

Vietnam National University HCMC
University Of Information Technology



BÁO CÁO ĐỒ ÁN CUỐI KỲ

CS112.L13.KHCL

KNIGHT DIALER

Thành viên nhóm

19520197 Lê Đoàn Thiện Nhân
19521281 Trương Minh Châu
19520658 Phạm Minh Khôi

Giảng viên phụ trách

Phạm Nguyễn Trường An

Ngày 28 tháng 1 năm 2021

Mục lục

1	Giới thiệu bài toán	1
1.1	Lịch sử bài toán	1
1.2	Đề bài toán	1
2	Phương pháp Dynamic Programming	2
2.1	Phương pháp thiết kế thuật toán	2
2.2	Lịch sử phát triển phương pháp	2
2.3	Pseudo Code	3
2.4	Phân tích độ phức tạp bằng các phương pháp toán học	3
2.5	Source Code	5
2.6	Phát sinh input và output	6
2.7	Phân tích độ phức tạp bằng thực nghiệm	7
3	Phương pháp Dynamic Programming kết hợp toán học	9
3.1	Đặt vấn đề bài toán	9
3.2	Phương pháp thiết kế thuật toán	9
3.3	Pseudo Code	10
3.4	Phân tích độ phức tạp bằng các phương pháp toán học	10
3.5	Source Code	11
3.6	Phát sinh input và output	13
3.7	Phân tích độ phức tạp bằng thực nghiệm	13
4	Đánh giá và so sánh	15
4.1	Phương pháp Dynamic Programming	15
4.2	Phương pháp Dynamic Programming kết hợp toán học	17
5	Tổng Kết	20

1 Giới thiệu bài toán

1.1 Lịch sử bài toán

Knight Dialer là một bài toán được Google đưa ra trong buổi phỏng vấn nhằm đánh giá khả năng suy nghĩ logic, mức độ thành thạo các thuật toán, khả năng triển khai ý tưởng trong khoảng thời gian hạn hẹp (45 phút) và đánh giá độ phức tạp thuật toán do chính mình đưa ra để giải bài toán đó của người ứng cử viên tham gia phỏng vấn. Vào khoảng thời gian ấy bài toán này vẫn chưa được Google công bố nên đa phần tất cả những ứng cử viên đi phỏng vấn đều khá "khó chịu" với bài toán này cho đến khi nó bị phát tán ra ngoài và Google đã không còn dùng bài toán này để phỏng vấn nữa.

1.2 Đề bài toán

Bài toán Knight Dialer có nội dung như sau: "Hãy tưởng tượng rằng bạn đặt một quân mã trên một phím số của điện thoại di động. Quân cờ này chỉ có thể di chuyển theo hình chữ "L": hai bước theo chiều dọc và một bước theo chiều ngang, hoặc một bước theo chiều dọc và hai bước theo chiều ngang.



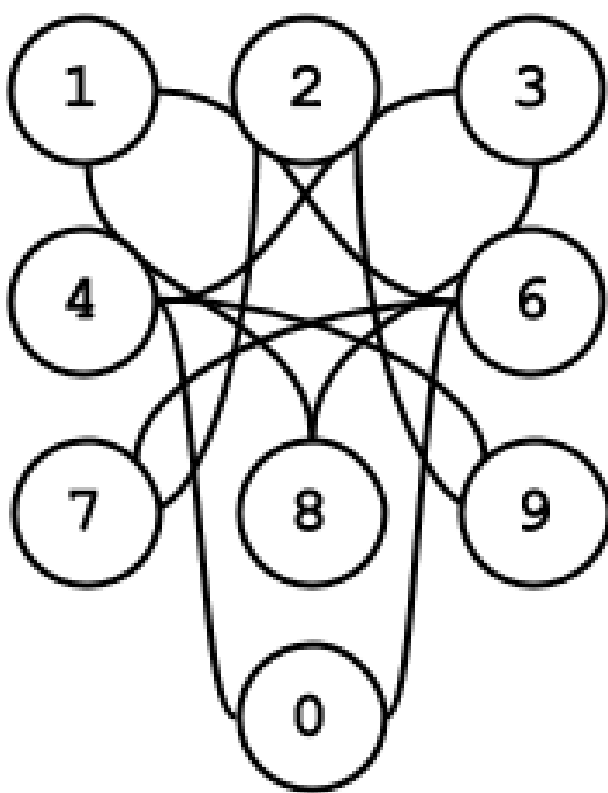
Hãy tưởng rằng bạn quay số điện thoại bằng cách sử dụng những bước nhảy của quân mã. Mỗi khi quân cờ đi qua một phím thì ta sẽ thêm phím đó vào dãy số điện thoại và tiếp tục di chuyển. Vị trí đứng ban đầu của quân mã cũng được tính là một số trong chuỗi số. Chúng ta có thể quay bao nhiêu chuỗi số khác nhau khi thực hiện N bước nhảy từ một vị trí ban đầu cụ thể ?"¹

¹trích từ alexgolec.dev/knights-dialer-logarithmic-time-edition/

2 Phương pháp Dynamic Programming

2.1 Phương pháp thiết kế thuật toán

Với bài toán trên, trong khoảng thời gian nhất định, những người tham gia phỏng vấn sẽ chọn việc sử dụng thuật toán quy hoạch động (Dynamic programming) và kỹ thuật sử dụng đồ thị để giải bài toán này vì với khoảng thời gian như vậy họ sẽ khó để nhận ra được tính quy luật của bài toán cũng như tìm ra được cách giải nào khác tối ưu hơn.



2.2 Lịch sử phát triển phương pháp

Kỹ thuật quy hoạch động (Dynamic programming) vừa là một kỹ thuật tối ưu hóa toán học, vừa là một kỹ thuật lập trình. Quy hoạch động được phát triển bởi Richard Bellman vào những năm 1950 và đến tận ngày nay kỹ thuật này vẫn được sử dụng rộng rãi trong các lĩnh vực khác nhau, nhất là trong lập trình. Kỹ thuật này thường được sử dụng để tối ưu hóa những bài toán đệ quy đơn giản bằng cách "ghi nhớ". Các bài toán đệ quy thường phải thử qua tất cả các trường hợp điều đó dẫn đến việc lặp đi lặp lại một bước nào đó rất nhiều lần sẽ dẫn đến làm chậm tốc độ của chương trình, quy hoạch động sẽ giải quyết điều đó bằng cách phân tích bài toán thành nhiều phần nhỏ hơn và giải quyết từng phần

nhỏ đó để tính kết quả của phần lớn hơn rồi từ đó suy ra được kết quả của toàn bài. Quy hoạch động sẽ lưu lại kết quả của những phần đã được tính trong một cấu trúc dữ liệu nào đó rồi truy xuất kết quả đã tính đó ra dùng để tính cho phần lớn hơn, chính điều này giúp cho kỹ thuật quy hoạch động sẽ tối ưu hóa được thời gian thực thi của nhiều bài toán tưởng chừng sẽ không tìm được kết quả vì thời gian thực thi quá lâu.

