UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

Descomposició de grafs en components i blocs: anàlisi, simulacions i aplicacions

Autor: Marc de San Martín Pons

Director:	Dr. Antoni Benseny Ardiaca
Realitzat a:	Departament de Matemàtiques i Informàtica
Barcelona,	20 de juny de 2019

Abstract

A block of a graph is a connected subgraph which does not contain any articulation point. A strongly connected component of a directed graph is a subraph in which every vertex is reachable from every other vertex in the subgraph through an oriented path. This project consists in applying extensions on search algorithms in order to detect blocks and strongly connected components in graphs and directed graphs respectively, and implement them in a computer program with a visual environment.

Resum

Un bloc d'un graf és un subgraf d'aquest tal que és connex i no té punts d'articulació. Un component fortament connex d'un graf dirigit és un subgraf d'aquest tal que per a qualsevol parella de vèrtexs existeix un camí orientat que vagi d'un a l'altre. Aquest treball se centra en aplicar extensions als algoritmes de cerca d'arbres expansius per tal de detectar blocs i components fortament connexos en grafs i dígrafs respectivament, i implementar-les en un programa informàtic amb entorn de visualització.

2010 Mathematics Subject Classification. 05C05, 05C20, 05C40, 05C85, 05C90

Agraïments

Principalment vull donar les gràcies al Dr. Antoni Benseny Ardiaca per accedir a ser el tutor d'aquest treball així com per tota la dedicació i interès que m'ha dedicat aquests dies. Sense la seva col·laboració aquest treball no hagués estat possible.

També vull agrair al meu germà l'haver-me aconsellat sàviament al llarg de la meva etapa com a estudiant de matemàtiques.

Finalment, m'agradaria dedicar-lo també a la família, als amics i a tothom a qui he conegut en aquesta etapa universitària.

Índex

1 Introducció

La teoria de grafs constitueix una de les branques principals de la matemàtica discreta. Es considera que aquesta àrea d'estudi fou engendrada com a conseqüència de la resolució del problema dels set ponts de Königsberg, duta a terme pel matemàtic suís Leonhard Euler. Königsberg (actualment Kaliningrad, Rússia) que fou la capital de Prússia Oriental, tenia un riu amb dues illes connectades entre elles per un pont, i amb cadascuna de les dues ribes per sis ponts més, tres des de cada illa. El problema plantejava si era possible creuar tots els ponts una única vegada i tornar al punt de sortida. Euler demostrà que no era possible simplificant els elements del problema convertint-los en un graf, representant les illes i les vores com a vèrtexs i els ponts com a arestes.

D'altra banda, la història de l'algorísmica és pot remuntar molts anys abans, ja que es pot considerar que els matemàtics han fet ús d'algoritmes des dels inicis de la seva història. El terme «algoritme» apareix per primer cop al segle IX, quan el matemàtic persa Muhammad ibn Musa al-Khwarizmi va escriure en llengua àrab un tractat sobre el sistema de numeració Hindú-Àrab, que va ser traduït al llatí al segle XII amb el títol *Algoritmi de numero Indorum*. La paraula *Algoritmi* fou la llatinització del nom d'al-Khwarizmi emprada pel traductor.

Aquest treball es pot englobar dins d'aquestes dues branques esmentades. La motivació per decidir-me per aquestes branques a l'hora d'escollir el tema del treball, ha vingut donada pel desig d'aprofundir en l'estudi dels conceptes vistos a l'assignatura de grafs, ampliant-los a les estructures dels grafs dirigits (o dígrafs), que no es van utilitzar; així com voler ampliar i millorar els programes que es van fer a les pràctiques de l'assignatura.

Les estructures singulars que es tracten principalment en aquest treball són els blocs dels grafs i els components fortament connexos dels dígrafs. L'estudi d'aquestes dues estructures ens pot aportar molta informació sobre la connectivitat del graf en qüestió. Tal i com s'explicarà en el treball, els blocs d'un graf són subgrafs connexos del graf que no contenen punts d'articulació. Els components fortament connexos d'un graf dirigit són subgrafs d'aquest tal que per a qualsevol parella de vèrtexs de cada subgraf existeix un camí orientat que vagi d'un vèrtex a l'altre.

Centrant-nos primer en els blocs, un dels seus trets importants radica en el fet que, per a alguns problemes, la identificació dels blocs pot ser una via de simplificació de grafs molt grans o complexos. Per exemple, la determinació de la planaritat d'un graf es pot simplificar cenyint-se a estudiar la planaritat de cadascun dels blocs que el formen. En altres problemes, un mateix bloc pot passar a ser tractat com un sol node, també amb l'objectiu de simplificar un graf.

Finalment, centrant-nos en els components fortament connexos d'un dígraf, aquests ens indiquen el nivell de connectivitat dels grafs dirigits. En els problemes on sigui necessari tenir una xarxa connectada fortament, identificar els components fortament connexos d'un dígraf ens permet discernir quines són les parts de l'estructura que necessiten millor connexió.

El projecte

Aquest treball té diferents objectius. El primer de tots és ampliar l'estudi de l'àrea matemàtica dels grafs fent recerca i anàlisi de les propietats dels dígrafs, així com de la resta de conceptes teòrics necessaris per explicar els algoritmes de cerca. A més, tindrem

per objectiu presentar una recopilació seriosa i ordenada dels conceptes.

Així mateix, es vol estudiar a nivell teòric els algoritmes de cerca, demostrant algunes de les propietats que se'n derivin i ampliant els algoritmes per detectar blocs en grafs i components fortament connexos en dígrafs.

Un altre objectiu central del projecte és implementar els diferents algoritmes estudiats en un programa informàtic de manera que la presentació dels resultats sigui el més visual, clara i entenedora possible.

Per últim, també intentarem aplicar un d'aquests algoritmes modificats per tal de trobar blocs o components fortament connexos en alguna estructura coneguda i analitzar quines conclusions en podem extreure.

Estructura de la Memòria

Aquesta memòria consta de tres grans blocs. El primer bloc, que es correspon amb la secció 2, conté la recopilació esmentada dels conceptes teòrics utilitzats. El segon bloc, que es correspon amb la secció 3, conté l'anàlisi conceptual dels mètodes de cerca d'arbres expansius. El darrer bloc; que es correspon amb les seccions 4, 5 i 6; conforma l'explicació de la implementació informàtica del projecte així com la presentació dels resultats aplicats a un exemple concret.

2 Conceptes preliminars

En aquesta primera secció es fa una recopilació dels conceptes de teoria de grafs que utilitzarem per desenvolupar el projecte. Fem servir de referència principalment a Bondy [1] i Soria [7], tanmateix, molts dels resultats es poden trobar en qualsevol llibre de teoria general de grafs.

2.1 Grafs

Definició 2.1. Un *graf* és un parell $G = (V, E)$ format per un conjunt no buit d'elements $V := V(G)$ anomenats *vèrtexs* o *nodes*, i un conjunt $E := E(G)$, disjunt de V, format per parells no ordenats d'elements de V que s'anomenen *arestes*.

En aquest treball considerarem que, tant el conjunt de vèrtexs V com el d'arestes E, seran sempre finits. Per indicar les arestes farem servir habitualment $e = \overline{uv}$ o $e = (u, v)$. La forma habitual de representar un graf és en el pla, utilitzant punts per a cadascun dels vèrtexs així com corbes per a les arestes.

Definició 2.2. Un *subgraf* $G' = (V', E')$ d'un graf $G = (V, E)$ és un graf tal que $V' \subseteq V$ i $E' \subseteq E$.

