APPLICATIONS OF TRANSITION MATRICES IN VARIOUS RECURSIVE MODELS

林士紘、張恩睿、李彥霆、林秉毅、葉宥均

Abstract. This report investigates the use of transition matrices to solve specific recursive relations, offering a more efficient alternative to traditional Dynamic Programming (DP) approaches for certain problems. By deriving the transition matrices and applying fast matrix exponentiation, we aim to reduce computational complexity. We compare the efficiency of these two methods through theoretical analysis and experimental results, examining whether actual computation times align with the predicted complexities. Our findings demonstrate the potential of transition matrices in optimizing the computation of high-order recurrences, particularly for large-scale problems.

1. Introduction

In mathematics and programming problems, we often encounter recursive expressions. Take the Fibonacci sequence for example, if we compute it directly, it will cause inefficiency, especially in cases that involve multiple iterations or higher-order systems. As the chart shows below, the computation time will soar up quickly when N is bigger thean 40, which is something we are unwilling to see.

By transforming recursive expressions into matrices form, we can use fast matrix exponentiation to hugely improve the computation efficiency.

2. Materials and Methods

2.1. **Materials.** Pen, paper, computer, C++, brains, curiosity for Linear Algebra.

2.2. **Methods.** From the Rail problem, we derived the recurrence relation and its transition matrix. Then, we implemented recursive functions, DP, and fast matrix exponentiation to compare their computation times across different values of N.

3. Result

Belowing are the graphs of computation time vs. N for each methods:

(1) **Recursive functions** : Execution time grows exponentially.
(2) **Time complexity $O(N^2)$ DP (hereafter referred to as DP-1)** : Execution time grows quadratically.
(3) **Time complexity $O(N)$ DP (hereafter referred to as DP-2)** : Execution time grows linearly.
(4) **Fast matrix exponentiation** : Execution time is roughly constant.

4. Discussion

Comparison of the time taken by each method within a specific range of N:

Notice that due to the significant differences in execution efficiency among the four methods, it is difficult to compare them directly for the same range of N. Therefore, we will discuss them in segments based on different ranges of N and select appropriate methods (two or more) for comparison.

(1) **When $N \leq 40000$:**
As shown in the figure below, the time taken by DP-2 grows quadratically, reaching approximately 7000ms at $N = 40000$. In contrast, the time taken by DP-1 and matrix exponentiation methods remains relatively small.
(2) **When $N \leq 10^7$:**
As shown in the figure below, the time taken by DP-1 grows linearly, reaching approximately 1 second at $N = 4 \times 10^7$. In this range, the matrix exponentiation method remains roughly constant and takes less time than DP-1.
(3) **When $N \leq 10^{400}$:**
In this very large range of numbers, the time taken by matrix exponentiation grows linearly with $\log N$, reaching approximately 5000ms at $N = 10^{400}$.

5. Conclusion

(1) If the n-th term of a recurrence relation is a linear combination of the previous k terms, with constant coefficients (such as the Fibonacci sequence), it can always be represented by a $k \times k$ transition matrix and accelerated using matrix exponentiation.
(2) From the recurrence derived in the Rail problem, we can see that although some recurrences initially do not take the form of a linear combination of previous terms, after simplification, they can still be rewritten as a linear combination of the previous k terms with constant coefficients, and can be handled similarly to (1) as described above.
(3) From the result graph, it can be observed that transforming the recurrence from its function form to matrix exponentiation significantly improves computational efficiency. The size of N that can be calculated within a fixed amount of time is roughly on the order of hundreds of powers of 10!

