Práctica de PRO2. Árbol filogenético.

Entrega final. VÍCTOR ASENJO CARVAJAL 20-05-2020

Generado por Doxygen 1.8.13

Índice general

Capítulo 1

Árbol filogenético.

Este programa permite construir i modificar el árbol filogenético para un conjunto de N especies utilizando el método conocido como WPGMA (weighted pair group with arithmetic mean). Los comandos que ofrece el programa son:

1. crea_especie: Crea una especie con el identificador y gen (dos strings) dados. Escribe un mensaje de error si ya existe una especie con el mismo identificador. La especie creada, si no hay error, se agrega al conjunto de especies.

2. obtener_gen: Dado un identificador de especie, imprime el gen asociado a la especie. Escribe un mensaje de error si no existe una especie con el identificador dado.

3. distancia: Dados dos identificadores de especies, imprime la distancia entre las dos especies. Se escribe un mensaje de error si alguna de las dos especies cuyos identificadores se dan no existen.

4. elimina_especie: Dado el identificador de una especie e la elimina del conjunto de especies. Escribe un mensaje de error si la especie con el identificador dado no existe.

5. existe_especie: Dado el identificador de una especie e imprime una indicación de si dicha especie existe (es decir, es parte del conjunto de especies).

6. lee_cjt_especies: Lee del canal estándar de entrada un entero n $>=$ 0 y a continuación una secuencia de n especies (pares identificador-gen). Las n especies dadas tienen identificadores distintos entre sí. Los contenidos previos del conjunto de especies se descartan —las especies dejan de existir— y las n especies leídas se agregan al conjunto de especies.

7. imprime_cjt_especies: Imprime en el canal estándar de salida el conjunto de especies. Si el conjunto es vacío, no imprime ninguna información.

8. tabla_distancias: Imprime la tabla de distancias entre cada par de especies del conjunto de especies. Si el conjunto es vacío, no imprime ninguna información.

9. inicializa_clusters: Inicializa el conjunto de clústers con el conjunto de especies en el estado en el que esté en ese momento, e imprime los clústers resultantes, así como la tabla de distancias entre clústers. Al imprimir la tabla de distancias se usarán los identificadores de los clústers para indexar filas y columnas. Si el conjunto es vacío, no imprime ninguna información.

10. ejecuta_paso_wpgma: ejecuta un paso del algoritmo WPGMA (fusiona los dos clústers a menor distancia en uno nuevo) e imprime la tabla de distancias entre clústers resultante. Al imprimir la tabla de distancias se usarán los identificadores de los clústers para indexar filas y columnas. En caso de que el número de clústers del conjunto sea menor o igual que uno solamente se debe imprimir un mensaje de error.

11. imprime_cluster: dado un identificador alpha, imprime el clúster (su “estructura arborescente”) con el identificador dado, o un error si no existe un clúster con dicho identificador en el conjunto de clústers.

12. imprime_arbol_filogenetico: imprime el árbol filogenético para el conjunto de especies actual; dicho árbol es el clúster que agrupa todas las especies, resultante de aplicar el algoritmo WPGMA. El contenido del conjunto de clústers previo se descarta y se reinicializa con el conjunto de especies en el estado en el que esté en ese momento, para a continuación aplicar el algoritmo. El conjunto de clústers final es el que queda después de aplicar el algoritmo. Se imprimirá la estructura arborescente del clúster con los identificadores de los clústers (raíces de los subárboles) y la distancia entre cada clúster y sus hojas descendientes (véase la figura 3; dichas distancias son los números a la izquierda, y se pueden calcular fácilmente a partir de la distancia entre los clústers cuya combinación da origen a cada clúster). El formato preciso en el que se ha de imprimir el árbol se mostrará en los juegos de pruebas públicos. Si el nuevo conjunto de clústers es vacío, solamente se ha de escribir un mensaje de error.

13. fin: finaliza la ejecución del programa.

Capítulo 2

Índice de clases

2.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

Capítulo 3

Indice de archivos

3.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

Capítulo 4

Documentación de las clases

4.1. Referencia de la Clase Cjt_clusters

Representa un conjunto de clusters que se pueden consultar y modificar sus elementos.

Métodos públicos

- Cjt_clusters ()

  Constructora Cjt_clusters.
- ~Cjt_clusters ()

  Destructora de Cjt_clusters.
- void inicializa_clusters (Cjt_especies &cjt_esp)

  A partir del conjunto de especies, imprime la tabla de distancias.
- void ejecuta_paso_wpgma ()

  Ejecuta un paso del algoritmo WPGMA (fusiona los dos clústers a menor distancia en uno nuevo) e imprime la tabla de distancias entre clústers resultante.
- void imprime_cluster (const string &id)

  Imprime el cluster en preorden.
- void imprime_tabla_distancias ()

  Imprime la tabla de distancias entre clústers.
- void imprime_arbol_filogenetico (Cjt_especies &especies)

  Inicializa los clsuters, ejecuta el algoritmo WPGMA por completo e imprime el árbol filogenético para el conjunto de especies actual.

Tipos privados

- typedef map< string, int > Mapa

  La clave es el identificador del clsuter y el valor es el índice del cluster.

Métodos privados

- void vaciar ()

  Borrar todo el contenido del conjunto. Forma parte de la inicialización de clusters.
- void crear_clusters (Cjt_especies &especies)

  Crea los clusters a partir del conjunto de especies y forma parte de la inicialización.
- void crear_tabla_distancias (Cjt_especies &especies)

  Crear la tabla de distancias a partir del conjunto de especies y forma parte de la inicialización.
- void inicializar (Cjt_especies &especies)

  Inicializa los clusters sin imprimir la tabla de distancias.
- void imprime_cluster_rec (int indice)

  Llamada recursiva de imprime_clsuter. Dado un índice válido, imprime el árbol en preorden.
- Cluster encontrar_distancia_minima ()

  Busca la distancia mínima en la tabla de distancias (es parte de ejecutar_paso_wpgma).
- void promediar_distancias (int x, int y)

  Calcula las distancias del nuevo cluster a los clusters restantes.
- void nuevo_cluster (Cluster &minimo)

  Crea un nuevo cluster a aprtir de sus hijos y los borra.
- bool unir_clusters ()

  Ejecuta un paso del algoritmo sin imprimir la tabla de distancias.
- string combinar_ids (int x, int y)

  Dados los índices de los hijos, genera el identificador del padre.

Atributos privados

- Mapa clusters_activos

  Diccionario con los clusters actualmente en juego: la clave es el identificador del clsuter y el valor es el índice del cluster.
- int clusters_sencillos

  Cantidad de clústers con una sola especie.
- int clusters_complejos

  Cantidad de clústers con varias especies.
- int clusters_ambos

  Cantidad de clústers.
- int cluster_actual

  Índice del nuevo clúster.
- vector< Cluster > clusters

  Vector con todos los clsuters.
- vector< vector< double > > distancias

  Tabla de distancias.

