```
////////////////////////////////////////////////////////////////////////
//
// Fundamentos de Programación
// ETS Informática y Telecomunicaciones
// Universidad de Granada
// Departamento de Ciencias de la Computación e Inteligencia Artificial
// Autor: Juan Carlos Cubero
//
////////////////////////////////////////////////////////////////////////

// Examen 2021 Enero. Palisimétrica o Lapíndrome

/*
Explicación de la solución:

   Cómo calcular fin_izda e ini_dcha:

      mitad = utilizados/2;
      fin_izda = mitad - 1;

      if (utilizados % 2 == 0)
         ini_dcha = mitad;
      else
         ini_dcha = mitad + 1;

   El procesamiento de la parte izquierda y derecha lo podemos
   hacer de dos formas -a) y b)- Ambas son completamente válidas:

   a) Podemos trabajar con dos secuencias izda, dcha
      (obviamente, serán locales al método EsLapindrome):

      izda = Subsecuencia(0, fin_izda)
      dcha = Subsecuencia(ini_dcha + 1 , utilizados - 1)

      for (int i=0 ; i<=fin_izda && es_lap; i++)
         char a_buscar = izda.Elemento(i);

         es_lap = izda.NumOcurrencias(a_buscar)
                  ==
                  dcha.NumOcurrencias(a_buscar);

      Es obvio que NumOcurrencias es un método necesario para
      así no repetir el código que cuenta el número de ocurrencias
      de un carácter en una secuencia.

   b) En vez de construir las secuencias izda y dcha
      podemos definir el método NumOcurrencias
      para que trabaje sólo en una parte de la secuencia.
      Para ello, basta definirlo pasándole como parámetros
      los índices inicial y final que delimitan la parte
      de la secuencia en la que voy a buscar:

      for (int i=0 ; i<=fin_izda && es_lap; i++)
         es_lap = NumOcurrencias(v[i], 0, fin_izda)
                  ==
                  NumOcurrencias(v[i], ini_dcha, utilizados-1);

   Independientemente de si es a) o b), para no volver a contar
   la frecuencia de un carácter que ya se hubiese contado anteriormente,
   podemos hacer lo siguiente:

   1) O bien usamos un vector de bool en el que el índice
      de la componente a modificar se obtiene a partir
      del orden del carácter actual

      const int NUM_CAR = 256;
      bool procesados[NUM_CAR];

      procesados[0] será true si 'a' ha sido procesada
      procesados[1] será true si 'b' ha sido procesada
      ...

      'a' -> 0
      'b' -> 1
      ...

      En general:

      car -> car - 'a'

   2) O bien usamos una secuencia de caracteres que almacene los
      caracteres que ya hayan sido procesados

      SecuenciaCaracteres procesados;

      Si ya han sido procesadas la 'g' y la 'k',
```

```
   la secuencia procesados contendrá {'g', 'k'}

Podemos combinar a) y b) con 1) y 2) como queramos.
Por ejemplo, la combinación de a) con 1) sería:

   izda = Subsecuencia(0, fin_izda)
   dcha = Subsecuencia(ini_dcha + 1 , utilizados - 1)

   for (int i...
      char a_buscar = izda.Elemento(i);
      int indx = a_buscar - 'a';

      if (!procesados[indx])
         es_lap = izda.NumOcurrencias(a_buscar)
                  ==
                  dcha.NumOcurrencias(a_buscar);
         procesados[indx] = true;


La combinación de b) con 2) sería:

   for (int i...
      char a_buscar = v[i];

      if (-1 == procesados.PrimeraOcurrencia(a_buscar)) {
         es_lap = NumOcurrencias(v[i], 0, fin_izda)
                  ==
                  NumOcurrencias(v[i], ini_dcha, utilizados-1);
         procesados.Aniade(a_buscar);


Como detalle final, la ejecución de NumOcurrencias podría realizarse
en la parte izquierda desde la posición i (en vez desde cero)
Incluso podríamos empezar la ejecución de NumOcurrencias en la parte
derecha desde una posición más avanzada de la mitad:
podríamos empezar desde ini_dcha + el número de elementos
que ya han sido añadidos a la secuencia procesados.
En cualquier caso, es una mejora muy marginal que no merece la pena
implementarla.
