

```
1: //////////////////////////////////////
2: //
3: // Fundamentos de Programación
4: // ETS Informática y Telecomunicaciones
5: // Universidad de Granada
6: // Departamento de Ciencias de la Computación e Inteligencia Artificial
7: // Autor: Juan Carlos Cubero
8: //
9: //////////////////////////////////////
10:
11: // Examen 2021 Enero. Palisimétrica o Lapíndrome
12:
13: /*
14: Explicación de la solución:
15:
16:     Cómo calcular fin_izda e ini_dcha:
17:
18:         mitad = utilizados/2;
19:         fin_izda = mitad - 1;
20:
21:         if (utilizados % 2 == 0)
22:             ini_dcha = mitad;
23:         else
24:             ini_dcha = mitad + 1;
25:
26: El procesamiento de la parte izquierda y derecha lo podemos
27: hacer de dos formas -a) y b)- Ambas son completamente válidas:
28:
29: a) Podemos trabajar con dos secuencias izda, dcha
30:     (obviamente, serán locales al método EsLapindrome):
31:
32:     izda = Subsecuencia(0, fin_izda)
33:     dcha = Subsecuencia(ini_dcha + 1 , utilizados - 1)
34:
35:     for (int i=0 ; i<=fin_izda && es_lap; i++)
36:         char a_buscar = izda.Elemento(i);
37:
38:         es_lap = izda.NumOcuurrencias(a_buscar)
39:             ==
40:             dcha.NumOcuurrencias(a_buscar);
41:
42: Es obvio que NumOcuurrencias es un método necesario para
43: así no repetir el código que cuenta el número de ocurrencias
44: de un carácter en una secuencia.
45:
46: b) En vez de construir las secuencias izda y dcha
47:     podemos definir el método NumOcuurrencias
48:     para que trabaje sólo en una parte de la secuencia.
49:     Para ello, basta definirlo pasándole como parámetros
50:     los índices inicial y final que delimitan la parte
51:     de la secuencia en la que voy a buscar:
52:
53:     for (int i=0 ; i<=fin_izda && es_lap; i++)
54:         es_lap = NumOcuurrencias(v[i], 0, fin_izda)
55:             ==
56:             NumOcuurrencias(v[i], ini_dcha, utilizados-1);
57:
58: Independientemente de si es a) o b), para no volver a contar
59: la frecuencia de un carácter que ya se hubiese contado anteriormente,
60: podemos hacer lo siguiente:
61:
62: 1) O bien usamos un vector de bool en el que el índice
63:     de la componente a modificar se obtiene a partir
64:     del orden del carácter actual
65:
66:     const int NUM_CAR = 256;
67:     bool procesados[NUM_CAR];
68:
69:     procesados[0] será true si 'a' ha sido procesada
70:     procesados[1] será true si 'b' ha sido procesada
71:     ...
72:
73:     'a' -> 0
74:     'b' -> 1
75:     ...
76:
77: En general:
78:
79:     car -> car - 'a'
80:
81: 2) O bien usamos una secuencia de caracteres que almacene los
82:     caracteres que ya hayan sido procesados
83:
84:     SecuenciaCaracteres procesados;
85:
86: Si ya han sido procesadas la 'g' y la 'k',
```

```

87:     la secuencia procesados contendrá {'g', 'k'}
88:
89:     Podemos combinar a) y b) con 1) y 2) como queramos.
90:     Por ejemplo, la combinación de a) con 1) sería:
91:
92:     izda = Subsecuencia(0, fin_izda)
93:     dcha = Subsecuencia(ini_dcha + 1 , utilizados - 1)
94:
95:     for (int i...
96:         char a_buscar = izda.Elemento(i);
97:         int indx = a_buscar - 'a';
98:
99:         if (!procesados[indx])
100:             es_lap = izda.NumOcurrencias(a_buscar)
101:                 ==
102:                 dcha.NumOcurrencias(a_buscar);
103:             procesados[indx] = true;
104:
105:
106:     La combinación de b) con 2) sería:
107:
108:     for (int i...
109:         char a_buscar = v[i];
110:
111:         if (-1 == procesados.PrimerOcurrencia(a_buscar)) {
112:             es_lap = NumOcurrencias(v[i], 0, fin_izda)
113:                 ==
114:                 NumOcurrencias(v[i], ini_dcha, utilizados-1);
115:             procesados.Aniade(a_buscar);
116:
117:
118:     Como detalle final, la ejecución de NumOcurrencias podría realizarse
119:     en la parte izquierda desde la posición i (en vez desde cero)
120:     Incluso podríamos empezar la ejecución de NumOcurrencias en la parte
121:     derecha desde una posición más avanzada de la mitad:
122:     podríamos empezar desde ini_dcha + el número de elementos
123:     que ya han sido añadidos a la secuencia procesados.
124:     En cualquier caso, es una mejora muy marginal que no merece la pena
125:     implementarla.
