



BÁO CÁO ĐỒ ÁN CUỐI KỲ

CS112.L13.KHCL

Instructor: Pham Nguyen Truong An

KNIGHT DIALER

Ngày 19 tháng 1 năm 2021

NHÓM

Le Doan Thien Nhan 19520197

Truong Minh Chau 19521281

Pham Minh Khoi 19520658

TABLE OF CONTENTS

1	Giới thiệu bài toán	2
2	Các ý tưởng thiết kế thuật toán cho bài toán trên	3

1 Giới thiệu 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 thục 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.

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.

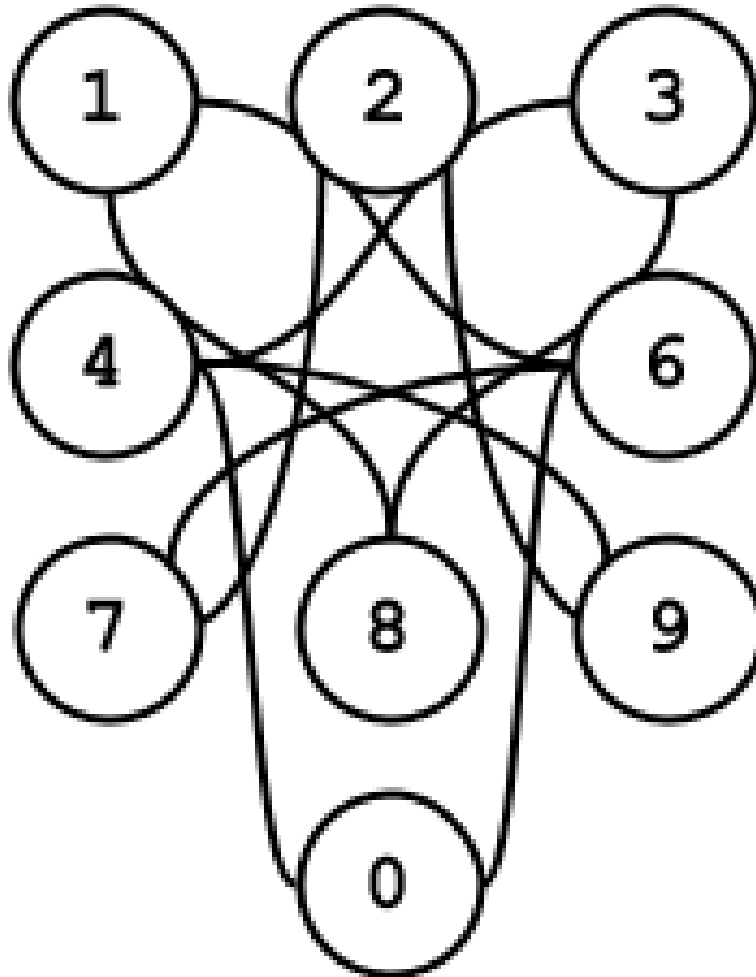


Hãy tưởng rằng bạn quay số điện thoại bằng cách sử dụng những bước nhảy của quân mã. Mỗi khi quân cờ đi qua một phím thì ta sẽ thêm phím đó vào dãy số điện thoại và tiếp tục di chuyển. Vị trí đứng ban đầu của quân mã cũng được tính là một số trong chuỗi số. Chúng ta có thể quay bao nhiêu chuỗi số khác 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 Các ý tưởng thiết kế thuật toán cho bài toán trên

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.



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.

Mã giả cho bài toán:

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 đến 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)

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

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

Ta có:

$$\begin{aligned} F(6, 2) &= F(4, 2) = 6 \\ F(6, 3) &= F(4, 3) = 16 \end{aligned}$$

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:

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

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:

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

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

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

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

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

$$= 4F(4,4) + 4F(4,2) + 8$$

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 $O(n)$.

Và source code dưới đây chính là cách giải điển 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.

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.

```

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

```

- 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 $\text{sum}(\text{cnt.values()})$.

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

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.

