UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2019

Versión 4: 31 de mayo de 2019

TPI - "Transporte Urbano"

Entrega: 7 de Junio de 2019 (hasta las 16:30)

Cambios versión 4

1. *completarHuecos* corregida nuevamente :).

Cambios versión 3

1. *completarHuecos* corregida y reescrita. Revisar porque cambió la definición de hueco.
2. Corregidos varios predicados para que tanto las celdas como las grillas tengan esquinas esq_0 = esqInferiorIzquierda (al sudoeste) de esq_1 esqSuperiorDerecha (al noreste).

Cambios versión 2

1. *aux velocidad* estaba mal especificada.
2. corregidos tipos en *minTiempo* y *maxTiempo*
3. en *puntosOk*, debería llamarse a *limitesPorNombre(g, esq1, lado, lado)* en vez de *limitesPorNombre(g, esq1, lado)*
4. en *limnitesPorNombre*: *esEsqSupDer(grilla[i]$_0$)* y *esEsqSupDer(grilla[i]$_1$)* en vez de *esEsqSupDer(lat(grilla[i]))* y *esEsqSupDer(lng(grilla[i]))*

1. Antes de empezar

1. Bajar del campus de la materia Archivos RTPI.
2. Descomprimir el ZIP.
3. Dentro del ZIP van a encontrar una carpeta llamada *transporteUrbano*, cargarla como proyecto en CLion.

2. Ejercicios

1. Implementar las funciones especificadas en la sección **Especificación**. Utilizar los tests provistos por la cátedra para verificar sus soluciones (los cuales **No pueden modificar**). Respetar los tiempos de ejecución en el peor caso para las siguientes funciones:
   - *tiempoTotal*: $O(n)$ donde n representa la longitud del viaje.
   - *distanciaTotal*: $O(n^2)$ donde n representa la longitud del viaje.
   - *recorridoNoCubierto*: $O(n \times m)$ donde n representa la longitud del viaje y m la longitud del recorrido.
2. Escribir tests para la función descripta en la sección **Testing**.
3. Implementar las funciones descriptas en la sección **Entrada/Salida**.
4. (*Opcional*) Utilizar la interfaz gráfica provista para probar las funciones de Entrada/Salida y visualizar la creación de la grilla. Ver sección **Interfaz gráfica**.
5. Completar (agregando) los tests necesarios para cubrir todas las líneas del archivo *solucion.cpp*. Utilizar la herramienta **lcov** para dicha tarea. Ver sección **Análisis de cobertura**.

3. Especificación

type $Tiempo = \mathbb{R}$

type $Dist = \mathbb{R}$

type $GPS = \mathbb{R} \times \mathbb{R}$

type $Recorrido = seq\langle GPS \rangle$

type $Viaje = seq\langle Tiempo \times GPS \rangle$

type $Nombre = \mathbb{Z} \times \mathbb{Z}$

type $Grilla = seq\langle GPS \times GPS \times Nombre \rangle$

proc excesoDeVelocidad (in v: *Viaje*, out res: Bool) {

Pre $\{validoV(v)\}$

Post $\{res = \text{true} \leftrightarrow (\exists vOrd : Viaje)(esViajeOrdenadoPorTiempo(v, vOrd) \wedge superoVelocidad(vOrd))\}$

pred superoVelocidad (v: *Viaje*) {

$(\exists i : \mathbb{Z})(0 < i < |v| \wedge_L velocidad(v[i-1], v[i]) \geq 80)$

}

aux velocidad (p1: $Tiempo \times GPS$, p2: $Tiempo \times GPS$) : $\mathbb{R} = (dist(gps(p2), gps(p1))/1000)/((t(p2) - t(p1))/3600)$;

}

proc tiempoTotal (in v: *Viaje*, out t : *Tiempo*) {

Pre $\{validoV(v)\}$

Post $\{(\exists max : \mathbb{R})(\exists min : \mathbb{R})(maxTiempo(v, max) \wedge minTiempo(v, min) \wedge t = max - min\}$

