```
#ifndef _TP1_PROBLEM_H_
#define _TP1_PROBLEM_H_ 1
    #include <list>
 4
     #include <algorithm>
 5
 6
     #include <map>
     #include <cmath>
 8
     #include "Matrix.h"
 9
10
    enum Method {
11
         BAND GAUSSIAN ELIMINATION,
         LU FACTORIZATION,
12
         SIMPLE ALGORITHM,
1.3
         SHERMAN MORRISON
14
15
    };
16
17
     typedef struct Leech {
18
    public:
19
         BDouble x;
         BDouble y;
20
21
         BDouble radio;
22
         BDouble temperature;
23
    } Leech;
2.4
    class Problem {
25
    public:
26
         // Invariante: // h /= 0
27
28
29
         // height /= 0
30
         // width /= 0
         // h | height
31
         // h | width
32
33
         Problem (enum Method method,
                  const BDouble &width,
34
35
                  const BDouble &height.
                  const BDouble &h,
36
37
                  std::list<Leech> &leeches)
38
                  : width(width), height(height), h(h), rows(round(height / h) + 1), columns(round(width / h) + 1),
39
                    dims(rows * columns), leeches(leeches), method(method) {
             std::cerr << "Method: " << this->method << std::endl; std::cerr << "Width: " << this->width << std::endl;
40
41
             std::cerr << "Height: " << this->height << std::endl;
42
             std::cerr << "h: " << this->h << std::endl;
43
             std::cerr << "Discretization rows: " << this->rows << std::endl;</pre>
44
             std::cerr << "Discretization rows." << this->rows << std::endl;
std::cerr << "Total dimensions: " << this->columns << std::endl;</pre>
4.5
46
             std::cerr << "Leeches: " << this->leeches.size() << std::endl;</pre>
47
48
49
50
         Matrix run() {
51
             Matrix system(this->dims, dims, this->columns, this->columns);
             BDouble *b = new BDouble[this->dims];
53
             Matrix temperatures(this->rows, this->columns);
54
55
             switch (method) {
                  case BAND GAUSSIAN ELIMINATION:
56
57
                      band_gaussian_elimination(system, b, temperatures);
58
                      break:
59
                  case LU FACTORIZATION:
60
                      lu factorization(system, b, temperatures);
61
                      break;
62
                  case SIMPLE ALGORITHM:
63
                      simple_algorithm(system, b, temperatures);
                      std::cout << "SLP:" << (double)singular_leeches_count()/(double)this->leeches.size() << std::endl;
64
65
                      break:
                  case SHERMAN MORRISON:
66
                      sherman_morrison_solution(system, b, temperatures);
67
                      std::cout << "SLP: " << (double) singular leeches count()/(double) this->leeches.size() << std::endl;
68
69
70
             delete[] b;
73
             return temperatures;
         }
74
75
76
    private:
77
         void load temperature matrix(BDouble *x, Matrix &temperatures) {
78
              // Cargar los datos en la matriz
79
              for (int i = 0; i < temperatures.rows(); ++i) {</pre>
80
                  for (int j = 0; j < temperatures.columns(); ++j) {</pre>
81
                      temperatures(i, j) = x[(i * temperatures.columns()) + j];
82
83
             }
84
8.5
         void band_gaussian_elimination(Matrix &system, BDouble *b, Matrix &temperatures) {
86
87
             build_system(system, b, this->leeches);
```

```
88
                 std::pair<BDouble *, enum Solutions> solution = gaussian elimination(system, b);
 89
                 std::cout << "CPT: " << critic_point_temperature(system, solution.first) << std::endl;</pre>
 90
 91
                  load temperature matrix(solution.first, temperatures);
 92
                 delete[] solution.first;
 93
 94
 95
            std::pair<BDouble *, enum Solutions> lu resolution(Matrix &L, Matrix &U, BDouble *b) {
                  //Resolvemos el sistema Ly = b
 96
 97
                 std::pair<BDouble *, enum Solutions> partialSolution = forward substitution(L, b);
 98
                  //Resolvemos el sistema Ux = v
 99
                 std::pair<BDouble *, enum Solutions> finalSolution = backward_substitution(U, partialSolution.first);
100
101
                 delete[] partialSolution.first;
102
                  return finalSolution;
103
104
105
            void lu factorization(Matrix &A, BDouble *b, Matrix &temperatures) {
106
107
                 build system(A, b, this->leeches);
108
109
                  // Sea A la matriz del sistema de ecuaciones,
110
                  // factorizamos A = LU con L, U triangulares inferior/superior
111
                 std::pair<Matrix, Matrix> factors = LU_factorization(A);
112
                 std::pair<BDouble *, enum Solutions> finalSolution = lu_resolution(factors.first, factors.second, b);
113
114
                 std::cout << "CPT: " << critic point temperature(A, finalSolution.first) << std::endl;</pre>
115
                  //Cargamos la solucion en la matriz de temperaturas
116
                 load temperature matrix(finalSolution.first, temperatures);
117
                  // Liberamos la memoria que usamos.