126: */
127:
128: #include <iostream>
129: #include <string>
130: using namespace std;
131:
132: class SecuenciaCaracteres{
133: private:
134:     static const int TAMANIO = 500;
135:     char v[TAMANIO];
136:     int utilizados;
137:
138:     bool EsCorrectaPos(int indice){
139:         return 0 <= indice && indice < utilizados;
140:     }
141: public:
142:     SecuenciaCaracteres()
143:         :utilizados(0) {
144:     }
145:
146:     int Utilizados(){
147:         return utilizados;
148:     }
149:
150:     int Capacidad(){
151:         return TAMANIO;
152:     }
153:
154:     void EliminaTodos(){
155:         utilizados = 0;
156:     }
157:
158:     void Aniade(char nuevo){
159:         if (utilizados < TAMANIO){
160:             v[utilizados] = nuevo;
161:             utilizados++;
162:         }
163:     }
164:
165:     void Modifica(int posicion, char nuevo){
166:         if (EsCorrectaPos(posicion))
167:             v[posicion] = nuevo;
168:     }
169:
170:     char Elemento(int indice){
171:         return v[indice];
172:     }

```

```

173:
174:     string ToString(){
175:         // Si el número de caracteres en memoria es muy grande,
176:         // es mucho más eficiente reservar memoria previamente
177:         // y usar push_back
178:
179:         string cadena;
180:
181:         cadena.reserve(utilizados);
182:
183:         for (int i=0; i < utilizados; i++)
184:             cadena.push_back(v[i]);
185:         //cadena = cadena + v[i] <- Evitarlo. Muy ineficiente para tamaños grandes;
186:
187:         return cadena;
188:     }
189:
190:     int PrimeraOcurrenciaEntre (int pos_izda, int pos_dcha, char buscado){
191:         int i = pos_izda;
192:         bool encontrado = false;
193:
194:         while (i <= pos_dcha && !encontrado)
195:             if (v[i] == buscado)
196:                 encontrado = true;
197:             else
198:                 i++;
199:
200:         if (encontrado)
201:             return i;
202:         else
203:             return -1;
204:     }
205:
206:     int PrimeraOcurrencia (char buscado){
207:         return PrimeraOcurrenciaEntre (0, utilizados - 1, buscado);
208:     }
209:
210:     //////////////////////////////////////
211:     // EXAMEN
212:     //////////////////////////////////////
213:
214:
215:     SecuenciaCaracteres Subsecuencia(int pos_ini, int pos_fin){
216:         SecuenciaCaracteres Subsecuencia;
217:
218:         for (int i=pos_ini; i<=pos_fin; i++)
219:             Subsecuencia.Aniade(v[i]);
220:
221:         return Subsecuencia;
222:     }
223:
224:     int NumOcurrencias(char car, int desde, int hasta){
225:         int num_ocurrencias;
226:
227:         num_ocurrencias = 0;
228:
229:         for (int i=desde; i<=hasta; i++)
230:             if (v[i]==car)
231:                 num_ocurrencias++;
232:
233:         return num_ocurrencias;
234:     }
235:
236:     int NumOcurrencias(char car){
237:         return NumOcurrencias(car, 0, utilizados - 1);
238:     }
239:
240:
241:     bool EsLapindrome_vs_a_1(){
242:         bool es_lap = true;
243:         const int NUM_CAR = 256;
244:         bool procesados[NUM_CAR];
245:         int mitad, fin_izda, ini_dcha;
246:         SecuenciaCaracteres izda, dcha;
247:
248:         for (int i=0; i<NUM_CAR; i++)
249:             procesados[i] = false;
250:
251:         mitad = utilizados/2;
252:         fin_izda = mitad - 1;
253:
254:         if (utilizados % 2 == 0)
255:             ini_dcha = mitad;
256:         else
257:             ini_dcha = mitad + 1;
258:

```

```

259:     izda = Subsecuencia(0, fin_izda);
260:     dcha = Subsecuencia(ini_dcha, utilizados - 1);
261:
262:     for (int i=0 ; i<=fin_izda && es_lap; i++){
263:         char a_buscar = izda.Elemento(i);
264:         int indx = a_buscar - 'a';
265:
266:         if (!procesados[indx]){
267:             es_lap = izda.NumOcurrencias(a_buscar)
268:                 ==
269:                 dcha.NumOcurrencias(a_buscar);
270:
271:             procesados[indx] = true;
272:         }
273:     }
274:
275:     return es_lap;
276: }
277:
278:
279: bool EsLapindrome_vs_b_2(){
280:     bool es_lap = true;
281:     SecuenciaCaracteres procesados;
282:     int mitad = utilizados/2;
283:     int fin_izda, ini_dcha;
284:
285:     fin_izda = mitad - 1;
286:
287:     if (utilizados % 2 == 0)
288:         ini_dcha = mitad;
289:     else
290:         ini_dcha = mitad + 1;
291:
292:
293:     for (int i=0 ; i<=fin_izda && es_lap; i++){
294:         char a_buscar = v[i];
295:
296:         if (-1 == procesados.PrimerOcurrencia(a_buscar)) {
297:             es_lap = NumOcurrencias(a_buscar, 0, fin_izda)
298:                 ==
299:                 NumOcurrencias(a_buscar, ini_dcha, utilizados-1);
300:
301:             procesados.Aniade(a_buscar);
302:         }
303:     }
304:
305:     return es_lap;
306: }
307: };
308:
309:
310: class LectorSecuenciaCaracteres{
311: private:
312:     char terminador;
313:     int tope;
314:     int capacidad_maxima;
315:
316:     bool FlujoAbierto(){
317:         return !cin.fail();
318:     }
319:
320: public:
321:     LectorSecuenciaCaracteres(){
322:         ResetRestricciones();
323:     }
324:
325:     void SetTerminador (char terminador_entrada){
326:         terminador = terminador_entrada;
327:     }
328:
329:     void SetTope(int num_valores_a_leer){
330:         if (0 < num_valores_a_leer && num_valores_a_leer <= capacidad_maxima)
331:             tope = num_valores_a_leer;
332:     }
333:
334:     void ResetRestricciones(){
335:         SecuenciaCaracteres cualquiera;
336:
337:         capacidad_maxima = cualquiera.Capacidad();
338:         tope = capacidad_maxima;
339:         terminador = '\n';
340:     }
341:
342:     SecuenciaCaracteres Lee(){
343:         SecuenciaCaracteres a_leer;
344:         char caracter;

```

```
345:     bool parar_lectura;
346:     bool es_terminador;
347:     int total_leidos = 0;
348:
349:     do{
350:         if (FlujoAbierto()){
351:             caracter = cin.get();
352:             total_leidos++;
353:             es_terminador = caracter == terminador;
354:
355:             if (!es_terminador)
356:                 a_leer.Aniade(caracter);
357:
358:             parar_lectura = es_terminador || total_leidos == tope;
359:         }
360:     }while (!parar_lectura);
361:
362:     return a_leer;
363: }
364: };
365:
366:
367:
368: int main(){
369:     SecuenciaCaracteres sec;
370:     LectorSecuenciaCaracteres lector;
371:
372:     lector.SetTerminador('#');
373:     sec = lector.Lee();
374:
375:     if (sec.EsLapindrome_vs_b_2())
376:         cout << "\nEs Lapíndrome";
377:     else
378:         cout << "\nNo es lapíndrome";
379:
380:     cout << "\n\n";
381:
382:     // acbbababac# SI
383:     // acbbagbabac# SI
384:     // acbbaaabac# NO
385: }
```

```
1: //////////////////////////////////////
2: //
3: // Fundamentos de Programación
4: // ETS Informática y Telecomunicaciones
5: // Universidad de Granada
6: // Departamento de Ciencias de la Computación e Inteligencia Artificial
7: // Autor: Juan Carlos Cubero
8: //
9: //////////////////////////////////////
10:
11: // Tablero EsHeterogeneo y EsHeterogeneoCompleto
12:
13: #include <iostream>
14: #include <string>
15:
16: using namespace std;
17:
18:
19: /*
20: Explicación de la solución:
21:
22:     clase SecuenciaEnteros:
23:         Añadimos el método Suma()
24:
25:     clase Tablero:
26:         Las cabeceras de los métodos son:
27:             bool EsHeterogeneo()
28:             bool EsHeterogeneoCompleto()
29:
30:         Para implementar los métodos anteriores
31:         usamos una secuencia de enteros
32:             sumas_col
33:         que contiene las sumas de las columnas
34:
35:     a) Versión ineficiente -> Se calculan las sumas de TODAS las columnas
36:
37:         bool EsHeterogeneo()
38:             for -i-
39:                 suma = Columna(i).Suma()
40:                 sumas_col.Aniade(suma)
41:
42:             Si hay algún repetido en sumas_col,
43:                 es_heterogeneo = false
44:
45:         bool EsHeterogeneoCompleto()
46:             for -i-
47:                 suma = Columna(i).Suma()
48:                 sumas_col.Aniade(suma)
49:
50:             sumas_col.Ordena()
51:             Si hay algún elemento no consecutivo en sumas_col,
52:                 es_heterogeneo = false
53:
54:         Podemos apreciar que la primera parte de EsHeterogeneoCompleto es
55:         la misma que EsHeterogeneo:
56:
57:             for -i-
58:                 suma = Columna(i).Suma()
59:                 sumas_col.Aniade(suma)
60:
61:         Lo que hace el anterior código es calcular una secuencia
62:         con las sumas de cada columna. Por lo tanto, para no repetir
63:         dicho código, debemos construir dentro de la clase Tablero
64:         un método que devuelva dicha secuencia:
65:
66:         class Tablero{
67:             ...
