

```

1 |-----TP1 SimCity-----
2 Grupo: Algonautas
3 Materia: Algoritmos 2
4 Integrantes:
5   Gomez Fabian    LU: 522/20
6   Lequio Santiago LU: 644/20
7   Massaccesi Laura LU: 1053/21
8   Velasquez Alison LU: 566/20
9
10 Año: 2022
11 -----
12 Decisiones:
13 1. Cuando hay dos casas ocupan la misma posición en diferentes SimCitys,
14     al unirlos nos quedamos con la casa de mayor nivel
15 2. Cuando hay una casa en un SimCity que ocupa la misma posición que un comercio
16     en otro SimCity, al unirlos nos quedamos con la casa
17 -----
18
19 TAD posicion es TUPLA (Nat x Nat)
20 TAD construccion es string
21 TAD nombre es string
22 TAD nivel es nat
23
24 TAD Mapa
25   Igualdad Observacional:
26   ( $\forall m, m' : \text{Mapa}$ ) ( $m =_{\text{obs}} m' \iff (\text{horizontales}(m) =_{\text{obs}} \text{horizontales}(m') \wedge$ 
27    $\text{vertiales}(m) =_{\text{obs}} \text{verticales}(m'))$ ;
28   Géneros: Mapa
29   Exporta: nombreTad, observadores, generadores,
30   Usa: Bool, Nat
31
32   Observadores Básicos:
33     horizontales : Mapa  $\rightarrow$  conj(Nat)
34     verticales   : Mapa  $\rightarrow$  conj(Nat)
35   Generadores:
36     crear : conj(Nat)  $\times$  conj(Nat)  $\rightarrow$  Mapa
37   Axiomas:
38     horizontales(crear(hs, vs))  $\equiv$  hs
39     verticales(crear(hs, vs))  $\equiv$  vs
40 Fin TAD
41
42
43
44 TAD SIMCITY
45   Igualdad Observacional: ( $\forall s, s' : \text{simcity}$ ) ( $s =_{\text{obs}} s' \iff$ 
46   ( $\text{mapa}(s) =_{\text{obs}} \text{mapa}(s') \wedge$ 
47    $\text{casas}(s) =_{\text{obs}} \text{casas}(s') \wedge$ 
48    $\text{comercios}(s) =_{\text{obs}} \text{comercios}(s') \wedge$ 
49    $\text{popularidad}(s) =_{\text{obs}} \text{popularidad}(s'))$ )
50
51
52   Géneros: simcity
53   Exporta: nombreTad, observadores, generadores,
54   Usa: Bool, Nat

```

```

55
56
57 Observadores Básicos:
58     mapa      : simcity -> m
59     casas     : simcity -> dicc(pos, nivel)
60     comercios : simcity -> dicc(pos, nivel)
61     popularidad : simcity -> Nat
62 Generadores:
63     iniciar    : mapa                                -> simcity
64     avanzarTurno : simcity s × dicc(pos, construccion) cs -> simcity
65         {restriccionesAvanzarTurno(s, cs)}
66     unir       : simcity a × simcity b                -> simcity
67         {restriccionesUnir(a, b)}
68 Otras Operaciones:
69     turnos : simcity -> nat
70
71     esCasa?: construccion constr -> bool
72     esComercio?: construccion constr -> bool
73
74     darCasas: simcity s × conj(pos) c × dicc(pos, construccion) cs -> dicc(pos, nivel)
75         {c ⊆ claves(cs)}
76     dameConstruccion: Pos p × dicc(pos, construccion) cs ->
77         • construccion
78             {def?(p, cs)}
79     aumentarTurnoAConstr: conj(pos) cc × dicc(pos, nivel) dc -> dicc(pos, nivel)
80         {cc ⊆ claves(dc)}
81     darComercios: simcity s × conj(pos) c × dicc(pos, construccion) cs -> dicc(pos,
82         • nivel)
83             {c ⊆ claves(cs)}
84     actuConDistManhattan: dicc(pos, nivel) dcs × conj(pos) c × dicc(pos, nivel) dcm ->
85         • dicc(pos, nivel)
86             {c ⊆ claves(dcm)}
87     haycasaAdistManhattan?: conj(pos) × pos -> bool
88     casasDistManhattan?: conj(pos) × pos -> conj(pos)
89     absDif: nat × nat -> nat
90
91     darUnion: dicc(pos, nivel) × dicc(pos, nivel) db × conj(pos) cb -> dicc(pos,
92         • nivel)
93         {cb ⊆ claves(db)}
94     borrarClaves: conj(pos) × dicc(pos, nivel) -> dicc(pos, nivel)
95
96     restriccionesAvanzarTurno: simcity s × dicc(pos, construccion) cs -> bool
97     restriccionesUnir: simcity × simcity -> bool
98     sonConstrucciones: dicc(pos, construccion) cs -> bool
99     //Los rios de a no se superponen con las construcciones de b
100    noSuperponenRios: simcity × simcity -> bool
101    //Los rios no se superponen con las construcciones
102    nuevasNoSuperponenRios: simcity × conj(pos) -> bool
103    //Las construcciones de a no se superponen con la maxima const de b
104    noSuperponenConstMax: simcity × simcity -> bool
105    construcciones: simcity -> dicc(pos, nivel)
106    auxnoSuperponenPos: conj × conj -> conj
107    auxMaxNivel: conj × dicc(pos, nivel) -> nivel
108    auxMaxConst: conj × dicc(pos, nivel) -> conj

```

```

106 Axiomas:
107     esCasa?(constr) ≡ constr = "casa"
108     esComercio?(constr) ≡ constr = "comercio"
109
110     mapa(iniciar(m)) ≡ m
111     mapa(avanzarTurno(s, dc)) ≡ mapa(s)
112     mapa(unir(a, b)) ≡
113         crear(horizontales(a) ∪ horizontales(b), verticales(a) ∪ verticales(b))
114
115     casas(iniciar(m)) ≡ vacio
116     casas(avanzarTurno(s, dc)) ≡ darCasas(s, claves(dc), dc)
117
118     darCasas(s, cd, dc) ≡
119         if vacía?(cd) then
120             aumentarTurnoAConstr(casas(s), claves(casas(s)))
121         else
122             if esCasa?(dameConstruccion(dameUno(cd), dc)) then
123                 definir(dameUno(cd), 1, darCasas(s, sinUno(cd), dc))
124             else
125                 darCasas(s, sinUno(cd), dc)
126             fi
127         fi
128
129     aumentarTurnoAConstr(cc, dc) ≡
130         if vacía?(cc) then
131             dc
132         else
133             definir(dameUno(cc), obtener(dameUno(cc), dc) + 1, aumentarTurnoAConstr(dc,
134             • sinUno(cc)))
135         fi
136
137     casas(unir(a, b)) ≡ darUnion(casas(a), casas(b), claves(casas(b)))
138
139     darUnion(da, db, cb) ≡
140         if vacío?(cb) then
141             da
142         else
143             if def?(dameUno(cb), da) ∧L obtener(dameUno(cb), da) > obtener(dameUno(cb),
144             • db) then
145                 darUnion(da, db, sinUno(cb))
146             else
147                 definir(dameUno(cb), obtener(dameUno(cb), db), darUnion(da, db, sinUno(cb)))
148             fi
149         fi
150
151     comercios(iniciar(m)) ≡ vacio
152
153     comercios(avanzarTurno(s, dc)) ≡
154         actuConDistManhattan(darCasas(s, claves(dc), dc), claves(darComercios(s, cd, dc)),
155         • darComercios(s, cd, dc))
156
157

