项目简介

1. 问题

返回长度为 n 的所有非负整数，使得每两个数字之间的绝对值之差为 k。请注意，答案中的每个数字不得有前导 0。例如，01 有一个前导 0，因此无效。结果需要按顺序返回答案。

2. 思路

首先，给定输入为数字串长度 n 和约束条件 k（相邻数字之间的绝对差），此时得出，当前一个数字确定时，后一个数字必定为前一个数字减去 k 或者加上 k，由此可得到第二个数字，再在第二数字的基础上得到第三个数字。以此类推，当得到含 n 个数字的串时，即得到了一个符合要求的结果。

而由于需要输出全部满足条件的数字串，因此考虑按一定的逻辑顺序进行数字串的生成，从第一个到最后一个，即可完成全部输出。所以，为了排除前导 0 的情况，可以首先决定第一个数字，然后依次输出由 1 到 9 开头的所有满足条件的数字串。其中，除了第一个数字外，后面的每一个数字由其前一个数字决定，并且减的优先级大于加的优先级（即当减去 k 不满足条件后，再选择加上 k；加上 k 也不满足条件，说明当前字符串无法生成，再对上一位进行判断调整），当得到以 9 开头的最后一个数字串时，全部输出完毕。

因为这种先进行最深度的尝试，再返回上一位进行调整，然后继续向深度进行尝试的方法和深度优先搜索非常相似，因此考虑以深搜为原型进行修改调整，并选择时间复杂度和空间复杂的都较小的非递归算法（即使用栈）。

3 重要实现

3.1 栈的定义

因为得到的满足条件的数字串存放在栈中，而输出时需要将数字串顺序输入，因此自定义一个 Stack 类，并添加需要的成员函数，以便简化后续的操作。

```cpp
class Stack {//定义栈及其成员函数，作为生成并输出要求数字串的工具。
private:
    int top = -1;
    int a[100];//栈的存储数组，用于存储数字串
public:
    bool isempty();
    void Push_back(int);
    void Pop();
    int Top();
    void print(int);
    void printfir(int);
};
```

```cpp
bool Stack::isempty() {//判断栈是否为空
    if (top == -1)return 1;
    else return 0;
}

void Stack::Push_back(int x) {//将数字压入到栈顶
    top ++;
    a[top] = x;
}

void Stack::Pop() {//将数字从栈顶弹出
    top --;
}

int Stack::Top() {//读取栈顶的数字，不弹出
    return a[top];
}

void Stack::print(int x)//先输出“，”，输出栈内的前x个数字
{
    cout << ",";
    for (int i = 0; i < x; i++)cout << a[i];
}

void Stack::printfir(int x) {//直接输出栈内的前x个数字
    for (int i = 0; i < x; i++)cout << a[i];
}
```

未找到相关问题

3.2 数字串的生成

首先，考虑到，当 n=1 时，必定五满足条件的数字串，因此直接输出空括号，当 n>1 时，进行下一步判断。

```
if (n <= 1)cout << "[]";//当n<=1时，无任何可能满足要求的串，所以无需继续判断，直接输出空
if (n > 1)//当n>1时，作进一步判断
{
    cout << "[";
    for (int i = 1; i <= 9; i++)//做循环，当第一个数字从1到9时，分别输出满足条件的所有数字串
    {
        if (i - k < 0 && i + k>9)continue;
        seq.Push_back(i);//将当前的循环变量i作为首数字放入栈中。
        count = 1;
        while (count < n)//当count小于n时进行循环。首先判断当前栈顶元素减去k是否符合范围，若符合则压入栈中，
        //否则判断加上k是否满足，进而决定是否压入栈中，并使得count加1.
        {
            int m = seq.Top();
            seq.Pop();
            count--;
            if (m - k >= 0) { seq.Push_back(m); seq.Push_back(m - k); count += 2; }
            else if (m + k <= 9) { seq.Push_back(m); seq.Push_back(m + k); count += 2; }
            else break;
        }//以减的优先级大于加的优先级进行生成数字串，并得到第一个数字串，此时栈中有n个元素进行输出。
        if (i == 1)seq.printfir(n);
        else seq.print(n);
```

首先，有一个从 1 到 9 大的循环，考虑到，如果第一位数减去 k 和加上 k 都不满足范围，则不需要继续进行，可以开始下一轮循环。并且，如果第一位数加上 k 或者减去 k 至少有一个满足范围，则必定至少存在一个满足条件的数字串（后面的数轮流加减 k 即可）。

因此，当第二个数字能满足范围时，则以上述的减法优先为原则，生成第一个满足条件的数字串，并将其输出。

```
while (!seq.isempty())//当栈不为空时，进行循环，方式类似于深度搜索的非递归形式，但做出了相应的调整
{
    int x = seq.Top();
    seq.Pop();
    count--;//首先弹出栈顶元素，并记录它的值，count减一
    if (x < seq.Top() && seq.Top() + k <= 9)//当此元素的值小于前一个元素，并且加上2k后的值仍在范围内时，
                                            //进行以下操作，否则完成此轮循环
    {
        seq.Push_back(seq.Top() + k); count++;//将当前栈顶元素加k压入栈顶
        while (count < n)//进行前面生成第一个数字串时的操作
        {
            int m = seq.Top();
            seq.Pop();
            count--;
            if (m - k >= 0) { seq.Push_back(m); seq.Push_back(m - k); count += 2; }
            else if (m + k <= 9) { seq.Push_back(m); seq.Push_back(m + k); count += 2; }
        }
        seq.print(n);//输出数字串
    }
}
```

以上述生成的第一个数字串为基础，以深搜为思路，以减法优先为方法，进行遍历，依次输出所有满足条件的数字串。具体操作可分为以下步骤。

当栈非空时，执行以下循环：

1. 弹出栈顶的元素，并记录其值。
2. 将它与前一个数字进行比较，如果大于前一个数字，则不进行操作，直接进行下一轮循环；如果小于前一个数字，根据减法优先原则，此时前一个数字加 k 的情况尚未遍历到，因此将前一个数字加 k 压入栈中。
3. 以减法优先原则，生成数字串后面的数字，当数字串长度为 n 时，对数字串进行输出，并开始下一轮循环。
4. 当无满足条件的数字串存在时，栈内的数字将依次弹出，没有数字再压入，栈空，第一轮大循环结束。

当 9 轮大循环结束时，所有满足条件的数字串均输出，得到结果，并由用户判断是否继续运行。