Tarea 2

CC4102 - Diseño y Análisis de Algoritmos

Estudiantes: - Agustín Solís Meza
Profesor: Gonzalo Navarro
Profesor Auxiliar: Diego Salas

Fecha de entrega: 2 de Junio de 2024

Índice de Contenidos

Índice de Figuras

Índice de Tablas

1. Introducción

1.1. Presentación

En el presente informe se aborda la implementación y experimentación de dos variantes del algoritmo de Dijkstra para encontrar caminos más cortos en un grafo no dirigido con pesos positivos. Los métodos que se usarán y compararán son: Dijkstra con heap binario y Dijkstra con colas de Fibonacci. El objetivo principal es evidenciar las diferencias en rendimiento y eficiencia entre estos dos enfoques al aplicarlos a grafos de distintos tamaños y densidades, con el fin de determinar cuál de los dos es más eficiente en términos de tiempo de ejecución y uso de recursos.

1.2. Resumen

Este informe detalla el desarrollo y evaluación de dos variantes del algoritmo de Dijkstra: Dijkstra con heap binario y Dijkstra con colas de Fibonacci. Se exploran los algoritmos y las estructuras de datos utilizadas en cada variante, destacando sus enfoques y diferencias fundamentales, especialmente en la forma en que manejan las operaciones de inserción, extracción del mínimo y actualización de claves (`decreaseKey`). Además, se describen los datos utilizados en los experimentos, incluyendo los parámetros de generación de los grafos y la configuración del entorno de sistema. Los resultados de las pruebas revelan diferencias significativas en los tiempos de ejecución y eficiencia, con gráficos que ilustran estas comparaciones. Concluimos con una evaluación de los resultados frente a las hipótesis planteadas y sugerencias para futuras investigaciones que podrían mejorar la eficiencia y efectividad de estos métodos.

1.3. Hipótesis

Basado en un análisis preliminar de los métodos presentados y comparando sus principales diferencias, se propone la siguiente hipótesis:

El algoritmo de Dijkstra con colas de Fibonacci, debido a la eficiencia en la operación `decreaseKey`, será más efectivo en términos de tiempo de ejecución en comparación con el algoritmo de Dijkstra con heap binario, especialmente en grafos grandes. Esto se debe a que las colas de Fibonacci permiten realizar la operación `decreaseKey` en tiempo constante amortizado, mientras que en los heaps binarios esta operación toma tiempo logarítmico.

Por otro lado, el algoritmo de Dijkstra con heap binario, aunque menos eficiente en la operación `decreaseKey`, podría tener un rendimiento competitivo en grafos de tamaño pequeño a mediano debido a su simplicidad y menor sobrecarga en la gestión de la estructura de datos.

2. Desarrollo de los Algoritmos y Estructuras de Datos

2.1. Presentación de Algoritmos

2.1.1. Dijkstra con Heap Binario

El algoritmo de Dijkstra con heap binario utiliza una estructura de datos tipo heap para manejar la cola de prioridad. El proceso se realiza en los siguientes pasos:

1. **Inicialización**: Creamos dos arreglos, `distancias` y `previos`. `distancias` guarda la distancia más corta desde el nodo origen a cada nodo, y `previos` guarda el nodo anterior en el camino más corto.
2. **Insertar nodo origen**: Ponemos el nodo de inicio en el heap con una distancia de 0.
3. **Proceso principal**:
   a) Extraemos el nodo con la distancia más corta del heap.
   b) Para cada vecino del nodo extraído, calculamos una nueva distancia posible. Si esta nueva distancia es menor que la registrada en `distancias`, actualizamos `distancias` y `previos` y usamos `decreaseKey` para actualizar la posición del vecino en el heap.

2.1.2. Dijkstra con Cola de Fibonacci

El algoritmo de Dijkstra con colas de Fibonacci sigue los mismos pasos generales que el de heap binario, pero utiliza una cola de Fibonacci. Esta estructura de datos permite realizar la operación `decreaseKey` en tiempo constante amortizado, lo cual mejora la eficiencia del algoritmo en grafos grandes.

2.2. Estructuras de Datos

Se utilizaron estructuras de datos específicas diseñadas para facilitar la implementación eficiente del algoritmo de Dijkstra:

- **Heap Binario**: Implementado como un vector de pares (distancia, nodo). Cada vez que se inserta o extrae un elemento, se reorganiza el heap para mantener su propiedad.
- **Cola de Fibonacci**: Implementada con nodos enlazados. Permite inserciones y `decreaseKey` en tiempo constante amortizado y extracción del mínimo en tiempo logarítmico amortizado.