68:             SecuenciaEnteros SumasCol() {
69:                 SecuenciaEnteros sumas_col
70:
71:                 for -i-
72:                     suma = Columna(i).Suma()
73:                     sumas_col.Aniade(suma)
74:
75:                 return sumas_col
76:             }
77:
78:         El método puede ser privado o público:
79:
80:         int main(){
81:             ...
82:             SecuenciaEnteros sumas_columnas(tablero.SumasCol()); // :-)
83:
84:         Nos quedaría:
85:
86:         bool EsHeterogeneo()
```

```

87:         sumas_col = SumasCol()
88:
89:         Si hay algún repetido en sumas_col,
90:         es_heterogeneo = false
91:
92:         bool EsHeterogeneoCompleto()
93:         {
94:             sumas_col = SumasCol()
95:             sumas_col.Ordena()
96:             Si hay algún elemento no consecutivo en sumas_col,
97:             es_heterogeneo = false
98:
99:         b) Versión eficiente -> Paramos de sumar columnas cuando
100:            una suma sea igual a alguna ya generada
101:            previamente
102:
103:            La idea es similar a lo visto en a)
104:            Pero ahora debemos parar en cuanto encontremos una suma
105:            ya generada previamente.
106:            Para no repetir código en los métodos EsHeterogeneo y EsHeterogeneoCompleto,
107:            seguimos necesitando un método
108:            SecuenciaEnteros SumasCol()
109:            pero ahora el método sólo contendrá las sumas de las primeras
110:            columnas que sean distintas.
111:            En este caso ya no tiene sentido que sea público ya que es
112:            un método muy específico que hemos diseñado para no repetir
113:            código.
114:            Será un tablero heterogéneo si el tamaño SumasCol es igual
115:            a la dimensión del tablero
116:
117:            Nos quedaría:
118:            private:
119:                SecuenciaEnteros SumasCol() {
120:                    SecuenciaEnteros sumas_col
121:
122:                    for -i- && son_distintos
123:                        suma = Columna(i).Suma()
124:
125:                    if (-1 == sumas_col.PrimerOcurrencia(suma))
126:                        son_distintos = true
127:                        sumas_col.Aniade(suma)
128:
129:                    return sumas_col
130:                }
131:
132:            public:
133:                bool EsHeterogeneo()
134:                {
135:                    es_heterogeneo = SumasCol().Utilizados() == dimension
136:
137:                }
138:
139:                bool EsHeterogeneoCompleto()
140:                {
141:                    sumas_col = SumasCol()
142:                    es_heterogeneo = sumas_col.Utilizados() == dimension
143:
144:                    if (es_heterogeneo)
145:                        sumas_col.Ordena()
146:                        Si hay algún elemento no consecutivo en sumas_col,
147:                        es_heterogeneo = false
148:                    */
149:
150:                }
151:
152:            class SecuenciaEnteros{
153:            private:
154:                static const int TAMANIO = 50;
155:                int v[TAMANIO];
156:                int utilizados;
157:            public:
158:                SecuenciaEnteros()
159:                {
160:                    :utilizados(0) {
161:
162:                }
163:
164:                int Utilizados(){
165:                    return utilizados;
166:                }
167:
168:                int Capacidad(){
169:                    return TAMANIO;
170:                }
171:
172:                void Aniade(int nuevo){
173:                    if (utilizados < TAMANIO){
174:                        v[utilizados] = nuevo;
175:                        utilizados++;
176:                    }
177:                }
178:
179:                int Elemento(int indice){
180:                    return v[indice];
181:                }
182:            };
183:
184:            bool EsHeterogeneo()
185:            {
186:                es_heterogeneo = SumasCol().Utilizados() == dimension
187:
188:            }
189:
190:            bool EsHeterogeneoCompleto()
191:            {
192:                sumas_col = SumasCol()
193:                es_heterogeneo = sumas_col.Utilizados() == dimension
194:
195:                if (es_heterogeneo)
196:                    sumas_col.Ordena()
197:                    Si hay algún elemento no consecutivo en sumas_col,
198:                    es_heterogeneo = false
199:            }
200:
201:            */
202:
203:            class SecuenciaEnteros{
204:            private:
205:                static const int TAMANIO = 50;
206:                int v[TAMANIO];
207:                int utilizados;
208:            public:
209:                SecuenciaEnteros()
210:                {
211:                    :utilizados(0) {
212:
213:                }
214:
215:                int Utilizados(){
216:                    return utilizados;
217:                }
218:
219:                int Capacidad(){
220:                    return TAMANIO;
221:                }
222:
223:                void Aniade(int nuevo){
224:                    if (utilizados < TAMANIO){
225:                        v[utilizados] = nuevo;
226:                        utilizados++;
227:                    }
228:                }
229:
230:                int Elemento(int indice){
231:                    return v[indice];
232:                }
233:            };
234:
235:            bool EsHeterogeneo()
236:            {
237:                es_heterogeneo = SumasCol().Utilizados() == dimension
238:
239:            }
240:
241:            bool EsHeterogeneoCompleto()
242:            {
243:                sumas_col = SumasCol()
244:                es_heterogeneo = sumas_col.Utilizados() == dimension
245:
246:                if (es_heterogeneo)
247:                    sumas_col.Ordena()
248:                    Si hay algún elemento no consecutivo en sumas_col,
249:                    es_heterogeneo = false
250:            }
251:
252:            */
253:
254:            class SecuenciaEnteros{
255:            private:
256:                static const int TAMANIO = 50;
257:                int v[TAMANIO];
258:                int utilizados;
259:            public:
260:                SecuenciaEnteros()
261:                {
262:                    :utilizados(0) {
263:
264:                }
265:
266:                int Utilizados(){
267:                    return utilizados;
268:                }
269:
270:                int Capacidad(){
271:                    return TAMANIO;
272:                }
273:
274:                void Aniade(int nuevo){
275:                    if (utilizados < TAMANIO){
276:                        v[utilizados] = nuevo;
277:                        utilizados++;
278:                    }
279:                }
280:
281:                int Elemento(int indice){
282:                    return v[indice];
283:                }
284:            };
285:
286:            bool EsHeterogeneo()
287:            {
288:                es_heterogeneo = SumasCol().Utilizados() == dimension
289:
290:            }
291:
292:            bool EsHeterogeneoCompleto()
293:            {
294:                sumas_col = SumasCol()
295:                es_heterogeneo = sumas_col.Utilizados() == dimension
