Universidad de Concepción
Facultad de Ingeniería
Departamento de Informática y Ciencias de la Computación

Tarea Computacional

Curso

Matemáticas Discretas (503213) S2-2023

Grupo 5

Antonio Skorin García
2022402164
Ingeniería Civil Informática

Matías Olivas Henríquez
2022421061
Ingeniería Civil Informática

Guillermo Oliva Orellana
2022429887
Ingeniería Civil Informática

1. Introducción

El desafío propuesto en la tarea computacional es el desarrollo de un programa que sea capaz de encontrar la ruta más corta para desplazarse en automóvil desde un punto hacia otro dentro de un área delimitada en la ciudad de Concepción. Adicionalmente, se podrá incluir un punto intermedio en la entrada al programa, el cual deberá ser incluido en la ruta final.

Para abordar el problema planteado se decidió modelar el área del centro de concepción a través de un grafo dirigido. En este, cada intersección entre calles se convierte en un vértice del grafo, y las conexiones viales entre estas intersecciones se traducen en los arcos del digrafo, se debe tener en consideración que esto solo aplica para sentidos transitables en vehículo, es decir, existen calles con una sola dirección e incluso existen calles no transitables. A partir del grafo dirigido generado, se pueden utilizar algoritmos propios de estos para encontrar el camino más corto.

La estructura de este informe se divide en diferentes secciones. En la sección 2 se busca plantear con claridad lo que se busca aprender a través de la tarea. La sección 3 constituye la base conceptual, detallando la representación del problema como un grafo dirigido. La sección 4 está dirigida a desglosar el funcionamiento del código utilizado para la implementación del programa. Finalmente en la sección 5 se evalúa el cumplimiento de los objetivos, los problemas que surgieron en el camino, cómo los solucionamos y en qué ámbitos podría mejorar la solución propuesta.

2. Objetivos

2.1. Objetivo General

Desarrollar un programa en C o C++ que determine e imprima la ruta más corta para movilizarse desde un lugar a otro en una zona acotada del centro de Concepción, considerando las orientaciones de las calles y excepciones como lo son los tramos peatonales.

2.2. Objetivos Específicos

- Implementar una interfaz que permita al usuario ingresar puntos de partida, llegada y opcionalmente un lugar intermedio.
- Analizar el sentido del tráfico de cada calle y las excepciones para proporcionar rutas posibles.
- Lograr una representación en forma de grafo dirigido para el análisis de la ruta más corta.
- A partir del modelo, encontrar la ruta más corta entre los puntos ingresados por el usuario.
- Retornar una ruta a seguir que sea entendible para el usuario.

2.3. Aprendizajes Esperados

- Aprender a identificar y modelar problemas reales que permitan el uso de grafos.
- Aprender a implementar algoritmos y a utilizar las propiedades de los grafos para acercarse a una solución real a través del modelo.

3. Modelo

El modelo diseñado para llevar a cabo el programa se trata de un grafo dirigido; los nodos modelan las intersecciones entre las calles y los arcos representan las calles por las cuales un vehículo puede desplazarse, donde la dirección representa el sentido del tráfico.

El digrafo fue generado a través de un algoritmo el cual crea una matriz con las coordenadas de cada intersección de calles, asignando un identificador único a cada vértice. Cada vértice cuenta con una lista de adyacencia, la cual fue abstraída a una estructura de conjunto; esta se va completando según el sentido de la calle, añadiendo los vértices hacia los cuales se puede mover un vehículo desde cada intersección de calles. Además cabe recalcar que las calles horizontales poseen un número de inmueble inicial 0, incrementando de izquierda a derecha, y que las calles verticales posee un un número de inmueble inicial de 100, incrementando de abajo hacia arriba. Adicionalmente la "Diagonal" posee un número de inmueble inicial 100 en su intersección con "Chacabuco", terminando en el número de inmueble 399 en su intersección con "O'Higgins" .

