Universidad de Valladolid
E.T.S Ingeniería Informática
Grado en Ingeniería Informática

Programación dinámica - Gerrymandering

Algoritmos y Computación

Curso 2018/2019

Universidad de Valladolid

Valdunciel Sánchez, Pablo

ÍNDICE GENERAL

1

INTRODUCCIÓN

En el presente documento se expone el problema del *Gerrymandering*, se analiza paso a paso la construcción e implementación de un algoritmo de **programación dinámica** para resolver dicho problema y, finalmente, se analiza el coste de dicho algoritmo.

1.1 ¿QUÉ ES *gerrymandering*?

Gerrymandering es un término que hace referencia a la manipulación de los distritos electorales de un territorio con el objetivo de conseguir un efecto concreto sobre los resultados electorales. Puede ser utilizado para mejorar o empeorar los resultados de un partido político o de un grupo social (religioso, lingüístico o étnico).

1.2 ORIGEN - UN POCO DE HISTORIA

El término *Gerrymander* es una combinación de dos palabras: *Gerry* y *salamander* (o salamandra en castellano).

Figura 1: Caricatura

En torno al año 1812, en Estados Unidos, el gobernador de Massachusetts, Elbridge Gerry, decidió hacer algo para que su partido, el Demócrata-Republicano, obtuviera la victoria en los distritos del norte y el oeste del estado de Massachusetts. Para ello, unificó todos los distritos problemáticos en uno solo, de modo que obtuvieran menos escaños en la legislatura.

Esta modificación de los distritos no pasó inadvertida para los periodistas de Massachusetts, que vieron como el nuevo distrito adquiría una inusual forma de salamandra (*salamander*). Fue así como surgió el juego de palabras *Gerry-mander*. El terminó se popularizó cuando se publicó la caricatura mostrada. El término *Gerrymandering* se refiere al proceso de hacer un *Gerrymander*.

2

EL PROBLEMA

Supongamos un Estado cuyo territorio se encuentra dividido en k **distritos** electorales. Cada uno de esos distritos está compuesto a su vez por un conjunto de **precintos**. Cada distrito electoral, D_i, tiene asignados un número de representantes, n_i , en el congreso o parlamento, de manera que el partido político que gane en el distrito i-ésimo obtendrá n_i representantes y el resto de partidos, independientemente de sus votos, no obtendrá ningún representante.

En este contexto, supongamos 2 distritos D_1 y D_2, y n precintos, siendo n par, con una población de *m* personas en cada precinto. Supongamos también que en cada precinto una persona sólo puede votar a dos partidos, A o B, sin la posibilidad de votar nulo. En cada precinto P_i se estima que A_i personas votarán al partido A, y por tanto, $B_i = m - A_i$ personas votarán al partido B (*Figura* 2).

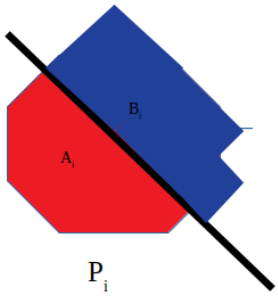


Figura 2: *Precinto P_i, con A_i votos al partido A y B_i votos al partido B*

Si fuéramos el dirigente del partido A nos interesaría que el número de personas que votan al partido A en el distrito D_1 y en el distrito D_2 fuera mayoría, para así obtener los representantes del distrito D_1 y del distrito D_2 , y que el partido B no obtuviera ningún representante.

La entradas de nuestro problema son por tanto:

1. **m**: número de personas en cada precinto.

2. **A_1,A_n,...,A_n:** las estimaciones de votos al partido A en *n* precintos.

3. Adyacencias entre los precintos.

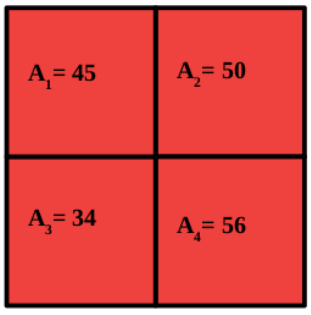
y buscamos una salida que consista en dos distritos D_1 y D_2 que verifican las siguientes condiciones:

1. Cada distrito tiene $\frac{n}{2}$ precintos, siendo **contiguos**.(Si creamos distritos electorales cuyo territorio no sea contiguo será difícil que no acabemos en el banquillo de los acusados.)

