**目 录**

[1 基本数据结构 1](#_Toc93868432)

[2 二叉树 1](#_Toc93868433)

[2.1 树的子结构 1](#_Toc93868434)

[3 回溯 1](#_Toc93868435)

[3.1 矩阵中的路径(单词搜索) 2](#_Toc93868436)

[3.2 机器人的运动范围 3](#_Toc93868437)

[4 动态规划 4](#_Toc93868438)

[4.1 剪绳子 4](#_Toc93868439)

[4.2 最长回文子串 5](#_Toc93868440)

[4.3 正则表达式匹配 5](#_Toc93868441)

[5 双指针 6](#_Toc93868442)

[5.1 三数之和 6](#_Toc93868443)

[6 滑动窗口 7](#_Toc93868444)

[6.1 解题框架 7](#_Toc93868445)

[6.2 无重复字符的最长子串 8](#_Toc93868446)

[7 链表 8](#_Toc93868447)

[7.1 反转链表 9](#_Toc93868448)

[8 快速幂 9](#_Toc93868449)

[8.1 数值的整数次方 9](#_Toc93868450)

[9 有限状态自动机 9](#_Toc93868451)

[9.1 表示数值的字符串 10](#_Toc93868452)

1. 基本数据结构
2. 二叉树
   1. 树的子结构

输入两棵二叉树A和B，判断B是不是A的子结构。(约定空树不是任意一个树的子结构)

B是A的子结构， 即 A中有出现和B相同的结构和节点值。

例如:

给定的树 A:

     3

    / \

   4   5

  / \

 1   2

给定的树 B：

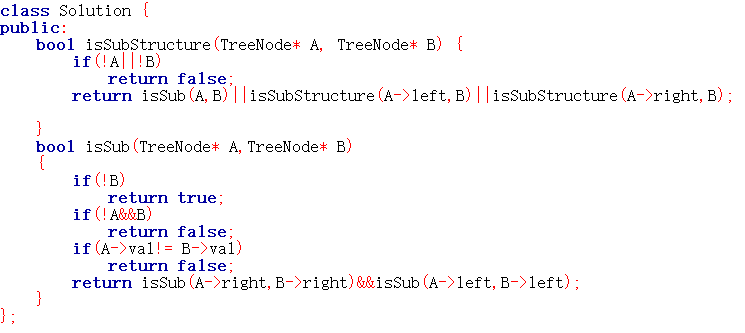
   4

  /

 1

返回 true，因为 B 与 A 的一个子树拥有相同的结构和节点值。

【解题思路】先遍历再递归，先判树A的根节点是否与树B的根节点相同，如果相同在递归判断（前序遍历）树A的子树和树B的子树每个节点是否相等。

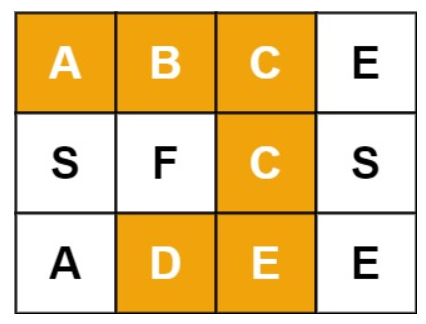


1. 回溯
   1. 矩阵中的路径(单词搜索)