2.3 Pseudo Code

Input:

- . N : số bước con mã phải đi
- . $graph$: một dictionary chứa các node từ 0 đến 9 và ứng với mỗi node là các mảng con chứa các node liền kề với node đó

Function DP-Graph (N , $graph$)

if $N = 1$ then
return 10

Khai báo cnt là một dictionary chứa các node từ 0 tới 9 và ứng với mỗi node là số các chuỗi số khác nhau được hình thành khi chữ số cuối cùng của chuỗi số tại bước thứ i kết thúc bằng chữ số tại node tương ứng, mặc định chỉ số đếm bằng 1

for $i: 0 \rightarrow N - 2$ do

Khai báo tmp là một dictionary có chức năng giống với cnt nhưng với chỉ số đếm bắt đầu bằng 0

foreach $k \in cnt$ do

foreach $v \in graph[k]$ do

$tmp[v] = tmp[v] + cnt[k]$

$cnt = tmp$

return sum (giá trị của mỗi node trong cnt)

2.4 Phân tích độ phức tạp bằng các phương pháp toán học

$$\begin{aligned}
 F(1, N) &= \begin{cases} 1 & \text{if } n = 1 \\ F(6, N - 1) + F(8, N - 1) & \text{if } n > 1 \end{cases} \\
 F(2, N) &= \begin{cases} 1 & \text{if } n = 1 \\ F(7, N - 1) + F(9, N - 1) & \text{if } n > 1 \end{cases} \\
 F(3, N) &= \begin{cases} 1 & \text{if } n = 1 \\ F(4, N - 1) + F(8, N - 1) & \text{if } n > 1 \end{cases} \\
 F(4, N) &= \begin{cases} 1 & \text{if } n = 1 \\ F(0, N - 1) + F(3, N - 1) + F(9, N - 1) & \text{if } n > 1 \end{cases} \\
 F(5, N) &= \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } n > 1 \end{cases} \\
 F(6, N) &= \begin{cases} 1 & \text{if } n = 1 \\ F(0, N - 1) + F(1, N - 1) + F(7, N - 1) & \text{if } n > 1 \end{cases} \\
 F(7, N) &= \begin{cases} 1 & \text{if } n = 1 \\ F(2, N - 1) + F(9, N - 1) & \text{if } n > 1 \end{cases} \\
 F(8, N) &= \begin{cases} 1 & \text{if } n = 1 \\ F(1, N - 1) + F(3, N - 1) & \text{if } n > 1 \end{cases} \\
 F(9, N) &= \begin{cases} 1 & \text{if } n = 1 \\ F(2, N - 1) + F(4, N - 1) & \text{if } n > 1 \end{cases}
 \end{aligned}$$

2 Phương pháp Dynamic Programming

$$F(0, N) = \begin{cases} 1 & \text{if } n = 1 \\ F(4, N-1) + F(6, N-1) & \text{if } n > 1 \end{cases}$$

Ta có:

$$F(6, 2) = F(4, 2) = 6$$

$$F(6, 3) = F(4, 3) = 16$$

Giả sử công thức đúng tới $N = k$:

$$\begin{aligned} F(6, k) &= F(4, k) \\ \Leftrightarrow F(1, k-1) + F(7, k-1) + F(0, k-1) &= F(3, k-1) + F(9, k-1) + F(0, k-1) \\ \Leftrightarrow F(1, k-1) + F(6, k-2) + F(2, k-2) &= F(3, k-1) + F(4, k-2) + F(2, k-2) \\ \Leftrightarrow F(1, k-1) &= F(3, k-1) \end{aligned}$$

Chứng minh tương tự ta được:

$$F(1, k) = F(3, k)$$

$$F(7, k) = F(9, k)$$

Vậy ta có:

$$\begin{aligned} F(6, k+1) &= F(1, k) + F(7, k) + F(0, k) \\ &= F(3, k) + F(9, k) + F(0, k) \\ &= F(4, k+1) \end{aligned}$$

Nên:

$$F(K, N) = \sum_{k \in \text{neighbors}(K)} F(k, N-1) \quad (1)$$

$\stackrel{(1)}{\Rightarrow} F(5, N) \leq F(K, N) \leq F(4, N)$ (vì 5 có tập neighbors = \emptyset còn 4 có tập neighbors có số phần tử nhiều nhất là 3)

Mà ta có:

$$F(4, 2) = F(0, 1) + F(3, 1) + F(9, 1)$$

$$\begin{aligned} F(4, 3) &= F(0, 2) + F(3, 2) + F(9, 2) \\ &= F(4, 1) + F(6, 1) + F(4, 1) + F(8, 1) + F(4, 1) + F(2, 1) \\ &= 4F(4, 1) + F(8, 1) + F(2, 1) = 4F(4, 1) + 2 \end{aligned}$$

$$\begin{aligned} F(4, 4) &= F(0, 3) + F(3, 3) + F(9, 3) \\ &= F(4, 2) + F(6, 2) + F(4, 2) + F(8, 2) + F(4, 2) + F(2, 2) \\ &= 4F(4, 2) + F(8, 2) + F(2, 2) \\ &= 4F(4, 2) + F(1, 1) + F(3, 1) + F(7, 1) + F(9, 1) \\ &= 4F(4, 2) + 4F(1, 1) = 4F(4, 2) + 4 \quad (\text{Vì } 1, 3, 7 \text{ và } 9 \text{ đối xứng nên hàm } F \text{ bằng nhau}) \end{aligned}$$

$$\begin{aligned} F(4, 5) &= F(0, 4) + F(3, 4) + F(9, 4) \\ &= F(4, 3) + F(6, 3) + F(4, 3) + F(8, 3) + F(4, 3) + F(2, 3) \\ &= 4F(4, 3) + F(8, 3) + F(2, 3) \\ &= 4F(4, 3) + F(1, 2) + F(3, 2) + F(7, 2) + F(9, 2) \\ &= 4F(4, 3) + 4F(1, 2) \\ &= 4F(4, 3) + 4(F(8, 1) + F(6, 1)) \end{aligned}$$

$$= 4F(4, 3) + 4F(4, 1) + 4$$

$$\begin{aligned} F(4, 6) &= F(0, 5) + F(3, 5) + F(9, 5) \\ &= F(4, 4) + F(6, 4) + F(4, 4) + F(8, 4) + F(4, 4) + F(2, 4) \\ &= 4F(4, 4) + F(8, 4) + F(2, 4) \\ &= 4F(4, 4) + F(1, 3) + F(3, 3) + F(7, 3) + F(9, 3) \\ &= 4F(4, 4) + 4F(1, 3) \\ &= 4F(4, 4) + 4(F(8, 2) + F(6, 2)) \\ &= 4F(4, 4) + 4(F(4, 2) + F(3, 1) + F(1, 1)) \\ &= 4F(4, 4) + 4F(4, 2) + 8 \end{aligned}$$