4.1.1. Descripción detallada

Representa un conjunto de clusters que se pueden consultar y modificar sus elementos.

Tipo de módulo: datos.

Definición en la línea 27 del archivo Cjt_clusters.hh.

4.1.2. Documentación de los 'Typedef' miembros de la clase

4.1.2.1. Mapa

```
typedef map<string, int> Cjt_clusters::Mapa  [private]
```

La clave es el identificador del clsuter y el valor es el índice del cluster.

Definición en la línea 34 del archivo Cjt_clusters.hh.

4.1.3. Documentación del constructor y destructor

4.1.3.1. Cjt_clusters()

```
Cjt_clusters::Cjt_clusters ( )
```

Constructora Cjt_clusters.

Precondición

Cierto.

Postcondición

El resultado es un conjunto de clusters vacío.

Definición en la línea 12 del archivo Cjt_clusters.cc.

```
13 {
14     clusters_sencillos = 0;
15     clusters_complejos = 0;
16     clusters_ambos = 0;
17
18     cluster_actual = 0;
19 }
```

4.1.3.2. ~Cjt_clusters()

```
Cjt_clusters::~Cjt_clusters ( )
```

Destructora de Cjt_clusters.

Definición en la línea 21 del archivo Cjt_clusters.cc.

```
22 {
23 }
```

4.1.4. Documentación de las funciones miembro

4.1.4.1. inicializa_clusters()

```
void Cjt_clusters::inicializa_clusters (
             Cjt_especies & cjt_esp )
```

A partir del conjunto de especies, imprime la tabla de distancias.

Precondición

Cierto.

Postcondición

Cierto.

Definición en la línea 92 del archivo Cjt_clusters.cc.

```
93 {
94     inicializar(especies);
95     imprime_tabla_distancias();
96 }
```

4.1.4.2. ejecuta_paso_wpgma()

```
void Cjt_clusters::ejecuta_paso_wpgma ( )
```

Ejecuta un paso del algoritmo WPGMA (fusiona los dos clústers a menor distancia en uno nuevo) e imprime la tabla de distancias entre clústers resultante.

Postcondición

Ejecuta un paso del algoritmo e imprmie la tabla de distancias entre clústers resultante.

Definición en la línea 269 del archivo Cjt_clusters.cc.

```
270 {
271     if ( unir_clusters() )
272     {
273         imprime_tabla_distancias();
274     }
275 }
```

4.1.4.3. imprime_cluster()

```
void Cjt_clusters::imprime_cluster (
             const string & id )
```

Imprime el cluster en preorden.

Precondición

Cierto.

Postcondición

Se ha escrito por el canal estándar de salida el clúster con el identificador dado.

Definición en la línea 119 del archivo Cjt_clusters.cc.

```
120 {
121     Mapa::const_iterator it = clusters_activos.find(id);
122
123     if (it == clusters_activos.end())
124     {
125         cout « "ERROR: El cluster " « id « " no existe." « endl;
126         return;
127     }
128
129     imprime_cluster_rec(it->second);
130     cout « endl;
131 }
```

4.1.4.4. imprime_tabla_distancias()

```
void Cjt_clusters::imprime_tabla_distancias ( )
```

Imprime la tabla de distancias entre clústers.

Precondición

Cierto.

Postcondición

Se ha escrito por el canal estándar de salida la tabla de distancias.

Definición en la línea 98 del archivo Cjt_clusters.cc.

```
99 {
100     Mapa::iterator it1;
101     for (it1 = clusters_activos.begin(); it1 != clusters_activos.end(); ++
      it1)
102     {
103         int x = it1->second;
104
105         cout « it1->first « ":";
106
107         Mapa::iterator it2;
108         for (it2 = it1, ++it2; it2 != clusters_activos.end(); ++it2)
109         {
110             int y = it2->second;
111
112             cout « " " « it2->first « " (" « distancias[x][y] « ")";
113         }
114
115         cout « endl;
116     }
117 }
```

4.1.4.5. imprime_arbol_filogenetico()

```
void Cjt_clusters::imprime_arbol_filogenetico (
            Cjt_especies & especies )
```

Inicializa los clsuters, ejecuta el algoritmo WPGMA por completo e imprime el árbol filogenético para el conjunto de especies actual.

Precondición

Cierto.

Postcondición

Se ha escrito por el canal estándar de salida el árbol filogenético para el conjunto de especies actual después de aplicar el algoritmo WPGMA.

Definición en la línea 277 del archivo Cjt_clusters.cc.

```
278 {
279     inicializar(especies);
280
281     if (clusters_sencillos < 1)
282     {
283         cout « "ERROR: El conjunto de clusters es vacio." « endl;
284         return;
285     }
286
287     for (int i = clusters_sencillos; i < clusters_ambos; ++i)
288     {
289         unir_clusters();
290     }
291
292     imprime_cluster_rec(cluster_actual - 1);
293     cout « endl;
294 }
```

4.1.4.6. vaciar()

```
void Cjt_clusters::vaciar ( )  [private]
```

Borrar todo el contenido del conjunto. Forma parte de la inicialización de clusters.

Precondición

Cierto.

Postcondición

Conjunto borrado.

Definición en la línea 25 del archivo Cjt_clusters.cc.

```
26 {
27     clusters_sencillos = 0;
28     clusters_complejos = 0;
29     clusters_ambos = 0;
30
31     cluster_actual = 0;
32
33     clusters.clear();
34     clusters_activos.clear();
35     distancias.clear();
36 }
```

4.1.4.7. crear_clusters()

```
void Cjt_clusters::crear_clusters (
            Cjt_especies & especies )  [private]
```

Crea los clusters a partir del conjunto de especies y forma parte de la inicialización.

Precondición

Cierto.

Postcondición

Cierto.

Definición en la línea 38 del archivo Cjt_clusters.cc.

```
39 {
40     vector<string> ids = especies.obtener_ids();
41
42     // Índices
43     clusters_sencillos = ids.size();
44     clusters_complejos = max(clusters_sencillos - 1, 0);
45     clusters_ambos = clusters_sencillos +
      clusters_complejos;
46
47     // Rellenar con 'c'
48     Cluster c;
49     clusters.resize( clusters_ambos, c );
50
51     // Clusters sencillos
52     for (int i = 0; i < clusters_sencillos; ++i)
53     {
54         clusters[i].modificar_identidad( ids[i], i );
55         clusters_activos[ids[i]] = i;
56     }
57
58     // Cluster actual
59     cluster_actual = clusters_sencillos;
60 }
```

4.1.4.8. crear_tabla_distancias()

```
void Cjt_clusters::crear_tabla_distancias (
            Cjt_especies & especies )  [private]
```

Crear la tabla de distancias a partir del conjunto de especies y forma parte de la inicialización.

