Algoritmia para problemas difíciles

Práctica 3

Álvaro Seral Gracia - 819425
Dorian Boleslaw Wozniak - 817570

Índice

Introducción

El objetivo de la práctica consiste en diseñar y poner a prueba un algoritmo probabilista capaz de simular los recorridos desde los almacenes hasta el punto de destino, estimando así el tiempo promedio de entrega. Además, se emplea el método de bootstrap para generar un intervalo de confianza del 90% sobre la media estimada, consiguiendo así un marco de certeza en la toma de decisiones.

Para el desarrollo de la práctica se han implementado dos programas distintos. El principal, en C++, contiene la implementación del algoritmo probabilista en cuestión. El segundo programa, en Python, está destinado a la generación de ficheros de pruebas que abarcan distintas configuraciones de carreteras e intersecciones, de modo que exploran la mayor cantidad de casos posibles, para luego analizar los resultados.

Detalles de implementación

El programa desarrollado implementa un algoritmo probabilista para simular el tiempo de entrega de paquetes desde dos almacenes diferentes hasta una casa, dadas las probabilidades de rutas entre intersecciones en la ciudad.

La ciudad se representa como un grafo donde las intersecciones son los vértices y las carreteras son las aristas. Para cada carretera entre intersecciones, se registra el tiempo de viaje y la probabilidad de tomar esa ruta. Dichas medidas se almacenan en dos vectores que hacen la función de matrices donde cada posición corresponde al tiempo o probabilidad respectiva de la carretera entre las intersecciones dadas por la fila y la columna.

El programa inicia tomando un archivo de entrada que contiene información sobre la ciudad, las intersecciones, las carreteras entre ellas, los tiempos de viaje y las probabilidades de elección de ruta. Se realizan comprobaciones para asegurarse de que los datos de entrada cumplen con ciertas restricciones, como que las probabilidades estén en el rango correcto, que las intersecciones estén dentro de los límites esperados, entre otros.

Posteriormente, se realiza una simulación para cada almacén (A y B) hasta la intersección de destino (la casa), seleccionando rutas aleatorias basadas en la distribución de probabilidades proporcionada, y calculando el tiempo estimado para llegar a la casa mediante dicho enfoque probabilístico.

Para la selección de rutas aleatorias se utiliza la función *uniform_real_distribution* en conjunto con el generador de números pseudoaleatorios *mt19937* para generar números en el intervalo [0.0, 1.0]. Se recorren todas las carreteras posibles desde la intersección actual. Para cada una de ellas, se acumula la probabilidad de tomar la ruta hacia esa dirección. Así, se compara el número aleatorio generado con la acumulación de probabilidades y se selecciona la carretera cuya probabilidad acumulada sea mayor o igual a dicho número.

Tras realizar dichas simulaciones, se utiliza el método de bootstrap para calcular intervalos de confianza sobre los tiempos estimados de entrega. Realiza múltiples simulaciones y calcula las medias de los tiempos, proporcionando un intervalo de confianza del 90%

Complejidad temporal del programa

Al crear el grafo con un número de vértices y aristas, la inicialización de cada uno de los dos vectores tiene una complejidad de $O(V^2)$, donde V es el número de vértices. Así, la complejidad de la operación $O(V^2) + O(V^2)$, que simplificandolo daría $O(V^2)$.

La función *casa_es_alcanzable* utiliza una búsqueda en profundidad para verificar si una casa es alcanzable desde un almacén. En los peores casos, donde deberá recorrer todos los vértices del grafo, tendrá una complejidad temporal de $O(V^2)$.

La función *simular*, que simula el tiempo de recorrido entre dos nodos siguiendo un camino aleatorio en el grafo, no está limitada por una cantidad específica de iteraciones. Sin embargo, debido a las características del problema, se puede estimar que en los peores casos recorrerá todos los vértices del grafo. Por tanto, su complejidad temporal sería de $O(V^2)$.

