表示数值的字符串

思路：

两种情况 `A[.[B]][e|EC]`、`.B[e|EC]`

- A: 第一个正数部分
  - 第一位可以为符号
- B: 第二个正数部分
  - 无符号
- C: 第三个正数部分
  - 第一位可以为符号

思路优化：`[F] [D1].[D2] [e|E [F] D3]`

- `F` 符号位：
- `D` 数值位
- `[F] [D1].[D2]`：数值部分
- `[e|E [F] D3]`：积分部分
- 数值部分两种情况：
  - 整数
    - 1位：`D`
    - 2位以上：`[F] D`
  - 小数
    - `[F]D.`
    - `[F].D`
    - `[F]D.D`

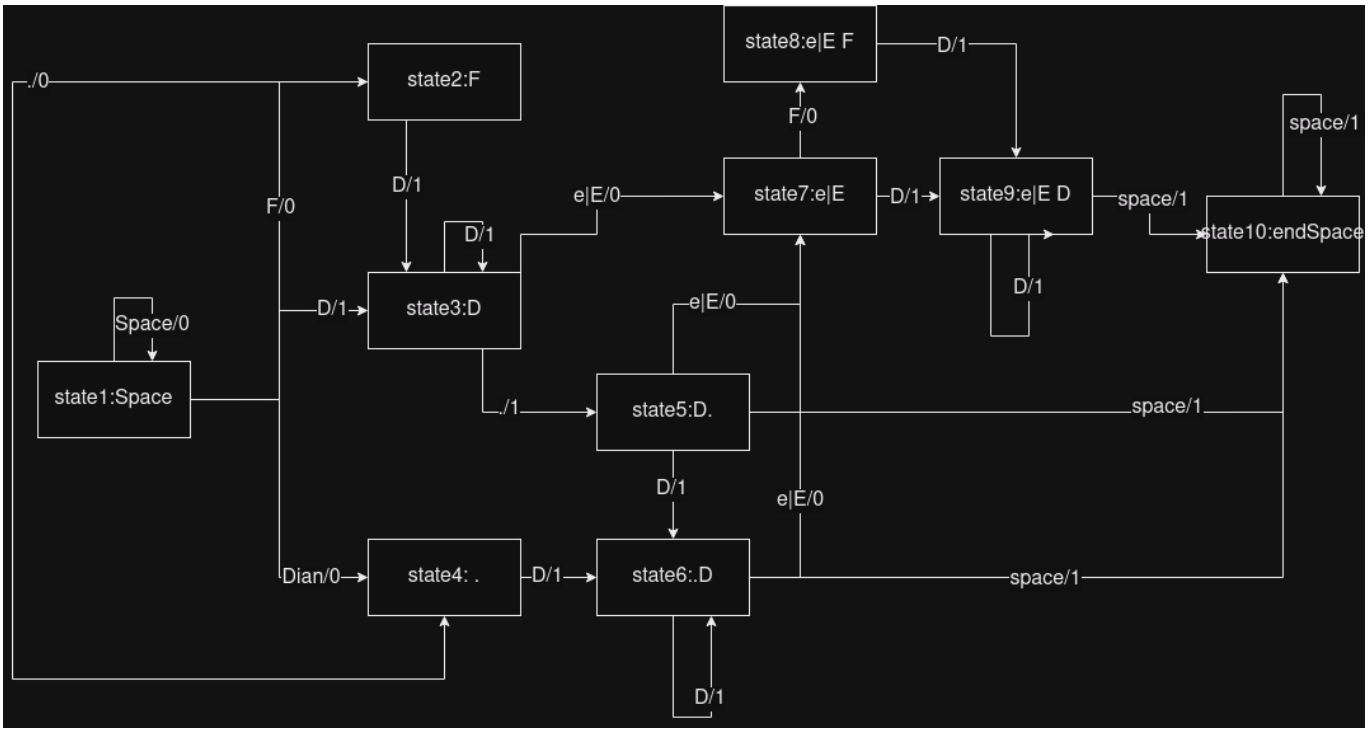
```
class Solution {
public:
    bool myIsDigit(string s, int l, int r){
        for(int i = l; i < r; ++i){
            if(s[i] < '0' || s[i] > '9') return 0;
        }
        return 1;
    }
    bool myIsDigitHasMark(string s, int l, int r){
        if(l-r == 1) return myIsDigit(s,l,r);
        else{
            bool firstOne = s[l] == '-' || s[l] == '+' || (s[l] >= '0' &&
s[l] <= '9');
            return firstOne && myIsDigit(s, l+1, r);
        }

    }
    bool isNumber(string s) {
        int n = s.size();
```

```
        //边界判断
        if( n <= 0 ) return 0;
        else if (n == 1){
            if(s[0] < '0' || s[0] > '9') return false;
            else return true;
        }
        int a1=-1, cn = n, tag0 = -1, tag1 = cn;

        //找到第一个非空格字符索引a1
        for(int i = 0; i < n; ++i){
            if(s[i] != ' '){
                a1 = i;
                break;
            }
        }

        //找到几个关键标志位
        for(int i = a1; i < n; ++i){

            //找到后空格序列第一个空格索引
            if(s[i] == ' '){
                cn = i;
                break;
            }
            else if(s[i] == '.'){ // 找到 . 的索引
                if(tag0 != -1) return 0;
                tag0 = i;
            }else if(s[i] == 'e' || s[i] == 'E') {// 找到 e|E 的索引
                if(tag1 != cn) return 0;
                tag1 = i;
            }
        }

        //调整无 . 和 e|E 的索引
        if(tag0 < a1-1){
