Tarea Integradora III: Inmobiliaria Bienco S.A.S.

Integrantes:

Diana Sofia Olano Montaño (A00369468)

Angélica Corrales Quevedo (A00367954)

Keren López Córdoba (A00368902)

Profesor: Johnatan Garzón

Algoritmos y estructuras de datos, grupo 1

Universidad Icesi

Cali, noviembre 14 del 2021

Enunciado de la situación problemática

Bienco S.A.S. es una inmobiliaria que ofrece servicio especializado en administración de inmuebles de vivienda y comercio para arrendamiento, venta de inmuebles usados, avalúos de inmuebles, alianzas y marcación de activos fijos.

La sucursal de esta compañía, ubicada en la ciudad de Cali, desea un programa que le permita a sus agentes de bienes raíces conocer el trayecto más corto que se adecue a sus agendas de trabajo, las cuales son creadas a partir de la organización de aquellos inmuebles que diariamente necesitan enseñar a sus clientes.

Por ello, la aplicación desarrollada debe estar en la capacidad de gestionar inmuebles (agregar, eliminar y actualizar sus atributos), en este caso, cada uno de ellos contará con las siguientes características: dirección, barrio, zona (norte, sur, centro, este, oeste), tipo de inmueble (apartaestudio, apartamento, casa, local, edificio, finca, lote, oficina), precio, observaciones e información de si este se encuentra disponible para la venta o alquiler. Los anteriores datos, se deben poder guardar en memoria secundaria e ingresar de manera manual o a través del importe de archivos con extensión .csv que contengan inmuebles con sus respectivos atributos, acción por la cual el sistema debe contar con una interfaz gráfica que sea amigable para el usuario.

Además, debe poderse realizar la búsqueda de los inmuebles disponibles para visitar, utilizando como **filtro** las características deseadas por el cliente, las cuales pueden ser barrio, zona, tipo de inmueble, rango de precio, o si es para venta o alquiler. Posteriormente, a aquellos inmuebles que resulten de este filtrado, se les debe agregar la distancia en metros que poseen hacia otro inmueble, siempre y cuando entre ellos haya una vía en común o se encuentren cerca el uno del otro. Durante este proceso, la información de las distancias de los inmuebles se mostrará a medida que estas son agregadas.

Asimismo, debe ser posible encontrar dentro de los inmuebles provenientes del filtro de búsqueda, el trayecto más corto desde un inmueble seleccionado por el usuario (el deseable para iniciar el recorrido) hacia los demás inmuebles disponibles. Principalmente, se requiere encontrar un recorrido de mínima distancia que abarque la mayor cantidad de propiedades con las características solicitadas por los clientes, el cual corresponderá al trayecto sugerido.

Finalmente, el usuario podrá optar por la opción de exportar un reporte en formato PDF con la información de los trayectos más cortos a partir del inmueble seleccionado hacia los demás

inmuebles, además del trayecto sugerido. Este informe también incluiría los atributos de los inmuebles provenientes del filtro de búsqueda.

Método de la ingeniería

Contexto de la problemática

La sucursal de la inmobiliaria Bienco S.A.S ubicada en la ciudad de Cali desea un programa que le permita a sus agentes de bienes raíces conocer el trayecto más corto que se adecue a sus agendas de trabajo, las cuales consisten en organizar todos los inmuebles que necesiten enseñar a sus clientes. Se requiere encontrar un recorrido de mínima distancia que abarque la mayor cantidad de propiedades con las características solicitadas por los clientes.

Fase 1: Identificación del problema

Identificación de necesidades y síntomas:

- La solución al problema debe permitir ingresar datos de los inmuebles disponibles de manera manual o por archivo .csv, tales como: dirección, barrio, zona (norte, sur, centro, este, oeste), tipo de inmueble (apartaestudio, apartamento, casa, local, edificio, finca, lote, oficina), precio, observaciones y si este para venta o alquiler.
- La solución al problema debe incluir una interfaz gráfica con el fin de que la interacción entre el usuario y el programa sea más amigable.
- Los usuarios de la aplicación requieren eliminar o modificar los datos ingresados de los inmuebles.
- Los usuarios de la aplicación requieren realizar la búsqueda de inmuebles utilizando como criterios de búsqueda las características deseadas por el cliente, las cuales pueden ser barrio, zona, tipo de inmueble, rango de precio, o si este para venta o alquiler.
- La solución del problema debe guardar la información de los inmuebles en la memoria secundaria.
- La inmobiliaria Bienco S.A.S. no tiene una aplicación que le permita a sus agentes de bienes raíces conocer el trayecto más corto a seguir, con el fin de recorrer la mayor cantidad de inmuebles que desean mostrar a sus clientes en la ciudad de Cali.

Definición del Problema:

La Inmobiliaria Bienco S.A.S. requiere el desarrollo de un programa de software que le permita a sus agentes de bienes raíces conocer el trayecto más corto a seguir, con el fin de recorrer la mayor cantidad de inmuebles que desean mostrar a sus clientes en la ciudad de Cali.

