INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS PUEBLA

Análisis y diseño de algoritmos avanzados (Gpo 602)

TC2038.602

E2. Actividad Integradora 2

Profesor:

Juan Manuel Ahuactzin Larios

Roberto Castro Gómez | A01425602

Santiago Rodríguez Gutiérrez | A01738097

Luis Antonio Salinas Gonzalez | A01735375

26 de Noviembre de 2025

Problema 1:

En este problema se solicitó modificar el código proporcionado con la implementación del algoritmo de Huffman para que lograra comprimir y descomprimir archivos de texto en tres idiomas: inglés, español y francés. La implementación debía incluir generación de estadísticas de frecuencia, el cálculo de bits promedio por símbolo, visualización del árbol, codificación en formato **.huff** y dos métodos de decodificación. Esto para analizar 10 archivos por idioma y comparar el desempeño de ambos decodificadores.

Para lograr esto se extendió el código base proporcionado. Primero se implementó una función que utilizaba un contador para calcular la frecuencia de aparición de cada carácter en el texto en tiempo O(m), donde m es la longitud total del texto. Para lograr construir el árbol de Huffman se creó una función que mediante un min-heap genera los códigos binarios óptimos y comprime el texto utilizando bitarray para un manejo eficiente a nivel de bits.

Para la decodificación se implementaron dos decodificadores, uno usando un diccionario para lograr búsquedas más rápidas con la forma {código: carácter} y el segundo navegando el árbol bit por bit logrando utilizar menos memoria pero más tiempo. En el caso del análisis se programó una función la cual recopila las métricas (tamaños, compresión factor, bits promedio) y otra para comparar los decodificadores ejecutando varias iteraciones y comparando los tiempos promedio.

Complejidad computacional:

Donde m es la longitud del texto (caracteres totales), n es el número de caracteres únicos, L es la longitud promedio de los códigos y h es la altura promedio del árbol:

- Construcción del árbol: O(n log n).
    - Creación del min-heap inicial O(n).
    - Más n-1 operaciones de extracción e inserción O(log n).
- Generación de códigos: O(n) - recorrido en preorden del árbol con 2n-1 nodos.
- Compresión del texto: O(m) - lectura lineal del texto con acceso O(1) al diccionario de códigos.
- Descompresión (diccionario): O(m*L).
- Descompresión (árbol): O(m*h).

