Thao tác với file và sinh test trong C++

Ban chuyên môn Tin Học – The Gifted Battlefield

Xuất bản vào Ngày 9 tháng 10 năm 2023

Mục lục

1 LỜI MỞ ĐẦU

- Việc đọc xuất dữ liệu ở file rất quan trọng trong lập trình nói chung. Trong lập trình thi đấu, việc thành thạo các kỹ năng thao tác file còn là lợi thế lớn, giúp ta tự chủ động kiểm tra được bộ code của mình. Ta có thể giảm thiểu những sai sót trong lúc làm bài xuống mức tối đa chỉ nhờ vào việc tự sinh test và tìm ra những test sai. Vậy kỹ thuật sinh test là gì, liệu nó có khó để thực hiện không, bài viết hôm nay sẽ chia sẻ và giúp các bạn làm chủ nó ở ngôn ngữ C++.
- Bài viết này chỉ đề cập đến những khái niệm, kiến thức cơ bản vừa đủ để người đọc ứng dụng vào công việc thao tác với file trong C++. Các tác giả sẽ cố gắng không đi quá sâu về mặt kỹ thuật cũng như chi tiết cách hoạt động để khiến cho bài viết dễ tiếp cận nhất có thể.

2 THAO TÁC TRÊN FILE

2.1 Hàm freopen()

- Hàm freopen() thường được sử dụng trong các bài tập lập trình thi đấu (CP) cần sự tương tác (đọc/viết) giữa code với file. Một trong những ứng dụng phổ biến nhất của hàm chính là chức năng "điều hướng" *cin*, *cout* (nhập xuất chuẩn từ terminal) thành nhập, xuất vào file được chỉ định.
- Hàm freopen() thuộc thư viện <cstdio> (nằm trong thư viện chuẩn STL C++).
- **Cú pháp hàm:**
  - **filename** (chuỗi): Tên file mà hàm sẽ mở.
  - **mode** (chuỗi): Khai báo chế độ truy cập của code vào file.

```
FILE *freopen(const char* filename, const char* mode, FILE* stream)
```

MỘT SỐ MODE CƠ BẢN TRONG HÀM FREOPEN()			
Mode	**Ý nghĩa**	**File tồn tại**	**File không tồn tại**
"r"	Mode nhập / đọc từ file	Đọc từ đầu file	Lỗi
"w"	Mode ghi / xuất đè ra file	Xóa tất cả nội dung trong file ngay khi mở file và bắt đầu viết.	Tạo 1 file mới (trống) có tên "filename", chung folder với file '.cpp'.
"a"	Mode ghi / xuất ra cuối file	Không tác động đến nội dung đã có sẵn. Bắt đầu viết/chèn nội dung vào cuối file.	Tạo 1 file mới (trống) có tên "filename", chung folder với file '.cpp'.

  - **stream**: Luồng nhập xuất được sử dụng để đọc/ghi vào file. Thông thường, để đọc từ file "read.txt" bằng cin và in ra file "write.txt" bằng cout, ta sẽ làm như sau:

```
freopen("read.txt", "r", stdin); //stdin là stream nhập chuẩn
freopen("write.txt", "w", stdout); //stdout là stream xuất chuẩn
```

  - **Kết quả trả về**: FILE*. Trong phạm vi bài viết sẽ không đề cập chi tiết cách sử dụng kết quả trả về này.
- **Cách hoạt động**: Trước hết, freopen() đóng file/luồng nhập xuất đang được mở bởi *stream*, và sau đó mở file được chỉ định bởi *filename* ở chế độ *mode* (file phải nằm chung folder với code .cpp đã gọi hàm freopen). Cuối cùng, hàm sẽ gán file với luồng nhập xuất *stream*.

