



Tecnológico de Monterrey

ITESM Campus Santa Fe

Nombre del bloque:

TC2038: Análisis y diseño de algoritmos avanzados (Gpo 602)

Nombre del entregable:

E2. Actividad Integradora 2 - Reflexión

Equipo:

Aquiba Yudah Benarroch Bittán, A01783710

Diego Cordova Rodríguez, A01781166

Lorena Estefanía Chewtat Torres, A01785378

Profesora:

Lizbeth Peralta Malváez

Fecha de entrega:

20 de noviembre de 2025

Introducción

En este documento de reflexión, se presentan las soluciones implementadas para dar respuesta a las necesidades de una empresa que quiere incursionar en los servicios de internet de una ciudad, en diferentes colonias.

Para esto, se requiere realizar un programa en C++ capaz de realizar lo siguiente:

- Desplegar la forma óptima de cablear con fibra óptica conectando colonia (lista de arcos de la forma (A,B).
- Mostrar la ruta a seguir por el personal que reparte correspondencia, considerando inicio y fin en la misma colonia.
- Distancia al servidor más cercano (indicando la ubicación del mismo), a partir de la ubicación de una nueva casa.

Con el fin de dar solución a los retos propuestos, se plantea la utilización de diferentes algoritmos (voraces, branch and bound y distancia euclidiana con pares cortos). En este documento, se analizará cada uno en términos de complejidad computacional.

Algoritmos aplicados a la situación problema

A continuación se presenta a detalle cada uno de los algoritmos implementados en el código de la solución final propuesta, así como su uso, forma de resolver el problema y complejidad computacional.

Algoritmos voraces (Kruskal)

Problema: Desplegar la forma óptima de cablear con fibra óptica conectando colonia (lista de arcos de la forma (A,B)).

El algoritmo de Kruskal fue implementado para encontrar la forma óptima de cablear (conectar) con fibra óptica colonias, de tal forma que puedan compartir información entre sí. De esta manera, Kruskal es un algoritmo voraz que busca tomar la mejor decisión posible en base a la información que tiene en el momento, consistiendo en lo siguiente:

- Ordenar todas las aristas del grafo en orden ascendente
- Seleccionar la menor arista que no haya sido visitada y comprobar que no se cree un ciclo entre los nodos si se utiliza.
- Repetir los pasos hasta haber utilizado $V-1$ (Vértices-1) aristas (Kershenbaum, A., & Van Slyke, R., 1972).

De esta manera, se obtiene el árbol de expansión mínima (Minimum Spanning Tree), un subconjunto de nodos y vértices del árbol original para recorrer todos con el menor peso posible. Es no direccionado y pesado (Sedgewick R., & Wayne, K., s.f.).

Su complejidad computacional está dada por $O(E \log E)$, donde E representa el número de aristas en el grafo. Esto es debido a que este algoritmo primero debe ordenar todas las aristas, y para cada una, realizar una búsqueda para verificar si su inclusión forma un ciclo (Cormen, T. H., et al., 2009).

De esta manera, nuestro algoritmo de Kruskal recibe como entrada la matriz de adyacencia que representa el grafo de todas las distancias de las colonias de la ciudad en kilómetros. Dentro del algoritmo, se obtienen todas las conexiones entre nodos del grafo, para ordenarlas (tal y como indica el algoritmo Kruskal) e ir las conectando de menor a mayor.

Branch and bound (Agente Viajero)

Problema: Mostrar la ruta a seguir por el personal que reparte correspondencia, considerando inicio y fin en la misma colonia.

El algoritmo del agente viajero (Traveling Salesman Problem - TSP) es un problema de optimización, donde se debe encontrar la ruta más corta que visita cada ciudad una sola vez y regresar a la ciudad de origen, conocido como un ciclo hamiltoniano. En este caso, utilizamos el algoritmo branch and bound (subrama de backtracking) para resolver el problema, permitiéndonos las posibles soluciones por niveles, podando “ramas” si no llevan a una mejor solución (Stützle, T., & Hoos, H., 2005).

En este caso, el algoritmo de branch and bound funciona dividiendo el problema en más pequeños, revisando si cada rama cumple con la solución planteada. Su funcionamiento es el siguiente:

- Para cada solución, revisa si el parámetro para cumplir con esta se cumple. Si la solución encontrada es peor, poda la rama; de lo contrario, continúa explorando (Morrison, D., et. al., 2016).