```

```

157
158     darComercios(s, cd, dc) ≡ //todos los comercios ya agregados
159     if vacía?(cd) then
160         aumentarTurnoAConstr(comercios(s), claves(comercios(s)))
161     else
162         if esComercio?(dameConstrucción(dameUno(cd), dc)) then
163             definir(dameUno(cd), 1, darComercios(s, sinUno(cd), dc))
164         else
165             darComercios(s, sinUno(cd), dc)
166         fi
167     fi
168
169     actuConDistManhattan(dc, cpos, dcA) ≡
170     if vacía?(cpos) then
171         dcA
172     else
173         if obtener(dameUno(cpos), dcA) < auxMaxNivel(casasDistManhattan(claves(dc),
174             • dameUno(cpos)), dc) then
175             definir(dameUno(cpos), auxMaxNivel(casasDistManhattan(claves(dc),
176             • dameUno(cpos)), dc), dcA)
177         else
178             actuConDistManhattan(dc, sinUno(cpos), dcA)
179         fi
180     fi
181
182     casasDistManhattan?(c, p) ≡
183     if vacío?(c) then
184         ∅
185     else
186         if (absDif(π1(dameUno(c)), π1(p)) + absDif(π2(dameUno(c)), π2(p))) ≤ 3 then
187             Ag(dameUno(c), casasDistManhattan?(sinUno(c), p))
188         else
189             casasDistManhattan?(sinUno(c), p)
190         fi
191     fi
192
193     absDif(n, m) ≡
194     if n < m then
195         m - n
196     else
197         n - m
198     fi
199
200     comercios(unir(a, b)) ≡
201         borrarClaves(claves(darUnionCasas(casas(a), casas(b), claves(casas(b)))), dc),
202         • darUnion(comercios(a), comercios(b), claves(comercios(b))))
203
204     borrarClaves(c, dc) ≡
205     if vacío?(c) then
206         dc
207     else
208         if def?(dameUno(c), dc) then
209             borrar(dameUno(c), borrarClaves(sinUno(c), dc))
210         else
211             borrarClaves(sinUno(c), dc)

```

```

209     fi
210   fi
211
212   popularidad(iniciar(m)) ≡ 0
213   popularidad(avanzarTurno(s, dc)) ≡ popularidad(s)
214   popularidad(unir(a, b)) ≡ popularidad(a) + popularidad(b) + 1
215
216   turnos(iniciar(m)) ≡ 0
217   turnos(avanzarTurno(s, dc)) ≡ turnos(s) + 1
218   turnos(unir(a, b)) ≡ maximo(turnos(a), turnos(b))
219
220   construcciones(s) ≡ auxUnirDiccionarios(casas(s), comercios(s))
221   auxUnirDiccionarios(da, db) ≡
222     if vacío?(claves(db)) then
223       da
224     else
225       if def?(dameUno(claves(db)), da) then
226         auxUnirDiccionarios(da, borrar(dameUno(claves(db)), db))
227       else
228         definir(dameUno(claves(db)), obtener(dameUno(claves(db))),,
229           auxUnirDiccionarios(da, borrar(dameUno(claves(db)), db)))
230       fi
231     fi
232
233   restriccionesAvanzarTurno(s, cs) ≡ nuevasNoSuperponenRios(s, claves(cs)) AL
234   • vacío?(claves(construcciones(s)) ∩ claves(cs)) AL
235   sonConstrucciones(cs)
236   restriccionesUnir(a, b) ≡ noSuperponenRios(a, b) AL noSuperponenRios(b, a) AL
237   • noSuperponenConstMax(a, b) AL noSuperponenConstMax(b, a)
238
239   sonConstrucciones(cs) ≡ vacío?(cs) VL ((esCasa?(dameUno(cs)) VL
240   • esComercio?(dameUno(cs))) ∧ sonConstrucciones(sinUno(cs)))
241
242   noSuperponenRios(a, b) ≡ ¬auxSuperponenRios(mapa(a), claves(construcciones(b)))
243   nuevasNoSuperponenRios(s, c) ≡ ¬auxSuperponenRios(mapa(s), c)
244   auxSuperponenRios(m, c) ≡ ¬vacío?(c) AL (seSuperpone(m, dameUno(c)) V
245   • auxSuperponenRios(m, sinUno(c)))//c es conj(Pos)
246   seSuperpone(m, p) ≡ p1 ∈ horizontales(m) V p2 ∈ verticales(m)
247
248   noSuperponenConstMax(a, b) ≡ auxnoSuperponenPos(claves(construcciones(a)),
249   • auxMaxConst(claves(construcciones(b)), construcciones(b)))
250   auxMaxConst(c, dc) ≡
251     if vacío?(c) then
252       ∅
253     else
254       if obtener(dameUno(c), dc) = auxMaxNivel(claves(dc), dc) then
255         Ag(dameUno(c), auxMaxConst(sinUno(c), dc))
256       else
257         auxMaxConst(sinUno(c), dc)
258       fi
259     fi
260   auxMaxNivel(c, dc) ≡
261     if vacío?(c) then
262       0

