题目：生物编码

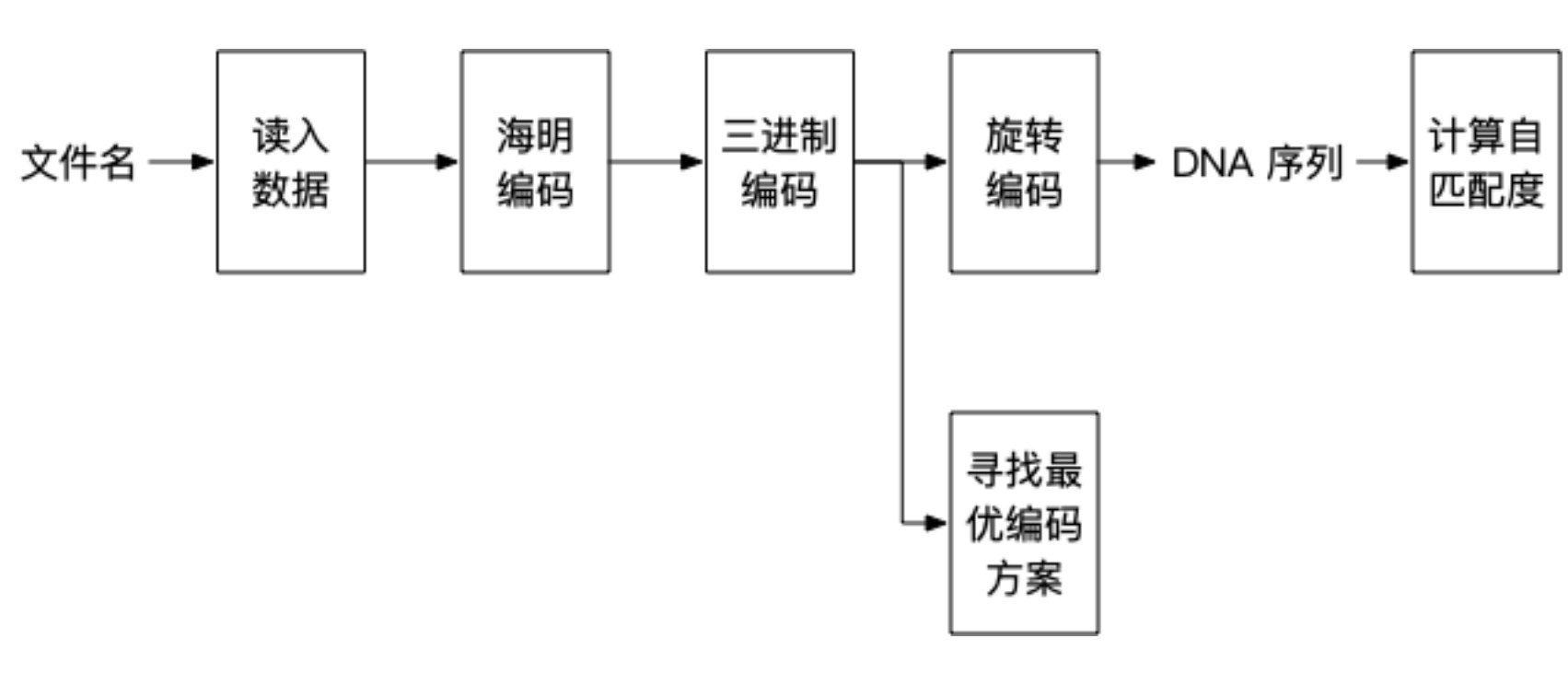
基本要求

- 语言不限，请根据题目选择合适的语言。
- 图形化框架不限，有特殊要求的请在提交时加入 `README.txt` 文件注明。
- 请将需要提交的文件放入指定目录（文件夹）中，并压缩后提交。
- 对于样例测试和小规模测试，测试环境为 2 CPU 核心，4 GB 内存，运行时间要求参加评分标准。

题目背景和概览

DNA 存储是一种新型存储技术，其使用 A、T、C、G 四种碱基保存编码后的数据。 在本题目中，你需要读取给定的文件数据，并对其内容进行编码，转换为用于 DNA 存储的碱基序列，并进行一系列相关的计算。

