COMPARATIVA DE LA EFICIENCIA EN TIEMPO DE LOS ALGORITMOS BFS y DFS

Q1 2021-2022

B7 Probabilidad y Estadística

Isha Shafqat

Hugo Pelayo

Sergio Sanz

Grupo 22

0. Resumen

- **Objetivo:** el objetivo de esta práctica es comparar el tiempo que tardan dos algoritmos de búsqueda en grafos, el DFS y el BFS, en encontrar un nodo determinado.

- **Metodología:** para recoger los grafos
  //lo que hacía era dependiendo de la hora actual de mi pc me generaba //una permutación random de n naturales

- **Resultados:**
- **Conclusión:** para los grafos utilizados donde el número de aristas es #nodos(#nodos-1)/4 hemos visto que el BFS es mejor que el DFS, aún así el estudio se ve limitado por el pequeño tamaño de la muestra y a las variables que no hemos tenido en cuenta.

1. Introducción

A la hora de programar tareas, la eficiencia en tiempo es casi siempre un factor a tener en cuenta. Consideramos importante tener conocimientos de variedad de algoritmos para, de esta manera, saber cuál se adapta mejor al problema que sea que estemos afrontando.

En este trabajo estaremos trabajando con grafos. Un grafo es un tipo de estructura de datos que permite representar conexiones entre diferentes elementos (lo llamamos nodos) y las conexiones entre estos nodos las llamamos aristas, también se usa el término “arcos”. Un ejemplo sería la conexión entre ordenadores por internet o las diferentes ciudades de un país.

El objetivo principal de este trabajo es comparar la eficiencia en tiempo de dos algoritmos populares que se usan para recorrer grafos. Estos son el algoritmo de búsqueda en profundidad o DFS (Depth First Search) y el algoritmo de búsqueda en anchura o BFS (Branch First Search).

Lo que queremos saber con más precisión es el tiempo que tardan los dos algoritmos en, partiendo de un nodo ***K,*** visitar un nodo ***P***.

2. Variables recogidas

Nuestra variable de interés ***Y*** es el tiempo de ejecución del algoritmo en segundos. La variable de interés ***X*** es el algoritmo utilizado para el recorrido de grafos (DFS o BFS). Por otra parte incluimos una variable ***Z*** que representa el orden de los grafos (número de nodos que contienen).

3. Metodología: obtención de los datos

Para la recogida de datos hemos generado un total de 40 grafos dirigidos, todos con grado diferente y con vértices indexados de 1 hasta n (donde n es el grado de dicho grafo). Las aristas de los grafos se han generado de manera aleatoria usando una herramienta online (random permutation) que se aprovecha del clock local (de nuestro sistema) para generar una *seed* que utiliza para obtener una permutación aleatoria de los vértices de un grafo cogidos de 2 en 2 (que serian los arcos/aristas). Posteriormente, para cada grafo, hemos escogido dos vértices al azar de su conjunto de vértices, uno de partida (`start_point`), otro de destino (`end_point`) y hemos tomado nota del tiempo en microsegundos (μs) del tiempo de ejecución que toman ambos algoritmos (BFS y DFS) en llegar al vértice destino.

El ordenador utilizado para obtener las medidas tiene las siguientes características:

- Intel Core i5 7300HQ a 2.5Ghz,
- 8GB de memoria RAM
- Arquitectura 64 bits

En cuanto al apartado del software, se ha utilizado el entorno de desarrollo (IDE) Visual Studio Code para escribir el código de los archivos fuente para nuestros tests. Se ha separado el programa en tres módulos principales. El primero que

contiene la definición del objeto Grafo, y otros dos: uno definiendo los procedimientos de BFS y DFS y otro con el programa principal. Para obtener el tiempo de ejecución hemos aprovechado las utilidades de la librería estándar de C++, la librería <chrono> en concreto. Esta librería viene con una utilidad que nos permite establecer una marca de tiempo de ejecución en un punto en concreto de nuestro programa. El tiempo de ejecución se ha obtenido haciendo la diferencia del tiempo pasado entre dos marcas situadas una antes de la llamada a nuestro procedimiento y otra justo después de la llamada (*Ver Figura 0*). El sistema operativo usado ha sido Pop!_OS 21.04.

```
bfs_start = std::chrono::high_resolution_clock::now();
bool case1 = bfs_find(G, _ini, _end);
bfs_end = std::chrono::high_resolution_clock::now();

dfs_start = std::chrono::high_resolution_clock::now();
bool case11 = dfs_find(G, _ini, _end);
dfs_end = std::chrono::high_resolution_clock::now();
```

Figura 0: marcas de tiempo en los extremos de nuestros procedimientos

Los datos a leer para nuestro programa se han escrito en un fichero con nombre “input”, con el formato siguiente:

```
número_de_nodos       # número de nodos de nuestro grafo

arista 1              # un arco o arista

arista 2              # un arco o arista

arista 3              # un arco o arista

arista 4              # un arco o arista

. . .

start_point end_point # vértice inicio y vértice fin
```

A la hora de la ejecución se han ejecutado los programas en la carpeta raíz de los ejecutables leyendo de nuestro fichero “input”.

```
-----|PROGRAMA DE TEST EMPIEZA AQUÍ|-----

Graph with degree: 15 node(s) and density: 0.07619

DFS: could reach target: 15, from start point: 2

finished computation at Mon Dec 15 14:42:37 2021

elapsed time program: 227.42700 µs

elapsed time dfs_find(): 6.11800 µs
```

Ejemplo resultado de ejecución de búsqueda con DFS

Cabe destacar que los tiempos de ejecución han sido muy variados entre ejecuciones, por esto se ha decidido hacer 3 ejecuciones de cada algoritmo y escoger el valor más grande.