```

```

257     else
258         maximo(obtener(dameUno(c), dc), auxMaxNivel(sinUno(c), dc))
259     fi
260     auxnoSuperponenPos(ca, cb) ≡ vacío?(ca ∩ cb)
261
262 Fin TAD
263
264 TAD SERVIDOR
265 Igualdad Observacional: ( $\forall s, s': \text{servidor}$ ) ( $s =_{\text{obs}} s' \iff$ 
266  $((\forall n: \text{nombre}) (\text{haySimcity?}(s, n) =_{\text{obs}} \text{haySimcity}(s', n) \wedge \text{haySimcity?}(s, n) \Rightarrow_L$ 
267  $\text{darSimcity}(s, n) =_{\text{obs}} \text{darSimcity}(s', n) \wedge$ 
268  $\text{esUnible}(s, n) =_{\text{obs}} \text{esUnible}(s', n) \wedge$ 
269  $\text{darConstrucciones}(s, n) =_{\text{obs}} \text{darConstrucciones}(s', n))));$ 
270
271 Géneros: nombreTad
272 Exporta: nombreTad, observadores, generadores,
273 Usa: Bool, Nat
274
275 Observadores Básicos:
276     haySimcity? : servidor × nombre → bool
277     darSimcity : servidor srv × nombre n → simcity
278         {haySimcity?(srv, n)}
279     esUnible? : servidor srv × nombre n → bool
280         {haySimcity?(srv, n)}
281     darConstrucciones : servidor srv × nombre n → dicc(pos × construcion) cs
282         {haySimcity?(srv, n)}
283
284 Generadores:
285     levantarServidor : → servidor
286
287     fundarSimcity : servidor srv × nombre n × mapa → servidor
288         {-haySimcity?(srv, n)}
289     avanzarTurnoPorServidor : servidor srv × nombre n × dicc(pos × construcion) cs →
290     • servidor
291
292         {haySimcity?(srv, n) ∧L esUnible?(srv, n) ∧L
293         • restriccionesAvanzarTurno(darSimcity(srv, n), cs)}
294
295     unirPorServidor : servidor srv × nombre n × nombre m → servidor
296         {¬(n = m) ∧ (haySimcity?(srv, n) ∧ haySimcity?(srv, m)) ∧L (esUnible?(srv, n) ∧
297         • esUnible?(srv, m)) ∧L restriccionesUnir(darSimcity(srv, n), darSimcity(srv, m))
298         • ∧ vacio?(darConstrucciones(srv, n))}
299
300
301     agregarCasa : servidor srv × nombre n × pos p → servidor
302         {haySimcity?(srv, n) ∧L esUnible?(srv, n) ∧
303         • nuevasNoSuperponenRios(darSimcity(srv, n), Ag(p, ∅)) ∧ ¬(p ∈
304         • claves(construcciones(darSimcity(srv, n)))) ∧ ¬(p ∈
305         • claves(darConstrucciones(srv, n))) }
```

```

-- .
305 agregarComercio : servidor srv x nombre n x pos p -> servidor
306   {haySimcity?(srv, n) AL esUnible?(srv, n) ∧
307   • nuevasNoSuperponenRios(darSimcity(srv, n), Ag(p, ∅)) ∧ ¬(p ∈
308   • claves(construcciones(darSimcity(srv, n)))) ∧ ¬(p ∈
309   • claves(darConstrucciones(srv, n))) }

310
311
312
313
314
315
316
317
318
319
320
321

Otras Operaciones:
311 turnosPorServidor : servidor srv x nombre n -> nat
312   {haySimcity?(srv, n)}
313 mapaPorServidor: servidor srv x nombre n -> mapa
314   {haySimcity?(srv, n)}
315 casasPorServidor: servidor srv x nombre n -> dicc(pos, nivel)
316   {haySimcity?(srv, n)}
317 comerciosPorServidor: servidor srv x nombre n -> dicc(pos, nivel)
318   {haySimcity?(srv, n)}
319 popularidadPorServidor: servidor srv x nombre n -> nat
320   {haySimcity?(srv, n)}

Axiomas:
323 haySimcity?(levantarServidor(), n) ≡ false
324 haySimcity?(fundarSimcity(srv, n), m) ≡ (n = m) ∨ haySimcity?(srv, m)
325 haySimcity?(avanzarTurnoPorServidor(srv, n, cs), m) ≡ haySimcity?(srv, m)
326 haySimcity?(unirPorServidor(srv, n, m), n') ≡ haySimcity?(srv, n')
327 haySimcity(agregarCasa(srv, n, p), m) ≡ haySimcity?(srv, m)
328 haySimcity(agregarComercio(srv, n, p), m) ≡ haySimcity?(srv, m)

329
330 darSimcity(fundarSimcity(srv, n, map), m) ≡
331   if n = m then
332     iniciar(map)
333   else
334     darSimcity(srv, m)
335   fi
336 darSimcity(avanzarTurnoPorServidor(srv, n, cs), m) ≡
337   if n = m then
338     avanzarTurno(darSimcity(srv, n), cs)
339   else
340     darSimcity(srv, m)
341   fi
342 darSimcity(unirPorServidor(srv, n, m), l) ≡
343   if n = l then
344     unir(darSimcity(srv, n), darSimcity(srv, m))
345   else
346     darSimcity(srv, l)
347   fi
348 darSimcity(agregarCasa(srv, n, p), m) ≡ darSimcity(srv, m)
349 darSimcity(agregarComercio(srv, n, p), m) ≡ darSimcity(srv, m)

350
351 esUnible?(fundarSimcity(srv, n, map), m) ≡ (n = m) ∨ esUnible?(srv, m)
352 esUnible?(avanzarTurnoPorServidor(srv, n, cs), m) ≡ esUnible?(srv, m)
353 esUnible?(unirPorServidor(srv, n, m), l) ≡ ¬(m = l) AL esUnible?(srv, l)
354 esUnible?(agregarCasa(srv, n, p), m) ≡ esUnible?(srv, m)
355 esUnible?(agregarComercio(srv, n, p), m) ≡ esUnible?(srv, m)

```

```

356
357     darConstrucciones(fundarSimcity(srv, n, m), l) ≡
358         if n = 1 then
359             ∅
360         else
361             darConstrucciones(srv, l)
362         fi
363     darConstrucciones(avanzarTurnoPorServidor(srv, n, cs), m) ≡
364         if n = m then
365             vacío
366         else
367             darConstrucciones(srv, m)
368         fi
369     darConstrucciones(unirPorServidor(srv, n, m), l) ≡ darConstrucciones(srv, l)
370     darConstrucciones(agregarCasa(srv, n, p), m) ≡
371         if n = m then
372             definir(p, "casa", darConstrucciones(srv, m))
373         else
374             darConstrucciones(srv, m)
375         fi
376     darConstrucciones(agregarComercio(srv, n, p), m) ≡
377         if n = m then
378             definir(p, "comercio", darConstrucciones(srv, m))
379         else
380             darConstrucciones(srv, m)
381         fi
382
383
384     turnosPorServidor(srv, n) ≡ turnos(darSimcity(srv, n))
385     mapaPorServidor(srv, n) ≡ mapa(darSimcity(srv, n))
386     casasPorServidor(srv, n) ≡ casas(darSimcity(srv, n))
387
388
389     comerciosPorServidor(srv, n) ≡ comercios(darSimcity(srv, n))
390     popularidadPorServidor(srv, n) ≡ popularidad(darSimcity(srv, n))
391
392 Fin TAD
393
394
395
396 -----
397 uso trie con clave nombre y significado simcity
398 agregar casa y comercios los guardamos en una "caja" en servidor
399 NO tenemos que modificar los niveles de un simcity que ya unimos
400 -----
401
402 Modulos de referencia:
403
404 Modulo Mapa:
405
406 Interfaz:
407     se explica con: Mapa
408     generos: mapa
409
410