}

proc distanciaTotal((in v: *Viaje*, out distancia : *Dist*) {

Pre $\{validoV(v)\}$

Post $\{(\exists vOrd : Viaje)(esViajeOrdenadoPorTiempo(v, vOrd) \wedge_L distanciaViaje(vOrd) = d)\}$

aux distanciaViaje (v: *Viaje*) : $Dist = \sum_{i=1}^{|v|-1} dist(v[i]_1, v[i-1]_1)$;

}

proc flota (in v: $seq\langle Viaje \rangle$, in t_0 : *Tiempo*, in t_f : *Tiempo*, out res : $\mathbb{Z}$) {

Pre $\{t_f > t_0 \wedge t_0 \geq 0 \wedge viajesValidos(v)\}$

Post $\{res = \sum_{i=0}^{|v|-1} \text{if } viajeEnFranjaHoraria(v[i], t_0, t_f) \text{ then } 1 \text{ else } 0 \text{ fi}\}$

pred viajeEnFranjaHoraria (v: *Viaje*, t_0: *Tiempo*, t_f: *Tiempo*) {

$\neg((\exists max : \mathbb{R})(\exists min : \mathbb{R})(maxTiempo(v, max) \wedge minTiempo(v, min) \wedge max < t_0 \vee min > t_f)$

}

}

proc recorridoNoCubierto (in v: *Viaje*, in r: *Recorrido*, in u : *Dist*, out res : $seq\langle GPS \rangle$) {

Pre $\{validoV(v) \wedge validoR(r) \wedge u{>}0\}$

Post $\{(|res| = cantNoCubiertos(v, r, u)) \wedge noCubiertosEnRes(v, r, u, res)\}$

aux cantNoCubiertos (v: *Viaje*, r: *Recorrido*, u: *Dist*) : $\mathbb{Z} = \sum_{i=0}^{|r|-1} \text{if } cubierto(v, u, r[i]) \text{ then } 0 \text{ else } 1 \text{ fi}$;

pred cubierto (v: *Viaje*, u: *Dist*, g: *GPS*) {

$(\exists x : GPS) estaEnViaje(x, v) \wedge_L dist(x, g){<}u$

}

pred noCubiertosEnRes (v: *Viaje*, r: *Recorrido*, u: *Dist*, res : $seq\langle GPS\rangle$) {

$(\forall g : GPS)((g \in r \wedge \neg cubierto(v, u, g)) \rightarrow g \in res)$

}

pred validoR (r: *Recorrido*) {

$(\forall i : \mathbb{Z})(0 \leq i < |r| \rightarrow_L gpsValido(r[i])$

}

}

proc construirGrilla (in esq1: *GPS*, in esq2: *GPS*, in n: $\mathbb{Z}$, in m: $\mathbb{Z}$, out g: *Grilla*) {

Pre $\{gpsValido(esq1) \wedge gpsValido(esq2) \wedge lat(esq1) < lat(esq2) \wedge lng(esq1) < lng(esq2)$
$\wedge n > 0 \wedge m > 0 \wedge permiteCeldasCuadradas(n, m, esq1, esq2)\}$

Post $\{(\exists gOrd : Grilla)|gOrd| = n \times m \wedge_L (nombresOk(gOrd, n, m) \wedge_L$
$puntosOk(gOrd, esq1, esq2, n, m)) \wedge mismaGrilla(g, gOrd)\}$

pred permiteCeldasCuadradas (n: $\mathbb{Z}$, m: $\mathbb{Z}$, esq1: *GPS*, esq2: *GPS*) {

$\frac{lat(esq2)-lat(esq1)}{n} = \frac{lng(esq2)-lng(esq1)}{m}$

}

pred mismaGrilla (g: *Grilla*, gOrd: *Grilla*) {

$|g| = |gOrd| \wedge (\forall x : Tiempo \times GPS)(cantAp(x, g) = cantAp(x, gOrd))$

}

}

proc aPalabra (in t: $seq\langle GPS\rangle$, in g: *Grilla*, out res: $seq\langle Nombre\rangle$) {