Precondición

Cierto.

Postcondición

Cierto.

Definición en la línea 63 del archivo Cjt_clusters.cc.

```
64 {
65     // Rellenar con ceros
66     distancias.resize(clusters_ambos, vector<double>(
       clusters_ambos, 0.0));
67
68     // Distancias sencillas
69     for (int x = 0; x < clusters_sencillos; ++x)
70     {
71         string id_x = clusters[x].consultar_id();
72
73         for (int y = x + 1; y < clusters_sencillos; ++y)
74         {
75             string id_y = clusters[y].consultar_id();
76
77             double d = especies.distancia(id_x, id_y);
78
79             distancias[x][y] = d;
80             distancias[y][x] = d;
81         }
82     }
83 }
```

4.1.4.9. inicializar()

```
void Cjt_clusters::inicializar (
             Cjt_especies & especies )  [private]
```

Inicializa los clusters sin imprimir la tabla de distancias.

Precondición

Cierto.

Postcondición

Cierto.

Definición en la línea 85 del archivo Cjt_clusters.cc.

```
86 {
87     vaciar();
88     crear_clusters(especies);
89     crear_tabla_distancias(especies);
90 }
```

4.1.4.10. imprime_cluster_rec()

```
void Cjt_clusters::imprime_cluster_rec (
            int indice )  [private]
```

Llamada recursiva de imprime_clsuter. Dado un índice válido, imprime el árbol en preorden.

Precondición

Cierto.

Postcondición

Imprime el árbol en preorden.

Definición en la línea 133 del archivo Cjt_clusters.cc.

```
134 {
135     Cluster& c = clusters[indice];
136
137     if ( c.consultar_hijo_izquierdo() < 0 )
138     {
139         cout « "[" « c.consultar_id() « "]";
140         return;
141     }
142
143     cout « "[(" « c.consultar_id() « ", " « c.
      consultar_distancia() « ") ";
144
145     imprime_cluster_rec(c.consultar_hijo_izquierdo());
146     imprime_cluster_rec(c.consultar_hijo_derecho());
147
148     cout « "]";
149 }
```

4.1.4.11. encontrar_distancia_minima()

```
Cluster Cjt_clusters::encontrar_distancia_minima ( )  [private]
```

Busca la distancia mínima en la tabla de distancias (es parte de ejecutar_paso_wpgma).

Precondición

Cierto.

Postcondición

Devuelve un falso cluster con la información de la distancia mínima.

Definición en la línea 166 del archivo Cjt_clusters.cc.

```
167 {
168     double min_distancia = 999.9999;
169     int min_x = -1;
170     int min_y = -1;
171     string min_id = "???";
172 
173     Mapa::iterator it1;
174     for (it1 = clusters_activos.begin(); it1 != clusters_activos.end(); ++
      it1)
175     {
176         int x = it1->second;
177 
178         Mapa::iterator it2;
179         for (it2 = it1, ++it2; it2 != clusters_activos.end(); ++it2)
180         {
181             int y = it2->second;
182 
183             // Comparar:
184             double distancia = distancias[x][y];
185 
186             if ( distancia < min_distancia )
187             {
188                 min_distancia = distancia;
189                 min_x = x;
190                 min_y = y;
191                 min_id = combinar_ids(x, y);
192 
193             }
194             else if ( distancia == min_distancia )
195             {
196                 string id = combinar_ids(x, y);
197 
198                 if ( id < min_id )
199                 {
200                     min_distancia = distancia;
201                     min_x = x;
202                     min_y = y;
203                     min_id = id;
204                 }
205             }
206         }
207     }
208 
209     Cluster minimo;
210     minimo.modificar_hijos(min_x, min_y, min_distancia);
211     return minimo;
212 }
```

4.1.4.12. promediar_distancias()

```
void Cjt_clusters::promediar_distancias (
              int x,
              int y )  [private]
```

Calcula las distancias del nuevo cluster a los clusters restantes.

Precondición

Cierto.

Postcondición

Cierto.

Definición en la línea 239 del archivo Cjt_clusters.cc.

```
240 {
241     for (int i = 0; i < cluster_actual; ++i)
242     {
243         double d_x = distancias[x][i];
244         double d_y = distancias[y][i];
245         double d = (d_x + d_y) / 2.0;
246 
247         distancias[cluster_actual][i] = d;
248         distancias[i][cluster_actual] = d;
249     }
250 }
```

4.1.4.13. nuevo_cluster()

```
void Cjt_clusters::nuevo_cluster (
              Cluster & minimo )  [private]
```

Crea un nuevo cluster a aprtir de sus hijos y los borra.

Precondición

Cierto.

Postcondición

Cierto.

Definición en la línea 214 del archivo Cjt_clusters.cc.

```
215 {
216     int x = minimo.consultar_hijo_izquierdo();
217     int y = minimo.consultar_hijo_derecho();
218     double distancia = minimo.consultar_distancia() / 2.0;
219 
220     string id_x = clusters[x].consultar_id();
221     string id_y = clusters[y].consultar_id();
222 
223     if ( id_x > id_y )
224     {
225         swap(x, y);
226     }
227 
228     string id = combinar_ids(x, y);
229     clusters[cluster_actual].modificar_identidad(id,
      cluster_actual);
230     clusters[cluster_actual].modificar_hijos(x, y, distancia);
231 
232     clusters_activos[id] = cluster_actual;
233     clusters_activos.erase(id_x);
234     clusters_activos.erase(id_y);
235 
236     promediar_distancias(x, y);
237 }
```

4.1.4.14. unir_clusters()

```
bool Cjt_clusters::unir_clusters ( )  [private]
```

Ejecuta un paso del algoritmo sin imprimir la tabla de distancias.

Precondición

Cierto.

Postcondición

Devuelve cierto si había dos cluster para unir y falso si había menos de dos.

Definición en la línea 252 del archivo Cjt_clusters.cc.

```
253 {
254     if ( clusters_activos.size() < 2 )
255     {
256         cout « "ERROR: num_clusters <= 1" « endl;
257         return false;
258     }
259
260     Cluster minimo = encontrar_distancia_minima();
261
262     nuevo_cluster(minimo);
263
264     ++cluster_actual;
265
266     return true;
267 }
```

4.1.4.15. combinar_ids()

```
string Cjt_clusters::combinar_ids (
              int x,
              int y )  [private]
```

Dados los índices de los hijos, genera el identificador del padre.

Precondición

Dos hijos que existen.

Postcondición

Devuelve el identificador del padre.

Definición en la línea 151 del archivo Cjt_clusters.cc.

```
152 {
153     string id_x = clusters[x].consultar_id();
154     string id_y = clusters[y].consultar_id();
155
156     if ( id_x < id_y )
157     {
158         return (id_x + id_y);
159     }
160     else
161     {
162         return (id_y + id_x);
163     }
164 }
```

4.1.5. Documentación de los datos miembro

4.1.5.1. clusters_activos

```
Mapa Cjt_clusters::clusters_activos  [private]
```

Diccionario con los clusters actualmente en juego: la clave es el identificador del clsuter y el valor es el índice del cluster.