Seguidament es llisten un seguit de definicions conegudes dels grafs deixant clara la notació que es farà servir.

Definicions. Sigui $G = (V, E)$ un graf:

- Anomenem $\nu(G)$ (o simplement ν) el nombre de vèrtexs del graf G.
- Anomenem $\alpha(G)$ (o simplement α) el nombre d'arestes del graf G.
- Dos vèrtexs són *vèrtexs adjacents* si existeix una aresta que els uneix. Direm també que tal aresta i vèrtexs són *incidents*.
- El *grau* d'un vèrtex $v \in V(G)$, que notem per $g(v)$, és el nombre d'arestes incidents al vèrtex v.
- Anomenem $\delta(G) = \min\{g(v)\ \forall v \in V\}$ el grau mínim de G.
- Anomenem $\Delta(G) = \max\{g(v)\ \forall v \in V\}$ el grau màxim de G.
- Si $v \in V(G)$, $G - v$ és el subgraf resultant d'eliminar v i totes les arestes incidents a v. Anàlogament, si $e \in E(G)$, $G - e$ és el subgraf resultant d'eliminar l'aresta e.
- El *graf contret* per una aresta e, notat per $G \cdot e$, és el graf resultant d'identificar en un sol vèrtex els vèrtexs incidents a e i eliminar aquesta mateixa aresta.
- Si dues o més arestes uneixen un mateix parell de vèrtexs s'anomenen *arestes múltiples* o *arestes paral·leles*.
- Si una aresta uneix un vèrtex amb ell mateix s'anomena *llaç*.

Definició 2.3. Un *graf simple* és un graf que no té arestes múltiples ni llaços.

Definició 2.4. Un *graf complet* és un graf simple on tot parell de vèrtexs són adjacents.

Definició 2.5 (matriu d'adjacència d'un graf)**.** Sigui $G = (V, E)$ un graf: La *matriu d'adjacència* de G, $A(G) = (a_{i,j})_{i,j=1...\nu}$, és una matriu quadrada $\nu \times \nu$ on $a_{i,j}$ és el nombre d'arestes que incideixen alhora en els vèrtexs i i j.

Definició 2.6 (matriu d'incidència d'un graf)**.** La *matriu d'incidència* de G, $I(G) = (b_{i,j})_{i=1...\nu,\ j=1...\alpha}$ és una matriu $\nu \times \alpha$ tal que:

$$b_{i,j} = \begin{cases} 1, & \text{si l'aresta } j \text{ és incident amb el vèrtex } i \\ 0, & \text{altrament} \end{cases}$$

Així doncs, les matrius d'adjacència i incidència d'un graf dependran de l'ordre amb que haguem numerat vèrtexs i arestes. En el cas de la matriu d'adjacència d'un graf, aquesta serà sempre simètrica (ja que $a_{i,j} = a_{j,i}$). És igualment fàcil de comprovar que G serà un graf simple si i només si $A(G)$ compleix que $a_{i,i} = 0\ \forall i \in V$ i $a_{i,j} \in \{0, 1\}$.

Per tal d'evitar el gran espai de memòria informàtica que sol suposar treballar amb matrius, sovint es fa servir la representació i emmagatzematge de la informació dels grafs mitjançant llistes d'adjacència.

Definició 2.7 (llista d'adjacències d'un graf)**.** Sigui $G = (V, E)$ un graf, la seva *llista d'adjacències* es forma associant a cada vèrtex una llista amb els seus vèrtexs adjacents.

2.2 Grafs dirigits

Definició 2.8. Un *graf dirigit* o *dígraf* és un parell $D = (V, A)$ format per un conjunt no buit d'elements $V := V(D)$ anomenats *vèrtexs* o *nodes*, i un conjunt $A := A(D)$, disjunt de V, format per parells ordenats d'elements de V anomenats *arcs*.

A és doncs un conjunt de parells de vèrtexs amb una ordenació donada. Per representar els arcs usarem normalment $a = \overrightarrow{uv}$ o $a = (u, v)$ si queda clar pel context que és un arc i no una aresta. La forma habitual de representació d'un dígraf és també en el pla, usant punts per als vèrtexs i fletxes per als arcs, per marcar l'orientació d'aquests. Considerarem igualment que el conjunt d'arcs A d'un dígraf serà sempre finit.

Notació 2.9. *D'ara en endavant, en aquest treball l'ús del terme graf farà referència a graf no dirigit.*

Els conceptes de *subdígraf*, *llaç*, $G - a$, *dígraf simple*, $\nu(D)$ i $\alpha(D)$ es poden definir anàlogament al cas dels grafs (no dirigits) substituint les *arestes* per *arcs*. Podem transformar un graf G en un dígraf amb els mateixos vèrtexs que G i on cada aresta de G se substitueix per dos arcs unint els mateixos vèrtexs en sentits oposats. El dígraf resultant se sol anomenar *dígraf associat a* G, notat $D(G)$. Cada llaç sobre un vèrtex a G equivaldrà doncs a dos llaços a sobre el mateix vèrtex a $D(G)$.

També podem transformar un dígraf D en un graf no dirigit G senzillament eliminant l'orientació dels arcs obtenint així les arestes de G. Aquest graf s'anomena *graf subjacent de* D.

Finalment, si $D = (V, A)$ és un dígraf, el dígraf $D' = (V', A')$ tal que $V' = V$ i $e'_i = uv \in A'$ si i només si $e_i = vu \in A\ \forall i \in \{1, ..., \alpha\}$ (és a dir, el dígraf resultant de canviar les orientacions dels arcs de D) s'anomena *dígraf invers de* D.

Exemple 2.10. El dígraf $D = (V, A)$ tal que $V = \{1, 2, 3, 4, 5\}$ i $A = \{a = \overrightarrow{12},\ b = \overrightarrow{14},\ c = \overrightarrow{24},\ d = \overrightarrow{32},\ e = \overrightarrow{35},\ f = \overrightarrow{43},\ g = \overrightarrow{45}\}$ es pot representar tal i com es mostra a la *Figura 1.*

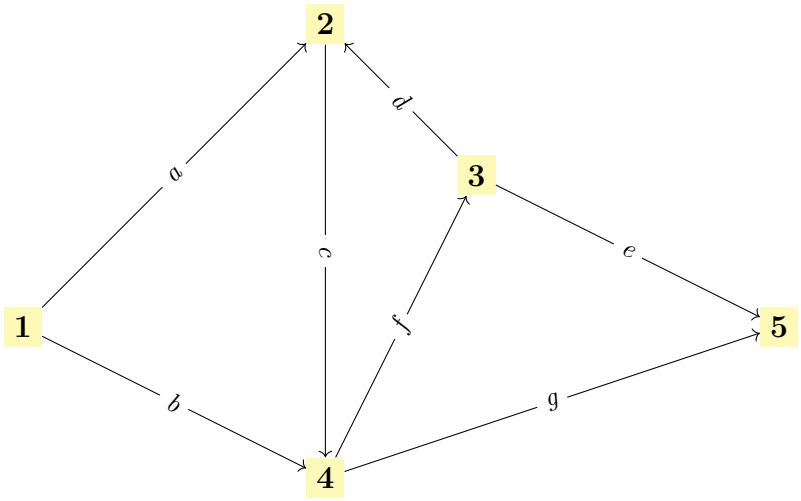


Figura 1: Exemple de representació d'un dígraf.

