ANÀLISI DE L'EFICIÈNCIA DE BFS I DFS (Protocol)

Autors: Hugo Pelayo Aseko Obama, Sergio Borque, Daniel Hergueta
Assignatura: Probabilitat i Estadística
Curs: 2025 - 2026
Grup: 42

1. INTRODUCCIÓ

En l'àmbit de la informàtica, els **algorismes de cerca en grafs** constitueixen un dels fonaments més importants per a la resolució de problemes complexos. Des de la planificació de rutes i la navegació per xarxes fins a la intel·ligència artificial, la representació del coneixement o l'anàlisi de xarxes socials, la capacitat de recórrer un graf de manera eficient és essencial. En aquest context, dos dels algorismes més coneguts i estudiats són la **cerca en amplada** (*Breadth-First Search*, BFS) i la **cerca en profunditat** (*Depth-First Search*, DFS).

Malgrat la seva aparent simplicitat, aquests dos algorismes presenten **diferències conceptuals i pràctiques notables**. Tots dos permeten explorar un conjunt de nodes i arestes amb l'objectiu de trobar un element concret o determinar l'estructura d'un graf, però la seva manera d'afrontar aquesta exploració és oposada. El BFS explora el graf nivell per nivell, garantint trobar el camí més curt (en termes de nombre d'arestes) entre dos nodes quan el graf és no ponderat. En canvi, el DFS s'endinsa primer en un camí fins que ja no hi pot continuar, i només aleshores retrocedeix per explorar altres branques. Aquesta diferència de comportament es tradueix en diferents patrons d'ús de memòria, rendiment temporal i adaptació segons la naturalesa del graf.

La motivació d'aquest treball neix de la necessitat de **quantificar i analitzar empíricament** aquestes diferències amb el fi d'**entendre en quines condicions cada un d'ells pot ser més eficient o més adequat**. Si bé la teoria ens proporciona una visió general —sabem que tant BFS com DFS tenen una complexitat temporal de O(V + E) on V és el nombre de nodes i E el d'arestes, la seva eficiència real pot variar significativament segons les condicions experimentals. Factors com la **densitat del graf**, el **nombre de connexions per node**, la **profunditat mitjana dels camins** o la **estructura de connexió** poden afectar el temps d'execució de manera diferent per a cada algorisme, o, fins i tot, la mateixa implementació de tots dos algorismes o la del graf.

Per tal de facilitar la comparativa experimental entre els algorismes BFS i DFS, en aquest projecte, s'implementarà una infraestructura pròpia de generació i representació de grafs, mitjançant una classe genèrica `gentree<T>` (basada en un **contenidor associatiu** `unordered_map` que enllaça cada node amb el conjunt dels seus successors immediats) que permet crear, modificar i serialitzar grafs dirigits de diferents mides i densitats. A partir d'aquesta base, es realitzaran múltiples experiments controlats, comparant el temps d'execució i la longitud dels camins trobats per BFS i DFS en situacions diverses. Les dades obtingudes com les densitats, els temps mitjans i les longituds de camí entre els nodes inici i destí es registraràn per al seu tractament estadístic posterior, utilitzant mètodes d'estadística descriptiva.

La implementació adoptada respon a criteris de simplicitat i eficiència en entorns on els valors dels nodes són de **tipus bàsics** (com enters o caràcters), comptant amb el fet que la inserció i consulta en un `unordered_map` ofereixen un rendiment mitjà proper a O(1). No obstant això, cal remarcar que aquesta elecció no seria òptima si els nodes representessin **objectes més complexos** amb múltiples atributs o relacions internes, ja que en aquest cas seria preferible utilitzar estructures més robustes i segures, com ara llistes d'adjacència basades en punters, referències o gestors de memòria dinàmica.

Un aspecte a considerar és que, en la implementació actual, **els nodes es copien dins de les llistes d'adjacència**, en lloc d'emmagatzemar referències o identificadors únics. Aquesta simplificació s'ha fet de manera deliberada, amb l'objectiu de mantenir el focus del treball en **l'anàlisi del comportament dels algorismes de cerca** i no en la gestió de memòria o optimitzacions d'estructura de dades, això precisament no seria eficient per objectes més complexos.