Especificación de requerimientos funcionales

R1. Gestionar un inmueble.

- Agregar un inmueble, con su dirección, barrio, zona (norte, sur, centro, este, oeste), tipo de inmueble (apartaestudio, apartamento, casa, local, edificio, finca, lote, oficina), precio, observaciones y si este para venta o alquiler. Cabe resaltar que no podrá existir un inmueble con la misma dirección que otro.

- Modificar un inmueble. Se podrá modificar cualquiera de sus características.

- Eliminar un inmueble.

R2. Realizar la búsqueda de inmuebles utilizando como **filtro** las características deseadas por el cliente, las cuales pueden ser barrio, zona, tipo de inmueble, rango de precio, o si es para venta o alquiler.

R3. Guardar toda la información del programa en archivos con extensión ".ackldo" . Es decir, cada vez que se registre o actualice información de los inmuebles, esta se guardará en dichos archivos. Se guardarán los inmuebles con sus atributos, mas no los trayectos calculados y las distancias entre inmuebles que sean ingresadas.

R4. Importar datos de un archivo csv con información de inmuebles.

R5. Agregar la distancia en metros para los inmuebles que resultaron del filtro de búsqueda (desde un cierto inmueble a otro).

R6. Encontrar, dentro de los inmuebles provenientes del filtro de búsqueda, el trayecto más corto a partir de un inmueble seleccionado por el usuario hacia los demás inmuebles.

R7. Mostrar la información de las distancias entre los inmuebles resultantes del filtro de búsqueda.

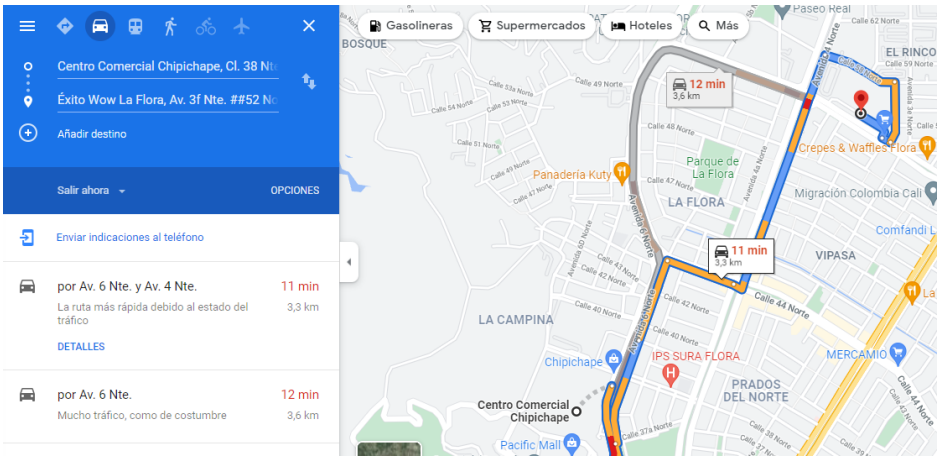
R8. Determinar, dentro de los inmuebles provenientes del filtro de búsqueda, el trayecto más corto a partir de un inmueble seleccionado por el usuario que abarque la mayor cantidad de inmuebles.

R9. Exportar reporte en formato PDF con información de los trayectos más cortos a partir de un inmueble seleccionado por el usuario hacia los demás inmuebles, además del trayecto sugerido. Este informe incluye los datos de los inmuebles provenientes del filtro de búsqueda.

Fase 2: Recopilación de la información necesaria

Definiciones

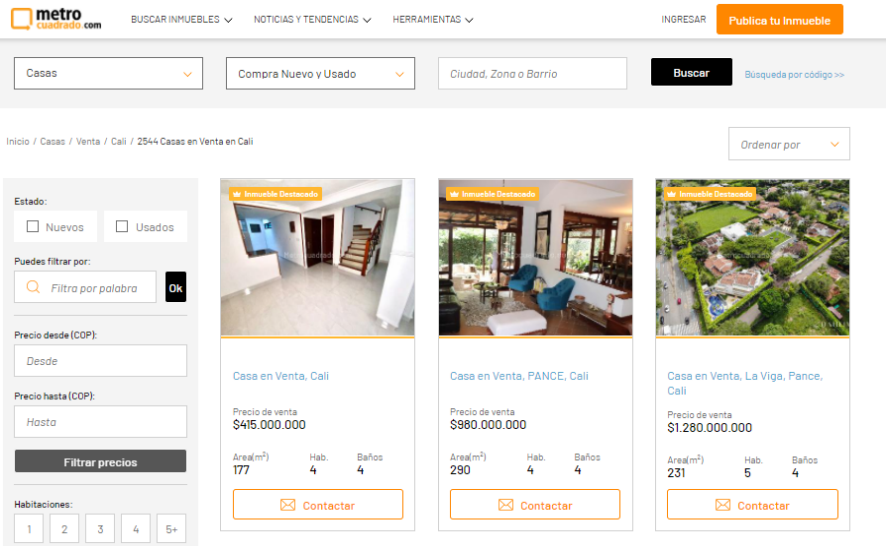
- **Ejemplo de plataforma web que brinda información de rutas entre dos lugares:**