```

```

410
411 Operaciones basicas de mapa:
412
413 CREAR(in hs: conj(Nat), in vs: conj(Nat))-> res:mapa
414 Pre ≡ {true}
415 Post ≡ {res =obs mapa(hs, vs)}
416 Complejidad: O(copy(hs), copy(vs))
417 Descripcion: crea un mapa
418
419 UNIR(in m1:mapa, in m2:mapa)-> res:mapa
420 Pre ≡ {true}
421 Post ≡ {res =obs mapa(horizontales(m1) U horizontales(m2), verticales(m1) U
422 • verticales(m2))}
422 Complejidad: O(#(m1.horizontales) + #(m2.verticales) + copy(hs) + copy(vs))
423 Descripcion: crea un mapa
424
425
426 Representacion:
427
428 Representacion de mapa
429
430 mapa se representa con estr
431 donde estr es tupla(horizontales: conj(Nat), verticales: conj(Nat))
432
433 Rep: estr -> bool
434 Rep(c) ≡ true ⇔ true
435
436 Abs: estr m -> mapa
437 Abs(m) ≡ horizontales(m) = estr.horizontales ∧ estr.verticales(m) =
438 • estr.verticales
439 Algoritmos:
440 -----
441 icrear(in hs:conj(Nat), in vs:conj(Nat)) -> res:estr
442 1: estr.horizontales <- hs
443 2: estr.verticales <- vs return estr
444
445 Complejidad: O(copy(hs) + copy(vs))
446 -----
447 iUnir(in m1:mapa, in m2:mapa) -> res:mapa
448 1: res <- m1
449 2: agregarAmapa(m1, m2)
450 3: res <- icrear(m1.horizontales, m1.verticales)
451
452 Complejidad: O(#(m1.horizontales) + #(m2.verticales) + copy(hs) + copy(vs))
453 Aliasing: no hay Aliasing por que cada mapa se usa para un solo simcity
454
455 agregarAmapa(m1, m2){
456     it = creariT(m2.horizontales)
457     it2 = creariT(m2.verticales)
458
459     while(siguiente(it) != NULL ){ //supongo que el ultimo elem del it apunta a NULL
460         avanzar(it)
461         agregarRapido(m1.horizontales, *it)

```

```

463     }
464
465     while(siguiente(it) != NULL ){
466         avanzar(it)
467         agregarRapido(m1.horizontales, *it) //O(1)
468     }
469     //estamos modificando m1
470 }
471 complejidad: O(#(m1.horizontales) + #(m2.verticales))
472 -----
473
474 Modulo Servidor:
475
476 Interfaz:
477     se explica con: Servidor
478     generos: servidor
479
480 Operaciones basicas de servidor:
481
482 LEVANTAR()-> res:servidor
483 Pre ≡ {true}
484 Post ≡ {res =obs levantarServidor()}
485 Complejidad: O(1)
486 Descripcion: crea un servidor
487
488
489 FUNDAR(in/out srv:servidor, in n:string, in m:mapa)
490 Pre ≡ {srv =obs srv0 ∧ \not haySimcity(srv0, n)}
491 Post ≡ {srv =obs fundarSimcity(srv0, n, m)}
492 Complejidad: O(|n|)
493 Descripcion: crea una nueva simcity
494
495 AGCASAxSERVIDOR(in/out srv:servidor, in n:string, p:posicion)
496 Pre ≡ {srv =obs srv0 ∧L haySimcity?(srv0, n) ∧L esUnible?(srv0, n) ∧L
497     restriccionesAvanzarTurno(darSimcity(srv0, n), cs)}
498 Post ≡ {srv =obs guardaCasaEnCajita(srv0, n, s)}
499 Complejidad: O(|n|)
500 Descripcion: agrega una casa(la guarda hasta avanzar turno)
501
502 AGCOMERCIOxSERVIDOR(in/out srv:servidor, in n:string, p:posicion)
503 Pre ≡ {srv =obs srv0 ∧L haySimcity?(srv0, n) ∧L esUnible?(srv0, n) ∧L
504     restriccionesAvanzarTurno(darSimcity(srv0, n), cs)}
505 Post ≡ {srv =obs guardaComercioEnCajita(srv0, n, s)}
506 Complejidad: O(|n|)
507 Descripcion: agrega un comercio(la guarda hasta avanzar turno)
508
509 TURNOxSERVIDOR(in/out srv:servidor, in n:string)
510 Pre ≡ {srv =obs srv0 ∧L haySimcity?(srv0, n) ∧L esUnible?(srv0, n) ∧L
511     restriccionesAvanzarTurno(darSimcity(srv0, n), cs)}//agregar cajita a tad servidor
512 Post ≡ {srv =obs AvanzarTurno(srv0, n, cajita)}
513 Complejidad: O(cantcasasTotales + cant comerciosTotales long(icomercios(e)))
514 Descripcion: avanza turno
515
516

```

```

51/
518 UNIR(in/out srv:servidor, in n:string, in m: string)
519 Pre ≡ {srv =obs srv0 ∧ ¬(n =obs m) ∧ (haySimcity?(srv, n) ∧ haySimcity?(srv, m)) ∧L
• (esUnible?(srv, n) ∧L
520 esUnible?(srv, m)) ∧L restriccionesUnir(darSimcity(srv, n), darSimcity(srv, m)) y
• cajita vacia}
521 Post ≡ {srv =obs unirPorServidor(srv, n, m)}
522 Complejidad: O(|n|+|m|)
523 Descripcion: une dos simcities
524
525 HAYSIMCITY?(in srv:servidor, in n:string)-> res:bool
526 Pre ≡ {true}
527 Post ≡ {res =obs haySimcity?(srv, n)}
528 Complejidad: O(|n|)
529 Descripcion: responde si existe un simcity con ese nombre
530
531 DARSIMCITY(in srv:servidor, in n:string)-> res: simcity
532
533 Pre ≡ {haySimcity?(srv, n)}
534 Post ≡ {res =obs darSimcity(srv, n)}
535 Complejidad: O(|n|)
536 Descripcion: busca y devuelve el simcity por nombre
537
538 ESUNIBLE?(in/out srv:servidor, in n:string)-> res:bool
539 Pre ≡ {haySimcity?(srv, n)}
540 Post ≡ {res =obs esUnible(srv, n)}
541 Complejidad: O(|n|)
542 Descripcion: responde si el simcity en cuestion se puede modificar
543
544 //LANZO LAS OPERACIONES DESDE EL SERVIDOR
545 //uso operaciones del interfaz del simcity (MAPA, CASAS, etc)
546 VERMAPA(in srv:servidor, in n:string) <- res: mapa
547 Pre ≡ {haySimcity?(srv, n)}
548 Post ≡ {res= MAPA(darsimcity(srv, n))}
549 Complejidad: O(|n|)
550 Descripcion: devuelve el mapa de un simcity
551
552 VERCASAS(in srv:servidor, in n:string) <- res: dicc(pos, nivel)
553 Pre ≡ {haySimcity?(srv, n)}
554 Post ≡ {res = CASAS(darSimcity(srv, n))}
555 Complejidad: O(|n|) + O(iCasas)
556 Descripcion: devuelve el casas de un simcity
557
558 VERCOMERCIOS(in srv:servidor, in n:string) <- res: dicc(pos, nivel)
559 Pre ≡ {haySimcity?(srv, n)}
560 Post ≡ {res = COMERCIOS(darSimcity(srv, n))}
561 Complejidad: O(|n|) + O(iComercios)
562 Descripcion: devuelve el comercios de un simcity
563
564 VERTURNO(in srv:servidor, in n:string) <- res: Nat
565 Pre ≡ {haySimcity?(srv, n)}
566 Post ≡ {res = TURNO(darSimcity(srv, n)) }
567 Complejidad: O(|n| + 1)
568 Descripcion: devuelve el turno de un simcity
569