            tag0 = a1-1;
        }
        if(tag1 > cn){
            tag1 = cn;
        }

        for(int i = cn; i < n; ++i){//检索后空格序列中是否存在非空格字符
            if(s[i] != ' ') return 0;
        }

        //边界检验
        if(cn - a1 <= 0) return false;
        else if(cn - a1 == 1){
            if(s[a1] < '0' || s[a1] > '9') return false;
            else return true;
        }

        if(tag1 == a1 //不存在数值部分
        // || (tag1 < cn && tag0 == tag1 - 1)) // .e 或.E 不存在积分后整数部分
```

```
        )
            return false;

        //数值部分
        if(tag1 - a1 == 1){ //数值部分只有1位
            if(!myIsDigit(s,a1,tag1)) return 0;
        }

        if(tag0 == a1){ //.B
            if(!myIsDigit(s, tag0+1, tag1)) return 0;
            if(tag0 == cn-1)//.
                return false;
        }else if(tag0 > a1){ //B1.B2 [e|EC]
            if(!myIsDigitHasMark(s, a1, tag0)) return 0; // B1
            if(!myIsDigit(s, tag0+1, tag1)) return 0; // B2
            if(tag0 == cn-1){ // B.
                if(!myIsDigit(s, tag0-1, tag0)) return 0;
                // if(tag0 < cn-1) return 0;
            }
        }else{// A
            if(!myIsDigitHasMark(s,a1,tag1)) return 0;
        }

        //积分部分
        if(tag1 < cn){ // 存在积分部分
            if(!myIsDigitHasMark(s, tag1+1, cn)) return false; // 积分后整数
部分
            if(cn - tag1 == 2 && !myIsDigit(s,tag1+1, cn)) return false;
        }

        return true;
    }
};
```

有限状态机

思路：



> 图中缺少部分情况
>
> a/b：a为输入，b为输出
>
> 1. Space : STATE_BEFORE_SAPCE
> 2. F : STATE_SIGN
> 3. D : STATE_INTEGER
> 4. . : STATE_POINT
> 5. D. : STATE_INTEGER_POINT 和STATE_POINT_INTEGER合并
> 6. .D : STATE_POINT_INTEGER
> 7. e|E : STATE_EXP
> 8. e|E F : STATE_EXP_SIGN
> 9. e|E D : STATE_EXP_INTEGER
> 10. endSpace : STATE_AFTER_INTEGER