Pel que fa a la implementació dels algorismes, la versió de la **cerca en profunditat (DFS)** s'ha implementat mitjançant **una pila iterativa** (`stack`) en lloc d'una versió recursiva tradicional. Aquesta decisió respon a motius de robustesa i control de recursos: la recursivitat, tot i ser més elegant des d'un punt de vista conceptual, pot comportar un risc d'**esgotament de l'espai d'execució (stack overflow)** en grafs de gran mida o amb camins molt profunds. En canvi, l'ús explícit d'una pila permet gestionar millor la memòria i mantenir un comportament previsible en tots els casos. Aquesta estratègia també facilita la mesura dels temps d'execució, ja que elimina la variabilitat introduïda per la gestió interna de la pila del sistema. En conjunt, aquestes decisions busquen un **equilibri entre simplicitat i control experimental**, assegurant que els resultats obtinguts reflecteixin principalment les diferències inherents als algorismes i no els efectes secundaris d'una implementació massa sofisticada.

En plantejar aquest treball, s'ha valorat acuradament la **viabilitat tècnica i temporal** del projecte per assegurar que l'objectiu sigui assolible amb els recursos disponibles i les eines vistes a l'assignatura. La comparativa d'eficiència entre els algorismes **BFS** i **DFS** s'ha considerat factible perquè es pot implementar i analitzar completament amb **coneixements de programació estructurada, gestió de dades i estadística descriptiva i inferencial**, competències ja assolides o pendent d'assolir en el marc del curs. A més, la infraestructura de proves s'ha dissenyat de manera automatitzada, fet que permet **generar, executar i registrar centenars de proves** sense necessitat d'intervenció manual, reduint així l'esforç temporal i garantint la coherència del conjunt de dades.

---

2. OBJECTIUS

L'objectiu principal és comparar el rendiment real de BFS i DFS sobre grafs dirigits de densitats i mides diverses. A més, s'estableixen altres objectius secundaris com estudiar la relació entre la densitat i el temps d'execució o el temps requerit per fer la cerca respecte la distància del camí recorregut. Aquests objectius s'han definit en termes **quantificables i observables**, fent servir variables numèriques concretes (temps en mil·lisegons de recorregut, nombre de nodes, nombre d'arestes, densitat i longitud del camí…). D'aquesta manera, l'estudi s'enfoca en resultats que es poden **analitzar estadísticament, representar gràficament.**

Per assolir aquest propòsit, s'han establert els següents objectius específics:

1. Mesurar el **rendiment temporal** de BFS i DFS sobre conjunts de grafs dirigits amb mides (nombre de nodes) i densitats variables. Aquesta mesura es fa a partir del temps total d'execució requerit per completar una cerca entre un node inicial i un

node final seleccionats aleatòriament.

2. Analitzar la **relació entre la densitat del graf i el temps d'execució** dels dos algorismes, per identificar si l'increment de connexions entre nodes té un impacte lineal, exponencial o insignificant sobre el cost computacional de cadascun.

3. Estudiar la **relació entre el temps d'execució i la longitud del camí recorregut**, per determinar si existeix correlació entre la distància real trobada i el cost temporal associat a la cerca, especialment en grafs de gran mida o densitat elevada.

Les mesures es registren en un fitxer `test_report.csv` per el seu anàlisi posterior.

---

3. VARIABLES

Per tal de garantir que els objectius del treball siguin mesurables, realistes i verificables, s'han definit un conjunt de variables quantitatives que permeten descriure tant les característiques estructurals dels grafs com el comportament dels algorismes aplicats sobre ells. Totes les dades es recullen automàticament durant l'execució del programa de proves (`tester_app`) i s'emmagatzemen en un fitxer CSV per a la seva anàlisi posterior.

3.1. Variables independents

Les variables independents són aquelles que es defineixen o controlen directament durant la generació dels grafs, i que poden influir en el rendiment dels algorismes. El següent conjunt de variables descriu l'escenari estructural sobre el qual operen els algorismes BFS i DFS:

- Nombre de nodes (`Nodes`): representa la mida del graf (el total de vèrtexs) i es genera de manera aleatòria dins d'un interval prefixat. Determina l'escala del problema i influeix directament en la complexitat temporal de les cerques.

- Nombre d'arestes (`Edges`): indica el nombre total de connexions dirigides entre nodes. Juntament amb el nombre de nodes, defineix la densitat del graf i la seva complexitat estructural.

- Densitat (`Density`): es calcula com D = E / (V*(V−1)), on E és el nombre d'arestes i V el nombre de nodes. Aquesta variable mesura el grau de connectivitat global del graf i és clau per entendre el comportament dels algorismes.