Definicions. Sigui $D = (V, A)$ un dígraf:

- Sigui $a \in A$ un arc tal que $a = \overrightarrow{uv}$, diem (de la mateixa manera que amb els grafs) que a és *incident* en u i v (i recíprocament). Així mateix, diem que u és *adjacent cap a* v (o que u *domina* v) i que v és *adjacent des de* u (o que v *és dominat* per u)
- Sigui $a \in A$ un arc tal que $a = \overrightarrow{uv}$, es diu que u és la *cua* d'a i v el *cap* d'a.
- Sigui v un vèrtex de D, anomenem *grau d'entrada de* v, que notarem per $g^-(v)$, al nombre d'arcs que tenen per cap al vèrtex v. Similarment, el *grau de sortida de* v, notat per $g^+(v)$, serà el nombre d'arcs que tenen per cua al vèrtex v. Formalment:

$$g^-(v) = \{v \in V \mid (u, v) \in A\}$$
$$g^+(v) = \{v \in V \mid (v, u) \in A\}$$

- Els vèrtexs que dominen un vèrtex v s'anomenen *veïns d'entrada*, tal conjunt el notem $N_D^-(v)$; mentre que els que són dominats pel vèrtex v s'anomenen *veïns de sortida*, i el notem $N_D^+(v)$.

Exemple 2.11. Donat el dígraf $D = (V, A)$ de la *Figura 1*, $N_D^-(2) = \{1, 3\}$ i $N_D^+(2) = \{4\}$.

Definició 2.12 (matriu d'adjacència d'un dígraf)**.** Sigui $D = (V, A)$ un dígraf: La *matriu d'adjacència* de D, $A(D) = (a_{i,j})$, és una matriu quadrada $\nu \times \nu$ on $a_{i,j}$ és el nombre d'arcs tals que tenen per cua i i per cap j.

Definició 2.13 (matriu d'incidència d'un dígraf)**.** Si D no té llaços, la *matriu d'incidència* de D, $I(D) = (b_{i,j})_{i=1\ldots\nu, j=1\ldots\alpha}$ és una matriu $\nu \times \alpha$ tal que:

$$b_{i,j} = \begin{cases} -1, & \text{si l'arc } j \text{ és incident amb el vèrtex } i \text{ per la cua} \\ 1, & \text{si l'arc } j \text{ és incident amb el vèrtex } i \text{ pel cap} \\ 0, & \text{altrament} \end{cases}$$

En el cas dels dígrafs es poden construir dos tipus diferents de llistes d'adjacències depenent de si llistem els veïns de sortida o els d'entrada.

Definició 2.14. Sigui $D = (V, A)$ un dígraf, la *llista d'adjacències de sortida* es construeix associant a cada vèrtex v els seus veïns de sortida $N_D^+(v)$. Així mateix, la *llista d'adjacències d'entrada* es construeix associant a cada vèrtex v els seus veïns d'entrada $N_D^-(v)$.

Exemple 2.15. Donat el dígraf $D = (V, A)$ de la *Figura 1*, les seves llistes d'adjacència de sortida i entrada són les següents:

1. | 2 | 4 |
2. | 4 |
3. | 2 | 5 |
4. | 3 | 5 |
5. $\emptyset$, la llista buida

1. $\emptyset$, la llista buida
2. | 1 | 3 |
3. | 4 |
4. | 1 | 2 |
5. | 3 | 4 |

Figura 2: Llistes d'adjacència de sortida (esquerra) i d'entrada (dreta)

2.3 Connectivitat de grafs i dígrafs

Definicions. Sigui $G = (V, E)$ un graf:

- Donada una successió de $n+1$ vèrtexs $v_1, ..., v_{n+1}$ en què cada v_i és adjacent a v_{i+1} $\forall i \in \{1...n\}$, un *camí de longitud k* és una successió de k arestes de G unint tals vèrtexs adjacents (de forma $\overline{v_1v_2}$, $\overline{v_2v_3}$, ... , $\overline{v_nv_{n+1}}$).
- Un *camí simple* és un camí tal que totes les arestes són diferents.
- Un *camí elemental* és un camí tal que totes les arestes i tots els vèrtexs recorreguts són diferents.
- Un *camí tancat de longitud k* és un camí que comença i acaba en un mateix vèrtex.
- Un *cicle* és un camí tancat tal que totes les arestes i tots els vèrtexs són diferents.

Les mateixes definicions són anàlogues pel cas dels dígrafs canviant que les successions són d'arcs i no d'arestes, i que la successió de $n+1$ vèrtexs ha de complir en aquest cas que v_i ha de ser adjacent cap a v_{i+1} $\forall i \in \{1...n\}$.

Per distingir-los dels camins en grafs, a un camí en un dígraf sovint se l'anomena *camí orientat* o *camí dirigit*.

Definició 2.16. Un graf es diu que és *connex* si per a cada parell de vèrtexs existeix un camí elemental.

Definició 2.17. Tot graf és unió disjunta de subgrafs connexos, els majors subgrafs connexos possibles formen els *components connexos* d'un graf.

La definició de connexió es pot enunciar de forma diferent, ja que s'observa fàcilment que, si existeix un camí qualsevol unint dos vèrtexs, existeix també un camí elemental que els uneix. El nombre de components connexos d'un graf G es denota per $\omega(G)$. Òbviament, si un graf és connex té un únic component connex.

Definicions. Sigui $G = (V, E)$ un graf connex:

- Una aresta e es diu que és una *aresta pont* si $G - e$ no és connex.
- Un vèrtex v es diu que és un *punt d'articulació* si $G - v$ no és connex.

Definició 2.18 (k-connectivitat)**.** Sigui $G = (V, E)$ un graf connex, $\kappa(G)$ indica el mínim nombre possible de vèrtexs tals que eliminant-los en resulta un graf no connex. Es diu que tal graf és *k-connex*, sempre que $k \leq \kappa(G)$.

Observació 2.19. Si G és un graf complet, llavors $\kappa(G) = \nu(G) - 1$.

Definició 2.20. Un *bloc* és un graf connex que no té punts d'articulació.

Observació 2.21. Tot bloc tal que $\nu \geq 3$ és 2-connex.

Definició 2.22. (bloc d'un graf) Sigui $G = (V, E)$ un graf, un *bloc de* G és un subgraf de G que és un bloc i que és maximal respecte tal propietat.

Així com en els grafs es dóna un únic tipus possible de connectivitat, en el cas dels dígrafs trobem tres maneres diferents per determinar la connectivitat d'aquests:

Definició 2.23 (connectivitat forta)**.** Un dígraf $D = (V, A)$ és *fortament connex* si per a qualsevol parella de vèrtexs (u, v) de V existeix un camí orientat que vagi de u a v.

Definició 2.24 (connectivitat unilateral)**.** Un dígraf $D = (V, A)$ és *unilateralment connex* si per a qualsevol parella de vèrtexs (u, v) de V existeix un camí orientat que vagi de u a v o de v a u.

Definició 2.25 (connectivitat feble)**.** Un dígraf $D = (V, A)$ és *feblement connex* si el seu graf subjacent és connex.

Qualsevol dígraf que no compleixi cap de les tres propietats de connexió enunciades es diu que és *inconnex*. Els majors subgrafs possibles que compleixin cadascuna de les propietats de connexió enunciades, formaran els *components fortament/unilateralment/ feblement connexos* del dígraf.

Observació 2.26. S'observa clarament que tot dígraf fortament connex és unilateralment connex i tot dígraf unilaterlament connex és feblement connex (i els recíprocs no es compleixen).

Teorema 2.27. *Un dígraf és fortament connex si i només si existeix un camí orientat tancat que passi per tots els seus vèrtexs (expansiu)*

Demostració.

$\Rightarrow$) Si $V(D) = \{v_1, ..., v_n\}$, en ser fortament connex existeix un camí orientat entre v_1 i v_2, un entre v_2 i v_3 i successivament fins a v_n i v_1. Unint cada camí obtenim un camí orientat tancat i expansiu.