Ta dự đoán công thức của hàm F là:

$$F(4, N) = 4 \sum_{i=2}^N \left(2 - 2^{N-i \bmod 2} \right) F(4, i-2) + C$$

Giả sử công thức trên đúng với $N = k$, tức là:

$$F(4, k) = 4 \sum_{i=2}^k \left(2 - 2^{k-i \bmod 2} \right) F(4, i-2) + C$$

Ta cần chứng minh công thức đúng với $N = k + 1$, tức là chứng minh:

$$F(4, k+1) = 4 \sum_{i=2}^{k+1} \left(2 - 2^{k-i+1 \bmod 2} \right) F(4, i-2) + C$$

Ta có:

$$\begin{aligned} F(4, k+1) &= F(0, k) + F(3, k) + F(9, k) \\ &= F(4, k-1) + F(6, k-1) + F(4, k-1) + F(8, k-1) + F(4, k-1) + F(2, k-1) \\ &= 3F(4, k-1) + F(6, k-1) + F(8, k-1) + F(2, k-1) \\ &= 4F(4, k-1) + F(1, k-2) + F(3, k-2) + F(7, k-2) + F(9, k-2) \\ &= 4F(4, k-1) + 2F(3, k-2) + 2F(9, k-2) \\ &= 4F(4, k-1) + 4F(4, k-3) + 2(F(8, k-3) + F(2, k-3)) \\ &= 4 \sum_{i=2}^{k+1} \left(2 - 2^{k-i+1 \bmod 2} \right) F(4, i-2) + C \end{aligned}$$

Vậy công thức sau đúng với mọi N:

$$F(4, N) = 4 \sum_{i=2}^N \left(2 - 2^{N-i \bmod 2} \right) F(4, i-2) + C \leq 4 \frac{N}{2} F(4, i-2) + C1 \in O(N)$$

$$G(N) = \sum_{i=0}^9 F(i, N) \leq 10F(4, N) \in O(N)$$

Vậy độ phức tạp của hàm số trên là $O(N)$.

2.5 Source Code

Và source code dưới đây chính là cách giải diễn hình của các coder cũng như những người tham gia phỏng vấn. Có thể nói việc sử dụng quy hoạch động vào bài toán này cũng là ý mà Google muốn người tham gia phỏng vấn nhìn ra và áp dụng nó.

```

1 def KnightDialer(n):
2     if n == 1:
3         return 10
4     graph = {0:[4, 6], 1:[6, 8], 2:[7, 9], 3:[4, 8], 4:[0, 3, 9], 6:[0, 1, 7], 7:[2, 6],
5             8:[1, 3], 9:[2, 4]}
6     cnt = {0:1, 1:0, 2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0}
7     for i in range(n - 1):
8         tmp = {0:0, 1:0, 2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0}
9         for k in cnt:
10            for v in graph[k]:
11                tmp[v] += cnt[k]
12            cnt=tmp
13     return sum(cnt.values()) % (10**9 + 7)
14 n = int(input())
15 print(KnightDialer(n))

```

Cách thức hoạt động của source code trên như sau:

- Bước 1: Tạo một biến graph với kiểu dữ liệu dict để chứa các node và danh sách các node liền kề với node đó.
- Bước 2: Nếu $n = 1$ thì trả về 10.
- Bước 3: Tạo một biến cnt với kiểu dữ liệu dict để chứa các node và số chuỗi số có chữ số kết thúc là node đó, mặc định là 1 (vì nếu khác 1 sẽ không thể tính các nước đi nếu $n > 1$). Tạo biến $i = 0$.
- Bước 4: So sánh $i < n-1$: Nếu đúng tiếp tục bước 5. Nếu sai thì đến bước 12
- Bước 5: Tạo một biến tmp với kiểu dữ liệu dict để lưu trữ các node và số lượng chuỗi số có chữ số kết thúc là node đó, mặc định là 0 (vì vòng lặp ở đây chỉ hoạt động khi $n \geq 2$ nên nếu đặt mặc định khác 0 thì sẽ làm cho node số 5 bị sai).
- Bước 6: Tạo biến $k = 0$.
- Bước 7: So sánh $k < \text{len}(\text{cnt})$. Nếu đúng thì tiếp tục bước 8. Nếu sai thì $i += 1$ rồi quay lại bước 4.
- Bước 8: Tạo biến $v = 0$.
- Bước 9: So sánh $v < \text{len}(\text{graph}[k])$. Nếu đúng thì tiếp tục bước 10. Nếu sai thì $k += 1$ rồi quay lại bước 7.
- Bước 10: Tính số lượng chuỗi số có chữ số kết thúc là số v . Công thức: $\text{tmp}[v] += \text{cnt}[k]$. Sau đó $v += 1$ rồi quay lại bước 9.
- Bước 11: Gán $\text{cnt} = \text{tmp}$ (vì cnt lúc này sẽ là số chuỗi số có thể thu được sau khi đã đi i bước).
- Bước 12: Trả về tổng các giá trị của các node trong cnt $\text{sum}(\text{cnt.values()})$.

2.6 Phát sinh input và output

Với source code trên ta có input và output như sau:

2 Phương pháp Dynamic Programming

input	output
25	715338219
50	267287516
75	533889181
100	540641702
500	84202957
750	185434245
1000	88106097
2500	851996060
5000	406880451
7500	549384636
10000	796663529
25000	600978592
50000	973717386
75000	414048711
100000	498938219
250000	287289220
500000	854741617
750000	414048711
1000000	643304746

Để có thể có được bộ test case như trên, chúng em đã sử dụng code giải sẵn và sau đó phát sinh input như trong bảng trên để có được bộ output tương ứng rồi sử dụng chúng để kiểm tra tính đúng đắn của code mà nhóm em đã trình bày.