```

```

570 VERPOPULARIDAD(in srv:servidor, in n:string) <- res: Nat
571 Pre ≡ {haySimcity?(srv, n)}
572 Post ≡ {res = POPULARIDAD(darSimcity(srv, n))}
573 Complejidad: O(|n| + 1)
574 Descripcion: devuelve el popularidad de un simcity
575
576 Representacion:
577
578 Representacion de servidor
579
580 Utilizamos un DictTrie para poder acceder a las operaciones pedidas con la complejidad
• requerida.
581 Recorrer el Trie lleva O(nombre) ya que sabemos que cada nodo tiene como maximo unos
• 28(suponiendo que
582 usamos el alfabeto en castellano) hijos y por lo tanto el unico numero no acotado es
• el largo del nombre.
583
584 servidor se representa con estr
585 donde estr es un dictTrie(nombre: string, tupla(s:simcity, modifiable:bool,
• constr:dict(pos, constr)))
586 //modifiable: aun no lo uni y puedo unirlo
587 //no modifiable: ya lo uni
588
589 Rep: estr -> bool
590 (forall e: estr)
591 Rep(e) ≡ (forall n:nombre)(n ∈ claves(e) =>L ( ~obtener(n, e).modifiable) =>
• estaVacia?(obtener(n, e).constr)
592 ∧L largosIguales(obtener(n, e).s) ∧L existenUniones(obtener(n, e).s, e))
593
594 largosIguales: simcity -> bool
595 largosIguales(s) ≡ largo(π2(*e.casas)) = largo(π2(*e.comercios)) ∧
• largo(π2(*e.casas)) = largo(π2(*e.mapas))
596
597 existenUniones: simcity s -> bool {largosIguales(s)}
598 existenUniones(s, e) ≡
599 (forall puntero: π2(*e.casas))((exists n:nombre)(n ∈ claves(e) ∧L (obtener(n, e).s.casas) =
• puntero ∧L
600 (exists p1: π2(*e.comercios))(obtener(n, e).s.comercios = p1) ∧L
601 (exists p2: π2(*e.mapas))(obtener(n, e).s.mapas = p2))
602
603 //el mapa es infinito y son nat
604
605 Abs: estr srv -> servidor {Rep(srv)}
606 Abs(srv) ≡ s:servidor/
607 (forall n:nombre)((n ∈ claves(srv) <=> haySimcity?(s, n) ) ∧L (haySimcity?(s, n) =>L (
• (π1(obtener(n, srv)) =obs darSimcity(s, n)) ∧L
608 (π2(obtener(n, srv)) =obs esUnible?(s, n)) ∧L ((π3(obtener(n, srv)) =obs
• darConstrucciones(s, n)))) ))
609
610
611 Algoritmos:
612 -----
613 iLevantar() -> res:estr
614 1: estr <- vacio()
615

```

```

615
616 Complejidad: O(1)
617 -----
618 iFundar(in/out srv:servidor, in n:string, in m:mapa)
619 1: definir(n, tupla(iniciar(m), true, vacio()), srv)
620
621 Complejidad: O(|n| + 1) = O(|n|)
622 -----
623 iAgCasaxServidor(in/out srv:servidor, in n:string, p:posicion)
624
625 1: definir(p, "Casa", obtener(n, srv).constr) //guardamos en una cajita Las
• construcciones
626
627 //obtener(n,srv) me una tupla {simcity, un bool, constr} donde constr es la "cajita"
628 Complejidad: O(|n| + 1) = O(|n|)
629 -----
630 iAgComercioxServidor(in/out srv:servidor, in n:string, p:posicion)
631 1: definir(p, "Comercio", obtener(n, srv).constr)
632
633 Complejidad: O(|n| + 1) = O(|n|)
634 -----
635 iTurnoPorServidor(in/out srv:servidor, in n:string)
636 1: iAvanzarTurno(iDarSimcity(srv, n), obtener(n, srv).constr)
637 //usa avanzarTurno del simcty y le paso un simcity y la "cajita" de construcciones
638
639 Complejidad: O(|n| + complejidad(iAvanzarTurno))
640 -----
641 iUnir(in/out srv:servidor, in n:string, in m: string)
642 1: iUnirSymcities(iDarSimcity(srv, n), iDarSimcity(srv, m))
643
644 Complejidad: O(|n| + |n|) = O(|Nombre|)
645 -----
646 iHaySimcity(in srv:servidor, in n:string) -> res: bool
647 1: res <- definido?(n, srv)
648
649
650 Complejidad: O(|n|)
651 -----
652 iDarSimcity(in srv:servidor, in n:string) -> res:simcity
653 1: res <- significado(n, srv).simcity
654
655 Complejidad: O(|n| + copy(simcity))
656 -----
657 iEsUnible(in/out srv:servidor, in n:string) -> res: bool
658 1: res <- significado(n, srv).modifiable
659
660 Complejidad: O(|n|)
661 -----
662
663 iverMapa(in srv:servidor, in n:string) <- res: mapa
664 1: res = iMapa(iDarSimcity(srv, n))
665
666 -----
667 iverCasas(in srv:servidor, in n:string) <- res: dicc(pos, nivel)
668 1: res = iCasas(iDarSimcity(srv, n))

```

```

669 -----
670 iverComercios(in srv:servidor, in n:string) <- res: dicc(pos, nivel)
671 1: res = iComercios(iDarSimcity(srv, n))
672 -----
673 iverTurno(in srv:servidor, in n:string) <- res: Nat
674 1: res = iTurno(iDarSimcity(srv,n))
675 -----
676 iverPopularidad(in srv:servidor, in n:string) <- res: Nat
677 1: res = iPopularidad(iDarSimcity(srv,n))
678 -----
679
680 Fin de Algoritmos;
681
682
683 Modulo SimCity:
684
685 Interfaz:
686 se explica con: SimCity
687 generos: simcity
688
689 Operaciones basicas de servidor:
690
691 INICIAR(in m: mapa) -> res: simcity
692 Pre ≡ {true}
693 Post ≡ {res =obs iniciar(m)}
694 Complejidad: O(1)
695 Descripcion: inicializo el SimCity
696 Aliasing:
697
698 AVANZARTURNO(in/out s: simcity, cs: dicc(pos, construccion))
699 Pre ≡ {s=s0 ∧ restriccionesAvanzarTurno(s, cs)}
700 Post ≡ {s =obs avanzarTurno(s,cs)}
701 Complejidad: O(iCasas + iComercios + cant construcciones totales + cant
    • construcciones totales * |cs|)
702 Descripcion: el SimCity avanza un turno
703 Aliasing:
704
705 UNIR(in/out s1: simcity, in s2: SimCity)
706 Pre ≡ {s1=s10 ∧ restriccionesUnir(s1, s2)}
707 Post ≡ {s1 =obs unir(s1, s2)}
708 Complejidad: O(1)
709 Descripcion: union de dos simcities
710 Aliasing:
711
712 MAPA(in s:simcity) -> res: mapa
713 Pre ≡ {true}
714 Post ≡ {res =obs mapa(s)}
715 Complejidad: O(iunir*popularidad)
716 Descripcion: me da el mapa del simcity
717 Aliasing:
718
719 CASAS(in s: simcity) -> res: dicc(pos, Nivel)
720 Pre ≡ {true}
721 Post ≡ {res =obs casas(s)}