总体来说，本题目中你需要做的事情如下图所示：



步骤与具体要求

0. 读入文件名

在本步骤中，你需要打印一行字符串，提示用户输入文件路径。 字符串内容为：“Please enter the input file path:”（不包括双引号）。 注意该字符串占一行。

此后，程序从标准输入中读入一个字符串，该字符串中不含空格，为一个输入文件的路径。 该路径可能为相对路径，也可能为绝对路径，但是一般情况下你的程序不需要对此进行特殊判断。

1. 读入文件数据，转换成二进制编码

在本步骤中，你需要打开文件，从中读取数据，并以字节为粒度将文件内容转换成二进制编码。

该步骤需要实现在一个名为 `LoadBinary` 的函数中，对于 C++ 来说，函数签名应符合以下形式（若使用其他语言，应具有相似的函数签名，以下各步骤要求与此相同）：

```
string LoadFile(const string &filepath);
```

该函数接受一个文件路径，返回该文件中数据的二进制的字符串表示。

- 注意，测试文件中可能会包含图片等二进制数据，因此你可能需要使用二进制方式打开文件。
- 在第四步的末尾，有前四步的一个综合样例。

【阶段性输出】

为了检测正确性，除了返回文件数据之外，在此步骤的 `LoadFile` 函数中，你还应对二进制中 0 和 1 的个数进行统计，并将这两个个数以如下形式打印到标准输出（注意空格），共占一行。

```
Phase 1: <0的个数> <1的个数>
```

2. 使用海明码增强容错性

海明码是一种用于可以增强容错性的编码，在本题目中我们所使用的海明码编码方法如下：

1. 将二进制数据每 7 个比特为一组进行分组，最后一组不足 7 个比特的，使用 0 进行补全；
2. 对于每组 7 个比特，从左到右记为 d1 到 d7，使用下列公式计算出 5 个校验比特；

   ```
   p1 = d1 XOR d2 XOR d4 XOR d5 XOR d7;
   p2 = d1 XOR d3 XOR d4 XOR d6 XOR d7;
   p3 = d2 XOR d3 XOR d4;
   p4 = d5 XOR d6 XOR d7;
   p5 = d1 XOR d2 XOR d3 XOR d4 XOR d5 XOR d6 XOR d7 XOR p1 XOR p2 XOR p3 XOR p4;
   ```

3. 将每组的 7 个比特和计算出的 5 个比特按照特定顺序组合起来；

   ```
   p1 p2 d1 p3 d2 d3 d4 p4 d5 d6 d7 p5
   ```

4. 将多组数据按照原顺序拼接在一起。

本步骤应实现在一个名为 `HammingEncode` 的函数中，函数签名形式为：

```
string HammingEncode(const string &binary);
```

该函数接受一个二进制比特串，并对其进行海明编码。

【阶段性输出】

为了检测正确性，在此步骤的 `HammingEncode` 函数中，你还应对编码后的二进制中 0 和 1 的个数进行统计，并将这两个个数以如下形式打印到标准输出，共占一行。

```
Phase 2: <0的个数> <1的个数>
```

3. 二进制到三进制编码

在本步骤中，你需要二进制编码变为三进制编码。具体来说，需要将每 9 个二进制比特位（能表示 0~511），转变 6 个三进制比特位（能表示 0~728），对于最后不足 9 位的，使用二进制 0 补足到 9 位。

本步骤应实现在一个名为 `TriEncode` 的函数中，函数签名形式为：

```
string TriEncode(const string &binary);
```

【阶段性输出】

为了检测正确性，在此步骤的 `TriEncode` 函数中，你还应对生成的三进制中 0、1 和 2 的个数进行统计，并将这三个个数以如下形式打印到标准输出，共占一行。

```
Phase 3: <0的个数> <1的个数> <2的个数>
```

4. 使用旋转编码转换为 DNA 序列

所谓旋转编码，是指生成的编码中相邻的两个碱基必定会不同。具体地，我们使用如下的旋转编码将三进制数据转换为 DNA 碱基序列：

- 根据第一个三进制比特决定第一个碱基，0 对应 A，1 对应 T，2 对应 C；
- 此后的每个碱基，由前一个碱基和当前的三进制比特共同决定，具体规则如下表所示，先根据前一个碱基确定一行，再根据当前三进制比特位找到对应的碱基。如当前一个碱基为 T 时，如果当前要转换的三进制比特为 2，则其应该被翻译为碱基 A。

