Universidad Autónoma Metropolitana Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Diseño de reglas de Golomb óptimas

José Julio Santana González
209302794

Trimestre 2014 Invierno
Reporte final

Asesores de proyecto terminal
Dr. Francisco Javier Zaragoza Martínez
Profesor Titular
Departamento de Sistemas

Dr. Crevel Bautista Santiago
Profesor Asociado
Departamento de Sistemas

Resumen

Las reglas de Golomb son un problema derivado de la combinatoria estudiado por primera vez por Simon Golomb, en el cual se trata de hacer un arreglo o acomodo de longitudes diferentes sin que ninguna de éstas o la suma de dos o más longitudes consecutivas se repitan dentro de la regla. En este trabajo nos enfocamos en encontrar todas las reglas de Golomb posibles para ciertas longitudes de prueba, así como también las reglas de Golomb de tamaño óptimo.

Tabla de contenido

1. Objetivos

1.1. Objetivo general

- Diseñar e implementar algoritmos que generen reglas de Golomb.

1.2. Objetivos específicos

- Diseñar e implementar un algoritmo que encuentre todas las reglas de Golomb de una longitud dada.
- Diseñar e implementar un algoritmo que encuentre al menos una regla de Golomb óptima de un orden dado.
- Diseñar e implementar un algoritmo que decida si existe una regla de Golomb que mide un conjunto dado de distancias.

2. Introducción

Una regla de Golomb de longitud L está graduada por n marcas las cuales tienen diferentes distancias entre sí. La variable n denota el orden de la regla de Golomb y la variable L su longitud. Por ejemplo, una regla de Golomb que contiene n marcas puede identificarse por un conjunto de números enteros positivos en el intervalo $[a_0, ..., a_{n-1}]$, los cuales pueden ser representados como las marcas sobre la regla y cuyas distancias $|a_i - a_j|$ para toda $i \neq j$, donde $a_0 = 0$ y $a_{n-1} = L$. La Figura 1 muestra una regla de Golomb de longitud L=7 y con n=4 marcas ($a_0 = 0, a_1 = 1, a_2 = 3, a_3 = 7$). En particular, se puede apreciar que las distancias que existen entre cualesquiera dos marcas nunca se repiten.

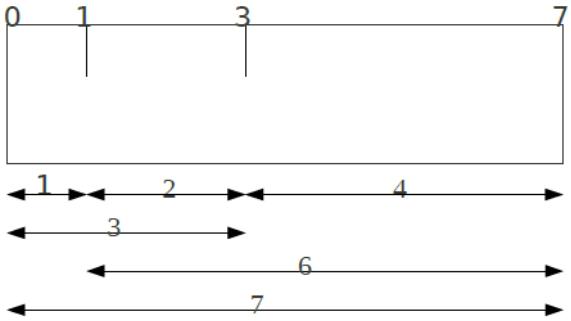


Figura 1: Regla de Golomb de longitud 7.

A continuación se presentan 3 reglas que no son de Golomb debido a que existen distancias repetidas (Figura 2):

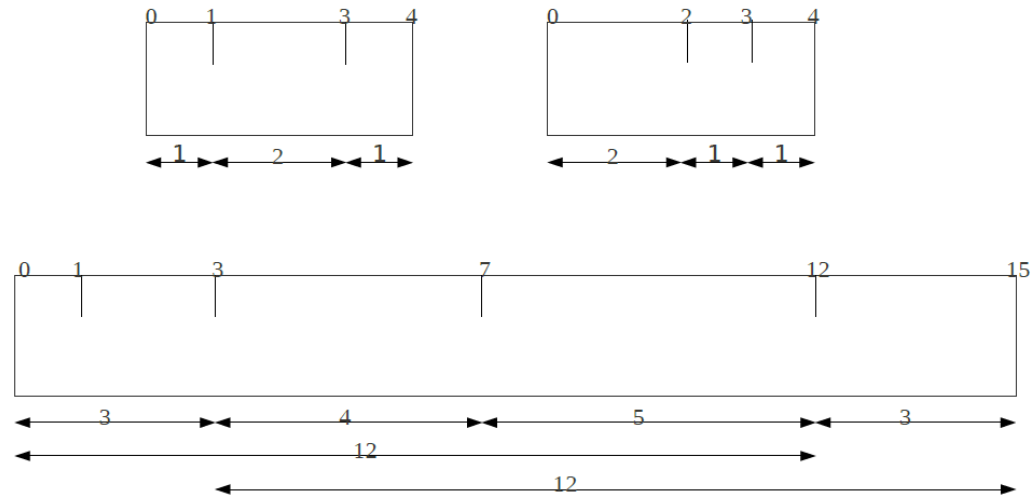


Figura 2: Reglas que no son de Golomb.

Es importante resaltar el hecho de que una regla de Golomb se considera óptima sólo si se puede construir una regla de Golomb de orden n con la menor distancia posible. La Figura 3 muestra una regla de orden 4 y longitud 6, la cuál es óptima ya que 6 es la longitud mínima de la regla que contiene 4 marcas.