Google. (2021). *Google Maps*. Google maps. Recuperado de

https://www.google.com.co/maps/

- **Ejemplo de plataforma web que brinda información de inmuebles de acuerdo a una búsqueda realizada:**



Metro cuadrado. (2021). *Casas en Venta en Cali - Vivienda Nueva y Usada*.

metrocuadrado.com. Recuperado de

https://www.metrocuadrado.com/casas/venta/cali/

- **Grafo dirigido:** Un grafo es una composición de un conjunto de objetos conocidos como nodos o vértices que se relacionan con otros nodos a través de un conjunto de conexiones conocidas como aristas. Un grafo dirigido, conocido también como dígrafo consta de un conjunto de vértices y aristas donde cada arista se asocia de forma unidireccional a través de una flecha con otro. Las aristas dependiendo de su salida o ingreso reciben la calificación de entrante o saliente, la condición común, es que siempre tienen un destino hacia un nodo.

  GraphEverywhere. (2020, 10 marzo). *Qué son los grafos*. Recuperado de https://www.grapheverywhere.com/que-son-los-grafos/

- **Algoritmo de Dijkstra:** Es un algoritmo eficiente (de complejidad O (n^2), donde "n" es el número de vértices) que sirve para encontrar el camino de coste mínimo desde un nodo origen a todos los demás nodos del grafo. Fue diseñado por el holandés Edsger Wybe Dijkstra en 1959. Este algoritmo es un típico ejemplo de algoritmo ávido, que resuelve los problemas en sucesivos pasos, seleccionando en cada paso la solución más óptima con el objeto de resolver el problema.

  Universidad Don Bosco. (2019). *Programación IV. Guía No. 10: Algoritmos para la ruta más corta en un grafo*. Recuperado de https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-10.pdf

- **Algoritmo de Floyd-Warshall:** Es para encontrar rutas más cortas en un gráfico ponderado con ponderaciones de borde positivas o negativas. Una sola ejecución del algoritmo encontrará las longitudes (pesos sumados) de las rutas más cortas entre todos los pares de vértices. Con una pequeña variación, puede imprimir la ruta más corta y puede detectar ciclos negativos en un gráfico. Floyd-Warshall es un algoritmo de programación dinámica.

  Algorithm - Algoritmo de Floyd-Warshall. (s. f.). Learntutorials. Recuperado de https://learntutorials.net/es/algorithm/topic/7193/algoritmo-de-floyd-warshall

- **Breadth-first-search (BFS):** Es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo, comenzando en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo), para luego explorar todos los vecinos de este nodo. A continuación, para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el grafo. Cabe resaltar que si se encuentra el nodo antes de recorrer todos los nodos, concluye la búsqueda.

  DFS vs BFS. (2020, 25 mayo). Encora. Recuperado de https://www.encora.com/es/blog/dfs-vs-bfs

- **Depth-first-search (DFS):** Es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo. Su funcionamiento consiste en ir expandiendo cada uno de los nodos que va localizando, de forma recurrente (desde el nodo padre hacia el nodo hijo). Cuando ya no quedan más nodos que visitar en dicho camino, regresa al nodo predecesor, de modo que repite el mismo proceso con cada uno de los vecinos del nodo. Cabe resaltar que si se encuentra el nodo antes de recorrer todos los nodos, concluye la búsqueda.

  DFS vs BFS. (2020, 25 mayo). Encora. Recuperado de https://www.encora.com/es/blog/dfs-vs-bfs

- **Algoritmo de Prim:** Se utiliza para encontrar el árbol de expansión mínimo de un grafo. Este encuentra el subconjunto de aristas que incluye todos los vértices del grafo de modo que la suma de los pesos de las aristas se pueda minimizar. El algoritmo comienza con un solo nodo y explora todos los nodos adyacentes con todos los bordes de conexión en cada paso. Se seleccionan los bordes con pesos mínimos que no causen ciclos en el grafo.

  Prim's Algorithm. (s. f.).Javatpoint. Recuperado de https://www.javatpoint.com/prim-algorithm

- **Algoritmo de Kruskal:** Es un algoritmo de la teoría de grafos para encontrar un árbol recubridor mínimo en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor total de todas las aristas del árbol es el mínimo. Si el grafo no es conexo, entonces busca un bosque expandido mínimo (un árbol expandido mínimo para cada componente conexa). El algoritmo de Kruskal es un ejemplo de algoritmo voraz.