El número de nodos y las aristas de los diferentes grafos han sido obtenidos a través de las siguientes páginas web:

- http://www.jerrydallal.com/random/random_permutation.htm

  Esta página usa como semilla la hora actual de nuestro sistema para crear una permutación aleatoria de un intervalo de valores cogidos en una cierta cantidad.

- https://csacademy.com/app/graph_editor/

  La hemos utilizado para visualizar los grafos.

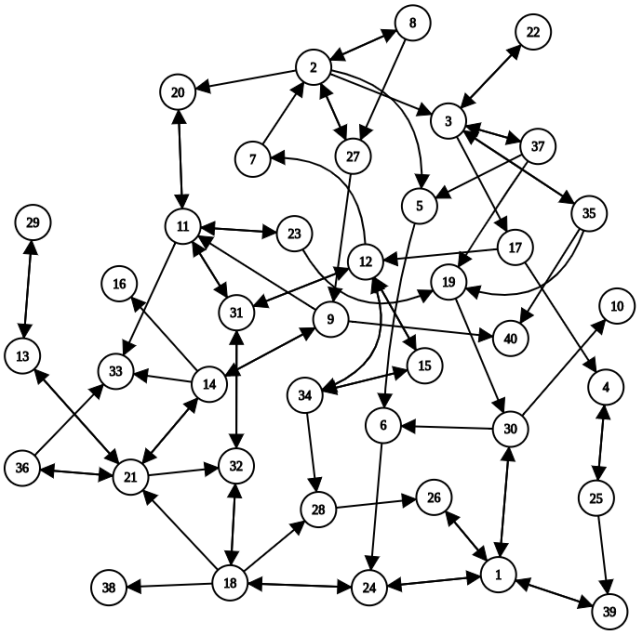


Figura 1: ejemplo del grafo 10

La implementación utilizada para representar los grafos, ejecutar los algoritmos y obtener el tiempo que han tardado en encontrar un nodo determinado se encuentran en los documentos adjuntos, aquí un ejemplo de la implementación del BFS:

```
bool bfs_find(const Graph& _graph, vertex start_point, vertex
target)
{
    queue<vertex> next_to_visit;
    int LIMIT = _graph.num_vertex; //número total de vértices
    //vector de vértices visitados
    vector<bool> visited_vertices(LIMIT, false);
    next_to_visit.push(start_point);
    visited_vertices[start_point-1] = true;
    bool reached = false;

    while (not next_to_visit.empty() and not reached) {
        //se trata el primero en la cola
        vertex _next = next_to_visit.front();
```

```
            next_to_visit.pop();
            if (not reached) {
                //explora todos los adyacentes a _next
                for (int current : _graph.adjList[_next-1]) {
                    if (not visited_vertices[current-1]) {
                        next_to_visit.push(current);
                        visited_vertices[current-1] = true;
                        if (current == target) reached = true;
                    }
                }
            }
        }
        return reached;
    }
```

Figura 2: Implementación del algoritmo BFS

Para cada grafo se ha ejecutado los dos algoritmos de búsqueda, por tanto tenemos una muestra de datos pareados.