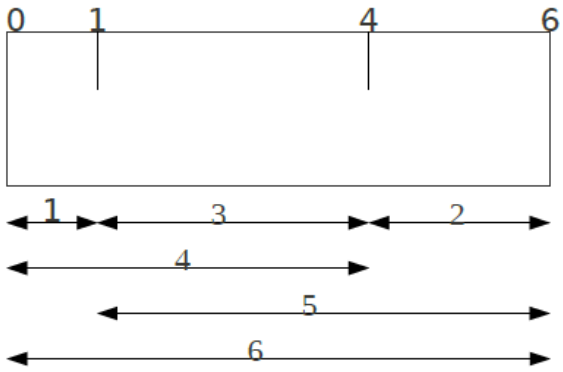


Figura 3: Regla óptima de orden 4.

3. Desarrollo

3.1. Algoritmo que encuentra todas las reglas de Golomb posibles de longitud L

Este algoritmo toma un número entero que representa la longitud y comienza a insertar longitudes en un vector de enteros desde 1 hasta L y verifica que cada longitud que se insertó no esté repetida así como tampoco cualesquiera de las sumas que se puedan realizar con dos o más distancias sucecivas. En caso de que exista alguna longitud repetida, descarta esa posibilidad y corta dicha rama del arbol para evaluar la siguiente.

3.1.1. Implementación

reglas(int n, Vector V)

1. Desde $i = 1$ hasta $L - n$
2. inserta i en V
3. si V es válido
   a) cuenta += reglas $(n + i\ ,\ V)$
4. Fin desde
5. regresa cuenta

3.2. Algoritmo que genera reglas de Golomb óptimas

Este algoritmo toma un entero n que representa el número de marcas de una regla de Golomb y genera todas las reglas posibles de longitud mínima.

3.2.1. Implementación

marcas(int n, vector V)

1. desde$i = 1$ hasta n
2. Si i es mayor al último elemento
   a) Si los elemntos de V forman una regla de Golomb válida
      1) marcas(n,V)
3. regresa la longitud óptima

3.3. Algoritmo que decide si un conjunto dado es una regla de Golomb

Este algoritmo toma como entrada un conjunto de números enteros y decide si dicho conjunto forma una regla de Golomb en el orden en que se recibió, sino, comienza a permutar dicho conjunto para averiguar si se puede generar una regla de Golomb con el. Este algoritmo toma como entrada vector de números enteros, los ordena, permuta. Despues decide si dicha permutación forma parte de una regla de Golomb válida, en caso contrario vuelve a permutar hasta encontrar una solución.

3.3.1. Implementación

1. Recive un vector de enteros V
2. Ordena V
3. Mientras puedas permutar
4. Si v es solución
5. entonce regresa true
6. regresa false

4. Resultados

A continuación se presenta la tabla de tiempos del primer algoritmo, que calcula las reglas de Golomb de tamaño L ejecutando las diferentes Longitudes.

Longitud	No. de Reglas	Tiempo
$L = 4$	3	0.001s
$L = 5$	5	0.001s
$L = 6$	7	0.001s
$L = 7$	13	0.002s
$L = 8$	15	0.002s
$L = 9$	27	0.004s
$L = 10$	25	0.004s
$L = 11$	45	0.004s
$L = 12$	59	0.004s
$L = 13$	89	0.008s
$L = 14$	103	0.008s
$L = 15$	163	0.012s
$L = 16$	187	0.016s
$L = 17$	281	0.024s
$L = 18$	313	0.032s
$L = 19$	469	0.045s
$L = 20$	533	0.062s
$L = 21$	835	0.084s
$L = 22$	873	0.116s
$L = 23$	1319	0.156s
$L = 24$	1551	0.212s
$L = 25$	2093	0.288s
$L = 26$	2347	0.388s
$L = 27$	3477	0.512s
$L = 28$	3881	0.676s
$L = 29$	5363	0.900s
$L = 30$	5871	1.184s
$L = 31$	8267	1.184s
$L = 32$	9443	1.200s
$L = 33$	12887	1.215s
$L = 34$	14069	1.368s
$L = 35$	19229	1.804s
$L = 36$	22113	2.360s
$L = 37$	29359	3.088s
$L = 38$	32229	4.020s
$L = 39$	44127	5.188s
$L = 40$	48659	6.824s

Tabla 1: Resultados del Algoritmo 1.

Longitud	No. de Reglas	Tiempo
$L = 41$	64789	8.633s
$L = 42$	71167	11.069s
$L = 43$	94625	14.157s
$L = 44$	105699	18.077s
$L = 45$	139119	22.905s
$L = 46$	151145	28.998s
$L = 47$	199657	36.650s
$L = 48$	223813	46.155s
$L = 49$	287135	58.020s
$L = 50$	312937	1m12.933s
$L = 51$	413855	1m31.038s
$L = 52$	455723	1m53.635s
$L = 53$	584353	2m21.093s
$L = 54$	642149	2m55.203s
$L = 55$	816241	3m37.174s
$L = 56$	912113	4m27.965s
$L = 57$	1165351	5m31.209s
$L = 58$	1261427	6m46.841s
$L = 59$	1614893	8m19.051s
$L = 60$	1785147	10m11.378s
$L = 61$	2244581	12m28.527s
$L = 62$	2445875	15m12.921s
$L = 63$	3127941	7m0.742s
$L = 64$	3421045	8m45.005s
$L = 65$	4257859	10m56.477s
$L = 66$	4659749	13m35.483s
$L = 67$	5857043	16m48.323s
$L = 68$	6450617	20m44.938s
$L = 69$	8057099	25m32.076s
$L = 70$	8674669	31m18.397s
$L = 71$	10894457	38m21.008s
$L = 72$	12009609	46m59.592s
$L = 73$	14769139	57m8.670s
$L = 74$	16034977	69m38.557s
$L = 75$	19990861	84m30.857s
$L = 76$	21845347	102m47.773s
$L = 77$	26804153	124m27.543s
$L = 78$	29138257	218m22.287s
$L = 79$	35936283	315m49.372s
$L = 80$	39240071	454m40.485s