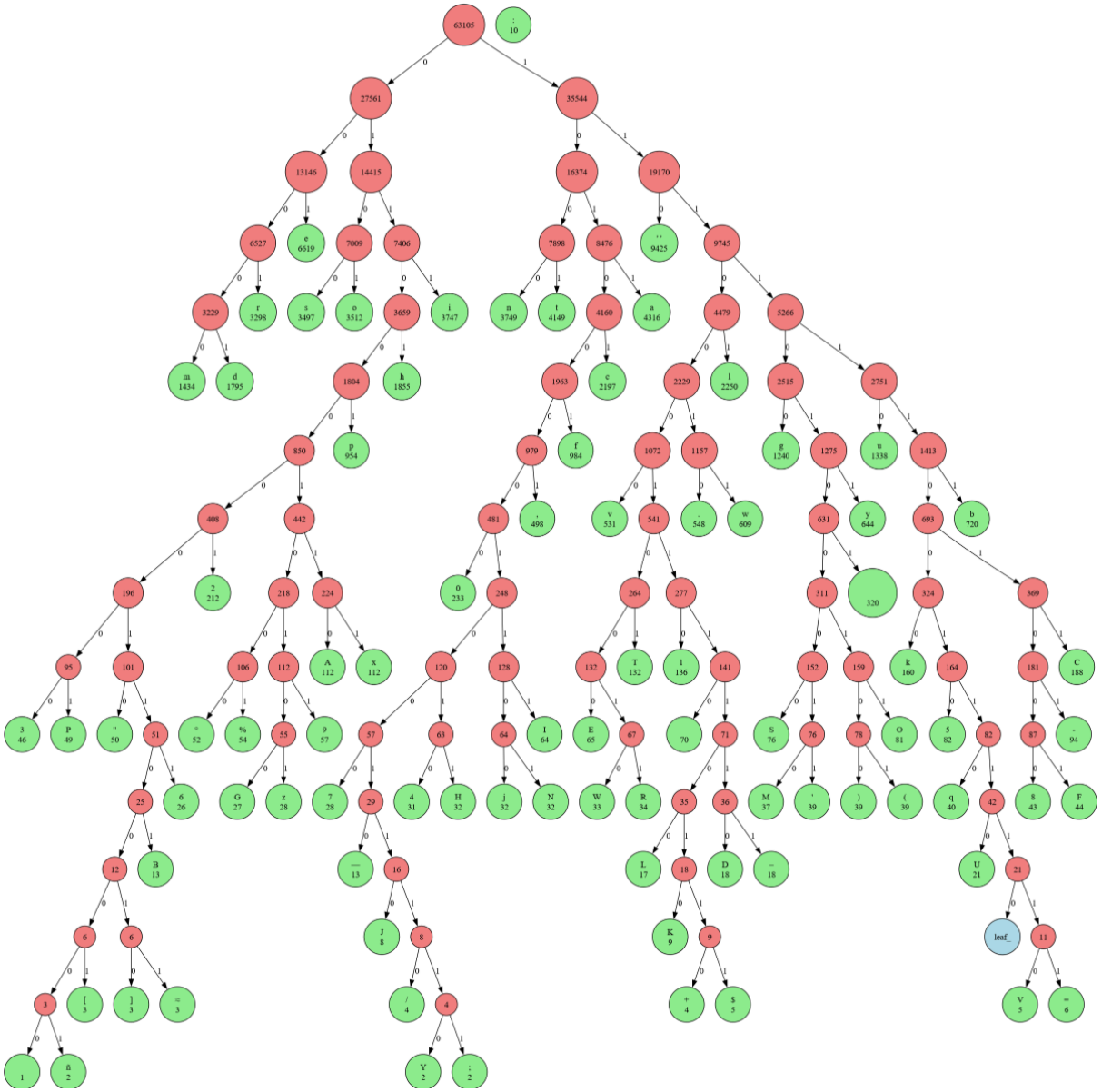


Imagen 1. Visualización del árbol generado

En la siguiente tabla generada por el mismo código, se pueden observar las estadísticas en cuanto a compresión de cada uno de los archivos, gracias a estos datos nos damos cuenta que el promedio de compresión es de 0.68, es decir, el archivo comprimido ocupa el 68% del tamaño del archivo original, mientras que el promedio de bits por símbolo es de 4.6.