2. En cada distrito el partido A tiene mayoría, es decir, el número de votos al partido A en el distrito D_1, $A(D_1)$, y el número de votos al partido A en el distrito D_2, $A(D_2)$, es mayor que la mitad de votos posibles en cada distrito. En cada precinto hay m personas y en cada distrito hay $\frac{n}{2}$ precintos, por lo que en cada distrito hay $\frac{n*m}{2}$ personas. Por lo tanto, para obtener la mayoría
$$A(D_1) > \frac{m*n}{4}$$
y
$$A(D_2) > \frac{m*n}{4}$$

Ejemplo:

Supongamos que existen 4 precintos (n=4) con 80 personas en cada precinto (m=80). Las estimaciones de votos al partido A son: A_1 = 45, A_2 = 50, A_3 = 34 y A_4 = 56. Dado que cada precinto tiene 80 personas y cada distrito debe tener 2 precintos, para que el partido A tenga la mayoría de votos en un precinto $A(D_i)$ debe ser mayor que $\frac{m*n}{4} = \frac{80*4}{4} = 80$.



En esta situación podríamos agrupar los precintos de dos formas distintas de manera que los precintos en ambos distritos sean contiguos:
a)

- D_1: A_1 y A_3 - 79 votos - Minoría del partido A en D_1.
- D_2: A_2 y A_4 - 106 votos - Mayoría del partido A en D_2.

El partido A tiene la mayoría en uno de los dos distritos.

b)

- D_1: A_1 y A_2 - 95 votos - Mayoría del partido A en D_1.
- D_2: A_3 y A_4 - 90 votos - Mayoría del partido A en D_2.

El partido A tiene la mayoría en los dos distritos.

Por tanto, con los cuatro precintos dados existe la posibilidad de obtener un Gerrymander, que es la distribución b).

3

ALGORITMO

3.1 IDEA FUNDAMENTAL

La idea fundamental del algoritmo es decidir si un determinado distrito debe ser asignado a D_1 o a D_2. La asignación a uno u otro distrito dependerá de si el partido A tiene ya una mayoría en uno u otro.

Imaginemos que queda el último precinto por asignar, P_n. En este momento el estado de los distritos es el siguiente:

- D_1: **k** precintos asignados, **x** votos al partido A.
- D_2: **n-1-k** precintos asignados, **y** votos al partido A.

Las dos posibilidades son asignar el precinto P_n a D_1 o a D_2:

1. Asignado a D_1:
   - D_1: **k+1** precintos asignados, $\mathbf{x + A_n}$ votos al partido A.
   - D_1: **n-k-1** precintos asignados, **y** votos al partido A.
2. Asignado a D_2:
   - D_1: **k** precintos asignados, **x** votos al partido A.
   - D_1: **n-k-1** precintos asignados, $\mathbf{y + A_n}$ votos al partido A.

El estado del problema se caracteriza por 4 valores: nº precintos asignados a D_1, nº de votos al partido A en D_1, nº de precintos asignados a D_2, y nº de votos al partido A en D_2.

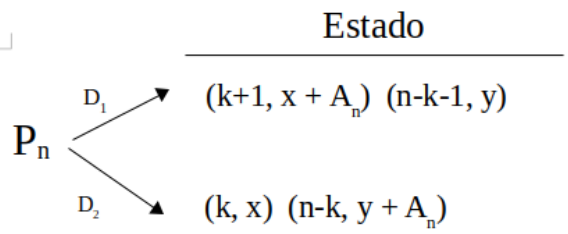


Figura 4: *Estado tras asignar el distrito P_n a D_1 o a D_2.*

Si analizamos los cuatro valores del estado nos damos cuenta de que el número de precintos asignados a D_2 no es necesario representarlo de forma explícita, ya que al trabajar únicamente con 2 distritos se puede obtener a partir del número de precintos asignados a D_1, **k**, y del número de precintos ya asignados, tanto a D_1 como a D_2, el cual podemos denotar por **j**. Con estos cuatro valores denotamos el estado del problema:

$$S_{j,k,x,y}$$

Un estado puede ser *true* o *false* en función de si es posible que exista, es decir, el estado $S_{j,k,x,y}$ es *true* sii existe una asignación de los primeros j precintos tal que:

- $|D_1| = k$
- $A(D_1) = x$
- $A(D_2) = y$

y *false* en caso contrario. Este parámetro S cobra sentido si consideramos qué es lo que ocurre cuando

$$S_{n,\frac{n}{2},\frac{m*n}{4}+1,\frac{m*n}{4}+1} = true$$

pues en ese caso existe una mayoría de votos del partido A en D_1 y en D_2 una vez se han asignado los **n** precintos.

Por lo tanto, si existe algún estado $S_{n,\frac{n}{2},x,y}$ que es *true* cuando $x > \frac{m*n}{4} + 1$ e $y > \frac{m*n}{4} + 1$ de manera que los precintos asignados a cada distrito son contiguos, se puede concluir que existe un **gerrymander**.

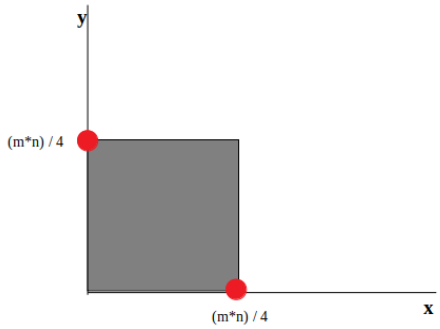
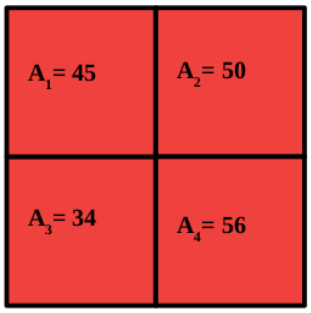


Figura 5: *Nos interesan los estados que sean ciertos y estén fuera de la zona gris.*

Ejemplo:

En el ejemplo anterior, aunque tanto $S_{4,2,95,90}$ como $S_{4,2,96,90}$ se encuentran fuera de la zona gris, $S_{4,2,95,90} = true$, y $S_{4,2,96,90} = false$ porque no es posible asignar los 4 precintos de manera que haya 96 votos al partido A en D_1 y 90 votos al partido A en D_2.



Programación dinámica

La clave para aplicar programación dinámica es encontrar que subestructura óptima del problema. Recordemos que si el estado $S_{j,k,x,y}$ es cierto existe una asignación para los **j primeros precintos** tal que D_1 tiene asignados **k precintos**, hay x votantes del partido A en D_1 y **y votantes del partido A** en D_2. De esta manera el estado $S_{j,k,x,y}$ puede haberse formado de dos opciones:

1. A partir del estado $S_{j-1,k-1,x-A_j,y}$ y asignando el precinto j al distrito D_1.
2. A partir del estado $S_{j-1,k,x,y-A_j}$ y asignando el precinto j al distrito D_2.

Por lo tanto, el estado $S_{j,k,x,y}$ será cierto sii $S_{j-1,k-1,x-A_j,y}$ o $S_{j-1,k,x,y-A_j}$ son ciertos:

$$S_{j,k,x,y} = \quad S_{j-1,k-1,x-A_j,y} \quad \vee \quad S_{j-1,k,x,y-A_j}$$

3.2 PSEUDOCÓDIGO

Función principal

A continuación, se muestra la parte principal del algoritmo de *gerrymandering*. Esta función, recibe como argumentos: un entero indicando un número par de precintos, **n**; un entero indicando la población de cada precinto, **m**; n enteros indicando los votos al partido A en cada uno de los n precintos, $A_1, A_2, ..., A_n$; y una matriz nxn que contiene las adyacencias entre los precintos, **ady[1..n][1..n]**. Esta función devolverá el *gerrymander* de los precintos si es posible obtenerlo, o *null* en su lugar.

```
funcion gerrymander (n, m, A1,A2, …, An, ady[1..n][1..n]) : gerrymander

    {Crear la tabla de estados s[n][n/2][m*n][m*n] y la tabla de
     resultados intermedios  c[n][n/2][m*n][m*n] }
    mayoría ← (m*n) / 4 + 1

    for j ← 1 to n
        for k← 0 to n/2
            for x← 0 to m*n
                for y← 0 to m*n
                    if  j = 0
                        if ( x == Aj  and k = 1   and   y = 0 )
                            s[j][k][x][y] ← true
                            c[j][k][x][y] ← 1
                        else if ( y == Aj  and  k = 0  and   x = 0)
                            s[j][k][x][y] ← true
                            c[j][k][x][y] ← 2
                        else
                            s[j][k][x][y] ← false
                    else
                        if ( x >= Aj  and   k >= 1   and  s[j-1][k-1][x-Aj][y] )
                            s[j][k][x][y] ← true
                            c[j][k][x][y] ← 1
                        else if ( y >= Aj  and   s[j-1][k][x][y-Aj])
                            s[j][k][x][y] ← true
                            c[j][k][x][y] ← 2
                        else
                            s[j][k][x][y] ← false

    for x ←mayoría to m*n
        for y ←mayoría to m*n
            if   s[n][n/2][x][y]
              d1, d2 ← getGerrymander2Distritos (n, A1,A2, …, An, c, x, y)
              if  gerrymanderValido (d1, d2, ady) then  return posibleGerrymander

    return null
```

Dentro de la función, podemos distinguir dos partes principales: la construcción de la tabla de estados y de la tabla de resultados, y la búsqueda de un estado $S_{n,\frac{n}{2},x,y}$ que es *true* cuando

$x > \frac{m*n}{4} + 1$ e $y > \frac{m*n}{4} + 1$ de manera que los precintos asignados a cada distrito son contiguos. Para ello, se buscan estados $S_{n,\frac{n}{2},x,y}$ que verifiquen $x > \frac{m*n}{4} + 1$ e $y > \frac{m*n}{4} + 1$, se construye el *gerrymander* y se comprueba que los precintos de ambos distritos verifiquen la condición de contigüidad. Si se encuentra un estado que verifica todas las condiciones se devuelve, y sino, se devuelve *null*.

Construcción del resultado

Esta función recibe como argumentos: la tabla de resultados intermedios, **c[1..n][0..k][0..m*n][0..m*n]**; los votos al partido A en los n precintos,**A_1, A_2, ..., A_n**; y dos enteros que indican el el x y el y del estado $S_{n,\frac{n}{2},x,y}$ que es *true* cuando $x > \frac{m*n}{4} + 1$ e $y > \frac{m*n}{4} + 1$, **xFinal** e **yFinal**. Estos dos últimos valores son necesarios porque en este caso no hay una única solución al problema, y por tanto no hay una única posición en la tabla **c** a partir de la que construir el resultado. Esta función devolverá el *gerrymander* de los precintos en dos distritos.

```
function getGerrymander2Distritos (c[1.. n][0..k][0..m*n][0..m*n],
                                   A1,A2, …, An, xFinal, yFinal) : gerrymander
    j← n -1
    k ← n/2
    x ← xFinal
    y ← yFinal
    d1 ← Ø
    d2 ← Ø

    while j > 0 do
        if c[j][k][x][y] = 1
            d1 ← d1 U { j }
            k ← k -1
            x ← x – Aj
        else
            d2 ← d2 U { j }
            y ← y – Aj

    return [d1, d2]
```

En cada posición de **C[j][k][x][y]** hay un **1** o un **2** que nos permite saber si el **precinto j** fue añadido al distrito 1 o al distrito 2, respectivamente, y añadirlo a dicho distrito. Por otro lado, restando los votos al partido A en el precinto, A_j, a x o a y, podemos obtener la posición del estado anterior, C[j-1][k-1][x-A_j][y] ó C[j-1][k][x][y-A_j]. Así, se recorre la tabla construyendo la solución hasta llegar a la posición C[0][1][A_1][0] ó C[0][0][0][A_1].