2.3. Diferencias entre los Algoritmos y sus Estructuras

2.3.1. Heap Binario

Descripción del Método: El algoritmo de Dijkstra con heap binario maneja la cola de prioridad utilizando un heap binario. Este enfoque es relativamente simple pero eficiente para grafos de tamaño pequeño a mediano.

Funciones Clave:

1. **insertar()**: Inserta un nodo en el heap binario, reorganizando la estructura para mantener la propiedad del heap.
2. **extraerMinimo()**: Extrae el nodo con la menor distancia del heap.
3. **decreaseKey()**: Actualiza la posición de un nodo en el heap cuando se encuentra una distancia menor.

2.3.2. Cola de Fibonacci

Descripción del Método: El algoritmo de Dijkstra con colas de Fibonacci utiliza una estructura de datos más avanzada que permite realizar la operación `decreaseKey` en tiempo constante amortizado. Este enfoque es más eficiente en grafos grandes.

Funciones Clave:

1. **insertar()**: Inserta un nodo en la cola de Fibonacci, manteniendo la estructura adecuada.
2. **extraerMinimo()**: Extrae el nodo con la menor distancia de la cola de Fibonacci.
3. **decreaseKey()**: Actualiza la distancia de un nodo en la cola de Fibonacci, reestructurando la cola si es necesario.

2.4. Operaciones Específicas

Además de las funciones clave, se implementaron las siguientes operaciones auxiliares:

1. **generarMultigrafo()**: Genera grafos aleatorios garantizando que sean conexos y no dirigidos.
2. **realizarExperimento()**: Realiza experimentos variando el número de nodos y aristas para comparar el rendimiento de ambas versiones del algoritmo de Dijkstra.

2.5. Complejidad de las Operaciones

Las diferencias en las operaciones clave pueden hacer que la cola de Fibonacci sea más eficiente en grafos grandes debido a la complejidad amortizada de sus operaciones:

- **Heap Binario**:
  - **Insertar**: O(log n).
  - **Extraer el mínimo**: O(log n).
  - `decreaseKey`: O(log n).
- **Cola de Fibonacci**:
  - **Insertar**: Tiempo constante.
  - **Extraer el mínimo**: O(log n) amortizado.
  - `decreaseKey`: Tiempo constante amortizado.

3. Especificaciones de los Datos

Esta sección tiene como objetivo proporcionar una base para la comprensión de los experimentos realizados, permitiendo una evaluación precisa de la efectividad y eficiencia de las variantes del algoritmo de Dijkstra.

A continuación, detallamos las características de los datos de prueba, los parámetros de configuración del experimento, y las condiciones bajo las cuales se realizaron las pruebas. Además, describimos el entorno del sistema donde se ejecutaron las pruebas, incluyendo detalles sobre el sistema operativo, la memoria RAM y la caché, los cuales pueden influir significativamente en el rendimiento de las operaciones de búsqueda y almacenamiento.

3.1. Datos Utilizados

- **Tipo de Datos**: Grafos no dirigidos con pesos positivos.
- **Distribución**: Pesos de las aristas distribuidos uniformemente entre 0.0 y 1.0.
- **Cantidad de Datos**: Pruebas realizadas con tamaños de grafos con v (número de nodos) variando entre 2^{10} y 2^{14}, y e (número de aristas) variando entre 2^{16} y 2^{22}.

3.2. Parámetros del Experimento

- **Número de Nodos (v)**: 2^i, con $i \in \{10, 12, 14\}$. Esto significa que tenemos grafos con 1024, 4096 y 16384 nodos.
- **Número de Aristas (e)**: 2^j, con $j \in [16...22]$. Esto significa que tenemos grafos con entre 65536 y 4194304 aristas.
- **Pesos de Aristas**: Los pesos de las aristas se generan aleatoriamente en el rango de 0 a 1.
- **Conectividad**: Nos aseguramos de que los grafos sean conexos (es decir, hay un camino entre cualquier par de nodos) y no dirigidos.

3.3. Configuración del Sistema

Para la ejecución de los experimentos relacionados con la implementación y evaluación de las variantes del algoritmo de Dijkstra, se utilizó un dispositivo que cuenta con las siguientes especificaciones:

- **Sistema Operativo**: Windows 11 Home Single Language, 64-bit.

- **RAM**: 16 GB (15.8 GB utilizable).
- **Procesador**: 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50 GHz.
- **Caché**:
  - **L1 Cache**: 320 KB.
  - **L2 Cache**: 5.0 MB.
  - **L3 Cache**: 8.0 MB.

