

0. Considere una máquina $M = (\Sigma, Q, \delta, q_0, q_f)$ con alfabeto es $\Sigma = \{0,1\}$, conjunto de estados $Q = \{q_0, q_1, q_2, q_3, q_f\}$ y la regla de transición es:

δ	0	1
q_0	$0Rq_1$	—
q_1	$0Rq_2$	$1Rq_1$
q_2	$0Rq_f$	$1Lq_3$
q_3	$1Rq_4$	—
q_4	$0Lq_5$	$1Rq_4$
q_5	—	$0Rq_f$
q_f	—	—

- a. ¿La máquina siempre llega a un estado final para alguna entrada $w = 01^*01^*0$?
Sí, siempre dejando dos ceros juntos al final de la cadena, uno al inicio y en el medio 1's.
- b. Si se detiene, escribir en pseudocódigo el algoritmo respectivo.

```

if ( cantCeros(n) == 3 ){
    if ( n > 0 && ~n&1 == 1 ){
        int x = 2;
        while ( ~n&x == 0 )
            x <<= 1;
        n ^= x^2;
    }
}

```

- c. ¿Cuántas operaciones le toma al autómata realizar la concatenación, $w \rightarrow w'$ (i.e. $w = 0w_10w_20$, $w' = 0w_1w_20$, $w_1 = 1^*$, $w_2 = 1^*$)

$$|w| = N$$

$$|w_1| = T$$

$$|w_2| = U$$

Contemplando que cada que hace una transición hace mínimo 3 operaciones:

- Cambio de carácter
- Avanzar o retroceder
- Cambio de estado

$$\text{Total} = 3 (1 + T + 2 + 1 + U + 2)$$

1 hace referencia a que lee el primer 0

T Lectura de la cadena completa de w_1

2 Lectura del segundo 0 y lectura del primer uno

1 Regreso al segundo 0

U Lectura de la cadena completa de w_2

2 Lectura del tercer 0 y regreso para el cambio del ultimo 1

Todo lo anterior multiplicado por 3 que son las operaciones que hace por cada transición.

d. ¿Cuántas unidades de memoria utiliza?

$$|w_1| = T$$

$$|w_2| = U$$

$$|w| = N = T + U + 3$$

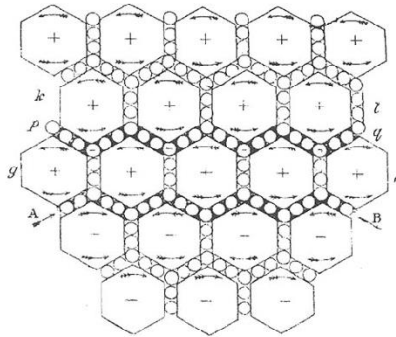
El 3 representa la cantidad de 0's.

Unidades de memoria = N enteros de la cadena de entrada + 3 x (matriz de 7 x 2) = 42 enteros + N enteros.

Un entero usa 32 bits (4 bytes), entonces mínimo se estaría ocupando 168 bytes.

e. Implemente el algoritmo en un sistema/fenómeno natural no convencional (p. ejemplo granos de arena, copos de nieve, hormigas, mariposas, abejas, corcholatas, frijolitos, etc.)

1. Describa las Ecuaciones de Maxwell de electromagnetismo como una maquinaria de ruedas y engranes y explique el efecto físico de cada una de las 4 ecuaciones.



Ley de Gauss para el campo eléctrico: En esta fórmula se expresa que la suma de las líneas de fuerza dentro de un área definida nos dará una equivalencia a la magnitud de la carga.

Como tal, el campo eléctrico es el que tiene influencia sobre otras cargas y actuarán atrayéndolas o repelándolas, formando así un flujo de corriente eléctrica.

Es importante decir que las líneas de fuerza pueden ir hacia afuera (se consideraran positivas) o hacia adentro (son negativas).

$$\oint \mathbf{E} \cdot d\mathbf{A} = \frac{q_{enc}}{\epsilon_0}$$

Ley de Gauss para el campo magnético: Recordar, que los polos magnéticos no existen como monopolos, siempre va acompañado un polo positivo de uno negativo.

Sus líneas de fuerza que emergen del polo positivo se suman en su mismo polo negativo, por lo que si sumamos las líneas de fuerza dentro de un área determinada nos dará 0.

$$\oint \mathbf{B} \cdot d\mathbf{A} = 0$$

Ley Faraday: Una corriente eléctrica activa un campo magnético, que se posa alrededor de la corriente.

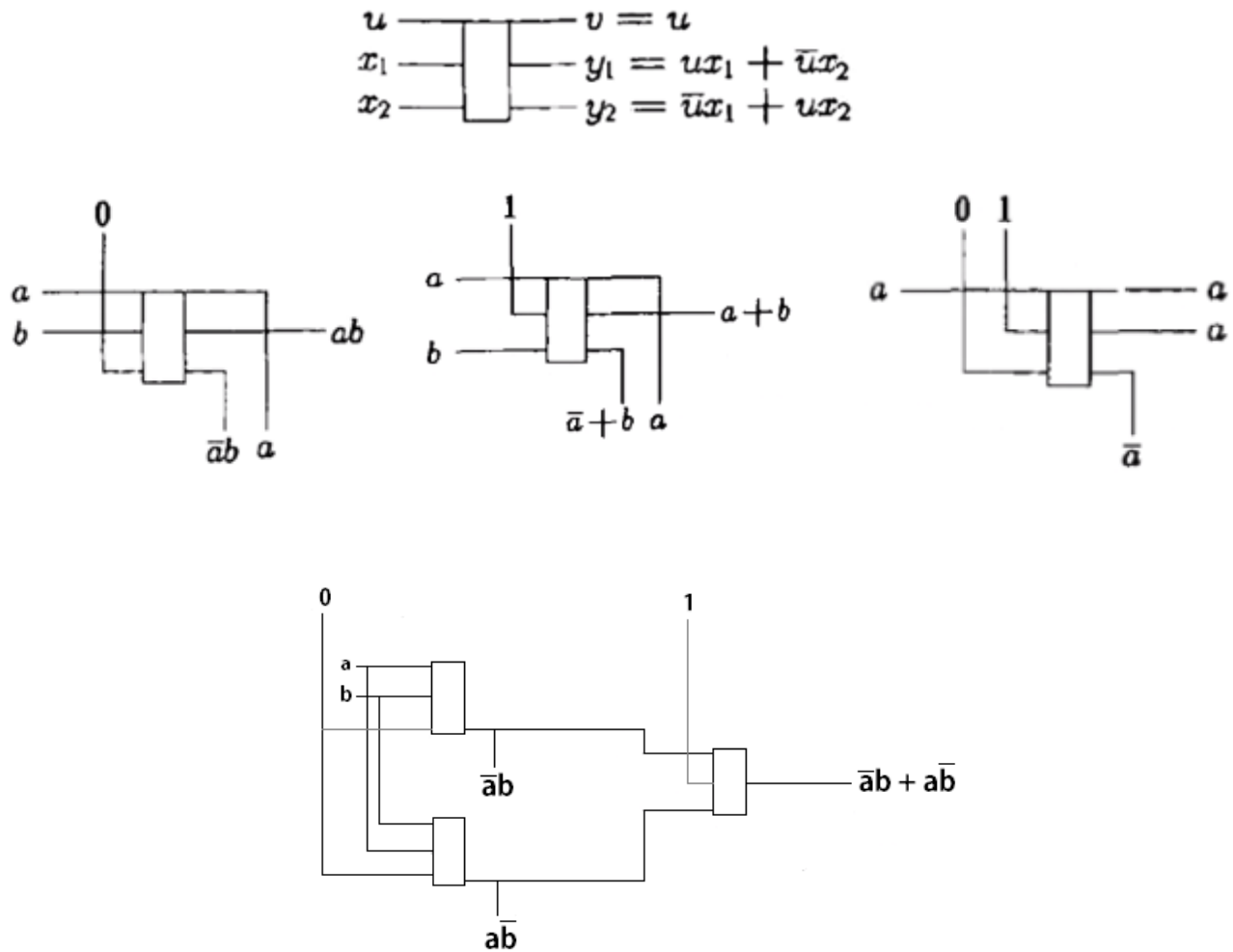
$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$

Ley Ampere: Un campo magnético que cambia con el tiempo activará un campo eléctrico rotacional.