前一个碱基	0	1	2
A	T	C	G
T	C	G	A
C	G	A	T
G	A	T	C

根据上述方法，三进制比特序列 `20101` 应被翻译为：`CGTCA`。

本步骤应实现在一个名为 `BioEncode` 的函数中，且接受一个三进制比特串，并返回编码成的 DNA 碱基序列，其函数签名形式为：

```
string BioEncode(const string &trinary);
```

【文件输出】

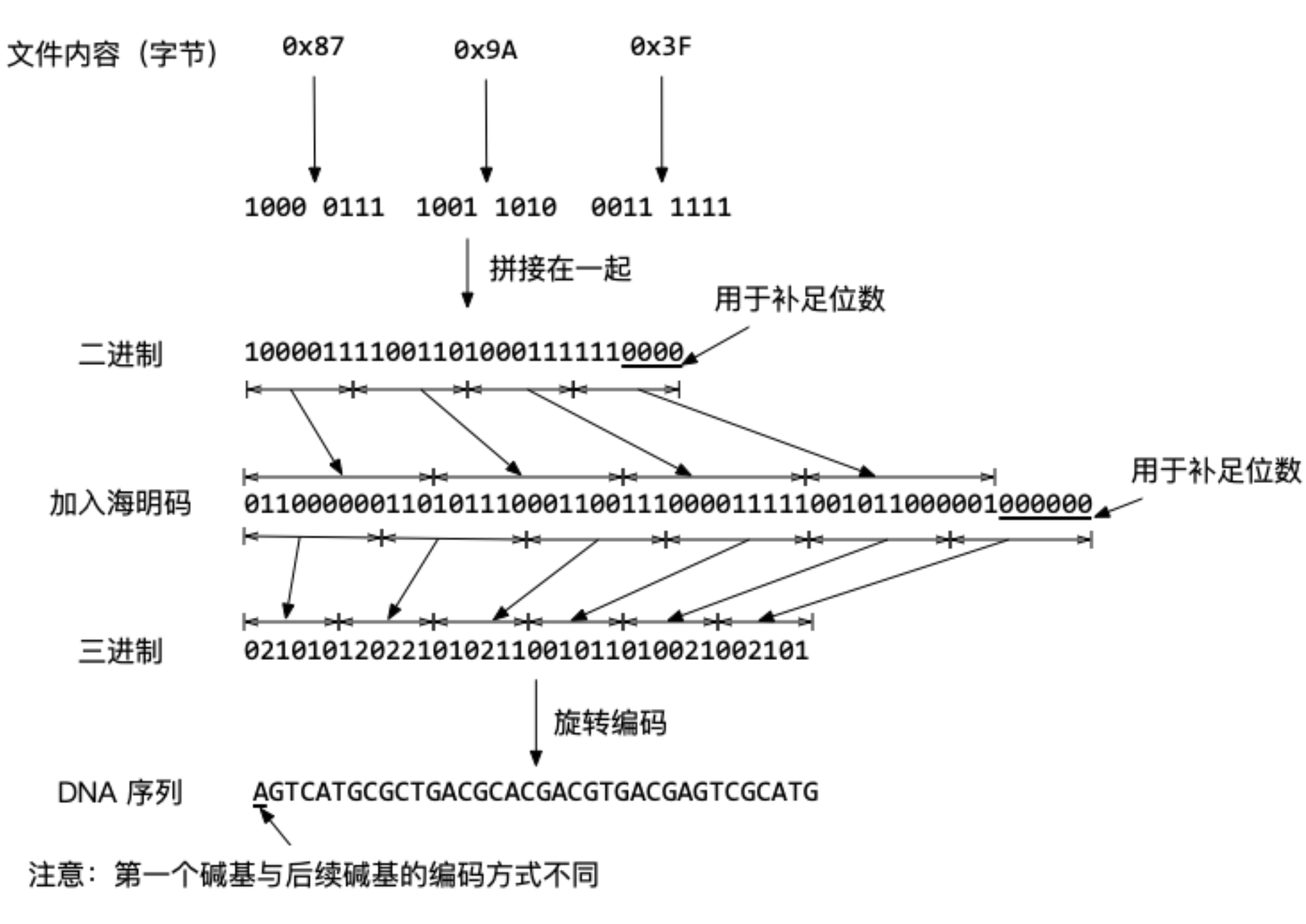
在本函数中，除了要返回该 DNA 碱基序列之外，还需要将该 DNA 碱基序列写入名为 `DNA.txt` 的文件中，共占一行。该文件保存到当前工作目录即可（即 `./DNA.txt`）。