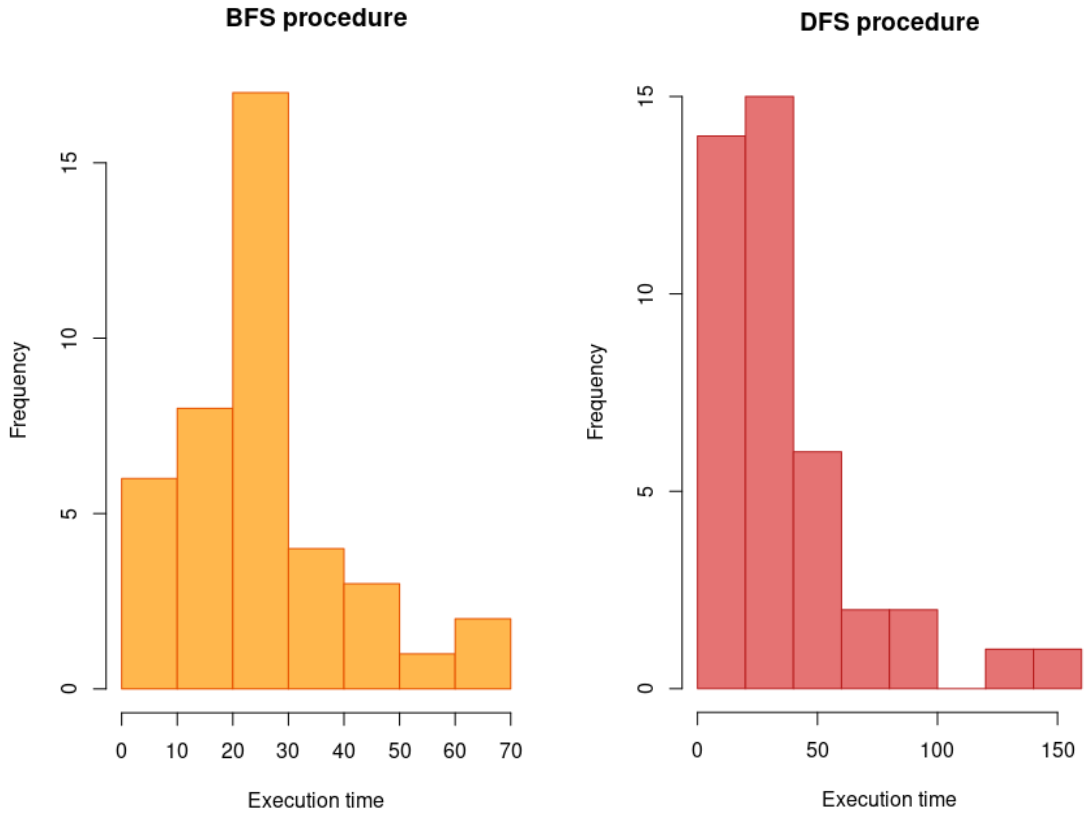
4.Metodología: tratamiento de los datos

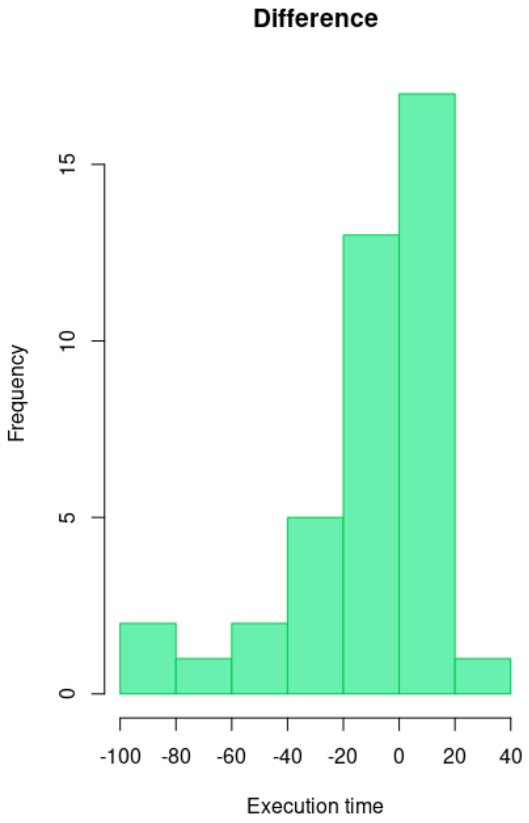
En este estudio vamos a comparar los tiempos de ejecución de dos algoritmos de búsqueda sobre grafos, BFS y DFS. En este caso vamos a comparar dos muestras recogidas a partir de aplicaciones que se ha hecho con estos algoritmos durante el estudio.

Entonces partimos de una hipótesis nula de que los dos algoritmos nos ofrecen tiempos de respuesta similares, es decir, como se ha explicado previamente en este estudio, en este apartado vamos a asumir que estos algoritmos en promedio recorren grafos en tiempos similares (esto sería nuestra hipótesis nula). Para contrastar, nuestra hipótesis alternativa indicaría que los algoritmos recorren grafos

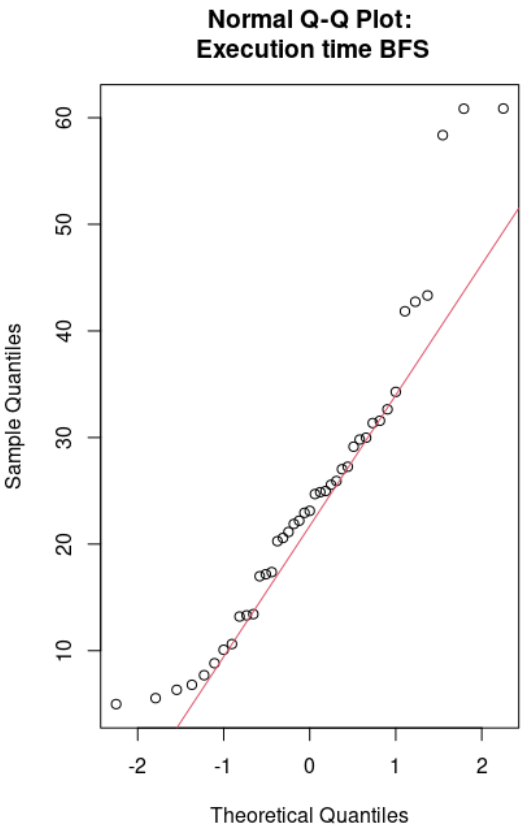
en promedio en tiempos diferentes, quiere decir que uno puede ser más rápido que otro (hipótesis alternativa bilateral, no decimos con certeza cual es más rápido).

