```
 1: /*
 2:  * cobertura.cpp
 3:  *
 4:  *  Created on: Jun 11, 2011
 5:  *      Author: darioandrade
 6:  */
 7:
 8: #include <sys/timeb.h>
 9: #include <time.h>
10: #include <string.h>
11: #include <stdio.h>
12: #include <cstdlib>
13: #include "AlgoritmoCoberturaGulosa.h"
14: #include "DegreeVectorAdjacencyList.h"
15: #include "VertexVectorAdjacencyList.h"
16: #include "DegreeHeapAdjacencyList.h"
17:
18: #include "GeraGrafo.h"
19:
20: typedef long unsigned int ltime;
21:
22: static ltime getMilliCount ( )
23: {
24:     timeb tb;
25:     ftime( &tb );
26:     time_t nCount = tb.millitm + ( tb.time * 1000 );
27:     return nCount;
28: }
29:
30: /*
31: static ltime getMicroCount ( )
32: {
33:     timeval tv;
34:     timezone tz;
35:     tm *tm;
36:     gettimeofday(&tv, &tz);
37:     return tv.tv_usec;
38: }
39: */
40:
41: void FacaTarefa ( AdjacencyList* pGrafo, int debug )
42: {
43:     AdjacencyList& grafo = *pGrafo;
44:
45:     AlgoritmoCoberturaGulosa guloso;
46:
47:     std::list<int> listaDaCobertura;
48:
49:     if ( debug >= 1 )
50:     {
51:         fprintf( stderr, "Calculando cobertura para o grafo com %d vertices\n", grafo.GetSize( ) );
52:     }
53:
54:     guloso.CalculateCobertura( listaDaCobertura, grafo, debug );
55:
56:     if ( debug >= 1 )
57:     {
58:         fprintf( stderr, "Cobertura para o grafo (nvertex: %d) tem %d vertices:\n", grafo.GetSize( ),
59:                 ( int ) listaDaCobertura.size( ) );
60:     }
61:
62:     for ( std::list<int>::iterator it = listaDaCobertura.begin( ); it != listaDaCobertura.end( ); it++ )
63:     {
64:         fprintf( stdout, "%d ", *it );
65:     }
66:
67:     fprintf( stdout, "\n" );
68: }
69:
70: AdjacencyList * EscolhaGrafo ( int tarefa )
71: {
72:     AdjacencyList * pGrafo = NULL;
73:
74:     switch (tarefa)
75:     {
76:         case 3:
77:             pGrafo = new DegreeVectorAdjacencyList( );
78:             break;
79:
80:         case 4:
81:             pGrafo = new DegreeHeapAdjacencyList( );
82:             break;
83:
84:         case 5:
85:             pGrafo = new VertexVectorAdjacencyList( );
86:             break;
87:     }
88:
89:     return pGrafo;
90: }
91:
92: void Benchmark ( int debug )
93: {
94:     struct BenchmarkData {
```

```
 95:         char sFilename[ 256 ];
 96:         int graphtype;
 97:         ltime loadtime, processingtime;
 98:         int algorithm;
 99:         int nvertex;
100:         int nedges;
101:         int ncobertura;
102:     };
103:
104:     BenchmarkData results[ 3 ][ NSEQS ][ NGRAFOS_POR_SEQ ];
105:
106:     for ( int algo = 3; algo <= 5; algo++ )
107:     {
108:         AdjacencyList * pGrafo = EscolhaGrafo( algo );
109:
110:         for ( int c = 0; c < NSEQS; c++ )
111:         {
112:             for ( int i = 1; i <= NGRAFOS_POR_SEQ; i++ )
113:             {
114:                 char path[256];
115:                 ltime deltams;
116:                 BenchmarkData & data = results[ algo - 3 ][ c ][ i - 1 ];
117:
118:                 sprintf( path, "grafo_p%d_%02d.grafo", c, i );
119:
120:                 if ( debug >= 1 )
121:                 {
122:                     fprintf( stderr, "Lendo grafo %s\n", path );
123:                 }
124:
125:                 data.algorithm = algo;
126:                 data.graphtype = c;
127:                 strcpy( data.sFilename, path );
128:
129:                 FILE * f = fopen( path, "r" );
130:
131:                 deltams = getMilliCount( );
132:
133:                 pGrafo->read( f );
134:
135:                 deltams = getMilliCount( ) - deltams;
136:
137:                 fclose( f );
138:
139:                 if ( debug >= 1 )
140:                 {
141:                     fprintf( stderr, "Grafo com %d vertices lido em %lu ms. Calculando cobertura com algoritmo %d\n",
142:                             pGrafo->GetSize( ), deltams, algo );
143:                 }
144:
145:                 data.loadtime = deltams;
146:                 data.nvertex = pGrafo->GetSize( );
147:                 data.nedges = pGrafo->GetEdges( );
148:
149:                 deltams = getMilliCount( );
150:
151:                 std::list<int> listaDaCobertura;
152:
153:                 AlgoritmoCoberturaGulosa guloso;
154:
155:                 guloso.CalculateCobertura( listaDaCobertura, *pGrafo, debug );
156:
157:                 deltams = getMilliCount( ) - deltams;
158:
159:                 if ( debug >= 1 )
160:                 {
161:                     fprintf( stderr, "Cobertura possui %d vertices, calculada em %lu ms\n",
162:                             ( int ) listaDaCobertura.size( ), deltams );
163:                 }
164:
165:                 data.ncobertura = listaDaCobertura.size();
166:                 data.processingtime = deltams;
167:
168:                 fprintf( stdout, "%d,%d,%s,%d,%d,%d,%lu,%lu\n",
169:                         data.algorithm,
170:                         data.graphtype,
171:                         data.sFilename,
172:                         data.nvertex,
173:                         data.nedges,
174:                         data.ncobertura,
175:                         data.loadtime,
176:                         data.processingtime );
177:
178:                 fflush( stdout );
179:             } // end for ngrafos por seq
180:         } // end for seqs
181:
182:         delete pGrafo;
183:     } // end for algos
184: }
185:
186: int main ( int argc, char ** argv )
187: {
188:     if ( argc < 2 )
```

```
189:     {
190:         fprintf( stderr, "Cobertura. Uso: %s (benchmark|(filename ([3|4|5]))) [debug]\n", argv[0] );
191:         exit( 1 );
192:     }
193:
194:     int debug = 1;
195:
196:     if ( strcmp( "benchmark", argv[1] ) == 0 )
197:     {
198:         if ( argc > 2 )
199:         {
200:             debug = atoi( argv[2] );
201:         }
202:
203:         Benchmark( debug );
204:     }
205:     else
206:     {
207:         int tarefa = atoi( argv[2] );
208:
209:         if ( argc > 3 )
210:         {
211:             debug = atoi( argv[3] );
212:         }
213:
214:         AdjacencyList * pGrafo = EscolhaGrafo( tarefa );
215:
216:         char * sFilename = argv[1];
217:
218:         if ( debug >= 1 )
219:         {
220:             fprintf( stderr, "Lendo grafo %s:\n", sFilename );
221:         }
222:
223:         FILE * f = fopen( sFilename, "r" );
224:
225:         pGrafo->read( f, debug );
226:
227:         fclose( f );
228:
229:         FacaTarefa( pGrafo, debug );
230:
231:         delete pGrafo;
232:     }
233:
234:     return 0;
235: }
236:
```