filename	language	original_size	compressed	compression_factor	avg_bits_per_symbol	huff_path
es_wiki_Intel	es	53131	34351	0.6465340385085919	4.3992190369410995	./huff_out/es_wiki_Inteligencia_artificial.txt.huff
es_wiki_Segu	es	442311	257559	0.5823029497344628	4.538214800018581	./huff_out/es_wiki_Segunda_Guerra_Mundial.txt.huff
es_wiki_Mon	es	26615	19774	0.7429644937065565	4.4522080242358335	./huff_out/es_wiki_Mona_Lisa.txt.huff
es_wiki_Pyth	es	19674	16611	0.8443122903324184	4.5839957883653595	./huff_out/es_wiki_Python_(lenguaje_de_programaci%C3%B3n).txt.huff
es_wiki_Esta	es	90784	56250	0.6196025731406415	4.557381339827258	./huff_out/es_wiki_Estados_Unidos.txt.huff
es_wiki_Evol	es	152862	88069	0.5761340293859821	4.407684451869035	./huff_out/es_wiki_Evoluci%C3%B3n.txt.huff
es_wiki_Abra	es	34713	24711	0.7118658715754904	4.515404002965159	./huff_out/es_wiki_Abraham_Lincoln.txt.huff
es_wiki_Indi	es	64738	44047	0.6803886434551577	4.5282904400025545	./huff_out/es_wiki_India.txt.huff
es_wiki_Cam	es	76682	47115	0.6144205941420412	4.427462606117774	./huff_out/es_wiki_Cambio_clim%C3%A1tico.txt.huff
es_wiki_Alen	es	83018	53231	0.6411982943458046	4.575535190979983	./huff_out/es_wiki_Alemania.txt.huff
fr_wiki_Secoi	fr	169371	103714	0.6123480406917359	4.600810034129275	./huff_out/fr_wiki_Seconde_Guerre_mondiale.txt.huff
fr_wiki_Pytho	fr	35554	26651	0.7495921696574225	4.665336892434953	./huff_out/fr_wiki_Python_(langage).txt.huff
fr_wiki_Allen	fr	90544	66013	0.7290709489309065	4.680781871250948	./huff_out/fr_wiki_Allemagne.txt.huff
fr_wiki_%C3%	fr	103556	64835	0.6260863687280313	4.676624083178851	./huff_out/fr_wiki_%C3%89tats-Unis.txt.huff
fr_wiki_%C3%	fr	2570	5001	1.9459143968871595	4.541580471659081	./huff_out/fr_wiki_%C3%89volution.txt.huff
fr_wiki_La_Jo	fr	48415	33812	0.6983786016730352	4.615439615201283	./huff_out/fr_wiki_La_Joconde.txt.huff
fr_wiki_Chan	fr	84835	52166	0.6149112984027819	4.494142351900972	./huff_out/fr_wiki_Changement_climatique.txt.huff
fr_wiki_Inde.	fr	100791	68410	0.6787312359238424	4.607418345535465	./huff_out/fr_wiki_Inde.txt.huff
fr_wiki_Abral	fr	47943	33362	0.6958680099284567	4.604728328	./huff_out/fr_wiki_Abraham_Lincoln.txt.huff
fr_wiki_Intell	fr	105548	64932	0.6151892977602608	4.547002812176229	./huff_out/fr_wiki_Intelligence_artificielle.txt.huff
en_wiki_Clim	en	63619	40829	0.6417736839623382	4.491355677046193	./huff_out/en_wiki_Climate_change.txt.huff
en_wiki_Abra	en	72718	46811	0.6437333259990649	4.572451447923146	./huff_out/en_wiki_Abraham_Lincoln.txt.huff
en_wiki_Unit	en	84113	54034	0.6423977268674284	4.643228886256774	./huff_out/en_wiki_United_States.txt.huff
en_wiki_Mon	en	34498	25858	0.7495506985912227	4.600732601	./huff_out/en_wiki_Mona_Lisa.txt.huff
en_wiki_Indi	en	64975	44406	0.683432089	4.611096461975151	./huff_out/en_wiki_India.txt.huff
en_wiki_Wor	en	85517	54375	0.6358384882537975	4.597425821784042	./huff_out/en_wiki_World_War_II.txt.huff
en_wiki_Artif	en	87363	54911	0.6285383972619988	4.535727452878013	./huff_out/en_wiki_Artificial_intelligence.txt.huff
en_wiki_Pyth	en	21951	18132	0.8260215935492689	4.618641331919015	./huff_out/en_wiki_Python_(programming_language).txt.huff
en_wiki_Evol	en	67355	41866	0.6215722663499369	4.390226428049017	./huff_out/en_wiki_Evolution.txt.huff
en_wiki_Gerr	en	55680	38374	0.6891882183908046	4.680227297404901	./huff_out/en_wiki_Germany.txt.huff
				0.683432089	4.602730465	

Tabla 1. Estadísticas por archivo.