【阶段性输出】

为了检测正确性，在此步骤的 `BioEncode` 函数中，你还应对生成的碱基序列中 A、T、C、G 的个数进行统计，并将这四个个数以如下形式打印到标准输出，共占一行。

```
Phase 4: <A的个数> <T的个数> <C的个数> <G的个数>
```

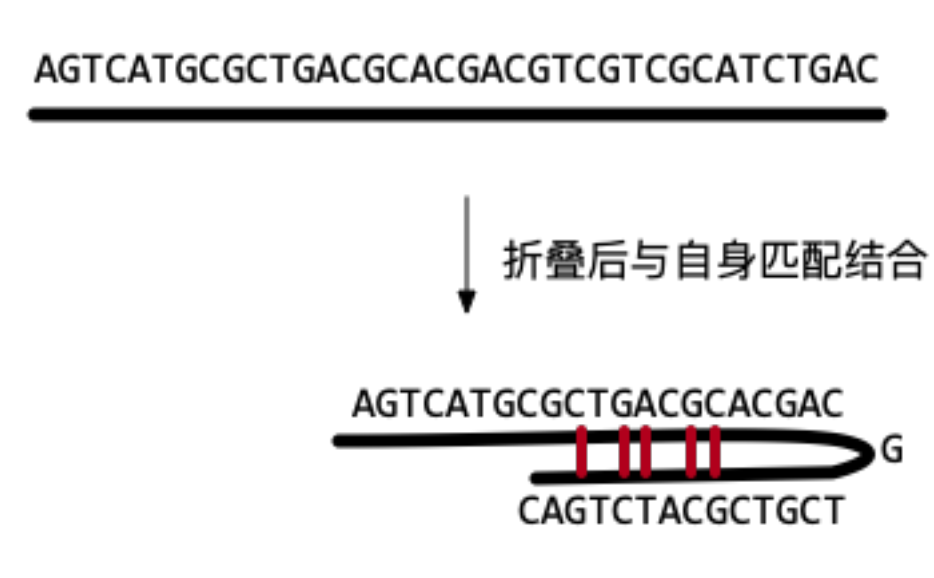
前四步骤的一个综合样例如下：



5. 计算自匹配度

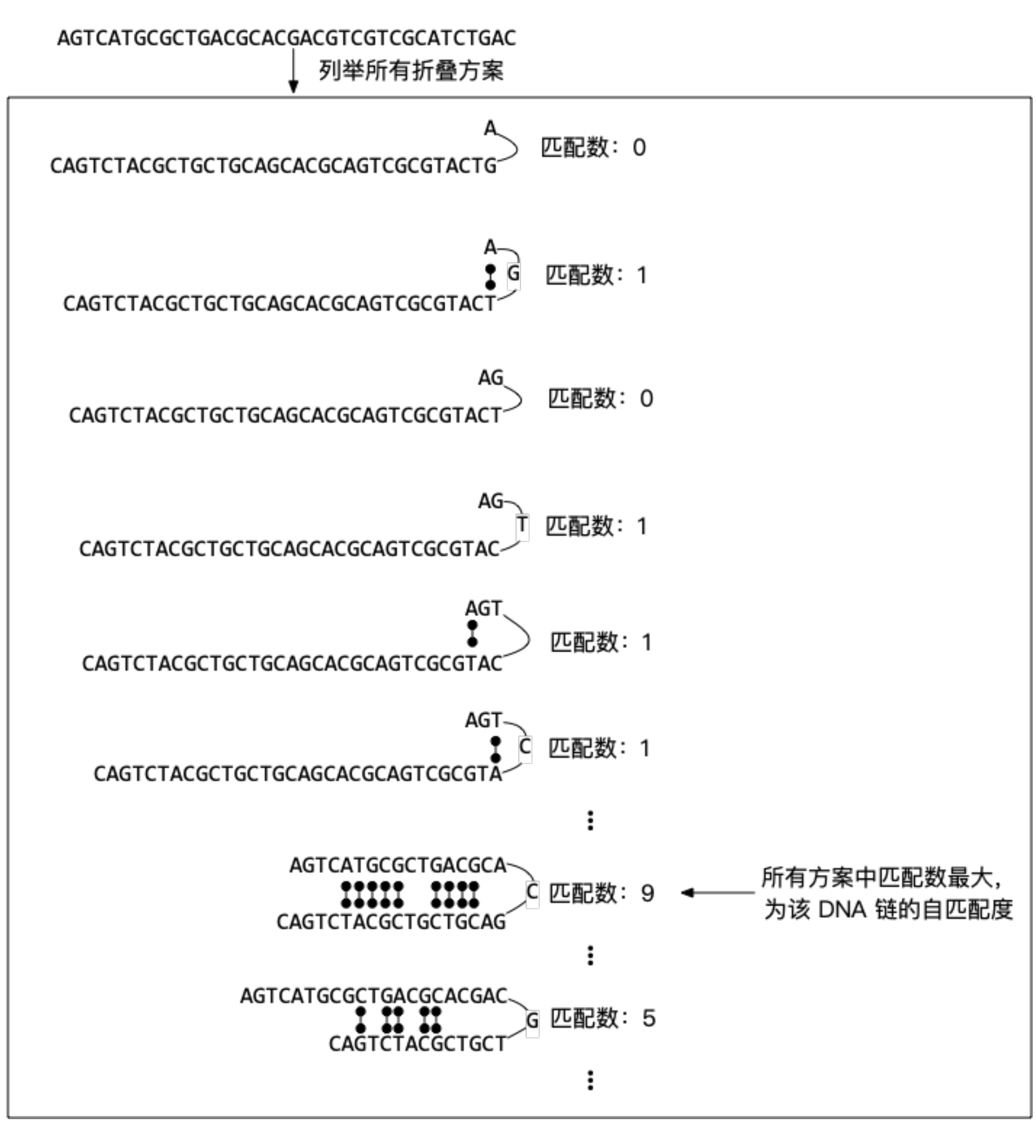
一条单链 DNA 可能会与自身结合，形成比较复杂的结构。在本题目中，我们只考虑一种简化的情况，即 DNA 链（即前面生成的碱基序列）进行对折后与自身的匹配结合情况。

如下图所示，一个单链 DNA 通过对折后，自己不同部位的碱基可能配对结合，结合规则为 **A 与 T 结合，C 与 G 结合**，因此在以下图中所示方法进行对折后，能够产生 5 对匹配的碱基对。



我们定义一条 DNA 序列的**自匹配度（SAV）**为**该 DNA 序列在所有对折情况下最大的匹配数**。

为了更加清楚地说明“对折”，我们在下图中给出了多种对折的方案作为示例。一个 DNA 链会以其链上的某个位置为中心进行“对折”。该位置可能是某两个相连的碱基之间（例如下图中的第一个对折方案中，在 A 和 G 之间进行了对折），也可以是某个碱基（例如下图中第二个方案中，按照 G 进行了对折）。当以某个碱基为中心进行对折时，该碱基不参与匹配。