Pre $\{trayectoValido(t, g) \wedge grillaValida(grilla) \wedge trayectoEnGrilla(trayecto, grilla)\}$

Post $\{sonPalabras(res, t, g)\}$

pred trayectoValido (t: $seq\langle GPS\rangle$, g: *Grilla*) {

$(\forall g : GPS)(g \in t \rightarrow gpsValido(g))$

}

pred trayectoEnGrilla (trayecto: $seq\langle GPS\rangle$, g: *Grilla*) {

$(\forall t : GPS)(t \in trayecto \rightarrow hayCeldaParaCoord(t, g))$

}

pred hayCeldaParaCoord (t: *GPS*, g: *Grilla*) {

$(\exists celda : GPS \times GPS \times Nombre)(c \in g \wedge esCeldaDeCoordenada(t, celda))$

}

pred esCeldaDeCoordenada (p: *GPS*, celda: $GPS \times GPS \times Nombre$) {

$lat(celda_0) \leq lat(p) < lat(celda_1) \wedge lng(celda_0) \leq lng(p) < lng(celda_1)$

}

}

proc cantidadDeSaltos (in g: *Grilla*, in v: *Viaje*, out res : $\mathbb{Z}$) {

Pre $\{validoV(v) \wedge grillaValida(g) \wedge viajeEnGrilla(v, g)\}$

Post $\{(\exists trayecto : seq\langle GPS\rangle)(\exists nombres : seq\langle Nombre\rangle)(esTrayectoDeViajeOrdenado(trayecto, v) \wedge$
$sonPalabras(nombres, trayecto, g) \wedge res = cantidadSaltos(nombres))\}$

pred esTrayectoDeViaje (t: $seq\langle GPS\rangle$, v: *Viaje*) {

$|t| = |v| \wedge_L (\forall g : GPS)(g \in t \rightarrow estaEnViaje(g, v))$

}

pred esTrayectoDeViajeOrdenado (t: $seq\langle GPS\rangle$, v: $Viaje$) {

$|t| = |v| \wedge esViajeOrdenadoPorTiempo(v) \wedge_L (\forall i : \mathbb{Z})(0 \leq i < |t| \rightarrow_L t[i] = v[i]_1)$

}

aux cantidadSaltos (nombres:$seq\langle Nombre\rangle$) : $\mathbb{Z}$ = $\sum_{i=1}^{|celdas|}$ if $distanciaEntreCeldas(nombres[i-1], nombres[i]) > 1$ then 1 else 0 fi ;

aux distanciaEntreCeldas (n1:$Nombre$, n2:$Nombre$) : $\mathbb{Z} = |n1_0 - n2_0| + |n1_1 - n2_1| - 1$;

}

3.1. Predicados y funciones auxiliares

aux lat (p : GPS) : $\mathbb{R} = p_0$;

aux lng (p : GPS) : $\mathbb{R} = p_1$;

aux t (medicion : $Tiempo \times GPS$) : $Tiempo = medicion_0$;

aux gps (medicion : $Tiempo \times GPS$) : $GPS = medicion_1$;

pred validoV (v: $Viaje$) {

$|v| > 2 \wedge tiemposDistintos(v) \wedge (\forall i : \mathbb{Z})(0 \leq i < |v| \rightarrow_L (t(v[i]) \geq 0 \wedge gpsValido(gps(v[i])))$

}

pred tiemposDistintos (v: $Viaje$) {

$(\forall i : \mathbb{Z})(\forall j : \mathbb{Z})((0 \leq i < |v| \wedge 0 \leq j < |v| \wedge i \neq j) \rightarrow_L t(v[i]) \neq t(v[j])$

}

pred gpsValido (g: GPS) {

$-90 \leq lat(g) \leq 90 \wedge -180 \leq lng(g) \leq 180$

}

pred sonPalabras (ns: $seq\langle Nombre\rangle$, t: $seq\langle GPS\rangle$, g: $Grilla$) {

