Miscelánea Matemática **54** (2012) 25–34 SMM

El problema de Langford

Erick Andrey Serratos Álvarez

prop@xanum.uam.mx

Universidad Autónoma Metropolitana
Posgrado en Ciencias y Tecnologías de la Información.

Resumen

En este trabajo se aborda el problema de Langford, se describe formalmente el problema y su generalización, posteriormente se analiza cuando es que existen soluciones y finalmente se muestra un programa codificado en el lenguaje de programación *prolog*, usando el algoritmo de búsqueda en profundidad para encontrar soluciones.

1. Introducción

En la década de 1950 [**6**], el matemático escocés C. Dudley Langford descubrió un problema muy curioso e interesante mientras observaba a su hijo que jugaba con cubos de colores. A Langford le llamó mucho la atención la configuración que el niño les había dado. Notó que había tres cubos entre un par de cubos amarillos, dos cubos entre un par de color azul y un cubo entre otro par de color rojo [**2**]. El arreglo es el siguiente:

Amarillo	Rojo	Azul	Rojo	Amarillo	Azul

Figura 1: Instancia del problema de Langford

2. El problema de Langford

Asignando un número a cada color, podemos transformar esta casualidad en un problema de ordenamiento. En este caso el problema consiste en arreglar pares de números del 1 al 3 tal que entre el par de 3's haya 3 números, entre el par de 2's haya 2 números y entre el par de 1's haya 1 número. La solución se muestra en la figura 2:

Amarillo	Rojo	Azul	Rojo	Amarillo	Azul
3	1	2	1	3	2

Figura 2: Instancia con tres dígitos del problema de Langford

Si se hace un pequeño ejercicio, podemos percatarnos que, para este caso, sólo existe una sucesión sin tomar en cuenta la sucesión simétrica es decir: si tenemos la sucesión $(3, 1, 2, 1, 3, 2)$, la sucesión simétrica es $(2, 3, 1, 2, 1, 3)$. Podemos preguntarnos ahora si sería posible encontrar una configuración similar pero agregando un par de cubos con un color distinto a los ya puestos. Hallaremos que se cuenta con sólo una sucesión que cumpla con estas características, además de la simétrica. Observe la siguiente figura:

Verde	Rojo	Amarillo	Rojo	Azul	Verde	Amarillo	Azul
4	1	3	1	2	4	3	2

Figura 3: Instancia con cuatro dígitos del problema de Langford

2.1. Descripción de $L(2, 4)$

A manera de introducción a la formalización del problema de Langford, a continuación se describe en lenguaje natural la instancia $L(2, 4)$.

Consideremos la solución de $L(2, 4)$ a la que denotaremos como una cadena (o arreglo) cuyos miembros están ordenados de tal manera que cumplen con las siguientes restricciones:

1. Los miembros del arreglo son exclusivamente números naturales.
2. Los números están entre el 1 y el 4, inclusive.
3. Cada número se repite exactamente 2 veces.
4. Entre el par de números k hay exactamente k números para todo $k = 1, 2, 3, 4$.

2.2. Descripción formal de $L(2, n)$

Sea $k, n \in \mathbb{N}$, con $1 \leq k \leq n$ y $n \geq 3$. Una cadena Langford se define como sigue [**4**]:

$$L(2, n) = (k_1, k_2, k_3, \ldots, k_{2n})$$

tal que:

1. $1 \leq k_i \leq n$,
2. la longitud de $L(2, n)$ es $2n$,
3. el elemento k_i ocurre exactamente 2 veces y
4. si $k_i = k_j$, entonces $j - i = k_i + 1$

2.3. Descripción formal de $L(m, n)$

Ahora que conocemos más el problema, podemos pensar en que los elementos de una cadena de Langford ¡pueden ocurrir más de dos veces! El problema de Langford ha sido generalizado para cualquier cantidad de números y cualquier cantidad de repeticiones de los números [**3**]. Observe el siguiente ejemplo:

$$L(3, 9) = (3, 4, 7, 9, 3, 6, 4, 8, 3, 5, 7, 4, 6, 9, 2, 5, 8, 2, 7, 6, 2, 5, 1, 9, 1, 8, 1)$$

La descripción de $L(m, n)$ será muy similar a la de $L(2, n)$. Veamos.