```

```

722 Complejidad: O(popularidad * long(n)), donde n es la cantidad maxima de claves de
    • todos los diccionarios de casas guardados en el arbol
723 Descripcion: devuelve todas las casas que hay en SimCity
724 Aliasing:
725
726 COMERCIOS(in s: simcity) -> res: dicc(pos, Nivel)
727 Pre ≡ {true}
728 Post ≡ {res =obs comercios(s)}
729 Complejidad: O(popularidad * long(n)), donde n es la cantidad maxima de claves de
    • todos los diccionarios de comercios guardados en el arbol
730 Descripcion:
731 Aliasing:
732
733 TURNO(in s:simcity) -> res: Nat
734 Pre ≡ {true}
735 Post ≡ {res =obs turno(s)}
736 Complejidad: O(1)
737
738 POPULARIDAD(in s: simcity) -> res: Nat
739 Pre ≡ {true}
740 Post ≡ {res =obs popularidad(s)}
741 Complejidad: O(1)
742
743 Fin de la Interfaz;
744
745 Representacion:
746
747
748 Representacion de simcity
749
750 Necesitamos que ciertas operaciones lanzadas desde el servidor ocurran en
    • O(|Nombre|),
751 donde Nombre es el nombre más largo de todos los simcities. Como el nombre solo nos
    • interesa para buscar un simcity,
752 efectivamente tenemos ciertas operaciones que deben ocurrir en O(1). Por eso,
    • elegimos la estructura de tal forma de que
753 todas las operaciones costosas se puedan hacer cuando los requisitos nos lo permiten.
    • Por eso, elegimos una tupla donde
754 casas, comercios y mapas se guardan como arboles, para poder unirlos rápidamente, y
    • que hay una relación natural de
755 simcity unión y simcity unido, y guardamos turnos(antiguedad) y popularidad como
    • valores precalculados.
756 Decidimos que luego de una unión, el simcity unido no es modificable. Para garantizar
    • eso, en casas y comercios solo
757 se modifica la raíz del árbol cambiando su valor o agregando uniones a la lista de
    • punteros. Si una unión cambia el
758 valor de algo (por ejemplo, un comercio por acercarlo a una casa) decidimos recorrer
    • el árbol apropiado, conseguir la
759
760
761 respuesta, y guardarla en la raiz. De la misma forma, avanzar turno implica recorrer
    • los árboles, conseguir el estado
762 actual, guardarlo en el primer nodo, y luego aplicarle los cambios.
763
764

```

```

765  Podríamos haber puesto un valor que marque si ya conseguimos el estado actual para no
    • hacerlo en cada operacion costosa,
766  y eso nos haría los algoritmos mucho mas eficientes(en promedio, no en peor caso)pero
    • la iRep de servidor se me hace nefasta.
767
768  simcity se representa con estr
769  donde estr es tupla<casas:arbolconst, comercios:arbolconst, mapa:arbolmapa,
    • turnos:nat, popularidad:nat>
770  donde arbolconst es puntero(nodo(dicc(Pos, nivel)))
771  donde arbolmapa es puntero(nodo(mapa))
772
773  donde nodo es tupla< $\alpha$ , lista(punteros(nodo( $\alpha$ )))>
774
775  //tamano= cantidad de uniones +1
776  Rep: estr -> bool
777  ( $\forall$  e: estr)
778  Rep(e)  $\equiv$ 
779  alturaMenorIgual(e.casas, 0, popularidad + 1)  $\wedge$ L alturaMenorIgual(e.comercios, 0,
    • popularidad + 1)  $\wedge$ L
780  alturaMenorIgual(e.mapas, 0, popularidad + 1)  $\wedge$ L popularidad + 1 =
    • tamano(e.comercios)  $\wedge$ L popularidad + 1 = tamano(mapa)
781   $\wedge$ L popularidad + 1 = tamano(e.casas)  $\wedge$ L  $\neg$ auxSuperponenRios(armarMapa(e.mapa),
    • listaTotalCasas(e.casas))  $\wedge$ L
782   $\neg$ auxSuperponenRios(borrarClaves(claves(listaTotalCasas(e.casas))),
    • listaTotalComercios(e.comercios)))  $\wedge$ L
783  e.turnos = maximo(maxnivel(e.casas), maxnivel(e.comercios))
784
785
786  alturaMenorIgual: puntero(nodo( $\alpha$ ))  $\times$  nat  $\times$  nat -> bool
787  alturaMenorIgual(arb, n, m)  $\equiv$ 
788  if n > m then
789      false
790  else
791      alturaHijosMenorIgual( $\pi_2(*arb)$ , n, m)
792  fi
793
794  alturaHijosMenorIgual: lista(puntero(nodo( $\alpha$ )))  $\times$  nat  $\times$  nat -> bool
795  alturaHijosMenorIgual(ls, n, m)  $\equiv$ 
796  if largo(ls) = 0 then
797      true
798  else
799      alturaMenorIgual(prim(ls), n+1, m) + sumanodos(fin(ls), n, m)
800  fi
801
802  tamano: puntero(nodo( $\alpha$ )) -> nat
803  tamano(arb)  $\equiv$ 
804  if arb = null then
805      0
806  else
807      1 + sumanodos(*arb2)
808  fi
809
810  sumanodos: lista(puntero(nodo( $\alpha$ ))) -> Nat
811  sumanodos(ls)  $\equiv$ 