También contamos con una tabla generada con estadísticas por idioma, en la que se observa que el idioma con mayor factor de compresión es el español, seguido muy de cerca por el idioma inglés y con el francés en último lugar.

language	avg_compression_factor	avg_bits_per_symbol	files_count
es	0.6659723778327147	4.498539568132264	10
fr	0.7966090368583633	4.6033864805961455	10
en	0.6762046488490963	4.574111340596885	10

Tabla 2. Estadísticas por idioma.

Por último, nos damos cuenta que la decodificación es más rápida si usamos directamente el árbol de Huffman en lugar de el diccionario inverso, esto debido a que al decodificar con el árbol, la velocidad de acceso es mayor hacia el siguiente bit.

language	huff_file	decoder	runs	mean_seconds	stdev_seconds
es	es_wiki_Inteligencia_artificial.txt.huff	inverse	500	0.02208980994997546	0.0014242692622442993
es	es_wiki_Inteligencia_artificial.txt.huff	tree	500	0.010518007470003796	0.0003506593455199867
fr	fr_wiki_Seconde_Guerre_mondiale.txt.huff	inverse	500	0.07262100719595764	0.0017308568415582621
fr	fr_wiki_Seconde_Guerre_mondiale.txt.huff	tree	500	0.03511164456798724	0.0007719805348986377
en	en_wiki_Climate_change.txt.huff	inverse	500	0.026842964537972875	0.0007620545459739062
en	en_wiki_Climate_change.txt.huff	tree	500	0.013055433061970689	0.0004798579721874541

Tabla 3. Tiempos por método de decodificación.

Problema 2:

Para esta sección se solicitó implementar un sistema de planeación de rutas entre las diversas estaciones del metro de la ciudad de méxico. Para ello, se construyó un grafo ponderado y se implementó una interfaz para seleccionar las diversas estaciones de inicio y final.

Para la conexión entre las diversas estaciones se modeló el sistema como un grafo ponderado, donde los nodos representan las estaciones y las aristas el tiempo de traslado. Se implementó el algoritmo de Dijkstra para el cálculo del camino mínimo, modificando su estructura lógica para considerar no solo la distancia, sino también el costo operativo de los transbordos.

Al momento de explorar las conexiones, el algoritmo evalúa la línea de llegada versus la línea de salida. Si estas difieren, se aplica una penalización de 4 minutos al costo total acumulado. De esta forma, el sistema prioriza rutas que optimizan el tiempo real del usuario, prefiriendo en ocasiones caminos más largos en distancia pero con menos cambios de línea.

La integración final se realizó en un notebook de Google Colab utilizando la librería `ipywidgets`. Esto permitió generar una interfaz gráfica sencilla donde se seleccionan las estaciones de origen y destino mediante menús desplegables, mostrando el itinerario detallado y el tiempo total sin necesidad de manipular el código fuente.

Complejidad computacional:

La complejidad temporal del algoritmo utilizado es **O(E log V)**.

