Pepito y su Balón

Pepito y su Balón

Por: Bruno Recasens Urzi

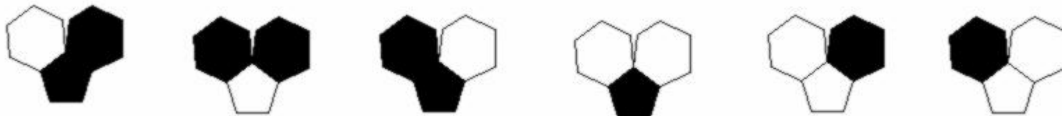
Tutor: Ginés Garcia Mateos

Prática optativa: reto del Balón

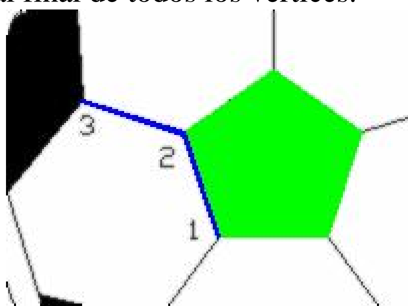
Analisis

A mi me dan una lista de números que se corresponden al contorno de una región del balón. El problema principal de este ejercicio ha sido encontrar una forma de averiguar cual es ese contorno. El segundo problema es encontrar la región que tiene en su interior, y contar las correspondientes caras, para determinar el número de caras de cada color.

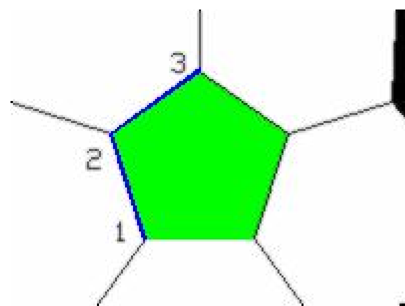
Para encontrar el contorno, debido a la gran simetría del grafo en cuestión, utilizo tres supuestos que resumen todas las posibles posiciones del contorno. Primero que nada, supongo que el primero de los números que se me pasa esta en un vértice determinado. Esto no implica ningún problema debido a que todos los vértices tienen dos hexágonos y un pentágono, en igual disposición con respecto a los demás. Este primer número puede ser un 1 o un 2. En caso de ser un 1, habría tres polígonos posibles que serían verdes, y dos que no. En caso contrario, habrían tres formas posibles de conseguir dos polígonos verdes y otro que no lo es. Estas son las tres primeras suposiciones que hago, para encontrar el camino real, de forma que debo revisar los tres caminos para encontrar el verdadero.



Dado uno cualquiera de los supuestos anteriores resulta fácil seguir el contorno de la parte para la que ya tengo determinado el color. En el siguiente vértice del contorno existen dos caminos posibles para continuar, uno sería teniendo dos verdes en el susodicho vértice, y otro sería teniendo uno solo. Puesto que esta información es la que se mete por parámetro, es fácil y rápido reconocer el camino real. Además, se sabrá si es el camino válido si se llega al vértice de partida al final de todos los vértices.



Para dos verdes en el vértice 2.
Se gira a la izquierda.



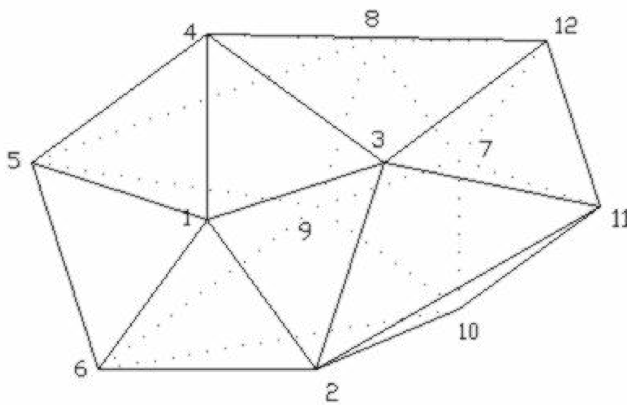
Para un verde en el vértice 2.
Se gira a la derecha.

Una vez encontrado el contorno resulta clara la forma de proceder para encontrar los vértices interiores y las caras verdes de cada tipo.