3.4. Metodología de Pruebas

- **Repetición de Pruebas**: Cada configuración de experimento se realizó con 50 consultas diferentes para evaluar la variabilidad del rendimiento.
- **Consulta**: Se realizan consultas desde un nodo de origen aleatorio buscando el camino más corto a todos los demás nodos.
- **Método de Construcción y Consulta**: Utilización de los algoritmos de Dijkstra con heap binario y Dijkstra con cola de Fibonacci para la construcción y evaluación de consultas en el grafo.

4. Resultados y Conclusiones

4.1. Resultados Dijkstra con Heap Binario

i	j	Tiempo Total (s)	Tiempo Promedio (s)
10	16	0.239826	0.00479652
10	17	0.416203	0.00832406
10	18	0.785618	0.0157124
12	16	0.401354	0.00802707
12	17	0.612548	0.012251
12	18	1.12226	0.0224452
12	19	1.96584	0.0393169
12	20	3.3625	0.06725
12	21	6.33734	0.126747
12	22	12.3189	0.246378
14	16	0.750976	0.0150195
14	17	1.11696	0.0223392
14	18	1.69673	0.0339346
14	19	2.69965	0.053993
14	20	4.38841	0.0877682
14	21	7.65408	0.153082
14	22	14.5214	0.290428

Tabla 1: Resultados del algoritmo Dijkstra con Heap Binario para diferentes tamaños de $n = 2^i$ y $m = 2^j$

4.2. Resultados Dijkstra con Cola de Fibonacci

i	j	**Tiempo Total (s)**	**Tiempo Promedio (s)**
10	16	0.0045663	9.1326e-05
10	17	0.0084023	0.000168046
10	18	0.0121173	0.000242346
12	16	0.002905	5.81e-05
12	17	0.0189647	0.000379294
12	18	0.0048389	9.6778e-05
12	19	0.0090151	0.000180302
12	20	0.0152578	0.000305156
12	21	0.0313809	0.000627618
12	22	0.0500906	0.00100181
14	16	0.0027804	5.5608e-05
14	17	0.0037125	7.425e-05
14	18	0.0049096	9.8192e-05
14	19	0.0063997	0.000127994
14	20	0.0095456	0.000190912
14	21	0.0262736	0.000525472
14	22	0.0263369	0.000526738

Tabla 2: Resultados del algoritmo Dijkstra con Cola de Fibonacci para diferentes tamaños de $n = 2^i$ y $m = 2^j$

4.3. Gráficos De Comparación

A partir de los resultados, se procedió a la elaboración de una serie de gráficos para visualizar y comparar de manera efectiva los diferentes métodos estudiados.