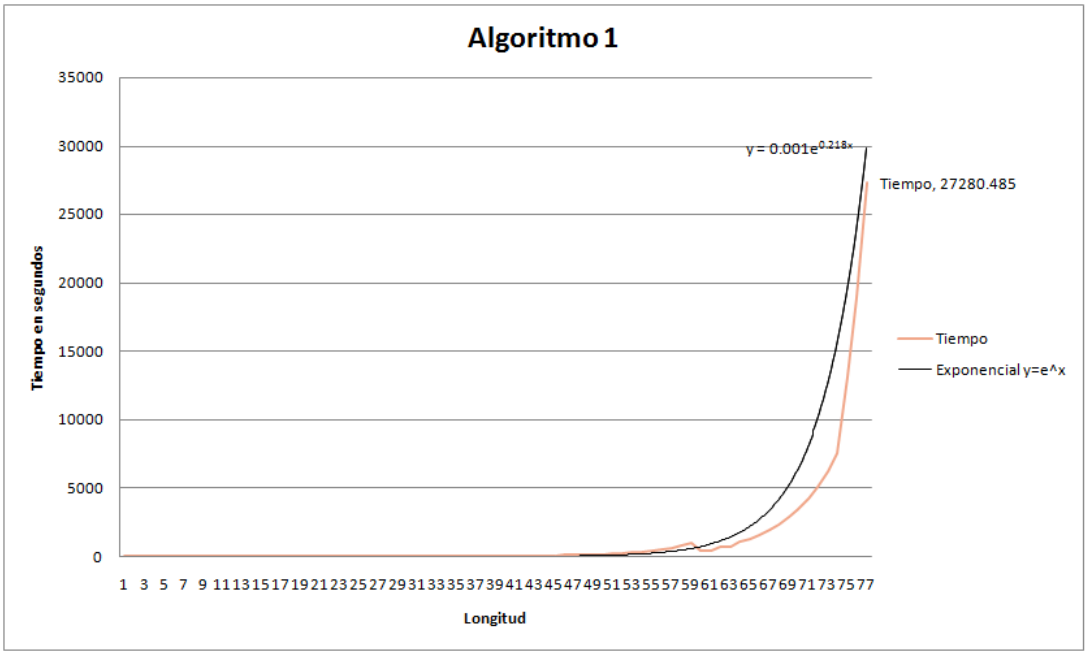


Figura 4: Gráfica de Resultados con respecto al tiempo.

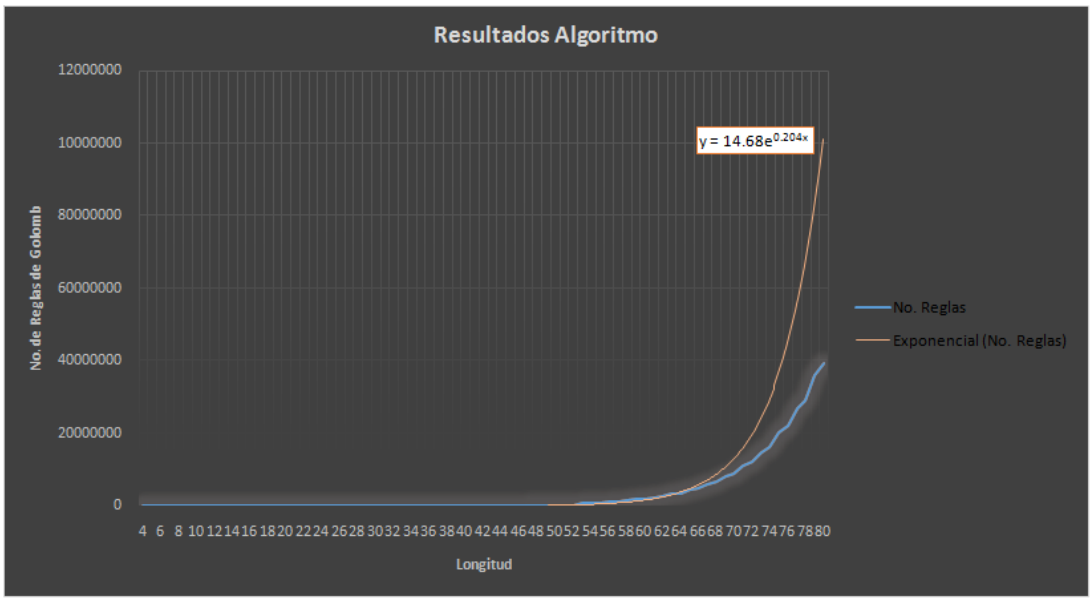


Figura 5: Gráfica de Resultados con respecto al número de reglas.