Aunque a primera vista esta es una buena solución se complica un poco a la hora de implementarla, esto no sería más que una aproximación a la solución hecha a mano, con un balón en tres dimensiones delante. Para nuestro caso, las cosas se complicaran un poco, como veremos en la fase de diseño.

Diseño

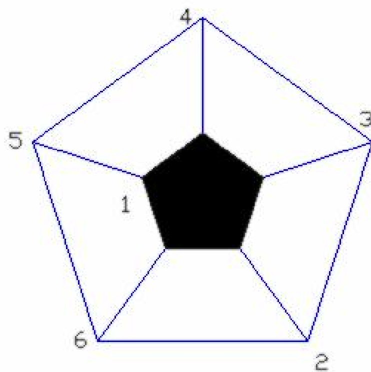
Lo primero y esencial es tener un balón para hacer los recorridos, y saber donde estamos a medida que recorremos el contorno, para saber si se ha llegado al inicio. Pero debido a que existen 60 vértices cada uno con tres aristas, aunque sería posible hacer a mano la lista de adyacencias, esto resulta un trabajo muy costoso, ya que es muy difícil incluso dibujar en papel el balón para numerar todos los vértices.
La posibilidad que se me ha ocurrido es simplificar el grafo de la forma que sigue. En vez de usar un grafo como el balón, he usado otro que me ayuda a simular el movimiento por el mismo. Este gafo representa los pentágonos, de forma que tiene 12 vértices, y cada pentágono se conecta con los 5 que tiene más cercanos. De forma que en cada arista hay dos vértices del grafo original. El número de aristas de este grafo es 30, con dos vértices simulados cada una, da un total de 60 vértices simulados, que es el número total.

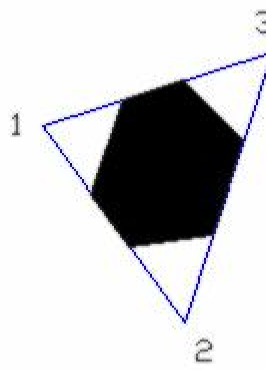


Grafo simplificado

```
int MiGrafo[13][5]=
{{0,  0,  0,  0,  0},
 {6,  5,  4,  3,  2},
 {1,  3, 11, 10,  6},
 {1,  4, 12, 11,  2},
 {1,  5,  8, 12,  3},
 {1,  6,  9,  8,  4},
 {1,  2, 10,  9,  5},
 {8,  9, 10, 11, 12},
 {7, 12,  4,  5,  9},
 {7,  8,  5,  6, 10},
 {7,  9,  6,  2, 11},
 {7, 10,  2,  3, 12},
 {7, 11,  3,  4,  8}};
```

Esta sería la lista de adyacencias del grafo nuevo, ordenadas en el sentido de las agujas del reloj. Así, el vértice 1 esta conectado en orden con 6, 5, 4, 3 y 2. Como cada arista tiene dos vértices simulados, estos se representan mediante pares ordenados, de forma que 1-2 y 2-1 son los dos vértices simulados que están en la arista que conecta a 1 y 2. También cabe destacar que los vértices simulados 1-2, 1-3, 1-4, 1-5 y 1-6, son los 5 puntos que forman uno de los pentágonos, y 1-2, 2-1, 2-3, 3-2, 3-1 y 1-3 forman uno de los hexágonos. Esto significa que desde 1-2, los tres vértices simulados que están adyacentes son 2-1, 1-6 y 1-3. Esto es, en general, para A-B los adyacentes son B-A, A-siguiente(B) y A-anterior(B), donde siguiente y anterior son los correspondientes según el sentido de las agujas del reloj, para el vértice A. Esto es importante entenderlo, puesto que define la forma de moverse a través del grafo simulado, usando el grafo simplificado que he ideado.

