实验任务2.2 NFA转DFA算法及实现

一、实验目的

1. 掌握非确定有限自动机（NFA）与确定有限自动机（DFA）的基本概念及其转换方法。
2. 了解NFA到DFA转换过程中的子集构造算法。
3. 实现NFA到DFA的转换算法，并验证DFA的正确性。
4. 设计合理的数据结构，延续上一次实验的结构，以便为后续DFA最小化实验任务做好准备。
5. 提高编程能力及算法设计和优化的技能。

二、实验内容

1. **理论背景**：NFA是一种可以处理多条路径的状态机，而DFA是其确定版本，不存在多条路径。通过子集构造算法（Subset Construction），可以将NFA转换为等价的DFA，从而实现字符串匹配的确定性处理。
2. **任务描述**：实现将NFA转换为DFA的算法，并对转换后的DFA进行验证。同时，设计适合DFA的数据结构，使其兼容前一次实验的NFA数据结构。
3. 实验步骤
   :
   - 理解子集构造算法的原理，包括ε-闭包的计算和状态集合的映射。
   - 利用子集构造算法，将NFA转换为DFA。
   - 设计并实现DFA的数据结构，确保其能够表示状态集合、状态转换、初始状态和接受状态。
   - 验证DFA的正确性，对比DFA与NFA在同一组测试输入上的匹配结果。

三、实验要求

1. 输入输出要求
   - 输入：一个NFA（包括状态集、转换表、初始状态和接受状态集合）和多个测试字符串。
   - 输出：生成的DFA状态集合及其转换关系，指明每个测试字符串是否被DFA接受。
2. 算法要求
   - 实现子集构造算法，将NFA状态集合的子集映射为DFA的单个状态。
   - 处理ε-闭包及其状态转换，生成对应的DFA。
3. 数据结构要求
   - 在上一实验的基础上，设计DFA的数据结构，包含状态集合、转换关系、初始状态和接受状态集合的表示。
   - 确保数据结构可以支持后续的DFA最小化任务，便于后续实验任务的延续。
4. 程序要求
   - 使用C/C++、Java、Python等语言编写程序，代码结构清晰，具备良好的注释。
   - 提供详细的实验报告，包括算法设计、实现过程、测试结果和问题分析。

5. 实验报告要求【整合到最后提交的个人所有实验报告中，加上目录】
    - 描述实验目的和内容。
    - 解释子集构造算法的原理、步骤和数据结构的设计思路。
    - 给出测试用例和结果，分析测试数据的正确性。
    - 总结实验的收获和遇到的挑战。

四、实验指南

1. **准备工作**
    - 复习NFA和DFA的相关理论知识，特别是**子集构造算法**及**ε-闭包**的计算方法。
    - 了解NFA和DFA在算法实现中的常见数据结构，如状态转换表的表示。
    - 安装编程环境（如Python的IDE，C/C++的编译器等），熟悉相关编程工具的使用。
2. **实验步骤**
    - **步骤1**：读取NFA的输入，包括状态集合、转换关系、初始状态和接受状态。
    - **步骤2**：计算NFA各状态的ε-闭包，并根据子集构造算法生成DFA的状态。
    - **步骤3**：根据输入符号集，对NFA状态集合进行扩展，生成对应的DFA转换关系。
    - **步骤4**：设计并实现DFA的数据结构，将其表示为状态集合、状态转换表、初始状态和接受状态。
    - **步骤5**：模拟DFA，验证对给定输入字符串的接受性，确保DFA与NFA的接受结果一致。
    - **步骤6**：进行测试，使用多个NFA和测试字符串验证程序的正确性和健壮性。【可采用作业题目来验证】
3. **注意事项**
    - 确保算法能够正确计算ε-闭包，并将状态集合映射为DFA的状态。
    - 数据结构的设计要考虑DFA最小化的需求，如状态集合的合并和查找效率。
    - 在程序中增加异常处理和边界情况的测试，确保算法的鲁棒性。
4. **扩展任务（可选）**
    - 优化DFA的数据结构，减少状态和转换关系的存储空间。
    - 实现从DFA到最小化DFA的算法，为后续实验提供支持。