Se puede apreciar que tanto el cálculo en el tiempo (Figura 4) como en el número total de reglas de Golomb (Figura 5) se presenta un crecimiento exponencial con forme aumenta la longitud de la regla.
Para poder apreciar algunas de las reglas de Golomb generadas por este algoritmo, pasar al Apéndice D.

No de Marcas	Longitud óptima	Tiempo
2	1	0 seg
3	3	0 seg
4	6	0 seg
5	11	0 seg
6	17	0.02 seg
7	25	0.29 seg
8	34	2.73 seg
9	44	21.58 seg
10	55	3.93 min
11	72	1 hora aprox
12	85	11 horas aprox
13	106	8 días aprox

Tabla 2: Resultados del Algoritmo 2.

Podemos observar que para un número muy pequeño de marcas es muy rápido el algoritmo para encontrar la longitud óptima, sin embargo, se puede apreciar cómo se incrementa el tiempo de forma exponencial a partir de un número de marcas de 11. Para poder apreciar algunas de las reglas de Golomb generadas por este algoritmo, pasar al Apéndice E.

5. Conclusiones

Al verificar los resultados arrojados por el algoritmo 1 podemos observar que existen reglas de Golomb repetidas, esto es, que tienen las mismas longitudes dentro de ellas aunque el orden en que estén acomodadas sea diferente. La diferencia fundamental entre el primer algoritmo y el segundo es que éste último sólo se enfoca en encontrar las reglas que tengan la distancia óptima partiendo de un orden dado y el primero arroja todas las combinaciones posibles dada una distancia. Es por eso que podemos observar grandes diferencias en los tiempos reflejados en las tablas 1 y 2, así como la longitud máxima que se pudo calcular en ambos algoritmos. En cuanto al tercer algoritmo, podemos observar que después de ordenar el vector de entrada comienza a permutar hasta encontrar una regla de Golomb válida.

Referencias

[1] C. A. Rodríguez Villalobos, *"Algoritmos para la coloración robusta de árboles binarios"*, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2008.

[2] C. A. Olaguibert Segura, *"Generación de polígonos con triangulaciones ortogonales"*, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2009.

[3] P. E. Torres Jiménez, *"Algoritmo de búsqueda de configuraciones de enlaces ortogonales en dos y tres dimensiones"*, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2011.

[4] G. S. Bloom and S. W. Golomb, *"Application of numbered unidirectional graphs"*, Proc. IEEE, vol. 65, no. 4, pp 562-570, Abr. 1977.

[5] J. B Shearer, *"Some new optimum Golomb rulers"*,IEEE Trans. Inf. Theory, vol. 36, no. 1, pp.183-184, Ene. 1990.

[6] J. P. Robinson, *"Optimum Golomb rulers"*, IEEE Trans. Inf. Theory, vol. c-28, no. 12, pp. 943-944, Dic. 1979.

A. Código Alg1

```
#include <algorithm>
#include <iostream>
#include <list>
#include <stdlib.h>
#include <utility>
#include <vector>

int L;
std::list< std::vector<int> > resulta;

struct Conjuntos
{
    std::vector<int> comidos;
    std::vector<bool> quedan;
};

std::pair<Conjuntos,bool> prendidos(const Conjuntos& v, int i)
{
    int suma=0;
    Conjuntos aux = v;

   aux.comidos.push_back(i);

   for(int j=aux.comidos.size()-1;j>=0;j--)
   {

      suma+=aux.comidos[j];

      if (!aux.quedan[suma]) {
         return std::make_pair(aux,false);
      }
      if(suma==L)
      {
          resulta.push_back(aux.comidos);
      }
      aux.quedan[suma]=false;
   }

   return std::make_pair(aux,true);
}

int reglas(int k, const Conjuntos& v)
{
    if (k == L) {
      return 1;
   }

   int cuantas = 0;

    for(int i=1;i<=L-k;i++) {
        std::pair<Conjuntos,bool> p= prendidos(v,i);

        if(p.second==true) {
            cuantas += reglas(k+i,p.first);
        }
    }

    return cuantas;
}

int main(int argc, char *argv[])
{
    Conjuntos h;
    std::cout<<"Longitud de la regla: ";
    L=atoi(argv[1]);
    std::cout<<L<<'\n';
    h.quedan.resize(L+1, true);
    std::cout<<"cantidad:"<<reglas(0,h)<<'\n';
    return 0;
}
```