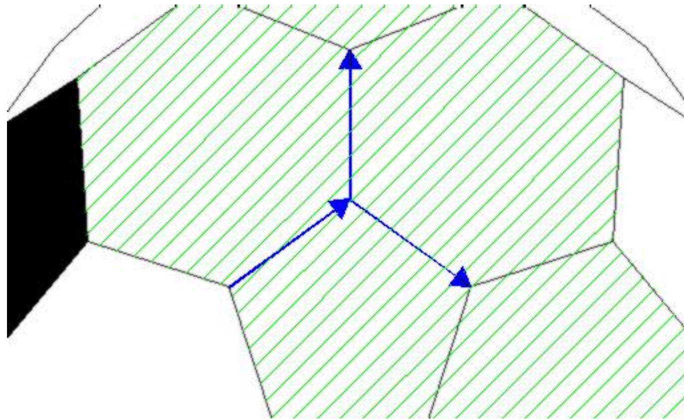




Pentágono simulado *Hexágono simulado*

Para moverme por el contorno conociendo si hay 1 o 2 verdes, me hacen falta dos cosas. Una es el vértice actual y la otra el vértice anterior. Además debo saber de que forma me estoy moviendo por el contorno. Yo he decidido siempre moverme por el contorno en el sentido de las agujas del reloj, por lo que siempre tengo a la derecha la región verde del balón, y a la izquierda la región no pintada. De esta forma el vértice actual y el anterior me definen una arista por la que he venido, quedando las otras dos para hacer la elección del camino a proseguir en función de la entrada del programa, como ya he dicho antes. De esta forma se deberá continuar por el mismo polígono si en el vértice solo hay un verde, o lo que es lo mismo, por la arista de la derecha, y en caso de haber dos verdes se cambiara de polígono, siguiendo por la arista de la izquierda.

Ahora me falta, dado el contorno, decir cuales son los polígonos que tiene en su interior. Esto me significa un nuevo problema, como representar cada polígono. Para el caso de los pentágonos es fácil, puesto que mi grafo representa en cada vértice un de estos, pero para los hexágonos ya no es tan trivial. He decidido representarlo mediante un trío, los tres pentágonos que tiene alrededor, puesto que cada hexágono tiene exactamente tres pentágonos alrededor, y entre tres pentágonos contiguos, solo hay un hexágono. Además me es muy fácil encontrar los pentágonos. El problema de esta representación es que no puedo hacer una tabla de tres variables, puesto que el desperdicio seria demasiado grande. Así que mi método para guardar los hexágonos es usar una lista, que contiene estos tres valores, y admite hasta 20 polígonos. Esto implica que cada vez que se añada un nuevo hexágono, se debe revisar toda la lista para ver que no esta ya puesto.
El algoritmo para encontrar los vértices internos, y sus polígonos es muy simple, primero marco todos los nodos del contorno, y después uso cada nodo en que halla dos verdes para entrar en el interior de la región, a un nodo supuestamente no marcado, y a partir de este, busco por cada uno de los nodos que tiene alrededor, marcando cada uno por los que paso, y no rehaciendo los ya marcados. Esto se repite para todos los vértices del contorno que tienen dos verdes, ya que no tienen porque estar unidas las regiones internas no marcadas. Puesto que me he marcado todos los vértices del contorno, simplemente haciendo que la recursividad pare en los vértices ya marcados (si la recursividad no se hace, entonces no marca sus 5 vértices de alrededor), ya consigo que los vértices por los que se pasa sean solo de la región verde.



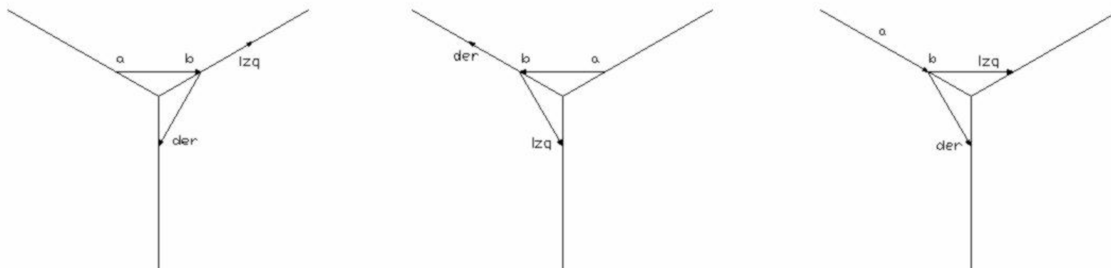
Se entra en la región a través del vértice con dos verdes y de ahí se buscan los demás.

Con esto consigo pasar por todos los vértices de la región. Si cada vez que paso por uno de los vértices del contorno, puesto que conozco cuales son los polígonos en verde (siempre el de la derecha es verde, y el de la izquierda no), y en los vértices internos también (todos los polígonos de alrededor son verdes), puedo marcar los polígonos, encontrando así una solución al problema.