Sea $k, m, n \in \mathbb{N}$, con $1 \leq k \leq n$, $m \geq 2$ y $n \geq 3$. Una cadena Langford se define como sigue:

$$L(m, n) = (k_1, k_2, k_3, \ldots, k_{mn})$$

tal que:

1. $1 \leq k_i \leq n$,
2. la longitud de $L(m, n)$ es mn,
3. el elemento k_i ocurre exactamente m veces y
4. si $k_i = k_j$, entonces $j - i = k_i + 1$.

Dicho esto, ahora podemos preguntarnos ¿Para qué valores de n y m es posible encontrar solución(es)? Y más aún complicado que eso, podemos preguntarnos: ¿Cuántas soluciones diferentes hay (sin contar las soluciones simétricas)?

Resulta interesante saber que $L(2, 3)$ y $L(2, 4)$ tienen sólo una solución (sin contar la solución simétrica). Resulta aún más interesante que para $L(2, 5)$ y $L(2, 6)$ no hay solución, mismo que se puede verificar con un ejercicio y algo de tiempo. Sin embargo $L(2, 7)$ y $L(2, 8)$ tienen 26 y 150 soluciones respectivamente [**6**, **4**].

Aunque se ha descrito $L(m,n)$, en el resto del documento sólo se trata $L(2,n)$. A continuación se explicará cuándo existe solución para cadenas de Langford del tipo $L(2,n)$.

3. Existencia de soluciones para $L(2,n)$

Roy O. Davies [**1**] argumenta cuándo es posible encontrar soluciones. El argumento dado se basa en la observación que, dado un número k en el arreglo, si la primer ocurrencia de k está en la posición p_k , entonces el siguiente está en la posición p_k+k+1, para $k = 1, 2, \ldots, n$. Observemos lo siguiente:

$$\{p_k, p_k + k + 1 \mid k = 1, 2, \ldots, n\} = \{1, 2, \ldots, 2n\},$$

es decir, p_k y $p_k + k + 1$ para $k = 1, 2, \ldots, n$ son los mismos números del arreglo $(1, 2, \ldots, 2n)$ que representan las posiciones en la cadena de Langford. Entonces,

$$\sum_{k=1}^{n} p_k + \sum_{k=1}^{n} (p_k + k + 1) = \sum_{k=1}^{2n} k,$$

luego

$$\sum_{k=1}^{n} (2p_k + k + 1) = \frac{2n(2n+1)}{2}.$$

Distribuyendo la suma anterior obtenemos:

$$\sum_{k=1}^{n} 2p_k + \sum_{k=1}^{n} k + n = n(2n+1),$$

y despejando la primer suma, resulta

$$2\sum_{k=1}^{n} p_k = \frac{3n^2 - n}{2}.$$

Factorizando el lado derecho de la igualdad anterior y sabiendo que el lado izquierdo de la misma igualdad debe ser un número entero, conseguimos:

$$\sum_{k=1}^{n} p_k = \frac{n(3n-1)}{4}.$$

Sólo bastaría encontrar los valores de n para los cuales el resultado es un número entero positivo y esto ocurre si y sólo si n es de la forma $4r$ o $4r-1$, donde r es un número entero positivo.

A continuación se hace referencia a algunos trabajos realizados sobre el problema de Langford y se presenta un algoritmo exhaustivo para encontrar soluciones.

3.1. Resultados previos

Sobre el problema de Langford ya se han realizado estudios y se encuentran publicados trabajos importantes, tal como el método algebraico de Godfrey para encontrar soluciones, este método es tratado con más detalle en [**3**]. En [**3**] también se propone la resolución del problema de Langford de dos maneras, a saber: la primera lista las soluciones mediante un programa escrito en lenguaje *C* y la biblioteca *OpenMP*, usando un algoritmo secuencial paralelizado y memoria compartida; la segunda lista las soluciones mediante un programa también escrito en lenguaje *C* pero con la biblioteca *MPI*, éste usa el método algebraico de Godfrey. En [**4**] se presentan otros dos programas que encuentran soluciones al problema de Langford.

Para tener una noción de lo complejo que es el problema de Langford, se presentan los siguientes datos. En [**2**] es mencionado que le tomó dos años y medio a una computadora, cuyo procesador era de 300MHz, conocer $L(2,19)$ en el año 1999. En cambio en el segundo trabajo presentado en [**3**] en el año 2004, se reporta que para resolver $L(2,19)$ tomó entre 149 y 162 minutos para encontrar las soluciones usando 8 procesadores; y entre 18 y 34 minutos usando 64 procesadores cuya frecuencia no es mencionada.