B. Código Alg2

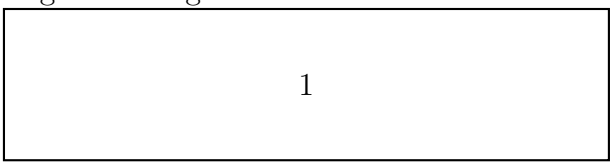
```
1  #include <algorithm>
2  #include <climits>
3  #include <cstdlib>
4  #include <iostream>
5  #include <stdlib.h>
6  #include <utility>
7  #include <vector>
8
9
10 int optima = INT_MAX;
11
12 struct Conjuntos
13 {
14         std::vector<int> marcas;
15         std::vector<bool> ocupados;
16 };
17
18 std::pair<bool,Conjuntos> verifica(const Conjuntos& v, int i,int& n)
19 {
20         Conjuntos nuevo=v;
21         nuevo.marcas.push_back(i);
22
23         nuevo.ocupados.resize(nuevo.marcas.back()+1,true);
24
25         for(int c = nuevo.marcas.size()-2 ; c >= 0; c--)
26         {
27                 int res = nuevo.marcas.back()-nuevo.marcas[c];
28                 if(!nuevo.ocupados[res])
29                         {return std::make_pair(false,nuevo);}
30
31                 nuevo.ocupados[res]=false;
32         }
33
34         if(nuevo.marcas.size()==n)
35                 {optima=nuevo.marcas.back();}
36
37         return std::make_pair(true,nuevo);
38
39 }
40
41 int poner_marcas(int n, Conjuntos& v)
42 {
43         for(int i=1; i<=optima;i++)
44         {
45                         if(i>v.marcas.back()){
46                                 std::pair<bool,Conjuntos> p = verifica(v,i,n);
47
48                                 if(p.first){
49                                         poner_marcas(n,p.second);}
50                         }
51
52         }
53         return optima;
54 }
55 int main(int argc, char *argv[])
56 {
57         Conjuntos inicial;
58         int n;
59
60         inicial.marcas.push_back(0);
61         n=atoi(argv[1]);
62
63         std::cout<<"Longitud de la regla optima con "<<n<<" marcas es:"<<poner_marcas(n,
               inicial)<<'\n';
64 }
```

C. Código Algo3

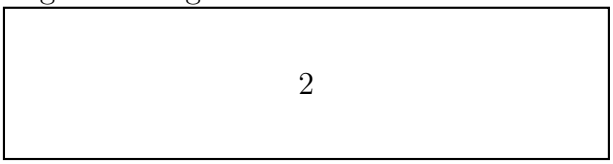
```
#include <algorithm>
#include <iostream>
#include <set>
#include <vector>

std::set<int> values;
std::set<int>::iterator it;
std::vector <int > numeros;


void sumavalida(int n){
    if (n == numeros.size( ) ){
        for (int i = 0; i < numeros.size(); ++i){
            std::cout << numeros[i] << ' ';
        }
        std::cout << '\n';
        exit(0);
                return;
    }
    it = values.find(numeros[n]);
    if (it != values.end()){
        return;
    }
    values.insert(numeros[n]);
    for (int i = 0; i < n; ++i){

        int suma = numeros[n];

        for (int j = i; j < n; ++j){
            suma += numeros[j];
        }

        it = values.find(suma);
        if (it != values.end()){
            return;
        }

        values.insert(suma);
    }

    sumavalida(n+1);
    return;
}

int main ()
{

    int n;
    std::cin >> n;

    for (int i = 0; i < n; ++i){
        int x;
        std::cin >> x;
        numeros.push_back(x);
    }
    std::vector<int> num_aux=numeros;
    std::sort(num_aux.begin(), num_aux.end());

    do{
       values.insert(num_aux[0]);
       sumavalida(1);
       values.clear();
    }while(std::next_permutation(numeros.begin(), numeros.end()));
}
```

