《数据结构与算法》实验报告

实验名称	KMP 算法的实现和应用				
姓名	陈思睿	**学号**	21030021007	**日期**	2024.4.29
实验内容	1.调通圆周率生成算法，生成尽可能长的圆周率序列。 2.利用自己实现的 KMP 算法比对圆周率序列中是否存在某个特殊序列并给出位置序号，比如本课程 ID，02003048；自己的生日，YYYYMMDD。				
实验目的	1.（3 分）调试正确圆周率生成函数，并生成“足够长”的圆周率序列 2.（3 分）根据课程内容实现 KMP 模式匹配算法 3.（3 分）尝试搜索匹配课程 ID 或 8 位生日序列 4.*（1 分）尝试搜索匹配其他感兴趣的数字序列				

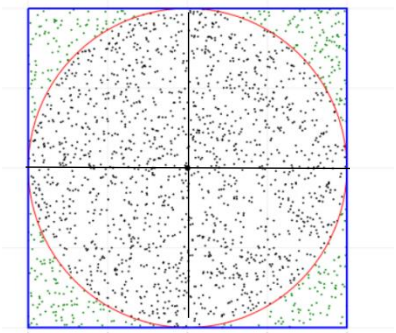
实验步骤

1、 圆周率生成函数

上网查找资料可知，数学中计算圆周率的公式有很多，不同的公式计算圆周率的收敛速度不一，公式复杂程度也不同。

（1）蒙特卡洛方法求圆周率公式

蒙特卡洛方法是根据概率来求解圆周率的方法，例如右图，在边长为 1 的正方形和单位圆中，点（x，y）到（0,0）的距离平方 x^2+y^2<=1 时，说明在圆内，当在 0<x<1,0<y<1 的范围内选择生成点时，可知在四分之一圆中



$$S_{圆}/S_{正}=\pi/4$$

因此可以通过该方式求得圆周率的大小。

```
//计算圆周率函数，采用蒙特卡洛算法求pi
double countPI(int n){
      double pi=0;
      double x,y=0;
      int i=0;
      int count=0;
      srand((unsigned)time(NULL));
      while(i<n){
        x=rand()/(double)(RAND_MAX);
        y=rand()/(double)(RAND_MAX);
        if((x*x+y*y)<=1)
        count++;
        i++;
      }
      pi=4*(double)count/n;
      return pi;
}
```

采用随机数生成方法，随机数范围在 0-RAND_MAX,因此类似于构建一个边长为 RAND_MAX 的正方形和边长为 RAND_MAX 的四分之一圆，投掷 n 次随机数后计算圆周率大小。

但该算法收敛速度较慢，当 n 取一亿时，才计算到圆周率后 3 位，收敛速度太慢了。

```
\Program Files\MinGW64\bin\gdb.exe
100000000
3.14136592000000010000
PS D:\code>
```

实验步骤

（2）利用 π 的级数公式计算

网上查找的 π 的级数公式很多，比较著名的有莱布尼茨收敛公式

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\ldots,$$

$$\pi=\sum_{0}^{\infty}\frac{(-1)^{n}}{(2n+1)}$$

代码如下

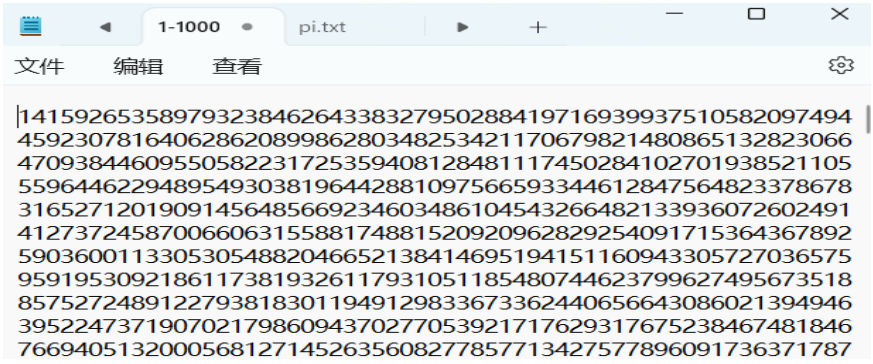
```
double count2Pi(int terms) {
    double pi = 0.0;
    int s = 1; // 符号位，用于交替加减
    int d = 1; // 分母

    for (int i = 0; i < terms; i++) {
        pi += s * (1.0 / d);
        s = -s; // 改变符号
        d += 2; // 下一个分母是当前分母加2
    }

    return 4.0 * pi; // 莱布尼兹级数中的π/4,
}
```

结果

```
输入级数的项数：100000000
用级数展开计算的π的近似值为：3.1415926436
```

但是这些求圆周率的方法位数较少还行，如果计算 10 亿位以上的话显然不行，因此在通过 kmp 算法在圆周率里面找想要的数字时，直接算出圆周率再求解显然很困难，因此，对于较大数字的圆周率求解，我打算直接在网上找已经计算出的圆周率文件，直接读取文件来匹配。