296:
297:                if (es_heterogeneo)
298:                    sumas_col.Ordena()
299:                    Si hay algún elemento no consecutivo en sumas_col,
300:                    es_heterogeneo = false
301:            }
302:
303:            */
304:
305:            class SecuenciaEnteros{
306:            private:
307:                static const int TAMANIO = 50;
308:                int v[TAMANIO];
309:                int utilizados;
310:            public:
311:                SecuenciaEnteros()
312:                {
313:                    :utilizados(0) {
314:
315:                }
316:
317:                int Utilizados(){
318:                    return utilizados;
319:                }
320:
321:                int Capacidad(){
322:                    return TAMANIO;
323:                }
324:
325:                void Aniade(int nuevo){
326:                    if (utilizados < TAMANIO){
327:                        v[utilizados] = nuevo;
328:                        utilizados++;
329:                    }
330:                }
331:
332:                int Elemento(int indice){
333:                    return v[indice];
334:                }
335:            };
336:
337:            bool EsHeterogeneo()
338:            {
339:                es_heterogeneo = SumasCol().Utilizados() == dimension
340:
341:            }
342:
343:            bool EsHeterogeneoCompleto()
344:            {
345:                sumas_col = SumasCol()
346:                es_heterogeneo = sumas_col.Utilizados() == dimension
347:
348:                if (es_heterogeneo)
349:                    sumas_col.Ordena()
350:                    Si hay algún elemento no consecutivo en sumas_col,
351:                    es_heterogeneo = false
352:            }
353:
354:            */
355:
356:            class SecuenciaEnteros{
357:            private:
358:                static const int TAMANIO = 50;
359:                int v[TAMANIO];
360:                int utilizados;
361:            public:
362:                SecuenciaEnteros()
363:                {
364:                    :utilizados(0) {
365:
366:                }
367:
368:                int Utilizados(){
369:                    return utilizados;
370:                }
371:
372:                int Capacidad(){
373:                    return TAMANIO;
374:                }
375:
376:                void Aniade(int nuevo){
377:                    if (utilizados < TAMANIO){
378:                        v[utilizados] = nuevo;
379:                        utilizados++;
380:                    }
381:                }
382:
383:                int Elemento(int indice){
384:                    return v[indice];
385:                }
386:            };
387:
388:            bool EsHeterogeneo()
389:            {
390:                es_heterogeneo = SumasCol().Utilizados() == dimension
391:
392:            }
393:
394:            bool EsHeterogeneoCompleto()
395:            {
396:                sumas_col = SumasCol()
397:                es_heterogeneo = sumas_col.Utilizados() == dimension
398:
399:                if (es_heterogeneo)
400:                    sumas_col.Ordena()
401:                    Si hay algún elemento no consecutivo en sumas_col,
402:                    es_heterogeneo = false
403:            }
404:
405:            */
406:
407:            class SecuenciaEnteros{
408:            private:
409:                static const int TAMANIO = 50;
410:                int v[TAMANIO];
411:                int utilizados;
412:            public:
413:                SecuenciaEnteros()
414:                {
415:                    :utilizados(0) {
416:
417:                }
418:
419:                int Utilizados(){
420:                    return utilizados;
421:                }
422:
423:                int Capacidad(){
424:                    return TAMANIO;
425:                }
426:
427:                void Aniade(int nuevo){
428:                    if (utilizados < TAMANIO){
429:                        v[utilizados] = nuevo;
430:                        utilizados++;
431:                    }
432:                }
433:
434:                int Elemento(int indice){
435:                    return v[indice];
436:                }
437:            };
438:
439:            bool EsHeterogeneo()
440:            {
441:                es_heterogeneo = SumasCol().Utilizados() == dimension
442:
443:            }
444:
445:            bool EsHeterogeneoCompleto()
446:            {
447:                sumas_col = SumasCol()
448:                es_heterogeneo = sumas_col.Utilizados() == dimension
449:
450:                if (es_heterogeneo)
451:                    sumas_col.Ordena()
452:                    Si hay algún elemento no consecutivo en sumas_col,
453:                    es_heterogeneo = false
454:            }
455:
456:            */
457:
458:            class SecuenciaEnteros{
459:            private:
460:                static const int TAMANIO = 50;
461:                int v[TAMANIO];
462:                int utilizados;
463:            public:
464:                SecuenciaEnteros()
465:                {
466:                    :utilizados(0) {
467:
468:                }
469:
470:                int Utilizados(){
471:                    return utilizados;
472:                }
473:
474:                int Capacidad(){
475:                    return TAMANIO;
476:                }
477:
478:                void Aniade(int nuevo){
479:                    if (utilizados < TAMANIO){
480:                        v[utilizados] = nuevo;
481:                        utilizados++;
482:                    }
483:                }
484:
485:                int Elemento(int indice){
486:                    return v[indice];
487:                }
488:            };
489:
490:            bool EsHeterogeneo()
491:            {
492:                es_heterogeneo = SumasCol().Utilizados() == dimension
493:
494:            }
495:
496:            bool EsHeterogeneoCompleto()
497:            {
498:                sumas_col = SumasCol()
499:                es_heterogeneo = sumas_col.Utilizados() == dimension
500:
501:                if (es_heterogeneo)
502:                    sumas_col.Ordena()
503:                    Si hay algún elemento no consecutivo en sumas_col,
504:                    es_heterogeneo = false
505:            }
506:
507:            */
508:
509:            class SecuenciaEnteros{
510:            private:
511:                static const int TAMANIO = 50;
512:                int v[TAMANIO];
513:                int utilizados;
514:            public:
515:                SecuenciaEnteros()
516:                {
517:                    :utilizados(0) {
518:
519:                }
520:
521:                int Utilizados(){
522:                    return utilizados;
523:                }
524:
525:                int Capacidad(){
526:                    return TAMANIO;
527:                }
528:
529:                void Aniade(int nuevo){
530:                    if (utilizados < TAMANIO){
531:                        v[utilizados] = nuevo;
532:                        utilizados++;
533:                    }
534:                }
535:
536:                int Elemento(int indice){
537:                    return v[indice];
538:                }