Ví dụ 1: Đọc 1 số nguyên dương n và n số nguyên dương từ file "sum.inp", xuất ra tổng của n số nguyên dương đó ra file "sum.out".

sum.inp	**sum.out**
4 3 6 7 4	20

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    freopen("sum.inp", "r", stdin);
    freopen("sum.out", "w", stdout);

    int n;
    cin >> n;

    int sum = 0;
    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;
        sum += x;
    }

    cout << sum;

    return 0;
}
```

Với test mẫu trên, đoạn code trên sẽ lần lượt đọc vào các số 4 3 6 7 4 từ file sum.inp và in ra file sum.out số 20.

Ví dụ 2: Đọc 1 số a từ file "a.inp" và 1 số b từ file "b.inp", xuất ra tổng 2 số đó vào file "sum.out" và hiệu 2 số vào file "diff.out".

a.inp	**b.inp**	**sum.inp**	**diff.out**
100	1	101	99

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
    freopen("a.inp", "r", stdin);
    cin >> a;
    freopen("b.inp", "r", stdin);
    cin >> b;

    freopen("sum.out", "w", stdout);
    cout << a + b;
    freopen("diff.out", "w", stdout);
    cout << a - b;

    return 0;
}
```

2.2 Object fstream

- Khái niệm (data) stream: Là luồng dữ liệu vào và ra của 1 chương trình C++, chẳng hạn như luồng dữ liệu đọc từ cin hoặc luồng dữ liệu ghi ra từ cout.

Ifstream & ofstream

- *Ifstream* và *ofstream* đều là các *data stream* ***class***. Object (có thể hiểu như biến) của 2 *class* này, tương tự như *cin* và *cout*, là các *data stream* ***object***, có chức năng đọc và ghi dữ liệu vào/ra từ luồng được chỉ định sẵn:
  - *cin* có chức năng đọc dữ liệu từ luồng nhập chuẩn *(stdin)*.
  - *cout* có chức năng đọc dữ liệu từ luồng xuất chuẩn *(stdout)*.
  - Object của 2 class *ifstream*, *ofstream* lần lượt có chức năng đọc, ghi dữ liệu ra *file stream* (luồng nhập xuất file) được chỉ định.
- Việc nhập/xuất dữ liệu bằng *data stream object* thường được thực hiện bằng operator « hoặc ».

Sử dụng ***ví dụ 1*** ở phần trên, ta có thể code nhập xuất file bằng *ifstream* và *ofstream* như sau:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    ifstream in("sum.inp"); /*Tạo object mang tên 'in' thuộc class ifstream (có chức năng đọc dữ liệu), trỏ đến file sum.inp*/
    ofstream out("sum.out"); /*Tạo object mang tên 'out' thuộc class ofstream (có chức năng ghi dữ liệu), trỏ đến file sum.out*/

    int n;
    in >> n; //Đọc dữ liệu bằng operator ">>"

    int sum = 0;
    for (int i = 1; i <= n; i++) {
        int x;
        in >> x;
        sum += x;
    }

    out << sum; //Ghi dữ liệu bằng operator "<<"

    in.close(); /*Đóng file stream đang được mở bởi ifstream object 'in'. Đây là 1 thói quen tốt trong lập trình*/
    out.close();

    return 0;
}
```

Ta cũng có thể code ***ví dụ 2*** ở phần trên như sau:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
    ifstream in; /*Khai báo object 'in' thuộc class ifstream, chưa chỉ định file stream cụ thể cho 'in'*/
    ofstream out; /*Khai báo object 'out' thuộc class ofstream, chưa chỉ định file stream cụ thể cho 'out'*/

    in.open("a.inp"); /*Trỏ 'in' đến file a.inp*/
    in >> a;
    in.close();
    in.open("b.inp");
    in >> b;
    in.close();

    out.open("sum.out");
    out << a + b;
    out.close();
```

```
    out.open("diff.out");
    out << a - b;
    out.close();

    return 0;
}
```