$\Leftarrow$) Evidentment, si existeix un camí orientat tancat i expansiu implica que existeix un camí orientat entre tot parell de vèrtexs $u, v \in V(D)$

□

2.4 Arbres

Definició 2.28. Un *arbre* és un graf connex i sense cicles.

Proposició 2.29. *Un graf G és un arbre si i només si per qualsevol parell de vèrtexs existeix un únic camí elemental.*

Demostració.

$\Rightarrow$) Si G és un arbre, llavors és connex, llavors existeix almenys un camí elemental entre qualsevol parell de vèrtexs. Si n'existissin dos, es podria formar un cicle i seria una contradicció amb el fet de ser arbre. Per tant, només n'existeix un.

$\Leftarrow$) Si G no fos un arbre existiria algun cicle i, per tant, algun parell de vèrtexs amb dos camins elementals que els uneixi, cosa que seria una contradicció.

□

Definicions.

- Sigui T un arbre, s'anomena *fulla* a qualsevol vèrtex de T de grau 1.
- Un *bosc* és un graf els components connexos del qual són arbres.

Teorema 2.30. *Sigui G un graf, les tres condicions següents són equivalents:*

1. *G és un arbre.*
2. *G no conté cap cicle, però afegint-li una aresta qualsevol es forma un cicle.*
3. *G és connex i qualsevol aresta de G és una aresta pont.*

Demostració.

1$\Rightarrow$2) Sigui G un arbre, si s'hi afegeix una aresta múltiple o un llaç es forma un cicle de forma evident. Suposem que afegim una aresta e_1 entre dos vèrtexs u i v. En ser G connex existeix un camí a entre u i v; llavors, a concatenat amb e_1 forma un cicle.

2$\Rightarrow$3) Prenem e_1 una aresta qualsevol de G incident en vèrtexs v_1 i v_2 i suposem que no és aresta pont. Aleshores, $G - e_1$ segueix sent connex, és a dir existiria un camí a entre v_1 i v_2 que no conté l'aresta e_1. Aleshores concatenant a amb e_1 es formaria un cicle a G obtenint així una contradicció.

3$\Rightarrow$1) Si G té un cicle, eliminant qualsevol aresta del cicle el graf seguiria sent connex. Obtenim així una contradicció amb que totes les arestes són aresta pont.

□

Observació 2.31. Del teorema anterior es pot extreure directament que un graf connex G és un arbre si i només si $\alpha = \nu - 1$. En particular, un arbre té el menor nombre possible d'arestes que fan que el graf sigui connex.

Definició 2.32. Un *arbre expansiu* (o *arbre d'expansió*) d'un graf connex G és un subgraf que és un arbre i que conté tots els vèrtexs de G.

Definició 2.33. Un *bosc expansiu* (o *bosc d'expansió*) d'un graf no connex G és un subgraf que conté tots els vèrtexs de G i tal que cada component connex del subgraf és un arbre expansiu.

El següent teorema ens relaciona els conceptes d'arbres amb les nocions de connectivitat de la subsecció anterior:

Teorema 2.34. *Un graf G és connex si i només si conté un arbre expansiu.*

Demostració.

$\Rightarrow$) Si G és un arbre el resultat és evident.

Si G no és un arbre, en ser connex existeix una aresta e que forma part d'un cicle (i aleshores no serà aresta pont). Si eliminem iterativament una aresta de cada cicle de G, que no són arestes pont, obtindrem un graf connex amb totes les arestes sent arestes pont i aplicant el Teorema 1.30 concloem que serà un arbre (i expansiu).

$\Leftarrow$) Si existeix un graf expansiu T, per la Proposició 1.29 dos vèrtexs qualssevol de G estan connectats per un camí de T.

□

Seguidament introduirem el concepte d'arbre en dígrafs. Per fer-ho hem d'introduir un seguit de definicions elementals.

Definicions.

- Un *cicle dirigit* és un cicle fortament connex, és a dir, un cicle on tots els arcs estan orientats en un mateix sentit.
- Un *dígraf acíclic* és un dígraf que no conté cap cicle dirigit.
- Sigui D un dígraf acíclic, una *font* és un vèrtex de D tal que existeix un camí orientat des d'ell a qualsevol altre vèrtex de D.
- Sigui D un dígraf acíclic, un *pou* és un vèrtex de D tal que existeix un camí orientat des de qualsevol altre vèrtex de D a ell.

Proposició 2.35. *Un dígraf acíclic té com a mínim un vèrtex que compleix $g^-(v) = 0$ i com a mínim un vèrtex que compleix $g^+(v) = 0$.*

Demostració.

Considerem el darrer vèrtex d'un camí maximal qualsevol del dígraf acíclic. Aquest no pot tenir veïns de sortida ja que si en tingués el dígraf tindria un cicle o bé el camí no seria maximal.

El mateix raonament es pot aplicar al primer vèrtex que recorre un camí maximal, en aquest cas, per concloure que no pot tenir veïns d'entrada.

□

Definició 2.36. Un *arbre de sortida* és un dígraf feblement connex que té un únic vèrtex tal que $g^-(v) = 0$ (que serà una font) i tots els altres vèrtexs compleixen que $g^-(v) = 1$.

Definició 2.37. Un *arbre d'entrada* és un dígraf feblement connex que té un únic vèrtex tal que $g^+(v) = 0$ (que serà un pou) i tots els altres vèrtexs compleixen que $g^+(v) = 1$.

Exemple 2.38. Exemplifiquem a la *Figura 3* un arbre de sortida i un d'entrada. El vèrtex 0 és una font i el vèrtex 6 és un pou.

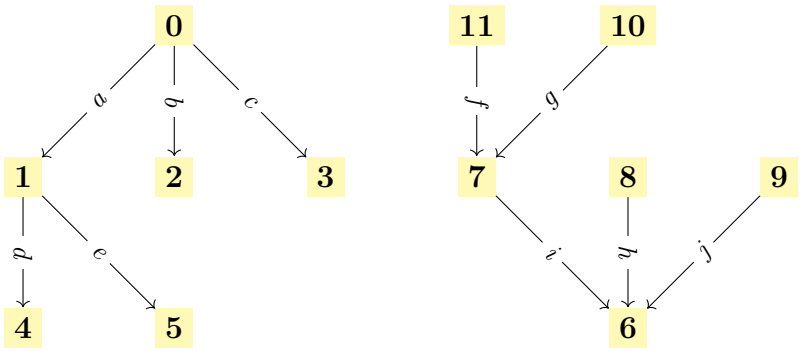


Figura 3: Un arbre de sortida (esquerra) i un arbre d'entrada (dreta).

3 Algoritmes de cerca

Tal i com s'ha explicat, recórrer un graf o un dígraf amb un cert ordre és un procés àmpliament necessitat en multitud d'algoritmes que treballen amb grafs i dígrafs. Aquest procés es pot fer de distintes maneres depenent del criteri que se segueixi per recórrer els vèrtexs. A l'hora de recórrer tots els vèrtex d'un graf, els mètodes generen arbres expansius, d'aquí que se'ls anomeni *algoritmes de cerca* (cerca d'arbres expansius). En aquesta secció s'introduiran els mètodes de cerca implementats al projecte així com les extensions als mètodes realitzades. La referència que es fa servir principalment en aquesta secció és Gibbons [2].

Els mètodes i les seves extensions es poden aplicar tant a grafs com a dígrafs. Començarem explicant els mètodes en els casos de grafs no dirigits. Cal destacar que tractant amb dígrafs, els mètodes de cerca ens generaran boscos expansius formats per arbres de sortida (anomenats *boscos expansius de sortida*).