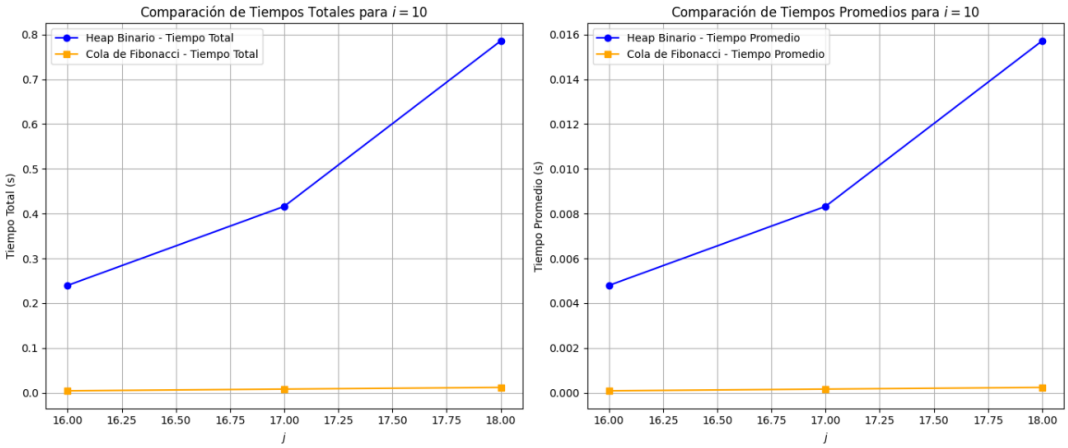

Figura 1: Comparación de Tiempos Totales y Promedios para $i = 10$

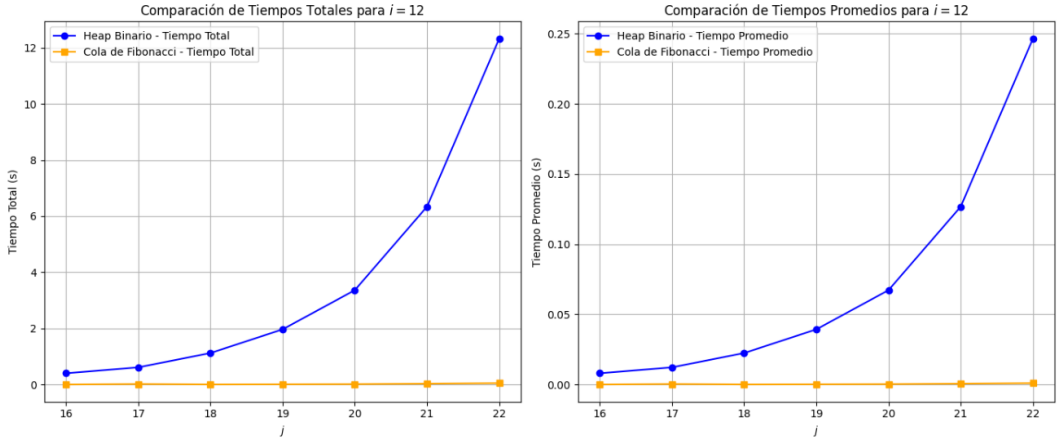

Figura 2: Comparación de Tiempos Totales y Promedios para $i = 12$

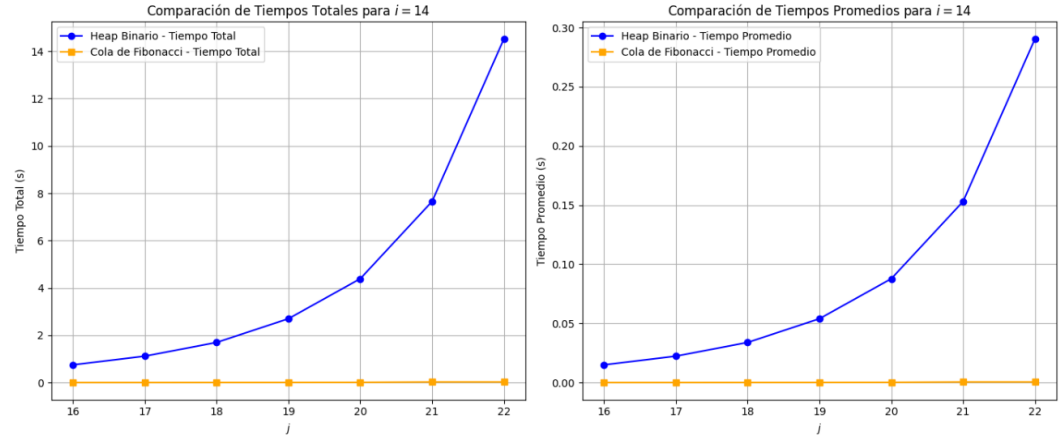

Figura 3: Comparación de Tiempos Totales y Promedios para $i = 14$

4.4. Observaciones Generales

Tiempo Total y Promedio: El tiempo necesario para ejecutar los algoritmos de Dijkstra usando Heap Binario y Cola de Fibonacci muestra una tendencia creciente a medida que aumenta el tamaño de n. Sin embargo, se observa que el tiempo total y promedio para Heap Binario es consistentemente más alto que para Cola de Fibonacci, indicando una mayor eficiencia en la Cola de Fibonacci, especialmente para tamaños grandes de n.

Comparación de Algoritmos:

- **Heap Binario:** A pesar de su mayor tiempo de ejecución, el algoritmo basado en Heap Binario muestra una escalabilidad lineal con respecto al incremento en j. Esto sugiere que aunque el tiempo de ejecución es mayor, la estructura del Heap Binario se mantiene relativamente eficiente con el aumento de complejidad.
- **Cola de Fibonacci:** Este método demuestra ser significativamente más eficiente en términos de tiempo de ejecución total y promedio, lo que sugiere que la estructura de la Cola de Fibonacci permite manejar las operaciones de disminución de clave y extracción de mínimo de manera más óptima.

Eficacia y Costo Computacional:

- **Eficiencia de Tiempo:** La Cola de Fibonacci es claramente más eficiente en términos de tiempo, haciendo que sea más adecuada para aplicaciones donde el tiempo de ejecución es crítico.
- **Eficiencia Operativa:** Aunque el Heap Binario es menos eficiente en términos de tiempo, su implementación y estructura pueden ser más simples y menos propensas a errores en comparación con la Cola de Fibonacci.