Definición en la línea 39 del archivo Cjt_clusters.hh.

4.1.5.2. clusters_sencillos

```
int Cjt_clusters::clusters_sencillos  [private]
```

Cantidad de clústers con una sola especie.

Definición en la línea 45 del archivo Cjt_clusters.hh.

4.1.5.3. clusters_complejos

```
int Cjt_clusters::clusters_complejos  [private]
```

Cantidad de clústers con varias especies.

Definición en la línea 50 del archivo Cjt_clusters.hh.

4.1.5.4. clusters_ambos

```
int Cjt_clusters::clusters_ambos  [private]
```

Cantidad de clústers.

Definición en la línea 55 del archivo Cjt_clusters.hh.

4.1.5.5. cluster_actual

```
int Cjt_clusters::cluster_actual  [private]
```

Índice del nuevo clúster.

Definición en la línea 60 del archivo Cjt_clusters.hh.

4.1.5.6. clusters

```
vector<Cluster> Cjt_clusters::clusters  [private]
```

Vector con todos los clsuters.

Definición en la línea 65 del archivo Cjt_clusters.hh.

4.1.5.7. distancias

```
vector<vector<double> > Cjt_clusters::distancias  [private]
```

Tabla de distancias.

Definición en la línea 70 del archivo Cjt_clusters.hh.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- Cjt_clusters.hh
- Cjt_clusters.cc

4.2. Referencia de la Clase Cjt_especies

Representa la información y las operaciones asociadas a un conjunto de especies.

Métodos públicos

- Cjt_especies ()

  Constructora por defecto.
- ~Cjt_especies ()

  Destructora de Cjt_especies.
- void anyadir_especie (const string &id, const string &gen)

  Añadir especie al conjunto.
- void elimina_especie (const string id)

  Eliminar especie del conjunto.
- void vaciar_cjt ()

  Vacía conjunto de especies.
- string obtener_gen (string id) const

  Obtener gen a partir de un identificador dado.
- vector< string > obtener_ids () const

  Obtener todos los identificadores.
- bool existe_especie (string id) const

  ¿Existe alguna especie con este ID?.
- double distancia (string id1, string id2)

  A partir de dos especies calcula la distancia que hay entre ellas.
- void lee_k ()

  Lee por el canal estándar de entrada la longitud de los meros.
- void lee_especie ()

  Lee por el canal estándar de entrada una especie (par id-gen)
- void lee_cjt_especies ()

  Borra el conjunto de especies que había y lee un listado de especies por el canal estándar de entrada.
- void verifica_especie (const string &id)

  Imprime SI o NO según existe la especie.
- void imprime_cjt_especies () const

  Imprime el cjt_especies.
- void imprime_tabla_distancias ()

  Imprime la tabla de distancias entre cada par de especies del conjunto especies si el conjunto no es vacío.

Tipos privados

- typedef map< string, Especie > Especies

  Diccionario de especies donde la clave es el identificador de la especie y el valor la especie.

Atributos privados

- Especies especies

4.2.1. Descripción detallada

Representa la información y las operaciones asociadas a un conjunto de especies.

Tipo de módulo: datos.

Definición en la línea 24 del archivo Cjt_especies.hh.

4.2.2. Documentación de los 'Typedef' miembros de la clase

4.2.2.1. Especies

```
typedef map<string, Especie> Cjt_especies::Especies  [private]
```

Diccionario de especies donde la clave es el identificador de la especie y el valor la especie.

Definición en la línea 31 del archivo Cjt_especies.hh.

4.2.3. Documentación del constructor y destructor

4.2.3.1. Cjt_especies()

```
Cjt_especies::Cjt_especies ( )
```

Constructora por defecto.

Precondición

Cierto.

Postcondición

El resultado es un conjunto de especies vacío.

Definición en la línea 11 del archivo Cjt_especies.cc.

```
12 {
13 }
```

4.2.3.2. ~Cjt_especies()

```
Cjt_especies::~Cjt_especies ( )
```

Destructora de Cjt_especies.

Definición en la línea 15 del archivo Cjt_especies.cc.

```
16 {
17 }
```

4.2.4. Documentación de las funciones miembro

4.2.4.1. anyadir_especie()

```
void Cjt_especies::anyadir_especie (
              const string & id,
              const string & gen )
```

Añadir especie al conjunto.

Precondición

El parámetro implícito no contiene ninguna especie con el identificador dado.

Postcondición

Se ha añadido la especie "e" al parámetro implícito.

Definición en la línea 116 del archivo Cjt_especies.cc.

```
117 {
118   if (existe_especie(id))
119   {
120       cout « "ERROR: La especie " « id « " ya existe." « endl;
121   }
122
123   else
124   {
125       Especie e(id, gen);
126       especies[id] = e;
127   }
128 }
```

4.2.4.2. elimina_especie()

```
void Cjt_especies::elimina_especie (
              const string id )
```

Eliminar especie del conjunto.

Precondición

El parámetro implícito contiene una especie con el identificador dado.

Postcondición

Se ha eliminado la especie "e" del parámetro implícito.

Definición en la línea 60 del archivo Cjt_especies.cc.

```
61 {
62     if (not existe_especie(id))
63     {
64         cout « "ERROR: La especie " « id « " no existe." « endl;
65     }
66     else
67     {
68         especies.erase(id);
69     }
70 }
```

4.2.4.3. vaciar_cjt()

```
void Cjt_especies::vaciar_cjt ( )
```

Vacía conjunto de especies.

Precondición

Hay un conjunto de especies no vacío.

Postcondición

Devuelve un conjunto de especies vacío.

Definición en la línea 72 del archivo Cjt_especies.cc.

```
73 {
74     especies.clear();
75 }
```

4.2.4.4. obtener_gen()

```
string Cjt_especies::obtener_gen (
             string id ) const
```

Obtener gen a partir de un identificador dado.

Precondición

El id proporcionado existe, de lo contrario devolverá un error.

Postcondición

El resultado es el gen asociado a la especie.

Definición en la línea 77 del archivo Cjt_especies.cc.

```
78 {
79     if (not existe_especie(id))
80     {
81         cout « "ERROR: La especie " « id « " no existe." « endl;
82         return "";
83     }
84     else
85     {
86         Especies::const_iterator it = especies.find(id);
87         return it->second.consultar_gen();
88     }
89 }
```

4.2.4.5. obtener_ids()

```
vector< string > Cjt_especies::obtener_ids ( ) const
```

Obtener todos los identificadores.

Precondición

Cierto.

Postcondición

El resultado es un vector de strings con todas las id del conjunto.