$|ns| = |t| \wedge_L (\forall i : \mathbb{Z})(0 \leq i < |ns| \rightarrow_L ((\exists j : \mathbb{Z})(0 \leq j < |g| \wedge_L ns[i] = g[j]_2 \wedge esCeldaDeCoordenada(t[i], g[j])))$

}

pred maxTiempo (v: $Viaje$, m: $\mathbb{R}$) {

$(\exists pos : \mathbb{Z})(0 \leq pos < |v| \wedge_L t(v[pos]) = m) \wedge (\forall i : \mathbb{Z})(0 \leq i < |v| \rightarrow_L m \geq t(v[i]))$

}

pred minTiempo (v: $Viaje$, m: $\mathbb{R}$) {

$(\exists pos : \mathbb{Z})(0 \leq pos < |v| \wedge_L t(v[pos]) = m) \wedge (\forall i : \mathbb{Z})(0 \leq i < |v| \rightarrow_L m \leq t(v[i]))$

}

pred esViajeOrdenadoPorTiempo (v: $Viaje$, vOrd: $Viaje$) {

$mismoViaje(v, vOrd) \wedge ordenadoPorTiempo(vOrd)$

}

pred mismoViaje (v: $Viaje$, vOrd: $Viaje$) {

$|v| = |vOrd| \wedge (\forall x : Tiempo \times GPS)(cantAp(x, v) = cantAp(x, vOrd))$

}

pred ordenadoPorTiempo (v: $Viaje$) {

$(\forall i : \mathbb{Z})1 \leq i < |v| \rightarrow_L t(v[i]) > t(v[i-1])$

}

`pred estaEnViaje` (g: *GPS*, v: *Viaje*) {

$(\exists i : \mathbb{Z})(0 \leq i < |v| \wedge_L gps(v[i]) = g$

}

`pred nombresOk` (g: *Grilla*, n: $\mathbb{Z}$, m: $\mathbb{Z}$) {

$(\forall i : \mathbb{Z})(\forall j : \mathbb{Z})((1 \leq i \leq n \wedge 1 \leq j \leq m) \rightarrow nombreEnGrilla(g, i, j)$

}

`pred nombreEnGrilla` (g: *Grilla*, i: $\mathbb{Z}$, j: $\mathbb{Z}$) {

$(\exists c : GPS \times GPS \times Nombre)(c \in Grilla \wedge (c_2)_0 = i \wedge (c_2)_1 = j)$

}

`pred grillaValida` (g: *Grilla*) {

$(\exists n : \mathbb{Z})(\exists m : \mathbb{Z})(\exists esq1 : GPS)(\exists esq2 : GPS)($
$gpsValido(esq1) \wedge$
$gpsValido(esq2) \wedge$
$lat(esq1) < lat(esq2) \wedge$
$lng(esq1) < lng(esq2) \wedge$
$n > 0 \wedge$
$m > 0 \wedge$
$permiteCeldasCuadradas(n, m, esq1, esq2) \wedge$
$|g| = n \times m \wedge$
$nombresOk(g) \wedge$
$puntosOk(g, esq1, esq2, n, m))$

}

`pred limitesOk` (grilla: *Grilla*, esq1: *GPS*, esq2: *GPS*, n: $\mathbb{Z}$, m: $\mathbb{Z}$) {

$limitesPorNombre(grilla, esq1, (lng(esq2) - lng(esq1))/m, (lat(esq2) - lat(esq1))/n)$

}

`pred limitesPorNombre` (grilla: *Grilla*, esq: *GPS*, ancho: $\mathbb{R}$, alto: $\mathbb{R}$) {

$(\forall i : \mathbb{Z})(0 \leq i < |grilla| \rightarrow_L$
$(esEsqInfIzq(grilla[i]_0), esq, ancho, alto, grilla[i]_2) \wedge esEsqSupDer(grilla[i]_1), esq, ancho, alto, grilla[i]_2))$

}

`pred esEsqSupDer` (p: *GPS*, esq: *GPS*, ancho: $\mathbb{R}$, alto: $\mathbb{R}$, nombre: *Nombre*) {