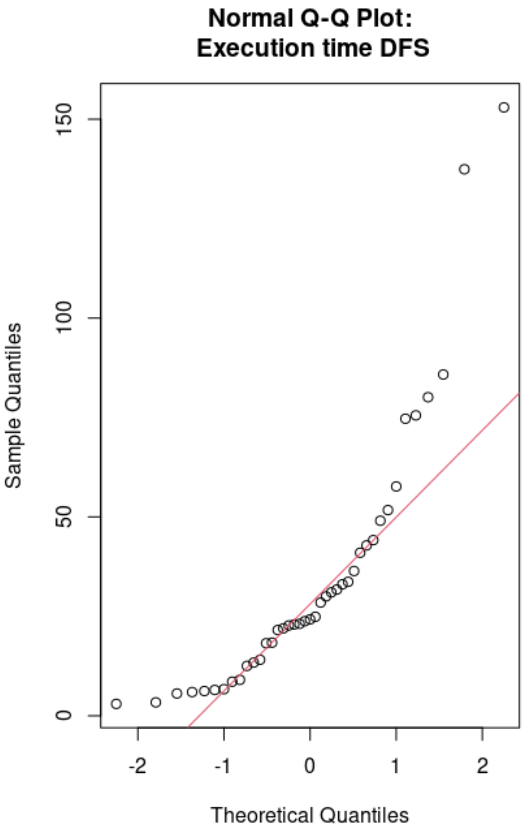
Para saber la distribución que se adecúa a nuestro estudio vamos a comprobar las premisas: vemos que en este casos estamos tratando con dos muestras aleatorias apareadas.





En este caso, para hacer el estudio, se asume que la diferencia de estos sigue una distribución normal. A continuación se muestran los gráficos Normal Q-Q Plot sobre las dos muestras recogidas.


Normal Q-Q Plot:
Execution time BFS
Sample Quantiles
Theoretical Quantiles
10
20
30
40
50
60
-2
-1
0
1
2


Normal Q-Q Plot:
Execution time DFS
Sample Quantiles
Theoretical Quantiles
0
50
100
150
-2
-1
0
1
2

Cálculos:

Estadístico: $\hat{t} = \dfrac{\bar{D} - \mu_D}{S_D \cdot \sqrt{\frac{1}{n_D}}}$

- $\bar{D}$ → diferencia mediana muestral
- $\mu_D = 0$ por hipótesis nula ya que $\mu_A = \mu_B$
- S_D desviación estandard diferencia muestral
- n_D tamaño de las muestras.

$\bar{D}$ se consigue por R con la función mean()

μ_D vale 0

S_D se consigue por R con la función sd()

Tenemos $\hat{t} = \dfrac{-10'30873 - 0}{28'088 \cdot \sqrt{\frac{1}{40}}} = -2'3211$

→ Calculemos el punto crítico suponiendo riesgo de 5%. Tenemos entonces $t_{0'975,\,39} = 2'022691$

5.Análisis estadístico

I. Premisas

- Normalidad para cada una de las variables
- Muestras aleatorias e independientes
- Homocedasticidad (varianza constante entre los dos algoritmos)

II. Prueba de hipótesis (bilateral)

$H_0: \mu_B = \mu_D \quad \rightarrow \quad H_0: \mu_D - \mu_B = 0 \quad \rightarrow \quad H_0: \mu_X = 0$

$H_1: \mu_B \neq \mu_D$

III. Cálculo del estadístico

Definimos B como el conjunto de tiempo de ejecución que tarda el algoritmo BFS en encontrar un nodo P partiendo de un nodo K. Y D como el conjunto de tiempo de ejecución que tarda el algoritmo DFS en encontrar un nodo P partiendo de un nodo K.

Vamos a comparar dos algoritmos diferentes, un diseño muy simple e intuitivo que podríamos utilizar es el diseño de datos emparejados. Entonces como el diseño es aplicable, podemos aplicar un método muy sencillo para el análisis estadístico. En efecto, los datos que obtuvimos podemos encontrar el incremento observado,

Xi = Di - Bi, por lo que el problema original ha quedado reducido a un problema sobre una única muestra. Por tanto, un diseño de muestras pareadas se resuelve como un problema de inferencia con un solo parámetro.

Tenemos que recordar que la hipótesis de normalidad se aplica únicamente a X, y no es necesario que B y D se distribuyan como una normal.

$H_0: \mu_D - \mu_B = 0 \quad \rightarrow \quad H_0: \mu_X = 0$

$$\hat{t} = \frac{\bar{X} - \mu_X}{S_X/\sqrt{n}} = (\text{si } \mu_X = 0) = \frac{\bar{X}}{S_X/\sqrt{n}} \sim t_{n-1}$$

$\bar{X}$ será la media de la variable diferencia

$\mathbf{S_X}$ será la desviación típica de $\bar{X}$

$\boldsymbol{\mu_X}$ será el valor a contrastar: 0 por la igualdad de los dos algoritmos

μ	$H_0 : \mu_D = \mu_0$ ($\mu_D = \mu_1 - \mu_2$)	$T = \frac{\bar{D} - \mu_0}{S_D/\sqrt{n}}$	D ~ N m.a. aparellada	$T \sim t_{n-1}$	Rebutjar H_0 si $\lvert T \rvert > t_{n-1,1-\alpha/2}$

D → A diferencia de las dos muestras, nosotros vamos a utilizar la letra X.