Definición en la línea 91 del archivo Cjt_especies.cc.

```
92 {
93     vector<string> ids;
94     Especies::const_iterator it;
95     for (it = especies.begin(); it != especies.end(); ++it)
96     {
97         ids.push_back(it->second.consultar_id());
98     }
99     return ids;
100 }
```

4.2.4.6. existe_especie()

```
bool Cjt_especies::existe_especie (
             string id ) const
```

¿Existe alguna especie con este ID?.

Precondición

Cierto.

Postcondición

Devuelve cierto si la especie existe y falso si no existe.

Definición en la línea 102 del archivo Cjt_especies.cc.

```
103 {
104     Especies::const_iterator it = especies.find(id);
105     return (it != especies.end());
106 }
```

4.2.4.7. distancia()

```
double Cjt_especies::distancia (
            string id1,
            string id2 )
```

A partir de dos especies calcula la distancia que hay entre ellas.

Precondición

Cierto.

Postcondición

El resultado es la distancia entre las especies a menos que una de las dos no exista.

Definición en la línea 19 del archivo Cjt_especies.cc.

```
20 {
21     // Detección errores
22     if (not existe_especie(id1))
23     {
24         if (existe_especie(id2) or id1 == id2)
25         {
26             cout « "ERROR: La especie " « id1 « " no existe." « endl;
27             return -1.0;
28         }
29         else
30         {
31             cout « "ERROR: La especie " « id1 « " y la especie " « id2 « " no existen." « endl;
32             return -1.0;
33         }
34     }
35     else if (not existe_especie(id2))
36     {
37         cout « "ERROR: La especie " « id2 « " no existe." « endl;
38         return -1.0;
39     }
40
41     // Caso trivial
42     if (id1 == id2)
43     {
44         return 0.0;
45     }
46
47     // Caso general
48     Especies::iterator it1 = especies.find(id1);
49     Especie& e1 = it1->second;
50
51     Especies::iterator it2 = especies.find(id2);
52     Especie& e2 = it2->second;
53
54     double dist = e1.distancia(e2);
55
56     return dist;
57 }
```

4.2.4.8. lee_k()

```
void Cjt_especies::lee_k ( )
```

Lee por el canal estándar de entrada la longitud de los meros.

Precondición

Cierto.

Postcondición

Cierto.

Definición en la línea 108 del archivo Cjt_especies.cc.

```
109 {
110     int k;
111     cin » k;
112
113     Especie::modifica_k(k);
114 }
```

4.2.4.9. lee_especie()

```
void Cjt_especies::lee_especie ( )
```

Lee por el canal estándar de entrada una especie (par id-gen)

Precondición

Cierto.

Postcondición

El resultado es una especie con el ID y gen entrados.

Definición en la línea 130 del archivo Cjt_especies.cc.

```
131 {
132     Especie e;
133     e.lee();
134
135     string id = e.consultar_id();
136     string gen = e.consultar_gen();
137
138     anyadir_especie(id, gen);
139 }
```

4.2.4.10. lee_cjt_especies()

```
void Cjt_especies::lee_cjt_especies ( )
```

Borra el conjunto de especies que había y lee un listado de especies por el canal estándar de entrada.

Precondición

Cierto.

Postcondición

Cierto.

Definición en la línea 141 del archivo Cjt_especies.cc.

```
142 {
143     vaciar_cjt();
144     int n;
145     cin » n;
146     for (int i = 0; i < n; ++i)
147     {
148         lee_especie();
149     }
150 }
```

4.2.4.11. verifica_especie()

```
void Cjt_especies::verifica_especie (
              const string & id )
```

Imprime SI o NO según existe la especie.

Precondición

Cierto.

Postcondición

SI o NO dependiendo de si la especie del parámetro explícito existe dentro del conjunto.

Definición en la línea 152 del archivo Cjt_especies.cc.

```
153 {
154     if (existe_especie(id))
155     {
156         cout « "SI" « endl;
157     }
158     else
159     {
160         cout « "NO" « endl;
161     }
162 }
```

4.2.4.12. imprime_cjt_especies()

```
void Cjt_especies::imprime_cjt_especies ( ) const
```

Imprime el cjt_especies.

Precondición

Cierto.

Postcondición

Se han escrito por el canal estándar de salida las especies del parámetro implícito.

Definición en la línea 164 del archivo Cjt_especies.cc.

```
165 {
166     Especies::const_iterator it;
167     for (it = especies.begin(); it != especies.end(); ++it)
168     {
169         it->second.imprime();
170     }
171 }
```

4.2.4.13. imprime_tabla_distancias()

```
void Cjt_especies::imprime_tabla_distancias ( )
```

Imprime la tabla de distancias entre cada par de especies del conjunto especies si el conjunto no es vacío.

Precondición

Cierto.

Postcondición

Se han escrito por el canal estándar de salida la tabla de distancias entre cada par de especies del conjunto especies.

Definición en la línea 173 del archivo Cjt_especies.cc.

```
174 {
175     Especies::iterator it1;
176     for (it1 = especies.begin(); it1 != especies.end(); ++it1)
177     {
178         Especie& e1 = it1->second;
179         cout « "" « e1.consultar_id() « ":";
180 
181         Especies::iterator it2;
182         for (it2 = it1, ++it2; it2 != especies.end(); ++it2)
183         {
184             Especie& e2 = it2->second;
185             cout « " " « e2.consultar_id() « " ";
186 
187             double dist = e1.distancia(e2);
188             cout « "(" « dist « ")";
189         }
190 
191         cout « endl;
192     }
193 }
```

4.2.5. Documentación de los datos miembro

4.2.5.1. especies

```
Especies Cjt_especies::especies  [private]
```

Definición en la línea 32 del archivo Cjt_especies.hh.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- Cjt_especies.hh
- Cjt_especies.cc

4.3. Referencia de la Clase Cluster

Representa un árbol de especies.

Métodos públicos

- Cluster ()
  Constructora por defecto de un cluster.
- ~Cluster ()
  Destructora de Cluster.
- void modificar_identidad (string id, int indice)
  Asignar un identificador al nodo del cluster y el índice del nodo raíz.
- void modificar_hijos (int izquierdo, int derecho, double distancia)
  Asignar los índices de los dos hijos y la distancia del nodo raíz a cada uno de ellos.
- string consultar_id () const
  Devolver el identificador del nodo raíz.
- int consultar_hijo_izquierdo () const
  Devolver el identificador del hijo izquierdo.
- int consultar_hijo_derecho () const
  Devolver el identificador del hijo derecho.
- double consultar_distancia () const
  Devolver la distancia del nodo raíz a cada uno de sus hijos.

Atributos privados

- string id
  Identificador del nodo raíz del árbol.
- int indice
  Índice del nodo raíz dentro del vector de Clusters.
- int indice_izquierdo
  Índice del nodo izquierdo.
- int indice_derecho
  Índice del nodo derecho.
- double distancia
  La distancia del nodo raíz a cada uno de sus hijos (que es la mitad de la distancia entre los hijos).