$lat(p) = nombre_0 * alto + lat(esq) \wedge$
$lng(p) = nombre_1 * ancho + lng(esq)$

}

`pred esEsqInfIzq` (p: *GPS*, esq: *GPS*, ancho: $\mathbb{R}$, alto: $\mathbb{R}$, nombre: *Nombre*) {

$lat(p) = (nombre_0 - 1) * alto + lat(esq) \wedge$
$lng(p) = (nombre_1 - 1) * ancho + lng(esq)$

}

`pred viajeEnGrilla` (v: *Viaje*, g: *Grilla*) {

$(\exists trayecto : seq\langle GPS\rangle)(esTrayectoDeViaje(t, v) \wedge trayectoEnGrilla(trayecto, g))$

}

`pred puntosOk` (g: *Grilla*, esq1: *GPS*, esq2: *GPS*, n: $\mathbb{Z}$, m: $\mathbb{Z}$) {

$(\exists lado : \mathbb{Z})lado = (lat(esq2) - lat(esq1))/n \wedge limitesPorNombre(g, esq1, lado, lado)$

}

`pred viajesValidos` (vs: $seq\langle Viaje\rangle$) {

$(\forall i : \mathbb{Z})(0 \leq i < |vs| \rightarrow_L validoV(vs[i]))$

}

4. Testing

Para el siguiente ejercicio se pide escribir tests para la siguiente especificación. Los tests deben ser útiles para encontrar errores en la implementación del algoritmo que la catedra posee. **Opcional**: Implementar el algoritmo.

`proc completarHuecos` (inout v: *Viaje*, in faltantes: $seq\langle\mathbb{Z}\rangle$) {

`Pre` $\{v = vPrev \wedge$
$semiValido(v, faltantes) \wedge$
$faltantesEnRango(faltantes, v) \wedge$
$noHuecosEnPuntas(faltantes) \wedge_L$
$hayHuecosEnPosicion(faltantes, v) \wedge$
$|faltantes| = cantidadHuecos(v)\}$

`Post` $\{|v| = |vPrev| \wedge_L$
$todosLosHuecosCompletos(v, vPrev, faltantes) \wedge$
$losNoHuecosNoCambian(v, vPrev, faltantes)\}$

}

`pred semiValido` (v: *Viaje*, faltantes: $seq\langle\mathbb{Z}\rangle$) {

$|v| > 2 \wedge tiemposDistintos(v) \wedge (\forall i : \mathbb{Z})(0 \leq i < |v| \rightarrow_L (t(v[i]) \geq 0 \wedge gpsValido(gps(v[i])) \vee i \in faltantes))$

}

`pred faltantesEnRango` (faltantes: $seq\langle\mathbb{Z}\rangle$, v: *Viaje*) {

$(\forall x : \mathbb{Z})x \in faltantes \longrightarrow 0 \leq x < |v|$

}

`pred noHuecosEnPuntas` (faltantes: $seq\langle\mathbb{Z}\rangle$, v: *Viaje*) {

$0 \notin faltantes \wedge |v| - 1 \notin faltantes$

}

`pred hayHuecosEnPosicion` (faltantes: $seq\langle\mathbb{Z}\rangle$, v: *Viaje*) {

$(\forall x : \mathbb{Z})x \in faltantes \rightarrow_L hayHueco(v[x])$

}

`aux cantidadDehuecos` (v: *Viaje*) : $\mathbb{Z} = \sum_{i=0}^{|v|-1}$ if $hayHueco(v[i])$ then 1 else 0 fi ;

`pred hayHueco` (x: $Tiempo \times GPS$) {

$lat(gps(x)) = -1000 \wedge lng(gps(x)) = -1000$

}

`pred todosLosHuecosCompletos` (v: *Viaje* , $vPrev$: *Viaje*, faltantes: $seq\langle\mathbb{Z}\rangle$) {

$(\forall x : \mathbb{Z})x \in faltantes \rightarrow_L huecoCompletoCorrectamente(v[x], vPrev)$