118
                 delete[] finalSolution.first;
            }
119
120
            /**
121
122
            * Resuelve el problema por eliminacion gaussiana. En caso de que la temperatura del
            * punto critico sea mayor o igual a 235.0 grados de temperatura, resuelve el sistema
123
124
               por cada sanguijuela, removiendo una de estas y se queda con la menor temperatura.
125
126
            void simple algorithm(Matrix &system, BDouble *b, Matrix &temperatures) {
127
                  // Observamos que sucede con el caso que no borramos sanquijuelas
128
                 build_system(system, b, this->leeches);
129
                 std::pair<BDouble *, enum Solutions> solution = gaussian elimination(system, b);
130
131
                  // Si no hace falta borrar ninguna, terminamos antes.
132
                 BDouble minT = critic_point_temperature(system, solution.first);
133
134
                 if (minT < 235.0) {</pre>
                       std::cout << "CPT: " << minT << std::endl;
135
                       std::cout << "REMOVED LEECH: -1" << std::endl;</pre>
136
137
                       load temperature matrix (solution first, temperatures);
138
                       delete[] solution.first;
139
                       return:
140
141
142
                 delete[] solution.first;
143
144
                  // Valores de salida por defecto
                 BDouble *minX = NULL;
145
146
                 minT = 0.0;
147
                 long taken = -1;
148
                 for (std::list<Leech>::iterator itLeech = leeches.begin(); itLeech != leeches.end(); ++itLeech) {
149
150
                       //Armamos una lista sin la sanguijuela
151
                       std::list<Leech> curLeeches(leeches);
152
                       auto curLeechesIterator = curLeeches.begin();
153
                       std::advance(curLeechesIterator, std::distance(leeches.begin(), itLeech));
154
                       curLeeches.erase(curLeechesIterator);
155
156
                       //Inicializamos el sistema sin la sanguijuela
157
                       //Matrix curSystem(system.rows(), system.columns(), system.lower bandwidth(), system.upper bandwidth());
                       BDouble *curB = new BDouble[system.columns()];
158
159
                       clean system(system);
160
                       build_system(system, curB, curLeeches);
161
162
                       //Resolvemos el sistema
163
                       std::pair<BDouble *, enum Solutions> curSolution = gaussian elimination(system, curB);
164
165
                       BDouble curT = critic point temperature(system, curSolution.first);
166
                       std::cerr << "Removing leech (" << itLeech->x << ", " << itLeech->y << ", " << itLeech->radio << ", " < itLeech->radio << ", " << itLeech->radio << ", " < itLeech->radio << ", " << itLeech->radio </ >
167
168
                       itLeech->temperature << ") gives a critic point temperature of " << curT << std::endl;
169
170
                       // Nos quedamos con la solucion si es mejor que la anterior
171
                       delete[] curB;
172
173
                       if (minX == NULL || curT <= minT) {</pre>
174
                            if (minX != NULL) {
175
                                  delete[] minX;
```

```
176
                      }
177
178
                      minX = curSolution.first;
                      minT = curT;
179
180
                      taken = std::distance(leeches.begin(), itLeech);
181
                  } else {
182
                      delete[] curSolution.first;
183
184
             }
185
186
             // Critic point temperature
             std::cout << "REMOVED_LEECH: " << taken << std::endl;
std::cout << "CPT: " << minT << std::endl;</pre>
187
188
189
             load_temperature_matrix(minX, temperatures);
190
191
             delete[] minX;
192
193
194
         int singular leeches count() {
195
             int singular_count = 0;
196
197
             for (std::list<Leech>::iterator itLeech = leeches.begin(); itLeech != leeches.end(); ++itLeech) {
198
                  Leech leech = *itLeech;
199
200
                  if (is singular leech(leech)) {
201
                      singular_count++;
202
203
204
             return singular count;
205
206
         std::pair<BDouble *, enum Solutions> singular_leech_resolution(Matrix &system, Matrix &L, Matrix &U, BDouble *b,
207
208