Quan es treballa amb arbres, és comú fer ús de vocabulari de genealogia per indicar tant la profunditat relativa de cada vèrtex com les adjacències relatives, així que prèviament a explicar els mètodes introduïm el vocabulari utilitzat per definir futurs conceptes.

Definicions.

- L'*arrel* és el vèrtex escollit des del qual s'inicia el mètode de cerca corresponent.
- Donat un vèrtex v, el vèrtex adjacent a v trobat amb anterioritat es diu que és el *pare*, mentre que els vèrtexs adjacents trobats amb posterioritat a v es diuen que són *fills*. Dos vèrtexs que tenen un mateix pare es diu que són *germans*.
- Donat un vèrtex v, els fills d'un fill de v i posteriors són *descendents* de v (els fills de v i v mateix també solen estar inclosos en la definició de descendents).
- Donat un vèrtex v, els pares d'un pare de v i anteriors són *ancestres* de v (i la definició inclou també els pares de v i v).

Observació 3.1. Donat un arbre expansiu, els vèrtexs només poden tenir un sol pare. En cas contrari es formaria un cicle amb els dos pares i l'arrel.

Observació 3.2. Tot mètode de cerca s'inicia escollint l'arrel i, mentre que en els grafs qualsevol vèrtex és elegible, en el cas dels dígrafs haurà de complir les condicions que el facin arrel d'un arbre de sortida.

3.1 Mètode de cerca en amplada (BFS)

El primer dels mètodes de cerca que presentem és el mètode de *cerca en amplada* (en anglès *Breadth First Search* i d'ara en endavant, *BFS*). El BFS recorre els vèrtexs d'un graf o dígraf ordenadament per nivells de profunditat respecte a l'arrel fixada. Per tant, el pas general del mètode seria visitar tots els vèrtexs adjacents als darrers vèrtexs visitats.

Així doncs, el mètode s'inicia en l'arrel. Seguidament es visiten tots els seus vèrtexs adjacents (fills), després tots els adjacents dels fills (o fills dels fills) i així successivament.

Exemple 3.3. A la *Figura 4* mostrem, de dalt a baix i d'esquerra a dreta, els passos que segueix el mètode BFS. En groc indiquem un vèrtex que no ha sigut visitat, en verd fort els últims vèrtex visitats i dels quals es buscaran els adjacents, i en verd clar un vèrtex ja visitat i ja expandit.

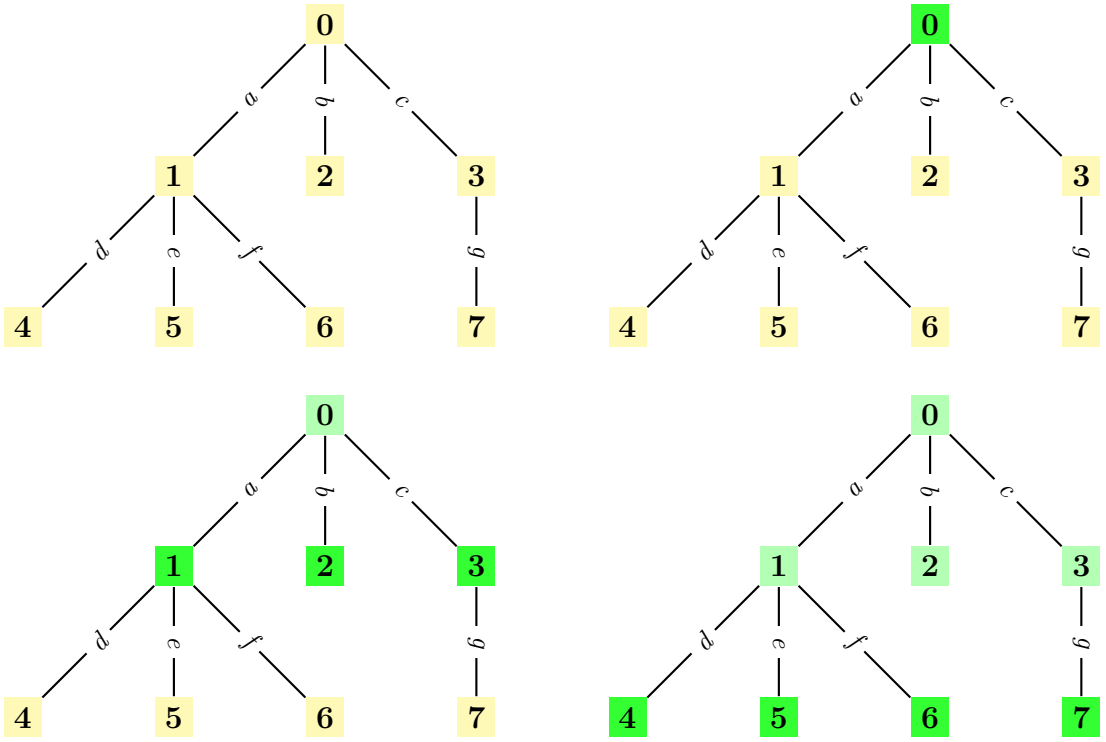


Figura 4: Ordre de visita dels vèrtexs per etapes del mètode BFS.

Observació 3.4. En l'exemple de la *Figura 4* s'exemplifica una cerca BFS en un graf que és un arbre per tal d'intentar que sigui més aclaridor. En general G no és necessàriament un arbre, com és evident.

Des del punt de vista de cerca d'arbres expansius, el mètode BFS es diu que és *complet* perquè si existeix una solució possible sempre la troba, i *òptim*, perquè ens proporciona la distància més curta possible de l'arrel a la resta de vèrtexs (considerant que els grafs no són ponderats).

L'aplicació del mètode BFS en dígrafs és anàloga al cas dels grafs, tenint en compte que els vèrtexs adjacents a considerar seran, en aquest cas, els veïns de sortida del vèrtex corresponent. El vocabulari genalògic introduït també es pot fer servir tenint en compte la mateixa consideració.