}

`pred huecoCompletoCorrectamente` (hueco: $Tiempo \times GPS$, $vPrev$: *Viaje*) {

$(\exists xAnterior : Tiempo \times GPS)(\exists xSiguiente : Tiempo \times GPS)$
$losDosPuntosMasCercanos(xAnterior, xSiguiente, hueco, vPrev) \wedge t(xAnterior) < t(hueco) < t(xSiguiente) \wedge$
$distanciaYTiempoProporcionales(xAnterior, xSiguiente, hueco)$

}

`pred distanciaYTiempoProporcionales` (xAnterior: $Tiempo \times GPS$, xSiguiente: $Tiempo \times GPS$, hueco: $Tiempo \times GPS$)) {

$(\exists T : \mathbb{R})T = \frac{t(hueco) - t(xAnterior)}{t(xSiguiente) - t(xAnterior)} \wedge$
$t(hueco) = t(xAnterior) + (t(xSiguiente) - t(xAnterior)) * T \wedge$

$dist(gps(hueco), gps(xAnterior)) = T * dist(gps(pAnterior), gps(xSiguiente))$

}

`pred losDosPuntosMasCercanos` (xAnterior: $Tiempo \times GPS$, xSiguiente: $Tiempo \times GPS$, hueco: $Tiempo \times GPS$, vPrev: $Viaje$)
{

$(\forall x : Tiempo \times GPS)(x \in vPrev \wedge t(xAnterior) < t(x) < t(hueco) \rightarrow_L esHueco(x)) \wedge$
$(\forall x : Tiempo \times GPS)(x \in vPrev \wedge t(hueco) < t(x) < t(xSiguiente) \rightarrow_L esHueco(x))$

}

`pred losNoHuecosNoCambian` (v: $Viaje$, $vPrev$: $Viaje$, faltantes: $seq\langle\mathbb{Z}\rangle$) {

$(\forall i : \mathbb{Z})(0 \leq i < |v| \wedge i \notin faltantes \rightarrow_L v[i] = vPrev[i])$

}

5. Entrada/Salida

Implementar las siguientes funciones.

Para todas las latitudes y longitudes es necesario que la salida contenga al menos 5 dígitos de precisión.

1. `void escribirGrilla(grilla g, string nombreArchivo)`

   Que dado un nombre de archivo `nombreArchivo` y una grilla `g`, almacene la grilla en el archivo indicado respetando el siguiente formato:

   Cada fila del archivo contendrá cada una de las celdas de la grilla, en donde cada columna (separadas por tabs) contenga:

   - Latitud de la esquina 1.
   - Longitud de la esquina 1.
   - Latitud de la esquina 2.
   - Longitud de la esquina 2.
   - Nombre de la celda

   Ver `grilla_ejemplo.csv` para un ejemplo de grilla.

2. `escribirRecorridos(vector<recorrido> recorridos, string nombreArchivo)` Que dado un nombre de archivo `nombreArchivo` y una secuencia de recorridos `r`, escriba todos los recorridos en un mismo archivo. Cada fila del archivo deberá contener un punto GPS con las siguientes columnas:

   - Identificador de recorrido (números de 0 a $n-1$ donde n es la cantidad de recorridos a dibujar).
   - Latitud del punto.
   - Longitud de punto.

   Dentro de cada recorrido, los puntos deberán estar ordenados (la primera línea debe corresponderse con el primer punto del recorrido, la segunda con el segundo, y así sucesivamente).

   Ver `recorridos_ejemplo.csv` para un ejemplo de recorridos.

6. Interfaz gráfica

Los archivos del TP incluyen el código de una interfaz gráfica que les va a permitir visualizar recorridos y grillas exportadas desde su programa en `c++`. El código de la misma se encuentra en el archivo *dibujar.py* (python).