IV. Estudio del p-valor (α = 0.05)

Si el p-valor < α podemos rechazar la hipótesis nula, con una confianza del (1-α).

6.Resultados

Después de hacer cálculos con el programa R, obtenemos los estadísticos muestrales de los datos. Podemos observar que la media de tiempo de ejecución del algoritmo BFS es inferior a la del DFS, diferencia que se puede observar claramente en el Gráfico 1 donde se ve que la mayoría de los valores del BFS están centrados alrededor de la media con unos extremos próximos y en cambio los valores del DFS llegan a extremos más lejanos. Gracias al Gráfico 2 podemos corroborar nuestra premisa de Normalidad en las variables y que por tanto nuestros cálculos son válidos, teniendo una certeza del 95% de que los tiempos de ambos algoritmos no son iguales.

	Media(μs)	**Desviación estándar**
DFS (x)	34.98161	33.22524
BFS (y)	24.67288	14.17814
Diferencia	10.30878	28.0884

Taula I. Estadísticos muestrales de los datos

El summary y el boxplot conjunto es:

```
summary(B)
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 4.99   13.43   23.12   24.67   29.98   60.86
summary(D)
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
2.991  13.374  24.241  34.982  42.839 152.968
```

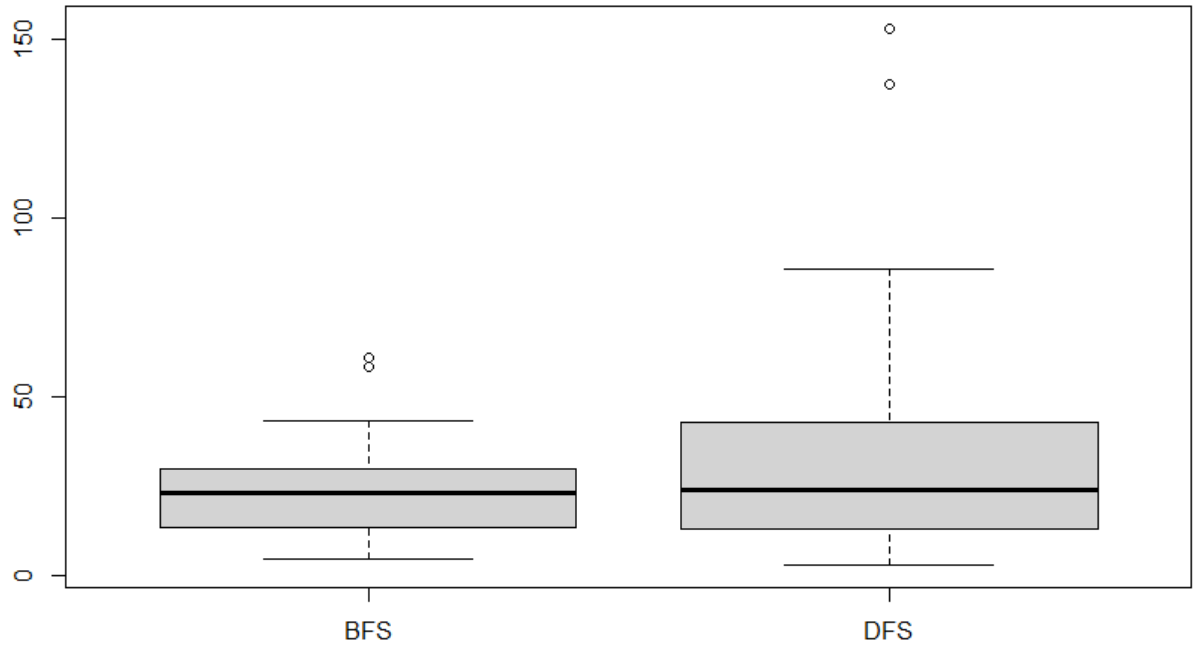


Gráfico 1. Diferencia de Boxplot

El boxplot de las dos muestras nos proporciona una imagen gráfica de los resultados obtenidos. Podemos ver claramente la media de los resultados de cada algoritmo.

En el caso del BFS observamos que la mayoría de las ejecuciones tienen tiempo de ejecución entre 13.443 y 29.98. Pero hay veces que el tiempo de ejecución es tan bajo como 5 y tan alto como 61. En el caso del DFS observamos que la mayoría de las ejecuciones tienen tiempo de ejecución entre 13.374 y 42.839. Pero en algunos casos el tiempo de ejecución es tan bajo como 2.991 y tan alto como 152.969. En los dos boxplots se puede observar outliers debido a la aleatoriedad de los resultados.