6. Reference

A. Contribution

113550002 撰寫遞迴 code、繪製內容圖片、製作書面報告、上台發表
113550007 撰寫 dp、矩陣快速冪 code、結論公式推導證明、上台發表
113550018 上影片字幕、上台發表
113550159 code 執行與生成結果、製作結果圖表、撰寫書面報告內容、上台發表
113550179 數據統整、製作影片、製作簡報、撰寫影片講稿、上台發表

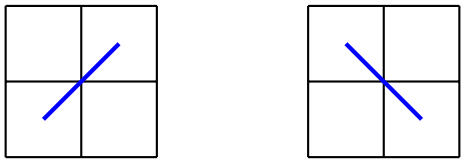
B. Rail

H. 鐵路鋪設 (rail)

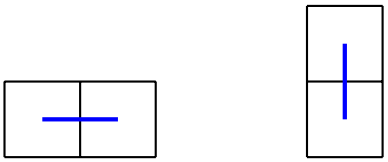
問題描述

古力德市是一座相當特殊的城市。不同於一般的同心圓狀，古力德市是 $2 \times L$ 的棋盤狀，從空中俯瞰就像一條巨大壯觀的蟒蛇，這個景色也吸引了不少觀光客。近年來，為了提升觀光客訪問古力德市的體驗，古力德市政府決定在每一格的正中央設立火車站，並鋪設鐵路路線來連接這 $2L$ 座火車站。

一段鐵路連接相鄰兩個方格的車站，並根據這兩個方格是否為對角線相鄰分為**長鐵路**與**短鐵路**，如下圖所示。



兩種長鐵路

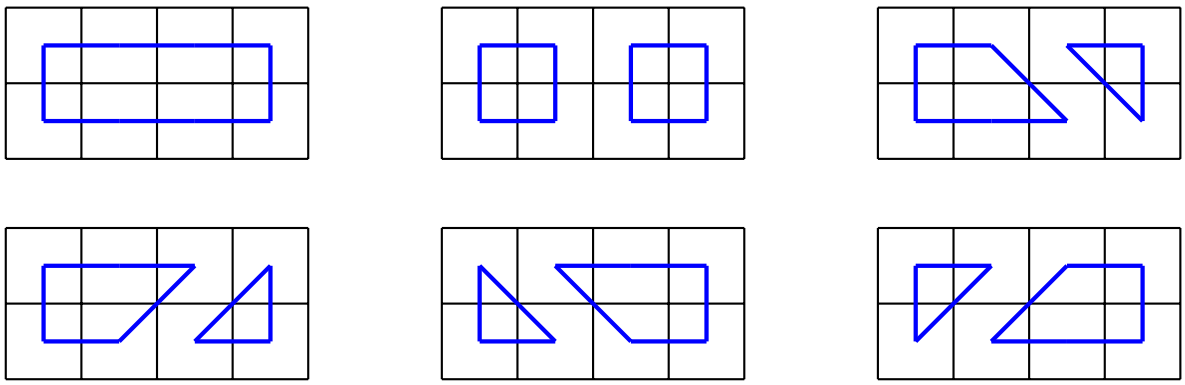


兩種短鐵路

若兩段鐵路共用同一座車站，則稱這兩段鐵路屬於同一條路線，當然**每座車站都要有一條路線經過**。另，鋪設多條路線是被允許的，但因成本問題每一條路線**最多只能有一段長鐵路**。最後，為了避免外地觀光客坐錯車降低訪問體驗，每條路線都必須是**環狀**的（確保搭乘順時針或逆時針方向的車都會抵達目的地），且任兩條路線**不會有任何的重疊或交叉**（意即每座車站皆恰有一條路線經過一次）。

給定古力德市的寬度 L，請求出有多少種可能的鋪設方式。因為這個數字可能很大，你只要求出鋪設方法數除以 $10^9 + 7$ 的餘數就行了。

以下為 $L = 4$ 的範例：在 2×4 的地圖中，共有 6 種鋪設方式。



輸入格式

L

- L 為古力德市的寬度

輸出格式

ans

- ans 為一整數，代表鐵路鋪設方法數除以 $10^9 + 7$ 的餘數

測資限制

- $1 \leq L \leq 10^{10}$
- 輸入的數皆為整數

範例測試

Sample Input	Sample Output
3	3
10	686
327	265488547

評分說明

本題共有四組子任務，條件限制如下所示。每一組可有一或多筆測試資料，該組所有測試資料皆需答對才會獲得該組分數。

子任務	分數	額外輸入限制
1	10	$L \leq 7$
2	20	$L \leq 10^3$
3	34	$L \leq 10^5$
4	36	無額外限制

C. Induction for the Transition Matrix of Rail

C.1. Induction to recursive formula.