Por otro lado, el primer programa presentado en [**3**] tardó 40 minutos en encontrar $L(2,20)$, usando 128 procesadores. El segundo programa presentado en [**3**] tomó casi 38 minutos en resolver $L(2,20)$, usando 128 procesadores.

Hasta el momento, según las referencias consultadas, se conoce cual es el número de soluciones de $L(2,24)$, los siguientes casos abiertos son $L(2,27)$ y $L(2,28)$. En la siguiente tabla se muestran el número de soluciones para $L(2,3)$ y hasta $L(2,24)$. Para conocer esta tabla en detalle consulte [**6**].

3.2. Encontrando soluciones para $L(2,n)$

A continuación se propone un algoritmo -exhaustivo- que fue codificado en el lenguaje de programación *prolog*, utilizando el algoritmo de

n	$L(2,n)$
3	1
4	1
7	26
8	150
11	17,792
12	108,144
15	39,809,640
16	326,721,800
19	256,814,891,280
20	2,636,337,861,200
23	3,799,455,942,515,488
24	46,845,158,056,515,936

Cuadro 1: Número de soluciones para $L(2,n)$

búsqueda en profundidad para encontrar soluciones de $L(2,n)$.

En un algoritmo de búsqueda debemos definir un estado inicial, alguna manera de pasar de un estado a otro y un estado meta [**5**]. Definiremos a continuación estos elementos para nuestro problema.

3.2.1. Estado inicial

La cadena "vacía" $[x,x,x,x,x,x]$ representa nuestro estado inicial.

3.2.2. Conjunto de estados

Si se tiene la cadena $[x,x,x,x,x,x]$ los posibles estados siguientes son: $[1,x,1,x,x,x]$, $[x,1,x,1,x,x]$, $[x,x,1,x,1,x]$, $[x,x,x,1,x,1]$, $[2,x,x,2,x,x]$, $[x,2,x,x,2,x]$, $[x,x,2,x,x,2]$, $[3,x,x,x,3,x]$ y $[x,3,x,x,x,3]$. En cambio si se tiene la cadena $[3,x,x,x,3,x]$, los posibles estados siguientes son: $[3,1,x,1,3,x]$, $[3,x,x,1,3,1]$ y $[3,x,2,x,3,2]$.

3.2.3. Estado meta

Aquella cadena que cumpla las restricciones mencionadas en la sección 2.2. Por ejemplo $[3,1,2,1,3,2]$ o $[2,3,1,2,1,3]$ (la solución simétrica) son estados meta.

3.2.4. Algoritmo exhaustivo para encontrar soluciones a $L(2,n)$

Para implementar un algoritmo que genere los estados, se propone declarar algunos predicados (en el lenguaje de programación *prolog*) que

hagan lo siguiente:

1. *Recibir* un arreglo y encontrar la longitud de éste.
2. Generar una lista $[1, 2, 3, \ldots, l]$, donde l es la longitud del arreglo recibido.
3. De la lista generada, encontrar todos los números que no son miembros del arreglo recibido.
4. Insertar todos los números no-miembros en lugares no ocupados del arreglo recibido.
5. Insertar el mismo número no-miembro en la siguiente posición, según las restricciones de la sección 2.2 de este documento.

Cabe mencionar que esta implementación es meramente didáctica. Por otro lado, no se tomaron tiempos de ejecución de esta implementación debido a la forma en que opera el intérprete de *prolog*. A continuación se muestra la implementación de los predicados enunciados en esta sección.

3.2.5. Código fuente del algoritmo presentado

```
%%%%%%%%%%%%%%%%%%%%%%%%
% Archivo: langdf.pl    %
% Autor: Erick Serratos %
%%%%%%%%%%%%%%%%%%%%%%%%

%Esto es un comentario

% Este predicado genera una lista que contenga los d{\'\i}gitos del N al M (N <= M)
genera(N, N, [N]):- !.
genera(N, M, [N|Resto]):- N<M,
Aux is N+1,
genera(Aux, M, Resto).

% Este predicado dice si la posici\'on Pi est\'A ocupada en la lista N|NS
esta_ocupada(Pi,[N|NS]):- Pi == 1,
N \= x;
Pf is Pi-1,
esta_ocupada(Pf, NS).

% Este predicado inserta un numero X en la posici\'on P de la lista L|LS
inserta(X,1,[L|LS],[X|LS]):- !.
inserta(X,P,[L|LS],[L|ZS]):-PAux is P-1,
inserta(X, PAux, LS, ZS).

% Ahora, se debe:
% Encontrar la longitud de la lista inicial L y ponerlo en la variable Longitud
% Generar una lista del tipo [1, 2, 3, ..., Longitud]
% Encontrar todos los n\'umeros que no son miembros (NM) de la lista actual L
% Insertar todos los no miembros (NM) en lugares no ocupados en L
% Insertar otro no miembro en la posici\'on donde que le corresponde
```

```
% Poner el estado siguiente en la lista S

siguiente_estado(L, S):-length(L, Longitud),
genera(1, Longitud, LAux1),
es_miembro(Pi, LAux1),
es_miembro(NM, LAux1),
not(es_miembro(NM, L)),
not(esta_ocupada(Pi, L)),
NM =< Longitud /2,
inserta(NM, Pi, L, LAux2),
Pf is Pi+NM+1,
not(esta_ocupada(Pf, LAux2)),
inserta(NM, Pf, LAux2, S).


%Dado el m\'etodo que utilizamos, basta que el caracter 'x'
%no sea miembro de nuestra lista para que generemos una soluci\'on.

%Dice si se llego al estado meta
meta(N):-es_miembro(X,N),
X \= x.
```