                                                                           std::list<Leech> &leeches, const Leech &removed_leech) {
209
              //Nos fijamos si otra sanguijuela afecta la posicion de esta
              int i = round(removed_leech.y / h);
210
211
             int j = round(removed_leech.x / h);
212
213
             //Tratamiento para sanguijuelas singulares (afectan una sola ecuacion)
214
             std::map<std::pair<int, int>, BDouble> affected positions = generate affected positions(leeches);
215
             bool affected position = affected positions.count(std::pair<int, int>(i, j)) >= 1;
216
             std::pair<BDouble *, enum Solutions> solution;
217
218
219
             if (affected_position) {
220
                  //Otra sanguijuela afecta la posicion => No podemos aprovechar sherman-morrison.
221
                  //Utilizamos unicamente la factorizacion LU.
222
                  BDouble newTemperature = affected positions.at(std::pair<int, int>(i, j));
223
224
                  // Inicializamos la solucion del sistema
225
                  BDouble *b2 = new BDouble[system.columns()];
226
                  std::copy(b, b + system.columns(), b2);
227
                  b2[(i * this->columns) + j] = newTemperature;
228
229
                  // Resolvemos utilizando LU
230
                  solution = lu_resolution(L, U, b2);
231
                  delete[] b2;
232
233
234
                  //Podemos aprovechar sherman-morrison!!
235
                  std::pair<BDouble *, BDouble *> uv = generate sherman morrison uv(system, i, j);
236
                  BDouble *u = uv.first;
                  BDouble *v = uv.second;
237
                  BDouble *b2 = generate_sherman_morrison_b(system, b, i, j);
238
239
240
                  //Resolvemos utilizando sherman-morrison
241
                  solution = sherman_morrison(L, U, u, v, b2);
242
243
                  //Liberamos memoria
244
                  delete[] u;
245
                  delete[] v;
246
                  delete[] b2;
247
248
249
             return solution;
250
251
         }
252
253
254
255
         * En caso de que la cantidad de sanquijuelas singulares (afectan una sola ecuacion del sistema discretizado)
256
         * sea menor o igual a 1 resuelve el problema usando simple algorithm.
          * En caso contrario obtiene la factorizacion LU del sistema y separa el tratamiento de sanguijuelas normales
257
         ^{\star} de las sanguijuelas singulares.
258
          * - Si la sanguijuela no es singular resuelve rehaciendo el sistema sin la sanguijuela como en simple_algorithm.
259
         * - Si la sanguijuela es singular a su vez separa en dos casos:
260
261
                - Si la posicion se encuentra afectada por otra sanguijuela, simplemente modifica el valor del vector
              correspondiente por el de mayor temperatura y resuelve utilizando la factorizacion LU.
262
263
               - Si la posicion no se encuentra afectada por otra sanguijuela, resuelve utilizando
```

```
void sherman_morrison_solution(Matrix &system, BDouble *b, Matrix &temperatures) {
2.65
266
267
                         if (singular leeches count() < 2) {</pre>
268
                               std::cerr << "Haciendo fallback al algoritmo simple" << std::endl;</pre>
269
270
                                // Si la cantidad de sanguijuelas singulares es menor es 0 o 1
271
                                // no tiene sentido obtener la factorizacion LU de la matriz.
                                // Basta con utilizar la version simple del metodo
272
273
                               simple_algorithm(system, b, temperatures);
274
                               return:
275
                        }
276
2.77
                        std::list<Leech> singularLeeches;
2.78
                        build_system(system, b, this->leeches);
279
280
                         //Calculamos la factorizacion LU para aprovechar en las sanguijuelas singulares
281
                        std::pair<Matrix, Matrix> factors = LU factorization(system);
282
                        Matrix &L = factors.first;
                        Matrix &U = factors.second;
283
284
285
                        //Solucion sin sacar sanguijuela
286
                        std::pair<BDouble *, enum Solutions> solution = lu_resolution(L, U, b);
2.87
288
                        // Si no hace falta borrar ninguna, terminamos antes.