Aplicabilidad: La elección entre el uso de Heap Binario y Cola de Fibonacci puede depender del contexto específico de uso.

- Si la prioridad es la eficiencia en tiempo de ejecución, especialmente para operaciones de consulta rápidas y frecuentes, la Cola de Fibonacci sería preferible.
- Si la simplicidad y la robustez de la implementación son factores importantes, el Heap Binario podría ser la opción más adecuada.

En general, la Cola de Fibonacci se muestra superior en términos de eficiencia temporal, haciendo que sea una opción atractiva para situaciones donde el rendimiento es crítico. Sin embargo, la simplicidad y la estructura robusta del Heap Binario no deben ser descartadas, especialmente en aplicaciones donde la estabilidad y la simplicidad del código son cruciales.

5. Recapitulación

Recapitulando, el proceso en el que se basó este informe fue la programación de dos métodos para la implementación del algoritmo de Dijkstra: uno utilizando Heap Binario y otro utilizando Cola de Fibonacci. Ambos métodos fueron probados con las mismas cantidades de datos, midiendo el tiempo que tomaron en la ejecución del algoritmo para diferentes configuraciones. Posteriormente, se recopiló información de la ejecución, en cuanto a tiempo y recursos, y se llevó a cabo un análisis de estos. Finalmente, se concluye que el método basado en Cola de Fibonacci es superior en términos de eficiencia temporal.

5.1. Hipótesis planteada

Tras un análisis más detallado de los métodos presentados, se llega a la conclusión de que la hipótesis inicial no parece ser adecuada: A pesar de que se planteó que el método de Heap Binario sería más efectivo en la ejecución del algoritmo de Dijkstra y que la Cola de Fibonacci sobresaldría en las operaciones de búsqueda, las observaciones indican que el método de Cola de Fibonacci es superior en ambos campos. Se debe mencionar que lo anterior puede ser causado por las optimizaciones inherentes a la estructura de la Cola de Fibonacci que permite manejar las operaciones de manera más eficiente.

5.2. Mejoras a futuro

Se plantean como mejoras a futuro las siguientes:

- Optimizar la estructura de datos utilizada en el método de Heap Binario para mejorar su eficiencia temporal.
- Paralelizar funciones críticas dentro de ambos métodos puede generar mejor eficiencia.
- Implementar una caché para almacenar resultados intermedios y evitar cálculos redundantes, mejorando así la eficiencia global del algoritmo.

5.3. Dificultades encontradas

Primeramente, se debe mencionar como dificultad la inexperiencia tanto con las estructuras de datos avanzadas como con los algoritmos específicos de Dijkstra implementados con Heap Binario y Cola de Fibonacci, lo que provocó que se debiese dedicar una gran cantidad de tiempo en lectura y comprensión de los conceptos asociados.

Específicamente para el método de Heap Binario, un factor que influyó en el proceso fue la falta de experiencia con las estructuras ofrecidas por C++, por lo que su implementación fue cambiada muchas veces durante el desarrollo, llegando finalmente a la conclusión de que `vector` y `priority_queue` eran las más adecuadas de forma tardía y teniendo poco margen de tiempo para optimizarlo o analizar más detalladamente su correctitud y la precisión de los resultados entregados.

Finalmente, un factor que influyó fuertemente en el tiempo de desarrollo fue la dificultad para encontrar a alguien con quien hacer el trabajo, ya que no se pudo realizar con mi grupo anterior. Esta situación ocurrió a última hora, lo que me impidió encontrar compañeros a tiempo y, en consecuencia, terminé haciéndolo solo. Esta circunstancia incrementó la carga de trabajo y limitó el tiempo disponible para obtener todos los resultados necesarios dentro del plazo previsto.

5.4. Datos Faltantes

Debido a las dificultades anteriormente mencionadas, el presente informe no contiene datos de la experimentación de Heap Binario y Cola de Fibonacci para todas las configuraciones posibles de i y j. Se recomienda realizar pruebas adicionales para completar el conjunto de datos y obtener una visión más completa del rendimiento de ambos métodos.

5.5. Extensión

Se concluye que el presente informe podría ser extendido mejorando la eficiencia del método de Heap Binario y, además, dándole más tiempo a ambos métodos para su ejecución, hasta obtener la totalidad de resultados esperados. Adicionalmente, se sugiere investigar otras estructuras de datos que puedan ofrecer un rendimiento superior en la implementación del algoritmo de Dijkstra.