Finalmente, la función *bootstrap*, encargada de realizar un muestreo con reemplazo sobre los tiempos de simulación, depende principalmente tanto del tamaño de vector de tiempos (V^2) como del número de iteraciones (N). Además, al final de la función utiliza el método de ordenación *introsort* para ordenar el vector de medias de tiempos generado, cuyo tamaño es el número de iteraciones (N). Con todo esto, la complejidad temporal de esta función en el peor caso es $O(V^2 * N + N \log N)$, que simplificando es $O(V^2 * N)$.

Teniendo en cuenta todas las complejidades temporales calculadas, la complejidad total del programa equivaldría a la suma de estas, que daría como resultado una complejidad polinómica de $O(V^2 * N)$, donde V es el número de vértices del grafo, y N es el número de iteraciones para el algoritmo *bootstrap*.

Pruebas

Para la realización de pruebas, se ha creado un programa que genera aleatoriamente grafos modificando el número de vértices y aristas que contiene. Adicionalmente, el programa implementado obtiene el tiempo medio de las simulaciones de ambos almacenes.

En caso de no encontrar un camino de la intersección origen (almacén) a la intersección destino (casa), se avisa. Esto suele ocurrir cuando hay más vértices que aristas, siendo el mínimo de aristas para que una entrada sea válida la mitad de aristas que de nodos. Pero en este caso se tendría que utilizar todas las aristas menos una para asegurar que cada vértice tiene al menos una arista. Esta limitación se ha tenido en cuenta, y se ha controlado el número de errores generados.

Los tests realizados han consistido en:

- Fijar el número de aristas a 100, y probar con vértices desde 25 a 200 en pasos de 25.
- Fijar el número de vértices a 100, y probar con aristas desde 50 a 300 en pasos de 25.