D. Code for Fibonacci

D.1. Using Recursion.

```
#include <iostream>
#include <sys/time.h>
#include <chrono>
#include <ctime>
#include <time.h>
using namespace std;
using std::chrono::duration_cast;
using std::chrono::milliseconds;
using std::chrono::seconds;
using std::chrono::system_clock;

const int MOD = 1e9 + 7;

int64_t cal(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return (cal(n - 1) + cal(n - 2)) % MOD;
}

int main() {
    int n;
    FILE *f1, *f2;
    f1 = fopen("num.txt","w");
    f2 = fopen("time.txt","w");
    for(n = 0;n <= 45; n++){
        int64_t ans = 0;
        fprintf(f1,"%d\n",n);
        int64_t t = duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count
    ();
        ans = cal(n);
        fprintf(f2,"%ld\n",duration_cast<milliseconds>(system_clock::now().time_since_epoch())
    .count()-t);
        cout << "cost time: " << duration_cast<milliseconds>(system_clock::now().
    time_since_epoch()).count() - t << '\n';
        cout << "ans: " << ans << '\n';
    }
    fclose(f1);
    fclose(f2);

}
```

D.2. Using DP.

```
#include <bits/stdc++.h>
#include <sys/time.h>
#include <chrono>
#include <ctime>
#include <iostream>
using std::cout;
using std::endl;
using std::chrono::duration_cast;
using std::chrono::milliseconds;
using std::chrono::seconds;
using std::chrono::system_clock;
using namespace std;

#define prime 1000000007

```

```cpp
int main(){
    int64_t n;
    FILE *f1, *f2;
    f1 = fopen("num.txt","w");
    f2 = fopen("time.txt","w");
    for(n=0;n<=40000;n += 400){
        int64_t a = 1, b = 1, c;
        fprintf(f1,"%ld\n",n);
        int64_t t = duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count();
        for(int64_t i = 3; i <= n; i ++){
            c = (a+b)%prime;
            a = b;
            b = c;
        }
        fprintf(f2,"%ld\n",duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t);
        cout << "Time Cost: " << duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t << "ms\n";
        cout << "Value: " << c << '\n';
    }
    fclose(f1);
    fclose(f2);
}
```

D.3. **Using Fast Matrix Power.**

```cpp
#include <bits/stdc++.h>
#include <sys/time.h>
#include <chrono>
#include <ctime>
#include <iostream>
using std::cout;
using std::endl;
using std::chrono::duration_cast;
using std::chrono::milliseconds;
using std::chrono::seconds;
using std::chrono::system_clock;
using namespace std;

#define prime 1000000007

struct Num{
    int size = 1005;
    int arr[1005];

    Num(){
        for(int i = 0; i < size; i ++) arr[i] = 0;
    }

    Num(string str){
        for(int i = 0; i < size; i ++) arr[i] = 0;
        for(int i = 0; i < str.length(); i ++){
            arr[i] = str[str.length()-i-1] - '0';
        }
    }

    void tten(){
        for(int i = size-1; i >= 0; i --){
            arr[i] = arr[i-1];
        }
        arr[0] = 0;
    }