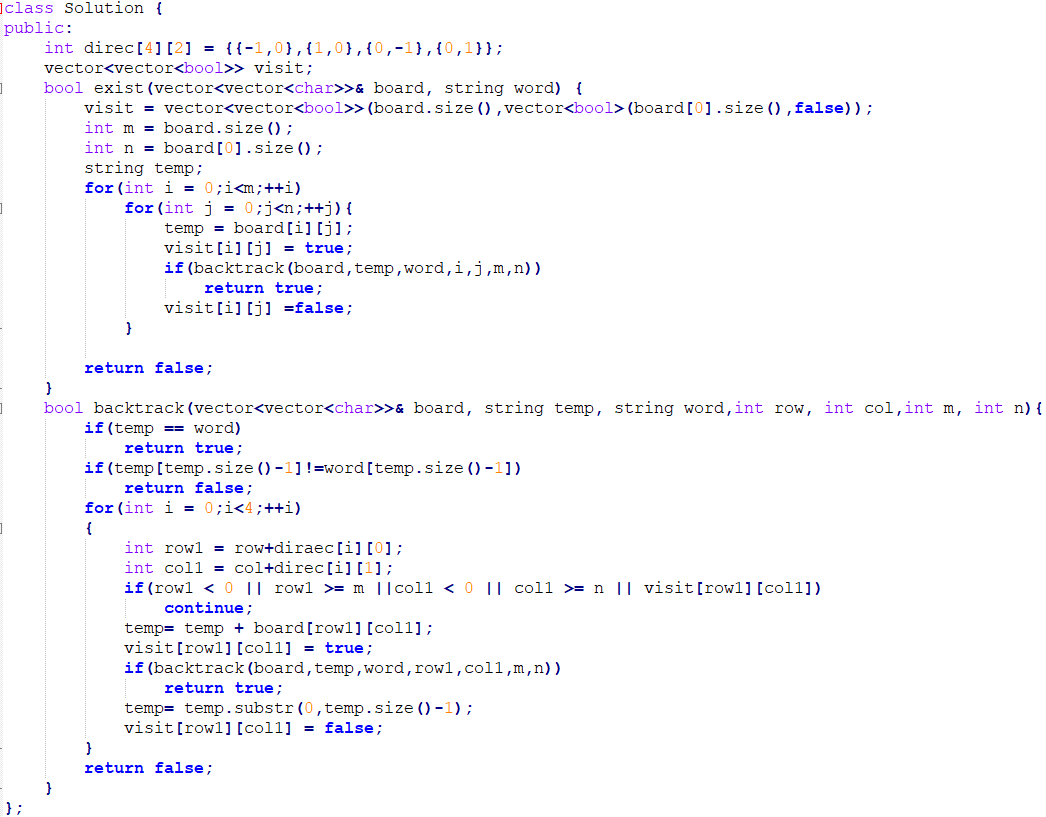
给定一个 m x n 二维字符网格 board 和一个字符串单词 word 。如果 word 存在于网格中，返回 true ；否则，返回 false 。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

例如，在下面的 3×4 的矩阵中包含单词 "ABCCED"（单词中的字母已标出）



**【解题思路】**非常明显的回溯题，注意：1. 在主函数里面需要遍历二维数组获取单词起点，再进行回溯；2. 根据每一位字符进行剪枝，而不是每次判断整个单词是否相等。



* 1. 机器人的运动范围

地上有一个m行n列的方格，从坐标 [0,0] 到坐标 [m-1,n-1] 。一个机器人从坐标 [0, 0] 的格子开始移动，它每次可以向左、右、上、下移动一格（不能移动到方格外），也不能进入行坐标和列坐标的数位之和大于k的格子。例如，当k为18时，机器人能够进入方格 [35, 37] ，因为3+5+3+7=18。但它不能进入方格 [35, 38]，因为3+5+3+8=19。请问该机器人能够到达多少个格子？

**【解题思路】**回溯时需注意两个问题，1，递归不回溯，返回后visit不需要设回false；2，只需要向右向下移动即可，不需要遍历四个方向。



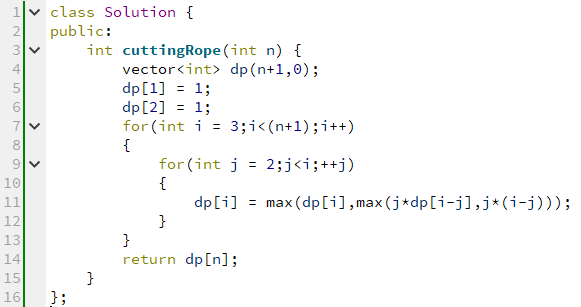
1. 动态规划
   1. 剪绳子

给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段（m、n都是整数，n>1并且m>1），每段绳子的长度记为 k[0],k[1]...k[m-1] 。请问 k[0]\*k[1]\*...\*k[m-1] 可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

**【解题思路】**能想到是动态规划，取最大值时实际应该是：（共四项）因为m>1，其中(i-j)\*j表示只剪一刀，dp[i-j]\*dp[j]表示至少3刀，dp[i-j]\*j,和dp[j]\*[i-j]表示至少两刀。

max(dp[i-j]\*dp[j], (i-j)\*j, dp[i-j]\*j, dp[j]\*[i-j])