状态包括：

1. 字符串前space
2. 符号位
3. 整数部分
4. 小数点
5. 整数小数点
6. 小数点整数

> 发现和上一个整数小数点二者遇到各种输入得到的输出一模一样，所以将二者合并

7. 指数符
8. 指数后符号位
9. 指数后整数
10. 字符串后space

```
class Solution {
public:
    enum State{
        STATE_BEFORE_SPACE,
        STATE_SIGN,
        STATE_INTEGER,
        STATE_POINT,
        // STATE_INTEGER_POINT, 发现和 STATE_POINT_INTEGER 一样
        STATE_POINT_INTEGER,
        STATE_EXP,
        STATE_EXP_SIGN,
        STATE_EXP_INTEGER,
        STATE_AFTER_SAPCE
    };
    enum CharType{
        CHAR_SPACE,
        CHAR_SIGN,
        CHAR_POINT,
        CHAR_NUMBER,
        CHAR_ENG,
        CHAR_EXP
    };
    CharType toCharType(char c){
        if(c == ' ') return CHAR_SPACE;
        else if(c == '-' || c=='+') return CHAR_SIGN;
        else if(c == '.') return CHAR_POINT;
        else if(c >= '0' && c <= '9') return CHAR_NUMBER;
        else if(c == 'e' || c == 'E') return CHAR_EXP;
        else return CHAR_ENG;
    }

    bool isNumber(string s){
        unordered_map<State, unordered_map<CharType, State>> transferState{
        {
            STATE_BEFORE_SPACE,
            {
                {CHAR_SPACE, STATE_BEFORE_SPACE},
                {CHAR_SIGN, STATE_SIGN},
                {CHAR_NUMBER, STATE_INTEGER},
                {CHAR_POINT, STATE_POINT}
            }
        },{
            STATE_SIGN,
            {
                {CHAR_POINT, STATE_POINT},
                {CHAR_NUMBER, STATE_INTEGER}
            }
        },{
            STATE_INTEGER,
            {
                {CHAR_NUMBER, STATE_INTEGER},
                {CHAR_POINT, STATE_POINT_INTEGER},
                {CHAR_EXP, STATE_EXP},
```

```
                {CHAR_SPACE, STATE_AFTER_SAPCE}
            }
        },{
            STATE_POINT,
            {
                {CHAR_NUMBER, STATE_POINT_INTEGER}
            }
        },{
            STATE_POINT_INTEGER,
            {
                {CHAR_NUMBER, STATE_POINT_INTEGER},
                {CHAR_EXP, STATE_EXP},
                {CHAR_SPACE, STATE_AFTER_SAPCE}
            }
        },{
            STATE_EXP,
            {
                {CHAR_SIGN, STATE_EXP_SIGN},
                {CHAR_NUMBER, STATE_EXP_INTEGER}
            }
        },{
            STATE_EXP_SIGN,
            {
                {CHAR_NUMBER, STATE_EXP_INTEGER}
            }
        },{
            STATE_EXP_INTEGER,
            {
                {CHAR_NUMBER, STATE_EXP_INTEGER},
                {CHAR_SPACE, STATE_AFTER_SAPCE}
            }
        },{
            STATE_AFTER_SAPCE,
            {
                {CHAR_SPACE, STATE_AFTER_SAPCE}
            }
        }
    };
    int n = s.size();
    if(n<=0) return false;
    if(n == 1){
        if(toCharType(s[0]) == CHAR_NUMBER) return true;
        else return false;
    }
    CharType ct;
    State cs = STATE_BEFORE_SPACE;
    for(int i = 0; i < n; ++i){
        ct = toCharType(s[i]);
        if(transferState[cs].find(ct) == transferState[cs].end() || ct ==
CHAR_ENG){
            return false;
        }
        cs = transferState[cs][ct];
    }
```

```
        if(transferState[cs][CHAR_SPACE] == STATE_AFTER_SAPCE) return true;
        else return false;

        }
    };
```