

```

1  #ifndef _TP1_PROBLEM_H_
2  #define _TP1_PROBLEM_H_ 1
3
4  #include <list>
5  #include <algorithm>
6  #include <map>
7  #include <cmath>
8  #include "Matrix.h"
9
10 enum Method {
11     BAND_GAUSSIAN_ELIMINATION,
12     LU_FACTORIZATION,
13     SIMPLE_ALGORITHM,
14     SHERMAN_MORRISON
15 };
16
17 typedef struct _Leech {
18 public:
19     BDouble x;
20     BDouble y;
21     BDouble radio;
22     BDouble temperature;
23 } Leech;
24
25 class Problem {
26 public:
27     // Invariante:
28     // h != 0
29     // height != 0
30     // width != 0
31     // h | height
32     // h | width
33     Problem(enum Method method,
34             const BDouble &width,
35             const BDouble &height,
36             const BDouble &h,
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) {}
40     std::cerr << "Method: " << this->method << std::endl;
41     std::cerr << "Width: " << this->width << std::endl;
42     std::cerr << "Height: " << this->height << std::endl;
43     std::cerr << "h: " << this->h << std::endl;
44     std::cerr << "Discretization rows: " << this->rows << std::endl;
45     std::cerr << "Discretization columns: " << this->columns << std::endl;
46     std::cerr << "Total dimensions: " << this->dims << std::endl;
47     std::cerr << "Leeches: " << this->leeches.size() << std::endl;
48 }
49
50 Matrix run() {
51     Matrix system(this->dims, dims, this->columns, this->columns);
52     BDouble *b = new BDouble[this->dims];
53     Matrix temperatures(this->rows, this->columns);
54
55     switch (method) {
56     case BAND_GAUSSIAN_ELIMINATION:
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);
64         std::cout << "SLP: " << (double)singular_leeches_count() / (double)this->leeches.size() << std::endl;
65         break;
66     case SHERMAN_MORRISON:
67         sherman_morrison_solution(system, b, temperatures);
68         std::cout << "SLP: " << (double)singular_leeches_count() / (double)this->leeches.size() << std::endl;
69         break;
70     }
71
72     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) {
80         for (int j = 0; j < temperatures.columns(); ++j) {
81             temperatures(i, j) = x[(i * temperatures.columns()) + j];
82         }
83     }
84 }
85
86 void band_gaussian_elimination(Matrix &system, BDouble *b, Matrix &temperatures) {
87     build_system(system, b, this->leeches);

```

```

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

```

```

176         }
177
178         minX = curSolution.first;
179         minT = curT;
180         taken = std::distance(leeches.begin(), itLeech);
181     } else {
182         delete[] curSolution.first;
183     }
184 }
185
186 // Critic point temperature
187 std::cout << "REMOVED LEECH: " << taken << std::endl;
188 std::cout << "CPT: " << minT << std::endl;
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
207 std::pair<BDouble *, enum Solutions> singular_leech_resolution(Matrix &system, Matrix &L, Matrix &U, BDouble *b,
208     std::list<Leech> &leeches, const Leech &removed_leech) {
209     //Nos fijamos si otra sanguijuela afecta la posicion de esta
210     int i = round(removed_leech.y / h);
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
217     std::pair<BDouble *, enum Solutions> solution;
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     } else {
234         //Podemos aprovechar sherman-morrison!!
235         std::pair<BDouble *, BDouble *> uv = generate_sherman_morrison_uv(system, i, j);
236         BDouble *u = uv.first;
237         BDouble *v = uv.second;
238         BDouble *b2 = generate_sherman_morrison_b(system, b, i, j);
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     return solution;
249 }
250
251 }
252
253 /**
254 * En caso de que la cantidad de sanguijuelas singulares (afectan una sola ecuacion del sistema discretizado)
255 * sea menor o igual a 1 resuelve el problema usando simple_algorithm.
256 * En caso contrario obtiene la factorizacion LU del sistema y separa el tratamiento de sanguijuelas normales
257 * 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 *     - Si la posicion se encuentra afectada por otra sanguijuela, simplemente modifica el valor del vector
261 *     correspondiente por el de mayor temperatura y resuelve utilizando la factorizacion LU.
262 *     - Si la posicion no se encuentra afectada por otra sanguijuela, resuelve utilizando

```

```

264  **/
265  void sherman_morrison_solution(Matrix &system, BDouble *b, Matrix &temperatures) {
266
267      if (singular_leeches_count() < 2) {
268          std::cerr << "Haciendo fallback al algoritmo simple" << std::endl;
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.
272          // Basta con utilizar la version simple del metodo
273          simple_algorithm(system, b, temperatures);
274          return;
275      }
276
277      std::list<Leech> singularLeeches;
278      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;
283      Matrix &U = factors.second;
284
285      //Solucion sin sacar sanguijuela
286      std::pair<BDouble *, enum Solutions> solution = lu_resolution(L, U, b);
287
288      // Si no hace falta borrar ninguna, terminamos antes.
289      BDouble minT = critic_point_temperature(system, solution.first);
290
291      if (minT < 235.0) {
292          std::cout << "CPT: " << minT << std::endl;
293          std::cout << "REMOVED_LEECH: -1" << std::endl;
294          load_temperature_matrix(solution.first, temperatures);
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
305      for (std::list<Leech>::iterator itLeech = leeches.begin(); itLeech != leeches.end(); ++itLeech) {
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
314              solution = singular_leech_resolution(system, L, U, b, curLeeches, *itLeech);
315          } else {
316              //Inicializamos el sistema sin la sanguijuela
317              BDouble *curB = new BDouble[system.columns()];
318              clean_system(system);
319              build_system(system, curB, curLeeches);
320
321              // Resolvemos el sistema por eliminaci3n 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          std::cerr << "Removing leech (" << itLeech->x << ", " << itLeech->y << ", " << itLeech->radio << ", " <<
331          itLeech->temperature << ") gives a critic point temperature of " << curT << std::endl;
332
333          if (curT <= minT || minX == NULL) {
334              if (minX != NULL) {
335                  delete[] minX;
336              }
337
338              minX = solution.first;
339              minT = curT;
340              taken = std::distance(leeches.begin(), itLeech);
341          } else {
342              delete[] solution.first;
343          }
344      }
345
346      // Critic point temperature
347      std::cout << "REMOVED_LEECH: " << taken << std::endl;
348      std::cout << "CPT: " << minT << std::endl;
349
350      load_temperature_matrix(minX, temperatures);
351      delete[] minX;