因为会进行迭代，所以有些像将是重复的，可以只保留其中两项。max(j\*dp[i-j],j\*(i-j))



* 1. 最长回文子串

给你一个字符串 s，找到 s 中最长的回文子串。

【解题思路】字符串子串的动态规划问题，需要定义2维dp数组，i,j分别表示子串的首位index。



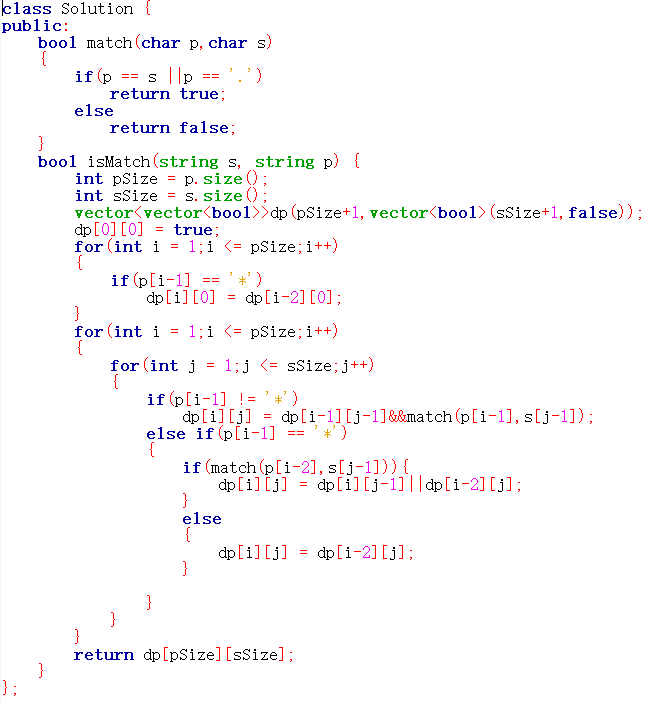
* 1. 正则表达式匹配

请实现一个函数用来匹配包含'. '和'\*'的正则表达式。模式中的字符'.'表示任意一个字符，而'\*'表示它前面的字符可以出现任意次（含0次）。在本题中，匹配是指字符串的所有字符匹配整个模式。例如，字符串"aaa"与模式"a.a"和"ab\*ac\*a"匹配，但与"aa.a"和"ab\*a"均不匹配。

【解题思路】难点在于遇到“\*”时状态转移表达式的写法，分两种情况考虑。

1. p[i-1] == s[j]，dp[i][j] = dp[i][j-1]||dp[i-2][j]
2. p[i-1] != s[j]，dp[i][j] = dp[i-2][j]

设计dp数组时，应使dp数组长宽比s,p长度大1，初始状态dp[0][0]为true，且x\*y\*这种类型的正则表达式可以和空字符串匹配。



1. 双指针
   1. 三数之和

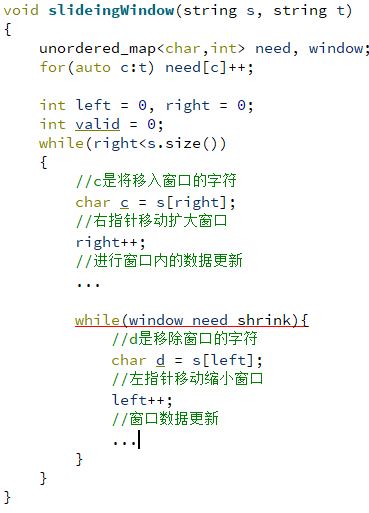
给你一个包含 n 个整数的数组 nums，判断 nums 中是否存在三个元素 a，b，c ，使得 a + b + c = 0 ？请你找出所有和为 0 且不重复的三元组。