289
                        BDouble minT = critic_point_temperature(system, solution.first);
290
291
                        if (minT < 235.0) {</pre>
292
                               std::cout << "CPT: " << minT << std::endl;
                                std::cout << "REMOVED LEECH: -1" << std::endl;</pre>
293
                               load temperature matrix(solution.first, temperatures);
294
295
                               delete[] solution.first;
296
                               return;
297
                        }
298
299
                        delete[] solution.first;
300
301
                         long taken = -1;
302
                        BDouble *minX = NULL;
303
                        minT = 0.0;
304
                        for (std::list<Leech>::iterator itLeech = leeches.begin(); itLeech != leeches.end(); ++itLeech) {
305
306
                                //Armamos una lista sin la sanguijuela
307
                                std::list<Leech> curLeeches(leeches);
308
                               auto curLeechesIterator = curLeeches.begin();
309
                               std::advance(curLeechesIterator, std::distance(leeches.begin(), itLeech));
310
                               curLeeches.erase(curLeechesIterator);
311
312
                               if (is singular leech(*itLeech)) {
313
                                       //Tratamos a las sanguijuelas singulares aparte
                                       solution = singular_leech_resolution(system, L, U, b, curLeeches, *itLeech);
314
315
                               } else {
                                       //Inicializamos el sistema sin la sanguijuela
316
317
                                       BDouble *curB = new BDouble[system.columns()];
318
                                       clean system(system);
319
                                       build_system(system, curB, curLeeches);
320
321
                                       // Resolvemos el sistema por eliminación gaussiana
322
                                       solution = gaussian elimination(system, curB);
323
324
                                       //Liberamos memoria
325
                                       delete[] curB;
326
                               }
327
328
                               BDouble curT = critic_point_temperature(system, solution.first);
329
330
                               \texttt{std}::\texttt{cerr} << \texttt{"Removing leech} (" << \texttt{itLeech-} \times << ", " << \texttt{itLeech-} \vee << ", " << \texttt{itLeech-} \wedge \texttt{radio} << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " << ", " <
                               itLeech->temperature << ") gives a critic point temperature of " << curT << std::endl;
331
332
333
                               if (curT <= minT || minX == NULL) {</pre>
                                       if (minX != NULL) {
334
335
                                              delete[] minX;
336
337
338
                                       minX = solution.first;
339
                                       minT = curT;
                                       taken = std::distance(leeches.begin(), itLeech);
340
341
                               } else {
                                       delete[] solution.first;
343
                               }
344
                        }
345
346
                        // Critic point temperature
                        std::cout << "REMOVED_LEECH: " << taken << std::endl;
std::cout << "CPT: " << minT << std::endl;</pre>
347
348
349
350
                        load_temperature_matrix(minX, temperatures);
351
                        delete[] minX;
```

2.64

```
352
         }
353
354
355
          * Devuelve true si la sanguijuela solo afecta una ecuacion del sistema.