539:            };
540:
541:            bool EsHeterogeneo()
542:            {
543:                es_heterogeneo = SumasCol().Utilizados() == dimension
544:
545:            }
546:
547:            bool EsHeterogeneoCompleto()
548:            {
549:                sumas_col = SumasCol()
550:                es_heterogeneo = sumas_col.Utilizados() == dimension
551:
552:                if (es_heterogeneo)
553:                    sumas_col.Ordena()
554:                    Si hay algún elemento no consecutivo en sumas_col,
555:                    es_heterogeneo = false
556:            }
557:
558:            */
559:
560:            class SecuenciaEnteros{
561:            private:
562:                static const int TAMANIO = 50;
563:                int v[TAMANIO];
564:                int utilizados;
565:            public:
566:                SecuenciaEnteros()
567:                {
568:                    :utilizados(0) {
569:
570:                }
571:
572:                int Utilizados(){
573:                    return utilizados;
574:                }
575:
576:                int Capacidad(){
577:                    return TAMANIO;
578:                }
579:
580:                void Aniade(int nuevo){
581:                    if (utilizados < TAMANIO){
582:                        v[utilizados] = nuevo;
583:                        utilizados++;
584:                    }
585:                }
586:
587:                int Elemento(int indice){
588:                    return v[indice];
589:                }
590:            };
591:
592:            bool EsHeterogeneo()
593:            {
594:                es_heterogeneo = SumasCol().Utilizados() == dimension
595:
596:            }
597:
598:            bool EsHeterogeneoCompleto()
599:            {
600:                sumas_col = SumasCol()
601:                es_heterogeneo = sumas_col.Utilizados() == dimension
602:
603:                if (es_heterogeneo)
604:                    sumas_col.Ordena()
605:                    Si hay algún elemento no consecutivo en sumas_col,
606:                    es_heterogeneo = false
607:            }
608:
609:            */
610:
611:            class SecuenciaEnteros{
612:            private:
613:                static const int TAMANIO = 50;
614:                int v[TAMANIO];
615:                int utilizados;
616:            public:
617:                SecuenciaEnteros()
618:                {
619:                    :utilizados(0) {
620:
621:                }
622:
623:                int Utilizados(){
624:                    return utilizados;
625:                }
626:
627:                int Capacidad(){
628:                    return TAMANIO;
629:                }
630:
631:                void Aniade(int nuevo){
632:                    if (utilizados < TAMANIO){
633:                        v[utilizados] = nuevo;
634:                        utilizados++;
635:                    }
636:                }
637:
638:                int Elemento(int indice){
639:                    return v[indice];
640:                }
641:            };
642:
643:            bool EsHeterogeneo()
644:            {
645:                es_heterogeneo = SumasCol().Utilizados() == dimension
646:
647:            }
648:
649:            bool EsHeterogeneoCompleto()
650:            {
651:                sumas_col = SumasCol()
652:                es_heterogeneo = sumas_col.Utilizados() == dimension
653:
654:                if (es_heterogeneo)
655:                    sumas_col.Ordena()
656:                    Si hay algún elemento no consecutivo en sumas_col,
657:                    es_heterogeneo = false
658:            }
659:
660:            */
661:
662:            class SecuenciaEnteros{
663:            private:
664:                static const int TAMANIO = 50;
665:                int v[TAMANIO];
666:                int utilizados;
667:            public:
668:                SecuenciaEnteros()
669:                {
670:                    :utilizados(0) {
671:
672:                }
673:
674:                int Utilizados(){
675:                    return utilizados;
676:                }
677:
678:                int Capacidad(){
679:                    return TAMANIO;
680:                }
681:
682:                void Aniade(int nuevo){
683:                    if (utilizados < TAMANIO){
684:                        v[utilizados] = nuevo;
685:                        utilizados++;
686:                    }
687:                }
688:
689:                int Elemento(int indice){
690:                    return v[indice];
691:                }
692:            };
693:
694:            bool EsHeterogeneo()
695:            {
696:                es_heterogeneo = SumasCol().Utilizados() == dimension
697:
698:            }
699:
700:            bool EsHeterogeneoCompleto()
701:            {
702:                sumas_col = SumasCol()
703:                es_heterogeneo = sumas_col.Utilizados() == dimension
704:
705:                if (es_heterogeneo)
706:                    sumas_col.Ordena()
707:                    Si hay algún elemento no consecutivo en sumas_col,
708:                    es_heterogeneo = false
709:            }
710:
711:            */
712:
713:            class SecuenciaEnteros{
714:            private:
715:                static const int TAMANIO = 50;
716:                int v[TAMANIO];
717:                int utilizados;
718:            public:
719:                SecuenciaEnteros()
720:                {
721:                    :utilizados(0) {
722:
723:                }
724:
725:                int Utilizados(){
726:                    return utilizados;
727:                }
728:
729:                int Capacidad(){
730:                    return TAMANIO;
731:                }
732:
733:                void Aniade(int nuevo){
734:                    if (utilizados < TAMANIO){
735:                        v[utilizados] = nuevo;
736:                        utilizados++;
737:                    }
738:                }
739:
740:                int Elemento(int indice){
741:                    return v[indice];
742:                }
743:            };
744:
745:            bool EsHeterogeneo()
746:            {
747:                es_heterogeneo = SumasCol().Utilizados() == dimension
748:
749:            }
750:
751:            bool EsHeterogeneoCompleto()
752:            {
753:                sumas_col = SumasCol()
754:                es_heterogeneo = sumas_col.Utilizados() == dimension
755:
756:                if (es_heterogeneo)
757:                    sumas_col.Ordena()
758:                    Si hay algún elemento no consecutivo en sumas_col,
759:                    es_heterogeneo = false
760:            }
761:
762:            */
763:
764:            class SecuenciaEnteros{
765:            private:
766:                static const int TAMANIO = 50;
767:                int v[TAMANIO];
768:                int utilizados;
769:            public:
770:                SecuenciaEnteros()
771:                {
772:                    :utilizados(0) {
773:
774:                }
775:
776:                int Utilizados(){
777:                    return utilizados;
778:                }
779:
780:                int Capacidad(){
781:                    return TAMANIO;
782:                }
783:
784:                void Aniade(int nuevo){