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \mu_0 \epsilon_0 \frac{\partial \mathbf{E}}{\partial t}$$

- Reconfigure la compuerta conservativa de *Fredkin-Toffoli* para obtener las funciones lógicas correspondientes a las compuertas *AND*, *OR* y *NOT*. Utilizando estas compuertas, construya un circuito para obtener la función *XOR*.

Las siguientes configuraciones son tomadas del libro de Fredkin-Toffoli:



3. Demostrar que el algoritmo de formación de parejas de Gale-Shapley entrega una solución perfecta y estable. **NO HAY ELEMENTOS SIN PAREJA Y NO HAY INESTABILIDADES.**

Enunciado (Consideraciones del algoritmo):

Dados dos conjuntos de elementos X & Y , de forma que:

1. $|X| = |Y| = N$
2. Razón por las que se rechaza: y ya tiene pareja & es de mayor prioridad que x .
3. Hay que considerar que en todo momento cada x o cada y que tenga pareja, este será uno y solo uno.

Demostración por contradicción:

Por el algoritmo, en algún momento se dará el caso de que algún x este buscando en su lista de preferencias algún y para relacionarse.

Supongamos que llega al final de su lista & este último elemento también tiene pareja. Por lo que hay N parejas.

Entonces $|Y| = N$ & $|X| = N + 1$

Por lo que $|Y| \neq |X|$.

q.e.d. por contradicción.

Demostración por Inferencia lógica (Modus Ponens):

Para cada elemento del conjunto X se tiene una pareja y entonces, por el *enunciado* dado, y es de prioridad para x .

Cada elemento de X tiene una pareja y .

Por lo tanto, y es de prioridad para x .

Y como ya fue demostrado que todos los elementos tienen pareja, todas las parejas serán estables.

q.e.d

4. Contar el número exacto de instrucciones que se ejecutan, así como la complejidad temporal en el mejor y peor de los casos, para cada uno de los siguientes ciclos:

```
for (k=N-1; k>=0; k--)
    arr[k]= k;
```

$$\begin{aligned}
 T_{for} &= T_{ini} + (N+1)T_{comparación} + NT_{actualización} + NT_{L1} \\
 T_{for} &= 1 + (N+1)(1) + N(1) + NT_{L1} \\
 T_{L1} &= 1 \text{ (asignación)} \\
 T_{for} &= 1 + (N+1)(1) + N(1) + N(1) \\
 T_{for} &= 1 + N + 1 + N + N \\
 T_{for} &= 3N + 2 \\
 R &= \Omega(1), O(N)
 \end{aligned}$$

```
for (k=0; k<N; k+=2)
    if (k==N/2)
        k=1;
```

Pruebas:

$N=22$.

$K=0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20$ (22)
 $it=0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ (11)

Se rompe

$N=20$

$K=0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20$
 $it=0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$

Entra al if y ejecutará:

$K=1, 3, 5, 7, 9, 11, 13, 15, 17, 19$ (21)
 $it=0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ (10)

Se rompe

Por las pruebas con $N=22$ y $N=20$ vemos que hay dos formas en las que se puede ejecutar el código:

No entra al condicional

$$T_{for} = T_{ini} + (M+1)T_{comparación} + MT_{actualización} + MT_{if}$$

$$M = \frac{N}{2}$$

$$= 1 + \left(\frac{N}{2} + 1\right)(1) + \left(\frac{N}{2}\right)(2) + \frac{N}{2}(1)$$

// El T_{if} es 1, ya que en este caso solo estará haciendo la // comparación

$$= 1 + \frac{N}{2} + 1 + N + \frac{N}{2} = 2N + 2$$

$$R = \Omega(1), O(N)$$

Si entra al condicional

Solamente calculamos las primeras iteraciones que hacen que entre al condicional.

$$M = \frac{N}{4}$$

$$= 1 + \left(\frac{N}{4} + 1\right)(1) + \left(\frac{N}{4}\right)(2) + \frac{N}{4}(1)$$

$$= 1 + \frac{N}{4} + \frac{N}{2} + \frac{N}{4} = N + 1$$

Si le sumamos el caso anterior, es decir, cuando ya no entra:

$$R = \Omega(1), O(N) \quad N+1 + 2N+2 = 3N+3$$

```
for (q=0; q<N; q+=7)
{
    c= q;
    while (q<N)
        q= 5*q+3;
    q= 2*c;
}
q= 11;
```

Para la solución de este ejercicio se consideró analizarlo como dos ciclos anidados, es decir, no importando que uno sea while y el otro for.

Para el ciclo while se presenta este patrón.

$$\begin{array}{lcl}
 i & & q \\
 0 & 5 \cdot 0 + 3 & = 3 \\
 1 & 5 \cdot 3 + 3 & = 18 \\
 2 & 5^2 \cdot 3 + 5 \cdot 3 + 3 & = 93 \\
 3 & 5^3 \cdot 3 + 5^2 \cdot 3 + 5 \cdot 3 + 3 & = 468 \\
 \vdots & & \\
 K & 5^K \cdot 3 + 5^{K-1} \cdot 3 + 5^{K-2} \cdot 3 + \dots + 5 \cdot 3 + 3 & = N, \text{quiza}
 \end{array}$$

no se puede factorizar a K de manera sencilla, simplemente opté por dar un aproximado con $\log_5(N)$ y así saldrá el valor de K que será el exponente que más aporte da al resultado.

Cuando hice el análisis para el for encontré un patrón similar:

$$\begin{array}{lcl}
 i & & q \\
 0 & 0 & \\
 1 & 2^0 \cdot 7 & = 2^2 + 3 = 7 \\
 2 & 2 \cdot 7 + 7 & = 21 \\
 3 & 2^2 \cdot 7 + 2 \cdot 7 + 7 & = 49 \\
 4 & 2^3 \cdot 7 + 2^2 \cdot 7 + 2 \cdot 7 + 7 & = 105 \\
 \vdots & & \\
 K & 2^{K-1} \cdot 7 + 2^{K-2} \cdot 7 + \dots + 2 \cdot 7 + 7 & = N
 \end{array}$$

ahora, para dar con el número de iteraciones del for aplicaremos $\log_2(N) - 2$ por la forma en la que podemos representar el número 7.

$$T_{\text{for}} = 1 + (M+1)T_{\text{comparación}} + MT_{\text{anísotope}} + MT_{\text{actualización}}$$

$$T_{\text{whilefor}} = T_{L1} + T_{\text{while}} + T_{L4} = 1 + \log_5(N) + 3 \log_5(N) + 2$$

$$T_{\text{while}} = (L)T_{\text{comparación}} + (L)T_{L3} = \log_5(N) \cdot 1 + \log_5(N) \cdot 3$$

$$T_{L3} = 3 \quad // \quad 1 \text{ multiplicación, 1 suma y 1 asignación.}$$

$$T_{L4} = 2 \quad // \quad 1 \text{ multiplicación, 1 asignación.}$$

$$T_{\text{for}} = 1 + (\log_2(N) - 1) \cdot 1 + (\log_2(N) - 2) (3 + 4 \log_5(N)) + (\log_2(N) - 2) (2)$$

$$T_{\text{for}} = \log_2(N) + 3 \log_2(N) + 4 \log_2(N) \log_5(N) - 6 - 8 \log_5(N) + 2 \log_2(N) - 4$$

$$T_{\text{for}} = 6 \log_2(N) + 4 \log_2(N) \log_5(N) - 8 \log_5(N) - 10$$

$$R = O(4 \log_2(N) \log_5(N)), \quad \underline{11} \quad (1)$$

5. Considere el siguiente ciclo, correspondiente al *Problema del Granizo / Conjetura de Collatz*

```
while (x>1)
{
    if (x%2==0)
        x= x/2;
    else
        x= 3*x+1;
}
```

¿Cuántas operaciones realiza el ciclo cuando a) $x = 9$, b) $x = 1619$, c) $x = 2^N$?