注意：答案中不可以包含重复的三元组。

【解题思路】首先排序（排序后才能使用双指针？），遍历数组取第一个数，第二个数和第三个数采用双指针，题目的重点在于去重。



1. 滑动窗口
   1. 解题框架



* 1. 无重复字符的最长子串

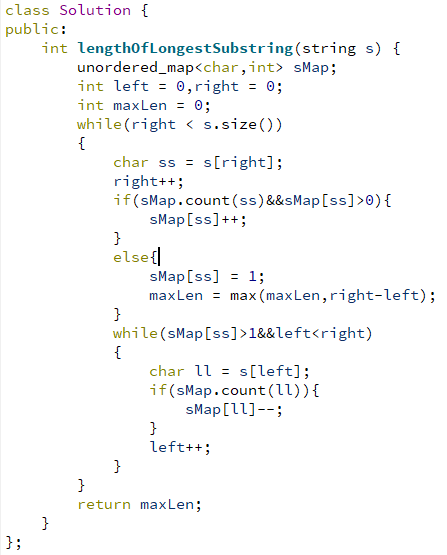
给定一个字符串 s ，请你找出其中不含有重复字符的 **最长子串**的长度。

**输入:** s = "abcabcbb"

**输出:** 3

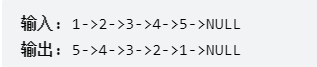
**解释:** 因为无重复字符的最长子串是 "abc"，所以其长度为 3。

【解题思路】双指针，套模板

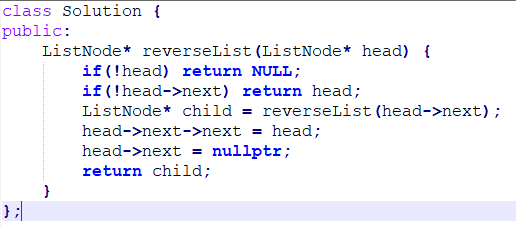


1. 链表
   1. 反转链表

定义一个函数，输入一个链表的头节点，反转该链表并输出反转后链表的头节点。



**【解题思路】**两种方法：递归、迭代。注意答案的最后一个节点需要指向null。



1. 图论
   1. 课程表

你这个学期必须选修 numCourses 门课程，记为 0 到 numCourses - 1 。

在选修某些课程之前需要一些先修课程。 先修课程按数组 prerequisites 给出，其中 prerequisites[i] = [ai, bi] ，表示如果要学习课程 ai 则 必须 先学习课程  bi 。

例如，先修课程对 [0, 1] 表示：想要学习课程 0 ，你需要先完成课程 1 。

请你判断是否可能完成所有课程的学习？如果可以，返回 true ；否则，返回 false 。

【解题思路】1.先构建邻接表，再通过深度优先搜索判断图是否有环。

1. 拓扑排序，先构建邻接表和入度表，建立一个queue用来临时储存入度为0的节点。

入度为0表示该课程没有任何依赖，可以进行学习，每次从queue中取出一个入度为0的节点，并把以该节点为依赖的所有节点的入度减1，如果入度减为0就push进queue，最后查看能够学习的课程数目是否等于课程总数。

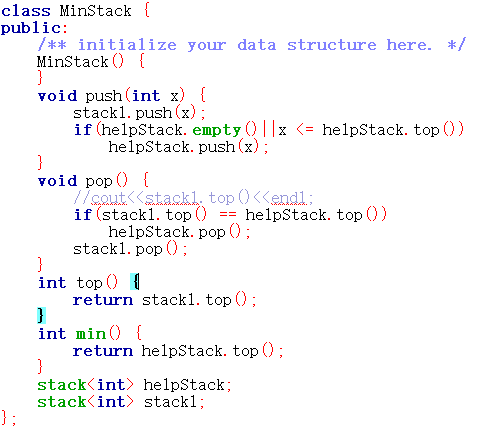




1. 基础数据结构
   1. 包含min函数的栈

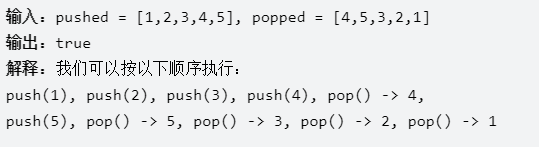
定义栈的数据结构，请在该类型中实现一个能够得到栈的最小元素的 min 函数在该栈中，调用 min、push 及 pop 的时间复杂度都是 O(1)。

