UNIVERSIDADE DA CORUÑA

Diseño y Análisis de Algoritmos: Práctica 3 (P3)

Árboles expandidos mínimos

Uxío Merino
uxio.merino@udc.es

Iván Castro
ivan.castro1@udc.es

Mario Chan
mario.chan@udc.es

1 Introducción

En esta práctica se ha implementado el algoritmo de Kruskal en Python, encargado de computar un árbol expandido mínimo a partir de un grafo completo. El algoritmo ha sido validado con una función de test, a partir de 2 grafos de prueba sobre los que se conoce de antemano la solución, y posteriormente se han hecho la medición de tiempos de ejecución y cálculo empírico de sus complejidades.

1.1 Contexto

1.1.1 ¿Qué se está midiendo?

Las mediciones se han realizado sobre el algoritmo de Kruskal, que recibe un grafo completo y se encarga de obtener su árbol expandido mínimo. No hay un análisis basado en casos, pues solo está el caso medio.

1.1.2 ¿Dónde se está midiendo?

Dichas mediciones se han llevado a cabo sin la ejecución de otros programas en paralelo, sobre un ordenador con las siguientes características:

- Sistema operativo: Windows 10 20H2 x64
- Procesador: Intel(R) Core(TM) i7-8565U CPU @ 1.80 *GHz* 1.99 *GHz*
- Memoria RAM: 16.00 *GB*
- Entorno: Spyder (Python 3.8)

1.1.3 Unidades de tiempo

Los tiempos se han medido en *µs* usando la función de Python *perfcounter*, que devuelve el tiempo en segundos, multiplicada por 10e6.

2 Medición de tiempos y análisis de la complejidad

A continuación se presentan los resultados obtenidos para el algoritmo de estudio (tiempos de ejecución y cota ajustada, ligeramente subestimada y sobre-estimada). La medición de tiempos pequeños está automatizada en el código, basándose en un umbral de confianza por defecto de 1000 μs.

2.1 Algoritmo de Kruskal

Se han utilizado valores de *n* de una progresión geométrica de razón 2, que toma los valores [20, 40, 80, 160, 320, 640, 1280, 2560). No se han detectado valores anómalos. A continuación se muestra el análisis:

n	t(n)	t(n) / f(n)	t(n) / g(n)	t(n) / h(n)
20	382.915*	4.2811266	0.2140563	0.0107028
40	1353.500	5.3501785	0.1337545	0.0033439
80	4886.800	6.8295106	0.0853689	0.0010671
160	31094.899	15.3641731	0.0960261	0.0006002
320	108875.700	19.0198021	0.0594369	0.0001857
640	878939.400	54.2861412	0.0848221	0.0001325
1280	5851628.900	127.7796875	0.0998279	0.0000780
2560	48362193.700	373.3756952	0.1458499	0.0000570
		Cota subestimada	**Cota ajustada** Cte ∈ [0.05, 0.21]	**Cota sobre-estimada**

*: tiempo promedio de K = 1000 ejecuciones del algoritmo
Tabla 1: estudio de la complejidad del algoritmo de Kruskal, caso medio $<n^{1.5}, n^{2.5}, n^{3.5}>$

Se han repetido varias veces las mediciones, siendo esta la mejor serie. Se acompañan de una cota ligeramente subestimada ($f(n) = n^{1.5}$), una cota ajustada ($g(n) = n^{2.5}$) y una cota ligeramente sobre-estimada ($h(n) = n^{3.5}$). La cota ligeramente subestimada *f(n)* crece asintóticamente más despacio que el tiempo de ejecución *t(n)*, resultando en una tendencia de *t(n) / f(n)* a ∞ (diverge). La cota ajustada *g(n)* representa la tasa de crecimiento de *t(n)*, por lo que tiende a una cte, y la cota sobre-estimada crece asintóticamente más rápido, llevando a *t(n) / h(n)* a converger a 0.

3 Conclusiones

Una vez obtenidos los resultados y medidas las complejidades a partir de los tiempos de ejecución, podemos concluir que la complejidad del algoritmo de Kruskal es $O(n^{2.5})$.

Hay que destacar que la complejidad del algoritmo se está midiendo en función del número de nodos *n*, y no del número de aristas, por lo que no se puede hacer una comparación directa con el algoritmo de Prim. En cualquier caso, este último será más eficiente que el algoritmo de Kruskal a medida que aumentemos el número de aristas del grafo.