356
357
         bool is singular leech (Leech leech) {
358
              // Ponemos el rango que vamos a chequear
359
              BDouble topJ = std::min(leech.x + leech.radio, this->width - this->h) / h;
              BDouble bottomJ = std::max(leech.x - leech.radio, this->h) / h;
360
361
              BDouble topI = std::min(leech.y + leech.radio, this->height - this->h) / h;
362
              BDouble bottomI = std::max(leech.y - leech.radio, this->h) / h;
363
364
              int coordinates_count = 0;
365
              for (int i = std::ceil(bottomI); BDouble(double(i)) <= topI; ++i) {</pre>
                  BDouble iA = BDouble(double(i));
366
367
                  for (int j = std::ceil(bottomJ); BDouble(double(j)) <= topJ; ++j) {</pre>
368
369
                      BDouble iJ = BDouble(double(j));
370
                      BDouble coef = std::pow(iA * this->h - leech.y, 2) + std::pow(iJ * this->h - leech.x, 2);
371
                      if (coef <= std::pow(leech.radio, 2)) {</pre>
372
373
                          coordinates_count++;
374
375
376
377
              return coordinates_count == 1;
378
         }
379
380
381
         BDouble *generate_sherman_morrison_b(const Matrix &system, BDouble *b, int leech_x, int leech_x) {
382
              int columns = system.upper bandwidth();
              BDouble *b2 = new BDouble[system.columns()];
383
              std::copy(b, b + system.columns(), b2);
//std::cerr << "b2[" << (leech_y * columns) + leech_x << "] = " << b2[(leech_y * columns) + leech_x] << std::endl;
384
385
386
              b2[(leech_y * columns) + leech_x] = 0.0;
387
              return b2;
388
         }
389
390
391
          std::pair<BDouble *, BDouble *> generate sherman morrison uv(const Matrix &system, int leech y, int leech x) {
392
              //Construimos el vector columna con un vector canonico
393
              //especificando la fila que corresponde a la ecuacion
394
              //donde hay una sanguijuela
395
              BDouble *u = new BDouble[system.rows()];
396
397
              for (int ijEq = 0; ijEq < this->dims; ijEq++) {
                  int i = ijEq / this->columns;
int j = ijEq % this->columns;
398
399
400
401
                  if (i == leech_y && j == leech_x) {
402
                      u[ijEq] = 1.0;
                  } else {
403
404
                      u[ijEq] = 0.0;
405
                  }
406
              }
407
408
              //Armamos el vector fila con un vector especificando
409
              //las columnas donde colocaremos las componentes
410
              //que corresponden a las diferencias finitas
411
              BDouble *v = new BDouble[system.rows()];
412
              for (int ijEq = 0; ijEq < this->dims; ijEq++) {
413
414
                  v[ijEq] = 0.0;
415
416
417
              int i = leech_y;
418
             int j = leech_x;
419
420
              if (j-1 >= 0) {
421
                  v[(i * this->columns) + j - 1] = -0.25;
422
423
424
              if (j+1 < this->columns) {
                  v[(i * this->columns) + j + 1] = -0.25;
425
426
427
              if (i-1 >= 0) {
428
429
                  v[((i-1) * this->columns) + j] = -0.25;
430
431
432
              if (i+1 < this->rows) {
                  v[((i + 1) * this->columns) + j] = -0.25;
433
434
435
436
              return std::pair<BDouble *, BDouble *>(u, v);
437
438
          }
439
```

```
440
          double critic_point_temperature(const Matrix &system, BDouble *solution) {
441
              double centerJ = this->width / 2.0;
442
              double centerI = this->height / 2.0;
443
              double topJ = std::min(centerJ / double(this->h) + 1.0, double(this->width) / double(this->h) - 1.0);
444
445
              double bottomJ = std::max(centerJ / double(this->h) - 1, 1.0);
446
              double topI = std::min(centerI / double(this->h) + 1.0, double(this->height) / double(this->h) - 1.0);
447
              double bottomI = std::max(centerI / double(this->h) - 1.0, 1.0);
448
449
450
              double output = 0.0;
451
              double k = 0;
452
              for (int i = std::ceil(bottomI); i <= std::floor(topI); ++i) {</pre>
453
                  for (int j = std::ceil(bottomJ); j <= std::floor(topJ); ++j) {</pre>
454
455
                      output += solution[i * this->columns + j];
456
                      ++k;
457
              }
458
459
460
              output /= k;
461
462
              return output;
463
         }
464
465
          void clean system(Matrix &system) {
              for (int ijEq = 0; ijEq < this->dims; ijEq++) {
466
467
                  system(ijEq, ijEq) = 0.0;
468
469
                  int bound = std::min(system.upper bandwidth(), system.lower bandwidth());
470
471
                  for (int 1 = 1; 1 <= bound; 1++) {</pre>
472
                      if (ijEq > 1) {
473
                          system(ijEq, ijEq - 1) = 0.0;
474
475
476
                      if (ijEq + 1 < this->dims) {
477
                          system(ijEq, ijEq + 1) = 0.0;
478
479
                  }
480
              }
481
         }
482
483
484
          std::map<std::pair<int, int>, BDouble> generate affected positions(const std::list<Leech> &leeches) const {
485
              std::map<std::pair<int, int>, BDouble> associations;
486
487
              for (auto &leech : leeches) {
488
                  // Ponemos el rango que vamos a generar
                  BDouble topJ = std::min(leech.x + leech.radio, this->width - this->h) / h; BDouble bottomJ = std::max(leech.x - leech.radio, this->h) / h;
489
490
491
492
                  BDouble topI = std::min(leech.y + leech.radio, this->height - this->h) / h;
493
                  BDouble bottomI = std::max(leech.y - leech.radio, this->h) / h;
494
495
                  // Seteamos las temperaturas en la matriz.