Verificación de la contigüidad de los precintos

Dentro de la tabla de estados s[j][k][x][y] puede haber varios estados que sean *true* y verifiquen $x > \frac{m*n}{4} + 1$ e $y > \frac{m*n}{4} + 1$; sin embargo, si aplicamos este algoritmo en la realidad para que el *gerrymander* generado no llame la atención de la prensa, es necesario que los precintos que forman cada distrito sean contiguos. Una vez construido el *gerrymander* a partir de la tabla de resultados intermedios, c[j][k][x][y], es necesario comprobar que se verifica la condición de contigüidad.

Esta función recibe como argumentos: un array de enteros indicando los precintos que forman D1, **d1[1..n/2]**; un array de enteros indicando los precintos que forman D2, **d2[1..n/2]**; y la matriz de adyacencias de los precintos, **ady[1..n, 1..n]**. Esta función devuelve **true** si tanto los precintos que forman D1 como los precintos que forman D2 son contiguos, y **false** en caso contrario.

```
funcion gerrymanderValido ( d1[1.. n/2], d2[1 .. n/2], ady[1..n, 1..n]) : boolean
        adyD1 ← Ø
        adyD2 ← Ø

        for each i in d1
            for j ← 1 to n
                if ady[i][j] = 1 then adyD1 ← adyD1 U { j }

        for each i in d2
            for j ← 1 to n
                if ady[i][j] = 1 then adyD1 ← adyD1 U { j }

        if d1 C adyD1  and  d2 C adyD2
            return true
        else
           return false
```

3.3 CÓDIGO

En la carpeta *codigo* hay presentes los siguientes archivos:

1. **Gerrymandering.java** - implementa el algoritmo de Gerrymander utilizando programación dinámica.
2. **Main.java** - implementa la interfaz de texto que permite utilizar la clase Gerrymandering.
3. **Gerrymandering.class** - bytecode de la clase Gerrymandering.java.
4. **Main.class** - bytecode de la clase Main.java.

5. **ejemplo01.txt** - fichero de texto que contiene un sencillo caso de 4 precintos para poder aplicar el algoritmo de Gerrymander.
6. **ejemplo01.pdf** - representación visual del ejemplo 1.
7. **ejemplo02.txt** - fichero de texto que contiene un caso algo más complicado con 12 precintos.
8. **ejemplo02.pdf**- representación visual del ejemplo 2.

Abriendo el terminal,y siendo la carpeta contenedora de estos archivos el directorio actual, se debe ejecutar el siguiente comando :

java Main

A continuación, el terminal pedirá la ruta de un fichero de texto (si el fichero está contenido en la misma carpeta que el fichero Main.class bastará con introducir el nombre de dicho fichero). El fichero deberá tener la estructura que se especifica en la documentación de la clase Main.java. Se debe introducir el nombre completo del fichero, incluyendo la extensión(.txt) y sin utilizar comillas.

java Main
> File route: ejemplo01.txt

Estructura de los ficheros de entrada

La estructura del fichero que contenga una determinada configuración de precintos debe ser la siguiente:

- Línea 1: número de precintos, **n**. Debe ser par y mayor que 0.
- Línea 2: población de cada precinto, **m**. Debe ser positiva.
- Línea 3: votos al partido A en cada. precinto,($\mathbf{A_1}$**,** $\mathbf{A_2}$**, ...,** $\mathbf{A_n}$). Cada cifra debe estar separada por un espacio.
- Líneas 4 a (4+n-1): matriz de adyacencia,**ady[1..n][1..n]**. Cada línea contendrá una fila de la matriz. En cada línea las columnas estarán separadas por un espacio. Cada posición de la matriz ady[i][j] representa si el precinto i-ésimo es contiguo al precinto j-ésimo ($ady[i][j] \leftarrow 1$) o no ($ady[i][j] \leftarrow 0$).

3.4 ANÁLISIS DE COSTE

Construcción del resultado

En primer lugar, la inicialización de las variables y de los conjuntos es Θ(cte).