4.3.1. Descripción detallada

Representa un árbol de especies.

Tipo de módulo: datos.

Definición en la línea 20 del archivo Cluster.hh.

4.3.2. Documentación del constructor y destructor

4.3.2.1. Cluster()

```
Cluster::Cluster ( )
```

Constructora por defecto de un cluster.

Precondición

Cierto.

Postcondición

El resultado es un cluster vacío.

Definición en la línea 10 del archivo Cluster.cc.

```
11 {
12     this->id = "???";
13 
14     this->indice = -1;
15     this->indice_izquierdo = -1;
16     this->indice_derecho = -1;
17 
18     this->distancia = 0.0;
19 }
```

4.3.2.2. ~Cluster()

```
Cluster::~Cluster ( )
```

Destructora de Cluster.

Definición en la línea 21 del archivo Cluster.cc.

```
22 {
23 }
```

4.3.3. Documentación de las funciones miembro

4.3.3.1. modificar_identidad()

```
void Cluster::modificar_identidad (
              string id,
              int indice )
```

Asignar un identificador al nodo del cluster y el índice del nodo raíz.

Precondición

Cierto.

Postcondición

Cierto.

Definición en la línea 25 del archivo Cluster.cc.

```
26 {
27     this->id = id;
28     this->indice = indice;
29 }
```

4.3.3.2. modificar_hijos()

```
void Cluster::modificar_hijos (
              int izquierdo,
              int derecho,
              double distancia )
```

Asignar los índices de los dos hijos y la distancia del nodo raíz a cada uno de ellos.

Precondición

Cierto.

Postcondición

Cierto.

Definición en la línea 31 del archivo Cluster.cc.

```
32 {
33     this->indice_izquierdo = izquierdo;
34     this->indice_derecho = derecho;
35
36     this->distancia = distancia;
37 }
```

4.3.3.3. consultar_id()

```
string Cluster::consultar_id ( ) const
```

Devolver el identificador del nodo raíz.

Precondición

Cierto.

Postcondición

Devuelve el identificador del nodo raíz.

Definición en la línea 39 del archivo Cluster.cc.

```
40 {
41     return id;
42 }
```

4.3.3.4. consultar_hijo_izquierdo()

```
int Cluster::consultar_hijo_izquierdo ( ) const
```

Devolver el identificador del hijo izquierdo.

Precondición

Cierto.

Postcondición

Devuelve el identificador del hijo izquierdo.

Definición en la línea 44 del archivo Cluster.cc.

```
45 {
46     return indice_izquierdo;
47 }
```

4.3.3.5. consultar_hijo_derecho()

```
int Cluster::consultar_hijo_derecho ( ) const
```

Devolver el identificador del hijo derecho.

Precondición

Cierto.

Postcondición

Devuelve el identificador del hijo derecho.

Definición en la línea 49 del archivo Cluster.cc.

```
50 {
51     return indice_derecho;
52 }
```

4.3.3.6. consultar_distancia()

```
double Cluster::consultar_distancia ( ) const
```

Devolver la distancia del nodo raíz a cada uno de sus hijos.

Precondición

Cierto.

Postcondición

Devuelve la distancia del nodo raíz a cada uno de sus hijos.

Definición en la línea 54 del archivo Cluster.cc.

```
55 {
56     return distancia;
57 }
```

4.3.4. Documentación de los datos miembro

4.3.4.1. id

```
string Cluster::id  [private]
```

Identificador del nodo raíz del árbol.

Definición en la línea 27 del archivo Cluster.hh.

4.3.4.2. indice

```
int Cluster::indice  [private]
```

Índice del nodo raíz dentro del vector de Clusters.

Definición en la línea 32 del archivo Cluster.hh.

4.3.4.3. indice_izquierdo

```
int Cluster::indice_izquierdo  [private]
```

Índice del nodo izquierdo.

Definición en la línea 37 del archivo Cluster.hh.

4.3.4.4. indice_derecho

```
int Cluster::indice_derecho  [private]
```

Índice del nodo derecho.

Definición en la línea 42 del archivo Cluster.hh.

4.3.4.5. distancia

```
double Cluster::distancia  [private]
```

La distancia del nodo raíz a cada uno de sus hijos (que es la mitad de la distancia entre los hijos).

Definición en la línea 47 del archivo Cluster.hh.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- Cluster.hh
- Cluster.cc

4.4. Referencia de la Clase Especie

Representa el conjunto de características y operaciones de las especies.

Métodos públicos

- Especie ()

  Constructora Especie vacía.
- Especie (string id, string gen)

  Constructora Especie.
- ∼Especie ()

  Destructora de Especie.
- string consultar_id () const

  Consultar id de la especie del parámetro implícito.
- string consultar_gen () const

  Consultar gen de la especie del parámetro implícito.
- void generar_meros ()

  Construir diccionario de mero a partir del gen y la longitud k.
- int num_meros ()

  Cantidad de meros diferentes dentro del gen implícito.
- int ocurrencias_mero (string mero)

  Dado un mero devuelve el contador.
- double distancia (Especie &other)

  A partir de dos especies calcula la distancia que hay entre ellas.
- void lee ()

  Lee por el canal estándar de entrada una especie (par identificador-gen).
- void imprime () const

  Imprime la especie del parámetro implícito.

Métodos públicos estáticos

- static void modifica_k (int k)
- static int consulta_k ()

Tipos privados

- typedef map< string, int > Meros

  Diccionario de los kmeros con contadores de cada mero.

Atributos privados

- string id

  Identificador de la especie.
- string gen

  Gen de la especie.
- Meros meros

Atributos privados estáticos

- static int k = 3

  Longitud de los meros.

4.4.1. Descripción detallada

Representa el conjunto de características y operaciones de las especies.

Tipo de módulo: datos.

Definición en la línea 23 del archivo Especie.hh.

4.4.2. Documentación de los 'Typedef' miembros de la clase

4.4.2.1. Meros

```
typedef map<string, int> Especie::Meros  [private]
```

Diccionario de los kmeros con contadores de cada mero.

Definición en la línea 39 del archivo Especie.hh.

4.4.3. Documentación del constructor y destructor

4.4.3.1. Especie() [1/2]

```
Especie::Especie ( )
```

Constructora Especie vacía.

Precondición

Cierto.

Postcondición

El resultado es una especie sin parámetros.

Definición en la línea 28 del archivo Especie.cc.

```
29 {
30 }
```

4.4.3.2. Especie() [2/2]

```
Especie::Especie (
            string id,
            string gen )
```

Constructora Especie.

Precondición

Cierto.

Postcondición

El resultado es una especie con el id y gen proporcionados.