在本步骤中，你需要对第五步中生成的 DNA 序列的自匹配度进行计算，并打印结果。

本步骤应实现在一个名为 `PrintSAV` 的函数中，其签名如下：

```
void PrintSAV(const string &dna);
```

该函数中应计算参数 DNA 的自匹配度（SAV），并将其打印到标准输出。

【阶段性输出】

此步骤需要输出的自匹配度格式如下，共占一行。

```
Phase 5: SAV = <自匹配度数值>
```

6. 计算最优编码方案

在第 4 步中，我们使用给定的旋转编码将三进制数据转换成了 DNA 序列，并在第 5 步中计算出该 DNA 序列的自匹配度。对于同一个三进制数据，使用不同的旋转编码方案，会得到不同的 DNA 序列。这些序列的自匹配度也可能会有所不同。

在本步骤中，你需要找到一种最优的旋转编码方案，使用该方案对前述三进制编码（即方案 3 生成的三进制编码）进行转换后，得到的 DNA 序列的自匹配度最小。如果有多种旋转编码方案符合要求，则输出生成 DNA 序列字典序最小的那个。

本步骤应实现在一个名为 `BestEncodingPlan` 的函数中，该函数签名形式为：

```
void BestEncodingPlan(const string &trinary);
```

【阶段性输出】

此步骤需要输出不同，共占一行。

```
Phase 6: Best SAV = <最优sav>; A: <A后编码方案>, T: <T后编码方案>, C: <C后编码方案>, G: <G后编码方案>
```