Iteración	Valor de x	Operaciones
0	9	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
1	28	2 comparaciones + 3 operaciones (modulo, división y asignación)
2	14	2 comparaciones + 3 operaciones (modulo, división y asignación)
3	7	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
4	22	2 comparaciones + 3 operaciones (modulo, división y asignación)
5	11	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
6	34	2 comparaciones + 3 operaciones (modulo, división y asignación)
7	17	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
8	52	2 comparaciones + 3 operaciones (modulo, división y asignación)
9	26	2 comparaciones + 3 operaciones (modulo, división y asignación)
10	13	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
11	40	2 comparaciones + 3 operaciones (modulo, división y asignación)
12	20	2 comparaciones + 3 operaciones (modulo, división y asignación)
13	10	2 comparaciones + 3 operaciones (modulo, división y asignación)
14	5	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
15	16	2 comparaciones + 3 operaciones (modulo, división y asignación)
16	8	2 comparaciones + 3 operaciones (modulo, división y asignación)
17	4	2 comparaciones + 3 operaciones (modulo, división y asignación)
18	2	2 comparaciones + 3 operaciones (modulo, división y asignación)
19	1	1 comparación
R_a	-	102 operaciones

Iteración	Valor de x	Operaciones
0	1619	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
1	4858	2 comparaciones + 3 operaciones (modulo, división y asignación)
2	2429	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
3	7288	2 comparaciones + 3 operaciones (modulo, división y asignación)
4	3644	2 comparaciones + 3 operaciones (modulo, división y asignación)
5	1822	2 comparaciones + 3 operaciones (modulo, división y asignación)
6	911	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
7	2734	2 comparaciones + 3 operaciones (modulo, división y asignación)
8	1367	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
9	4102	2 comparaciones + 3 operaciones (modulo, división y asignación)
10	2051	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
11	6154	2 comparaciones + 3 operaciones (modulo, división y asignación)
12	3077	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
13	9232	2 comparaciones + 3 operaciones (modulo, división y asignación)
14	4616	2 comparaciones + 3 operaciones (modulo, división y asignación)
15	2308	2 comparaciones + 3 operaciones (modulo, división y asignación)
16	1154	2 comparaciones + 3 operaciones (modulo, división y asignación)
17	577	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
18	1732	2 comparaciones + 3 operaciones (modulo, división y asignación)
19	866	2 comparaciones + 3 operaciones (modulo, división y asignación)
20	433	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
21	1300	2 comparaciones + 3 operaciones (modulo, división y asignación)
22	650	2 comparaciones + 3 operaciones (modulo, división y asignación)
23	325	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
24	976	2 comparaciones + 3 operaciones (modulo, división y asignación)
25	488	2 comparaciones + 3 operaciones (modulo, división y asignación)
26	244	2 comparaciones + 3 operaciones (modulo, división y asignación)

27	122	2 comparaciones + 3 operaciones (modulo, división y asignación)
28	61	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
29	184	2 comparaciones + 3 operaciones (modulo, división y asignación)
30	92	2 comparaciones + 3 operaciones (modulo, división y asignación)
31	46	2 comparaciones + 3 operaciones (modulo, división y asignación)
32	23	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
33	70	2 comparaciones + 3 operaciones (modulo, división y asignación)
34	35	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
35	106	2 comparaciones + 3 operaciones (modulo, división y asignación)
36	53	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
37	160	2 comparaciones + 3 operaciones (modulo, división y asignación)
38	80	2 comparaciones + 3 operaciones (modulo, división y asignación)
39	40	2 comparaciones + 3 operaciones (modulo, división y asignación)
40	20	2 comparaciones + 3 operaciones (modulo, división y asignación)
41	10	2 comparaciones + 3 operaciones (modulo, división y asignación)
42	5	2 comparaciones + 4 operaciones (modulo, multiplicación, suma y asignación)
43	16	2 comparaciones + 3 operaciones (modulo, división y asignación)
44	8	2 comparaciones + 3 operaciones (modulo, división y asignación)
45	4	2 comparaciones + 3 operaciones (modulo, división y asignación)
46	2	2 comparaciones + 3 operaciones (modulo, división y asignación)
47	1	1 comparación
R_b	-	250 operaciones

R_c = Para cuando $x = 2^N$, las iteraciones las iteraciones son N y las operaciones son $5(N - 1) + 1 = 5N - 4$

6. Calcule la complejidad temporal, en el peor de los casos, del siguiente segmento de código:

```
int c=0, k=0, q=0, N=0;
int arr[N];

for (q=1, k=0; k<N; q++)
{
    for (c=0; c<q && k<N; c++, k++)
        arr[k]= c;

    if (k<N)
    {
        arr[k]= -1;
        k++;
    }
}
```

Para analizar la complejidad temporal de este código se observó su comportamiento y resulto lo siguiente:

$$N = 10$$

0	-1	0	1	-1	0	1	2	-1	0
---	----	---	---	----	---	---	---	----	---

Por lo que se observa que solamente recorre el arreglo una sola vez y va llenándolo, siguiendo un patrón.

$$R = O(N)$$

7. Calcule la cantidad exacta de pasos que se ejecuta al correr el siguiente código. Asuma que los tiempos de ejecución de las funciones $u(int N)$ y $v(int N)$ son $T_u(N) = 7N^5$ y $T_v(N) = 3N \log_6 N$. Determine la complejidad temporal en el mejor y peor de los casos.

```
for (k=0, q=1; k<N && q<N; k+=3, q*=2)
{
    w= u(N) * v(N);
    for (x=N, y=0; x>N/3 || y<N/3; x--, y++)
        w= u(N) + v(N);
}
```

$$T_{for} = T_{ini} + (M+1)T_{comparación} + MT_{condición} + MT_{actualización}$$

$$T_{ini} = 2$$

$$T_{comparación} = 3$$

$$T_{condición} = T_{L1} + T_{L2}$$

$$T_{actualización} = 4 \quad // \text{ 1 suma, 1 multiplicación y 2 asignaciones}$$

$$T_{L1} = 7N^5 + 3N \log_6 N + 2$$

$$T_{L2} = 2 + 3(L+1) + T_{L3} + 4L$$

$$T_{L3} = 7N^5 + 3N \log_6 N + 2$$

$$\begin{aligned} T_{L2} &= 2 + 3\left(\frac{2}{3}N + 1\right) + \left(\frac{2}{3}N\right)(7N^5 + 3N \log_6 N + 2) + 4\left(\frac{2}{3}N\right) \\ &= 2 + 2N + 3 + \frac{14}{3}N^6 + 2N^2 \log_6 N + \frac{4}{3}N + \frac{8}{3}N \\ &= 5 + 6N + 2N^2 \log_6 N + \frac{14}{3}N^6 \end{aligned}$$

$$\begin{aligned} T_{condición} &= 7N^5 + 3N \log_6 N + 2 + 5 + 6N + 2N^2 \log_6 N + \frac{14}{3}N^6 \\ &= 7 + 6N + 3N \log_6 N + 2N^2 \log_6 N + 7N^5 + \frac{14}{3}N^6 \end{aligned}$$

$$T_{for} = 2 + 3(\log_2 N + 1) + (\log_2 N)(7 + 6N + 3N \log_6 N + 2N^2 \log_6 N + 7N^5 + \frac{14}{3}N^6) + 4 \log_2 N$$

$$\begin{aligned} T_{for} &= 2 + 3 \log_2 N + 3 + 7 \log_2 N + 6N \log_2 N + 3N \log_6 N \log_2 N + 2N^2 \log_6 N \log_2 N + 7N^5 \log_2 N + \\ &\quad \frac{14}{3}N^6 \log_2 N + 4 \log_2 N \end{aligned}$$