785:                    if (utilizados < TAMANIO){
786:                        v[utilizados] = nuevo;
787:                        utilizados++;
788:                    }
789:                }
790:
791:                int Elemento(int indice){
792:                    return v[indice];
793:                }
794:            };
795:
796:            bool EsHeterogeneo()
797:            {
798:                es_heterogeneo = SumasCol().Utilizados() == dimension
799:
800:            }
801:
802:            bool EsHeterogeneoCompleto()
803:            {
804:                sumas_col = SumasCol()
805:                es_heterogeneo = sumas_col.Utilizados() == dimension
806:
807:                if (es_heterogeneo)
808:                    sumas_col.Ordena()
809:                    Si hay algún elemento no consecutivo en sumas_col,
810:                    es_heterogeneo = false
811:            }
812:
813:            */
814:
815:            class SecuenciaEnteros{
816:            private:
817:                static const int TAMANIO = 50;
818:                int v[TAMANIO];
819:                int utilizados;
820:            public:
821:                SecuenciaEnteros()
822:                {
823:                    :utilizados(0) {
824:
825:                }
826:
827:                int Utilizados(){
828:                    return utilizados;
829:                }
830:
831:                int Capacidad(){
832:                    return TAMANIO;
833:                }
834:
835:                void Aniade(int nuevo){
836:                    if (utilizados < TAMANIO){
837:                        v[utilizados] = nuevo;
838:                        utilizados++;
839:                    }
840:                }
841:
842:                int Elemento(int indice){
843:                    return v[indice];
844:                }
845:            };
846:
847:            bool EsHeterogeneo()
848:            {
849:                es_heterogeneo = SumasCol().Utilizados() == dimension
850:
851:            }
852:
853:            bool EsHeterogeneoCompleto()
854:            {
855:                sumas_col = SumasCol()
856:                es_heterogeneo = sumas_col.Utilizados() == dimension
857:
858:                if (es_heterogeneo)
859:                    sumas_col.Ordena()
860:                    Si hay algún elemento no consecutivo en sumas_col,
861:                    es_heterogeneo = false
862:            }
863:
864:            */
865:
866:            class SecuenciaEnteros{
867:            private:
868:                static const int TAMANIO = 50;
869:                int v[TAMANIO];
870:                int utilizados;
871:            public:
872:                SecuenciaEnteros()
873:                {
874:                    :utilizados(0) {
875:
876:                }
877:
878:                int Utilizados(){
879:                    return utilizados;
880:                }
881:
882:                int Capacidad(){
883:                    return TAMANIO;
884:                }
885:
886:                void Aniade(int nuevo){
887:                    if (utilizados < TAMANIO){
888:                        v[utilizados] = nuevo;
889:                        utilizados++;
890:                    }
891:                }
892:
893:                int Elemento(int indice){
894:                    return v[indice];
895:                }
896:            };
897:
898:            bool EsHeterogeneo()
899:            {
900:                es_heterogeneo = SumasCol().Utilizados() == dimension
901:
902:            }
903:
904:            bool EsHeterogeneoCompleto()
905:            {
906:                sumas_col = SumasCol()
907:                es_heterogeneo = sumas_col.Utilizados() == dimension
908:
909:                if (es_heterogeneo)
910:                    sumas_col.Ordena()
911:                    Si hay algún elemento no consecutivo en sumas_col,
912:                    es_heterogeneo = false
913:            }
914:
915:            */
916:
917:            class SecuenciaEnteros{
918:            private:
919:                static const int TAMANIO = 50;
920:                int v[TAMANIO];
921:                int utilizados;
922:            public:
923:                SecuenciaEnteros()
924:                {
925:                    :utilizados(0) {
926:
927:                }
928:
929:                int Utilizados(){
930:                    return utilizados;
931:                }
932:
933:                int Capacidad(){
934:                    return TAMANIO;
935:                }
936:
937:                void Aniade(int nuevo){
938:                    if (utilizados < TAMANIO){
939:                        v[utilizados] = nuevo;
940:                        utilizados++;
941:                    }
942:                }
943:
944:                int Elemento(int indice){
945:                    return v[indice];
946:                }
947:            };
948:
949:            bool EsHeterogeneo()
950:            {
951:                es_heterogeneo = SumasCol().Utilizados() == dimension
952:
953:            }
954:
955:            bool EsHeterogeneoCompleto()
956:            {
957:                sumas_col = SumasCol()
958:                es_heterogeneo = sumas_col.Utilizados() == dimension
959:
960:                if (es_heterogeneo)
961:                    sumas_col.Ordena()
962:                    Si hay algún elemento no consecutivo en sumas_col,
963:                    es_heterogeneo = false
964:            }
965:
966:            */
967:
968:            class SecuenciaEnteros{
969:            private:
970:                static const int TAMANIO = 50;
971:                int v[TAMANIO];
972:                int utilizados;
973:            public:
974:                SecuenciaEnteros()
975:                {
976:                    :utilizados(0) {
977:
978:                }
979:
980:                int Utilizados(){
981:                    return utilizados;
982:                }
983:
984:                int Capacidad(){
985:                    return TAMANIO;
986:                }
987:
988:                void Aniade(int nuevo){
989:                    if (utilizados < TAMANIO){
990:                        v[utilizados] = nuevo;
991:                        utilizados++;
992:                    }
993:                }
994:
995:                int Elemento(int indice){
996:                    return v[indice];
997:                }
998:            };
999:
1000:            bool EsHeterogeneo()
1001:            {
1002:                es_heterogeneo = SumasCol().Utilizados() == dimension
1003:
1004:            }
1005:
1006:            bool EsHeterogeneoCompleto()
1007:            {
1008:                sumas_col = SumasCol()
1009:                es_heterogeneo = sumas_col.Utilizados() == dimension
1010:
1011:                if (es_heterogeneo)
1012:                    sumas_col.Ordena()
1013:                    Si hay algún elemento no consecutivo en sumas_col,
1014:                    es_heterogeneo = false
1015:            }
1016:
1017:            */
1018:
1019:            class SecuenciaEnteros{
1020:            private:
1021:                static const int TAMANIO = 50;
1022:                int v[TAMANIO];
1023:                int utilizados;
1024:            public:
1025:                SecuenciaEnteros()