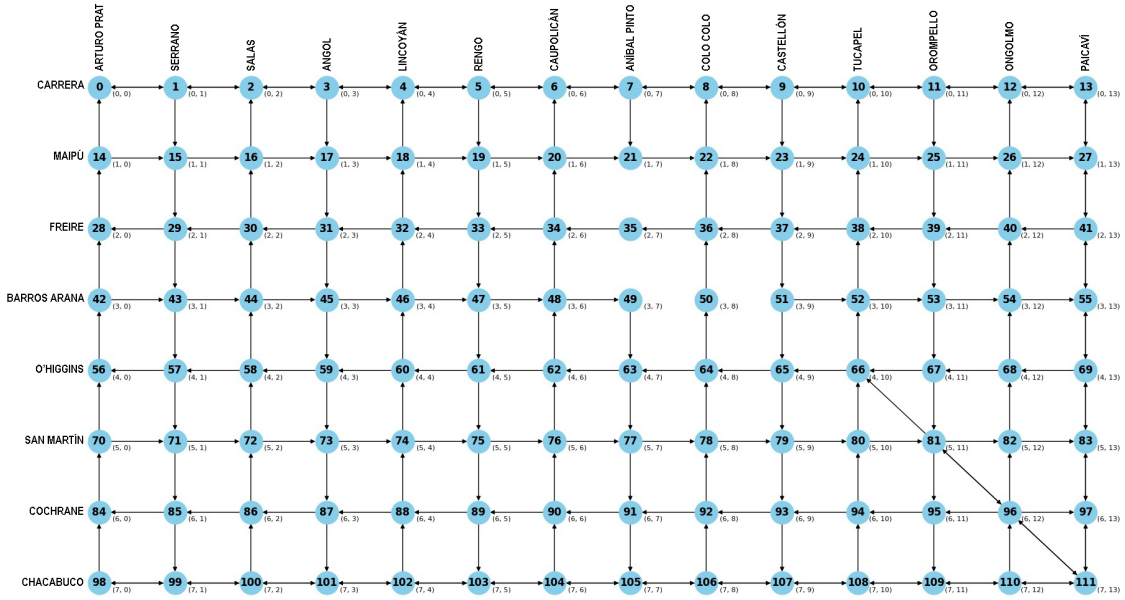

Figura 1: Representación visual del digrafo que modela el centro de Concepción.

4. Implementación

4.1. Interpretación de la Entrada

En una primera instancia, es necesario poder interpretar la entrada del usuario y traducirla a los elementos correspondientes del modelo. Para ello, el programa al iniciarse explicita el formato en el que debe hacerse ingreso de los datos, para posteriormente escanear las direcciones entregadas, realizando las revisiones oportunas e informando al usuario en caso de haberse hecho un ingreso erróneo. Una propiedad importante del modelo es aprovechar la disposición cuadriculada de las calles que permiten abstraer las intersecciones, y por lo tanto, los vértices como coordenadas, facilitando la automatización de la creación del digrafo, entre otros procesos. Debido a esto, representamos las direcciones de las intersecciones como coordenadas de una matriz. Además, para no perder información de aquellas direcciones que no se encuentran en una intersección, las coordenadas pueden tomar valores decimales. De esta manera, el programa traduce la entrada en una serie de coordenadas de valores de punto flotante.

4.2. Conjuntos de Adyacencia

La lista de adyacencia requería de ciertas propiedades; existencia única de elementos en ella, ser iterable, y ser posible la adición o eliminación de elementos con solo indicar su valor; en vista de eso, resultaba conveniente la abstracción de esta a una suerte de conjunto iterable. Su implementación concreta se logró a través de la creación de un struct "set", inicializable a través de una función, y sobre el cual se pueden realizar distintas operaciones, tales como: añadir elementos, remover elementos, revisar si un conjunto se encuentra vacío, y revisar si un elemento se encuentra en un conjunto.

4.3. Creación de Listas de Adyacencia por Vértice

La creación de las listas de adyacencia se hizo con un algoritmo el cual según las direcciones de cada calle, agrega el vértice adyacente a éste en su lista de adyacencia. Luego de forma manual, se eliminan las excepciones al algoritmo. Este se puede ver de forma más clara en el siguiente pseudocódigo.

Creación de listas de adyacencia para cada vértice

```
for CALLE in CALLES_HORIZONTALES do
    if CALLE.SENTIDO = DERECHA then
        for VERTICE in CALLE do
            Añadir a la lista de adyacencia del vértice el vértice que está a la derecha.
        end for
    else if CALLE.SENTIDO = IZQUIERDA then
        for VERTICE in CALLE do
            Añadir a la lista de adyacencia del vértice el vértice que está a la izquierda.
        end for
    else
        for VERTICE in CALLE do
            Añadir a la lista de adyacencia del vértice el vértice que está a la izquierda y a la derecha.
        end for
    end if
end for
```

El algoritmo es análogo para el caso de las Calles Verticales.