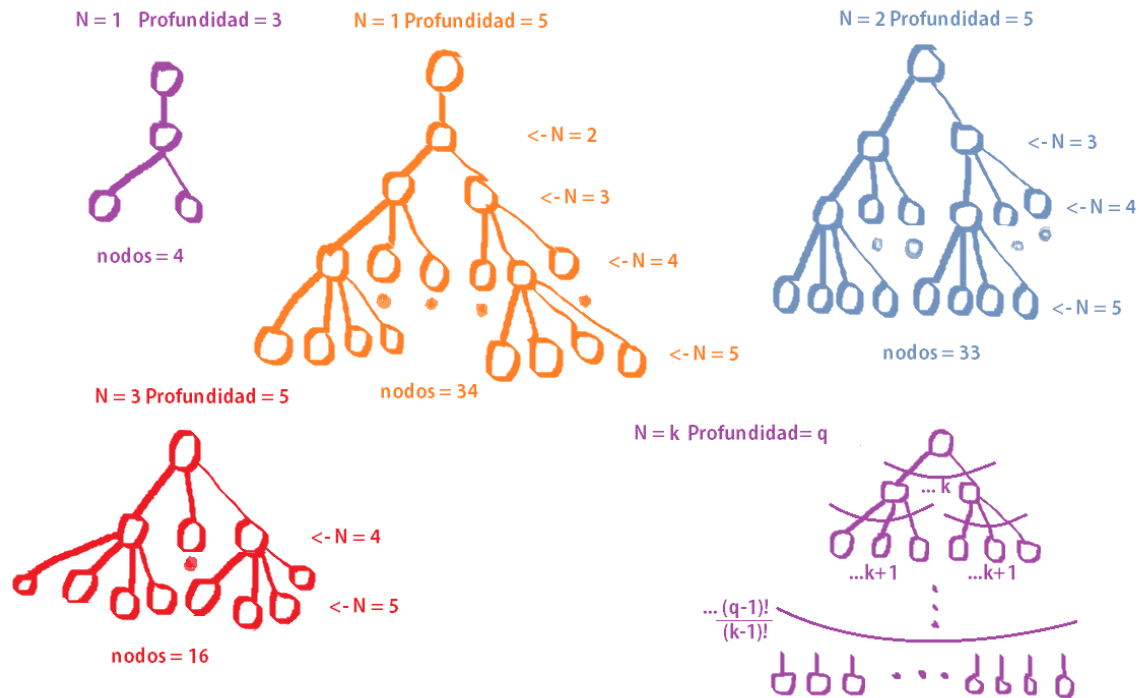
$$\begin{aligned} T_{for} &= 5 + 14 \log_2 N + 6N \log_2 N + 3N \log_6 N \log_2 N + 2N^2 \log_6 N \log_2 N + 7N^5 \log_2 N + \\ &\quad \frac{14}{3}N^6 \log_2 N \end{aligned}$$

$$R = O(N^6 \log_2 N), \quad \Omega(1)$$

8. Observe a la siguiente función recursiva:

```
void funcionRecursiva(int N, int profundidad)
{
    int *arr= NULL;
    if (N==profundidad)
        return;
    arr= (int *) malloc(N*sizeof(int));
    while (x<N)
    {
        funcionRecursiva(N+1, profundidad);
        x++;
    }
    free(arr);
}
```

a) Dibujar el árbol de llamadas recursivas.



b) ¿Cuál es la complejidad temporal correspondiente a la generación de todo el árbol de llamadas?

$$Tiempo\ total = \left(\frac{1}{(N-1)!} \sum_{k=N}^{P-1} k! \right) + 1$$

- c) ¿Cuál es la complejidad espacial correspondiente a la generación de todo el árbol de llamadas?

$$\text{Espacio total} = \left(\frac{1}{(N-1)!} \sum_{k=N}^{P-1} (T_{\text{malloc}}(k) + T_{\text{free}}(k)) (k!) \right) + (T_{\text{malloc}}(N) + T_{\text{free}}(N))$$

- d) ¿Cuántas bifurcaciones recursivas tiene cada invocación?

Considerando que cada nodo es una llamada recursiva, considero que las bifurcaciones de ese nodo quedaran en términos del nivel en el que se encuentre. Considero a la raíz como el nivel 0 del árbol:

$$\text{Bifurcaciones en el nivel } K = N + (K - 1)$$

- e) ¿Cuántas hojas tiene el árbol en su nivel más profundo?

$$\text{Número de hojas} = \frac{(P-1)!}{(N-1)!}$$

9. Escriba un programa que corresponda a la función temporal

$$T(N) = N^6 + 3N^5 + 9N^4 + 6N \log N$$

```
void ejercicioNueve( int N ){
    int i = 0, j = 0, x = 0, m = 0;
    for ( i = 0; i < N; i++ )
        for ( j = 0; j < N; j++ )
            nCuarta ( N );
    while ( x < 9 ){
        if ( x < 3 )
            for ( i = 0; i < N; i++ )
                nCuarta ( N );
        nCuarta ( N );
        x++;
    }
    for ( m = 0; m < 6; m++ )
        for ( i = 0; i < N; i++ )
            for ( j = 0; j < N; j*=2 )
                x += 2;
}

void nCuarta ( int N ){
    int i = 0, j = 0, k = 0, l = 0, x = 1;
    for ( i = 0; i < N; i++ )
        for ( j = 0; j < N; j++ )
            for ( k = 0; k < N; k++ )
                for ( l = 0; l < N; l++ )
                    x *= 1;
}
```

10. Determinar la ecuación de recurrencia y la complejidad temporal de los algoritmos “Las Torres de Hanoi” y “Quicksort”.

El algoritmo de las Torres de Hanoi se puede describir de la siguiente manera:

Moveremos n discos, se analizará por casos:

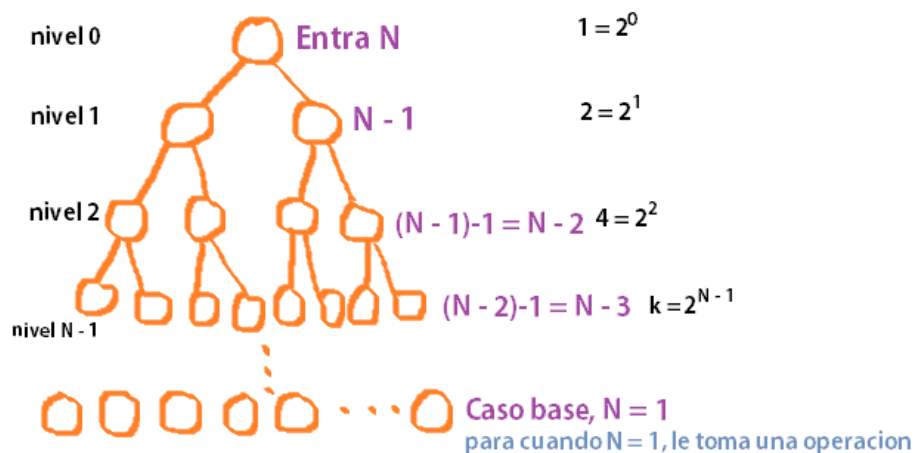
- Supongamos que $n = 1$, lo que nos lleva un movimiento para cambiarlo de aguja.
- Supongamos que $n = 2$, por lo que tendríamos que mover el disco pequeño a la aguja auxiliar, el disco más grande a la aguja final y, para terminar, el disco pequeño lo colocaremos en la aguja final. Por lo que requerimos de 3 movimientos.
- Supongamos que $n = 3$, por lo que primero tendremos que mover los dos primeros discos a una aguja auxiliar, lo que nos llevaría 3 movimientos, como lo vimos en el ejemplo anterior, después mover el disco más grande a la aguja final y finalmente mover los otros tres discos a la aguja final, lo que nos llevaría otros 3 movimientos. En total 7 movimientos.
- Supongamos que $n = 4$, por lo que primero tendremos que mover los primeros 3 discos a una aguja auxiliar, lo que nos lleva 7 movimientos, después, el disco mayor lo movemos a la aguja final y, para terminar, movemos los tres discos que se encontraban en la aguja auxiliar a la final, lo que nos lleva otros 7 movimientos.