2.7 Phân tích độ phức tạp bằng thực nghiệm

2 Phương pháp Dynamic Programming

N	t	$14.40943975 \cdot \lg(n) - 135.16076602$		$0.34607078 \cdot \sqrt{n} - 33.92075127$		$0.00041295 \cdot n - 12.97214127$		$(4.78934874 \text{E-} 10) \cdot n^2 - 0.34656343$		$(5.02092652 \text{E-} 16) \cdot n^3 + 6.06230614$		$(2.1081061 \text{E-} 05) \cdot n \lg(n) - 11.279042$	
25	0.00028	-68.24540005	4657.472845	-32.19039737	1036.23971	-12.96181752	168.0159721	-0.346563131	0.120300157	6.06230614	36.74428133	-11.27659456	127.1678999
50	0.0006	-53.8359603	2898.375224	-31.47366132	990.6291254	-12.95149377	167.756733	-0.346562233	0.120521616	6.06230614	36.74416009	-11.27309308	127.0961556
75	0.00061	-45.40697839	2061.849083	-30.9236904	956.3123552	-12.94117002	167.4896701	-0.346560736	0.12052752	6.06230614	36.74112931	-11.26919374	127.0084764
100	0.00086	-39.42652055	1554.518337	-30.46004347	927.8666402	-12.93084627	167.2290271	-0.346558641	0.120699712	6.062306141	36.73070437	-11.26503605	126.9204137
500	0.00172	-5.968837559	35.64755756	-26.18237338	685.6067461	-12.76566627	163.0061522	-0.346443696	0.121217959	6.062306203	36.70392221	-11.18453788	125.1323653
750	0.00393	2.460144351	6.03298894	-24.44321264	597.662783	-12.66242877	160.4366445	-0.346294029	0.122656871	6.062306352	36.69205056	-11.12803709	123.9206914
1000	0.00491	8.440602191	71.16090275	-22.97703231	528.1696721	-12.55919127	157.8566407	-0.346084495	0.123197136	6.062306642	36.5603635	-11.06895269	122.630435
2500	0.01579	27.48884543	754.7687746	-16.61721227	276.6567645	-11.93976627	142.9353257	-0.343570087	0.129139672	6.062313985	36.31995559	-10.68414962	114.4887078
5000	0.03571	41.89828518	1752.475201	-9.449851739	89.97588151	-10.90739127	119.7514654	-0.334590058	0.137122133	6.062368902	36.00370691	-9.983851929	100.3916212
7500	0.06206	50.32726709	2526.591044	-3.950142571	16.09776947	-9.87501627	98.7454848	-0.319623343	0.145682175	6.06251796	35.57046448	-9.243769667	86.59846579
10000	0.09842	56.30772493	3159.485961	0.68632673	0.345634323	-8.84264127	79.94257663	-0.298669943	0.157680423	6.062808233	32.42685456	-8.477851248	73.55242852
25000	0.36835	75.35596817	5623.142878	20.79784355	417.3642068	-2.64839127	9.10072789	-0.047229134	0.172706016	6.070151338	22.82834119	-3.579373949	15.58452438
50000	1.29225	89.76540792	7827.499672	43.46302764	1778.374487	7.67535873	40.74407706	0.850773755	0.194901275	6.125067722	10.79561238	5.174347151	15.07067829
75000	2.8394	98.19438983	9092.574085	60.85463508	3365.767501	17.99910873	229.8167688	2.347445236	0.24201949	6.274126478	2.020879682	14.32591399	131.9400036
100000	4.85255	104.1748477	9864.918814	75.51643837	4993.38512	28.32285873	550.8553919	4.44278531	0.167907101	6.564398792	511.8124002	23.7358424	356.578732
250000	29.18767	123.2230909	8842.660384	139.1146387	12083.93845	90.26535873	3730.484061	29.5868662	0.159357602	13.90750383	10579.96075	83.22508071	2920.041756
500000	116.76644	137.6325307	435.3937391	210.788244	8840.099635	193.5028587	5888.47796	119.3871551	6.868147478	68.82388764	37078.21456	188.2697339	5112.721041
750000	261.38093	146.0615126	13298.56804	265.7853357	19.39878972	296.7403587	1250.2892	269.0543032	58.88065619	217.8826437	70593.17745	297.2928445	1289.665603
1000000	483.57641	152.0419704	109915.0846	312.1500287	29387.0042	399.9778587	6988.717774	478.5883106	24.88113592	508.1549581	258221.4615	408.8995708	5576.630308
	MSE		9704.12		3525.84		1067.46		4.89398		19864.1		877.534

3 Phương pháp Dynamic Programming kết hợp toán học

3.1 Đặt vấn đề bài toán

Sau khi thực nghiệm để tính độ phức tạp của đoạn code trên nhóm em đã nghĩ rằng: "Nếu N đủ lớn thì liệu code trên có khả thi hay không?". Nếu ta để ý code trên thì sẽ thấy dữ liệu sẽ được lưu trữ trên 10 ô nhớ, và sau mỗi vòng lặp dữ liệu trong các ô nhớ ấy sẽ tăng lên cũng như nó còn phải truy xuất từng ô nhớ và cộng vào ô dữ liệu tạm rồi lại phải chuyển tất cả dữ liệu tạm đó qua bộ nhớ dữ liệu chính. Ta còn có thể thấy sự khác biệt rõ rệt nếu N khoảng trên 250000. Như vậy, với $N \geq 250000$, thuật toán trên sẽ chậm rất nhiều vì nó phải thực hiện ít nhất 250000 lần gọi vòng lặp và trong 250000 lần đó nó còn phải thực hiện thao tác tạo mảng tạm, gọi vòng lặp, tính các dữ liệu trong mảng tạm và cộng lại rồi chuyển toàn bộ kết quả từ mảng tạm sang mảng chính. Sau cùng thực hiện việc cộng các kết quả lại trong mảng chính, trả về giá trị cần tính cho đề bài khi hết vòng lặp.

3.2 Phương pháp thiết kế thuật toán

Sau khi phân tích các kết quả thu được từ code nguồn ở trên, nhóm em đã nhìn ra được quy luật chung của nó, cụ thể vẫn sử dụng thuật toán Dynamic programming và dùng thêm phương pháp bottom up để tối ưu hóa nó.