D. Reglas de Golomb de diferentes Longitudes

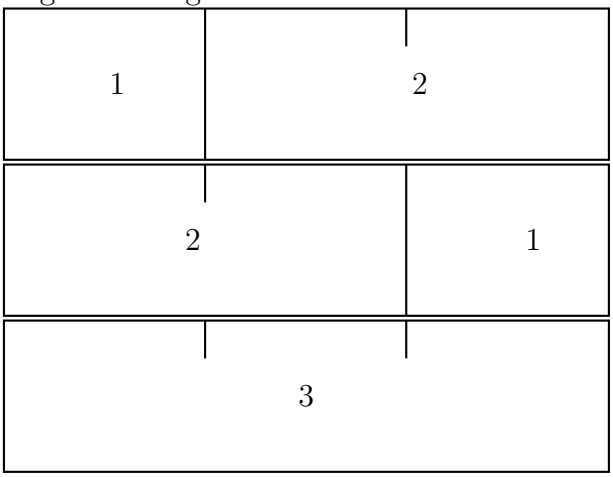
Reglas de longitud 1



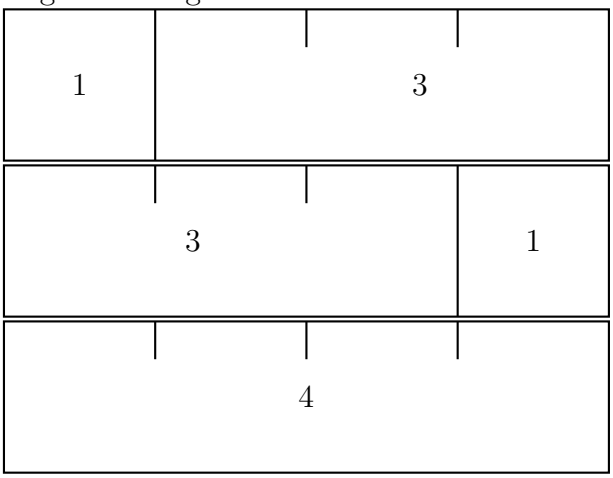
Reglas de longitud 2



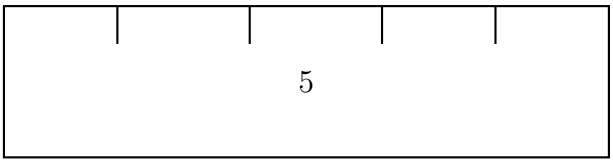
Reglas de longitud 3



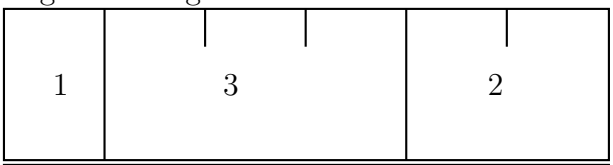
Reglas de longitud 4

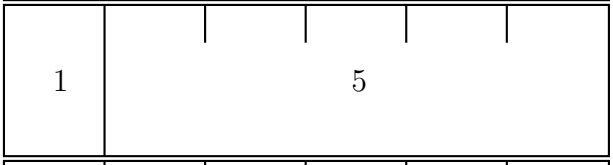


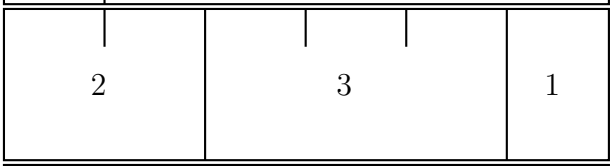
Reglas de longitud 5

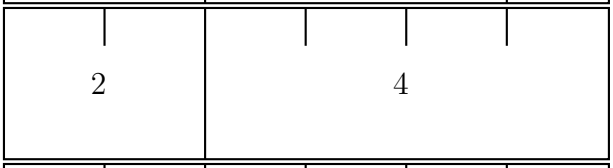


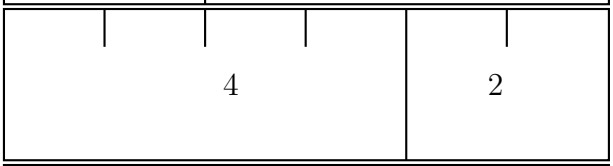
Reglas de longitud 6