A continuación se muestra la implementación del algoritmo de búsqueda en profundidad, dicho código utiliza al anterior para obtener el siguiente estado y así encontrar una meta.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Autor: Dr. Ren\'e Mac Kinney Romero %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Este codigo encuentra soluciones con el algoritmo de b\'usqueda en profundidad
solucion(N, [N], R):- meta(N).

solucion (N, [N|Solucion1], R):-not(member(N, R)),
siguiente_estado(N, N1),
solucion(N1, Solucion1, [N|R]).

solucion (N, S):-solucion(N, S, []).


%Dice si X es miembro de la lista
es_miembro(X, [X|Xs]).
es_miembro(X, [Y|Ys]):- es_miembro(X, Ys).

%fin de archivo
```

4. Resultados

A continuación se presentan los resultados de la ejecución del programa para :

```
   ?- sol([x,x,x,x,x,x],L).
L = [[x,x,x,x,x,x],[1,x,1,x,x,x]] ? ;
L = [[x,x,x,x,x,x],[1,x,1,x,x,x],[1,2,1,x,2,x]] ? ;
L = [[x,x,x,x,x,x],[1,x,1,x,x,x],[1,3,1,x,x,3]] ? ;
L = [[x,x,x,x,x,x],[2,x,x,2,x,x]] ? ;
```