1026:                {
1027:                    :utilizados(0) {
1028:
1029:                }
1030:
1031:                int Utilizados(){
1032:                    return utilizados;
1033:                }
1034:
1035:                int Capacidad(){
1036:                    return TAMANIO;
1037:                }
1038:
1039:                void Aniade(int nuevo){
1040:                    if (utilizados < TAMANIO){
1041:                        v[utilizados] = nuevo;
1042:                        utilizados++;
1043:                    }
1044:                }
1045:
1046:                int Elemento(int indice){
1047:                    return v[indice];
1048:                }
1049:            };
1050:
1051:            bool EsHeterogeneo()
1052:            {
1053:                es_heterogeneo = SumasCol().Utilizados() == dimension
1054:
1055:            }
1056:
1057:            bool EsHeterogeneoCompleto()
1058:            {
1059:                sumas_col = SumasCol()
1060:                es_heterogeneo = sumas_col.Utilizados() == dimension
1061:
1062:                if (es_heterogeneo)
1063:                    sumas_col.Ordena()
1064:                    Si hay algún elemento no consecutivo en sumas_col,
1065:                    es_heterogeneo = false
1066:            }
1067:
1068:            */
1069:
1070:            class SecuenciaEnteros{
1071:            private:
1072:                static const int TAMANIO = 50;
1073:                int v[TAMANIO];
1074:                int utilizados;
1075:            public:
1076:                SecuenciaEnteros()
1077:                {
1078:                    :utilizados(0) {
1079:
1080:                }
1081:
1082:                int Utilizados(){
1083:                    return utilizados;
1084:                }
1085:
1086:                int Capacidad(){
1087:                    return TAMANIO;
1088:                }
1089:
1090:                void Aniade(int nuevo){
1091:                    if (utilizados < TAMANIO){
1092:                        v[utilizados] = nuevo;
1093:                        utilizados++;
1094:                    }
1095:                }
1096:
1097:                int Elemento(int indice){
1098:                    return v[indice];
1099:                }
1100:            };
1101:
1102:            bool EsHeterogeneo()
1103:            {
1104:                es_heterogeneo = SumasCol().Utilizados() == dimension
1105:
1106:            }
1107:
1108:            bool EsHeterogeneoCompleto()
1109:            {
1110:                sumas_col = SumasCol()
1111:                es_heterogeneo = sumas_col.Utilizados() == dimension
1112:
1113:                if (es_heterogeneo)
1114:                    sumas_col.Ordena()
1115:                    Si hay algún elemento no consecutivo en sumas_col,
1116:                    es_heterogeneo = false
1117:            }
1118:
1119:            */
1120:
1121:            class SecuenciaEnteros{
1122:            private:
1123:                static const int TAMANIO = 50;
1124:                int v[TAMANIO];
1125:                int utilizados;
1126:            public:
1127:                SecuenciaEnteros()
1128:                {
1129:                    :utilizados(0) {
1130:
1131:                }
1132:
1133:                int Utilizados(){
1134:                    return utilizados;
1135:                }
1136:
1137:                int Capacidad(){
1138:                    return TAMANIO;
1139:                }
1140:
1141:                void Aniade(int nuevo){
1142:                    if (utilizados < TAMANIO){
1143:                        v[utilizados] = nuevo;
1144:                        utilizados++;
1145:                    }
1146:                }
1147:
1148:                int Elemento(int indice){
1149:                    return v[indice];
1150:                }
1151:            };
1152:
1153:            bool EsHeterogeneo()
1154:            {
1155:                es_heterogeneo = SumasCol().Utilizados() == dimension
1156:
1157:            }
1158:
1159:            bool EsHeterogeneoCompleto()
1160:            {
1161:                sumas_col = SumasCol()
1162:                es_heterogeneo = sumas_col.Utilizados() == dimension
1163:
1164:                if (es_heterogeneo)
1165:                    sumas_col.Ordena()
1166:                    Si hay algún elemento no consecutivo en sumas_col,
1167:                    es_heterogeneo = false
1168:            }
1169:
1170:            */
1171:
1172:            class SecuenciaEnteros{
1173:            private:
1174:                static const int TAMANIO = 50;
1175:                int v[TAMANIO];
1176:                int utilizados;
1177:            public:
1178:                SecuenciaEnteros()
1179:                {
1180:                    :utilizados(0) {
1181:
1182:                }
1183:
1184:                int Utilizados(){
1185:                    return utilizados;
1186:                }
1187:
1188:                int Capacidad(){
1189:                    return TAMANIO;
1190:                }
1191:
1192:                void Aniade(int nuevo){
1193:                    if (utilizados < TAMANIO){
1194:                        v[utilizados] = nuevo;
1195:                        utilizados++;
1196:                    }
1197:                }
1198:
1199:                int Elemento(int indice){
1200:                    return v[indice];
1201:                }
1202:            };
1203:
1204:            bool EsHeterogeneo()
1205:            {
1206:                es_heterogeneo = SumasCol().Utilizados() == dimension
1207:
1208:            }
1209:
1210:            bool EsHeterogeneoCompleto()
1211:            {
1212:                sumas_col = SumasCol()
1213:                es_heterogeneo = sumas_col.Utilizados() == dimension
1214:
1215:                if (es_heterogeneo)
1216:                    sumas_col.Ordena()
1217:                    Si hay algún elemento no consecutivo en sumas_col,
1218:                    es_heterogeneo = false
1219:            }
1220:
1221:            */
1222:
1223:            class SecuenciaEnteros{
1224:            private:
1225:                static const int TAMANIO = 50;
1226:                int v[TAMANIO];
1227:                int utilizados;
1228:            public:
1229:                SecuenciaEnteros()
1230:                {
1231:                    :utilizados(0) {
1232:
1233:                }
1234:
1235:                int Utilizados(){
1236:                    return utilizados;
1237:                }
1238:
1239:                int Capacidad(){
1240:                    return TAMANIO;
1241:                }
1242:
1243:                void Aniade(int nuevo){
1244:                    if (utilizados < TAMANIO){
1245:                        v[utilizados] = nuevo;
1246:                        utilizados++;
1247:                    }
1248:                }
1249:
1250:                int Elemento(int indice){
1251:                    return v[indice];
1252:                }
1253:            };
1254:
1255:            bool EsHeterogeneo()
1256:            {
1257:                es_heterogeneo = SumasCol().Utilizados() == dimension
1258:
1259:            }
1260:
1261:            bool EsHeterogeneoCompleto()
1262:            {
1263:                sumas_col = SumasCol()
