的方法，提升数据结构设计与编程实践能力。

3、能够根据程序设计的需求，具体练使用主流开发工具的能力和不断学习新技术的能力。

二、原理

使用对一张 10*10 无向图进行 BFS 搜索。

三、环境

1、操作系统：Windows 11。

2、集成开发环境：Visual Studio 2022。

3、编程语言：C++

四、内容与步骤

1、定义 Graph 类，并分配空间。

使用邻接矩阵实现一个最多 10 个顶点的无向图，提供添加边和广度优先搜索(BFS)遍历功能。

```
class Graph {
public:
    int a[10][10];
    int log[10];
    int count;

    Graph() {
        memset(a, 0, sizeof(a));
        memset(log, 0, sizeof(log));
```

```
        count = 0;
    }
    void addEdge(int u, int v);
    void bfs(int start) ;
};
```

2、创建边函数

在无向图的邻接矩阵中添加一条边，并更新图中实际使用的顶点数量。

```
void Graph::addEdge(int u, int v) {
    if (u >= 0 && u < 10 && v >= 0 && v < 10) {
        a[u][v] = 1;
        a[v][u] = 1;
        if (u + 1 > count) count = u + 1;
        if (v + 1 > count) count = v + 1;
    }
}
```

3、创建广度优先搜索（BFS）函数

创建 BFS 代码如下：

```
void Graph::bfs(int start) {
    memset(log, 0, sizeof(log));

    queue<int> q;
    q.push(start);
    log[start] = 1;
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";

        for (int i = 0; i < count; i++) {
            if (a[node][i] == 1 && !log[i]) {
                log[i] = 1;
```

```
                    q.push(i);
                }
            }
        }
        cout << endl;
    }
```

4、在主函数中创建 Graph 对象并进行操作

创建对象，代码如下：

```
int main() {
    Graph g;
    int u, v;
    while (true) {
        cin >> u >> v;
        if (u == 0 && v == 0) break;
        g.addEdge(u, v);
    }
    g.bfs(0);
    return 0;
}
```

3、编译与运行程序

在 Visual Studio 2022 中，对编写好的代码进行编译，检查并修正语法错误。编译成功后，运行程序，观察控制台输出结果，验证功能是否正确实现。

五、结果

程序运行后，控制台输出如下：

```
//样例 1
1 3
0 1
1 9
2 3
9 7
```

```
2 7
0 0
0 1 3 9 2 7
```

```
//样例 2
1 0
1 0
0 0
0 1
```

```
//样例 3
8 5
5 6
6 4
4 4
4 9
9 2
2 8
0 0
0
```

从输出结果可以看出，代码基本实现了对输入的正确处理。

六、问题分析与解决

尽管程序思路基本没什么大问题，但还是针对部分可能出现的问题进行了不同程度的规避：

1、数组越界访问问题。当输入顶点值不在[0,9]范围时，访问 a[u][v]会越界，导致程序崩溃或数据损坏。对此，我们使用了严格的边界检查，保证只有符合范围的才进行读入。

```
if (u >= 0 && u < 10 && v >= 0 && v < 10)
```

2、非连通图遍历不完整问题。BFS 仅从起点遍历连通分量，孤立顶点被忽略，容易导致图中部分顶点未被访问。因此我们使用 count 控制遍历范围。

```
for (int neighbor = 0; neighbor < count; neighbor++)
```

3、重复访问死循环。如果未标记已访问节点导致节点重复入队，会陷入死循环。我们采取入队时立即在 log 中标记访问状态。

```
Log[i] = 1;
q.push(i);
```

七、总结

通过实现邻接矩阵图类及其 BFS 算法，我掌握了图结构的核心存储方式、边界检查的重要性、状态标记的时序控制、无向图对称性处理的必要性，以及用户输入验证的实战技巧，这些具体技能可直接迁移至其他数据结构实现。