- ***Lưu ý:***

```
    ifstream in;
    in.open("file.txt");
```

cũng có chức năng tương đương

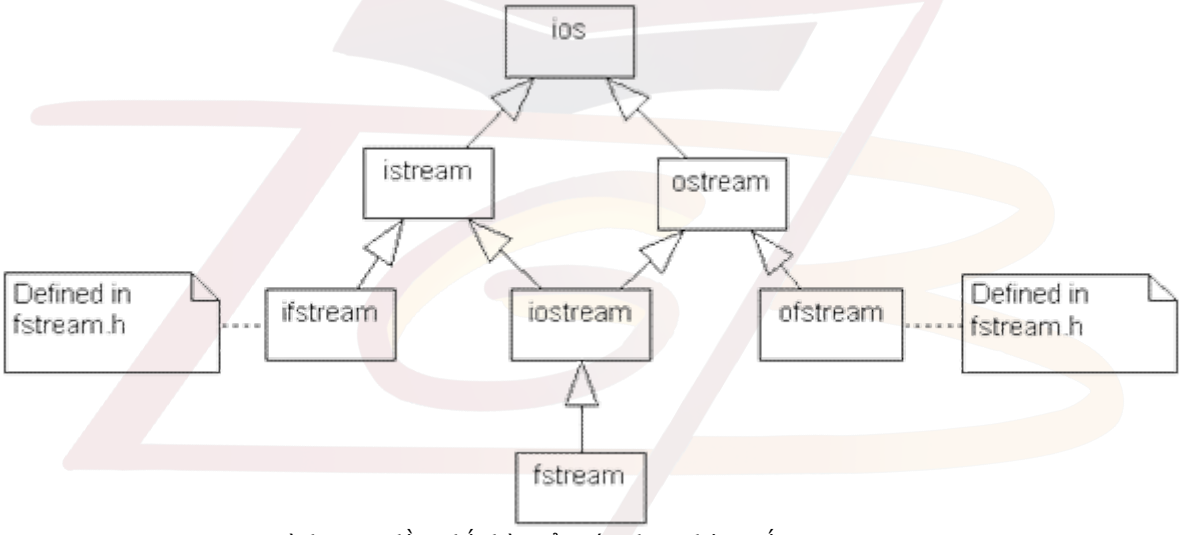
```
    ifstream in("file.txt");
```

Tương tự với ofstream.

- ***Tham khảo thêm***: Một số khái niệm, phương thức nâng cao, mở rộng hơn của ifstream và ofstream.

Class istream và ostream

- Istream và ostream lần lượt là các class xử lý luồng dữ liệu vào, ra của chương trình C++. Class ifstream được kế thừa từ istream và ofstream được kế thừa từ ostream.



Hình 1. Sơ đồ sự kế thừa của các Class nhập xuất trong C++

- Nói cách khác, một object thuộc class ifstream (hoặc ofstream) cũng chính là một object thuộc class istream (hoặc ostream). cin cũng là một object thuộc istream và cout là object thuộc ostream.

Ứng dụng 1: Truyền tham số vào hàm linh hoạt hơn.

```
#include <bits/stdc++.h>
using namespace std;

void print(int n, ostream& out) {
    //Print 1...n
    for (int i = 1; i <= n; i++) {
        out << i << ' ';
    }

    out << endl;
}

int main() {
    ofstream output_file("nums.txt");

```

```
    print(10, output_file);

    return 0;
}
```

Chương trình trên sẽ in ra dãy số từ 1 đến 10 vào file nums.txt. Hàm print() trên cũng có thể được sử dụng như sau:

```
print(15, cout);
```

Code này sẽ in ra dãy số từ 1 đến 15 ra luồng xuất chuẩn (thường là terminal).

Ứng dụng 2: Sử dụng operator overloading để in ra pair hoặc user-defined object.

```
#include <bits/stdc++.h>
using namespace std;

void operator << (ostream& out, pair<int, int> p) {
    out << p.first << ' ' << p.second;
    return;
}

int main() {
    pair<int, int> hello(1, 2);
    cout << hello;

    ofstream output_file("pair.txt");
    output_file << hello; //in pair 'hello' ra file pair.txt

    return 0;
}
```

Tài liệu

[1] C++ Files and Streams - *https://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm*

[2] Sử Dụng File Siêu Cơ Bản Với C++ - *https://codelearn.io/sharing/su-dung-file-sieu-co-ban-voi-cpp*

[3] File Handling through C++ Classes - GeeksforGeeks - *https://www.geeksforgeeks.org/file-handling-c-classes/*