Seguidamente, comprobamos la premisa de normalidad de las variables mediante qqnorm de R:

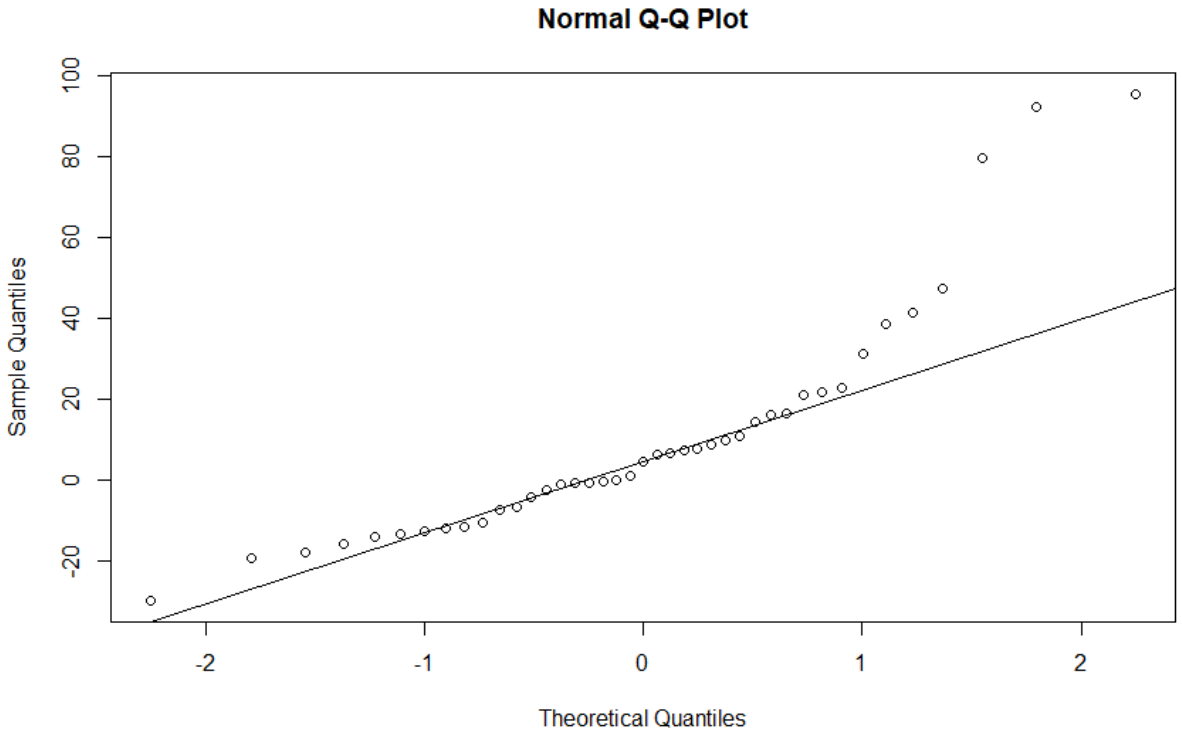


Gráfico 2. Q-Q Normal Difference

Aunque haya outliers, podemos afirmar que la diferencia de los datos del BFS y del DFS cumple la premisa de normalidad.

Analizando el Gráfico 3 podemos concluir 2 cosas:

- El algoritmo DFS acostumbra a tardar más a la vez de encontrar el mismo nodo P partiendo del mismo nodo K que el BFS, el cual tarda menos.
- Las muestras muestran cierta aleatoriedad.

A partir del Gráfico 3, también podemos ver que la mayoría de resultados están por encima del 0, lo que quiere decir que la mayoría de diferencias es positiva y por lo tanto favorable a la idea de que el algoritmo BFS es más rápido que el algoritmo DFS.