Giả sử ta có 4 base case bao gồm:

$$N = 1 \rightarrow 10$$

$$N = 2 \rightarrow 20$$

$$N = 3 \rightarrow 46$$

$$N = 4 \rightarrow 104$$

$$N = 5 \rightarrow 240$$

Với $N = \{1, 2, 3\}$ là 3 base case gốc của bài toán này. Ta có:

$$N = 1 \rightarrow 10$$

$$N = 2 \rightarrow 20$$

$$N = 3 \rightarrow 20 \cdot 2 + 6 = 46$$

$$N = 4 \rightarrow 46 \cdot 2 + 6 = 104$$

$$N = 5 \rightarrow 104 \cdot 2 + 6 = 240$$

3 Phương pháp Dynamic Programming kết hợp toán học

Ta dễ dàng thấy được tại vị trí N thì ta chỉ cần giá trị liền kề trước nó nhân đôi và cộng thêm với 1 biến số cũng mang tính quy luật.

Cụ thể ta để ý biến số này sẽ chia ra hai trường hợp:

. Nếu N lẻ thì giá trị của biến số sẽ bằng hiệu của giá trị tại $N - 1$ và $2*$ giá trị tại $N - 2$, sau đó cộng cho giá trị tại $N - 3$.

. Nếu N chẵn thì giá trị của biến số sẽ bằng hiệu của giá trị tại $N - 1$ và $2*$ giá trị tại $N - 2$, sau đó nhân cả hiệu cho 2.

3.3 Pseudo Code

Input:

. N : số bước con mã phải đi

Function DP-Bottomup (n)

Tạo một mảng chứa 4 base case, 1 biến cnt có giá trị bằng 5

$F = \{10, 20, 46, 104\}$

if $n < 0$:

return $F[n - 1]$

while $cnt \in n$ do:

$temp = F[3] - F[2] * 2$

if $cnt \% 2 \neq 0$ do:

$temp = temp + F[1]$

else do:

$temp = temp * 2$

$F.append(F[3] * 2 + temp)$

$F.pop(0)$

$cnt = cnt + 1$

return $F[-1] \% (10 * 9 + 7)$

3.4 Phân tích độ phức tạp bằng các phương pháp toán học

Ta có công thức tổng quát như sau:

$$F[i] = F[i - 1] * 2 + temp$$

Nếu i lẻ:

$$temp = F[i - 1] - F[i - 2] * 2 + F[i - 3]$$

Nếu i chẵn:

$$\text{temp} = 2 * F[i - 1] - 4 * F[i - 2]$$

Như vậy với mỗi lần lặp chúng ta cần nhiều nhất ba vị trí liền kề trước đó, đồng thời cả ba vị trí trước đó đều có kết quả nên $F[i - 1] = F[i - 2] = F[i - 3] = O(1)$. Với mỗi lần lặp ta thực hiện một phép gán $F[i]$ và thực hiện $(N - 1) - 3$ (cần nhiều nhất 3 $F[i]$ trước đó để tính) nên tổng là $O(N - 4) = O(N)$.

$$\Rightarrow O(N) + O(1) + O(1) + O(1) = O(N)$$

Và vì chúng ta chỉ cần 4 không gian trong một mảng để lưu kết quả cho các vị trí cần thực thi ($O(4)$) nên ta không cần phải chứng minh độ phức tạp không gian lưu trữ mà ta có thể suy ra thẳng là $O(1)$

Vậy độ phức tạp thuật toán là $O(N)$, độ phức tạp không gian lưu trữ là $O(1)$

3.5 Source Code

Từ mã giả trên, ta khai triển được source code sau:

```

1 def KnightDialer(n):
2     f = [10, 20, 46, 104]
3     if n <= 4:
4         return f[n-1]
5     cnt = 5
6     while cnt <= n:
7         temp = f[3] - f[2]*2
8         if cnt % 2 != 0:
9             temp += f[1]
10        else:
11            temp *= 2
12        f.append(f[3]*2 + temp)
13        f.pop(0)
14        cnt += 1
15    return f[-1] % (10**9+7)
16 n = int(input())
17 print(KnightDialer(n))

```

Cách thức hoạt động của source code trên là:


- Bước 1: Tạo một mảng f chứa kết quả của 4 base case là $n = 1, 2, 3, 4$.

3 Phương pháp Dynamic Programming kết hợp toán học

- Bước 2: Nếu $n \leq 4$ thì trả về $f[n-1]$. Nếu sai tiếp tục bước 3.
- Bước 3: Tạo biến $cnt = 5$.
- Bước 4: So sánh $cnt \leq n$. Nếu đúng tiếp tục bước 5. Nếu sai thì đến bước 9.
- Bước 5: Tạo một biến $temp = f[3] - 2 * f[2]$
- Bước 6: Nếu cnt là số lẻ: $temp = temp + f[1]$. Ngược lại cnt là số chẵn: $temp = 2 * temp$
- Bước 7: Thêm $2 * f[3] + temp$ vào mảng cnt .
- Bước 8: Bỏ phần tử đầu tiên của mảng cnt ra khỏi mảng. Sau đó $cnt = cnt + 1$, quay lại bước 4.
- Bước 9: Trả về giá trị của phần tử cuối trong mảng chính là số chuỗi khác nhau sau khi quân mã thực hiện n bước đi.

Dưới đây là link bài viết của em về code của em trên trang leetcode về vấn đề này:
<https://leetcode.com/problems/knight-dialer/discuss/992550>

[< Back](#) Python 3 solution solved with formula, no graph, no DFS. Beat 97.5%

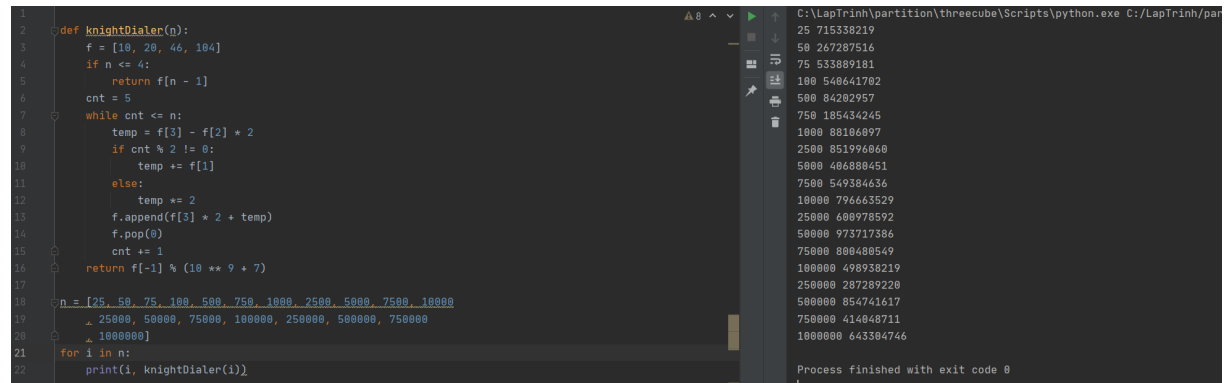
 19520197 ★ 1 Last Edit: December 30, 2020 4:19 AM 67 VIEWS