*/

#include <iostream>
#include <string>
using namespace std;

class SecuenciaCaracteres{
private:
   static const int TAMANIO = 500;
   char v[TAMANIO];
   int utilizados;

   bool EsCorrectaPos(int indice){
      return 0 <= indice && indice < utilizados;
   }
public:
   SecuenciaCaracteres()
      :utilizados(0) {
   }

   int Utilizados(){
      return utilizados;
   }

   int Capacidad(){
      return TAMANIO;
   }

   void EliminaTodos(){
      utilizados = 0;
   }

   void Aniade(char nuevo){
      if (utilizados < TAMANIO){
         v[utilizados] = nuevo;
         utilizados++;
      }
   }

   void Modifica(int posicion, char nuevo){
      if (EsCorrectaPos(posicion))
         v[posicion] = nuevo;
   }

   char Elemento(int indice){
      return v[indice];
   }
```

```
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.PrimeraOcurrencia(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: }
```

```
////////////////////////////////////////////////////////////////////////
//
// Fundamentos de Programación
// ETS Informática y Telecomunicaciones
// Universidad de Granada
// Departamento de Ciencias de la Computación e Inteligencia Artificial
// Autor: Juan Carlos Cubero
//
////////////////////////////////////////////////////////////////////////

// Tablero EsHeterogeneo y EsHeterogeneoCompleto

#include <iostream>
#include <string>

using namespace std;


/*
Explicación de la solución:

   clase SecuenciaEnteros:
      Añadimos el método Suma()

   clase Tablero:
      Las cabeceras de los métodos son:
         bool EsHeterogeneo()
         bool EsHeterogeneoCompleto()

   Para implementar los métodos anteriores
   usamos una secuencia de enteros
      sumas_col
   que contiene las sumas de las columnas

   a) Versión ineficiente -> Se calculan las sumas de TODAS las columnas

      bool EsHeterogeneo()
         for -i-
            suma = Columna(i).Suma()
            sumas_col.Aniade(suma)

         Si hay algún repetido en sumas_col,
            es_heterogeneo = false

      bool EsHeterogeneoCompleto()
         for -i-
            suma = Columna(i).Suma()
            sumas_col.Aniade(suma)

         sumas_col.Ordena()
         Si hay algún elemento no consecutivo en sumas_col,
            es_heterogeneo = false

      Podemos apreciar que la primera parte de EsHeterogeneoCompleto es
      la misma que EsHeterogeneo:

         for -i-
            suma = Columna(i).Suma()
            sumas_col.Aniade(suma)

      Lo que hace el anterior código es calcular una secuencia
      con las sumas de cada columna. Por lo tanto, para no repetir
      dicho código, debemos construir dentro de la clase Tablero
      un método que devuelva dicha secuencia:

         class Tablero{
            ...
            SecuenciaEnteros SumasCol() {
               SecuenciaEnteros sumas_col

               for -i-
                  suma = Columna(i).Suma()
                  sumas_col.Aniade(suma)

               return sumas_col
            }

      El método puede ser privado o público:

         int main(){
            ...
            SecuenciaEnteros sumas_columnas(tablero.SumasCol());  // :-)

      Nos quedaría:

         bool EsHeterogeneo()
```

```
87:             sumas_col = SumasCol()
88:
89:             Si hay algún repetido en sumas_col,
90:                es_heterogeneo = false
91:
92:          bool EsHeterogeneoCompleto()
93:             sumas_col = SumasCol()
94:             sumas_col.Ordena()
95:             Si hay algún elemento no consecutivo en sumas_col,
96:                es_heterogeneo = false
97:
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.PrimeraOcurrencia(suma))
126:                   son_distintos = true
127:                   sumas_col.Aniade(suma)
128:
129:             return sumas_col
130:          }
131:
132:          public:
133:          bool EsHeterogeneo()
134:             es_heterogeneo = SumasCol().Utilizados() == dimension
135:
136:          bool EsHeterogeneoCompleto()
137:             sumas_col = SumasCol()
138:             es_heterogeneo = sumas_col.Utilizados() == dimension
139:
140:             if (es_heterogeneo)
141:                sumas_col.Ordena()
142:                Si hay algún elemento no consecutivo en sumas_col,
143:                   es_heterogeneo = false
144: */
145:
146: class SecuenciaEnteros{
147: private:
148:    static const int TAMANIO = 50;
149:    int v[TAMANIO];
150:    int utilizados;
151: public:
152:    SecuenciaEnteros()
153:       :utilizados(0) {
154:    }
155:
156:    int Utilizados(){
157:       return utilizados;
158:    }
159:
160:    int Capacidad(){
161:       return TAMANIO;
162:    }
163:
164:    void Aniade(int nuevo){
165:       if (utilizados < TAMANIO){
166:          v[utilizados] = nuevo;
167:          utilizados++;
168:       }
169:
170:    }
171:    int Elemento(int indice){
172:       return v[indice];
```

```
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:
```