Con la descripción anterior se pretende encontrar su ecuación de recurrencia:

Para mover n discos es necesario mover $n - 1$ discos y a esto le sumamos 1, que representa el disco mayor.

$$disco_n = 2disco_{n-1} + 1$$

El programa de mínimo tendrá 2 llamadas recursivas, formando un árbol como se muestra a continuación:



$$Tiempo\ total = \sum_{q=0}^{N-1} 2^q = \frac{2^{N-1+1} - 1}{2 - 1} = 2^N - 1$$

Algoritmo de Quicksort:

```

void quicksort (int* v, int b, int t){
    int pivote;
    if ( b < t ){
        pivote = colocar ( v, b, t );
        quicksort( v, b, pivote - 1 );
        quicksort( v, pivote + 1, t );
    }
}

int colocar ( int* v, int b, int t ){
    int i;
    int pivote, valor_pivote;
    int temp;
    pivote = b;
    valor_pivote = v[pivote];
    for ( i = b + 1; i <= t; i++ ){
        if ( v[i] < valor_pivote ){
            pivote++;
            temp = v[i];
            v[i] = v[pivote];
            v[pivote] = temp;
        }
    }
    temp = v[b];
    v[b] = v[pivote];
    v[pivote] = temp;

    return pivote;
}

```

$$\text{quicksort}(N) = 3 + \text{quicksort}(\text{pivote} - 1 - b) + \text{quicksort}(t - \text{pivote} + 1) + \text{colocar}(N)$$

Para el análisis del algoritmo Quicksort es indispensable el posicionamiento del pivote. Si el pivote se elige de una forma adecuada en todas las invocaciones y la carga en una por cada invocación será la mitad de la carga de la que la mando a llamar, este algoritmo estaría pintando como una complejidad del orden de $\log_2 N$.

Supondremos que el pivote es el adecuado y divide nuestro vector en dos partes posiblemente iguales:

$$\text{quicksort}(N) = 3 + \text{quicksort}\left(\frac{N}{2}\right) + \text{quicksort}\left(\frac{N}{2}\right) + \text{colocar}(N)$$

$$\text{quicksort}(N) = 2\text{quicksort}\left(\frac{N}{2}\right) + \text{colocar}(N) + 3$$

La última ecuación de recurrencia se puede resolver por Método Maestro, pero primero deberé calcular el orden de la función $\text{colocar}(N)$.

$$\text{colocar}(N) = 7N + 6 \Leftrightarrow O(N)$$

Aplicando Método Maestro:

$$\text{quicksort}(N) = \sum_{k=0}^{\log_2 N - 1} 2^k \frac{N}{2^k} + O(N^{\log_2 2}) = N \sum_{k=0}^{\log_2 N - 1} 1 + O(N) = N \log_2 N + N$$

Por lo que su tiempo de ejecución en el mejor de los casos será: $\Omega(N \log_2 N)$.

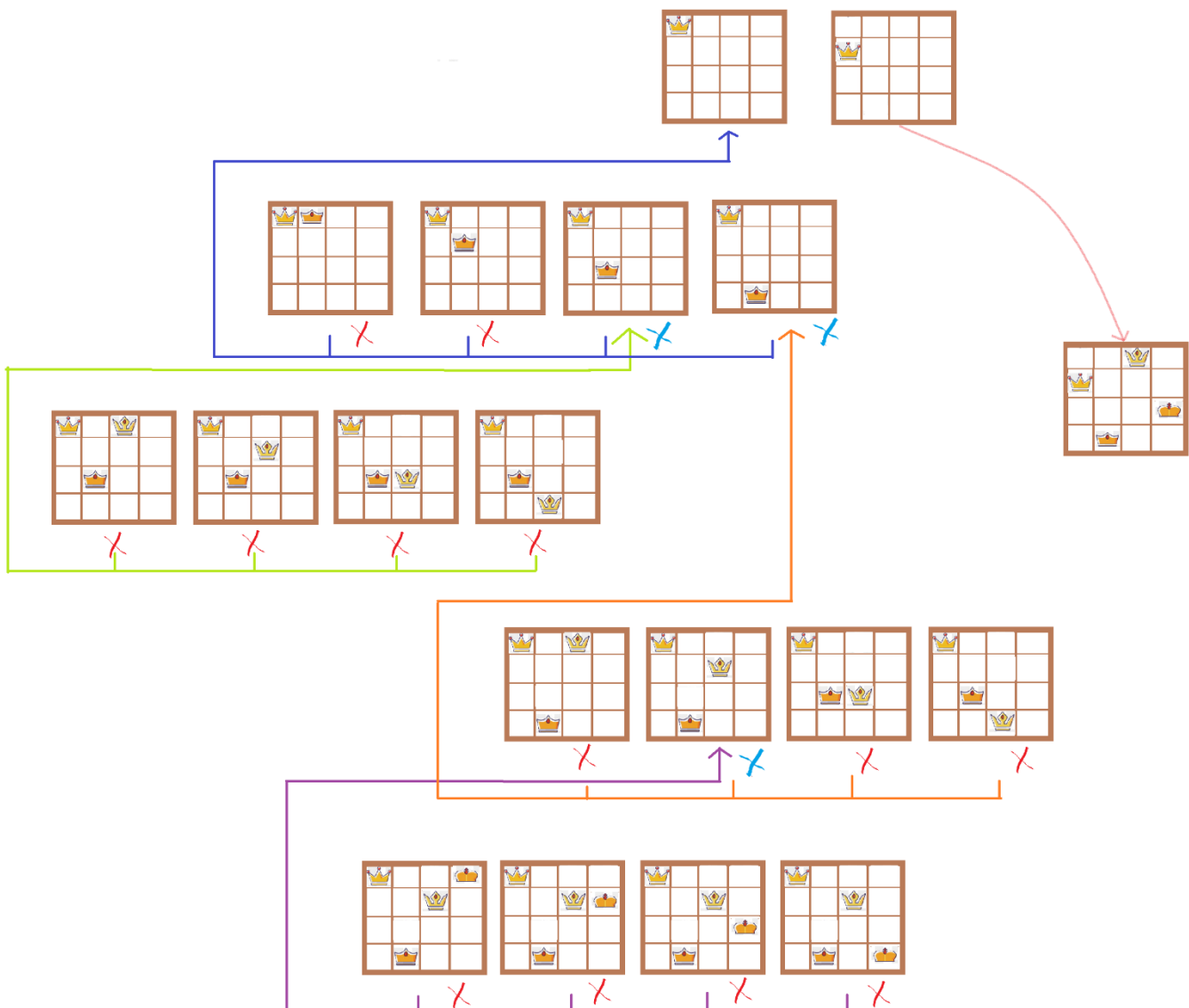
La elección de un buen pivote es la clave para que este algoritmo funcione de la mejor manera.

Ahora bien, el peor de los casos se presentará cuando el pivote sea el número mas pequeño de la lista o el más grande de la lista en todas las llamadas recursivas por lo que quedaría ordenar $N - 1$ elementos.

En el análisis anterior $\log_2 N$ equivale a la profundidad del árbol que se va formando con las llamadas recursivas, pero ahora, el árbol que se formará será semejante a una lista de tamaño N , por lo que la complejidad en el peor de los casos será $O(N^2)$.

11. Escriba un programa para resolver el problema de “Las 8 Reinas” y grafique el árbol de backtracking.

El programa que se pide es una práctica, por lo que aquí no se anexará el código.