1

I have realized the principle of the solution and this source will run as fast as it can.
Thumbs up if you like this and to support me ^^.
for example:
with $n = 2 \Rightarrow 20$ ways
with $n = 3 \Rightarrow 46$ ways = $40 + 6$
with $n = 4 \Rightarrow 104$ ways = $92 + 12$
with $n = 5 \Rightarrow 240$ ways = $208 + 32$
with $n = 6 \Rightarrow 544$ ways = $480 + 64$
So:
We will create a 'cnt' starts at 5 because 1 2 3 4 are special cases and we already know the answer so just put it in an array and we only need 4 elements in this array to use, to reduce space and to reduce time when we compute.
At start the array will be like [10, 20, 46, 104].
We use a loop to compute:
The loop will stop when $cnt > n$ and cnt starts at 5. In case $n < 5$ we will return $f[n-1]$
Let's do it.
 $temp = f[3] - f[2] * 2$
if cnt is odd we will plus it with $f[1]$:
 $temp += f[1]$
elif cnt is even we double temp:
 $temp = temp * 2$
Then we use the formula:
 $x = f[3] * 2 + temp$
its look seem like fibo a little bit
Then we append x into f and pop first element in f because our formula just need $f[1]$ $f[2]$ $f[3]$
Then we plus cnt by 1:
 $cnt += 1$
After this loop, we return $f[3] \% (10^{**9} + 7)$

3.6 Phát sinh input và output

Với cùng bộ input như phía trên đã trình bày thì code sử dụng phương pháp bottom-up của nhóm em cũng ra kết quả giống với output của bộ test case. Chứng tỏ rằng source code sử dụng phương pháp bottom-up này có tính đúng đắn.



```
1 def knightDialer(n):
2     f = [10, 20, 46, 104]
3     if n <= 4:
4         return f[n - 1]
5     cnt = 5
6     while cnt <= n:
7         temp = f[3] - f[2] * 2
8         if cnt % 2 != 0:
9             temp += f[1]
10        else:
11            temp *= 2
12        f.append(f[3] * 2 + temp)
13        f.pop(0)
14        cnt += 1
15    return f[-1] % (10 ** 9 + 7)
16
17 n = [25, 50, 75, 100, 500, 750, 1000, 2500, 5000, 7500, 10000]
18     25000, 50000, 75000, 100000, 250000, 500000, 750000
19     1000000]
20
21 for i in n:
22     print(i, knightDialer(i))
```

C:\LapTrinh\partition\threecube\Scripts\python.exe C:/LapTrinh/par
25 715338219
50 267287516
75 533889181
100 540641702
500 84202957
750 185434245
1000 80106097
2500 851996060
5000 406808451
7500 549384636
10000 796663529
25000 608978592
50000 973717386
75000 808480549
100000 498938219
250000 287289220
500000 854741617
750000 414048711
1000000 643304746
Process finished with exit code 0

3.7 Phân tích độ phức tạp bằng thực nghiệm

3 Phương pháp Dynamic Programming kết hợp toán học

N	t	$3.43914494 \cdot \lg(n) - 32.24937551$		$0.08244813 \cdot \sqrt{n} - 8.05095223$		$(9.82006242E-05) \cdot n - 3.03381703$		$(1.13551219E-10) \cdot n^2 + 0.00266263$		$(1.18778825E-16) \cdot n^3 + 1.54379044$		$(5.01191389E-06) \cdot n \lg(n) - 2.62768665$	
25	0.00015	-16.27848099	264.993827	-7.63871158	58.35220624	-3.031362014	9.190065093	0.002662701	6.31E-06	1.54379044	2.382825808	-2.627104785	6.902467704
50	0.00021	-12.83933605	164.8539428	-7.467955912	55.77350209	-3.028906999	9.175549792	0.002662914	6.02E-06	1.54379044	2.382640575	-2.626272324	6.898409398
75	0.00025	-10.82756523	117.2415826	-7.336930479	53.83421739	-3.026451983	9.160924895	0.002663269	5.82E-06	1.54379044	2.38251709	-2.625345277	6.89375056
100	0.00048	-9.400191113	88.37261737	-7.22647093	52.22881974	-3.023996968	9.147460927	0.002663766	4.77E-06	1.54379044	2.381807115	-2.624356806	6.889768261
500	0.00041	-1.414743854	2.002660431	-6.207355997	38.53635867	-2.984716718	8.910981522	0.002691018	5.20E-06	1.543790455	2.382023228	-2.605218781	6.789301342
750	0.00107	0.59702697	0.35516471	-5.793017199	33.57144647	-2.960166562	8.768921975	0.002726503	2.74E-06	1.54379049	2.379986511	-2.59178601	6.722902287
1000	0.00123	2.024401086	4.093221242	-5.443713434	29.647409	-2.935616406	8.625066811	0.002776181	2.39E-06	1.543790559	2.379493078	-2.577738997	6.651081089
2500	0.00365	6.570703404	43.12619041	-3.92854573	15.46216326	-2.78831547	7.795071183	0.003372325	7.71E-08	1.543792296	2.372038292	-2.486254044	6.199622148
5000	0.00585	10.00984834	100.0799829	-2.220989048	4.958812146	-2.542813909	6.495687721	0.00550141	1.22E-07	1.543805287	2.365306466	-2.319761868	5.408470562
7500	0.01253	12.02161917	144.2182227	-0.910734723	0.852417748	-2.297312349	5.335371675	0.009049886	1.21E-05	1.54384055	2.344912	-2.143811115	4.649807003
10000	0.01735	13.44899328	180.4090413	0.19386077	0.031156052	-2.051810788	4.281426367	0.014017752	1.11E-05	1.543909219	2.330383049	-1.961717948	3.916709941
25000	0.0956	17.9952956	320.3991027	4.985241751	23.90859645	-0.578801425	0.454817282	0.073632142	0.000482587	1.545646359	2.102634444	-0.797130149	0.79696712
50000	0.31568	21.43444054	446.0020469	10.3850101	101.3914087	1.87621418	2.435266927	0.286540678	0.0008491	1.558637793	1.544944075	1.284022046	0.937686317
75000	0.67456	23.44621137	518.548106	14.52839808	191.9288296	4.331229785	13.37123392	0.641388237	0.001100366	1.593900257	0.845186508	3.45976002	7.757339149
100000	1.16751	24.87358548	561.9780148	18.02143573	284.0548126	6.78624539	31.57018738	1.13817482	0.000860553	1.662569265	0.245083676	5.69692213	20.51557424
250000	7.04601	29.4198878	500.5904079	33.17311277	682.6254992	21.51633902	209.3904219	7.099613818	0.002873369	3.399709581	13.29550675	19.84018275	163.6908562
500000	28.704	32.85903274	17.26429708	50.24867959	464.1732186	46.06649507	301.4562351	28.39046738	0.098302704	16.39114357	151.6064336	44.81400909	259.5323928
750000	63.2116	34.87080357	803.2007425	63.35122284	0.019494539	70.61665112	54.83478209	63.87522332	0.440395908	51.65360724	133.5871967	70.73369322	56.58188635
1000000	113.85157	36.29817768	6014.52866	74.39717777	1556.649066	95.16680717	349.120362	113.5538816	0.088618366	120.3226154	41.87442909	97.26761871	275.0274403
	MSE		541.6977806		191.9999702		55.23788603		0.033344191		19.53607095		44.8822333

