Algoritmo de Hoshen-Kopelman

El Algoritmo de Hoshen-Kopelman se usa para identificación de imágenes. Por ejemplo, la siguiente es una imagen de `C++` realizada con "1" y " ". En nuestro caso, usaremos el algoritmo para identificar fragmentos de una red cuadrada bidimensional. Se dará una versión simple del algoritmo[1] .

```
11111111111
111               11          11
111               11          11
111           1111111111  1111111111
111               11          11
111               11          11
11111111111
```

(1) Poblamos la red

Primero poblamos una red cuadrada de $N \times N$ con 1 (sitio ocupado) o 0 (sitio vacío). La red será un "puntero" llamado `*red`. Asignamos una probabilidad de ocupación de sitio (variable `prob`). La asignación se realiza recorriendo la red del siguiente modo

```
for (i=0;i<n*n;i++) if (prob<myrandom(p)) *(red+i)=0 else *(red+i)=1;
```

nota: en C no es necesario abrir/cerrar llaves cuando se encierra una única sentencia.

(2) Clasificamos fragmentos

Un sitio ocupado de la red forma parte del mismo fragmento que otro que esté arriba, abajo, a izquierda o derecha. Estos se llaman "primeros vecinos". Los siguientes son ejemplos de sitios que forman un único fragmento.

```
11   10   11     010     100
00   10   01     111     111
                 010     001
```

Los sitios ocupados en diagonal se consideran "segundos vecinos" y correpsonden a fragmentos distintos. Los siguientes son ejemplos de sitios que forman distintos fragmentos.

[1] El algoritmo completa se encuentra en *Phys- Rev. B* 14(8) 3438-3445 (1976) y *J. Stat. Phys.* 19(3) 219-242 (1978)

```
10    01     010       100
01    10     101       010
               010       001
```

Para "identificar" fragmentos o clusters, debemos recorrer la red y "etiquetar" cada sitio ocupado según corresponda. El recorrido se realiza a lo largo del "puntero". Pero podemos imaginar que este puntero es un arreglo de $N \times N$, recorrido en el sentido de lectura. Las reglas de clasificación que debemos seguir son las siguientes:

(a) Para un dado sitio ocupado, si su vecino de "arriba" **e** "izquierda" están vacíos, entonces se inicia una nueva etiqueta.

(b) Para un dado sitio ocupado, si su vecino de "arriba" **o** "izquierda" (pero no simultáneamente; sólo uno de ellos!) están ocupados, entonces se copia la etiqueta del ocupado.

(c) Para un dado sitio ocupado, si su vecino de "arriba" **e** "izquierda" están ocupados, entonces puede haber un "conflicto" de etiquetas que debe ser resuelto.

nota: para el sito `*(red+i)` , su vecino de "arriba" es el sito `*(red+i-n)` y su vecino a "izquierda" es `*(red+i-1)`.

El siguiente es un ejemplo de clasificación de etiquetas.

```
11011        22033
00100        00400
01100   -->  04400
10011        60077
10001        60007
```

El sitio $(0, 0)$ no tiene conflictos por ser el primero. A este sitio le cabe aplicar la regla (a). Así que se le asigna la etiqueta "2" (no usamos el "1" como etiqueta porque ésta se usa para inidicar que el sitio está ocupado). La asignación es simple: `*(red+0)=2`. Por el momento esta es la etiqueta de mayor valor (la única en verdad) y la guardaremos en la varible `frag=2` para futuros usos.

Necesitamos crear un "registro de etiquetas" para saber a qué sitio le correspondió el "2". Este registro lo armaremos como un "puntero" de largo $N \times N$ y lo llamamos `*clase`. Inicialmente a este puntero le asignamos los valores

```
*clase = 0 1 2 3 4 5 6 7 8 9 10 11 12...
```

Entonces si quiero saber qué etiqueta tiene `*(red+0)` hago `s = *(red+0)` y busco su valor en `*(clase+s)`. Por el momento esto parece redundante, pero servirá para solucionar conflictos entre etiquetas.

El sitio $(0, 1)$ está ocupado, lo mismo que su vecino izquierdo. La etiqueta de su vecino izquierdo es `s = *(red+0)` (y vale "2"). Al sitio actual `*(red+1)` le cabe aplicar la regla (b). Entonces le asignamos `*(red+1)=s`. Y así seguimos recorriendo la red.

Cuando llegamos al sitio $(2, 2)$ (recordar que la numeración de filas y columnas comienza con cero), nos encontramos con el primer conflicto. Este sitio tiene ocupado su vecino superior e izquierdo. Probablemente el vecino izquierdo se clasificó antes como "5", según la regla (b). Es decir,

```
00400       00400
05100  --> 04400
```

y ahora debe corregirse a "4" dado que pertenece al mismo fragmento que el $(1, 2)$. La etiqueta "5" ya no será válida, y lo dejaremos indicado en el registro como `*(clase+5)=-4`. Esto significa lo siguiente: cuando una etiqueta no es válida, le ponemos signo negativo y el valor con el que fue re-clasificado. En este caso es se re-clasificó a "4", y ponemos "-4" para indicar cuál es el valor nuevo.

Este mecanismo de registración es muy bueno porque permite "rescatar" la etiqueta verdadera fácilmente. Por ejemplo, una etiqueta podría haberse corregido muchas veces a lo largo del recorrido, así que haciendo el siguiente recorrido inverso

```
while (*(clase+s)<0)  s=-(*(clase+s));
```

llegamos al valor de `s` verdadero.

Resumiendo:

Si no hay conflicto entre vecinos y la etiqueta del vecino ocupado es `s`, entonces procedemos del siguiente modo:

```
if (s)        /*funcion actualizar*/
   {
   while (*(clase+s)<0)  s=-(*(clase+s));
   *(red+i)=s;
   *(clase+s)=s;
   }
else
   {
   *(red+i)=frag;
   *(clase+frag)=frag;  // por seguridad!
   frag++;
   }
```

Si ambos vecinos (vertical `s1` y horizontal `s2`) están ocupados y tienen etiquetas diferentes, el sito `*(red+i)` tomará la etiqueta de menor valor entre ambas (la menor de las etiquetas *verdaderas*!!!). Pero debemos dejar anotado esto, porque ambas etiquetas corresponden al mismo fragmento. Por ejemplo, si `min=minimo(s1,s2)` y `max=maximo(s1,s2)` (etiqueta de menor valor y de más valor, respectivamente),

```
/*funcion etiqueta_falsa*/

while (*(clase+s1)<0) s1=-(*(clase+s1)); /*etiqueta s1 verdadera*/
while (*(clase+s2)<0) s2=-(*(clase+s2)); /*etiqueta s2 verdadera*/

min=minimo(s1,s2);
max=maximo(s1,s2);

*(red+i)=min;
*(clase+min)=min;

if (min<max) *(clase+max)=-min; else *(clase+max)=min;
```

Así procedemos con cada sitio de la red. Finalizado el recorrido, hay que barrer toda la red para corregir cualquier etiqueta inválida que hubiere quedado. La corrección es del siguiente modo.

```
for(i=0;i<n*n;i++)
  {
    s=*(red+i);
    while (*(clase+s)<0) s=-(*(clase+s));
    *(red+i)=*(clase+s);
  }
```