3 SINH TEST

3.1 Tính chất ngẫu nhiên trong C++

Ngẫu nhiên là gì, tại sao cần sự ngẫu nhiên

- Trong lập trình thi đấu, việc gặp phải những vấn đề như code chạy quá thời gian (Time limit exceeded), Lỗi Runtime (Runtime error) hoặc in ra sai kết quả (Wrong answer) là điều không thể tránh khỏi. Với một số người, họ sẽ chỉ đọc lại code hoặc tự sinh ra một số "test tay" để kiểm tra code. Tuy nhiên, với những lỗi chỉ xảy ra ở một số trường hợp nào đó, làm theo cách này hoàn toàn không hiệu quả trong việc tìm ra bug bởi vì những test tự sinh thường sẽ không bao quát các trường hợp. Để có thể tạo ra được nhiều bộ test, đặc biệt là các bộ test lớn, ta không thể nào chỉ sinh ra các test từ bàn phím được. Do đó, việc sử dụng các bộ test có các số được sinh ngẫu nhiên là một điều thiết yếu. Vậy trước hết, chúng ta cần hiểu ngẫu nhiên là gì?
- Nói một cách ngắn gọn, sự ngẫu nhiên là một sự kiện không tuân theo một quy luật nhất định và không dễ đoán trước được. Trong lập trình, một dãy số ngẫu nhiên là các số được được tạo ra với mỗi số có xác suất xuất hiện là như nhau và kết quả của chúng sẽ **độc lập**, tức là kết quả của 2 số đôi một bất kỳ trong dãy sẽ không phụ thuộc vào nhau. Để sinh ra một số ngẫu nhiên thì sẽ có rất nhiều cách khác nhau được chia thành 2 loại chính: Pseudorandom number generator (deterministic random bit generator) và True Random Number Generator (non-deterministic random bit generator). Trong phần này, mình sẽ chú ý nhiều vào Pseudorandom number generator.

■ Randomizer hoạt động như thế nào

- Pseudorandom number generator (PRNG) hay là một thuật toán sinh các số gần như ngẫu nhiên thông qua các phép toán đủ phức tạp để khó có thể đoán trước kết quả. Lý do mình dùng từ "gần như" là bởi vì dãy số tạo bởi PRNG được tạo ra hoàn toàn phụ thuộc vào một tham số truyền vào gọi là "seed". Do đó, nếu chúng ta bằng một cách nào đó có thể tìm ra được seed và công thức toán phức tạp kia thì chúng ta hoàn toàn có thể đoán trước được đáp án.

■ Các hàm, object random trong C++

- Trong C++, có 2 cách được sử dụng phổ biến để sinh một số ngẫu nhiên, đó là rand() và mt19937

1. rand()

Hàm rand() là một hàm sinh số ngẫu nhiên rất dễ sử dụng nhờ vào việc cú pháp ngắn gọn. Khi hàm rand() được gọi, giá trị được trả ra là một số nguyên ngẫu nhiên từ 0 cho tới RAND_MAX. Tuy nhiên, theo CPP Reference, giá trị của hằng số RAND_MAX này phụ thuộc vào compiler / máy và giá trị RAND_MAX chỉ được đảm bảo tối thiểu là $32767 (2^{15} - 1)$. Do đó, ở một số website, ví dụ như Codeforces, giá trị của RAND_MAX là đúng 32767, khiến cho việc sử dụng rand() trong code không được hiệu quả. Thông thường, khi sử dụng hàm rand(), mọi người thường sử dụng nó chung với hàm srand() để có thể đặt seed cho rand() và chọn seed là time(0) để solution khó bị hack hơn. Để in ra một số ngẫu nhiên bằng hàm rand(), ta sẽ cài đặt như sau:

```cpp
#include <iostream>

using namespace std;

int main(){
    srand(time(0));
    cout << rand();
    return 0;
}
```

2. mt19937

mt19937 Là một class thuộc thư viện <random> trong C++ hỗ trợ việc tạo Pseudorandom numbers một cách hiệu quả bằng thuật toán Mersenne Twister. Lúc sử dụng, nó sẽ tạo ra một dãy các số nguyên 32-bit và chỉ lặp lại sau khi đã sinh ra $2^{19937} - 1$ số. Con số này còn được gọi là Mersenne prime. So với rand(), mt19937 có tốc độ sinh số ngẫu nhiên nhanh hơn cũng như có thể sinh số ngẫu nhiên trong khoảng $[0, 2^{32} - 1]$ thay vì chỉ là $[0, 32676]$ như rand(). Nếu muốn tăng giới hạn tối đa của mt19937 lên $2^{64} - 1$, ta sẽ sử dụng mt19937_64.