Phân tích độ phức tạp cài đặt bằng thực nghiệm

A	B	C	D	E	F	G	H	I	J	K	L	M	N
N	t	$14.40943975 \cdot \lg(n) - 135.16076602$		$0.34607078 \cdot \sqrt{n} - 33.92075127$		$0.00041295 \cdot n - 12.97214127$		$(4.78934874E-10) \cdot n^2 - 0.34656343$		$(5.02092652E-16) \cdot n^3 + 6.06230614$		$(2.1081061E-05) \cdot \lg(n) - 11.279042$	
25	0.00028	-68.24540005	4657.47	-32.19039737	1036.24	-12.96181752	168.016	-0.346563131	0.1203	6.06230614	36.7443	-11.27659456	127.168
50	0.0006	-53.8359603	2898.38	-31.47366132	990.629	-12.95149377	167.757	-0.346562233	0.12052	6.06230614	36.7442	-11.27309308	127.096
75	0.00061	-45.40697839	2061.85	-30.9236904	956.312	-12.94117002	167.49	-0.346560736	0.12053	6.06230614	36.7411	-11.26919374	127.008
100	0.00086	-39.42652055	1554.52	-30.46004347	927.867	-12.93084627	167.229	-0.346558641	0.1207	6.062306141	36.7307	-11.26503605	126.92
500	0.00172	-5.968837559	35.6476	-26.18237338	685.607	-12.76566627	163.006	-0.346443696	0.12122	6.062306203	36.7039	-11.18453788	125.132
750	0.00393	2.460144351	6.03299	-24.44321264	597.663	-12.66242877	160.437	-0.346294029	0.12266	6.062306352	36.6921	-11.12803709	123.921
1000	0.00491	8.440602191	71.1609	-22.97703231	528.17	-12.55919127	157.857	-0.346084495	0.1232	6.062306642	36.5604	-11.06895269	122.63
2500	0.01579	27.48884543	754.769	-16.61721227	276.657	-11.93976627	142.935	-0.343570087	0.12914	6.062313985	36.32	-10.68414962	114.489
5000	0.03571	41.89828518	1752.48	-9.449851739	89.9759	-10.90739127	119.751	-0.334590058	0.13712	6.062368902	36.0037	-9.983851929	100.392
7500	0.06206	50.32726709	2526.59	-3.950142571	16.0978	-9.87501627	98.7455	-0.319623343	0.14568	6.06251796	35.5705	-9.243769667	86.5985
10000	0.09842	56.30772493	3159.49	0.68632673	0.34563	-8.84264127	79.9426	-0.298669943	0.15768	6.062808233	32.4269	-8.477851248	73.5524
25000	0.36835	75.35596817	5623.14	20.79784355	417.364	-2.64839127	9.10073	-0.047229134	0.17271	6.070151338	22.8283	-3.579373949	15.5845
50000	1.29225	89.76540792	7827.5	43.46302764	1778.37	7.67535873	40.7441	0.850773755	0.1949	6.125067722	10.7956	5.174347151	15.0707
75000	2.8394	98.19438983	9092.57	60.85463508	3365.77	17.99910873	229.817	2.347445236	0.24202	6.274126478	2.02088	14.32591399	131.94
100000	4.85255	104.1748477	9864.92	75.51643837	4993.39	28.32285873	550.855	4.44278531	0.16791	6.564398792	511.812	23.7358424	356.579
250000	29.1877	123.2230909	8842.66	139.1146387	12083.9	90.26535873	3730.48	29.5868662	0.15936	13.90750383	10580	83.22508071	2920.04
500000	116.766	137.6325307	435.394	210.788244	8840.1	193.5028587	5888.48	119.3871551	6.86815	68.82388764	37078.2	188.2697339	5112.72
750000	261.381	146.0615126	13298.6	265.7853357	19.3988	296.7403587	1250.29	269.0543032	58.8807	217.8826437	70593.2	297.2928445	1289.67
1000000	483.576	152.0419704	109915	312.1500287	29387	399.9778587	6988.72	478.5883106	24.8811	508.1549581	258221	408.8995708	5576.63
	MSE		9704.12		3525.84		1067.46		4.89398		19864.1		877.534

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.

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:

$$N = 1 \rightarrow 10$$

$$N = 2 \rightarrow 20$$

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

$$N = 3 \rightarrow 46 \cdot 2 + 6 = 104$$

$$N = 3 \rightarrow 104 \cdot 2 + 6 = 240$$

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.