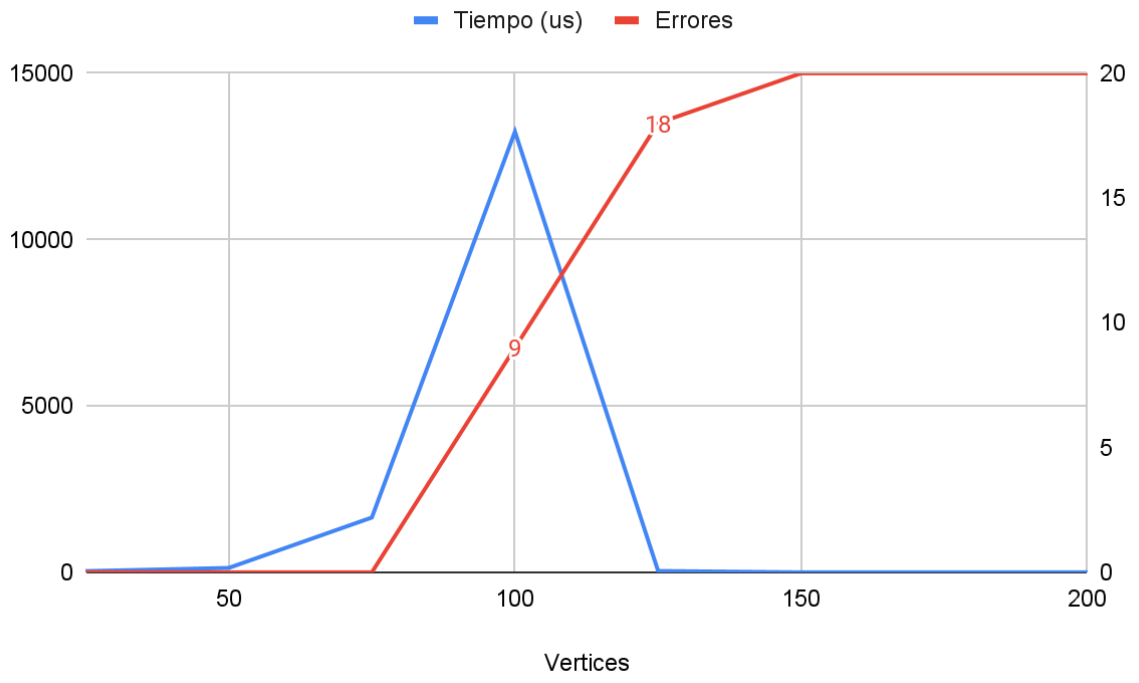

Fig. 1: Evolución del tiempo medio para simular según los vértices de un grafo

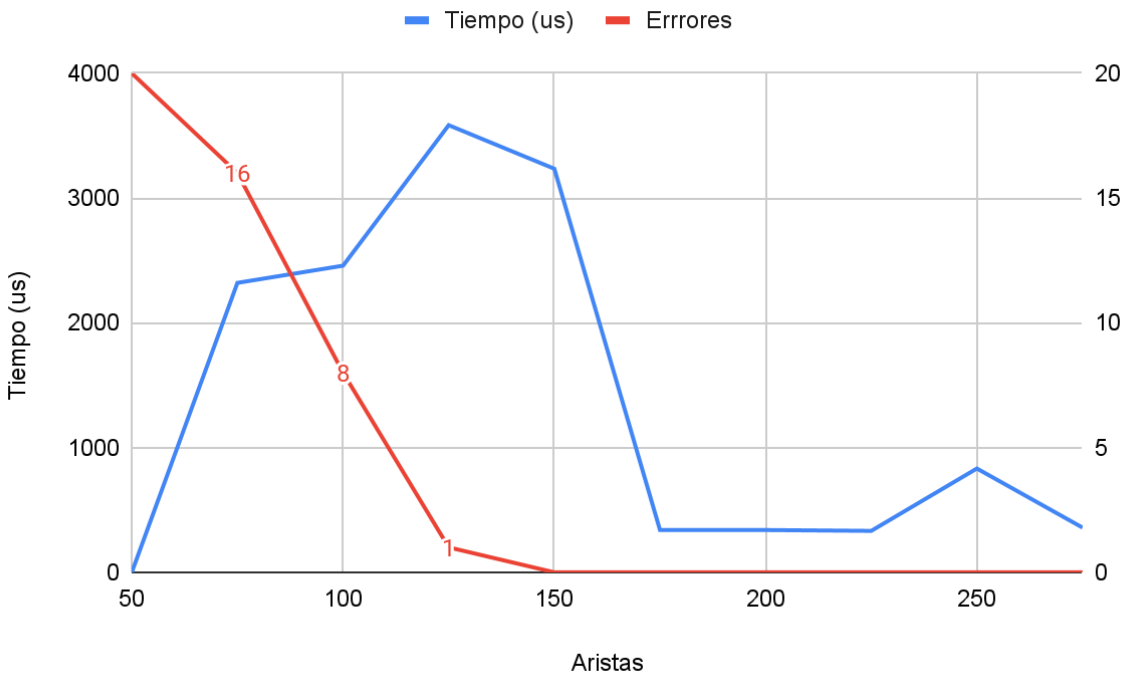

Fig. 2: Evolución de tiempo medio para simular según el número de aristas

Se puede observar en las gráficas que los casos más complicados para resolver una ruta es cuando el número de vértices y el número de aristas son similares. Para la simulación, es ventajoso que los grafos sean densos (es decir, que tengan muchas aristas), pues facilita encontrar rápidamente una ruta válida. En cambio, cuando el grafo es poco denso, no solo es más improbable de que exista un camino, sino que además la simulación puede ir yendo frecuentemente de un vértice a otro sin avanzar realmente.

Distribución del trabajo

	Álvaro	Dorian
Implementación del programa	6h	7h
Realización de pruebas y análisis	4h	7h
Documentación	3h	3h
Total	13h	18h

Bibliografía

https://en.cppreference.com/w/cpp/algorithm/sort