```

```
38      Num dtwo(){
39          Num ans;
40          int tmp = 0;
41          for(int i = size-1; i >= 0; i --){
42              tmp = tmp*10 + arr[i];
43              ans.tten();
44              ans.arr[0] = tmp/2;
45              tmp = tmp % 2;
46          }
47          return ans;
48      }
49
50      int ptwo(){
51          return arr[0]%2;
52      }
53
54      bool is_one(){
55          for(int i = size-1; i > 0; i --){
56              if(arr[i] != 0) return false;
57          }
58          if(arr[0] != 1) return false;
59          return true;
60      }
61      bool is_zero(){
62          for(int i = size-1; i >= 0; i --){
63              if(arr[i] != 0) return false;
64          }
65          return true;
66      }
67
68      void print(){
69          bool start = false;
70          for(int i = size-1; i >= 0; i --){
71              if(start){
72                  cout << arr[i];
73              }
74              else{
75                  if(arr[i] != 0){
76                      cout << arr[i];
77                      start = true;
78                  }
79              }
80          }
81      }
82  };
83
84  struct M{
85      int64_t arr[2][2];
86
87      M(){
88          arr[0][0] = 1;
89          arr[0][1] = 1;
90          arr[1][0] = 1;
91          arr[1][1] = 0;
92      }
93
94
95      M operator * (const M &b) const {
96          M tmp;
97          for(int64_t i = 0; i < 2; i ++){
98              for(int64_t j = 0; j < 2; j ++){
99
100                 int64_t s = 0;
```

```
                for(int64_t k = 0; k < 2; k ++){
                    s += arr[i][k] * b.arr[k][j];
                    s %= prime;
                }
                tmp.arr[i][j] = s;
            }
        }
        return tmp;
    }
};

M solve_2(Num &n){
    M tmp;

    vector<bool> v;

    while(!n.is_one()){
        v.emplace_back(n.ptwo());
        n = n.dtwo();
    }

    M tmp_2;
    for(int i = v.size()-1; i >= 0; i --){
        tmp_2 = tmp_2*tmp_2;
        if(v[i]) tmp_2 = tmp_2*tmp;

    }

    return tmp_2;
}

int64_t solve(Num &n){
    if(n.is_one()) return 1;
    if(n.is_zero()) return 1;

    M tmp = solve_2(n);
    return tmp.arr[0][0];
}

int main(){
    FILE *f1, *f2;
    f1 = fopen("num.txt","w");
    f2 = fopen("time.txt","w");
    int cnt = 4;
    for(int i=0;i<100;i++){
        string str = "";
        str += to_string(cnt);
        str += "00";
        cnt += 4;
        fprintf(f1,"%s\n",str.c_str());
        Num n(str);
        int64_t t = duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count
    ();
        auto ans = solve(n);
        fprintf(f2,"%ld\n",duration_cast<milliseconds>(system_clock::now().time_since_epoch())
    .count()-t);
        cout << "Time Cost(Mat): " << duration_cast<milliseconds>(system_clock::now().
    time_since_epoch()).count()-t << "ms\n";
        cout << "Value: " << ans << "\n";
    }
    fclose(f1);
    fclose(f2);
}
```

E. Code for Rail

E.1. Using Recursion.

```cpp
#include <iostream>
#include <sys/time.h>
#include <chrono>
#include <ctime>
#include <time.h>
using namespace std;
using std::chrono::duration_cast;
using std::chrono::milliseconds;
using std::chrono::seconds;
using std::chrono::system_clock;

const int MOD = 1e9 + 7;

int64_t cal(int n) {
    if (n == 0) return 1;
    if (n == 1) return 0;
    if (n == 2) return 1;
    return (2 * cal(n - 1) % MOD + cal(n - 3)) % MOD;
}

int main() {
    int n;
    FILE *f1, *f2;
    f1 = fopen("num.txt","w");
    f2 = fopen("time.txt","w");
    for(n = 0;n <= 55; n++){
        int64_t ans = 0;
        fprintf(f1,"%d\n",n);
        int64_t t = duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count
    ();
        ans = cal(n);
        fprintf(f2,"%ld\n",duration_cast<milliseconds>(system_clock::now().time_since_epoch())
    .count()-t);
        cout << "cost time: " << duration_cast<milliseconds>(system_clock::now().
    time_since_epoch()).count() - t << '\n';
        cout << "ans: " << ans << '\n';
    }
    fclose(f1);
    fclose(f2);
}
```