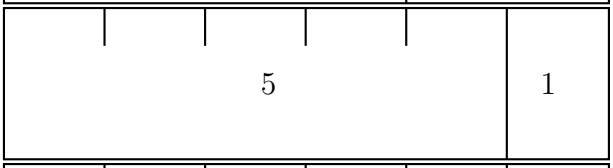


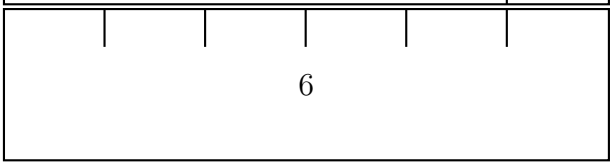




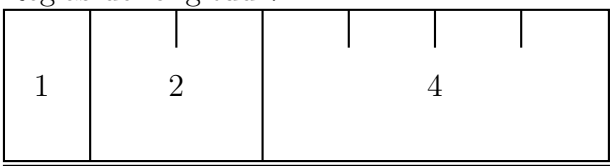


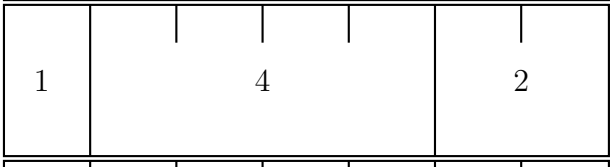


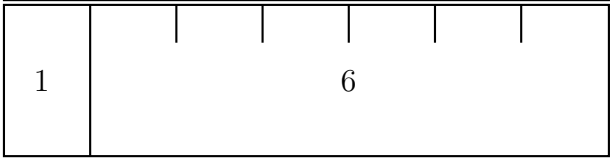


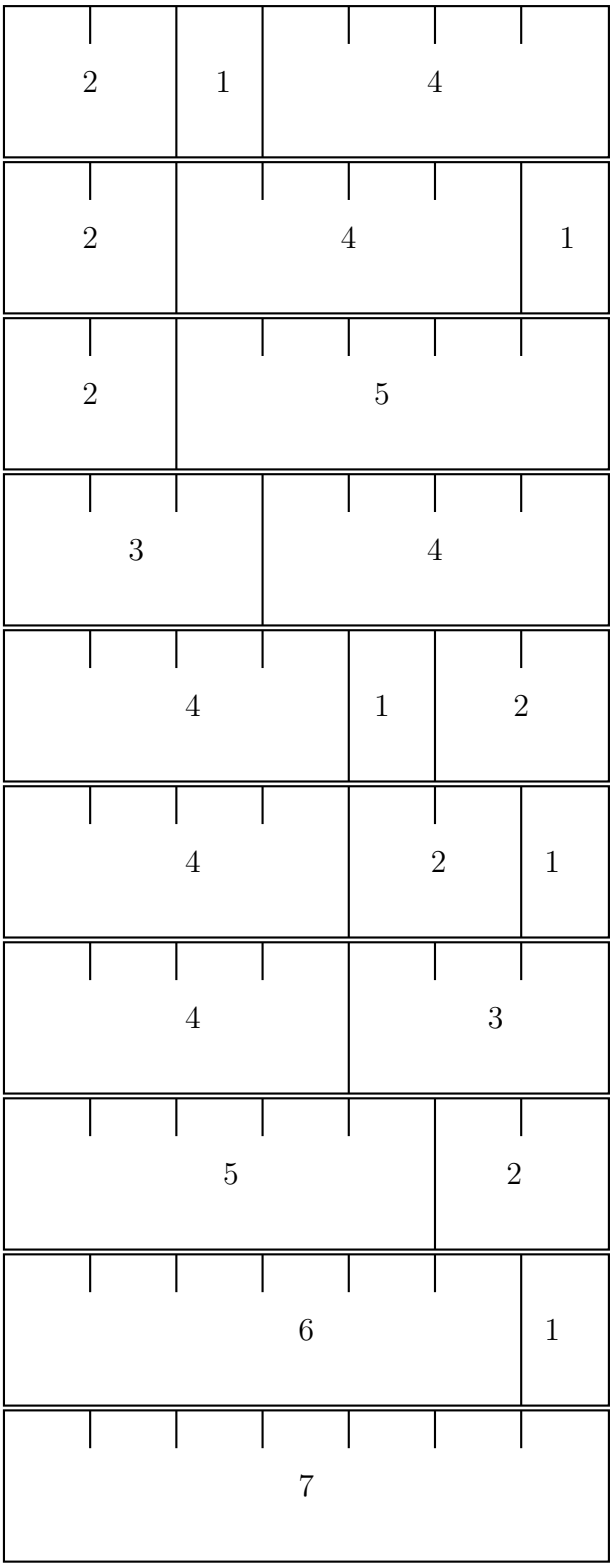


Reglas de longitud 7

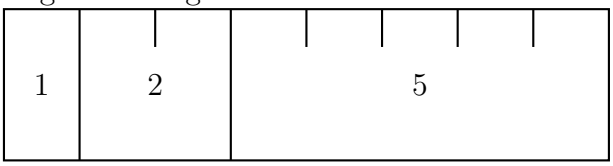


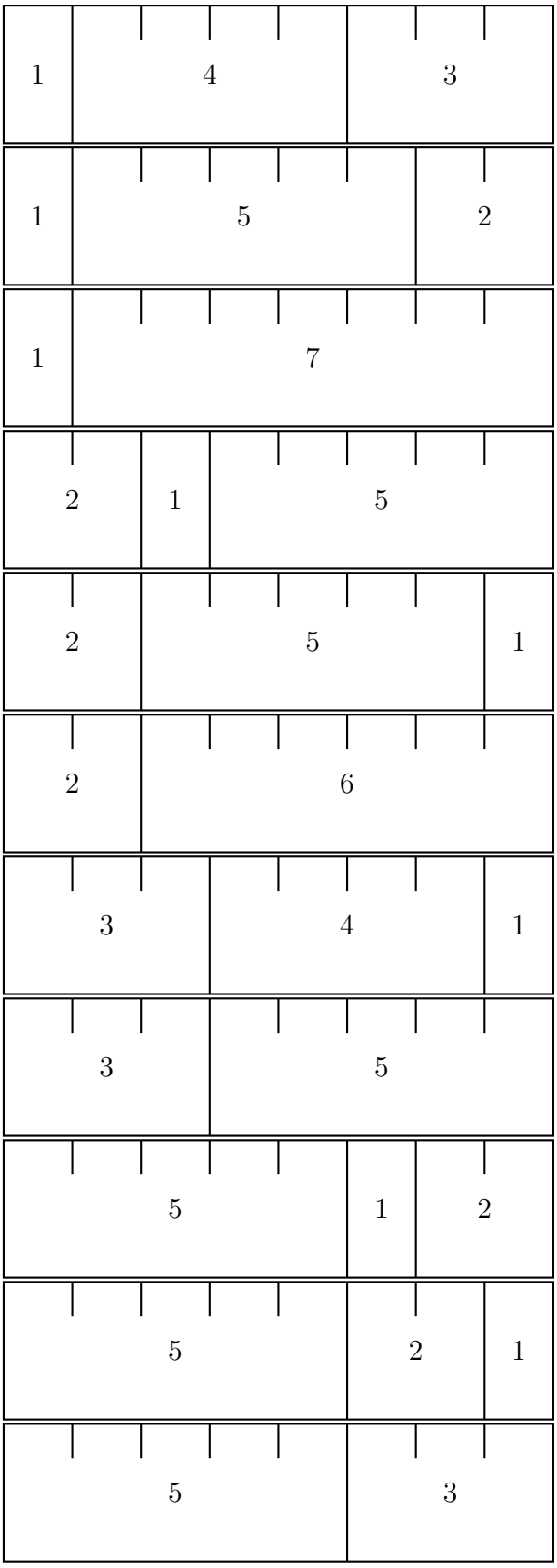


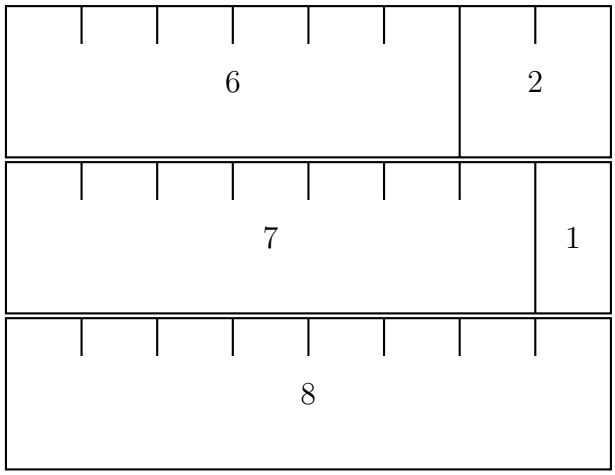




Reglas de longitud 8

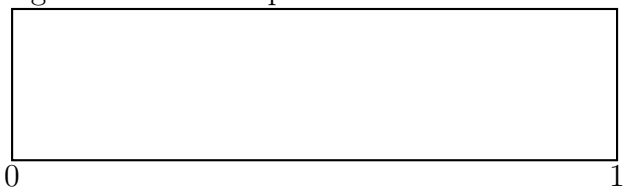



1 4 3
1 5 2
1 7
2 1 5
2 5 1
2 6
3 4 1
3 5
5 1 2
5 2 1
5 3