496
                  // Cabe destacar, la temperatura de cada sanguijuela es igual para todos los puntos que cubre.
497
                  for (int i = std::ceil(bottomI); BDouble(double(i)) <= topI; ++i) {</pre>
498
                      BDouble iA = BDouble(double(i));
499
500
                      for (int j = std::ceil(bottomJ); BDouble(double(j)) <= topJ; ++j) {</pre>
501
                           BDouble iJ = BDouble(double(j));
502
                           BDouble coef = std::pow(iA * this->h - leech.y, 2) + std::pow(iJ * this->h - leech.x, 2);
503
504
                           if (coef <= std::pow(leech.radio, 2)) {</pre>
505
506
                                   if (associations.at(std::pair<int, int>(i, j)) < leech.temperature) {</pre>
507
                                       associations[std::pair<int, int>(i, j)] = leech.temperature;
508
509
                               } catch (...) {
510
                                   associations[std::pair<int, int>(i, j)] = leech.temperature;
511
512
                          }
513
514
515
516
517
              return associations;
518
          }
519
520
          * Construimos:
521
           ^{\star} - system, la matriz de ecuaciones que representa la relación de las temperaturas.
522
          * - b, el vector de resultados que representa las condiciones del sistema.
523
          * */
524
525
          void build system(Matrix &system, BDouble *b, const std::list<Leech> &leeches) const {
526
             int columns = system.upper_bandwidth();
527
              int rows = system.rows() / columns;
```

```
528
               int limit = columns * rows;
529
530
               std::map<std::pair<int, int>, BDouble> associations = generate_affected_positions(leeches);
531
532
               for (int ijEq = 0; ijEq < limit; ijEq++) {</pre>
                    system(ijEq, ijEq) = 1.0;
533
534
                    int i = ijEq / columns;
535
                    int j = ijEq % columns;
536
                    if (i == 0 || j == 0 || i == rows - 1 || j == columns - 1) {
537
                         //Si esta en el borde el valor esta fijo en -100.0 y no hay que usar
538
539
                         //la ecuacion de laplace
540
                         b[ijEq] = -100.0;
541
542
                    } else {
543
                         try {
544
                              //Si la posicion se encuentra en el radio de una sanguijuela
545
                              //la temperatura que afecta la posicion es la de la sanguijuela
546
                              //y no hay que usar la ecuacion de laplace
547
                             b[ijEq] = associations.at(std::pair<int, int>(i, j));
548
549
                         } catch (...) {
550
                              //Finalmente si no es borde ni sanguijuela, hay que usar la
551
                              //ecuacion de laplace.
552
                              //Las posiciones de los bordes se ignoran porque figuran con -100.0
553
                              //y fija el valor.
                             b[ijEq] = 0.0;
554
555
                             // t[i-1][j] + t[i, j-1] - 4*t[i, j] + t[i+1, j] + t[i, j+1] = 0
// -t[i-1][j] - t[i, j-1] - t[i+1, j] - t[i, j+1] = 0 con t[i, j] = 0
system(ijEq, (i * columns) + j - 1) = -0.25;
556
557
558
                             system(ijEq, (i * columns) + j + 1) = -0.25;
system(ijEq, (i - 1) * columns) + j) = -0.25;
system(ijEq, ((i + 1) * columns) + j) = -0.25;
559
560
561
562
563
                    }
564
               }
565
          }
566
567
          BDouble width;
           BDouble height;
568
          BDouble h:
569
570
           int rows;
571
           int columns;
572
           int dims;
573
           std::list<Leech> leeches;
574
           enum Method method;
575
     };
576
577
578 #endif // TP1 PROBLEM H
```