3.2 Mètode de cerca en profunditat (DFS)

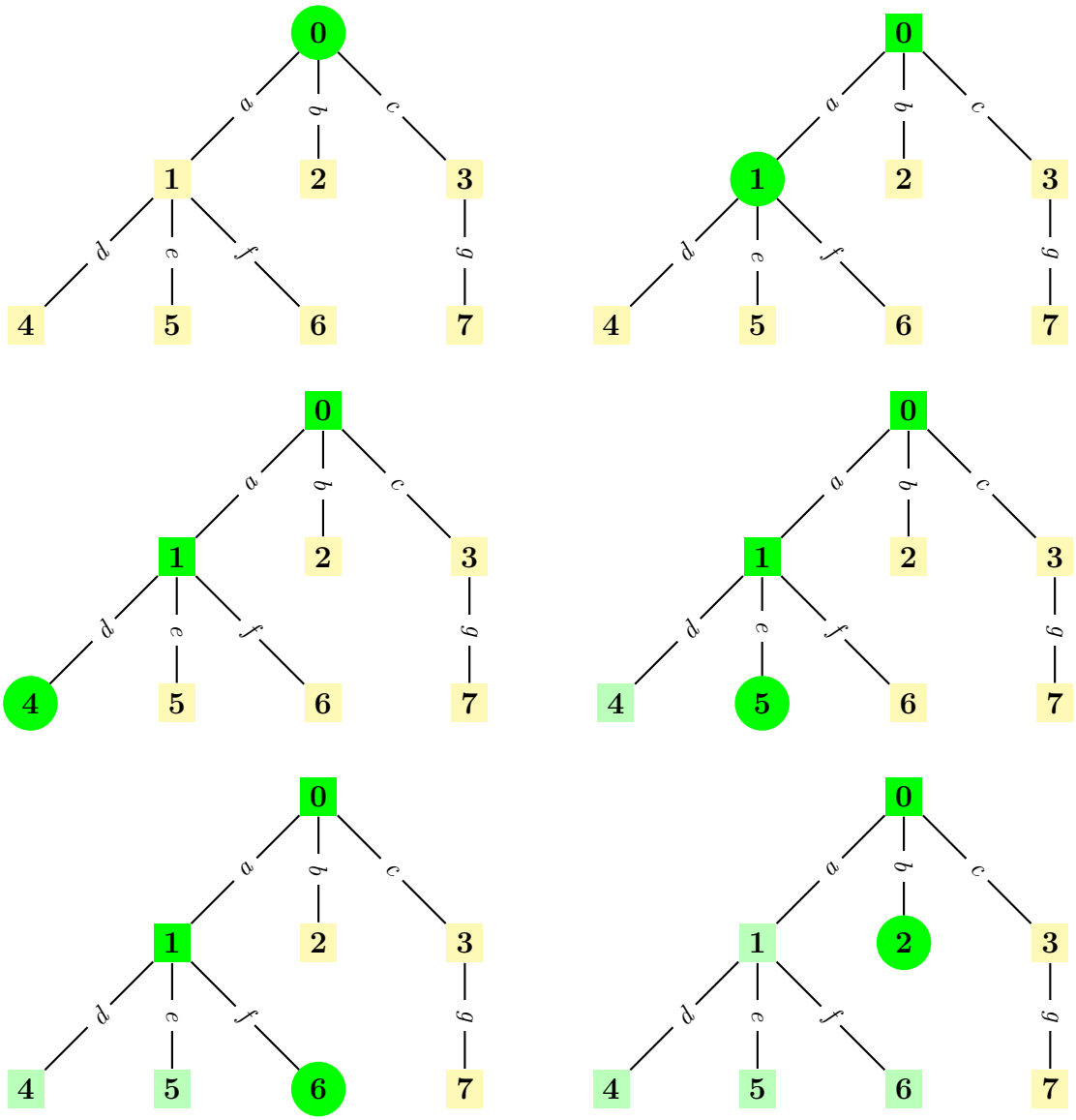
Ens fixem ara en el mètode de *cerca en profunditat* (en anglès *Depth First Search* i d'ara en endavant, *DFS*). El pas general que segueix el DFS a l'hora de recórrer els vèrtexs és el següent: des de l'últim vèrtex visitat es visita el següent vèrtex adjacent no visitat. Si no existeix tal vèrtex (tots els vèrtexs adjacents han sigut visitats) es torna enrere i s'observa si el vèrtex ancestre té adjacents no visitats (*backtracking*). Aquest pas de backtracking es fa usant obligatòriament les arestes emprades anteriorment per visitar vèrtexs adjacents.

Si s'arriba a un punt en què no troba vèrtexs no visitats, es comprova si tots els vèrtexs del graf han sigut visitats o no. Si es troben vèrtexs no visitats, s'escull el següent com a arrel i s'inicia el mètode des d'aquell vèrtex (cosa que implica que s'ha trobat un

component connex diferent). El mètode finalitza quan tots els vèrtexs del graf han sigut visitats.

Exemple 3.5. A la *Figura 5* mostrem, de dalt a baix i d'esquerra a dreta, els passos que segueix el mètode DFS aplicat al mateix graf tipus arbre de l'*Exemple 2.2*.

En aquest cas indiquem en color groc un vèrtex que no ha sigut visitat, encerclat en color verd fort l'últim vèrtex visitat, en color verd fosc sense encerclar un vèrtex visitat que té vèrtexs adjacents no visitats (s'hi ha de fer backtracking), i en color verd clar un vèrtex visitat que ja no té adjacents no visitats.



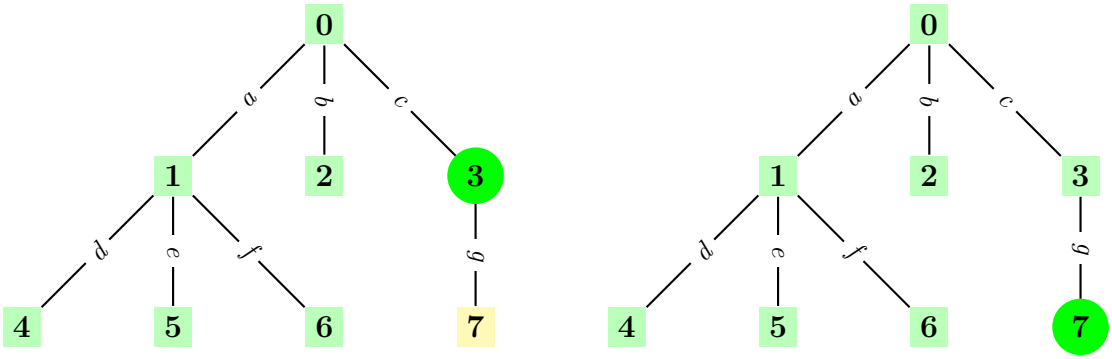


Figura 5: Ordre de visita dels vèrtexs per etapes del mètode DFS.

A diferència del mètode BFS, el DFS no és ni complet (en el cas que tractéssim amb grafs infinits, cosa que hem desconsiderat tal i com s'ha dit prèviament) ni òptim, ja que l'arbre expansiu solució pot no ser de distància mínima de l'arrel a la resta de vèrtexs.

Proposició 3.6 (classificació d'arestes)**.** *Sigui $G = (V, E)$ un graf, el mètode DFS aplicat a G classifica les arestes en una partició de dos subconjunts: les arestes que s'han utilitzat i que, per tant, formen part del bosc expansiu (que notarem per F); i el subconjunt d'arestes no utilitzades $B = E - F$, que s'anomenen arestes de tornada (back-edges en anglès).*

La necessitat d'ordenar els vèrtexs segons l'ordre de visita adquireix certa rellevància en el DFS. Aprofitem per introduir l'índex que guarda tal informació usant notació específica.

Notació 3.7. *Notem d'ara en endavant per DFI(v) l'ordre en què ha sigut visitat cada vèrtex v en el transcurs d'una cerca DFS. Es pot anomenar índex de cerca en profunditat de v.*

Observació 3.8. Si convinguéssim que d'entrada $DFI(v) = 0\ \forall v \in V$, i posteriorment cada vèrtex aniria prenent el seu valor $DFI(v)$, una manera per determinar el final de la rutina seria comprovar que no quedin vèrtexs tals que $DFI(v) = 0$.

El teorema següent ens relaciona la classificació d'arestes enunciada a la *Proposició 3.6* amb el mètode DFS.

Teorema 3.9. *Aplicant un DFS a un graf $G = (V, E)$, tota aresta de tornada connecta un descendent amb un predecessor seu.*

Demostració.

Siguin $u, v \in V$, suposem que u és visitat abans que v, sabem que u està a la llista d'adjacència de v i viceversa. Aleshores, per funcionament de l'algoritme DFS, es visitarà v, quan s'estigui expandint u o algun dels seus descendents (com que sabem que estan connectats per una aresta de tornada, no de l'arbre, no es visitarà expandint u sinó un dels seus descendents). Per tant, v només pot ser un descendent de u, i u un ancestre seu.