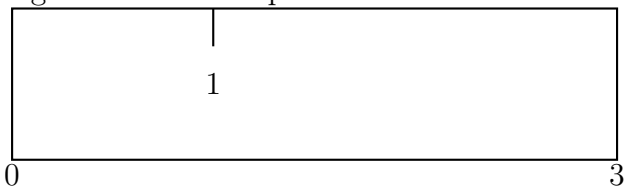

6
2
7
1
8

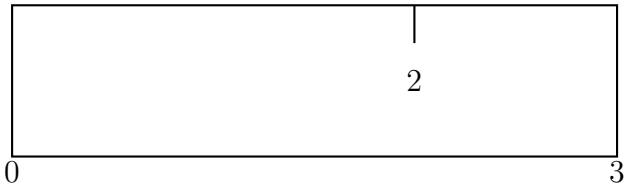
E. Reglas de Golomb óptimas

Reglas de Golomb óptimas de 2 marcas

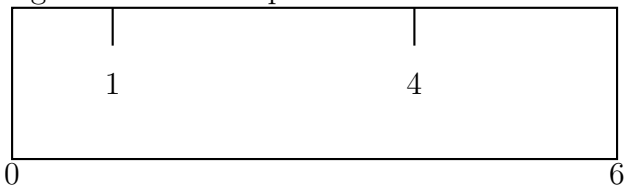


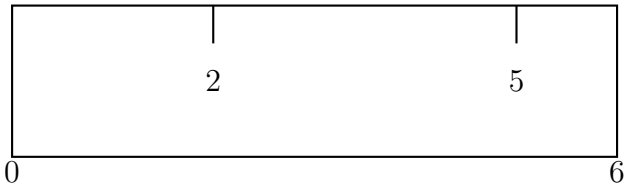
Reglas de Golomb óptimas de 3 marcas



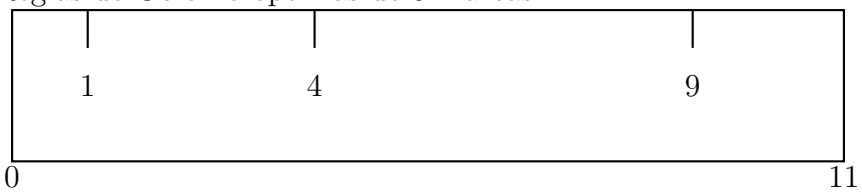


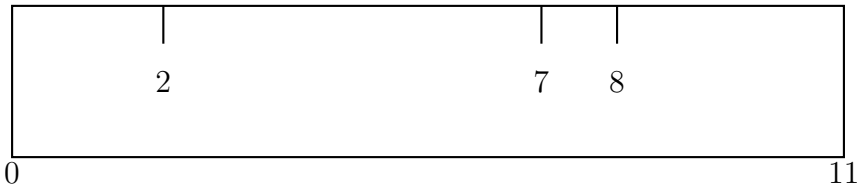
Reglas de Golomb óptimas de 4 marcas

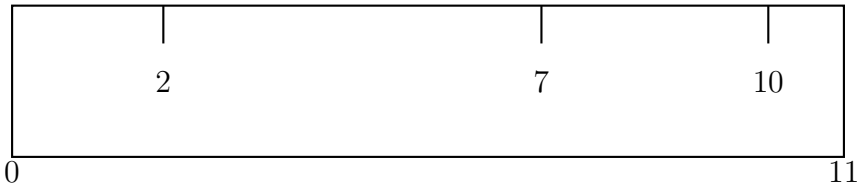


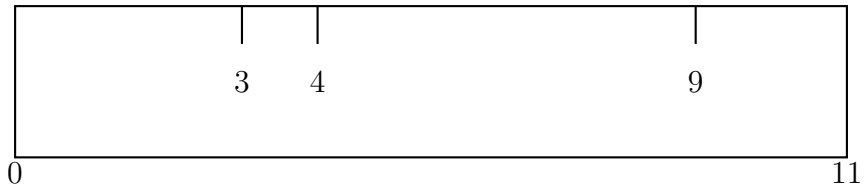


Reglas de Golomb óptimas de 5 marcas








3
4
9
0
11