  Algoritmo de Kruskal. (s. f.). i3campus. Recuperado de http://i3campus.co/CONTENIDOS/wikipedia/content/a/algoritmo_de_kruskal.html

Fase 3: Búsqueda de soluciones creativas

Técnica empleada: Lluvia de ideas

"Es una técnica de pensamiento creativo utilizada para estimular la producción de un elevado número de ideas, por parte de un grupo, acerca de un problema y de sus soluciones o, en general, sobre un tema que requiere de ideas originales".

Consultores, A. (2019, 16 septiembre). *Tormenta de Ideas: Creatividad para la Mejora*. Aiteco Consultores. Definición recuperada de https://www.aiteco.com/tormenta-de-ideas/

Alternativas propuestas de estructuras de datos para el desarrollo del problema:

Alternativa 1: Implementar Stack.

El stack se utilizaría para guardar a cada uno de los inmuebles resultado del filtrado, con su respectiva información. Cada inmueble se guardaría en una posición del stack, y cada posición del stack tendría además un atributo de stack en donde almacenaría los inmuebles que tienen cercanía con el determinado inmueble, con sus respectivas distancias.

Alternativa 2: Implementar Queue.

La queue se utilizaría para guardar cada uno de los inmuebles con su respectiva información, los cuales son resultado de la búsqueda realizada para encontrar aquellos que se adaptan a las necesidades del cliente. En este caso, cada inmueble además de ser almacenado dentro de una queue, estos tendrían como atributo una queue en la que se almacenarán aquellos inmuebles que se encuentran cerca, incluyendo sus respectivas distancias y demás características.

Alternativa 3: Implementar árbol binario de búsqueda.

Se crearía primeramente un árbol binario de búsqueda ordenado alfabéticamente por las direcciones de cada uno de los inmuebles que resultan del filtrado, cuyos nodos almacenarán cada uno de ellos con su respectiva información. Asimismo, cada nodo del árbol anterior tendría como atributo otro árbol binario de búsqueda que contiene a los inmuebles cercanos a él, en este caso sería ordenado por las distancias de aquellos inmuebles adyacentes.

Alternativa 4: Implementar hashtable.

Existiría una hashtable que tenga como keys las direcciones de cada uno de los inmuebles (aquellos que resultan de la búsqueda realizada para encontrar los que se adaptan a las necesidades del cliente) y como valores una hashtable, que almacenaría los inmuebles cercanos y, en este caso, las keys serían las distancias y los valores vendrían siendo los inmuebles como tal.

Alternativa 5: Implementar árbol n-ario.

El árbol n-ario almacenaría en sus nodos a cada uno de los inmuebles provenientes del proceso de filtrado con su respectiva información. En este caso, cada nodo tendría como hijos a aquellos inmuebles que son cercanos a él. No obstante, esta estructura podría resultar poco conveniente cuando exista el caso en el que algún nodo hijo tenga más de un padre, según el contexto del problema.

Alternativa 6: Implementar grafo ponderado dirigido.

Para esta estructura cada uno de los inmuebles resultado del proceso de filtrado representan los vértices del grafo, las aristas indicarían la conexión de un inmueble a otro (si son cercanos) y, el peso de las aristas sería la distancia que existe desde un inmueble a otro.

Fase 4: Transición de las ideas a los diseños preliminares.

La alternativas 1, 2 y 5 las descartamos, ya que el uso de estructuras como lo son el Stack, el Queue y el árbol n-ario no es adecuado, teniendo en cuenta el contexto del problema. Respecto a las dos primeras estructuras tenemos que sus comportamientos LIFO (last-in-first-out) y FIFO (first-in-first-out) serían inoperantes para el problema a resolver. Por otro lado, el árbol n-ario no es conveniente, dado que, según el contexto del problema, un inmueble podría tener más de una conexión a otro inmueble cercano, por lo tanto, cuando se presente el caso en el que algún nodo hijo tenga más de un padre, esta estructura no sería de utilidad.

Si revisamos cuidadosamente las demás alternativas obtenemos lo siguiente:

- **Alternativa 3:** Implementar árbol binario de búsqueda.
  Esta alternativa se puede considerar para resolver el problema, teniendo en cuenta que asigna a cada inmueble la conexión con otros inmuebles cercanos. Sin embargo, determinar qué inmueble tiene conexión con otro no se resolvería de manera directa, y hasta podría ser complejo, pues el árbol que tiene cada uno de los inmuebles está ordenado con las distancias.

- **Alternativa 4:** Implementar hashtable.
  Esta alternativa permite simular la situación, pero determinar qué inmueble tiene conexión con otro dentro de esta estructura no se resolvería de manera directa, y podría ser complejo, pues en las hashtable que tiene cada uno de los inmuebles las keys corresponden a las distancias.

- **Alternativa 6:** Implementar grafo ponderado dirigido.
  Esta alternativa se adapta completamente para la solución del problema dado, ya que cada uno de los inmuebles resultado del proceso de filtrado representan los vértices del grafo, las aristas indicarían la conexión de un inmueble a otro (si son cercanos) y, el peso de las aristas sería la distancia que existe desde un inmueble a otro.