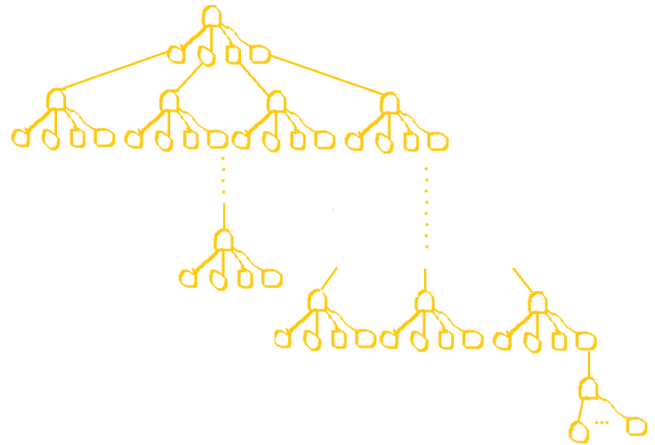
12. Dibuje el árbol de invocaciones recursivas y escriba la ecuación de recurrencia del siguiente código. Calcule su complejidad temporal en el peor de los casos. (Nota: asuma que la funcionLineal tiene una complejidad $O(N)$).

```
void funcionRecursiva(int N)
{
    if (N<=0)
        return;

    funcionRecursiva(N/7);
    funcionRecursiva(N/5);

    funcionLineal();

    funcionRecursiva(N/3);
    funcionRecursiva(N/2);
}
```



Ecuación de Recurrencia: $T(N) = T\left(\frac{N}{7}\right) + T\left(\frac{N}{5}\right) + T\left(\frac{N}{3}\right) + T\left(\frac{N}{2}\right) + f(N)$

En el peor de casos, asumiremos que las 4 llamadas recursivas trabajan con $\frac{N}{2}$ que es el peor de los casos de las 4 llamadas recursivas reales que hace el algoritmo. Por lo que podríamos calcularlo mediante el Método Maestro.

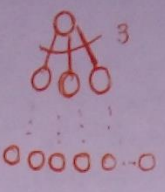
$$T(N) = \sum_{k=0}^{L-1} \left(a^k \frac{f(N)}{b^k} \right) + O(N^{\log_b a})$$

$$\begin{aligned} T(N) &= \sum_{k=0}^{\log_2 N - 1} \left(4^k \frac{N}{2^k} \right) + O(N^{\log_2 4}) = N \sum_{k=0}^{\log_2 N - 1} 2^k + O(N^2) \\ &= N \left(\frac{2^{\log_2 N - 1 + 1} - 1}{2 - 1} \right) + O(N^2) = N(N^{\log_2 2} - 1) + O(N^2) \\ &= N(N - 1) + O(N^2) = N^2 - N + N^2 = 2N^2 - N \end{aligned}$$

13. Sean $T_1(m) = 3T_1\left(\frac{m}{7}\right) + f(m)$ y $T_2(m) = 2T_2\left(\frac{m}{5}\right) + f(m^2)$ dos ecuaciones de recurrencia. Obtenga el orden de complejidad temporal de ambas ecuaciones y bosqueje sus árboles recursivos.

$$T_1(m) = 3T_1\left(\frac{m}{7}\right) + f(m)$$

$a=3, b=7, f(m)=m$



$$T(m) = \sum_{k=0}^{L-1} \left(3^k \cdot \frac{m}{7^k} \right) + O(n^{\log_b a})$$

$$L = \log_7 m$$

$$T(m) = \sum_{k=0}^{L-1} \left(3^k \cdot \frac{m}{7^k} \right) + O(m^{\log_7 3})$$

$$T(m) = m \sum_{k=0}^{L-1} \left(\frac{3}{7} \right)^k + O(m^{0.564})$$

$$\sum_{k=0}^M d^k = \frac{d^{M+1} - 1}{d - 1} = \frac{d^{L-1+1} - 1}{d - 1} = \frac{\left(\frac{3}{7}\right)^L - 1}{\frac{3}{7} - 1}$$

$M = L - 1$

$$\sum_{k=0}^M d^k = \frac{\frac{3}{7}^{\log_7 m} - 1}{\frac{3}{7} - 1} = \frac{m^{\log_7 \left(\frac{3}{7}\right)} - 1}{-\frac{4}{7}} = \frac{m^{-0.4} - 1}{-\frac{4}{7}} = \frac{-1}{-\frac{4}{7}} = \frac{7}{4}$$

cuando m tiende a ser muy grande, se hace 0.

$$T(m) = \frac{7}{4} m + O(m^{0.564})$$

$$R = O(m)$$

$$T_2(m) = 2T_2\left(\frac{m}{5}\right) + f(m^2)$$

$$\begin{aligned}
 a &= 2, \quad b = 5, \quad g(m) = m^2 \\
 T_2(m) &= \sum_{k=0}^{L-1} 2^k f\left(\frac{m}{b^k}\right) + O(m^{\log_5 2}) \\
 L &= \log_5 m \\
 T_2(m) &= \sum_{k=0}^{L-1} 2^k \left(\frac{m}{5^k}\right)^2 + O(m^{\log_5 2}) \\
 T_2(m) &= \sum_{k=0}^{L-1} 2^k \left(\frac{m^2}{25^{2k}}\right) + O(m^{\log_5 2}) \\
 T_2(m) &= \sum_{k=0}^{L-1} 2^k / 25^{2k} + O(m^{\log_5 2}) \\
 T_2(m) &= m^2 \sum_{k=0}^{L-1} \left(\frac{2}{25}\right)^k + O(m^{\log_5 2}) \\
 \sum_{k=0}^{L-1} \left(\frac{2}{25}\right)^k &= \frac{\left(\frac{2}{25}\right)^{L-1+1} - 1}{\frac{2}{25} - 1} = \frac{\frac{2}{25}^{\log_5 m} - 1}{\frac{2}{25} - 1} = \frac{m^{\log_5 \frac{2}{25}} - 1}{-\frac{23}{25}} \\
 &= \frac{m^{-1.56} - 1}{-\frac{23}{25}} = \frac{-1}{-\frac{23}{25}} = \frac{25}{23} \quad \text{cuando } m \text{ tiende a ser muy grande, se hace } 0 \\
 T_2(m) &= m^2 \left(\frac{25}{23}\right) + O(m^{0.43}) = \frac{25}{23} m^2 + m^{0.43} \\
 R &= O(m^2)
 \end{aligned}$$

19. Considere un grafo $G = (V, E)$ ($|V| = N, |E| = M$) ¿Cuántos subgrafos contiene el grafo G ? Demostrar que el número de árboles dentro del grafo es N^{N-2} (Formula de Cayley)

Se utilizará la demostración que el matemático Heinz Prüfer obtuvo.

Antes de comenzar con la demostración, es importante mencionar 2 Teoremas que se utilizaron:

Teorema: En cualquier árbol existe un vértice de grado 1.

Teorema: Todo árbol con n vértices, tiene $n - 1$ aristas.

La demostración hace uso de una secuencia de pasos que demostrara que existe una relación 1:1 entre un árbol etiquetado y un código o secuencia de números, al que llamaremos Código de Prüfer.

Tomaremos el grafo G que nos dan en el problema y vamos a considerar que corresponde a un árbol.

Luego, etiquetaremos sus vértices con números del 1 al número de vértices con los que cuente el grafo, seguido de esto elegimos un vértice de grado 1 este deberá tener como etiqueta el número más pequeño de todos los vértices de grado 1 disponibles, lo llamaremos w .

Tendremos que ir haciendo una lista que contendrá la etiqueta del vértice adyacente a w , al que llamaremos t y pasaremos a eliminar el vértice w junto con la arista wt del grafo original. Volveremos a aplicar este paso a los vértices de grado 1 que nos queden junto con los que se vayan generando, hasta tener solo 2 vértices y una única arista.

La lista que se ha generado con las etiquetas de los vértices t se le llama Código de Prüfer, que nos quedara de tamaño $N - 2$.

Hasta ahorita lo único que se lleva es que existe una lista de números que pertenecen al conjunto $\{1, 2, \dots, n\}$. Pero ¿este código de Prüfer generará un único árbol y este será G ?

Para formar un árbol a partir de un código de Prüfer también se tiene una secuencia de pasos que se mostrara a continuación:

Se observa el código Prüfer $\{a_1, a_2, \dots, a_{n-2}\}$ y las etiquetas del grafo G original $\{1, 2, \dots, n\}$. El primer vértice que pondremos será el número más pequeño que pertenezca al conjunto de etiquetas de G , pero que no se encuentre en el código de Prüfer y lo uniremos con un vértice que tenga como etiqueta a a_1 . El conjunto de las etiquetas se actualiza quitando el número que ya se encuentra formando parte del árbol y el código de Prüfer también se actualiza quitando a a_1 , volvemos a repetir el proceso buscando el número más pequeño que este en la conjunto de las etiquetas y que no esté en el código, pero tampoco debe ser un número que ya hayamos puesto en el árbol.