```
L = [[x,x,x,x,x,x],[2,x,x,2,x,x],[2,3,x,2,x,3]] ? ;
L = [[x,x,x,x,x,x],[2,x,x,2,x,x],[2,3,x,2,x,3],[2,3,1,2,1,3]] ? ;
L = [[x,x,x,x,x,x],[2,x,x,2,x,x],[2,x,1,2,1,x]] ? ;
L = [[x,x,x,x,x,x],[2,x,x,2,x,x],[2,x,1,2,1,x],[2,3,1,2,1,3]] ? ;
L = [[x,x,x,x,x,x],[3,x,x,x,3,x]] ? ;
L = [[x,x,x,x,x,x],[3,x,x,x,3,x],[3,1,x,1,3,x]] ? ;
L = [[x,x,x,x,x,x],[3,x,x,x,3,x],[3,1,x,1,3,x],[3,1,2,1,3,2]] ? ;
L = [[x,x,x,x,x,x],[3,x,x,x,3,x],[3,x,2,x,3,2]] ? ;
L = [[x,x,x,x,x,x],[3,x,x,x,3,x],[3,x,2,x,3,2],[3,1,2,1,3,2]] ? ;
L = [[x,x,x,x,x,x],[3,x,x,x,3,x],[3,x,x,1,3,1]] ? ;
L = [[x,x,x,x,x,x],[x,1,x,1,x,x]] ? ;
L = [[x,x,x,x,x,x],[x,1,x,1,x,x],[3,1,x,1,3,x]] ? ;
L = [[x,x,x,x,x,x],[x,1,x,1,x,x],[3,1,x,1,3,x],[3,1,2,1,3,2]] ? ;
L = [[x,x,x,x,x,x],[x,1,x,1,x,x],[x,1,2,1,x,2]] ? ;
L = [[x,x,x,x,x,x],[x,1,x,1,x,x],[x,1,2,1,x,2],[3,1,2,1,3,2]] ? ;
L = [[x,x,x,x,x,x],[x,2,x,x,2,x]] ? ;
L = [[x,x,x,x,x,x],[x,2,x,x,2,x],[1,2,1,x,2,x]] ? ;
L = [[x,x,x,x,x,x],[x,2,x,x,2,x],[x,2,x,1,2,1]] ? ;
L = [[x,x,x,x,x,x],[x,3,x,x,x,3]] ? ;
L = [[x,x,x,x,x,x],[x,3,x,x,x,3],[1,3,1,x,x,3]] ? ;
L = [[x,x,x,x,x,x],[x,3,x,x,x,3],[2,3,x,2,x,3]] ? ;
L = [[x,x,x,x,x,x],[x,3,x,x,x,3],[2,3,x,2,x,3],[2,3,1,2,1,3]] ? ;
L = [[x,x,x,x,x,x],[x,3,x,x,x,3],[x,3,1,x,1,3]] ? ;
L = [[x,x,x,x,x,x],[x,3,x,x,x,3],[x,3,1,x,1,3],[2,3,1,2,1,3]] ? ;
L = [[x,x,x,x,x,x],[x,x,1,x,1,x]] ? ;
L = [[x,x,x,x,x,x],[x,x,1,x,1,x],[2,x,1,2,1,x]] ? ;
L = [[x,x,x,x,x,x],[x,x,1,x,1,x],[2,x,1,2,1,x],[2,3,1,2,1,3]] ? ;
L = [[x,x,x,x,x,x],[x,x,1,x,1,x],[x,3,1,x,1,3]] ? ;
L = [[x,x,x,x,x,x],[x,x,1,x,1,x],[x,3,1,x,1,3],[2,3,1,2,1,3]] ? ;
L = [[x,x,x,x,x,x],[x,x,2,x,x,2]] ? ;
L = [[x,x,x,x,x,x],[x,x,2,x,x,2],[3,x,2,x,3,2]] ? ;
L = [[x,x,x,x,x,x],[x,x,2,x,x,2],[3,x,2,x,3,2],[3,1,2,1,3,2]] ? ;
L = [[x,x,x,x,x,x],[x,x,2,x,x,2],[x,1,2,1,x,2]] ? ;
L = [[x,x,x,x,x,x],[x,x,2,x,x,2],[x,1,2,1,x,2],[3,1,2,1,3,2]] ? ;
L = [[x,x,x,x,x,x],[x,x,x,1,x,1]] ? ;
L = [[x,x,x,x,x,x],[x,x,x,1,x,1],[3,x,x,1,3,1]] ? ;
L = [[x,x,x,x,x,x],[x,x,x,1,x,1],[x,2,x,1,2,1]] ? ;
```

En estos resultados podemos notar que la primera columna es el nodo raíz del Árbol de búsqueda generado por el intérprete de *prolog* y la última columna son las hojas que representan una solución a nuestro problema.

5. Trabajo a futuro

Se puede optimizar el código pues se genera una solución y la solución simétrica, ambas en más de una ocasión.

El lenguaje de programación *prolog*, soporta el uso de *sockets*, por lo que, si se desea, se puede hacer una implementación de una interfaz gráfica en otro lenguaje de programación y comunicar ambos programas a través de *sockets*.

6. Agradecimientos

Agradezco a las siguientes personas por sus valiosas sugerencias, ayuda y estímulo para la realización de este trabajo de divulgación.

Dr. Miguel Ángel Pizaña López
Dr. Luis Franco Pérez
Dr. René Mac Kinney Romero
Mtro. Jesús A. Torres ChÁzaro
Mtro. Alan Gustavo Lazalde Cruz
Lic. Araceli SÁnchez Balbuena
Dr. Sergio G. De los Cobos Silva
Dr. Bernardo Llano Pérez

Bibliografía

1. R. O. Davies, On langford's problem (2), *The Mathematical Gazette* **43** (1959) 253–255.
2. Z. Habbas, M. Krajecki, y D. Singer, The Langford's problem: a challenge for parallel resolution of csp, tomo 2328, Parallel Processing and Applied Mathematics, no. 3, 2002, 789–796.
3. C. Jaillet y M. Krajecki, Solving the Langford problem in parallel, Proceedings of Third International Symposium on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004, 83–90.
4. E. Kalvelagen, *Langford's problem, http://www.amsterdamoptimization.com/pdf/langford.pdf.*
5. J. Larrosa y P. Meseguer, Restricciones blandas: modelos y algoritmos, *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial* **20** (2003) 69–82.
6. J. E. Miller, *Langford's Problem*, http://legacy.lclark.edu/ miller/langford.html.