Fase 5: Evaluación y selección de la mejor solución.
Se definen los criterios que permitirán evaluar las alternativas de solución y con base en este resultado elegir la solución que mejor satisface las necesidades del problema planteado. Los criterios que escogimos en este caso son los que enumeramos a continuación. Al lado de cada

uno se ha establecido un valor numérico con el objetivo de establecer un peso que indique cuáles de los valores posibles de cada criterio tienen más peso (i.e., son más deseables).

Criterios

- **Criterio A: Facilidad para determinar conexión entre inmuebles cercanos.**
  [1] Es más complejo
  [2] No es complejo

- **Criterio B: Posible adaptación de las estructuras escogidas con respecto al contexto del problema.**
  [1] No se adapta
  [2] Se adapta

Evaluación

Alternativa	Criterio A	Criterio B	Total
# 3: Implementar árbol binario de búsqueda.	[1] Es más complejo	[2] Se adapta	3
#4: Implementar hashtable.	[1] Es más complejo	[2] Se adapta	3
#6: Implementar grafo ponderado dirigido.	[2] No es complejo	[2] Se adapta	4

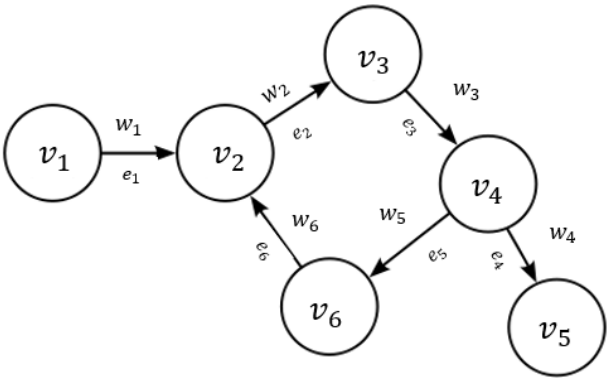
Selección

De acuerdo con la evaluación anterior se debe seleccionar la **Alternativa 6**, ya que obtuvo la mayor puntuación (4) de acuerdo con los criterios definidos.

Implementación del diseño.

Diseño del TAD para la estructura de datos requerida

TAD Grafo ponderado dirigido



$$G = (V, E)$$

$$V = \{v_1, v_2, v_3, \dots, v_n\}$$

$$E = \{e_1, e_2, e_3, \dots, e_n\} \mid e_n = (v_i, v_j, w_n)$$

$$W = \{w_1, w_2, w_3, \dots, w_n\}$$

Donde V es el conjunto de vértices, E el conjunto de aristas y W los pesos de las aristas.

$$\{inv: v_i \in V,\ e_i \in E,\ w_i \in W,$$
$$w_i \geq 0 \quad \wedge \quad w_i = \infty \leftrightarrow e_i \notin E\ \}$$

Operaciones principales:

- $Graph(Constructora)$: $\rightarrow Graph$
- $AddVertex(Modificadora)$ $Graph\ x\ Vertex \rightarrow Graph$
- $AddEdge\ (Modificadora)$: $Graph\ x\ Vertex\ x\ Vertex\ x\ Weight \rightarrow Graph$
- $RemoveVertex\ (Modificadora)$: $Graph\ x\ Vertex \rightarrow Graph$
- $RemoveEdge\ (Modificadora)$: $Graph\ x\ Edge \rightarrow Graph$
- $Dijkstra\ (Analizadora)$: $Graph\ x\ Vertex \rightarrow Graph$
- $Floyd - Warshall\ (Analizadora)$: $Graph \rightarrow Graph$
- $BFS\ (Analizadora)$: $Graph\ x\ Vertex \rightarrow Graph$
- $DFS\ (Analizadora\ )$: $Graph \rightarrow Graph$

Graph()

"Crea un nuevo grafo ponderado dirigido vacío"

{pre: TRUE}

{post: grafo vacío creado}

AddVertex(Graph, Vertex)

"Agrega un vértice v_i al grafo ponderado dirigido"
{pre: grafo previamente inicializado, vértice v_i diferente de nulo y $v_i \notin V$ }
{post: grafo con un vértice v_i añadido}

AddEdge(Graph, Vertex, Vertex, Weight)

"Agrega una arista entre dos vértices y con su peso, al grafo ponderado dirigido"

{pre: grafo previamente inicializado, vértices diferentes de nulo, peso≥ 0 }

{post: grafo con una arista añadida}

RemoveVertex(Graph, Vertex)

"Remueve un vértice del grafo ponderado dirigido"

{pre: grafo previamente inicializado, vértice v_i diferente de nulo y $v_i \in V$ }
{post: vértice v_i del grafo eliminado}

RemoveEdge(Graph, Edge)

"Remueve una arista del grafo ponderado dirigido"

{pre: grafo previamente inicializado, arista $e_i \in E$ }
{post: arista del grafo eliminada}