【解题思路】定义辅助栈，栈顶永远存储min

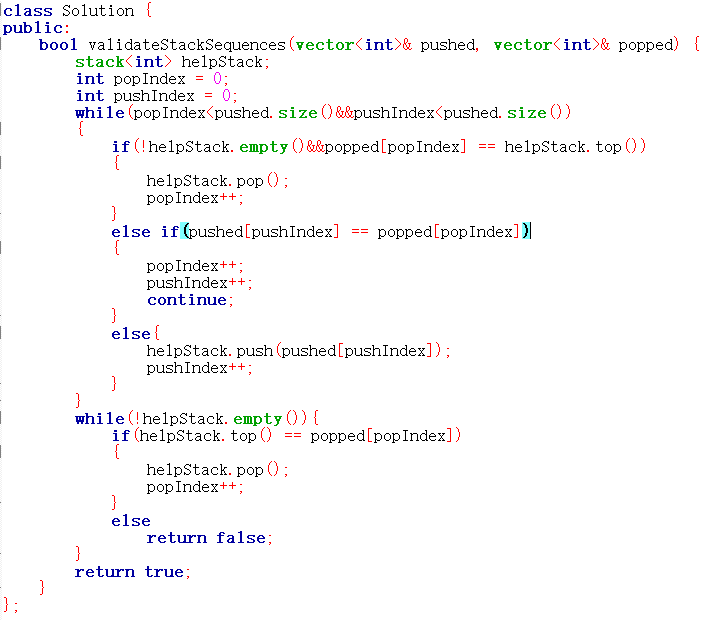


* 1. 栈的压入、弹出序列

输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如，序列 {1,2,3,4,5} 是某栈的压栈序列，序列 {4,5,3,2,1} 是该压栈序列对应的一个弹出序列，但 {4,3,5,1,2} 就不可能是该压栈序列的弹出序列。



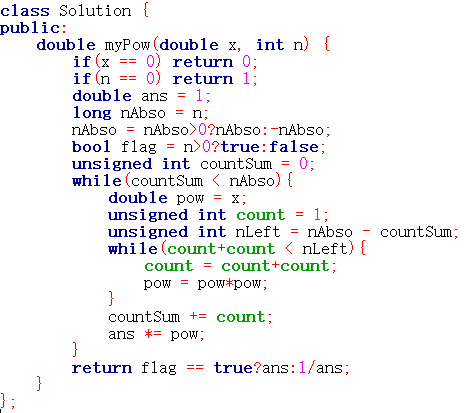
【解题思路】定义辅助栈



1. 快速幂
   1. 数值的整数次方

实现 [pow(x, n)](https://www.cplusplus.com/reference/valarray/pow/) ，即计算 x 的 n 次幂函数（即，xn）。不得使用库函数，同时不需要考虑大数问题。

【解题思路】快速幂法时间复杂度为log(n)，注意int 的溢出问题。



1. 有限状态自动机
   1. 表示数值的字符串

请实现一个函数用来判断字符串是否表示数值（包括整数和小数）。

数值（按顺序）可以分成以下几个部分：

1. 若干空格
2. 一个 小数 或者 整数
3. （可选）一个 'e' 或 'E' ，后面跟着一个 整数
4. 若干空格

小数（按顺序）可以分成以下几个部分：

1. （可选）一个符号字符（'+' 或 '-'）
2. 下述格式之一：

至少一位数字，后面跟着一个点 '.'

至少一位数字，后面跟着一个点 '.' ，后面再跟着至少一位数字

一个点 '.' ，后面跟着至少一位数字

整数（按顺序）可以分成以下几个部分：

（可选）一个符号字符（'+' 或 '-'）

至少一位数字

部分数值列举如下：

["+100", "5e2", "-123", "3.1416", "-1E-16", "0123"]

部分非数值列举如下：

["12e", "1a3.14", "1.2.3", "+-5", "12e+5.4"]

**预备知识**

确定有限状态自动机（以下简称「自动机」）是一类计算模型。它包含一系列状态，这些状态中：

有一个特殊的状态，被称作「初始状态」。

还有一系列状态被称为「接受状态」，它们组成了一个特殊的集合。其中，一个状态可能既是「初始状态」，也是「接受状态」。

起初，这个自动机处于「初始状态」。随后，它顺序地读取字符串中的每一个字符，并根据当前状态和读入的字符，按照某个事先约定好的「转移规则」，从当前状态转移到下一个状态；当状态转移完成后，它就读取下一个字符。当字符串全部读取完毕后，如果自动机处于某个「接受状态」，则判定该字符串「被接受」；否则，判定该字符串「被拒绝」。

