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

Recurrence relations are central in mathematics and computer science, appearing frequently in problems such as the Fibonacci sequence or divide-and-conquer strategies. While direct computation of recurrence relations can be inefficient, especially for large inputs, transforming them into matrices offers a more efficient alternative.

In this report, we explore how matrix exponentiation can simplify and speed up the computation of recurrence relations. We implement this approach using C++ and compare its performance with a dynamic programming solution, highlighting its advantages for solving high-order or iterative recurrence problems.

2. Materials and Methods

2.1. **Materials.** Tools and resources utilized include: pen, papers, brains, computers, and the C++ programming language.

2.2. **Methods.** Initially, the transition matrix for the recursive formula is derived, and its theoretical time complexity is calculated. Subsequently, C++ code is implemented to evaluate its performance through execution time measurement. The consistency between the measured execution time and the theoretical time complexity is verified. Finally, the results are compared with those obtained from a dynamic programming (DP) implementation.

3. Results

4. Discussion

4.1. **Fibonacci.** $F(n) = F(n-1) + F(n-2)$

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F(n-1) \\ F(n-2) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2} \begin{bmatrix} F(2) \\ F(1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

4.2. **Rail.** $D_i = \sum_{j=0}^{i-2} (2(i-j)-3) D_j$

A detailed description of the Rail recurrence relation is provided in Appendix A on page i.

$$\begin{bmatrix} D_i \\ D_{i-1} \\ D_{i-2} \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} D_{i-1} \\ D_{i-2} \\ D_{i-3} \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^{i-2} \begin{bmatrix} D_2 \\ D_1 \\ D_0 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^{i-2} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

The complete induction process is detailed in Appendix B on page iv.

5. Conclusion

1. For recursive relations with fixed coefficients, representing them in matrix form enables efficient computation through repeated multiplication of the same matrix. This approach facilitates the application of fast exponentiation, thereby reducing the number of operations and significantly shortening computation time.

2. A comparison between fast exponentiation and the DP approach demonstrates that the time difference becomes increasingly significant as n grows. This observation aligns closely with the theoretical time complexity analysis.

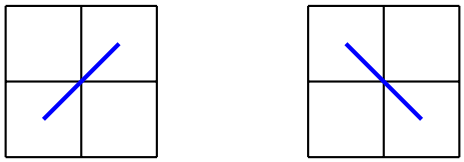
6. Reference

H. 鐵路鋪設 (rail)

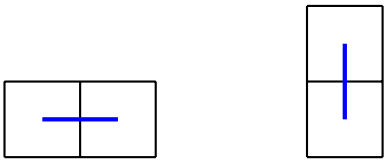
問題描述

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

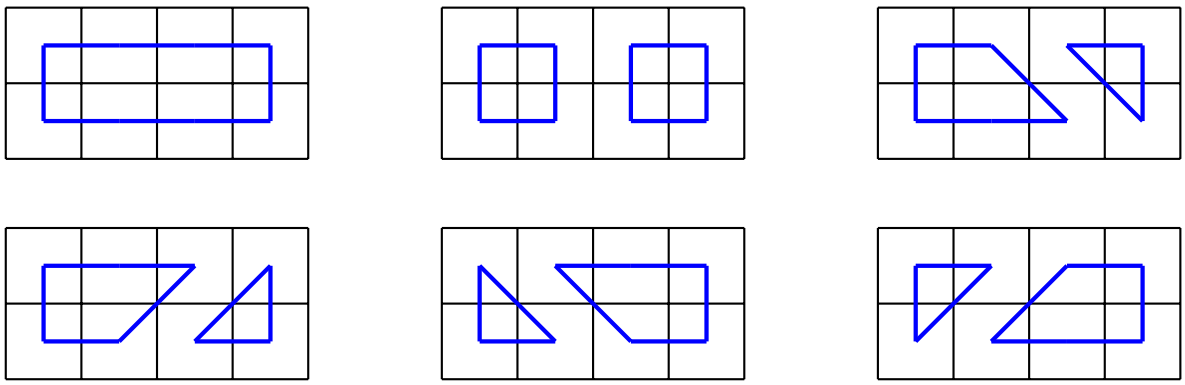


兩種短鐵路

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

B. Induction for the Transition Matrix of Rail

We start with the recursive formula:

$$D_i = \sum_{j=0}^{i-2} (2(i-j)-3)D_j \tag{1}$$

Expanding the summation:

$$\begin{aligned} D_i &= \sum_{j=0}^{i-2} (2i-2j-3)D_j \\ &= \sum_{j=0}^{i-2} (2i-3)D_j - \sum_{j=0}^{i-2} 2jD_j \\ &= (2i-3)\sum_{j=0}^{i-2} D_j - 2\sum_{j=0}^{i-2} jD_j \end{aligned}$$

Using induction, we express D_i and D_{i-1}:

$$D_i = (2i-3)\sum_{j=0}^{i-2} D_j - 2\sum_{j=0}^{i-2} jD_j \tag{2}$$

$$D_{i-1} = (2i-5)\sum_{j=0}^{i-3} D_j - 2\sum_{j=0}^{i-3} jD_j \tag{3}$$

To simplify further, subtract Equation (3) from Equation (2):

$$\begin{aligned} D_i - D_{i-1} &= (2i-3)D_{i-2} + 2\sum_{j=0}^{i-3} D_j - 2(i-2)D_{i-2} \\ &= D_{i-2} + 2\sum_{j=0}^{i-3} D_j \end{aligned}$$