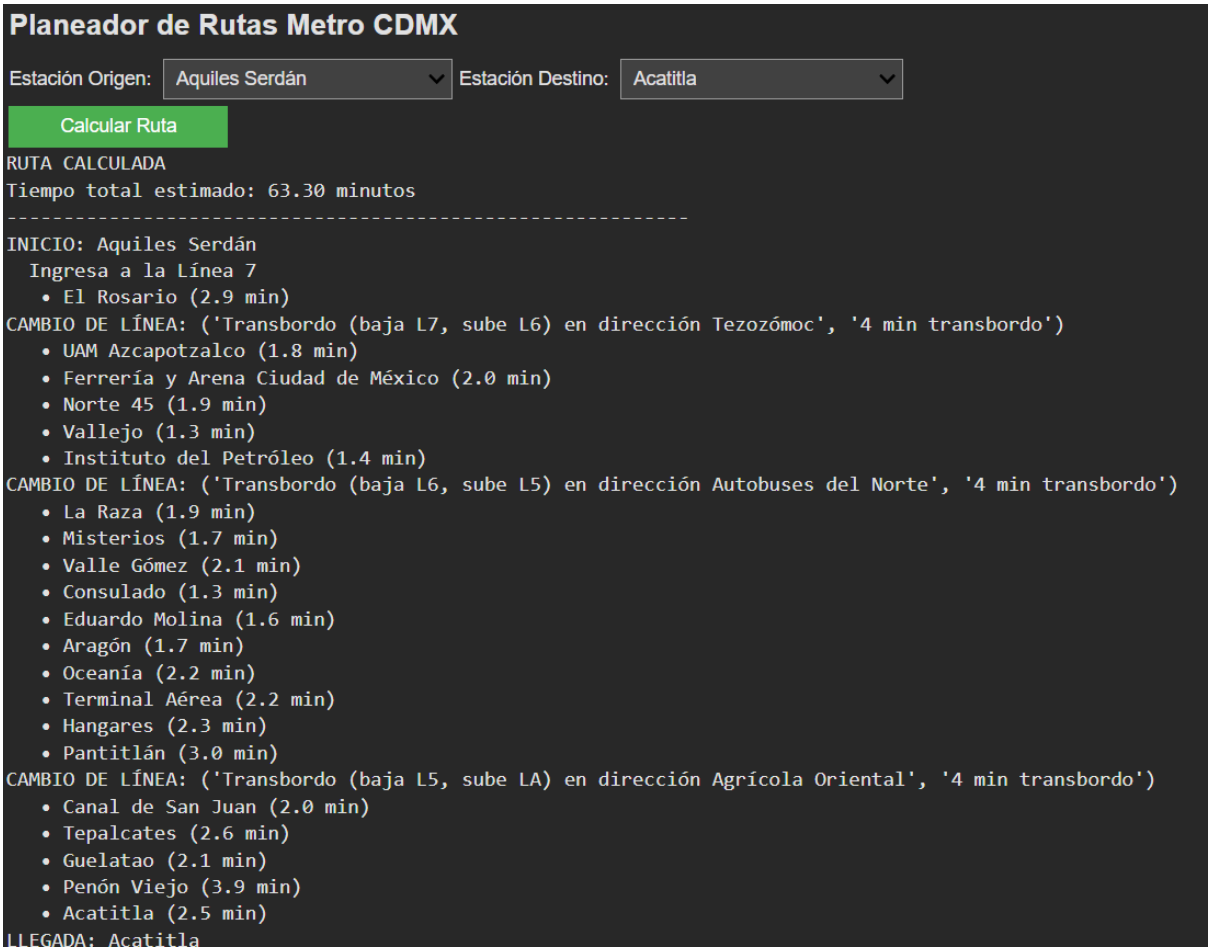
Dentro de esa complejidad, la E representa el número de conexiones entre estaciones y la V el número de estaciones del metro. Para la construcción de las rutas, se utiliza una cola de prioridad (Min-Heap) para seleccionar eficientemente el siguiente nodo con menor costo acumulado. Dado que cada arista se relaja a lo sumo una vez y las operaciones en el heap toman tiempo logarítmico, el rendimiento es altamente eficiente para grafos dispersos como la red del metro.