Finalmente, eliminamos los pasos peatonales y añadimos las relaciones que se escapan del algoritmo.

4.4. Búsqueda de Camino Más Corto

Para la búsqueda del camino más corto se utilizó el algoritmo de Dijkstra, este resultaba ser una buena opción teniendo en consideración la presencia de un digrafo simple con pesos uniformes. En general se realiza el siguiente procedimiento sobre una entrada de vértices en su representación numérica:

1. De acuerdo al formato de entrada por el usuario, los vértices estarían ordenados de la manera $(v_0, v_f, v_1, \ldots, v_n)$. Pero la orientación que la que debían ser recorridos es la siguiente: $(v_0, v_1, \ldots, v_n, v_f)$.

2. Una vez ordenados los vértices de la manera correcta, se debía aplicar el algoritmo de Dijkstra sobre cada par adyacente, para luego formar un camino final concatenado. Sin embargo, Dijkstra solo era capaz de generar un arreglo de padres para cada vértice, por lo que hacía falta un procedimiento para generar un camino en base al arreglo. El siguiente algoritmo genera en camino en base a un arreglo de padres, un nodo fuente, y un nodo destino:

3. Cabe mencionar que el arreglo generado habría estado al revés, por lo que al momento de visualizar el camino se debía imprimir el arreglo en orden invertido.

4. Finalmente, se debía aplicar Dijkstra para generar un camino de padres, para cada par de vértices de la entrada, para luego concatenar el resultado en un camino final:

Generar Camino: generar_camino(source, destiny, padres)

```
vertex ← destiny
path[]
while vertex ≠ source do
    path.append(vertex)
    vertex ← padre[vertex]
end while
return path
```

Generar Camino Final: generar_camino_final(nodos)

```
for i in length(nodos) do
    source ← nodos[i]
    destiny ← nodos[i + 1]
    padre ← dijkstra(source)
    camino ← generar_camino(source, destiny, padre)
    camino.reverse()
    camino.print()
end for
```

4.5. Representación de Vértices como Enteros

Al momento de traspasar intersecciones a coordenadas de enteros, se mantenía la dificultad que conlleva trabajar con la indexación de una estructura como un arreglo, por lo que era necesario llevar a cabo algún tipo de transformación a las coordenadas con tal de poder representarlas con un número entero positivo incluyendo al cero. Esto nos permitiría poder indexar cada vértice en el arreglo y conseguir su conjunto de adyacencia.

1. Para cada número en nuestro recorrido, debía existir un único par de coordenadas que fuera mapeable a él, en efecto, esta es la definición de inyectividad.

2. Debíamos ser capaces de realizar la transformación inversa para los elementos particulares sobres los cuales aplicamos la función en un comienzo. Luego, se quería que la función fuese biyectiva.

3. El intervalo de recorrido debía ser lo suficientemente acotado como para no incurrir en un mal uso de la memoria; a modo de ejemplo, si nuestro intervalo era de la manera $I = [0, 10^9 - 1]$, se hubiera tenido que generar un arreglo con mil millones de elementos, que en caso de haber sido tipo char, incurriría en el uso de mil millones de bytes, es decir, 1 GB de memoria, se quería evitar esto.

Con los requerimientos anteriores en consideración, se implementó la siguiente solución, que se nombró "linealización", debido a que se puede visualizar cómo "estirar" un arreglo bidimensional en un arreglo de una dimensión:

Tuple a entero: tuple_a_int(int a, int b, int n)

```
return a * n + b
```

Donde se puede considerar al parámetro n como la cantidad de columnas de la matriz a estirar, y la tupla (a, b) como la indicación en coordenadas (fila, columna) de un vértice.

Entero a tupla: int_a_tupla(int c, int n)

return (c / n, c % n)

4.6. Interpretación de la Salida

Para concluir la interacción con el usuario, es necesario transferir la información calculada en los pasos previos. Esto se logra transformando las coordenadas de cada vértice en un texto que indica las calles que se cruzan en dicho punto, con el siguiente formato:"[Calle Horizontal] con [Calle Vertical]". En la imagen siguiente, la solicitud fue ir desde "Paicaví 240" hasta "Barros Arana 910". El programa imprimirá las intersecciones resaltadas en color azul.