1264:                es_heterogeneo = sumas_col.Utilizados() == dimension
1265:
1266:                if (es_heterogeneo)
1267:                    sumas_col.Ordena()
1268:                    Si hay algún elemento no consecutivo en sumas_col,
1269:                    es_heterogeneo = false
1270:            }
1271:
1272:            */
1273:
1274:            class SecuenciaEnteros{
1275:            private:
1276:                static const int TAMANIO = 50;
1277:                int v[TAMANIO];
1278:                int utilizados;
1279:            public:
1280:                SecuenciaEnteros()
1281:                {
1282:                    :utilizados(0) {
1283:
1284:                }
1285:
1286:                int Utilizados(){
1287:                    return utilizados;
1288:                }
1289:
1290:                int Capacidad(){
1291:                    return TAMANIO;
1292:                }
1293:
1294:                void Aniade(int nuevo){
1295:                    if (utilizados < TAMANIO){
1296:                        v[utilizados] = nuevo;
1297:                        utilizados++;
1298:                    }
1299:                }
1300:
1301:                int Elemento(int indice){
1302:                    return v[indice];
1303:                }
1304:            };
1305:
1306:            bool EsHeterogeneo()
1307:            {
1308:                es_heterogeneo = SumasCol().Utilizados() == dimension
1309:
1310:            }
1311:
1312:            bool EsHeterogeneoCompleto()
1313:            {
1314:                sumas_col = SumasCol()
1315:                es_heterogeneo = sumas_col
```

```

173:     }
174:
175:     void EliminaTodos(){
176:         utilizados = 0;
177:     }
178:
179:     int Suma(){
180:         int suma = 0;
181:
182:         for (int i=0; i<utilizados; i++)
183:             suma += v[i];
184:
185:         return suma;
186:     }
187:
188:     void Ordena(){
189:         int izda, i;
190:         long a_desplazar;
191:
192:         for (izda=1; izda < utilizados; izda++){
193:             a_desplazar = v[izda];
194:
195:             for (i=izda; i > 0 && a_desplazar < v[i-1]; i--){
196:                 v[i] = v[i-1];
197:
198:                 v[i] = a_desplazar;
199:             }
200:         }
201:
202:         int PrimeraOcurrenciaEntre (int pos_izda, int pos_dcha, long buscado){
203:             int i = pos_izda;
204:             bool encontrado = false;
205:
206:             while (i <= pos_dcha && !encontrado)
207:                 if (v[i] == buscado)
208:                     encontrado = true;
209:                 else
210:                     i++;
211:
212:             if (encontrado)
213:                 return i;
214:             else
215:                 return -1;
216:         }
217:
218:         int PrimeraOcurrencia (long buscado){
219:             return PrimeraOcurrenciaEntre (0, utilizados - 1, buscado);
220:         }
221:
222: };
223:
224:
225: class Tablero{
226: private:
227:     static const int MAX = 40;
228:     int t[MAX][MAX];
229:     int dimension;
230:
231:     // Calcula las sumas de las columnas,
232:     // mientras éstas sean distintas
233:     SecuenciaEnteros SumasCol(){
234:         SecuenciaEnteros sumas_col;
235:         int suma;
236:
237:         bool son_distintos = true;
238:
239:         for (int col=0; col<dimension && son_distintos; col++){
240:             suma = Columna(col).Suma();
241:             son_distintos = -1 == sumas_col.PrimeraOcurrencia(suma);
242:
243:             if (son_distintos)
244:                 sumas_col.Aniade(suma);
245:         }
246:
247:         return sumas_col;
248:     }
249:
250: public:
251:     // Prec: 0 < numero_de_columnas <= MAX(40)
252:     Tablero(int dimension_tablero)
253:         :dimension(dimension_tablero)
254:     {
255:     }
256:
257:     int Dimension(){
258:         return dimension;

```



```

259:     }
260:
261:     int Elemento(int fil, int col){
262:         return t[fil][col];
263:     }
264:
265:     void Modifica(int fil, int col, int valor){
266:         t[fil][col] = valor;
267:     }
268:
269:     SecuenciaEnteros Fila(int fil){
270:         SecuenciaEnteros fila;
271:
272:         for (int col = 0; col < dimension; col++){
273:             fila.Aniade(t[fil][col]);
274:
275:         return fila;
276:     }
277:
278:     SecuenciaEnteros Columna(int col){
279:         SecuenciaEnteros columna;
280:
281:         for (int fil = 0; fil < dimension; fil++){
282:             columna.Aniade(t[fil][col]);
283:
284:         return columna;
285:     }
286:
287:
288:     bool EsHeterogeneo(){
289:         return SumasCol().Utilizados() == dimension;
290:     }
291:
292:     bool EsHeterogeneoCompleto(){
293:         bool es_heterogeneo;
294:         SecuenciaEnteros sumas_col(SumasCol());
295:
296:         es_heterogeneo = sumas_col.Utilizados() == dimension;
297:
298:         if (es_heterogeneo){
299:             sumas_col.Ordena();
300:
301:             for (int i=0; i<dimension-1 && es_heterogeneo; i++){
302:                 es_heterogeneo = sumas_col.Elemento(i)+1
303:                                 ==
304:                                 sumas_col.Elemento(i+1);
305:             }
306:
307:             return es_heterogeneo;
308:         }
309:
310:         SecuenciaEnteros DiagonalInversa(){
311:             SecuenciaEnteros diagonal;
312:
313:             for (int i=0; i<dimension; i++){
314:                 diagonal.Aniade(t[dimension - i - 1][i]);
315:
316:             return diagonal;
317:         }
318:
319:         SecuenciaEnteros Diagonal(){
320:             SecuenciaEnteros diagonal;
321:
322:             for (int i=0; i<dimension; i++){
323:                 diagonal.Aniade(t[i][i]);
324:
325:             return diagonal;
326:         }
327:
328:         string ToString(){
329:             string cadena;
330:
331:             cadena.reserve(dimension * dimension);
332:
333:             for (int i = 0; i < dimension; i++){
334:                 cadena.push_back('\n');
335:
336:                 for (int j = 0; j < dimension; j++){
337:                     cadena.append(to_string(t[i][j]));
338:                     cadena.append(" ");
339:                 }
340:             }
341:
342:             return cadena;
343:         }
344:     };

```

```
345:
346: int main(){
347:     int dimension;
348:     int casilla;
349:
350:     cin >> dimension;
351:
352:     Tablero tablero(dimension);
353:
354:     for (int i = 0; i < dimension; i++){
355:         for (int j = 0; j < dimension; j++){
356:             cin >> casilla;
357:             tablero.Modifica(i, j, casilla);
358:         }
359:     }
360:
361:     cout << "\n\n";
362:     cout << "Tabla original:\n";
363:     cout << tablero.ToString();
364:     cout << "\n\n";
365:
366:     if (tablero.EsHeterogeneo()){
367:         cout << "\nEs heterogéneo";
368:
369:         if (tablero.EsHeterogeneoCompleto())
370:             cout << "\nAdemás, es completo";
371:     }
372:     else
373:         cout << "\nNo es heterogéneo";
374: }
375: // 4      15 2 12 4   1 14 10 5   8 9 3 16   11 13 6 7
376: // Heterogéneo pero no completo
377:
378: // 3      9 8 6   0 2 3   3 2 5
379: // No es heterogéneo
380:
381: // 3      9 8 6   0 2 3   2 2 1
382: // Heterogéneo completo
383:
```