4 Đánh giá và so sánh

Sau khi phân tích độ phức tạp bằng thực hiện ở cả hai source code trên, nhóm em đều nhận được kết quả cuối cùng là $O(N^2)$ khác so với kết quả độ phức tạp đã chứng minh bằng các phương pháp toán học là $O(N)$. Từ đó chúng em đã nhận ra rằng vì dữ liệu quá lớn nên khi thực hiện các phép toán và gọi vòng lặp sẽ làm cho chương trình chậm đi rất nhiều. Vì thế để thuật toán có thể thực hiện với tốc độ tối ưu và tránh việc lưu số quá lớn sẽ làm chậm chương trình nên chúng em đã thay đổi bằng cách chia lấy phần dư ngay trong bước cộng để giảm thời gian tính toán các số lớn từ đó tăng hiệu quả của chương trình lên rất nhiều.

4.1 Phương pháp Dynamic Programming

Source code mẫu sau khi đã tối ưu:

```
1 def KnightDialer(n):
2     if n == 1:
3         return 10
4     graph = {0:[4, 6], 1:[6, 8], 2:[7, 9], 3:[4, 8], 4:[0, 3, 9], 6:[0, 1, 7],
5             7:[2, 6], 8:[1, 3], 9:[2, 4]}
6     cnt = {0:1, 1:0, 2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0}
7     for i in range(n - 1):
8         tmp = {0:0, 1:0, 2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0}
9         for k in cnt:
10            for v in graph[k]:
11                tmp[v] = (tmp[v] + cnt[k]) % (10**9+7)
12            cnt=tmp
13     return sum(cnt.values()) % (10**9 + 7)
14 n = int(input())
15 print(KnightDialer(n))
```

Bảng thực nghiệm độ phức tạp:

4 Đánh giá và so sánh

N	t	$0.37562518 \cdot \lg(n) - 3.41947518$		$0.00819578 \cdot \sqrt{n} - 0.58636561$		$(9.17431942 \cdot 10^{-6}) \cdot n - 0.00177538$		$(9.89584508 \cdot 10^{-12}) \cdot n^2 + 0.35293736$		$(9.95347464 \cdot 10^{-18}) \cdot n^3 + 0.52000358$		$(4.64898369 \cdot 10^{-7}) \cdot \lg(n) + 0.04545163$	
25	0	-1.675125863	2.806046656	-0.54538671	0.297446663	-0.001546022	2.39018E-06	0.352937366	0.124564784	0.52000358	0.270403723	0.045505603	0.00207076
50	0	-1.299500683	1.688702025	-0.528412694	0.279219975	-0.001316664	1.7336E-06	0.352937385	0.124564798	0.52000358	0.270403723	0.045582821	0.002077794
75	0.001	-1.079774038	1.168072522	-0.515388073	0.266656642	-0.001087306	4.35685E-06	0.352937416	0.123859945	0.52000358	0.269364716	0.045668813	0.001995303
100	0.001	-0.923875503	0.855394696	-0.50440781	0.255437054	-0.000857948	3.45197E-06	0.352937459	0.123859975	0.52000358	0.269364716	0.045760502	0.002003503
500	0.004	-0.051700844	0.003102584	-0.403102398	0.165732362	0.00281178	1.41187E-06	0.352939834	0.121759008	0.520003581	0.266259696	0.047535719	0.001895359
750	0.007	0.1680258	0.025929308	-0.361914931	0.136098226	0.00510536	3.58966E-06	0.352942926	0.119676508	0.520003584	0.263172677	0.048781725	0.001745713
1000	0.009	0.323924336	0.099177337	-0.32719229	0.113025256	0.007398939	2.56339E-06	0.352947256	0.118299715	0.52000359	0.261124669	0.050084707	0.001687953
2500	0.024	0.820473814	0.634370537	-0.17657661	0.040230976	0.021160419	8.06322E-06	0.352999209	0.10824048	0.520003736	0.246019706	0.058570728	0.001195135
5000	0.046	1.196098994	1.322727697	-0.006836448	0.00279169	0.044096217	3.62439E-06	0.353184756	0.094362474	0.520004824	0.224680573	0.074014317	0.000784802
7500	0.07	1.415825639	1.811246651	0.123409758	0.002852602	0.067032016	8.80893E-06	0.353494001	0.080368849	0.520007779	0.202507001	0.090335272	0.000413523
10000	0.09201	1.571724174	2.189554038	0.23321239	0.019938115	0.089967814	4.17052E-06	0.353926945	0.068600486	0.520013533	0.183187025	0.107225988	0.000231526
25000	0.22702	2.068273653	3.390215014	0.70950099	0.232787906	0.227582606	3.16525E-07	0.359122263	0.017451008	0.520159103	0.085930534	0.215251581	0.000138496
50000	0.45903	2.443898833	3.939704284	1.246266511	0.619741324	0.456940591	4.36563E-06	0.377676973	0.006618315	0.521247764	0.00387105	0.40829645	0.002573893
75000	0.68507	2.663625478	3.914681778	1.658141181	0.946867524	0.686298577	1.5094E-06	0.408601489	0.076434838	0.524202702	0.025878288	0.610114968	0.005618257
100000	0.91607	2.819524013	3.623137179	2.00536759	1.18656924	0.915656562	1.70931E-07	0.451895811	0.215457678	0.529957055	0.149083207	0.817631107	0.009690216
250000	2.27717	3.316073492	1.079320465	3.51152439	1.52363076	2.291804475	0.000214168	0.971427678	1.704963013	0.675526621	2.565261513	2.129540875	0.021794358
500000	4.56335	3.691698672	0.759776038	5.208926005	0.416768378	4.58538433	0.000485512	2.82689863	3.01526336	1.76418791	7.835308406	4.446079305	0.013752416
750000	6.87952	3.911425316	8.809586052	6.511388074	0.135521115	6.878964185	3.0893E-07	5.919350218	0.921926011	4.719125694	4.667303558	6.850354227	0.000850642
1000000	9.1867	4.067323852	26.20801215	7.60941439	2.487829896	9.17254404	0.000200391	10.24878244	1.128019109	10.47347822	1.655798187	9.311605349	0.015601346
	MSE		2.117819159		0.368961989		4.16953E-05		0.39812618		1.00328471		0.003917758