Las máquinas de los laboratorios ya tienen casi todas las dependencias necesarias (en Linux), pero en caso de querer utilizar una máquina personal, se necesita tener instalado python3 (recomendamos a través de anaconda que cuenta con versiones para Linux, Windows y Mac. Por último, necesitamos la librería para dibujar mapas folium. En caso de utilizar anaconda, instalar el paquete `folium` con (`pip3 install folium --user`).

El script *dibujar.py* debe ejecutarse desde una terminal en donde recibe los parámetros necesarios.

Ejemplo de uso:

```
⟶  python3 dibujar.py --grilla ./grilla_ejemplo.csv --recorridos ./recorridos_ejemplo.csv
```

- `--grilla`: nombre del archivo csv en donde se exportó la grilla generada desde su programa.
- `--recorridos`: nombre del archivo csv en donde se exportaron los distintos recorridos desde su programa.

Al ejecutar la interfaz, se generará un archivo *mapa.html* en donde verán un mapa de la ciudad de córdoba junto a los recorridos que hayan decidido exportar.

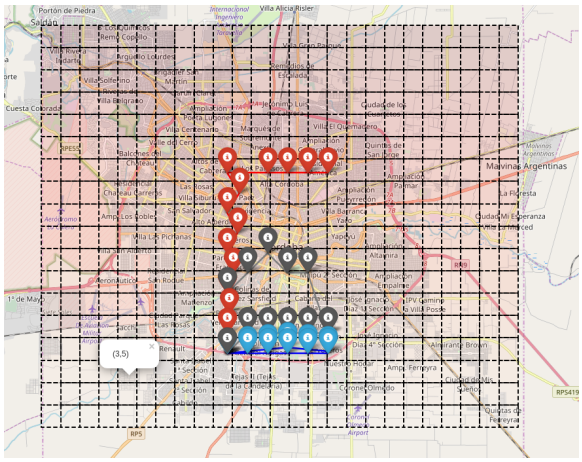

Figura 1: Ejemplo grilla

Además haciendo click en las celdas podrán ver el nombre asociado.

7. Análisis de cobertura

Para realizar el análisis de cobertura de código utilizaremos la herramienta Gcov, que es parte del compilador GCC. El target *transporteUrbanoTest* ya está configurado para generar información de cobertura de código en tiempo de compilación. Esta información estará en archivos con el mismo nombre que los códigos fuente del TP, pero con extensión `*.gcno`. Al ejecutar los casos de test, se generarán en el mismo lugar que los `*.gcno` otra serie de archivos con extensión `*.gcda`. Una vez que tenemos ambos conjuntos de archivos, ejecutar el siguiente comando:

```
⟶  lcov --capture --directory transporteUrbano --output-file coverage.info
```

Se generará el archivo `coverage.info` que luego podremos convertir a HTML para su visualización con el siguiente comando:

```
⟶  genhtml coverage.info --output-directory cobertura
```

Finalmente, se generará un archivo `index.html` dentro de `salida/cobertura` con el reporte correspondiente. Utilizar cualquier navegador para verlo.

Para mayor información, visitar:
`https://medium.com/@naveen.maltesh/generating-code-coverage-report-using-gnu-gcov-lcov-ee54a4de3f11`.

Términos y condiciones

El trabajo práctico se realiza de manera grupal con grupos de exactamente 2 personas. Para aprobar el trabajo se necesita:

- Que todos los ejercicios estén resueltos.
- Que las soluciones sean correctas.
- Que todos los tests provistos por la cátedra funcionen.
- Que las soluciones sean prolijas: evitar repetir implementaciones innecesariamente y usar adecuadamente funciones auxiliares.
- Que los test cubran todas las lineas de las funciones.

Pautas de Entrega

Se debe enviar un e-mail a la dirección tpalgo1@gmail.com. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[ALGO1;TPI]` seguido inmediatamente del nombre del grupo.
- En el cuerpo del email deberán indicar: Nombre, apellido, libreta universitaria de cada integrante.
- Debe enviarse el archivo comprimido (.zip) que contenga el archivo *solucion.cpp* completo, la carpeta de tests y la salida de lcov (el html para solucion.cpp).

Importante: se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.