Lo que obtendremos al final es el árbol G . por lo que se tiene una correspondencia 1:1. Ya que tenemos esta correspondencia lo que sigue es encontrar el número de árboles que se pueden crear con esas N etiquetas. Es decir, todas las permutaciones posibles contemplando repeticiones de dichas etiquetas.

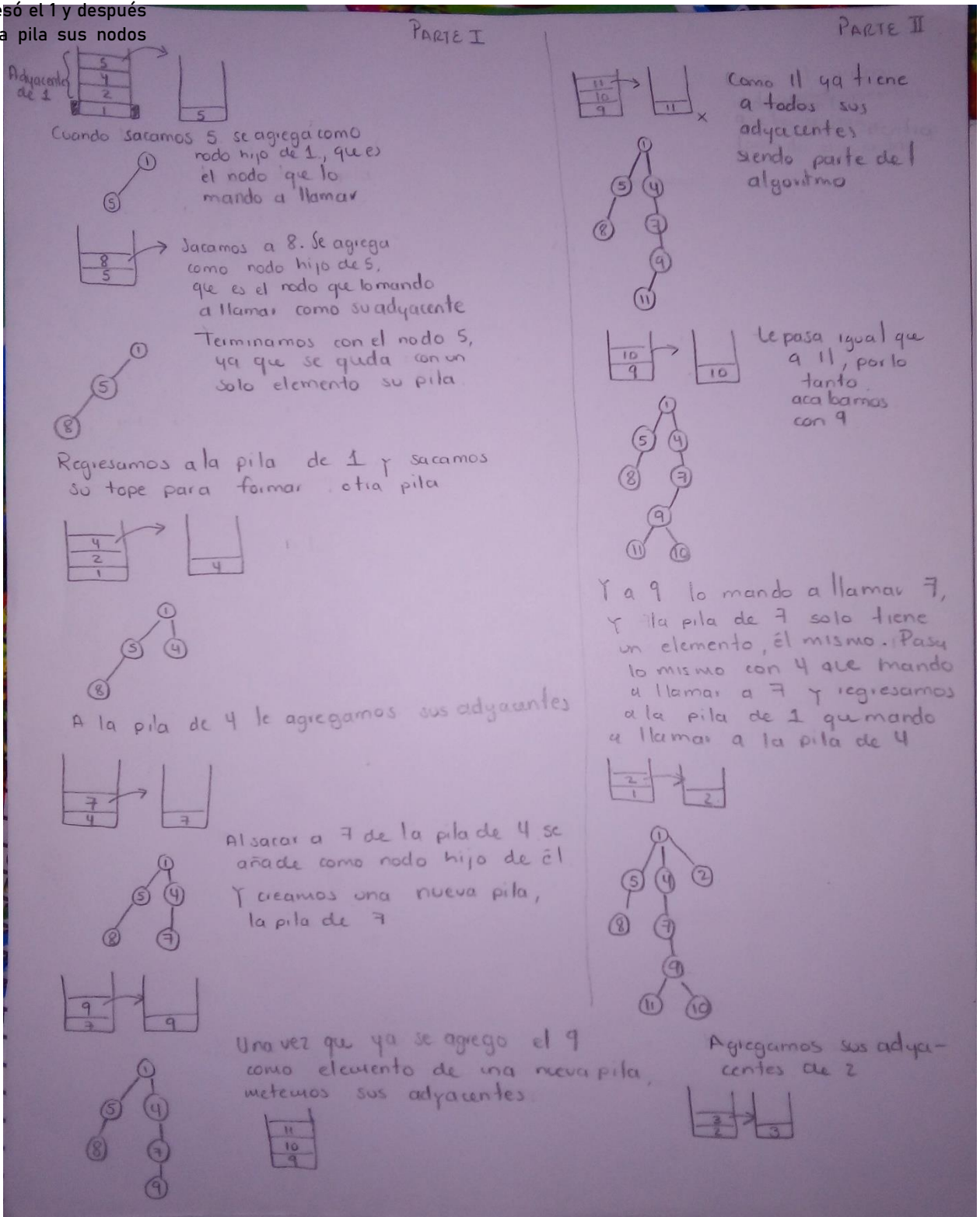
Lo que sigue es hacer uso de las permutaciones, como dicho código tiene $N - 2$ elementos (o posiciones) y tenemos N números $\{1, 2, \dots, n\}$ posibles por cada posición (recordemos que se aceptan repeticiones en el código de Prüfer), entonces nos queda:

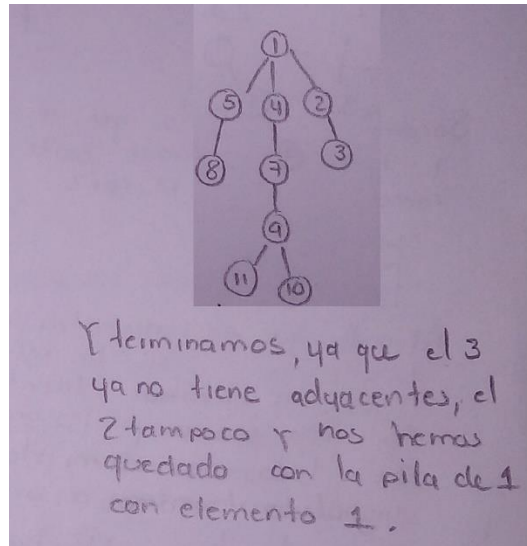
$$N \cdot N \cdot \dots \cdot N = N^{N-2}$$

20. Considerar un árbol A, formado a partir de un grafo G. Bosquejar la construcción del árbol por profundidad (p. ej. utilizando pilas) y por anchura (p. ej. utilizando colas).

Para mostrar la construcción de un árbol se usará un ejemplo: **Construcción por profundidad**

Primero se ingresó el 1 y después se metieron a la pila sus nodos adyacentes.





Construcción por amplitud.

Construcción de un árbol por amplitud, usando colas, a partir de un grafo G.

Grafo G:

Primero, la raíz será el nodo de inicio del grafo G, y lo colocamos en la cola vacía:

① ← Raíz

Después, metemos a los nodos adyacentes de 1 a la cola, y sacamos un elemento, es decir, se sacará un elemento de la cola después de haber ingresado a sus nodos adyacentes. Y esos nodos serán nodos hijos del nodo que se eliminó.

Cola: ~~1~~ | 2 | 5 | 4

Árbol:

```

    1
   / \
  2  5  4

```

Miramos quien está en el tope de la cola y metemos a los nodos adyacentes a él.

Cola: ~~2~~ | 5 | 4 | 3

Árbol:

```

    1
   / \
  2  5  4
 /
3

```

Siempre y cuando no estén en la cola o ya sean parte del árbol. Seguido de esto, agregamos ese nodo al árbol como hijo del nodo Tope y sacamos elemento.

Repetimos los pasos anteriores hasta que quede vacía la cola.

Cola: ~~5~~ | 4 | 3 | 6 | 8

Árbol:

```

    1
   / \
  2  5  4
 / \
3  6
   8

```

Cola: ~~4~~ | 3 | 6 | 8 | 7

Árbol:

```

    1
   / \
  2  5  4
 / \
3  6
   8
   7

```

Como el adyacente de 3 ya se encuentra en el árbol, solo sacamos ese elemento.

Cola: ~~3~~ | 6 | 8 | 7 | 9

Árbol:

```

    1
   / \
  2  5  4
 / \
3  6
   8
   7
   9

```

Y el árbol queda igual.

Cola: ~~6~~ | 7 | 9

Árbol:

```

    1
   / \
  2  5  4
 / \
3  6
   8
   7
   9

```

Y el árbol queda igual.

Cola: ~~7~~ | 9

Árbol:

```

    1
   / \
  2  5  4
 / \
3  6
   8
   7
   9

```

Finalmente, 11 y 10 salen de la cola porque ya no hay adyacentes sin agregar.