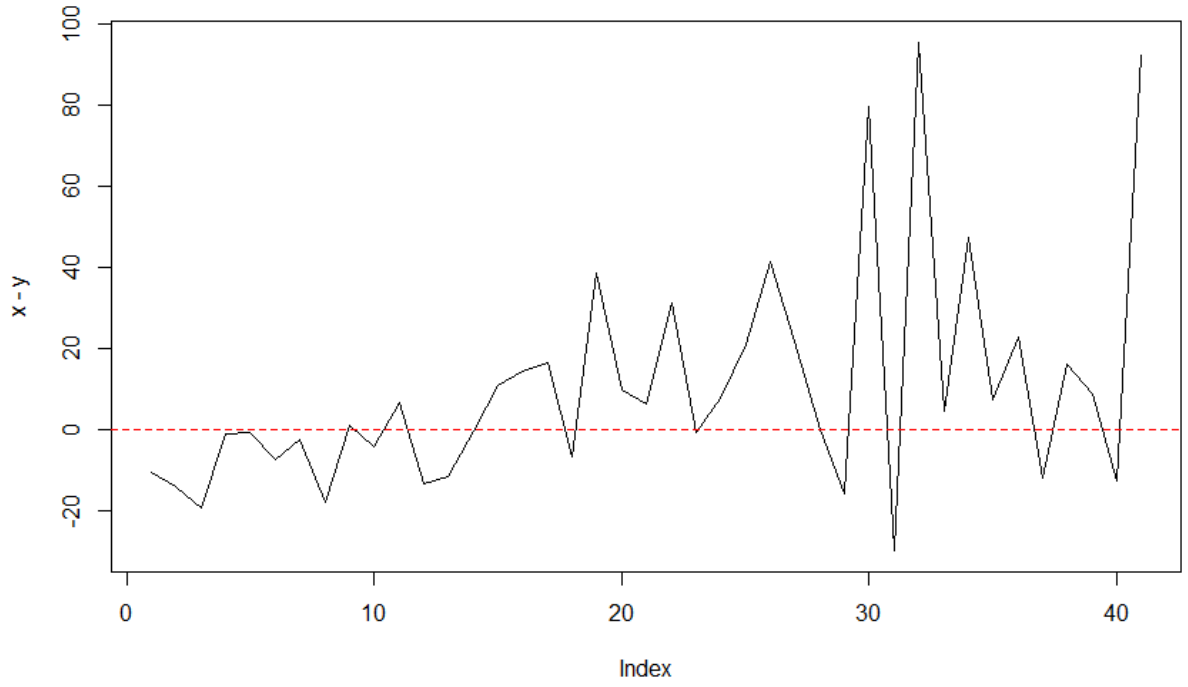


Gráfico 3. Gráfico de líneas de la diferencia

Finalmente, habiendo comprobado todas las premisas, y de nuevo mediante R, llevaremos a cabo la prueba de hipótesis, con el comando t.test, obteniendo los siguientes resultados:

```
Paired t-test

data: x and y

t = 2.35, df = 40, p-value = 0.02379

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval: [1.442989, 19.174572]

sample estimates: 10.30878
```

*Esquema 1: Resultado de aplicar el t-test a R sobre **x** y **y***

Gráficos de dispersión de los tiempos de ejecución de ambos algoritmos respecto el grado de los grafos:

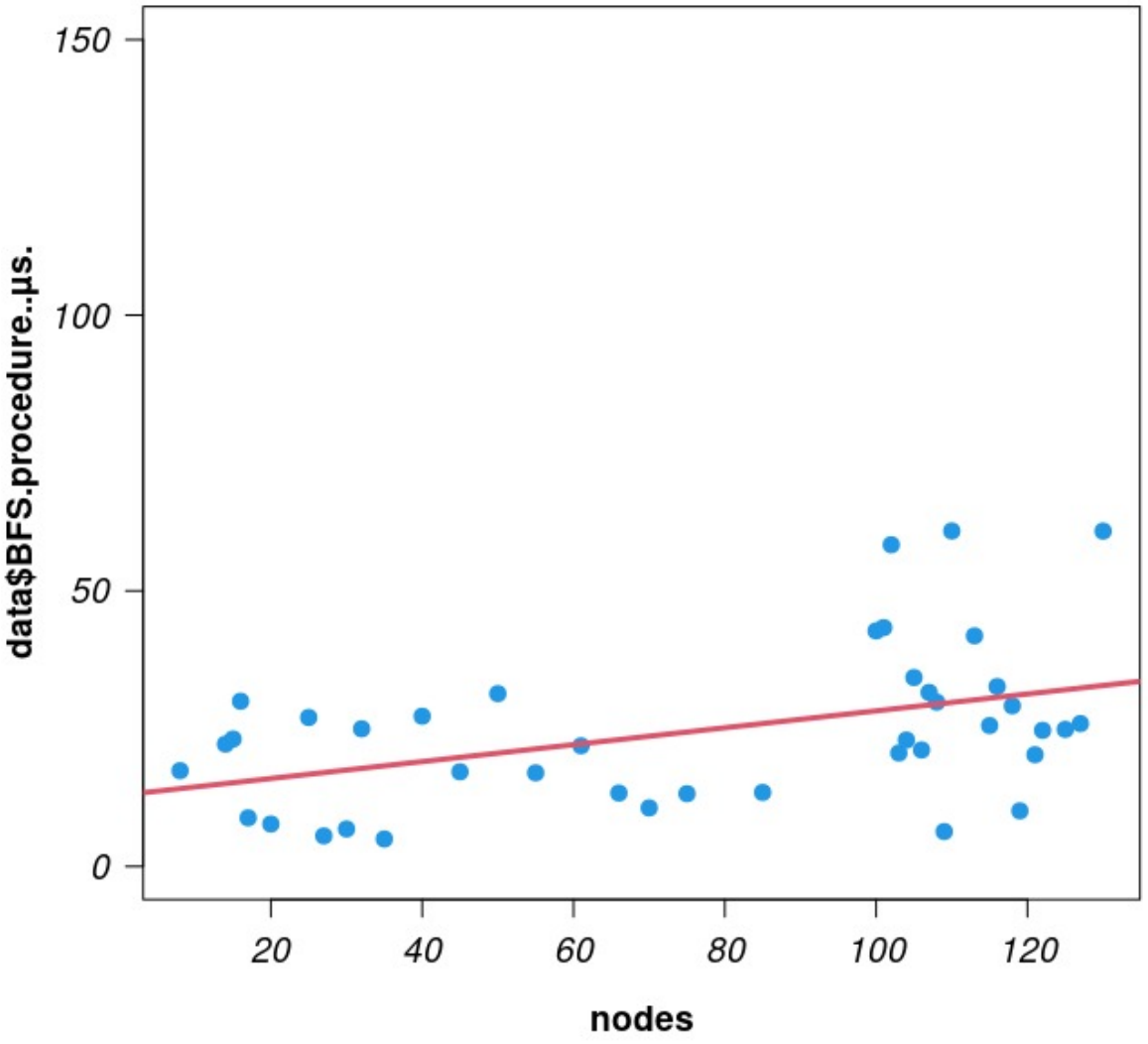


Gráfico de dispersión tiempo de ejecución BFS respecto nodos

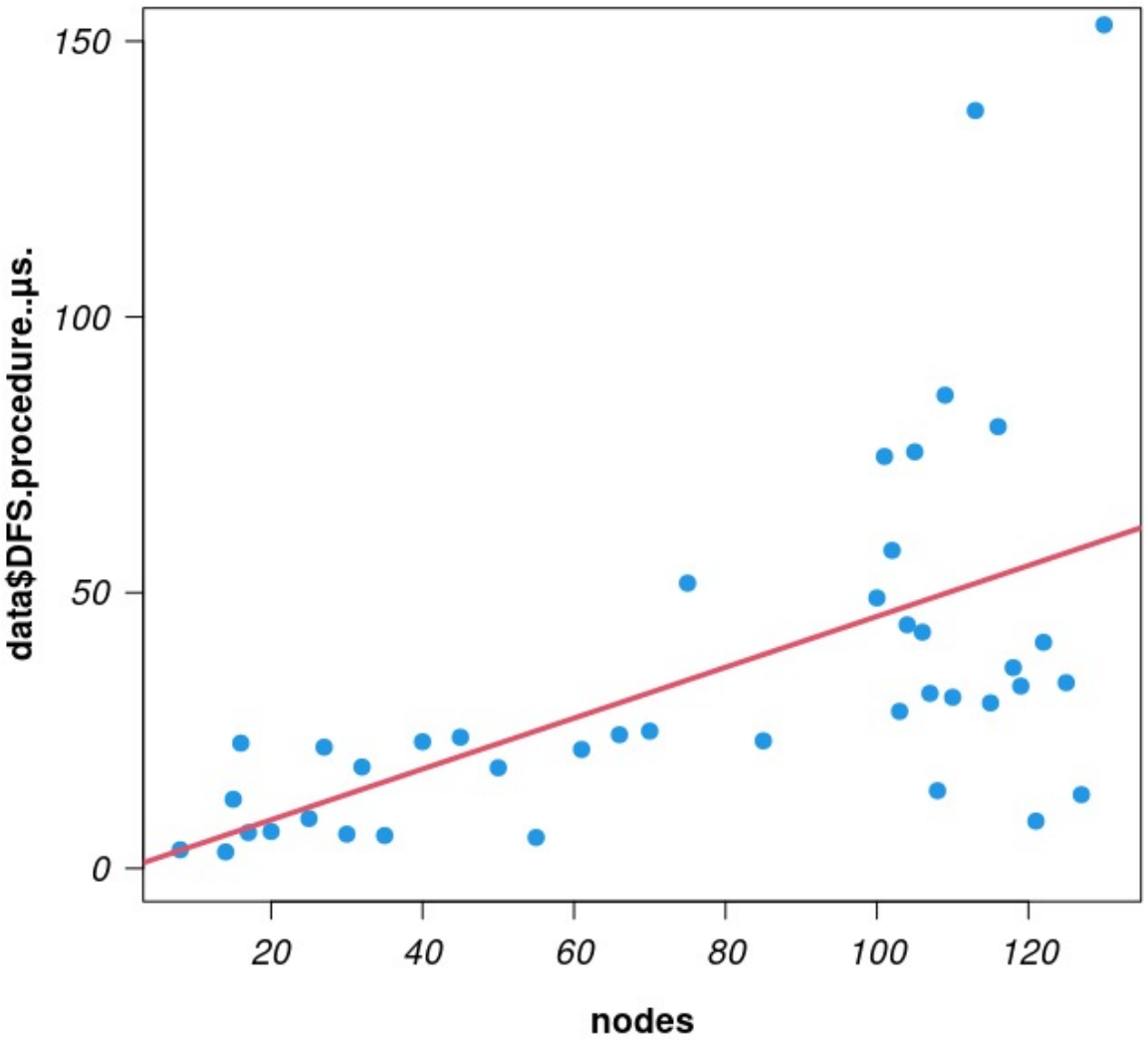


Gráfico de dispersión tiempo de ejecución DFS respecto nodos

7. Conclusión y discusión

Con los datos obtenidos y el análisis realizado podríamos concluir que el BFS es más rápido que el DFS (1.41 veces más rápido en promedio) y también decir que los valores del BFS se mantenían mucho más cerca del promedio que los del DFS (la desviación estándar del DFS es 2.343 veces mayor).

Hemos observado también que conforme el número de nodos aumenta el tiempo del DFS tiende a dispararse más que el del BFS.

Pensamos que el estudio no es extrapolable para otros grafos por los siguientes motivos:

- La muestra es pequeña (solo 40 grafos)
- El tipo de grafo utilizado es muy concreto (número de aristas fijado a #nodos(#nodos-1)/4)
- No hemos tenido en cuenta otras propiedades de los grafos como son la densidad ni tampoco la profundidad a la que se encontraban los nodos que buscábamos aunque ya sabíamos de antemano que a menos profundidad es mejor el BFS.

Para mejorar un estudio de este tipo y que realmente sea útil y significativo a la hora de trabajar con grafos creemos se debería usar una muestra mucho más grande, incluir otras variables como la densidad y la profundidad a la que se encuentran los nodos a buscar y también variar el número de aristas de los grafos ya que en la práctica rara vez van a estar fijadas en un número.