这边直接下载了前 1000000000 位的圆周率。



2. kmp 算法

Next 数组求法

```
// 计算模式串的部分匹配表（next数组）
void get_next(const char pattern[], int next[]) {
    int len = strlen(pattern);
    int i = 0;
    int j = -1;
    next[0] = -1; // next数组中第一个值为-1

    while (i < len - 1) {
        if (j == -1 || pattern[i] == pattern[j]) {
            ++i;
            ++j;
            next[i] = j; // 正确赋值为j
        } else {
            j = next[j];
        }
    }
}
```

Kmp 算法显然已经非常熟悉，利用公共前后缀和来进行匹配，有效的降低了时间复杂度。

```
// 使用KMP算法在文本串S中查找模式串T的位置
int Index_KMP(const char S[], const char T[], const int next[]) {
    int i = 0;
    int j = 0;
    int sLen = strlen(S);
    int tLen = strlen(T);

    while (i < sLen && j < tLen) {
        if (j == -1 || S[i] == T[j]) {
            ++i;
            ++j;
        } else {
            j = next[j];
        }
    }
    if (j == tLen)
        return i - j;
    else
        return -1; // 未找到匹配位置
}
```

3、各类数字串求解（位数为小数点后 X 位）

（1）14159（检查是否正确）

```
const char *pattern = "14159"; // 要查找的数字序列
```

```
找到目标数字序列在圆周率中的位置：1
PS D:\code>
```

（2）新中国成立日子

```
const char *pattern = "19491001"; // 要查找的数字序列
```

```
\Program Files\mingw64\bin\gdb.exe' '--interprete
找到目标数字序列在圆周率中的位置：82267377
PS D:\code>
```

（3）本人生日

6 位串能找到

```
const char *pattern = "021108"; // 要查找的数字序列
```

```
\Program Files\mingw64\bin\gdb.exe' '--interpreter=mi'
找到目标数字序列在圆周率中的位置：1001077
PS D:\code>
```

8 位串（20021108）

```
未找到目标数字序列在圆周率中
PS D:\code>
```

应该在 1 亿位之后了……

（4）选课号

```
const char *pattern = "02003046"; // 要查找的数字序列
```

```
找到目标数字序列在圆周率中的位置：17837021
PS D:\code>
```

4、源代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 计算模式串的部分匹配表（next 数组）
void get_next(const char pattern[], int next[]) {
    int len = strlen(pattern);
    int i = 0;
    int j = -1;
    next[0] = -1;

    while (i < len - 1) {
        if (j == -1 || pattern[i] == pattern[j]) {
            ++i;
            ++j;
            next[i] = j;
        } else {
            j = next[j];
```

```
        }
    }
}

// 使用 KMP 算法在文本串 S 中查找模式串 T 的位置
int Index_KMP(const char S[], const char T[], const int next[]) {
    int i = 0;
    int j = 0;
    int sLen = strlen(S);
    int tLen = strlen(T);

    while (i < sLen && j < tLen) {
        if (j == -1 || S[i] == T[j]) {
            ++i;
            ++j;
        } else {
            j = next[j];
        }
    }

    if (j == tLen) {
        return i - j; // 返回模式串在文本串中的起始位置
    } else {
        return -1; // 未找到匹配位置
    }
}

int main() {
    FILE *file;
    char *pi_digits = NULL;
    long file_size;
    const char *pattern = "123456789"; // 要查找的数字序列
    int pattern_length = strlen(pattern);
    int *next = (int *)malloc(sizeof(int) * pattern_length);

    // 打开文件
    file = fopen("D:\\pi.txt", "r");
    if (file == NULL) {
        fprintf(stderr, "无法打开文件\n");
        return 1;
    }

    // 获取文件大小
    fseek(file, 0, SEEK_END);
```

```
    file_size = ftell(file);
    rewind(file);

    // 分配内存并读取文件内容
    pi_digits = (char *)malloc(file_size + 1);
    if (pi_digits == NULL) {
        fprintf(stderr, "内存分配失败\n");
        fclose(file);
        return 1;
    }

    fread(pi_digits, 1, file_size, file);
    pi_digits[file_size] = '\0'; // 添加字符串结束符

    fclose(file);

    // 计算模式串的部分匹配表（next 数组）
    get_next(pattern, next);

    // 在圆周率字符串中查找目标数字序列
    int pos = Index_KMP(pi_digits, pattern, next);

    if (pos != -1) {
        printf("找到目标数字序列在圆周率中的位置：%d\n", pos+1);
    } else {
        printf("未找到目标数字序列在圆周率中\n");
    }

    free(pi_digits);
    free(next);

    return 0;
}
```

实验总结

本次实验利用圆周率来作为文本串求解各类数字串，复习了 kmp 算法，同时也了解到了各类求圆周率的方法，另外也在圆周率中找到各类数字串，趣味性十足。