其中每个碱基后的编码方案，为该碱基之后的 0、1、2 分别对应的碱基字母。

如果步骤 4 表格中给出的编码方案恰好为最优方案，且最优 SAV 为 12，则该步骤输出应为：

```
Phase 6: Best SAV = 12; A: TCG, T: CGA, C: GAT, G: ATC
```

注意，在选择最优方案过程中，第一个碱基的生成规则始终不变，即 0 对应 A，1 对应 T，2 对应 C；在第 4 步中所使用的编码方法中，恰好是 ATCG 的一个循环，在选择最优编码的过程中，并不需要所有的编码组成循环，即 `A: TGC, T: GCA, C: ATG, G: ATC` 也是一种合法的方案。

【文件输出】

在本函数中，你还需要将上述输出写入名为 `Best.txt` 的文件中，共占一行。

图形化实现

前文中我们需要编写一个命令行程序。除了该命令行程序外，你还需要实现一个图形化程序，完成相同的功能。在该图形化程序的界面如下图所示。在点击按钮选择了文件之后，应更新“当前选择的文件”，在开始编码被点击之后，应读取文件进行编码和计算，更新界面上的其余信息。在图形化程序中，不要求在标准输出和 `DNA.txt` 文件的中进行输出。**请注意图形化实现的分数占比，合理安排考试时间。**

当前选择的文件为：（未选择文件）

自匹配度为：（未计算）

最优编码方案为：（未计算）

最优自匹配度为：（未计算）

测试样例

sample.txt 文件内容（注意该文件末尾有 `\n` 换行符，如果你是要编辑器打开了该文件，请确保该文件后的换行符不会发生变化）

```
Hello BioEncoding!
```

其内容的字节表示（十六进制，共 19 个字节）

```
48 65 6c 6c 6f 20 42 69 6f 45 6e 63 6f 64 69 6e 67 21 0a
```

标准输入内容：

```
sample.txt
```

正确的标准输出：

```
Please enter the input file path:
Phase 1: 80 72
Phase 2: 138 126
Phase 3: 78 57 45
Phase 4: 46 47 49 38
Phase 5: SAV = 34
Phase 6: Best SAV = 27; A: CTG, T: GCA, C: TAG, G: CTA
```

正确的 `DNA.txt` 文件内容：

```
ATGCTCGATGCTGCGACGAGATAGATACGCACTCACGAGAGAGATATGTACAGTCATACTCTCGATCTATCACTATCAGATCACATCAGTGT
CGCTGTCTCTGCGTATGTCTACGTGATCTCGTGTCATCGCGCAGATCATACACACGCTCGTCGATATCACATGCGATGCACATCTGAC
```

正确的 `Best.txt` 文件内容：

```
Phase 6: Best SAV = 27; A: CTG, T: GCA, C: TAG, G: CTA
```

评分标准

本题目总分 100 分，各个步骤占分如下。**如果你无法通过测试，依然可以根据所写代码得到部分分数。**

步骤	分数	备注
1	10	
2	10	
3	10	
4	15	
5	10	
6	15	
样例测试 (sample.txt)	5	运行时间限时 10 秒，要求标准输出和生成的 `DNA.txt` 的正确性
小规模测试 (small.txt)	10	运行时间限时 30 秒，要求标准输出和生成的 `DNA.txt` 的正确性
大规模测试 (large.jpg)	10	仅要求提交的 `large-DNA.txt` 和 `large-Best.txt` 的正确性，分别对应 large 测试中的 `DNA.txt` 和 `Best.txt` 。
图形化实现	5	

提交要求

提交文件应包括：

- 命令行程序源代码；
- 图形化程序源代码，主文件（即包含程序入口函数的文件）应保存在名为 `gui` 的目录中，该目录中可有内部结构，允许引用命令行程序所使用的类库/头文件；
- 如果有运行的特殊要求，请同时提交一个 `README.txt` 文件，并在其中注明；
- 大规模测试的 `large-DNA.txt` 和 `large-Best.txt`，用于检验大规模测试的正确性。

所有文件应**放在以申请编号命名的文件夹中**，并将这个文件夹**压缩为 7z 文件，命名为** **`申请编号-姓名.7z`**，例如 `210001-张三.7z`。