- Nodes d'inici i de destí (`Start Node`, `End Node`): seleccionats de forma aleatòria per a cada prova, defineixen el punt de partida i el punt final de la cerca. Tot i que són aleatoris, la seva variació introdueix diversitat en les situacions de cerca.

3.2. Variables dependents

Les variables dependents són aquelles que resulten de l'execució dels algorismes i que permeten quantificar-ne el comportament i l'eficiència. Les següents variables permeten establir una comparació directa entre els dos algorismes, tant des del punt de vista temporal com estructural:

- Temps d'execució de BFS (`BFS Time (ms)`): mesura el temps total, en mil·lisegons, que l'algorisme BFS triga a completar la cerca des del node inicial fins al node final.

- Temps d'execució de DFS (`DFS Time (ms)`): paral·lel a la variable anterior. Permet comparar directament el cost temporal de DFS amb el de BFS.

- Longitud del camí òptim de BFS i DFS (`Path Length`): indica el nombre d'arestes que componen el camí trobat entre els dos nodes seleccionats. En grafs no ponderats, aquest valor correspon al camí més curt possible.

3.4. Naturalesa i tractament de les dades

Totes les variables són discretes o contínues, la qual cosa permet aplicar tècniques estadístiques clàssiques (mitjanes, desviacions, regressions i proves t). Les mesures es prenen amb alta precisió mitjançant el temporitzador `base::scoped_timer`, i s'emmagatzemen en format decimal de doble precisió per evitar pèrdues d'exactitud.

El component `base::scoped_timer` utilitza el **patró RAII**, de manera que inicia el comptador en crear-se l'objecte i calcula automàticament el temps transcorregut en destruir-se. Això permet mesurar durades sense necessitat de gestionar manualment l'inici o el final del temporitzador. A més a més en qualsevol moment es pot consultar el temps transcorregut des de la creació de l'objecte.

---

4. PLA DE RECOLLIDA DE DADES

El procés de recollida de dades s'ha estructurat al voltant de dos programes principals que treballen de manera complementària:

1. Un mòdul generador de grafs, responsable de crear i emmagatzemar estructures dirigides amb propietats aleatòries controlades.

2. Un mòdul de proves que hem anomenat "tester_app", encarregat d'executar els algorismes BFS i DFS sobre cada graf i registrar els resultats en un fitxer de dades csv.

Aquesta arquitectura modular permet separar clarament la fase de generació de la fase d'experimentació, garantint que els grafs utilitzats siguin consistents i que els temps mesurats reflecteixin exclusivament el cost de les cerques, no la construcció dels grafs.

4.1. Generació automàtica de grafs

La generació dels grafs s'ha dut a terme mitjançant l'aplicació `generator_app`, implementada en C++20. Cada graf es construeix a partir de la classe `gentree<T>`, una estructura genèrica basada en un `unordered_map` que associa cada node amb el conjunt dels seus successors. Aquesta representació ofereix accés constant mitjà a les adjacències i resulta adequada per a tipus de dades bàsics. El nombre de nodes de cada graf es tria de forma aleatòria dins d'un interval configurat (per exemple, entre 500 i 1500), i per a cada node es generen un nombre aleatori d'arestes sortints, controlat pel paràmetre `max_adjacencies_per_node`.

Els valors aleatoris es generen mitjançant el mòdul `base::random`, que assegura una distribució uniforme dels nodes i connexions. Per a cada connexió, es crea una aresta dirigida `(from → to)`, no estan permesos autoenllaços i la unicitat dels nodes en les adjacències està garantida perquè fem servir un `unordered_set`. Finalment, el graf resultant es **serialitza a disc** en format de text mitjançant el mètode `gentree::serialize()`.

4.2. Execució de les proves i recollida de resultats

El segon programa, `tester_app`, és l'encarregat de **carregar els grafs generats** i d'executar les cerques BFS i DFS sobre cadascun d'ells. Aquest mòdul també fa ús la llibería externa **Taskflow** per paral·lelitzar les proves i maximitzar l'ús dels recursos del sistema disponibles. Els fitxers de grafs es carreguen mitjançant `tree_loader`, que reconstrueix la representació interna `gentree<int>` a partir dels fitxers `.txt` generats anteriorment. El format de serialització dels grafs és el següent:

1

2

3

1 2

2 3

El nombres individuals representen nodes i els parells de nombre X, Y representen una aresta que va de X a Y, ja que els grafs representats per la classe gentree son dirigits.