```

```

812 if largo(ls) = 0 then
813   0
814 else
815   tamano(prim(ls)) + sumanodos(fin(ls))
816 fi
817
818 listaTotalCasas: arbolconst -> dicc(pos, nivel)
819 listaTotalCasas(arb) =
820   if arb = null then
821     vacio
822   else
823     darUnion(listaTotalCasasNodos(pi2(*arb)), pi1(*arb), claves(pi1(*arb)))
824 fi
825
826 listaTotalCasasNodos: lista(arbolconst) -> dicc(pos, nivel)
827 listaTotalCasasNodos(ls) =
828   if largo(ls) = 0 then
829     vacio
830   else
831     darUnion(listaTotalCasasNodos(fin(ls)), listaTotalCasas(prim(ls)),
832       • claves(listaTotalCasas(prim(ls))))
833 fi
834
835 listaTotalComercios: arbolconst -> dicc(pos, nivel)
836 listaTotalComercios(arb) =
837   if arb = null then
838     vacio
839   else
840     darUnion(listaTotalComerciosNodos(pi2(*arb)), pi1(*arb), claves(pi1(*arb)))
841 fi
842
843 listaTotalComerciosNodos: lista(arbolconst) -> dicc(pos, nivel)
844 listaTotalComerciosNodos(ls) =
845   if largo(ls) = 0 then
846     vacio
847   else
848     darUnion(listaTotalComerciosNodos(fin(ls)), listaTotalComercios(prim(ls)),
849       • claves(listaTotalComercios(prim(ls))))
850 fi
851
852 maxnivel: arbolconst -> nat
853 maxnivel(arb) =
854   if arb = null then
855     0
856   else
857     maximo(maxconst(*arb1), maxnodos(*arb2))
858 fi
859
860 maxnodos: lista(punteros(nodo)) -> Nat
861 maxnodos(ls) =
862   if largo(ls) = 0 then
863     0
864

```

```

865      0
866 else
867   maximo(maxnivel(prim(ls)), maxnodos(fin(ls)))
868 fi
869
870 maxconst: dicc(Pos, Nivel) -> Nat
871 maxconst(d) ≡
872 if vacío?(claves(d)) then
873   0
874 else
875   maximo(obtener(dameUno(claves(d))), maxconst(borrar(dameUno(claves(d)), d)))
876 fi
877
878 armarmapa: arbolmapa -> mapa
879 armarmapa(arb) ≡
880   if arb = null then
881     crear(∅, ∅)
882   else
883     unir(π1(*arb), sumanodos(π2(*arb)))
884 fi
885
886 armarmapanodos: lista(arbolmapa) -> Nat
887 armarmapanodos(ls) ≡
888 if largo(ls) = 0 then
889   vacío()
890 else
891   unir(armarmapa(prim(ls)), armarmapanodos(fin(ls)))
892 fi
893
894
895
896
897 Abs: estr e -> servidor           {Rep(e)}
898 Abs(e) ≡ s: simcity/
899 #(claves(π1(*(e.casas))) = #(claves(π1(*(e.casas))) ∧ sonIguales(π1(*(e.casas)),
900   • casas(s), 0, #(claves(π1(*(e.casas))))) ∧
901   sonIguales(π1(*(e.comercios)), comercios(s), 0, #(claves(π1(*(e.comercios)))) ∧
902   e.popularidad = popularidad(s) ∧ π1(*e.mapa) = mapa(s) ∧ e.turnos = turnos(s)
903
904 sonIguales(d, dcs, i, long) ≡
905   if i >= long then
906     true
907   else
908     if esAlgunoDelDicc(dameUno(claves(d)), obtener((dameUno(claves(d))), dcs) then
909       sonIguales(1, dcs, i++, long )
910     else
911       false
912     fi
913   fi
914
915 esAlgunoDelDicc(c, o, dcs) ≡
916   if vacía?(dcs) then
917     false
918   else

```

```

918     if (c = dameUno(claves(dcs))) ∧ o = obtener(dameUno(claves(dcs)),dcs)) then
919         true
920     else
921         esAlgunoDelDicc(t, borrar(dameUno(claves(dcs)), dcs))
922     fi
923   fi
924 e.casas =obs casas(s)
925
926 fin de representacion;
927
928
929 Algoritmos:
930 -----
931 iIniciar(in m: mapa) -> res:simcity
932 1: estr <- { vacia() x vacia() x vacia x 0 x 0 }
933
934 Complejidad: O(1)
935 -----
936 iAvanzarTurno(in/out e: estr, cs: dicc(pos, construccion))
937 //sumar uno a todas(casas y comercios), agregar construcciones, para todo comercio
• chequear nivel
938 //cs es la "cajita"
939 1: e.turnos++
940 2: π1(*e.casas) <- iCasas(e)
941 3: π1(*e.comercios) <- iComercios(e)
942 4: itdicc <- crearIt(siguiente(π1(*e.casas)))
943 5: subirUnNivelAlDicc(π1(*e.casas), itdicc)
944 6: itdicc <- crearIt(siguiente(π1(*e.comercios)))
945 7: subirUnNivelAlDicc(π1(*e.comercios), itdicc)
946 8: agregarConstrucciones(e, cs)
947 //iCasas me devuelve todas las casas, incluida las casas uniones
948
949 Complejidad: O(iCasas + iComercios + cant construcciones totales + cant construcciones
• totales * |cs| )
950
951 subirUnNivelAlDicc(d, itdicc){
952   while(haySiguiente(itdicc)){
953     //sobreescribe la clave que ya tengo
954     definir(siguienteClave(itdicc), siguienteSignificado(itdicc) + 1, d)
955     itdicc = avanzar(itdicc)
956   }
957 }
958 //agrego nuevas construcciones
959 agregarConstrucciones(e, cs){
960   //agregar casas directamente
961   itdicc = creariT(cs)
962
963   while(haySiguiente(itdicc)){
964     if(siguienteSignificado(itdicc) = "casa"){
965
966       definir(siguienteClave(itdicc), 1, π1(*e.casas))
967
968       //el primero de la lista de casas son las casas originales del simcity
969     }
970

```

```

971     else if(siguienteSignificado(itdicc) = "comercio"){
972         if(~estaAdistManhattan(iCasas(e), siguienteClave(itdicc)){ //aca solo quiero ver
973             las casas totales
974             definir(siguienteClave(itdicc), 1, π1(*e.comercios))
975         }
976         else{ //hay alguna casa a dist de manhattan del comercio que estoy agregando
977             definir(siguienteClave(itdicc), NivCasaCercaxManhattan(iCasas(e),
978             siguienteClave(itdicc)), π1(*e.comercios))
979         }
980     }
981 }
982 estaAdistManhattan(cs, posComercio): res <- bool {
983     res <- false
984     itcs = crearIt(cs)
985     while(haySiguiente(itcs)){
986         if(formulaManhattan(siguienteClave(itcs), posComercio){
987             //como agrego comercios nuevos, no me fijo el nivel mayor
988             res <- true
989         }
990         itcs = avanzar(itcs)
991     }
992 }
993 //decision: de todas las casas a dist de Manhattan tomamos la de mayor nivel
994 NivCasaCercaxManhattan(cs, posComercio)<- res: nat {
995     itcs = crearIt(cs)
996     nat maxNiv = 1
997     while(haySiguiente(itcs)){
998         if(formulaManhattan(siguienteClave(itcs), posComercio){
999             if( siguienteSignificado(itcs) > maxNiv){
1000                 maxNiv = siguienteSignificado(itcs)
1001             }
1002         }
1003         itcs = avanzar(itcs)
1004     }
1005     res <- maxNiv
1006 }
1007
1008 formulaManhattan(pos1, pos2)<- res:bool {
1009     res <- false
1010     if ( |π1(pos1)-π1(pos2)| + |π2(pos1)-π2(pos2)| ) <= 3 ){
1011         res <- true
1012     }
1013 }
1014
1015
1016 -----
1017 //no modiflico los niveles de el simcity union por que tengo que poder seguir mostrando
1018 • ese simcity sin ser modificado (modifico los niveles en iCasas e iComercios)
1019 iUnirSymcities(in/out e1: estr, in e2: estr)
1020 1: agregarAtras((e1.casas), &e2.casas)
1021 2: agregarAtras((e1.comercios), &e2.comercios)
1022 3: agregarAtras(π2(e1.mapas), &e2.mapas)