**注意：**如果输入的过程中某一步转移失败了，即不存在对应的「转移规则」，此时计算将提前中止。在这种情况下我们也判定该字符串「被拒绝」。

一个自动机，总能够回答某种形式的「对于给定的输入字符串 S，判断其是否满足条件 P」的问题。在本题中，条件 P 即为「构成合法的表示数值的字符串」。

自动机驱动的编程，可以被看做一种暴力枚举方法的延伸：它穷尽了在任何一种情况下，对应任何的输入，需要做的事情。

自动机在计算机科学领域有着广泛的应用。在算法领域，它与大名鼎鼎的字符串查找算法「KMP」算法有着密切的关联；在工程领域，它是实现「正则表达式」的基础。

**问题描述**

在 C++ 文档 中，描述了一个合法的数值字符串应当具有的格式。具体而言，它包含以下部分：

1. 符号位，即 ++、-− 两种符号
2. 整数部分，即由若干字符 0-90−9 组成的字符串
3. 小数点
4. 小数部分，其构成与整数部分相同
5. 指数部分，其中包含开头的字符 \text{e}e（大写小写均可）、可选的符号位，和整数部分

相比于 C++ 文档而言，本题还有一点额外的不同，即允许字符串首末两端有一些额外的空格。

在上面描述的五个部分中，每个部分都不是必需的，但也受一些额外规则的制约，如：

如果符号位存在，其后面必须跟着数字或小数点。

小数点的前后两侧，至少有一侧是数字。

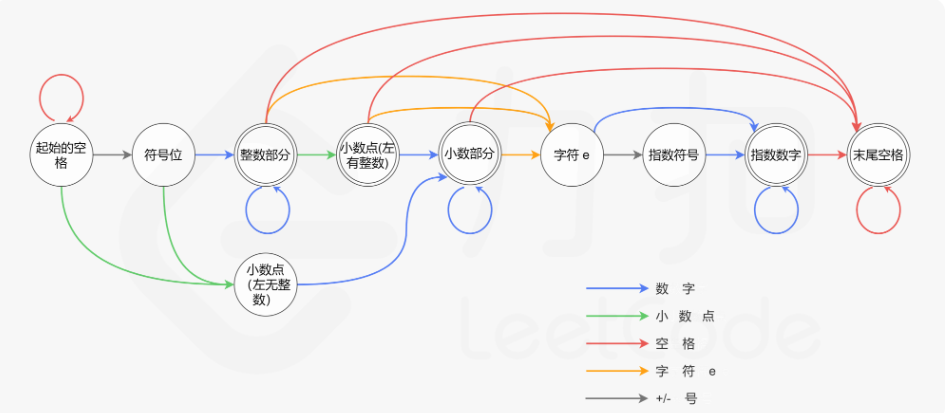
**思路与算法**

根据上面的描述，现在可以定义自动机的「状态集合」了。那么怎么挖掘出所有可能的状态呢？一个常用的技巧是，用「当前处理到字符串的哪个部分」当作状态的表述。根据这一技巧，不难挖掘出所有状态：

1. 起始的空格
2. 符号位
3. 整数部分
4. 左侧有整数的小数点
5. 左侧无整数的小数点（根据前面的第二条额外规则，需要对左侧有无整数的两种小数点做区分）
6. 小数部分
7. 字符 \text{e}e
8. 指数部分的符号位
9. 指数部分的整数部分
10. 末尾的空格

下一步是找出「初始状态」和「接受状态」的集合。根据题意，「初始状态」应当为状态 1，而「接受状态」的集合则为状态 3、状态 4、状态 6、状态 9 以及状态 10。换言之，字符串的末尾要么是空格，要么是数字，要么是小数点，但前提是小数点的前面有数字。