Definición en la línea 32 del archivo Especie.cc.

```
33 {
34   this->id = id;
35   this->gen = gen;
36     generar_meros();
37 }
```

4.4.3.3. ∼Especie()

```
Especie::∼Especie ( )
```

Destructora de Especie.

Definición en la línea 39 del archivo Especie.cc.

```
40 {
41 }
```

4.4.4. Documentación de las funciones miembro

4.4.4.1. modifica_k()

```
void Especie::modifica_k (
            int k )  [static]
```

obtener la longitud de los meros

Definición en la línea 16 del archivo Especie.cc.

```
17 {
18     Especie::k = k;
19 }
```

4.4.4.2. consulta_k()

```
int Especie::consulta_k ( )  [static]
```

Definición en la línea 21 del archivo Especie.cc.

```
22 {
23     return Especie::k;
24 }
```

4.4.4.3. consultar_id()

```
string Especie::consultar_id ( ) const
```

Consultar id de la especie del parámetro implícito.

Precondición

Cierto.

Postcondición

El resultado es el id del parámetro implícito.

Definición en la línea 44 del archivo Especie.cc.

```
45 {
46   return id;
47 }
```

4.4.4.4. consultar_gen()

```
string Especie::consultar_gen ( ) const
```

Consultar gen de la especie del parámetro implícito.

Precondición

Cierto.

Postcondición

El resultado es el gen del parámetro implícito.

Definición en la línea 49 del archivo Especie.cc.

```
50 {
51   return gen;
52 }
```

4.4.4.5. generar_meros()

```
void Especie::generar_meros ( )
```

Construir diccionario de mero a partir del gen y la longitud k.

Precondición

Cierto.

Postcondición

Cierto.

Definición en la línea 60 del archivo Especie.cc.

```
61 {
62     meros.clear();
63 
64   for (int i = 0; i < num_meros(); ++i)
65   {
66         string mero = gen.substr(i, Especie::k);
67         Meros::iterator it = meros.find(mero);
68 
69         if (it == meros.end())
70         {
71             meros[mero] = 1;
72         }
73         else
74         {
75             ++meros[mero];
76         }
77   }
78 }
```

4.4.4.6. num_meros()

```
int Especie::num_meros ( )
```

Cantidad de meros diferentes dentro del gen implícito.

Precondición

Cierto.

Postcondición

Número de meros diferentes dentro del gen implícito.

Definición en la línea 54 del archivo Especie.cc.

```
55 {
56     int num = gen.size() + 1 - Especie::k;
57     return max(num, 0);
58 }
```

4.4.4.7. ocurrencias_mero()

```
int Especie::ocurrencias_mero (
            string mero )
```

Dado un mero devuelve el contador.

Precondición

Cierto.

Postcondición

Devuelve el contador de meros.

Definición en la línea 80 del archivo Especie.cc.

```
81 {
82   Meros::iterator it = meros.find(mero);
83   if (it == meros.end())
84   {
85         return 0;
86   }
87   return meros[mero];
88 }
```

4.4.4.8. distancia()

```
double Especie::distancia (
            Especie & other )
```

A partir de dos especies calcula la distancia que hay entre ellas.

Precondición

Cierto.

Postcondición

El distancia a otra especie desde el parámetro implícito.

Definición en la línea 90 del archivo Especie.cc.

```
91 {
92     int num_interseccion = 0;
93
94     Meros::iterator it;
95     for (it = meros.begin(); it != meros.end(); ++it)
96     {
97         string mero = it->first;
98         int ocur1 = ocurrencias_mero(mero);
99         int ocur2 = other.ocurrencias_mero(mero);
100
101          num_interseccion += min(ocur1, ocur2);
102     }
103
104     int num_union = num_meros() + other.num_meros() - num_interseccion;
105
106     //cout « num_interseccion « "/" « num_union « "=";
107
108     double fraccion = (double) num_interseccion / (double) num_union;
109
110     return (( 1.0 - fraccion ) * 100.0);
111 }
```

4.4.4.9. lee()

```
void Especie::lee ( )
```

Lee por el canal estándar de entrada una especie (par identificador-gen).

Precondición

Cierto.

Postcondición

Cierto.

Definición en la línea 113 del archivo Especie.cc.

```
114 {
115   cin » id;
116     cin » gen;
117   generar_meros();
118 }
```

4.4.4.10. imprime()

```
void Especie::imprime ( ) const
```

Imprime la especie del parámetro implícito.

Precondición

Cierto.

Postcondición

Se ha escrito por el canal estándar de salida la especie del parámetro implícito.

Definición en la línea 120 del archivo Especie.cc.

```
121 {
122     cout « id « " " « gen « endl;
123 }
```

4.4.5. Documentación de los datos miembro

4.4.5.1. id

```
string Especie::id  [private]
```

Identificador de la especie.

Definición en la línea 30 del archivo Especie.hh.

4.4.5.2. gen

```
string Especie::gen  [private]
```

Gen de la especie.

Definición en la línea 34 del archivo Especie.hh.

4.4.5.3. meros

```
Meros Especie::meros  [private]
```

Definición en la línea 40 del archivo Especie.hh.

4.4.5.4. k

```
int Especie::k = 3  [static], [private]
```

Longitud de los meros.

Definición en la línea 46 del archivo Especie.hh.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- Especie.hh
- Especie.cc

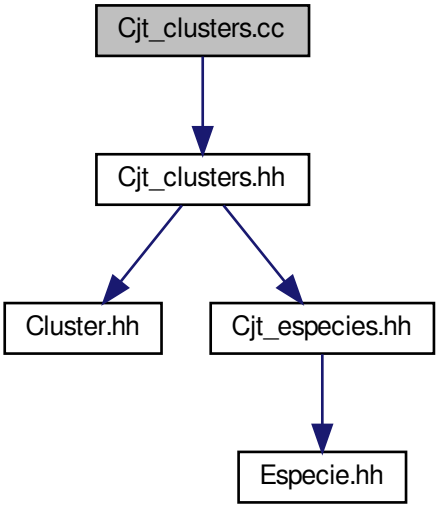
Capítulo 5

Documentación de archivos

5.1. Referencia del Archivo Cjt_clusters.cc

Código de la clase Cjt_clusters.

Dependencia gráfica adjunta para Cjt_clusters.cc:



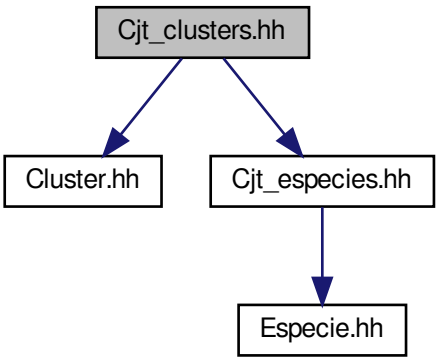
5.1.1. Descripción detallada

Código de la clase Cjt_clusters.