Una vez que ya sé representar la información, y tengo los algoritmos para utilizarla solo me falta crear funciones que las implementen.
Las funciones que he utilizado son dos. La primera es la función `pertenece`, que comprueba si la secuencia que números es válida, y cuando encuentra la solución devuelve cual de los supuestos que antes mencione que serían necesarios, es el correcto.
La otra función se llama `cuantos`, y dado el verdadero supuesto, empieza a marcar los nodos y los polígonos, según el algoritmo anterior.

Implementación

Tengo dos funciones que implementar, `pertenece` y `cuantos`.
La primera hace uso de otras dos funciones, que detallaré ahora. `Indice` devuelve el valor del índice dentro de `MiGrafo` en que se encuentra el vértice `numero`, para el vértice `nodo`. Esto se hace mediante un simple bucle `for`, que busca en las 5 posiciones posibles. La otra función es `perteneceaux`, que sigue el contorno, dado el supuesto de que se parte del vértice simulado `b1-b2`, y el vértice anterior a este `a1-a2`. En `numeros` se pasan todos los números que hay el vértices, en la forma que se pasaron por entrada estándar. El algoritmo que se sigue para cada cambio es el descrito anteriormente. Para cada entrada de `numeros`, se sigue por la derecha si es un 1, y por la izquierda si es un 2. Para cada uno de los casos en que pueden estar combinados dos vértices, el cálculo de derecho e izquierdo es el que sigue:



El procedimiento `pertenece` solo pretende revisar cada uno de los posibles supuestos. Estos son partir de 1-2 y llegar a 2-1; partir de 1-2 y llegar a 1-3; y partir de 1-2 y llegar a 1-6. Se ve que estos supuestos son los equivalentes a los descritos al principio, ya que se parte del vértice 1-2, y se dirige hacia cada una de las tres posibles direcciones.

El procedimiento `cuantos` hace el mismo recorrido que el procedimiento anterior, ya teniendo el camino correcto, y en dos etapas. En una primera etapa se marcan todos los vértices del contorno, y se agregan los polígonos de la derecha. Y en la segunda etapa se entra en la región interior a través de los nodos que tiene dos verdes. Para marcar los vértices se usa una tabla, `tablanodos`, que tiene para cada uno de los vértices de `MiGrafo` las 5 adyacencias posibles, sabiendo así que si tiene un 1 esta marcada y si tiene un 0 no lo esta. Para los pentágonos se usa `tablapen`, que tiene una posición para cada vértice de `MiGrafo`. Y por último esta la lista de hexágonos en `tablahex`. Este procedimiento hace uso de otros tres nuevos: `marcarhex`, `marcar` y `marcartodos`. El procedimiento `marcarhex` marca un hexágono, si no esta marcado ya. El procedimiento `marcar` marca, dados dos vértices, el polígono que tiene a su derecha. Y el procedimiento `marcartodos` marca todos los vértices no marcados, y los polígonos de su alrededor, de forma recursiva.