Dijkstra(Graph, Vertex)
"Permite obtener las distancias más cortas entre un vértice v_i y los demás vértices del grafo ponderado dirigido"

{pre: grafo previamente inicializado, vértice v_i diferente de nulo y $v_i \in V$ }

{post: Distancias obtenidas del grafo}

Floyd - Warshall(Graph)
“Permite obtener las distancias más cortas entre los vértices del grafo ponderado dirigido”

{pre: grafo previamente inicializado, $\forall v_i \mid v_i \in V$ }
{post: Distancias obtenidas del grafo}

BFS (Graph, Vertex)
“Recorre el grafo ponderado dirigido a partir de un vértice, por anchura”

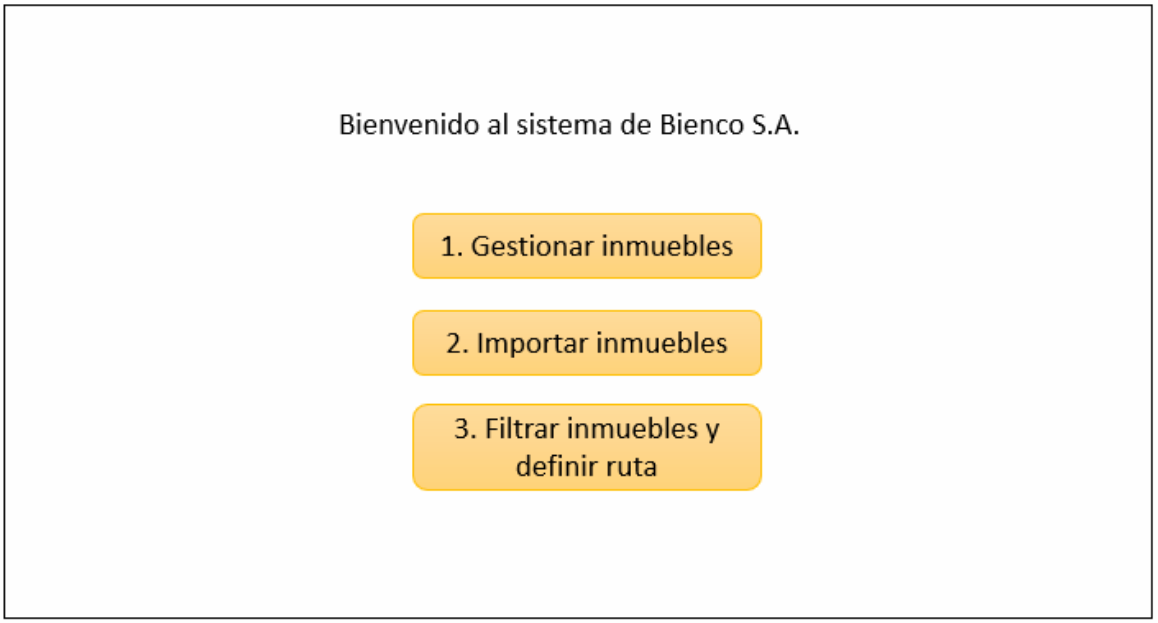
{pre: grafo previamente inicializado, vértice $v_i \in V$ }
{post: grafo recorrido}

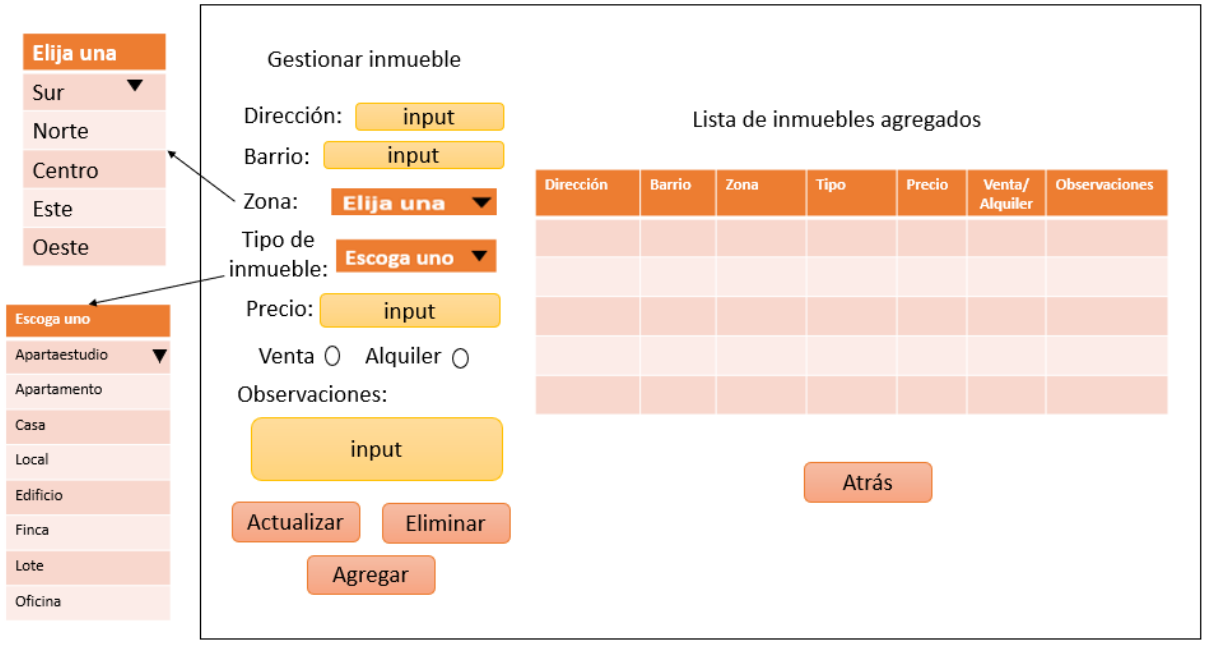
DFS (Graph)
“Recorre el grafo ponderado dirigido, por profundidad”