Su complejidad computacional, aplicada al problema del agente viajero y en el peor de los casos, está dada por $O(n!)$, donde n es el número de ciudades que se están visitando. Esto es debido a que el algoritmo puede llegar a ser capaz de tener que explorar todas las posibles combinaciones de las ciudades (Davendra, D., & Bialic-Davendra, M., 2020).

Dentro del código realizado, obtenemos la matriz de distancias que interconecta todas las colonias de la ciudad. Después de esto, llamamos a nuestra función de agente viajero, donde inicializamos las variables que vamos a utilizar. Finalmente, llamamos a una función recursiva dedicada a realizar el proceso branch and bound, explorando los vecinos de cada nivel. Una vez se cumpla el caso base (si llegamos al último nivel), revisamos si el costo total calculado es menor al mejor al que hayamos obtenido hasta ese punto. Si es el caso, actualizamos el nuevo mejor recorrido y costo.

Distancia Euclidiana (Pares Cortos)

Problema: Obtener la distancia al servidor más cercano (indicando la ubicación del mismo), a partir de la ubicación de una nueva casa.

Para obtener la distancia Euclidiana se utiliza la siguiente fórmula:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Donde:

x_1 = Coordenada x en el punto inicial

x_2 = Coordenada x en el punto de comparación

y_1 = Coordenada y en el punto inicial

y_2 = Coordenada y en el punto de comparación (EITCA, 2023).

Dentro del código, dado una nueva casa (Punto nuevo), calculamos la distancia euclidiana para saber cuáles son las centrales más cercanas. En concreto, se realizan las siguientes acciones:

- Recorre todas las coordenadas de las centrales y calcula la distancia euclidiana con la fórmula presentada arriba.
- Se mantiene el menor valor encontrado.
- Se crea un vector con las coordenadas del nuevo punto, la central y la distancia.
- Devuelve el o los vectores creados (vectores con distancia mínima).

Reflexión final

De acuerdo a la evidencia desarrollada, fuimos capaces de dar soluciones a las necesidades de una empresa que quiere incursionar en los servicios de internet de una ciudad, en diferentes colonias, a través de la implementación de diferentes algoritmos. Por un lado, Kruskal nos ayudó a implementar una forma de algoritmos voraces, siendo capaces de determinar la forma óptima para determinar la conexión entre todas las colonias. Por otro lado, branch and bound fue útil para solucionar el problema del agente viajero, mostrando la ruta a seguir por el personal que reparte correspondencia. Finalmente, la distancia euclidiana nos permitió obtener la distancia mínima de una nueva casa a alguno de los servidores existentes.

A través de la solución del problema e investigación de complejidades para cada uno de los algoritmos implementados, fuimos capaces de siempre tener en consideración la complejidad del código realizado. Así, podemos ser capaces de implementar algoritmos útiles, que se adapten a las necesidades del problema a resolver, así como implementarlos de manera eficiente.

Referencias

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms (Third Edition)*. MIT Press.
https://computo.fisimat.umich.mx/~htejeda/books/algoritmos/Introduction_to_algorithms_Third_Edition_Cormen_Leiserson_Rivest_Stein.pdf
- Davendra, D., & Bialic-Davendra, M. (2020, 19 de diciembre). *Introductory Chapter: Traveling Salesman Problem - An Overview*.
<https://www.intechopen.com/chapters/74003>
- EITCA. (2023, 7 de agosto). *¿Qué es la distancia euclidiana y por qué es importante en el aprendizaje automático?*
<https://es.eitca.org/artificial-intelligence/eitc-ai-mlp-machine-learning-with-python/programming-machine-learning/euclidean-distance/examination-review-euclidean-distance/what-is-euclidean-distance-and-why-is-it-important-in-machine-learning/>
- Kershenbaum, A. & Van Slyke, R. (1972). Computing minimum spanning trees efficiently. *ACM '72: Proceedings of the ACM annual conference*, 1(72), 518-527.
<https://doi.org/10.1145/800193.569966>
- Morrison, D., Jacobson, S., Sauppe J., Sewell, E (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization* 19, 79-102. <https://doi.org/10.1016/j.disopt.2016.01.005>
- Sedgewick R. & Wayne, K. (s.f.). *Algorithms, 4th Edition*. Pearson.
<https://algs4.cs.princeton.edu/43mst/>
- Stützle, T., & Hoos, H. (2005). *Stochastic Local Search, 1 - INTRODUCTION*.
<https://www.sciencedirect.com/topics/computer-science/traveling-salesman-problem>