□

Exemple 3.10. Un cop vista l'explicació del mètode, a la *Figura 6* es mostra una exemplificació en pseudocodi de com es desenvolupa la rutina.

```
procedure DFS(G)
1. set i:=0, F:=void and vn=size(G)
2. for all v in V do DFI(v)=-1
3. while for some u DFI(u)=-1, do dfs(u)
4. procedure dfs(v)
        begin
5.      DFI(v)=i
6.      i=i+1
7.      for all v' in A(v) do
8.          if DFI(v')=-1 then
9.                  add (v,v') to F
10.                 dfs(v')
        end of dfs
11. output F
```

Figura 6: Escriptura en pseudocodi de l'algoritme DFS.

Observem que el nombre de cops que el codi entra en la rutina *dfs* (en minúscules) es correspon amb el nombre de components connexos del graf.

3.2.1 DFS en dígrafs: classificació d'arcs

El DFS es pot aplicar de la mateixa manera en dígrafs, tenint en compte unes consideracions: en primer lloc, hem de fixar que l'arrel $r \in V$ on iniciem el mètode per tal que compleixi que $g^+(r) > 0$. Com ja hem explicat, els vèrtexs adjacents que consideràvem en els grafs, són en aquest cas els veïns de sortida de cada vèrtex v.

La diferència substancial de l'aplicació del mètode en dígrafs respecte en grafs rau en la classificació dels arcs que se'n deriva. De forma general, les diferències ocorren com a conseqüència de l'orientació dels arcs. Per exemple, on abans teníem *back-edges* ara tindrem dues classes diferents. També tenim una nova classe d'arcs creuats. Així doncs, la partició amb què es classifiquen els arcs en el cas dels dígrafs està formada pels 4 subconjunts següents:

1. El subconjunt d'arcs del bosc expansiu de sortida, F, format pels arcs usats pel mètode per trobar vèrtexs veïns.
2. El subconjunt d'arcs format per arcs orientats des d'un descendent cap a un ancestre. Els anomenem *arcs de tornada*, notats per B_1.
3. El subconjunt d'arcs format per arcs orientats des d'un ancestre cap a un descendent. Els anomenem *arcs d'anada*, notats per B_2.
4. El subconjunt d'arcs que va d'un vèrtex cap a un altre que no és descendent ni ancestre seu, és a dir, $C = A - F - B_1 - B_2$. Els anomenem *arcs creuats*.

Teorema 3.11. *Aplicant un DFS a un dígraf $D = (V, A)$, si $a = \overrightarrow{uv}$ és un arc creuat, aleshores $DFI(u) > DFI(v)$.*

Demostració.

Per contradicció, suposem que $DFI(u) < DFI(v)$. Com que v està en la llista d'adjacència (de sortida) de u i es visita primer u, sabem que v es visitarà quan s'estigui expandint u. Llavors només pot ser o bé un arc del bosc expansiu ($a \in F$), si v s'explora usant l'arc a; o bé un arc d'anada ($a \in B_2$), si és visitat posteriorment per un descendent de u. Per tant, no pot ser un arc creuat i s'arriba a una contradicció.

□

3.3 Mètode DFS per trobar blocs de grafs

Tot seguit s'explica l'extensió del mètode DFS duta a terme per tal de trobar blocs en grafs. Es pot intuir fàcilment que, per tal de trobar i identificar els blocs d'un graf, es buscaran quins dels vèrtexs són punts d'articulació. L'acompliment de l'objectiu de l'extensió es basa fonamentalment en el teorema que enunciem a continuació:

Teorema 3.12. *Un punt d'articulació v compleix:*

(i) Si v és l'arrel d'un arbre del bosc expansiu generat pel DFS, aleshores v té més d'un fill.

(ii) En cas contrari, v té un fill v' tal que cap descendent de v' (inclòs v' mateix) està connectat amb un ancestre propi de v mitjançant una aresta de tornada.

Observació 3.13. Per ancestre *propi* de v s'entén un ancestre del vèrtex no incloent el mateix v.

De la mateixa manera que hem realitzat anteriorment amb el $DFI(v)$, ens convé definir un nou índex per a la implementació posterior.

Definició 3.14. Notarem d'ara en endavant per $P(v)$ el mínim entre $DFI(v)$ i el DFI dels vèrtexs connectats mitjançant una aresta de tornada a un descendent de v (v inclòs). Per tant:

$$P(v) = \min \begin{cases} DFI(v) \\ DFI(v') \parallel \overline{vv'} \in B \\ P(v') \text{ on } v' \text{ és un fill de } v. \end{cases}$$

Observació 3.15. Evidentment, es compleix que $P(v) \leq DFI(v)$.

Proposició 3.16. *La segona conclusió del Teorema 3.12 es pot reescriure usant l'índex definit $P(v)$ de la manera següent:*

(ii) En cas contrari, v té un fill v' tal que $P(v') \geq DFI(v)$.

Notació 3.17. *Anomenarem per B(G) el nombre de blocs d'un graf (caldrà diferenciar-la de la B que indica el subconjunt d'arestes de tornada pel context).*

L'objectiu conegut de l'extensió és trobar els diferents blocs d'un graf, per tant, s'haurà de recollir la informació de les arestes que formen cada bloc. Per resoldre-ho, l'algoritme fa ús d'una pila (*stack* en anglès) on s'aniran apilant les arestes fins al moment que l'algoritme trobi un bloc. Aquest moment serà quan es trobi un vèrtex v tal que té un fill v' que compleix $P(v') \geq DFI(v)$. Deixarem la demostració per més endavant.

Exemple 3.18. Un cop vistos els detalls de l'extensió de l'algoritme DFS perquè trobi blocs, a la *Figura 7* mostrem com seria en pseudocodi el desenvolupament de la nova rutina, usant de base el pseudocodi del mètode DFS.

```
procedure DFSB(G)
1. set i:=0
2. empty the stack
3. for all v in V do DFI(v)=-1
4. while for some u, DFI(u)=-1 do dfsb(u)
5. procedure dfsb(v)
        begin
6.      DFI(v)=i
7.      P(v)=DFI(v)
8.      i=i+1
9.      for all v' in A(v) do
            begin
10.         stack (v, v') if not stacked
11.         if DFI(v')=-1 then
                    begin
12.                 v=father(v')
13.                 dfsb(v')
14.                 if P(v') >= DFI(v) then
15.                     pop and output the stack up to
                        and including (v,v')
16.                 P(v)=min(P(v),P(v'))
                    end
17.         else if v' =! father(v) then
18.                 P(v)=min(P(v),DFI(v'))
            end
        end of dfsb
```

Figura 7: Escriptura en pseudocodi de l'algoritme DFS per trobar blocs en grafs.

Destaquem els següents canvis respecte la rutina del DFS: a la línia 2 s'inicialitza la pila, i a la 10 hi apilem les arestes. S'inicialitza l'índex $P(v)$ en el seu valor màxim a la línia 7 i posteriorment s'actualitza a les línies 16 i 18 si es troba un valor menor.

A la següent proposició demostrem la condició per la qual es detecta un bloc.

Proposició 3.19. *Si en la realització del mètode DFS per blocs es troba v tal que té un fill v' complint $P(v') \geq DFI(v)$, aleshores les arestes apilades a la pila junt amb l'aresta $\overline{vv'}$ formen un bloc.*

Demostració.

Ho demostrem per inducció respecte el nombre de blocs. Si $B = 1$ no hi hap punts d'articulació, per tant l'únic v que compleix la condició és l'arrel. Suposem ara que es compleix per $B = n - 1$, i comprovem si es compleix per $B = n$.