5.2. Referencia del Archivo Cjt_clusters.hh

Especificación de la clase Cjt_clusters.

Dependencia gráfica adjunta para Cjt_clusters.hh:



Clases

- class Cjt_clusters

  Representa un conjunto de clusters que se pueden consultar y modificar sus elementos.

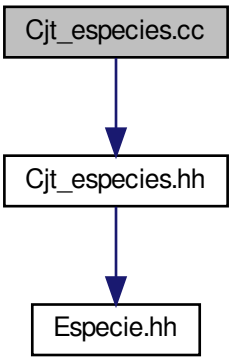
5.2.1. Descripción detallada

Especificación de la clase Cjt_clusters.

5.3. Referencia del Archivo Cjt_especies.cc

Código de la clase Cjt_especies.

Dependencia gráfica adjunta para Cjt_especies.cc:



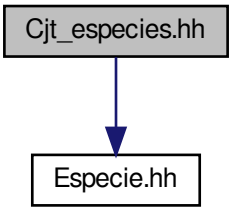
5.3.1. Descripción detallada

Código de la clase Cjt_especies.

5.4. Referencia del Archivo Cjt_especies.hh

Especificación de la clase Cjt_especies.

Dependencia gráfica adjunta para Cjt_especies.hh:



Clases

- class Cjt_especies

  Representa la información y las operaciones asociadas a un conjunto de especies.

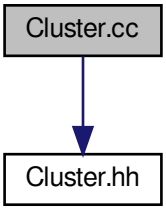
5.4.1. Descripción detallada

Especificación de la clase Cjt_especies.

5.5. Referencia del Archivo Cluster.cc

Código de la clase Cluster.

Dependencia gráfica adjunta para Cluster.cc:



5.5.1. Descripción detallada

Código de la clase Cluster.

5.6. Referencia del Archivo Cluster.hh

Especificación de la clase Cluster.

Clases

- class Cluster

  Representa un árbol de especies.

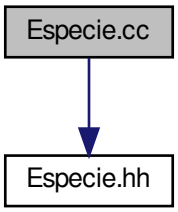
5.6.1. Descripción detallada

Especificación de la clase Cluster.

5.7. Referencia del Archivo Especie.cc

Código de la clase Especie.

Dependencia gráfica adjunta para Especie.cc:



5.7.1. Descripción detallada

Código de la clase Especie.

5.8. Referencia del Archivo Especie.hh

Especificación de la clase Especie.

Clases

- class Especie

  Representa el conjunto de características y operaciones de las especies.

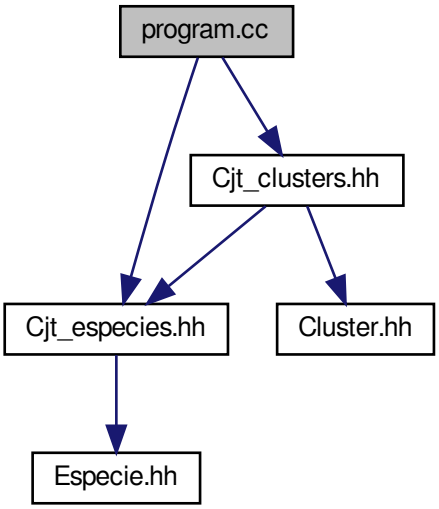
5.8.1. Descripción detallada

Especificación de la clase Especie.

5.9. Referencia del Archivo program.cc

Programa principal de la Práctica de PRO2 cuatrimestre de primavera 2020.

Dependencia gráfica adjunta para program.cc:



Funciones

- int main ()

5.9.1. Descripción detallada

Programa principal de la Práctica de PRO2 cuatrimestre de primavera 2020.

5.9.2. Documentación de las funciones

5.9.2.1. main()

```
int main ( )
```

Definición en la línea 15 del archivo program.cc.

```
16 {
17     Cjt_especies cjt_esp;
18     Cjt_clusters cjt_clus;
19
20     cjt_esp.lee_k();
21
22     bool fin = false;
23
24     while (not fin)
25     {
26         string comando;
27         cin » comando;
28
29         if (comando == "crea_especie")
30         {
31             string id, gen;
32             cin » id » gen;
33             cout « "# " « comando « " " « id « " " « gen « endl;
34             cjt_esp.anyadir_especie(id, gen);
35         }
36         else if (comando == "obtener_gen")
37         {
38             string id;
39             cin » id;
40
41             cout « "# " « comando « " " « id « endl;
42
43             string gen = cjt_esp.obtener_gen(id);
44             if ( gen != "" )
45             {
46                 cout « gen « endl;
47             }
48         }
49         else if (comando == "distancia")
50         {
51             string id1, id2;
52             cin » id1 » id2;
53
54             cout « "# " « comando « " " « id1 « " " « id2 « endl;
55
56             double dist = cjt_esp.distancia(id1, id2);
57             if (dist != -1.0)
58             {
59                 cout « dist « endl;
60             }
61         }
62         else if (comando == "elimina_especie")
63         {
64             string id;
65             cin » id;
66
67             cout « "# " « comando « " " « id « endl;
68             cjt_esp.elimina_especie(id);
69         }
70         else if (comando == "existe_especie")
71         {
72             string id;
73             cin » id;
74             cout « "# " « comando « " " « id « endl;
75             cjt_esp.verifica_especie(id);
76         }
77         else if (comando == "lee_cjt_especies")
78         {
79             cout « "# " « comando « endl;
80             cjt_esp.lee_cjt_especies();
81         }
82         else if (comando == "imprime_cjt_especies")
83         {
84             cout « "# " « comando « endl;
85             cjt_esp.imprime_cjt_especies();
86         }
87         else if (comando == "tabla_distancias")
88         {
89             cout « "# " « comando « endl;
90             cjt_esp.imprime_tabla_distancias();
91         }
92         else if (comando == "inicializa_clusters")
93         {
94             cout « "# " « comando « endl;
95             cjt_clus.inicializa_clusters(cjt_esp);
96         }
97         else if (comando == "ejecuta_paso_wpgma")
98         {
99             cout « "# " « comando « endl;
100             cjt_clus.ejecuta_paso_wpgma();
101         }
102         else if (comando == "imprime_cluster")
```

```
103         {
104             string id;
105             cin » id;
106             cout « "# " « comando « " " « id « endl;
107             cjt_clus.imprime_cluster(id);
108         }
109         else if (comando == "imprime_arbol_filogenetico")
110         {
111             cout « "# " « comando « endl;
112             cjt_clus.imprime_arbol_filogenetico(cjt_esp);
113         }
114         else if (comando == "fin")
115         {
116             fin = true;
117         }
118         else
119         {
120             cout « "ERROR: Comando no reconocido." « endl;
121             fin = true;
122         }
123 
124         // Salto de línea
125         if (not fin)
126         {
127             cout « endl;
128         }
129     }
130 }
```

Índice alfabético