Cola: ~~9~~ | 11 | 10

Árbol:

```

    1
   / \
  2  5  4
 / \
3  6
   8
   7
   9
  11
  10

```

21. Sea M una mochila con un volumen V y con una capacidad para almacenar un peso máximo W . Contabilice el número de posibles combinaciones para incluir un conjunto de N objetos, con diferentes volúmenes vk , pesos wk y utilidad uk , tal que no rebasen la capacidad de la mochila y maximicen su utilidad y proponga una forma “eficiente” para resolver este problema NP-Completo.

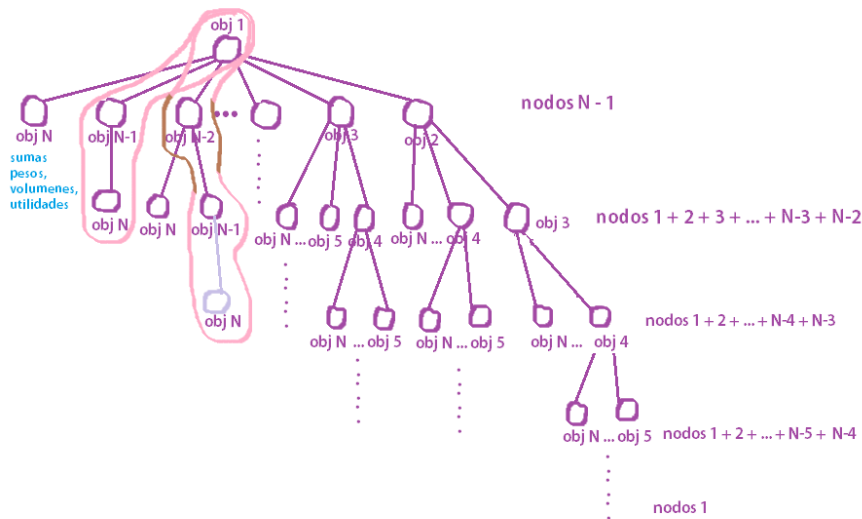
Digamos que O_1, \dots, O_k forman una mochila con una eficiencia S que es máxima en cuestión de utilidad, ya que supongamos que se formó con base a una lista ordenada de forma decreciente de los objetos en función de su utilidad. Lo que me tomo conseguir esta lista en función de la utilidad fue N a lo más.

Para el resto de la mochila tomare en cuenta el peso y el volumen del que aun dispone la mochila

Al resto de la mochila la consideramos como una mochila nueva y procedo a analizar su peso y su volumen (aclaración: se quitarán los objetos que ya sean parte de la mochila vieja) de la siguiente manera:

Si el peso es considerablemente mayor al volumen, es decir, si es muy notorio que el peso $>$ volumen, se hará una lista de objetos según el valor de su volumen de forma decreciente y si la relación entre estas dos variables es inversa, hare la lista respecto al peso y el siguiente proceso aplica para ambos casos.

Entonces, nombrare al objeto número uno de la lista como obj 1, el obj 2 será el que va detrás de obj 1 y así sucesivamente. Posteriormente, se hará las siguientes combinaciones, consiguiendo analizar todas las posibilidades con respecto a sus pesos, volúmenes y utilidad. El objetivo principal de hacer todas estas combinaciones será encontrar los objetos que mejor se ajusten al espacio y peso disponible maximizando la utilidad.



$$\text{Cantidad de invocaciones} = \left(\sum_{i=1}^{N-2} \frac{(i)(i+1)}{2} \right) + N$$

Partiendo de este árbol que se creara una sola vez y utilizando programación dinámica se puede reducir el tiempo para la creación del mismo árbol y para todas las combinaciones posibles que se harían, que son las siguientes:

$$Cantidad\ de\ combinaciones = \sum_{i=1}^N \binom{N}{i}$$

24. Plantee un isomorfismo entre los Teoremas de Incompletitud e Inconsistencia de Gödel y el Problema de Detención de la Máquina de Turing. (Tip: Considere los argumentos utilizados por Gödel y Turing en las demostraciones para resolver el problema de encontrar soluciones enteras a las ecuaciones Diofánticas).

La similitud que hay entre los Teoremas de incompletitud e inconsistencia de Gödel y el Problema de detención de la maquina de Turing se analizaran a partir de sus demostraciones que se muestran en algunos sitios de internet.

Para la demostración de los Teoremas de Gödel ¹se hace referencia a el uso de una formula, a dicha formula le pertenece un numero único y ese numero le pertenecerá una única fórmula, llamémosla *Dem* que dada una oración, que es una formula con variable libre solo que con sustitución, encontrara la codificación de la formula a la que pertenece, pero solamente lo podrá hacer siempre y cuando a esta fórmula se pueda llegar mediante los axiomas que se presentan dentro del sistema al que pertenece, de esta forma *Dem* será deducible.

Una vez llegado a este punto, se tomará la definición de un conjunto² de números *K* y se denotará con la siguiente formula:

$$n \in K \Leftrightarrow \overline{Dem([F_n(n)])}$$

Usando la definición que describe a *Dem* se puede concluir que: $n \in K$ si y solo si $F_n(n)$ no es deducible en su sistema. ... (1)

Puesto que en este sistema todo lo podemos formalizar, es decir que al conjunto *K* le pertenece un codigo al que llamaremos *q*. Entonces, $n \in K$ si y solo si $F_q(n)$ es deducible en el sistema.... (2)

Consideremos, $F_q(q)$:

Si decimos que $F_q(q)$ es deducible en el sistema en el que vive, por (1) $q \notin K$ pero por (2) $F_q(q)$ no es deducible.

Ahora, tomare tal cual un razonamiento de la fuente de esta demostración respecto a los Teoremas de Gödel:

¹ La demostración la tome de un archivo de Claudio Gutiérrez, *El Teorema de Incompletud de Gödel*, 1999. Enlace <https://users.dcc.uchile.cl/~cgutierrez/otros/godel.pdf>

² Este conjunto de números estará conformado por aquellos códigos de una fórmula de variable libre.

Es decir, es imposible que $F_q(q)$ sea deducible en \mathbf{N} (pues suponerlo lleva a una contradicción). También es imposible que $\neg F_q(q)$ sea deducible en \mathbf{N} , pues si suponemos que $\neg F_q(q)$ es deducible, entonces como \mathbf{N} es consistente, $F_q(q)$ no es deducible, pero eso significa que $F_q(q)$ es deducible, contradicción. Es decir, demostramos:

Teorema 1 (Gödel, 1931) *La oración $F_q(q)$ es indecidible en \mathbf{N} , es decir en \mathbf{N} no se puede deducir $F_q(q)$ ni su negación.*

Lo que quisiera resaltar de dicha demostración es que se utiliza una fórmula Dem para saber si una fórmula $F_{estrella}(fantasma)$ se puede deducir de los axiomas que describe el sistema.

El isomorfismo con el problema de detención es que dado un sistema X , el cual tiene sus propios axiomas y enunciados que describen dicho sistema. Entonces, nos ayudamos de una función Y (que en el caso del problema de parada es una supuesta Máquina de Turing) que nos entrega una respuesta, Dem y $Diagonal$ ³ (se describen en las demostraciones de los Teoremas de Gödel y Turing respectivamente). Dichas formulas tienen en su interior otra función con 2 parámetros y cuando le suministramos parámetros con valores iguales, se llegan a conclusiones semejantes. En los Teoremas de Gödel, se concluye que es indecidible en un sistema \mathbf{N} tratar de probar una sentencia de \mathbf{N} y para el problema de detención se termina con una paradoja, concluyendo que no existe una máquina de Turing con las características de $Diagonal$, por lo tanto, es no computable.

³ La demostración a la que hago referencia es la que presenta el sitio de Wikipedia, *Problema de la parada*, 2009. Específicamente, la que se presenta en el apartado de *Irresolubilidad del problema*. Enlace: https://es.wikipedia.org/wiki/Problema_de_la_parada