```

```

1022 4: e1.popularidad++
1023 5: e1.turnos <- maximo(e1.turno, e2.turno)
1024
1025 Complejidad: O(1 + 1 + 1 + 1 + 1) = O(1)
1026 -----
1027 iMapa(in e: estr)-> res:mapa
1028 1: res <- mapaTotal(e.mapa)
1029
1030     mapaTotal(mp){
1031         tmp <- π1(*mp)
1032         for(int i= 0; i< long((π2(*mp))); i++){
1033             tmp <- iUnir(res, mapaTotal(π2(*mp)[i]))
1034         }
1035         return tmp
1036     }
1037 Complejidad: O(iunir*popularidad)
1038
1039 -----
1040 iCasas(in e: estr) -> res: dicc(pos, Nivel)
1041 1: res <- π1(*e.casas)
1042 2: if (~vacía?(π2(*e.casas))) {
1043 3:     itlist = crearIt(π2(*e.casas))
1044 4:     while(haySiguiente(itlist)){
1045 5:         recorrerUniones(siguiente(itlist), res)
1046 6:         itlist = avanzar(itlist)
1047     }
1048 }
1049     recorrerUniones(nodo, res){
1050         itdicc = crearIt(π1(nodo))
1051         res = agregarCasasDeUnion(itdicc, res)
1052         if (~vacía?(π2(nodo))) {
1053             itlist = crearIt(π2(nodo))
1054             while(haySiguiente(itlist)){
1055                 res = recorrerUniones(siguiente(itlist), res)
1056                 itlist = avanzar(itlist)
1057             }
1058         }
1059 //Res son las casas antes de agregar el dicc que esta en el itdicc
1060     agregarCasasDeUnion(it, res){
1061         while(haySiguiente(it)){
1062             if(~definido?(siguienteClave(it,res))){
1063                 definir(siguienteClave(it), siguienteSignificado(it), res)
1064                 it = avanzar(it)
1065             }else{
1066                 if(siguienteSignificado(it) > obtener(siguienteClave(it), res)
1067                     definir(siguienteClave(it), siguienteSignificado(it), res)
1068             }
1069         }
1070     }
1071
1072
1073
1074 //Las uniones anteriores ya esta unidas , por lo cual me muestran sus respectivas casas
1075 //La idea es iterar el conj de claves de uno de los dicc de casas de la union y agregar
-----
```

```

1076
1077
1078 Complejidad: O(popularidad * long(n)),
1079 donde n es la cantidad maxima de claves de todos los diccionarios de casas guardados
• en el arbol
1080 -----
1081
1082 iComercios(in e: estr)
1083 1: res <- π1(*e.comercios)
1084 2: if (¬vacía?(π2(*e.comercios))) {
1085 3: itlist = crearIt(π2(*e.comercios))
1086 4: while(haySiguiente(itlist)){
1087 5:     recorrerUniones(siguiente(itlist), res)
1088 6:     itlist = avanzar(itlist)
1089 7: }
1090 8: }
1091 9: tmpCasas <- iCasas(e)
1092 10: itdicc <- crearIt(tmp)
1093 11: quitarComerciosSolapadosConCasas(itdicc, res)
1094 //subo nivel (ya estan todos los comercios unidos en res)
1095 12: itdiccComercios = crearIt(res)
1096 13: while(haySiguiente?(itdiccComercios)){
1097 14: if(estaAdistManhattan(tmpCasas, siguienteClave(itdiccComercios)){ //aca solo
• quiero ver las casas totales
1098 15: if(NivCasaCercaxManhattan(tmpCasas, siguienteClave(itdiccComercios)) >
• siguienteSignificado(itdiccComercios) {
1099 16:     definir(siguienteClave(itdiccComercios), NivCasaCercaxManhattan(tmpCasas,
• siguienteClave(itdiccComercios)), res)
1100 17: }
1101 18: }
1102 19: itcomercios = avanzar(itcomercios)
1103 20: }
1104     recorrerUniones(nodo, res){
1105         itdicc = crearIt(π1(nodo))
1106         res = agregarCasasDeUnion(itdicc, res)
1107         if (¬vacía?(π2(nodo))) {
1108             itlist = crearIt(π2(nodo))
1109             while(haySiguiente(itlist)){
1110                 res = recorrerUniones(siguiente(itlist), res)
1111                 itlist = avanzar(itlist)
1112             }
1113         }
1114
1115 //elimino comercio si estoy uniendo un simcity y hay una casa en la misma posicion
• del comercio
1116 quitarComerciosSolapadosConCasas(itdicc, res, e){ //le paso el it de diccionarios
1117     itCasas = crearIt(iCasas(e))
1118     while(HaySiguiente(itCasas)){ //hay un dict union
1119         if(definido?(siguienteClave(itCasas), res)){ //res es el dict de comercios
1120             borrar(siguienteClave(itCasas), res) // borro el comercio
1121         }
1122     }
1123 }
1124 //el res es dict Unidos tal que aun no esta el dict del it
1125 agregarComerciosDeUnion(it, res){
```

```

1126
1127     while(haySiguiente(it)){
1128         //caso de solapamento de comercios y quedarme con el mayor nivel
1129         if(definido?(siguienteClave(it), res) ){
1130             if(siguienteSignificado(it) > obtener(siguienteClave(it), res)) {
1131                 definir(siguienteClave(it), siguienteSignificado(it), res)
1132             }
1133             else{
1134                 definir(siguienteClave(it), siguienteSignificado(it), res)
1135             }
1136         }
1137         else{
1138             definir(siguienteClave(it), siguienteSignificado(it), res)
1139         }
1140         it = siguiente(it)
1141     }
1142 }
1143
1144 Complejidad: O(popularidad * long(n) * popularidad * long(m)), donde n y m son la
• cantidad maxima de claves de todos los
1145 diccionarios de casas y comercios guardados en los arboles respectivamente
1146 -----
1147 iPopularidad(in e: estr) -> res: Nat
1148 1: res <- e.popularidad
1149
1150 //cada vez que unimos aumentamos la popularidad
1151 Complejidad: O(1)
1152 -----
1153
1154 iTurno(in e: estr) -> res: Nat
1155 1: res <- e.turnos
1156
1157 Complejidad: O(1)
1158
1159 Fin de Algoritmos;
1160 |

```