4.2 Phương pháp Dynamic Programming kết hợp toán học

Source code của nhóm sau khi đã tối ưu:

```
1 def KnightDialer(n):
2     f = [10, 20, 46, 104]
3     if n <= 4:
4         return f[n-1]
5     cnt = 5
6     while cnt <= n:
7         temp = f[3] - f[2]*2
8         if cnt % 2 != 0:
9             temp += f[1]
10        else:
11            temp *= 2
12        f.append((f[3]*2 + temp) % (10**9+7))
13        f.pop(0)
14        cnt += 1
15    return f[-1] % (10**9+7)
16 n = int(input())
17 print(KnightDialer(n))
```

Bảng thực nghiệm độ phức tạp:

4 Đánh giá và so sánh

N	t	$0.04817743 \cdot \lg(n) - 0.43840558$		$0.00104942 \cdot \sqrt{n} - 0.07461928$		$(1.173288E-06) \cdot n + 0.00044382$		$(1.26362835E-12) \cdot n^2 + 0.04600024$		$(1.26970684E-18) \cdot n^3 + 0.0674388$		$(5.94463344E-08) \cdot \lg(n) + 0.00650788$	
25	0	-0.214676523	0.04608601	-0.06937218	0.004812499	0.000473152	2.23873E-07	0.046000241	0.002116022	0.0674388	0.004547992	0.006514782	4.24424E-05
50	0	-0.166499093	0.027721948	-0.06719876	0.004515673	0.000502484	2.52491E-07	0.046000243	0.002116022	0.0674388	0.004547992	0.006524655	4.25711E-05
75	0	-0.138317104	0.019131621	-0.065531036	0.004294317	0.000531817	2.82829E-07	0.046000247	0.002116023	0.0674388	0.004547992	0.006535651	4.27147E-05
100	0	-0.118321663	0.014000016	-0.06412508	0.004112026	0.000561149	3.14888E-07	0.046000253	0.002116023	0.0674388	0.004547992	0.006547375	4.28681E-05
500	0.001	-0.006457135	5.56089E-05	-0.051153535	0.002719991	0.001030464	9.28055E-10	0.046000556	0.00202505	0.0674388	0.004414114	0.006774372	3.33434E-05
750	0.001	0.021724855	0.00042952	-0.04587973	0.002197709	0.001323786	1.04837E-07	0.046000951	0.002025086	0.067438801	0.004414114	0.006933698	3.52088E-05
1000	0.001	0.041720295	0.001658142	-0.041433706	0.001800619	0.001617108	3.80822E-07	0.046001504	0.002025135	0.067438801	0.004414114	0.007100309	3.72138E-05
2500	0.003	0.105407393	0.010487274	-0.02214828	0.000632436	0.00337704	1.42159E-07	0.046008138	0.0018497	0.06743882	0.004152362	0.008185413	2.68885E-05
5000	0.006	0.153584823	0.02178128	-0.00041408	4.11404E-05	0.00631026	9.62613E-08	0.046031831	0.001602547	0.067438959	0.003774746	0.010160177	1.73071E-05
7500	0.008	0.181766813	0.030194905	0.016263158	6.82798E-05	0.00924348	1.54624E-06	0.046071319	0.001449425	0.067439336	0.003533035	0.01224713	1.80381E-05
10000	0.011	0.201762253	0.036390237	0.03032272	0.000373368	0.0121767	1.38462E-06	0.046126603	0.001233878	0.06744007	0.003185481	0.014406938	1.16072E-05
25000	0.029	0.265449351	0.055908296	0.091308591	0.003882361	0.02977602	6.02207E-07	0.046790008	0.000316484	0.067458639	0.001479067	0.028220119	6.08214E-07
50000	0.066	0.313626781	0.061319023	0.160038166	0.008843177	0.05910822	4.74966E-05	0.049159311	0.000283609	0.067597513	2.55205E-06	0.052904675	0.000171488
75000	0.08701	0.341808771	0.064922414	0.212776223	0.015817143	0.08844042	2.0461E-06	0.053108149	0.001149335	0.067974458	0.000362352	0.078711114	6.88715E-05
100000	0.11701	0.361804211	0.059924206	0.257236462	0.019663461	0.11777262	5.81589E-07	0.058636524	0.003407463	0.068708507	0.002333034	0.105246104	0.000138389
250000	0.29502	0.42549131	0.017022763	0.45009072	0.024046928	0.29376582	1.57297E-06	0.124977012	0.028914618	0.087277969	0.043156751	0.272999385	0.000484907
500000	0.58806	0.47366874	0.01308536	0.667432718	0.006300028	0.58708782	9.45134E-07	0.361907328	0.051145031	0.226152155	0.130977288	0.569214058	0.00035517
750000	0.88407	0.501850729	0.146091571	0.834205099	0.002486508	0.88040982	1.33969E-05	0.756791187	0.016199896	0.603096373	0.078946179	0.876647554	5.50927E-05
1000000	1.17007	0.52184617	0.420194134	0.97480072	0.038130092	1.17373182	1.34089E-05	1.30962859	0.0194766	1.33714564	0.027914269	1.19136657	0.000453544
	MSE		0.034789455		0.005922648		3.96508E-06		0.006782853		0.016852064		9.02627E-05

4 Đánh giá và so sánh

→ Cả hai source code sau khi đã tối ưu đều có độ phức tạp là $O(N)$ giống với kết quả đã chứng minh bằng phương pháp toán học.

5 Tổng Kết

Knight Dialer không phải là một bài toán quá khó cho các lập trình viên. Mục đích chính của bài toán này chỉ để đánh giá khả năng tư duy, kỹ năng, mức độ thành thục các thuật toán và khả năng trình bày ý tưởng, chứng minh tính đúng đắn của thuật toán họ có thể đưa ra. Đây là một bài toán khá hay để kiểm tra kiến thức của sinh viên sau khi học qua thuật toán Dynamic Programming và Backtracking.