Để khai báo một hàm rd() dùng thuật toán Mersenne Twister, ta sẽ làm như sau:

```cpp
    mt19937 rd(seed);
```

Khi code, mọi người có thể chọn seed là một số bất kỳ. Tuy nhiên, để có thể giảm thiểu tỉ lệ thuật toán mt19937 bị crack, mình khuyến khích mọi người nên sử dụng seed random như time() hoặc steady_clock() trong thư viện <chrono>.

Để sinh một số ngẫu nhiên trong có giá trị trong khoảng $[L, R]$ thì ta có 2 cách như sau:

- Cách 1:

```cpp
    int randnum(int l, int r){
        return rd() % (r - l + 1) + l;
    }
```

- Cách 2:

```cpp
    int randnum(int l, int r){
        return uniform_int_distribution<int>(l, r)(rd);
    }
```

3.2 Sinh test

a. Một số lệnh liên quan đến file / cần phải sử dụng

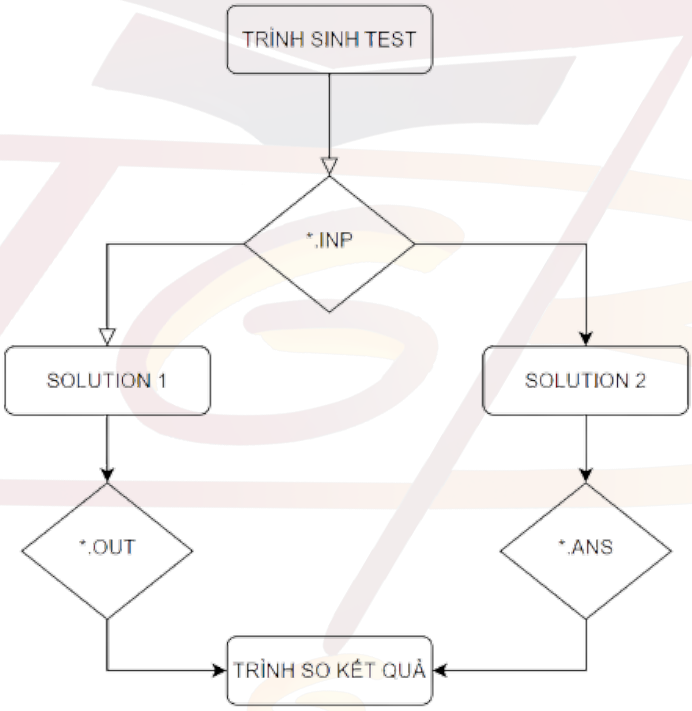
- system(const char* command) : dùng để chạy command trong command processor hoặc terminal trên máy (Command Prompt trên Windows và Terminal ở MacOS)
- ifstream, ofstream
- freopen

b. Lưu ý:

Trong suốt phần dưới của bài viết, mình sẽ sử dụng khá nhiều ký hiệu "*.xxx", với xxx là kiểu file, * là tên của file.
Thông thường, một bộ trình chấm sẽ bao gồm 4 phần chính, đó là:

- File sinh test: Chương trình này có nhiệm vụ sinh input ngẫu nhiên vào file *.inp.
- Solution 1: Là chương trình *.exe cần kiểm tra tính chính xác hoặc cần xác định test sai.
- Solution 2: Chương trình *.exe (mình sẽ gọi đây là *_trau.exe) chứa code trâu hoặc code chắc chắn đưa ra đáp án chính xác được dùng để kiểm tra Solution 1.
- File so kết quả: Sau khi solution 1 và solution 2 đã chạy xong, file này sẽ có nhiệm vụ so sánh 2 file output (của mình ở đây mình sẽ đặt kiểu file là "*.out" và "*.ans", ứng lần lượt với solution 1 và solution 2).

Thông thường, khi sinh test, chúng ta sẽ làm theo quy trình như sau:



Để có thể giúp mọi người dễ mường tượng, chúng ta hãy cùng nhau sinh test cho bài tập sau: MAGICSUM.