Mã giả dưới đây sẽ thể hiện ý vừa nêu trên:

Input:

. N : số bước con mã phải đi

`function knightdialer(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 ≤ n do:`

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

`if cnt % 2 ≠ 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)`

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

Ta có công thức tổng quát như sau:

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

Nếu i lẻ:

$$\text{temp} = F[i - 1] - F[i - 2] * 2 + F[i - 3]$$

Nếu i chẵn:

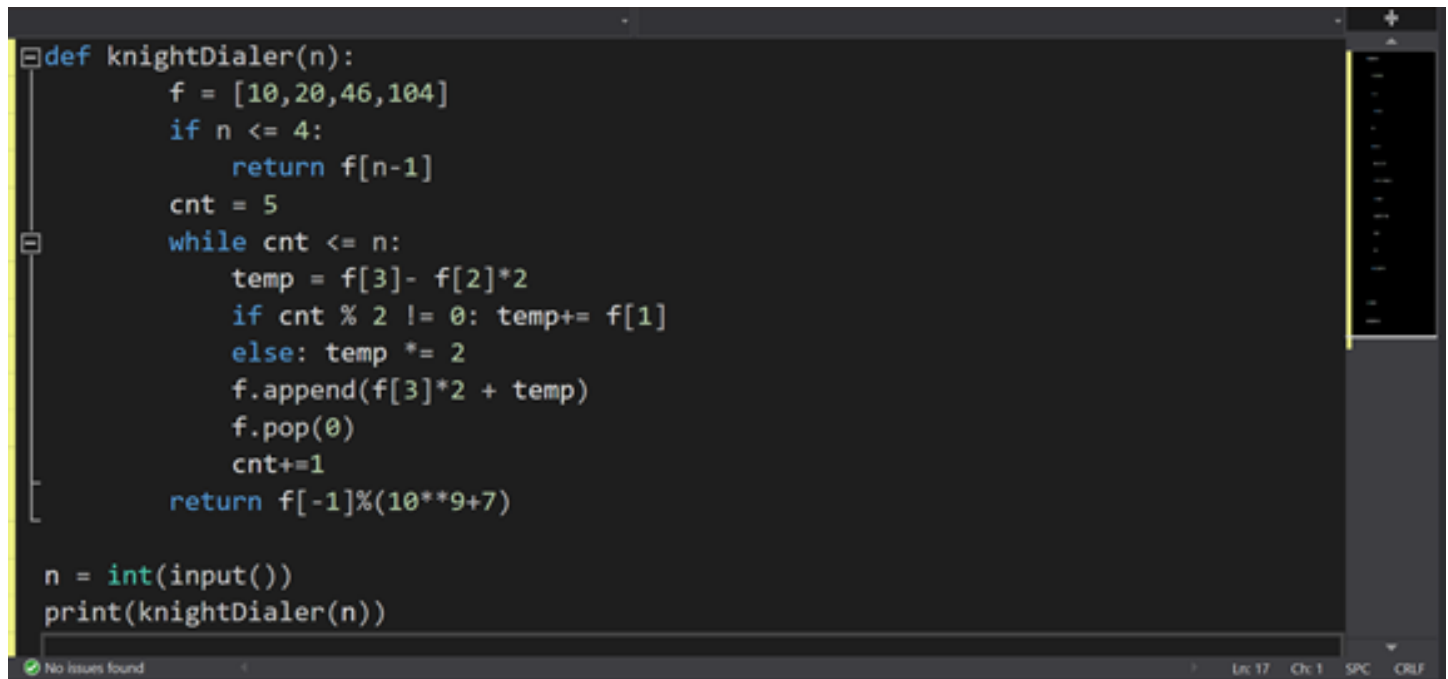
$$\text{temp} = 2 * F[i - 1] - 4 * F[i - 2]$$

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 $O(N-4) = O(N)$.

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

Vậy độ phức tạp là $O(N)$

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



```
def knightDialer(n):
    f = [10,20,46,104]
    if n <= 4:
        return f[n-1]
    cnt = 5
    while cnt <= n:
        temp = f[3] - f[2]*2
        if cnt % 2 != 0: temp += f[1]
        else: temp *= 2
        f.append(f[3]*2 + temp)
        f.pop(0)
        cnt += 1
    return f[-1]%(10**9+7)

n = int(input())
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 $n = 1, 2, 3, 4$.
- Bước 2: Nếu $n \geq 4$ 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.

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.