最后，需要定义「转移规则」。结合数值字符串应当具备的格式，将自动机转移的过程以图解的方式表示出来：



class Solution {

public:

    enum State {

        STATE\_INITIAL,

        STATE\_INT\_SIGN,

        STATE\_INTEGER,

        STATE\_POINT,

        STATE\_POINT\_WITHOUT\_INT,

        STATE\_FRACTION,

        STATE\_EXP,

        STATE\_EXP\_SIGN,

        STATE\_EXP\_NUMBER,

        STATE\_END

    };

    enum CharType {

        CHAR\_TYPE\_INTEGER,

        CHAR\_TYPE\_DOT,

        CHAR\_TYPE\_SIGN,

        CHAR\_TYPE\_E,

        CHAR\_TYPE\_SPACE,

        CHAR\_TYPE\_ILLEGAL

    };

    CharType toCharType(char c){

        if((c>='0')&&(c<='9'))

            return CHAR\_TYPE\_INTEGER;

        else if(c == '.')

            return CHAR\_TYPE\_DOT;

        else if(c == 'e'||c == 'E')

            return CHAR\_TYPE\_E;

        else if(c == '+'||c == '-')

            return CHAR\_TYPE\_SIGN;

        else if(c == ' ')

            return CHAR\_TYPE\_SPACE;

        else

            return CHAR\_TYPE\_ILLEGAL;

    }

    bool isNumber(string s) {

        unordered\_map<State,unordered\_map<CharType,State>> transfer =

        {

            {

                STATE\_INITIAL, {

                    {CHAR\_TYPE\_INTEGER,STATE\_INTEGER},

                    {CHAR\_TYPE\_SPACE,STATE\_INITIAL},

                    {CHAR\_TYPE\_SIGN,STATE\_INT\_SIGN},

                    {CHAR\_TYPE\_DOT,STATE\_POINT\_WITHOUT\_INT}

                }

            },

            {

                STATE\_INTEGER, {

                    {CHAR\_TYPE\_INTEGER,STATE\_INTEGER},

                    {CHAR\_TYPE\_DOT,STATE\_FRACTION},

                    {CHAR\_TYPE\_E,STATE\_EXP},

                    {CHAR\_TYPE\_SPACE,STATE\_END}

                }

            },

            {

                STATE\_POINT,{

                    {CHAR\_TYPE\_INTEGER,STATE\_FRACTION},

                    {CHAR\_TYPE\_E,STATE\_EXP},

                    {CHAR\_TYPE\_SPACE,STATE\_END}

                }

            },

            {

                STATE\_INT\_SIGN,{

                    {CHAR\_TYPE\_INTEGER,STATE\_INTEGER},

                    {CHAR\_TYPE\_DOT,STATE\_POINT\_WITHOUT\_INT}

                }

            },

            {

                STATE\_POINT\_WITHOUT\_INT,{

                    {CHAR\_TYPE\_INTEGER,STATE\_FRACTION}

                }

            },

            {

                STATE\_FRACTION,{

                    {CHAR\_TYPE\_INTEGER,STATE\_FRACTION},

                    {CHAR\_TYPE\_E,STATE\_EXP},

                    {CHAR\_TYPE\_SPACE,STATE\_END}

                }

            },

            {

                STATE\_EXP,{

                    {CHAR\_TYPE\_SIGN,STATE\_EXP\_SIGN},

                    {CHAR\_TYPE\_INTEGER,STATE\_EXP\_NUMBER}

                }

            },

            {

                STATE\_EXP\_SIGN,{

                    {CHAR\_TYPE\_INTEGER,STATE\_EXP\_NUMBER}

                }

            },

            {

                STATE\_EXP\_NUMBER,{

                    {CHAR\_TYPE\_INTEGER,STATE\_EXP\_NUMBER},

                    {CHAR\_TYPE\_SPACE,STATE\_END}

                }

            },

            {

                STATE\_END,{

                    {CHAR\_TYPE\_SPACE,STATE\_END}

                }

            }

        };

        State state = STATE\_INITIAL;

        for(auto c:s)

        {

            CharType type = toCharType(c);

            //cout<<type;

            auto map = transfer[state];

            if(map.find(type) == map.end())

                return false;

            state = map[type];

        }

        return (state == STATE\_INTEGER||state == STATE\_FRACTION||state == STATE\_POINT||state == STATE\_EXP\_NUMBER||state == STATE\_END);

    }

};