E.2. Using DP.

```cpp
#include <bits/stdc++.h>
#include <sys/time.h>
#include <chrono>
#include <ctime>
#include <iostream>
using std::cout;
using std::endl;
using std::chrono::duration_cast;
using std::chrono::milliseconds;
using std::chrono::seconds;
using std::chrono::system_clock;
using namespace std;

#define prime 1000000007

struct M{
    int64_t arr[3][3];

    M(){
```

```
        arr[0][0] = 2;
        arr[0][1] = 0;
        arr[0][2] = 1;
        arr[1][0] = 1;
        arr[1][1] = 0;
        arr[1][2] = 0;
        arr[2][0] = 0;
        arr[2][1] = 1;
        arr[2][2] = 0;
    }


    M operator * (const M &b) const {
        M tmp;
        for(int64_t i = 0; i < 3; i ++){
            for(int64_t j = 0; j < 3; j ++){

                int64_t s = 0;
                for(int64_t k = 0; k < 3; k ++){
                    s += arr[i][k] * b.arr[k][j];
                    s %= prime;
                }
                tmp.arr[i][j] = s;
            }
        }
        return tmp;
    }
};

M solve_2(int64_t n){
    M tmp;

    if(n == 1) return tmp;


    M tmp_2 = solve_2(n/2);
    tmp_2 = tmp_2 * tmp_2;

    if(n%2){
        tmp_2 = tmp_2 * tmp;
    }

    return tmp_2;
}

int64_t solve(int64_t n){
    if(n == 0) return 1;
    if(n == 1) return 0;
    if(n == 2) return 1;

    M tmp = solve_2(n-2);
    return (tmp.arr[0][0]+tmp.arr[0][2])%prime;
}

int main(){
    int64_t n;
    FILE *f1, *f2;
    f1 = fopen("num.txt","w");
    f2 = fopen("time.txt","w");
    for(n=0;n<=40000;n += 400){
        //cout << '\n';
        int64_t dp[n+1];
        dp[0] = 1;
```

```
        dp[1] = 0;
        int64_t t = duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count();
        for(int i = 2; i <= n; i ++){
            dp[i] = 0;
            for(int j = 0; j <= i-2; j ++){
                dp[i] += (2*(i-j)-3)*dp[j];
                dp[i] %= prime;
            }
        }
        fprintf(f1,"%ld\n",n);
        fprintf(f2,"%ld\n",duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t);
        cout << "Time Cost(O(N^2)): " << duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t << "ms\n";
        cout << "Value: " << dp[n] << "\n\n";
    }

    for(n = 0; n <= 40000000; n += 400000){
        int64_t dp2[3], a[3], b[3];
        dp2[0] = 1;
        dp2[1] = 0;
        a[0] = 1;
        a[1] = 1;
        b[0] = 0;
        b[1] = 0;
        int64_t t2 = duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count();
        for(int64_t i = 2; i <= n; i ++){
            dp2[2] = ((( (2*i-3)%prime * a[0] - 2*b[0] % prime ) % prime)+prime)%prime;
            a[2] = ((a[1] + dp2[2]) % prime + prime) % prime;
            b[2] = ((b[1] + i*dp2[2] % prime) % prime + prime) % prime;
            dp2[0] = dp2[1];
            dp2[1] = dp2[2];
            a[0] = a[1];
            a[1] = a[2];
            b[0] = b[1];
            b[1] = b[2];
            //if(i % 100000000 == 0) cout << i << " " << dp2[2] << '\n';
        }
        fprintf(f1,"%ld\n",n);
        fprintf(f2,"%ld\n",duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t2);
        cout << "Time Cost(O(N)): " << duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t2 << "ms\n";
        cout << "Value: " << (dp2[2]+prime)%prime << "\n\n";
    }
    //fclose(f1);
    //fclose(f2);
    for(n = 0; n <= 40000000; n+= 400000){
        int64_t t3 = duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count();
        int64_t ans = solve(n);
        fprintf(f1,"%ld\n",n);
        fprintf(f2,"%ld\n",duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t3);
        cout << "Time Cost(Mat): " << duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t3 << '\n';
        cout << "Value: " << ans << '\n';
    }

    fclose(f1);
    fclose(f2);
```

```
}
```