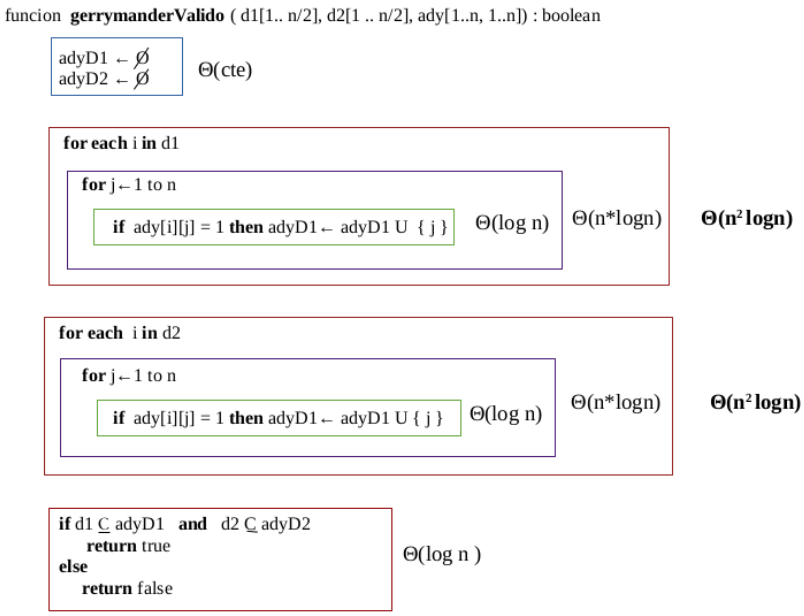
En segundo lugar, se recorren n posiciones en la tabla de resultados intermedios. En cada posición, la comprobación del valor almacenado y la modificación de las variables es Θ(1). En este caso, no se utiliza una implementación de conjunto (*TreeSet, HashSet, ..*), sino que se utiliza la clase *ArrayList*. La adición de n elementos a un ArrayList tiene un tiempo amortizado constante de O(n). Por lo que podemos concluir que la complejidad del bucle es Θ(n).

La complejidad de la construcción del *gerrymander* es, por tanto, **Θ(n)**.

```
function getGerrymander2Distritos (c[1.. n][0..k][0..m*n][0..m*n], A1,A2, …, An, xFinal, yFinal) : gerrymander
    j ← n -1
    k ← n/2
    x ← xFinal
    y ← yFinal
    d1 ← Ø
    d2 ← Ø                                              Θ(cte)
    while j > 0 do
        if c[j][k][x][y] = 1
            d1 ← d1 U { j }
            k ← k -1
            x ← x – Aj                                  Θ(cte)      Θ(n)
        else
            d2 ← d2 U { j }
            y ← y – Aj
    return [d1, d2]
```

Verificación de la contigüidad de los registros

Para comprender el análisis de coste de está función se debe tener en cuenta que la a implementación de conjunto utilizada (*TreeSet*) garantiza que la adición de un elemento, la eliminación de un elemento y la comprobación de si un conjunto pertenece a otro conjunto es Θ(log n).



En primer lugar, la inicialización de los conjuntos vacíos *adyD1* y *adyD2* es $\Theta(1)$.
En segundo lugar, para cada distrito:

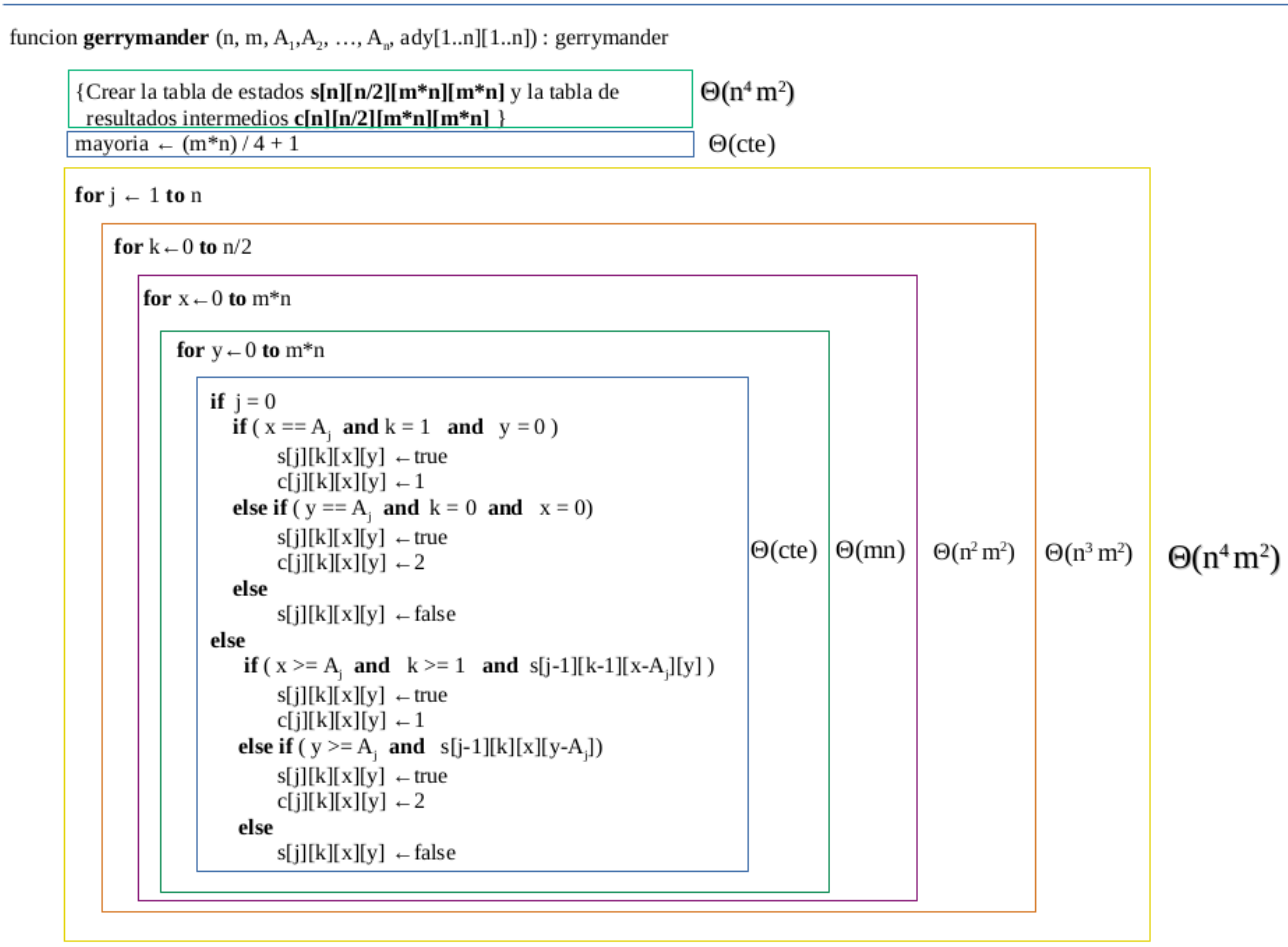
- Para cada precinto se añaden los precintos que son adyacentes al conjunto *adyD1* o *adyD2*, respectivamente (Θ(log n)), en base a la matriz de adyacencia. En el peor de los casos un precinto será adyacente a los n-1 precintos restantes, por lo que el coste para cada precinto es (Θ(n log n)).

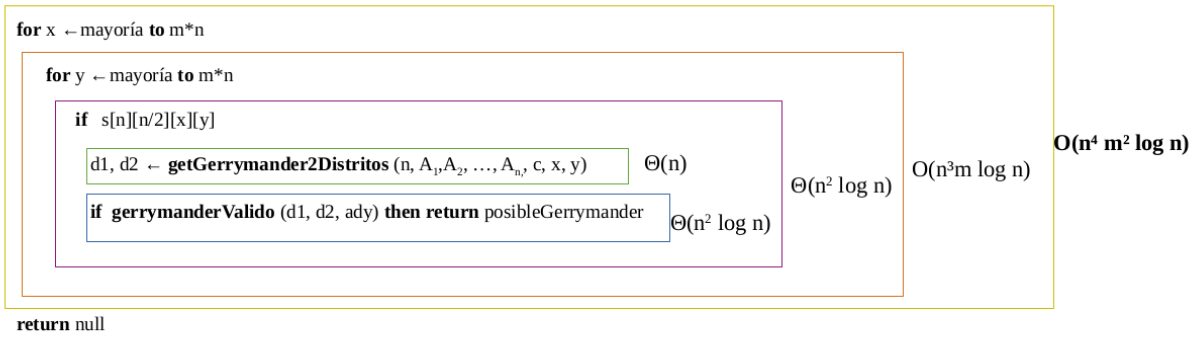
Cada distrito tiene $\frac{n}{2}$ precintos, por lo que el coste para cada distrito es $\frac{n}{2} * n * log(n)$, es decir, $\Theta(n^2 \log n)$. Por último, se realiza la comprobación de si *d1* está contenido en *adyD1* y si *d2* está contenido en *adyD2*, lo que tiene un coste *2*log n* es decir, Θ(log n).
El coste de la comprobación de la contigüidad es, por tanto, $\Theta(n^2 \log n)$.