```

n = [25,50,75,100,500,750,1000,2500,5000,7500,10000,25000,75000,100000,250000,750000,1000000]
def knightDialer(n):
    f = [10,20,46,104]
    if n <= 4:
        return f[n-1]
    cnt = 5
    while cnt <= n:
        temp = f[3]- f[2]*2
        if cnt % 2 != 0: temp+= f[1]
        else: temp *= 2
        f.append(f[3]*2 + temp)
        f.pop(0)
        cnt+=1
    return f[-1]%(10**9+7)
for i in n:
    print(i,knightDialer(i))

```

```

25 715338219
50 267287516
75 533889181
100 540641702
500 84202957
750 185434245
1000 88106097
2500 851996060
5000 406880451
7500 549384636
10000 796663529
25000 600978592
75000 800480549
100000 498938219
250000 287289220
750000 414048711
1000000 643304746

```

Phân tích độ phức tạp cài đặt bằng thực nghiệm

A	B	C	D	E	F	G	H	I	J	K	L	M	N
N	t	$3.43914494 \cdot \lg(n) - 32.24937551$		$0.08244813 \cdot \sqrt[n]{n} - 8.05095223$		$(9.82006242E-05) \cdot n - 3.03381703$		$(1.13551219E-10) \cdot n^2 + 0.00266263$		$(1.18778825E-16) \cdot n^3 + 1.54379044$		$(5.01191389E-06) \cdot \lg(n) - 2.62768665$	
25	0.00015	-16.27848099	264.994	-7.63871158	58.3522	-3.031362014	9.19007	0.002662701	6.31E-06	1.54379044	2.38283	-2.627104785	6.90247
50	0.00021	-12.83933605	164.854	-7.467955912	55.7735	-3.028906999	9.17555	0.002662914	6.02E-06	1.54379044	2.38264	-2.626272324	6.89841
75	0.00025	-10.82756523	117.242	-7.336930479	53.8342	-3.026451983	9.16092	0.002663269	5.82E-06	1.54379044	2.38252	-2.625345277	6.89375
100	0.00048	-9.400191113	88.3726	-7.22647093	52.2288	-3.023996968	9.14746	0.002663766	4.77E-06	1.54379044	2.38181	-2.624356806	6.88977
500	0.00041	-1.414743854	2.00266	-6.207355997	38.5364	-2.984716718	8.91098	0.002691018	5.20E-06	1.543790455	2.38202	-2.605218781	6.7893
750	0.00107	0.59702697	0.35516	-5.793017199	33.5714	-2.960166562	8.76892	0.002726503	2.74E-06	1.54379049	2.37999	-2.59178601	6.7229
1000	0.00123	2.024401086	4.09322	-5.443713434	29.6474	-2.935616406	8.62507	0.002776181	2.39E-06	1.543790559	2.37949	-2.577738997	6.65108
2500	0.00365	6.570703404	43.1262	-3.92854573	15.4622	-2.78831547	7.79507	0.003372325	7.71E-08	1.543792296	2.37204	-2.486254044	6.19962
5000	0.00585	10.00984834	100.08	-2.220989048	4.95881	-2.542813909	6.49569	0.00550141	1.22E-07	1.543805287	2.36531	-2.319761868	5.40847
7500	0.01253	12.02161917	144.218	-0.910734723	0.85242	-2.297312349	5.33537	0.009049886	1.21E-05	1.54384055	2.34491	-2.143811115	4.64981
10000	0.01735	13.44899328	180.409	0.19386077	0.03116	-2.051810788	4.28143	0.014017752	1.11E-05	1.543909219	2.33038	-1.961717948	3.91671
25000	0.0956	17.9952956	320.399	4.985241751	23.9086	-0.578801425	0.45482	0.073632142	0.00048	1.545646359	2.10263	-0.797130149	0.79697
50000	0.31568	21.43444054	446.002	10.3850101	101.391	1.87621418	2.43527	0.286540678	0.00085	1.558637793	1.54494	1.284022046	0.93769
75000	0.67456	23.44621137	518.548	14.52839808	191.929	4.331229785	13.3712	0.641388237	0.0011	1.593900257	0.84519	3.45976002	7.75734
100000	1.16751	24.87358548	561.978	18.02143573	284.055	6.78624539	31.5702	1.13817482	0.00086	1.662569265	0.24508	5.69692213	20.5156
250000	7.04601	29.4198878	500.59	33.17311277	682.625	21.51633902	209.39	7.099613818	0.00287	3.399709581	13.2955	19.84018275	163.691
500000	28.704	32.85903274	17.2643	50.24867959	464.173	46.06649507	301.456	28.39046738	0.0983	16.39114357	151.606	44.81400909	259.532
750000	63.2116	34.87080357	803.201	63.35122284	0.01949	70.61665112	54.8348	63.87522332	0.4404	51.65360724	133.587	70.73369322	56.5819
1000000	113.852	36.29817768	6014.53	74.39717777	1556.65	95.16680717	349.12	113.5538816	0.08862	120.3226154	41.8744	97.26761871	275.027
MSE			541.698		192		55.2379		0.03334		19.5361		44.8822

Sau khi phân tích độ phức tạp bằng thực nghiệm 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.

Source code mẫu sau khi đã tối ưu:

```
def knightDialer(n):
    if n == 1:
        return 10
    graph = {0:[4, 6], 1:[6, 8], 2:[7, 9], 3:[4, 8], 4:[0, 3, 9], 6:[0, 1, 7], 7:[2, 6], 8:[1, 3], 9:[2, 4]}
    cnt = {0:1, 1:1, 2:1, 3:1, 4:1, 6:1, 7:1, 8:1, 9:1}
    for i in range(n - 1):
        tmp = {0:0, 1:0, 2:0, 3:0, 4:0, 6:0, 7:0, 8:0, 9:0}
        for k in cnt:
            for v in graph[k]:
                tmp[v] = (tmp[v] + cnt[k]) % (10**9 + 7)
        cnt = tmp
    return sum(cnt.values()) % (10**9 + 7)

n = int(input())
print(knightDialer(n))
```

Bảng thực nghiệm độ phức tạp:

A	B	C	D	E	F	G	H	I	J	K	L	M	N
N	t	$0.37562518 \cdot \lg(n) - 3.41947518$		$0.00819578 \cdot \sqrt{n} - 0.58636561$		$[9.17431942 \cdot 10^{-6}] \cdot n - 0.00177538$		$[9.89584508 \cdot 10^{-12}] \cdot n^2 + 0.35293736$		$[9.95347464 \cdot 10^{-18}] \cdot n^3 + 0.52000358$		$[4.64898369 \cdot 10^{-07}] \cdot n \cdot \lg(n) + 0.04545163$	
25	0	-1.675125863	2.80605	-0.54538671	0.29745	-0.001546022	2.3902E-06	0.352937366	0.12456	0.52000358	0.2704	0.045505603	0.00207
50	0	-1.299500683	1.6887	-0.528412694	0.27922	-0.001316664	1.7336E-06	0.352937385	0.12456	0.52000358	0.2704	0.045582821	0.00208
75	0.001	-1.079774038	1.16807	-0.515388073	0.26666	-0.001087306	4.3568E-06	0.352937416	0.12386	0.52000358	0.26936	0.045668813	0.002
100	0.001	-0.923875503	0.85539	-0.50440781	0.25544	-0.000857948	3.452E-06	0.352937459	0.12386	0.52000358	0.26936	0.045760502	0.002
500	0.004	-0.051700844	0.0031	-0.403102398	0.16573	0.00281178	1.4119E-06	0.352939834	0.12176	0.520003581	0.26626	0.047535719	0.0019
750	0.007	0.1680258	0.02593	-0.361914931	0.1361	0.00510536	3.5897E-06	0.352942926	0.11968	0.520003584	0.26317	0.048781725	0.00175
1000	0.009	0.323924336	0.09918	-0.32719229	0.11303	0.007398939	2.5634E-06	0.352947256	0.1183	0.52000359	0.26112	0.050084707	0.00169
2500	0.024	0.820473814	0.63437	-0.17657661	0.04023	0.021160419	8.0632E-06	0.352999209	0.10824	0.520003736	0.24602	0.058570728	0.0012
5000	0.046	1.196098994	1.32273	-0.006836448	0.00279	0.044096217	3.6244E-06	0.353184756	0.09436	0.520004824	0.22468	0.074014317	0.00078
7500	0.07	1.415825639	1.81125	0.123409758	0.00285	0.067032016	8.8089E-06	0.353494001	0.08037	0.520007779	0.20251	0.090335272	0.00041
10000	0.09201	1.571724174	2.18955	0.23321239	0.01994	0.089967814	4.1705E-06	0.353926945	0.0686	0.520013533	0.18319	0.107225988	0.00023
25000	0.22702	2.068273653	3.39022	0.70950099	0.23279	0.227582606	3.1652E-07	0.359122263	0.01745	0.520159103	0.08593	0.215251581	0.00014
50000	0.45903	2.443898833	3.9397	1.246266511	0.61974	0.456940591	4.3656E-06	0.377676973	0.00662	0.521247764	0.00387	0.40829645	0.00257
75000	0.68507	2.663625478	3.91468	1.658141181	0.94687	0.686298577	1.5094E-06	0.408601489	0.07643	0.524202702	0.02588	0.610114968	0.00562
100000	0.91607	2.819524013	3.62314	2.00536759	1.18657	0.915656562	1.7093E-07	0.451895811	0.21546	0.529957055	0.14908	0.817631107	0.00969