Tóm đề : Cho n cặp số $a_i, b_i (n \leq 10^5, 1 \leq a_i \leq b_i \leq 10^6)$, hãy tính $\sum_{i=1}^{n} \sum_{j=a[i]}^{b[i]} j$

1. Solution 1:

- Mình sẽ lấy một submission cũ của bản thân bị WA (đúng 15/20 test, lý do là bị tràn số) Code

2. Solution 2

- Code trâu với độ phức tạp $O(n * \sum_{i=1}^{n} (r_i - l_i))$, với mỗi dòng đi qua mọi số từ l_i tới r_i rồi cộng vào kết quả. Code

3. Trình sinh test và trình so kết quả

- Mô tả cách file hoạt động :
  - **Lưu ý** : Các file được sử dụng phải được bỏ vào chung một folder
  - Đầu tiên, file sẽ sinh ra input vào "MAGICSUM.inp" đúng theo định dạng trong bài, ở đây là : "Dòng thứ nhất có một

số nguyên dương n ($1 \leq n \leq 10^6$); n dòng tiếp theo, mỗi dòng chứa 2 số a_i, b_i ($1 \leq a_i \leq b_i \leq 10^6$)".

```
    ofstream inp("MAGICSUM.inp");
    int n = randnum(1, 1e5);
    inp << n << endl;
    for(int i = 1; i <= n; i++){
        int a = randnum(1, 1e6), b = randnum(a, 1e6);
            inp << a << " " << b << endl;
    }
    inp.close();
```

- Sau đó, chúng ta sẽ gọi 2 chương trình MAGICSUM.exe và MAGICSUMtrau.exe để chạy với input vừa được sinh ra bằng lệnh system với cú pháp system("tênfile.đuôifile").
- Sau khi 2 chương trình đã cho ra đáp án ở trong 2 file MAGICSUM.ans và MAGICSUM.out, ta sẽ so sánh kết quả của 2 file bằng lệnh system("fc tênfile2.đuôifile1 tênfile2.đuôifile2") và in ra kết quả tương ứng.

```
    ofstream inp("MAGICSUM.inp");
    int n = randnum(1, 1e5);
    inp << n << endl;
    for(int i = 1; i <= n; i++){
        int a = randnum(1, 1e6), b = randnum(a, 1e6);
        inp << a << " " << b << endl;
    }
    inp.close();
         system("MAGICSUM.exe"); // solution 1
         system("MAGICSUM_trau.exe"); // solution 2
    if(system("fc MAGICSUM.out MAGICSUM.ans") == 0){
        cout << "AC" << endl;
    }
    else{
        cout << "WA" << endl;
    }
```

- Tuy code trên đã hoàn chỉnh, với mỗi lần chạy thì chỉ chấm được một test. Với một tốc độ như vậy thì thời gian chạy cho đến khi tìm được test sai sẽ vô cùng lâu. Do đó, ta sẽ chạy code trên nhiều lần bằng vòng for, và chỉ dừng khi đã chạy đủ số lần hoặc tìm thấy test sai.

```
signed main(){
    ios_base::sync_with_stdio(0); cin.tie(0);
    for(int test = 1; test <= TESTCASES; test++){
        ofstream inp("MAGICSUM.inp");
        int n = randnum(1, 1e5);
        inp << n << endl;
        for(int i = 1; i <= n; i++){
            int a = randnum(1, 1e6), b = randnum(a, 1e6);
            inp << a << " " << b << endl;
        }
        inp.close();
        system("MAGICSUM.exe"); // solution 1
        system("MAGICSUM_trau.exe"); // solution 2
        if(system("fc MAGICSUM.out MAGICSUM.ans") == 0){
            cout << "AC" << endl;
        }
        else{
            cout << "WA" << endl;
            return 0;
        }
    }
    return 0;
}
```

- Code full

- Chạy thử chương trình, ta tìm được test sai như sau:

```
3
1 1000000
2841 41717
50281 69182
```

- Kết quả của file solution 1 là:

```
1631995512
```

- Kết quả của file solution 2 là:

```
501995685496
```

- Tips: Chúng ta có thể sinh test với giá trị nhỏ trước để tránh những lỗi xuất hiện từ những test nhỏ, những test mà chúng ta có thể dễ dàng debug được.

Tài liệu

[1] Tự code, tự chấm, tự sướng - Bí kíp thi offline - *https://vnoi.info/wiki/algo/skill/viet-trinh-cham.md*

[2] Bassem Marji. What is Mersenne Twister? - *https://www.educative.io/answers/what-is-mersenne-twister*