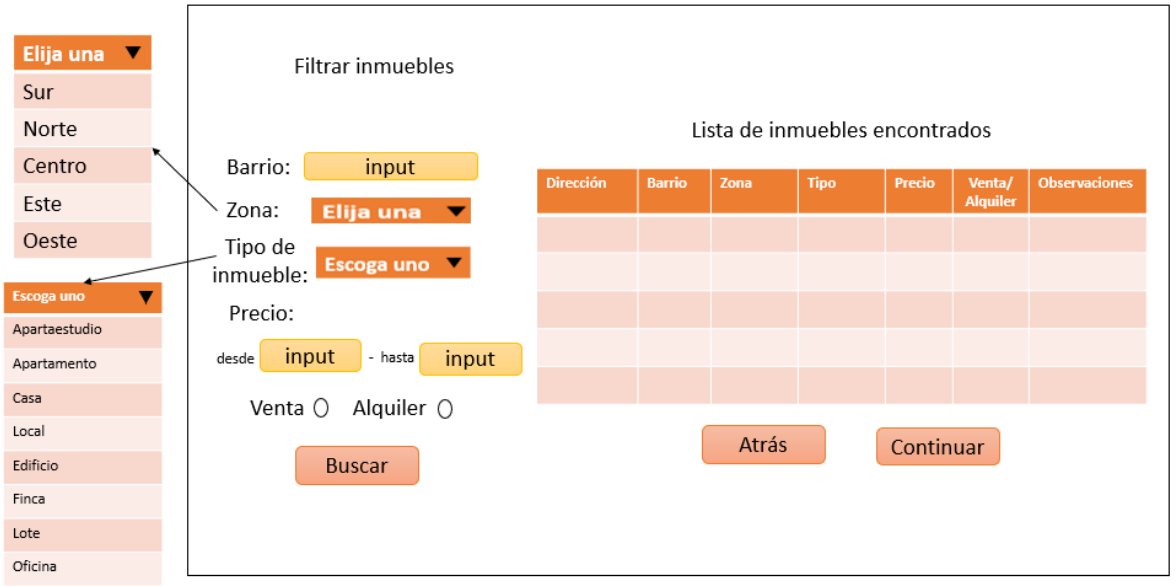
{pre: grafo previamente inicializado}

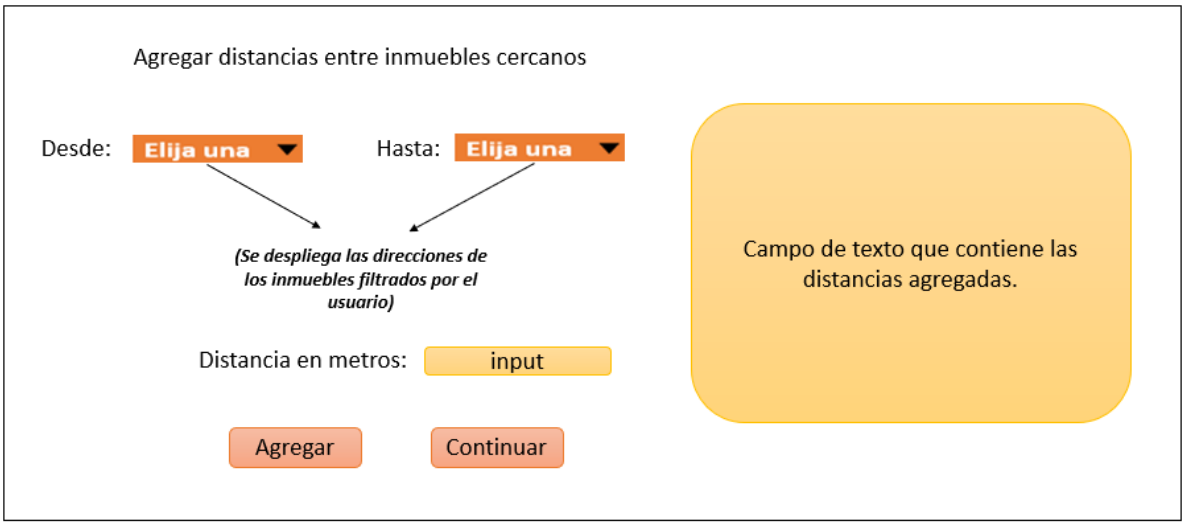
{post: grafo recorrido}

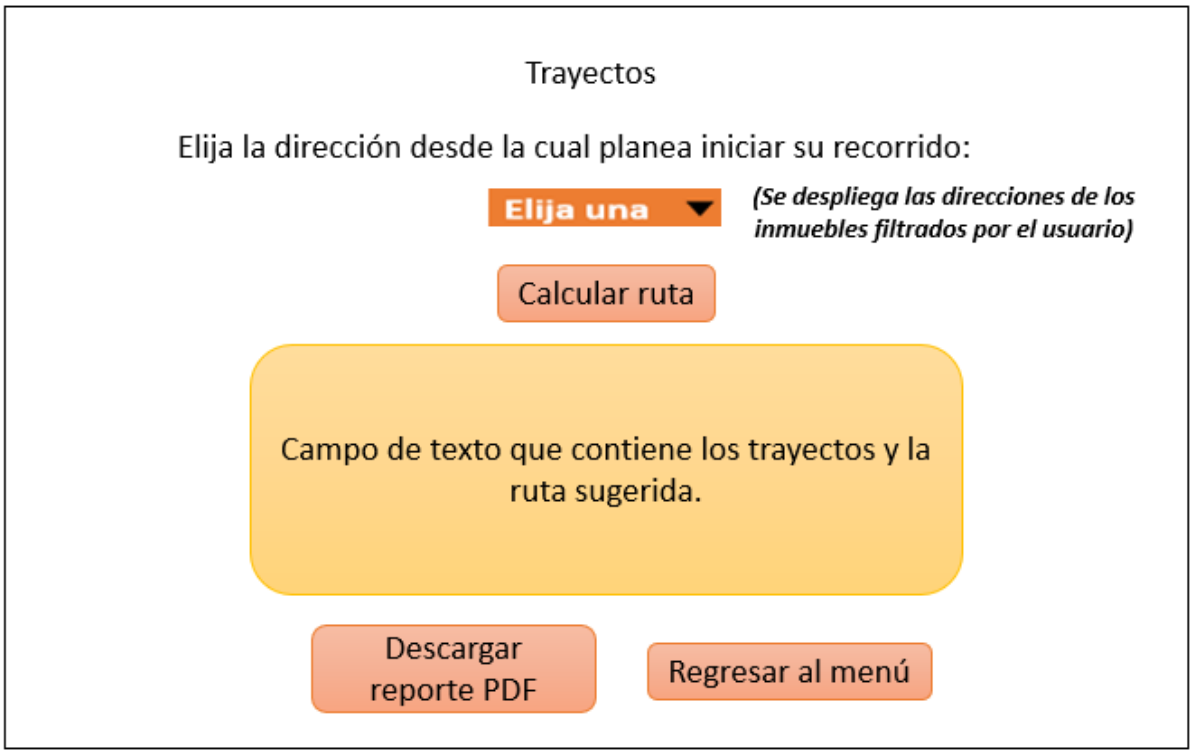
Mockups preliminares




Elija una
Sur
Norte
Centro
Este
Oeste
Escoga uno
Apartaestudio
Apartamento
Casa
Local
Edificio
Finca
Lote
Oficina
Gestionar inmueble
Dirección: input
Barrio: input
Zona: Elija una
Tipo de inmueble: Escoga uno
Precio: input
Venta Alquiler
Observaciones:
input
Actualizar
Eliminar
Agregar
Lista de inmuebles agregados
Dirección
Barrio
Zona
Tipo
Precio
Venta/ Alquiler
Observaciones
Atrás


Elija una
Sur
Norte
Centro
Este
Oeste
Escoga uno
Apartaestudio
Apartamento
Casa
Local
Edificio
Finca
Lote
Oficina
Filtrar inmuebles
Barrio: input
Zona: Elija una
Tipo de inmueble: Escoga uno
Precio:
desde input - hasta input
Venta Alquiler
Buscar
Lista de inmuebles encontrados
Dirección
Barrio
Zona
Tipo
Precio
Venta/ Alquiler
Observaciones
Atrás
Continuar


Agregar distancias entre inmuebles cercanos
Desde:
Elija una
Hasta:
Elija una
(Se despliega las direcciones de los inmuebles filtrados por el usuario)
Distancia en metros:
input
Agregar
Continuar
Campo de texto que contiene las distancias agregadas.


Trayectos
Elija la dirección desde la cual planea iniciar su recorrido:
Elija una
(Se despliega las direcciones de los inmuebles filtrados por el usuario)
Calcular ruta
Campo de texto que contiene los trayectos y la ruta sugerida.
Descargar reporte PDF
Regresar al menú