Per cada graf, el sistema selecciona **dos nodes d'origen i destí de manera aleatòria**. A continuació, s'executen els dos algorismes. El temps d'execució de cada cerca s'obté amb la classe `base::scoped_timer`, que mesura amb alta precisió el temps transcorregut entre l'inici i el final de l'operació. Les dades recollides s'agrupen en una estructura `test_report` per a cada graf, i posteriorment es **serialitza a un fitxer CSV únic (un per totes les proves)**.

El fitxer conté una línia per graf amb el format:

```
Graph Name, Density, Nodes, Edges, DFS Time (ms), BFS Time (ms),
Path Length, Start Node, End Node
```

4.3. Control experimental i garantia de validesa

Per assegurar que els resultats siguin fiables, cada prova s'ha executat sota condicions controlades: minimitzant el nombre de processos ocupant CPU i amb la mateixa configuració de compilació. Els experiments s'han paral·lelitzat, però no solapats en el temps dins del mateix graf, evitant efectes de dependència. Així mateix, s'ha verificat que el càlcul de densitat coincideixi amb el nombre d'arestes reals i que els nodes d'origen i destí escollits siguin vàlids dins del conjunt de nodes existents.

A continuació, s'adjunta un exemple de visualització d'un CSV generat:

	Graph Name	Density	Nodes	Edges	DFS Time (ms)	BFS Time (ms)	DFS Path Length	BFS Path Length	Start Node	End Node
1	../generator/resources/graph_707.txt	0.007825254743005123	1147	10286	0.734	0.487	219	3	44	1079
2	../generator/resources/graph_2727.txt	0.007473011822602961	1204	10824	0.563	0.179	159	2	792	954
3	../generator/resources/graph_318.txt	0.008226061523288045	1085	9675	0.927	0.864	195	3	1043	14
4	../generator/resources/graph_2607.txt	0.006697496855765913	1341	12035	1.935	1.252	65	3	218	24
5	../generator/resources/graph_3895.txt	0.008370909141870136	1069	9557	1.065	0.426	147	3	538	213
6	../generator/resources/graph_143.txt	0.00671250325131643 45	1339	12026	1.965	2.581	58	4	1303	681
7	../generator/resources/graph_764.txt	0.007467333889352238	1200	10744	1.267	2.362	135	5	1	182
8	../generator/resources/graph_285.txt	0.006051944530135963	1483	13301	0.913	1.599	276	4	4	659
9	../generator/resources/graph_3867.txt	0.00647119826140129	1392	12530	2.022	2.41	65	4	22	386
10	../generator/resources/graph_3640.txt	0.008831169333921755	1017	9125	1.627	1.095	27	3	514	126
11	../generator/resources/graph_426.txt	0.006584275939114649	1365	12259	1.913	0.927	76	3	475	1177
12	../generator/resources/graph_2081.txt	0.007045866264809145	1274	11427	0.027	1.604	5	4	149	150
13	../generator/resources/graph_1340.txt	0.007882436151729016	1141	10253	0.599	2.168	179	4	483	572
14	../generator/resources/graph_389.txt	0.007372347797879712	1222	11000	2.225	2.062	6	4	592	337
15	../generator/resources/graph_4914.txt	0.00890098674358853	1008	9035	0.934	0.222	159	2	35	899
16	../generator/resources/graph_2532.txt	0.006943066255314798	1290	11545	0.779	0.054	234	2	644	285
17	../generator/resources/graph_3970.txt	0.006644112684855031	1349	12082	1.176	1.036	221	3	628	1321
18	../generator/resources/graph_4295.txt	0.007616267077098837	1178	10560	2.074	1.095	17	3	624	399
19	../generator/resources/graph_1481.txt	0.006109973989891526	1472	13230	2.323	1.696	46	4	574	884
20	../generator/resources/graph_225.txt	0.00743516167433923	1206	10805	1.527	1.26	88	3	24	1059
21	../generator/resources/graph_2733.txt	0.006332534715966607	1418	12724	1.805	1.621	103	3	223	53
22	../generator/resources/graph_2309.txt	0.00671861213996838	1337	12001	1.97	2.424	55	4	163	790
23	../generator/resources/graph_941.txt	0.006349199948383766	1409	12596	1.642	0.408	149	3	523	104