Using induction on the differences, we arrive at:

$$\begin{aligned} D_i - D_{i-1} - D_{i-2} &= 2\sum_{j=0}^{i-3} D_j \\ D_{i-1} - D_{i-2} - D_{i-3} &= 2\sum_{j=0}^{i-4} D_j \\ D_i - 2D_{i-1} + D_{i-3} &= 2D_{i-3} \end{aligned}$$

Thus, we obtain the recurrence relation:

$$D_i = 2D_{i-1} + D_{i-3} \tag{4}$$

B.1. **Transition Matrix Representation.** We represent this recurrence relation using a transition matrix:

$$\begin{bmatrix} D_i \\ D_{i-1} \\ D_{i-2} \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} D_{i-1} \\ D_{i-2} \\ D_{i-3} \end{bmatrix}$$

The general term can be expressed as:

$$\begin{bmatrix} D_i \\ D_{i-1} \\ D_{i-2} \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^{i-2} \begin{bmatrix} D_2 \\ D_1 \\ D_0 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^{i-2} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

C. Code for Fibonacci

C.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 0;
    if (n == 1) return 1;
    return (cal(n - 1) + cal(n - 2)) % MOD;
}

int main() {
    int n;
    cin >> n;
    int64_t ans = 0;
    int64_t t = duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count();
    ans = cal(n);
    cout << "cost time: " << duration_cast<milliseconds>(system_clock::now().time_since_epoch
    ()).count() - t << '\n';
    cout << "ans: " << ans << '\n';
    return 0;
}
```

C.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

int main(){
    int64_t n;
    cin >> n;
    int64_t a = 1, b = 1, c;
    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;
    }
    cout << duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t << '
    \n';
    cout << c << '\n';

```

```
}
```

C.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;
    }

    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;
    }
```

```
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;
101                 for(int64_t k = 0; k < 2; k ++){
102                     s += arr[i][k] * b.arr[k][j];
103                     s %= prime;
104                 }
105                 tmp.arr[i][j] = s;
106             }
107         }
108         return tmp;
109     }
110 };
111
112 M solve_2(Num &n){
113     M tmp;
114
115     vector<bool> v;
116
117     while(!n.is_one()){
118         v.emplace_back(n.ptwo());
119         n = n.dtwo();
120     }
121
122     M tmp_2;
123     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(){
    string str;
    cin >> str;
    Num n(str);
    int64_t t = duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count();
    auto ans = solve(n);
    cout << duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t << '\n';
    cout << ans << "\n";
}
```

D. Code for Rail

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 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;
    cin >> n;
    int64_t ans = 0;
    int64_t t = duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count();
    ans = cal(n);
    cout << "cost time: " << duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count() - t << '\n';
    cout << "ans: " << ans << '\n';
    return 0;
}
```

D.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

int main(){
    int64_t n;
    cin >> 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;
        }
    }
    cout << duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t << '\n';
    cout << dp[n] << "\n\n";

    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];
    }
    cout << duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t2 << '\n';
    cout << (dp2[2]+prime)%prime << "\n\n";

}
```

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

```cpp
#include <bits/stdc++.h>
#include <sys/time.h>
#include <chrono>
#include <ctime>
```

```cpp
#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;
        }
```

```
68          if(arr[0] != 2) return false;
69          return true;
70      }
71      bool is_one(){
72          for(int i = size-1; i > 0; i --){
73              if(arr[i] != 0) return false;
74          }
75          if(arr[0] != 1) return false;
76          return true;
77      }
78      bool is_zero(){
79          for(int i = size-1; i >= 0; i --){
80              if(arr[i] != 0) return false;
81          }
82          return true;
83      }
84
85      void print(){
86          bool start = false;
87          for(int i = size-1; i >= 0; i --){
88              if(start){
89                  cout << arr[i];
90              }
91              else{
92                  if(arr[i] != 0){
93                      cout << arr[i];
94                      start = true;
95                  }
96              }
97          }
98      }
99  };
100
101 struct M{
102     int64_t arr[3][3];
103
104     M(){
105         arr[0][0] = 2;
106         arr[0][1] = 0;
107         arr[0][2] = 1;
108         arr[1][0] = 1;
109         arr[1][1] = 0;
110         arr[1][2] = 0;
111         arr[2][0] = 0;
112         arr[2][1] = 1;
113         arr[2][2] = 0;
114     }
115
116
117     M operator * (const M &b) const {
118         M tmp;
119         for(int64_t i = 0; i < 3; i ++){
120             for(int64_t j = 0; j < 3; j ++){
121
122                 int64_t s = 0;
123                 for(int64_t k = 0; k < 3; k ++){
124                     s += arr[i][k] * b.arr[k][j];
125                     s %= prime;
126                 }
127                 tmp.arr[i][j] = s;
128             }
129         }
130         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(){
    string str;
    cin >> str;
    Num n(str);
    int64_t t3 = duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count();
    int64_t ans = solve(n);
    cout << duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count()-t3 <<
    '\n';
    cout << ans << '\n';

}
```