250000	2.27717	3.316073492	1.07932	3.51152439	1.52363	2.291804475	0.00021417	0.971427678	1.70496	0.675526621	2.56526	2.129540875	0.02179
500000	4.56335	3.691698672	0.75978	5.208926005	0.41677	4.58538433	0.00048551	2.82689863	3.01526	1.76418791	7.83531	4.466079305	0.01375
750000	6.87952	3.911425316	8.80959	6.511388074	0.13552	6.878964185	3.0893E-07	5.919350218	0.92193	4.719125694	4.6673	6.850354227	0.00085
1000000	9.1867	4.067323852	26.208	7.60941439	2.48783	9.17254404	0.00020039	10.24878244	1.12802	10.47347822	1.6558	9.311605349	0.0156
MSE			2.11782		0.36896		4.1695E-05		0.39813		1.00328		0.00392

Source code của nhóm sau khi đã tối ưu:

```
def knightDialer(n):
    f = [10, 20, 46, 104]
    if n <= 4:
        return f[n-1]
    cnt = 5
    while cnt <= n:
        temp = f[3] - f[2]*2
        if cnt % 2 != 0: temp += f[1]
        else: temp *= 2
        f.append((f[3]*2 + temp)%(10**9+7))
        f.pop(0)
        cnt += 1
    return f[-1]%(10**9+7)

n = int(input())
print(knightDialer(n))
```

Bảng thực nghiệm độ phức tạp:

A	B	C	D	E	F	G	H	I	J	K	L	M	N
N	t	0.04817743*lg(n)- 0.43840558		0.00104942*sqrt(n)-0.07461928		(1.173288E- 06)*n+0.0004438 2		(1.26362835E- 12)*n^2+0.04600 024		(1.26970684E- 18)*n^3+0.06743 88		(5.94463344E- 08)*nlg(n)+0.0065 0788	
25	0	-0.214676523	0.04608601	-0.06937218	0.004812499	0.000473152	2.23873E-07	0.046000241	0.002116	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	0.002116	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	0.002116	0.0674388	0.004547992	0.006535651	4.27547E-05
100	0	-0.118321663	0.014000016	-0.06412508	0.004112026	0.000561149	3.14888E-07	0.046000253	0.002116	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	0.002025	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	0.002025	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	0.002025	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	0.00185	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	0.001603	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	0.001449	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	0.001234	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	0.000316	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	0.000284	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	0.001149	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	0.003407	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	0.028915	0.087277969	0.043156751	0.272999385	0.000484907
500000	0.58806	0.47366874	0.01308536	0.667412718	0.006300028	0.58708782	9.45134E-07	0.361907328	0.051145	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	0.0162	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	0.019477	1.33714564	0.027914269	1.19136657	0.000453544
MSE			0.034789455		0.005922648		3.96508E-06		0.006783		0.016852064		9.02627E-05

→ 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.