Por otra parte, la complejidad espacial termina siendo **O(V + E)**

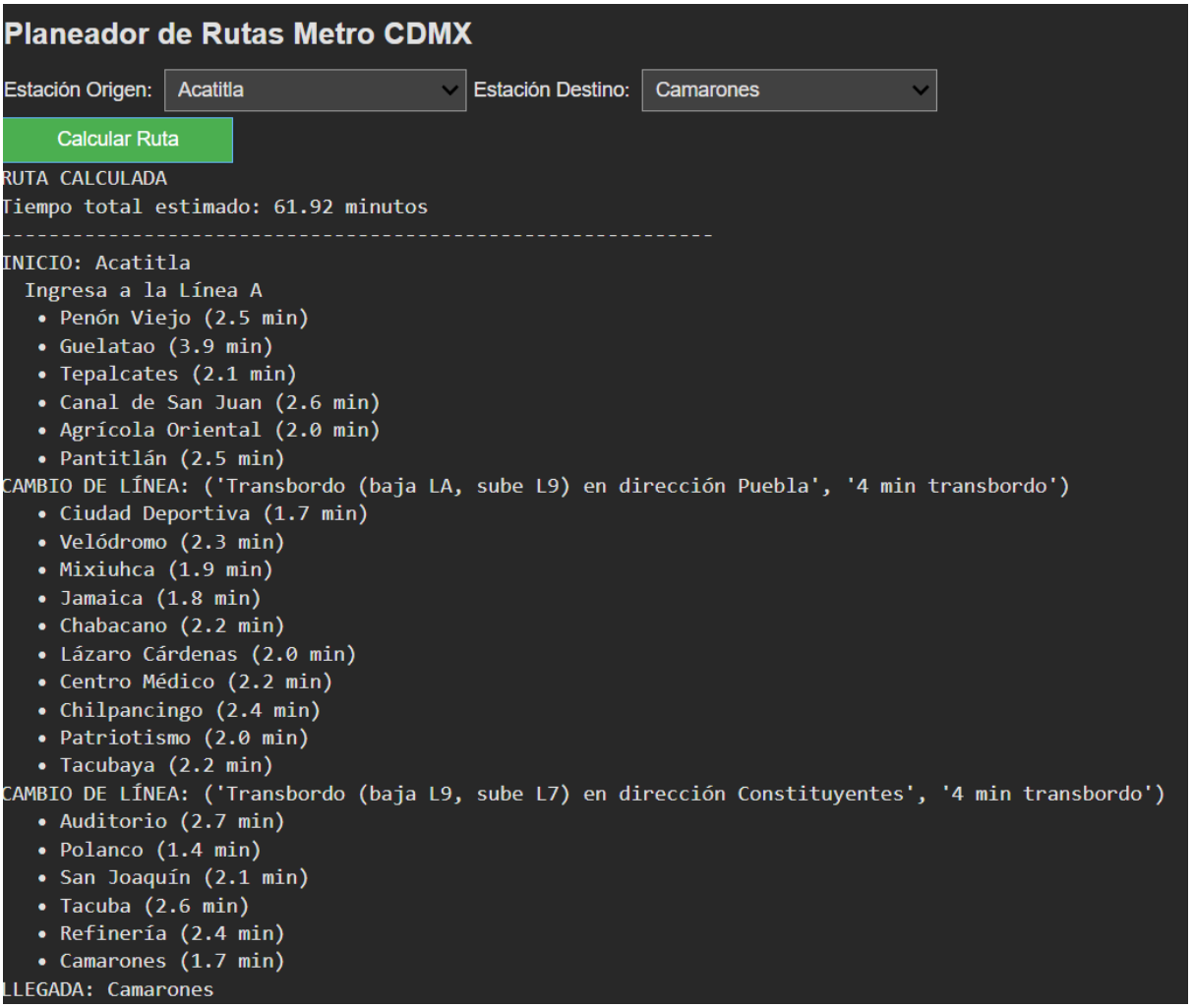
El espacio requerido crece linealmente con respecto al tamaño de la red, ya que es necesario almacenar la lista de adyacencia (grafo), el diccionario de costos mínimos (`min_costs`) y la cola de prioridad durante la ejecución.

Capturas de pantalla:

Repositorio de Github: https://github.com/Otrebor280971/ActividadIntegradora2



Repositorio de Github: https://github.com/Otrebor280971/ActividadIntegradora2



Anexos:

Repositorio de Github

Google colab - Ejercicio 2