Código

```
#include "grafo.h"

int MiGrafo[13][5]=          // grafo que utilizo
{{0,  0,  0,  0,  0}, // nulo, no es valido
 {6,  5,  4,  3,  2}, // 1
 {1,  3, 11, 10,  6}, // 2
 {1,  4, 12, 11,  2}, // 3
 {1,  5,  8, 12,  3}, // 4
 {1,  6,  9,  8,  4}, // 5
 {1,  2, 10,  9,  5}, // 6
 {8,  9, 10, 11, 12}, // 7
 {7, 12,  4,  5,  9}, // 8
 {7,  8,  5,  6, 10}, // 9
 {7,  9,  6,  2, 11}, // 10
 {7, 10,  2,  3, 12}, // 11
 {7, 11,  3,  4,  8}};// 12

// indice en MiGrafo para numero en nodo
int indice(int nodo, int numero){
       for (int i=0; i<5; i++)
              if (MiGrafo[nodo][i]==numero)
                     return i;
       return -1;
}

// busca para las posiciones de inicio pasadas por parametro
// devuelve true si el final se llega al mismo sitio, se hace un ciclo
bool perteneceAux(int* numeros, int ae1, int ae2, int be1, int be2){
       int a1,a2,b1,b2,c1,c2;
       a1=ae1;
       a2=ae2;
       b1=be1;
       b2=be2;
       for(int i=0; numeros[i]!=0; i++){    // revisa cada entrada
              if ((a1==b2) && (a2==b1)){    // tipo 1
                     c1=b1;
                     if (numeros[i]==1)     // derecha
                            c2=MiGrafo[c1][(indice(b1,b2)+4)%5];
                     else          // izquierda
                            c2=MiGrafo[c1][(indice(b1,b2)+1)%5];
              }
              else if ((a1==b1) && ((indice(b1,b2)-indice(a1,a2)+5)%5 == 1)){    // tipo 2
                     if (numeros[i]==1){    // derecha
                            c1=b1;
                            c2=MiGrafo[c1][(indice(b1,b2)+1)%5];
                     }
                     else {         // izquierda
                            c1=b2;
                            c2=b1;
                     }
              }
              else if ((a1==b1) && ((indice(a1,a2)-indice(b1,b2)+5)%5 == 1)){    // tipo 3
                     if (numeros[i]==1){    // derecha
                            c1=b2;
                            c2=b1;
                     }
                     else {         // izquierda
                            c1=b1;
                            c2=MiGrafo[c1][(indice(b1,b2)+4)%5];
                     }
              }
              a1=b1; // pasa al siguiente
              a2=b2;
              b1=c1;
              b2=c2;
       }
```

```
        if ((a1==ae1)&&(a2==ae2)&&    // comprueba que se llegue al principio
                (b1==be1)&&(b2==be2))
                return true;
        else
                return false;
}

int pertenece(int* numeros){  // comprueba cada una de las tres posibilidades
        if (perteneceAux(numeros,1,2,2,1))
                return 1;
        else if (perteneceAux(numeros,1,2,1,6))
                return 2;
        else if (perteneceAux(numeros,1,2,1,3))
                return 3;
        return 0;
}

// marca el hexagono correspondiente a los tres
// pentagonos a, b y c que tiene alrededor
void marcarhex(int tablahex[20][3],int a, int b, int c){
        if (a>b){int d=a;a=b;b=d;}    // los ordena
        if (b>c){int d=b;b=c;c=d;}
        if (a>b){int d=a;a=b;b=d;}
        int i;
        for (i=0; tablahex[i][0]!=0; i++)    // si ya esta se sale
                if ((tablahex[i][0]==a)&&
                        (tablahex[i][1]==b)&&
                        (tablahex[i][2]==c))
                                return;
        if (i<20){     // sino lo coloca
                tablahex[i][0]=a;
                tablahex[i][1]=b;
                tablahex[i][2]=c;
        }
}

// marca el poligono de la derecha
void marcar(int tablaper[13],
                        int tablahex[20][3],int a1,int a2,int b1,int b2)
{
        if ((a1==b2) && (a2==b1)) // tipo 1
                marcarhex(tablahex,a1,a2,MiGrafo[a1][(indice(a1,a2)+1)%5]);
        else if ((a1==b1) && ((indice(b1,b2)-indice(a1,a2)+5)%5 == 1)) // tipo 2
                tablaper[a1]=1;
        else if ((a1==b1) && ((indice(a1,a2)-indice(b1,b2)+5)%5 == 1)) // tipo 3
                marcarhex(tablahex,a1,a2,b2);
}

// marca recursivamente lo que no este marcado ya
void marcartodos(int tablaper[13],
                                int tablahex[20][3],