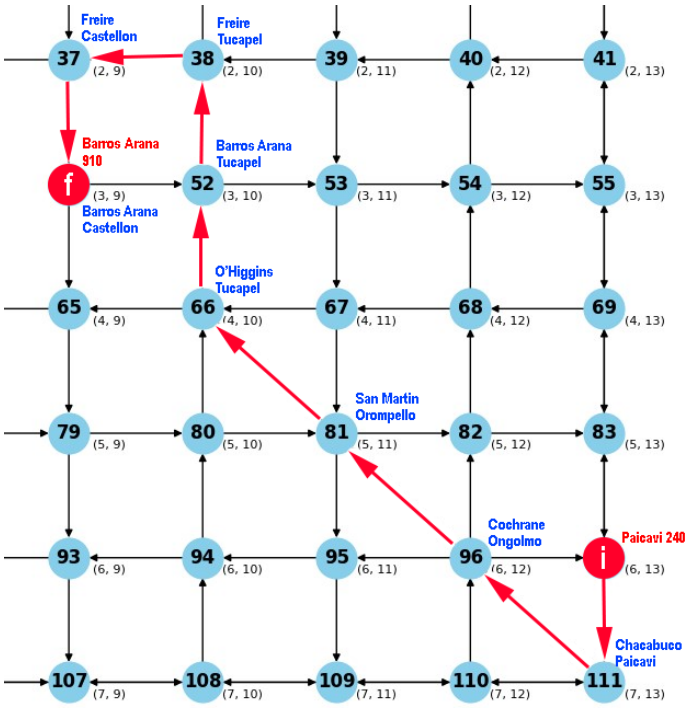

Figura 2: Representación del funcionamiento del programa.

4.7. Integración

Una vez implementado todo lo anterior, se debió integrar en una secuencia de pasos descrita de la siguiente manera:

1. Generar la lista adyacencia con los conjuntos.
2. Interpretar la entrada del usuario.
3. Convertir la entrada en tuple a enteros en el intervalo $I = [0, 111]$
4. Generar un camino final con la lista de nodos ordenada.
5. Transformar los vértices en su estado numérico a nombres de intersecciones de calles.
6. Desplegar la información del camino al usuario.
7. Repetir hasta que el usuario decida salir de programa.

5. Conclusiones

En este trabajo logramos desarrollar un proyecto que cumple satisfactoriamente los desafíos planteados a resolver. Se desarrolló un programa que permite solucionar una problemática mediante la abstracción a un modelo matemático.

Se representó la situación descrita por el problema en un grafo orientado, para así aplicar las propiedades y resultados que posee esta estructura. Concretamente, aplicamos el algoritmo de Dijkstra que nos permite encontrar siempre la ruta más corta, y así, resolver la problemática planteada.

El proceso de traducir la realidad en el modelo no es trivial, y fue la dificultad principal del proyecto: La cantidad de información presentada demandaba de una forma inteligente de recabarla. Analizando el problema, concluimos aprovechar la disposición espacial de la información para automatizar la creación del modelo.

Durante el desarrollo del proyecto resultó crucial lograr una coordinación del grupo óptima, donde cada integrante se asegurará de estar trabajando sobre el mismo modelo para así conseguir un reparto de las responsabilidades que permitieran hacer del desarrollo un proceso expedito y producir un resultado cohesivo.

En el desarrollo procuramos hacer un proyecto extensible, y por lo tanto, hay varias funcionalidades que se podrían implementar en un futuro. Actualmente, las características de las cuales carece el proyecto que consideramos más pertinentes son: Mejorar la impresión de la respuesta del programa de forma que sea más legible, ya sea mediante una interfaz gráfica o mediante instrucciones más claras del camino a recorrer; obtener la información de las calles y del transito de forma más escalable, de manera que sea posible extender las calles que contempla el programa. Entre otras.

En general, este proyecto fue para nosotros una instancia para poder aplicar los conocimientos adquiridos durante este semestre en el curso, aprendiendo a modelar una situación para así poder resolver un problema real.