```

```

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;
360     BDouble bottomJ = std::max(leech.x - leech.radio, this->h) / h;
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) {
366         BDouble iA = BDouble(double(i));
367
368         for (int j = std::ceil(bottomJ); BDouble(double(j)) <= topJ; ++j) {
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
372             if (coef <= std::pow(leech.radio, 2)) {
373                 coordinates_count++;
374             }
375         }
376     }
377     return coordinates_count == 1;
378 }
379
380 BDouble *generate_sherman_morrison_b(const Matrix &system, BDouble *b, int leech_y, int leech_x) {
381     int columns = system.upper_bandwidth();
382     BDouble *b2 = new BDouble[system.columns()];
383     std::copy(b, b + system.columns(), b2);
384     //std::cerr << "b2[" << (leech_y * columns) + leech_x << "] = " << b2[(leech_y * columns) + leech_x] << std::endl;
385     b2[(leech_y * columns) + leech_x] = 0.0;
386     return b2;
387 }
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++) {
398         int i = ijEq / this->columns;
399         int j = ijEq % this->columns;
400
401         if (i == leech_y && j == leech_x) {
402             u[ijEq] = 1.0;
403         } else {
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
413     for (int ijEq = 0; ijEq < this->dims; ijEq++) {
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) {
425         v[(i * this->columns) + j + 1] = -0.25;
426     }
427
428     if (i-1 >= 0) {
429         v[((i - 1) * this->columns) + j] = -0.25;
430     }
431
432     if (i+1 < this->rows) {
433         v[((i + 1) * this->columns) + j] = -0.25;
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
444     double topJ = std::min(centerJ / double(this->h) + 1.0, double(this->width) / double(this->h) - 1.0);
445     double bottomJ = std::max(centerJ / double(this->h) - 1, 1.0);
446
447     double topI = std::min(centerI / double(this->h) + 1.0, double(this->height) / double(this->h) - 1.0);
448     double bottomI = std::max(centerI / double(this->h) - 1.0, 1.0);
449
450     double output = 0.0;
451     double k = 0;
452
453     for (int i = std::ceil(bottomI); i <= std::floor(topI); ++i) {
454         for (int j = std::ceil(bottomJ); j <= std::floor(topJ); ++j) {
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) {
466     for (int ijEq = 0; ijEq < this->dims; ijEq++) {
467         system(ijEq, ijEq) = 0.0;
468
469         int bound = std::min(system.upper_bandwidth(), system.lower_bandwidth());
470
471         for (int l = 1; l <= bound; l++) {
472             if (ijEq > l) {
473                 system(ijEq, ijEq - l) = 0.0;
474             }
475
476             if (ijEq + l < this->dims) {
477                 system(ijEq, ijEq + l) = 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
489         BDouble topJ = std::min(leech.x + leech.radio, this->width - this->h) / h;
490         BDouble bottomJ = std::max(leech.x - leech.radio, this->h) / h;
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) {
498             BDouble iA = BDouble(double(i));
499
500             for (int j = std::ceil(bottomJ); BDouble(double(j)) <= topJ; ++j) {
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)) {
505                     try {
506                         if (associations.at(std::pair<int, int>(i, j)) < leech.temperature) {
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 /**
521  * Construimos:
522  * - system, la matriz de ecuaciones que representa la relación de las temperaturas.
523  * - b, el vector de resultados que representa las condiciones del sistema.
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++) {
533         system(ijEq, ijEq) = 1.0;
534         int i = ijEq / columns;
535         int j = ijEq % columns;
536
537         if (i == 0 || j == 0 || i == rows - 1 || j == columns - 1) {
538             //Si esta en el borde el valor esta fijo en -100.0 y no hay que usar
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.
554                 b[ijEq] = 0.0;
555
556                 //  $t[i-1][j] + t[i, j-1] - 4*t[i, j] + t[i+1, j] + t[i, j+1] = 0$ 
557                 //  $- t[i-1][j] - t[i, j-1] - t[i+1, j] - t[i, j+1] = 0$  con  $t[i, j] = 0$ 
558                 system(ijEq, (i * columns) + j - 1) = -0.25;
559                 system(ijEq, (i * columns) + j + 1) = -0.25;
560                 system(ijEq, ((i - 1) * columns) + j) = -0.25;
561                 system(ijEq, ((i + 1) * columns) + j) = -0.25;
562             }
563         }
564     }
565 }
566
567 BDouble width;
568 BDouble height;
569 BDouble h;
570 int rows;
571 int columns;
572 int dims;
573 std::list<Leech> leeches;
574 enum Method method;
575 };
576
577
578 #endif // _TPl_PROBLEM_H_

```