                                int tablanodos[13][5],int a1,int a2)
{
        if (tablanodos[a1][indice(a1,a2)]==0){       // condicion de salida
                tablanodos[a1][indice(a1,a2)]=1;     // marca el vertice
                tablaper[a1]=1;                              // marca los poligonos
                marcarhex(tablahex,a1,a2,MiGrafo[a1][(indice(a1,a2)+1)%5]);
                marcarhex(tablahex,a1,a2,MiGrafo[a1][(indice(a1,a2)+4)%5]);
                marcartodos(tablaper,tablahex,tablanodos,a2,a1);    // recursividad para los tres
vertices de alrededor
                marcartodos(tablaper,tablahex,tablanodos,a1,MiGrafo[a1][(indice(a1,a2)+1)%5]);
                marcartodos(tablaper,tablahex,tablanodos,a1,MiGrafo[a1][(indice(a1,a2)+4)%5]);
        }
}
```

```
// devuelve cuantos hay de cada tipo
void cuantos(int* numeros, int forma, int& pen, int& hex, int& ver){
	int tablaper[13];
	int tablahex[20][3];
	int tablanodos[13][5];
	int a1,a2,b1,b2,c1,c2;
	for (int i=0; i<13; i++)
		tablaper[i]=0;
	for (int i=0; i<20; i++)
		tablahex[i][0]=0;
	for (int i=0; i<13; i++)
		for (int j=0; j<5; j++)
			tablanodos[i][j]=0;
	pen=12;// inicializados como si no hubiera ningun verde
	hex=20;
	ver=0;
	a1=1;
	a2=2;
	switch (forma){		// calcula el inicicio en funcion de la forma devuelta por
pertenece
		case 0:
			return;
			break;
		case 1:
			b1=2;
			b2=1;
			break;
		case 2:
			b1=1;
			b2=6;
			break;
		case 3:
			b1=1;
			b2=3;
			break;
	}
	for (int k=0; k<2; k++){	// ejecuta dos veces todo
		for(int i=0; numeros[i]!=0; i++){	// para cada entrada
			if (k==0){	// primero marca el borde y el poligono de la derecha
				tablanodos[a1][indice(a1,a2)]=1;
				marcar(tablaper,tablahex,a1,a2,b1,b2);
			}
			// en el segundo si hay un dos entra en la region y llamma a la recursion
			else if (numeros[i]==2){
				if ((a1==b2) && (a2==b1))	// tipo 1
					marcartodos(tablaper, tablahex, tablanodos, b1,
MiGrafo[c1][(indice(b1,b2)+4)%5]);
				else if ((a1==b1) && ((indice(b1,b2)-indice(a1,a2)+5)%5 == 1))
	// tipo 2
					marcartodos(tablaper, tablahex, tablanodos, b1,
MiGrafo[c1][(indice(b1,b2)+1)%5]);
				else if ((a1==b1) && ((indice(a1,a2)-indice(b1,b2)+5)%5 == 1))
	// tipo 3
					marcartodos(tablaper, tablahex, tablanodos, b2, b1);
			}
			if ((a1==b2) && (a2==b1)){	// tipo 1
				c1=b1;
				if (numeros[i]==1)	// derecha
					c2=MiGrafo[c1][(indice(b1,b2)+4)%5];
				else		//izquierda
					c2=MiGrafo[c1][(indice(b1,b2)+1)%5];
			}
			else if ((a1==b1) && ((indice(b1,b2)-indice(a1,a2)+5)%5 == 1)){	// tipo
2
				if (numeros[i]==1){	// derecha
					c1=b1;
					c2=MiGrafo[c1][(indice(b1,b2)+1)%5];
				}
				else {		//izquierda
					c1=b2;
					c2=b1;
				}
			}
```

```
                        else if ((a1==b1) && ((indice(a1,a2)-indice(b1,b2)+5)%5 == 1)){    // tipo
3
                                if (numeros[i]==1){    // derecha
                                        c1=b2;
                                        c2=b1;
                                }
                                else {          //izquierda
                                        c1=b1;
                                        c2=MiGrafo[c1][(indice(b1,b2)+4)%5];
                                }
                        }
                        a1=b1; // pasa al siguiente
                        a2=b2;
                        b1=c1;
                        b2=c2;
                }
        }
        // cuenta el numero de poligonos de cada tipo encontrados que son verdes
        for (int i=0; (i<20) && (tablahex[i][0]!=0); i++){
                ver++;
                hex--;
        }
        for (int i=0; (i<13);i++){
                if (tablaper[i]!=0){
                        ver++;
                        pen--;
                }
        }
}


//////////////////////
// funcion principal //
//////////////////////
int main (){
        int* numeros;
        int pen,hex,ver,max;
        cin >> max;
        numeros=new int[max+1];
        for (int i=0; i<max; i++)
                cin >> numeros[i];
        numeros[max]=0;
        cuantos (numeros,pertenece(numeros),pen,hex,ver);
        cout << pen << " " << hex << " " << ver << "\n";
        return 0;
}
```