E.3. **Using Fast Matrix Power.**

```
#include <bits/stdc++.h>
#include <sys/time.h>
#include <chrono>
#include <ctime>
#include <iostream>
using std::cout;
using std::endl;
using std::chrono::duration_cast;
using std::chrono::milliseconds;
using std::chrono::seconds;
using std::chrono::system_clock;
using namespace std;

#define prime 1000000007

struct Num{
    int size = 1005;
    int arr[1005];

    Num(){
        for(int i = 0; i < size; i ++) arr[i] = 0;
    }

    Num(string str){
        for(int i = 0; i < size; i ++) arr[i] = 0;
        for(int i = 0; i < str.length(); i ++){
            arr[i] = str[str.length()-i-1] - '0';
        }
    }

    void tten(){
        for(int i = size-1; i >= 0; i --){
            arr[i] = arr[i-1];
        }
        arr[0] = 0;
    }

    Num dtwo(){
        Num ans;
        int tmp = 0;
        for(int i = size-1; i >= 0; i --){
            tmp = tmp*10 + arr[i];
            ans.tten();
            ans.arr[0] = tmp/2;
            tmp = tmp % 2;
        }
        return ans;
    }

    int ptwo(){
        return arr[0]%2;
    }

    void mtwo(){
        arr[0] -= 2;
        int n = 0;
        while(arr[n] < 0){
            arr[n] += 10;
            arr[n+1] --;
            n ++;
```

```
        }
    }

    bool is_two(){
        for(int i = size-1; i > 0; i --){
            if(arr[i] != 0) return false;
        }
        if(arr[0] != 2) return false;
        return true;
    }
    bool is_one(){
        for(int i = size-1; i > 0; i --){
            if(arr[i] != 0) return false;
        }
        if(arr[0] != 1) return false;
        return true;
    }
    bool is_zero(){
        for(int i = size-1; i >= 0; i --){
            if(arr[i] != 0) return false;
        }
        return true;
    }

    void print(){
        bool start = false;
        for(int i = size-1; i >= 0; i --){
            if(start){
                cout << arr[i];
            }
            else{
                if(arr[i] != 0){
                    cout << arr[i];
                    start = true;
                }
            }
        }
    }
};

struct M{
    int64_t arr[3][3];

    M(){
        arr[0][0] = 2;
        arr[0][1] = 0;
        arr[0][2] = 1;
        arr[1][0] = 1;
        arr[1][1] = 0;
        arr[1][2] = 0;
        arr[2][0] = 0;
        arr[2][1] = 1;
        arr[2][2] = 0;
    }


    M operator * (const M &b) const {
        M tmp;
        for(int64_t i = 0; i < 3; i ++){
            for(int64_t j = 0; j < 3; j ++){

                int64_t s = 0;
                for(int64_t k = 0; k < 3; k ++){
```

```
                    s += arr[i][k] * b.arr[k][j];
                    s %= prime;
                }
                tmp.arr[i][j] = s;
            }
        }
        return tmp;
    }
};

M solve_2(Num &n){
    M tmp;

    vector<bool> v;

    while(!n.is_one()){
        v.emplace_back(n.ptwo());
        n = n.dtwo();
    }

    M tmp_2;
    for(int i = v.size()-1; i >= 0; i --){
        tmp_2 = tmp_2*tmp_2;
        if(v[i]) tmp_2 = tmp_2*tmp;

    }

    return tmp_2;
}

int64_t solve(Num n){
    if(n.is_zero()) return 1;
    if(n.is_one()) return 0;
    if(n.is_two()) return 1;
    n.mtwo();
    M tmp = solve_2(n);
    return (tmp.arr[0][0]+tmp.arr[0][2])%prime;
}

int main(){
    FILE *f1,*f2;
    f1 = fopen("num.txt","w");
    f2 = fopen("time.txt","w");
    int tmp = 4;
    for(int i=0;i<100;i++){
        string str = "";
        str += to_string(tmp);
        str += "00000";
        tmp += 4;
        cout << str << endl;
        Num n(str);
        int64_t t3 = duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count
    ();
        int64_t ans = solve(n);
        cout << "Time Cost(Mat): " << duration_cast<milliseconds>(system_clock::now().
    time_since_epoch()).count()-t3 << "ms\n";
        //cout << "Value: " << ans << '\n';
        fprintf(f1,"%s\n",str.c_str());
        fprintf(f2,"%ld\n",duration_cast<milliseconds>(system_clock::now().time_since_epoch())
    .count()-t3);
    }
    fclose(f1);
    fclose(f2);
```

```
184 }
```