Función principal

En primer lugar, la inicialización de la tabla de estados, *s[n][n/2][m*n][m*n]*, y de la tabla de resultados intermedios, *c[n][n/2][m*n][m*n]*, tiene un coste $\Theta(nm)$. La inicialización de la variable *mayoria* tiene un coste constante.

En segundo lugar, acceder a la tabla de estados y la tabla de resultados intermedios en cada iteración de los cuatro bucles anidados tiene un coste $\Theta(cte)$. El coste total de rellenar las dos tablas tiene un coste $\Theta(n^4m^2)$. Debemos tener en cuenta que pesar de que la inicialización de las tablas tiene el mismo coste asintótico que la ejecución de los bucles anidados, es evidente que la constante de proporcionalidad que multiplica al coste de la ejecución de los bucles será mayor.





Una vez se han rellenado las tablas de estados y de resultados intermedios, se recorren tres cuartas partes de los valores de x e y para j=n y k=$n/2$, es decir, se recorren aquellos estados que pueden constituir un *gerrymander* porque $x > \frac{m*n}{4} + 1$ e $y > \frac{m*n}{4} + 1$. Si un estado es *true*, se construye el *gerrymander* y se comprueba si verifica la restricción de contigüidad. Como ya hemos visto, el coste de la construcción del posible *gerrymander* es $\Theta(n)$ y el coste de la verificación de la contigüidad es $\Theta(n^2 \log n)$. El coste del *if* es, por tanto,$\Theta(n^2 \log n)$.

El cuerpo del *if* sólo se ejecutará para aquellos estados s[n][n/2][x][y] que sean *true*, y una vez que un estado s[n][n/2][x][y] sea true y verifique la contigüidad, no se volverá a ejecutar. El número de estados que s[n][n/2][x][y] con $x > \frac{m*n}{4} + 1$ e $y > \frac{m*n}{4} + 1$ que sean *true* depende de los valores de los votos al partido A en los distintos precintos Por un lado, es posible que no haya ningún estado *true* en ese rango de valores si el partido A tiene votos por debajo de la mayoría en casi todos o todos los precintos; y no se podrá hacer un *gerrymander*. Por otro lado, si el partido A tiene muchos votos, por encima de la mayoría, en todos los precintos, habrá un gran número de estados en ese rango que sean *true*, siendo posible contruir diferentes *gerrymanders*. El cálculo de este coste es demasiado complejo y no soy capaz de calcularlo, por lo que he decidido establecer la cota superior de coste considerando que el if se ejecuta en todas las iteraciones de bucles, es decir, en $n^4 m^2$ iteraciones.

Podemos concluir así que el coste total del algoritmo de gerrymander es $O(n^4 m^2 ogn)$.

BIBLIOGRAFÍA

3.5 INTRODUCCIÓN

- **Origen e historia** - https://es.wikipedia.org/wiki/Gerrymandering
- **Figura 1** - https://upload.wikimedia.org/wikipedia/commons/thumb/9/96/The_Gerry-Mander_Edit.png/200px-The_Gerry-Mander_Edit.png

3.6 EL PROBLEMA

- **Gerrymandering: Controversial Political Redistricting Explained**- https://www.youtube.com/\watch?v=Fm9hi1QkLVo
- https://www.washingtonpost.com/news/wonk/wp/2015/03/01/this-is-the-best-ex?noredirect=on&utm_term=.37b97a2d9eac

3.7 ANÁLISIS DE COSTE

- **ArryList Doc.** - https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html
- **TreeSet Doc.** - https://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html