Sigui v el primer vèrtex que es troba tal que $P(v') \geq DFI(v)$. Aleshores, la pila és plena i les arestes apilades sobre $\overline{vv'}$ són incidents amb vèrtexs descendents de v'. Llavors

v és un punt d'articulació que no té cap descendent que ho sigui, tals arestes incloent $\overline{vv'}$ han de formar un bloc.

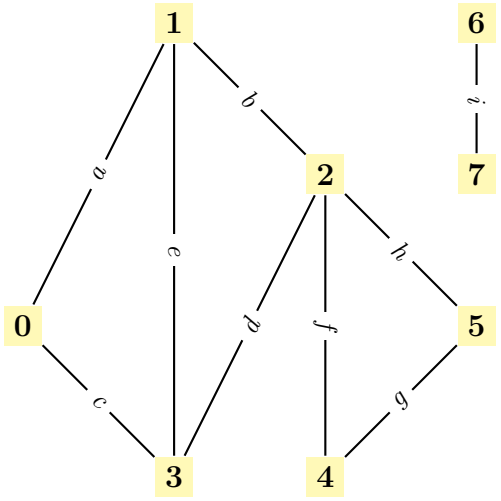
Un cop eliminades de la pila les arestes trobades que formen un bloc, ens queda un graf tal que $B = n - 1$ i es pot aplicar la hipòtesis d'inducció, quedant així demostrada la proposició.

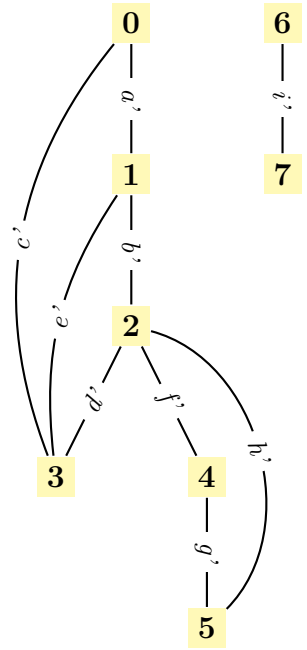
□

De la *Proposició 3.19* anterior en podem extreure el següent corol·lari, que ens assegura que la proposició inclou també els casos en què l'arrel de l'arbre DFS és alhora un punt d'articulació.

Corol·lari 3.20. *Si r és l'arrel d'un arbre expansiu del DFS aplicat a un graf que té almenys un bloc, llavors tots els fills v' de r compleixen $P(v') \geq DFI(r)$, ja que $DFI(r) = 0$. Per tant, cada cop que el mètode re-visita r, les arestes de la pila fins a $\overline{vr}$ formaran un bloc (on v és en aquest cas el vèrtex des del qual es re-visita r).*

Exemple 3.21. Mostrem el funcionament recursiu del mètode aplicant-ho a un exemple. El graf G escollit al que aplicarem el mètode és el que està representat a dalt a l'esquerra de la figura de l'exemple.





u	*DFI(u)*	*P(u)*
0	0	0
1	1	0
2	2	0
3	3	0
4	4	2
5	5	2
6	6	6
7	7	6

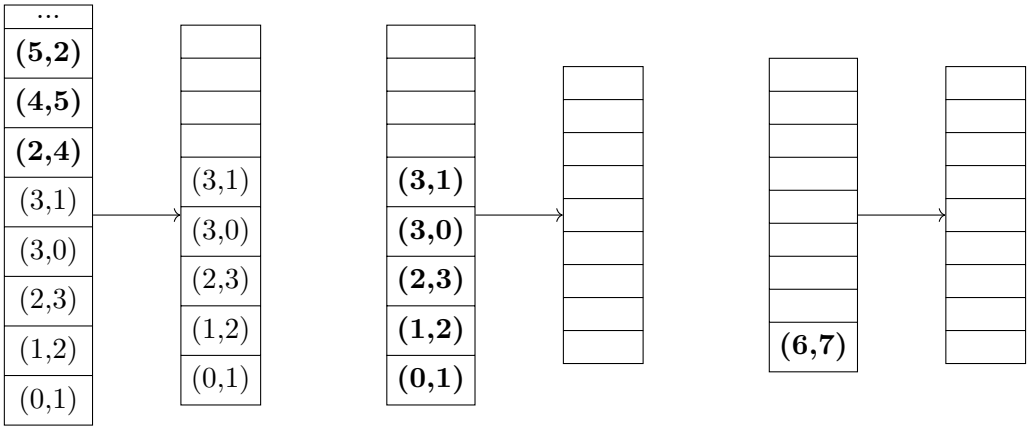


Figura 8: Exemple d'aplicació del mètode DFS per trobar blocs.

Primerament hem representat el graf original (a l'esquerra) i el bosc expansiu generat pel mètode DFS (a la dreta, les arestes rectes són les del bosc expansiu). També hem representat les arestes de tornada (back-edges) amb arestes corbades al dibuix de la dreta. Posteriorment, mostrem una taula amb els vèrtexs i els índexs $DFI(u)$ i $P(u)$ de cada vèrtex. Per últim, hem representat els tres cops que el codi detecta un bloc (quan es compleix la condició de la línia 14 del pseudocodi) i quines arestes de la pila formen tal bloc (en negreta). Veiem, en cada cas, quines arestes hi havia apilades i quines queden apilades a la pila després que es conformi cadascun dels blocs.

El primer cop que es detecta un bloc succeeix quan el codi està realitzant DFSB(2) un cop acabats DFSB(5) i DFSB(4), i estant encara dins DFSB(1) i DFSB(0). La condició es compleix ja que $P(4) = 2 \geq DFI(2) = 2$.

El segon cas succeeix quan el codi està en DFSB(0) un cop acabats DFSB(2) i DFSB(1), i en aquest cas es compleix la condició perquè $P(2) = 0 \geq DFI(0) = 0$.

L'últim cas succeeix quan el codi està en DFSB(6) un cop acabat DFSB(7), i en aquest cas es compleix la condició perquè $P(7) = 6 \geq DFI(6) = 6$.

3.4 Mètode DFS per trobar components forts de dígrafs

Tot seguit s'exposa l'extensió del mètode DFS duta a terme per tal de trobar components fortament connexos en dígrafs (per simplificar, d'ara en endavant *components forts*). L'element clau per discernir els components forts serà, en aquest cas, trobar les arrels dels components forts, un element que es definirà més endavant. L'acompliment de l'objectiu de l'extensió es basa fonamentalment en el *Teorema 3.22* que enunciarem tot seguit.

Teorema 3.22. *Sigui $D_i = (V_i, A_i)$ un component fort del dígraf D, i sigui $F = (V, E_F)$ un bosc expansiu de sortida generat pel DFS en el dígraf. Aleshores, $T_i = (V_i, E_i \cap E_F)$ és un arbre.*

Demostració.

Demostrem primer que dos vèrtexs $u, v \in V_i$, tenen un ancestre comú a V_i. Suposem sense pèrdua de generalitat, que $DFI(u) < DFI(v)$, al formar part del mateix component connex, existeix un camí orientat c des d'u fins a v. Definim un nou vèrtex w en c que compleix $DFI(w) < DFI(x)$ per tot vèrtex x en el camí c.

Centrant-nos en l'arbre expansiu de sortida, si iniciem el camí c des de u, un cop s'arriba a w, tots els vèrtexs pels que passarà c seran descendents de w. Això és perquè els arcs que van de descendents de w a vèrtexs que no ho són només poden ser arcs de tornada o arcs creuats, i per tant aniran a vèrtexs amb DFI més petits que els seus (en els arcs creuats s'utilitza el *Teorema 3.11*). Per tant, v ha de ser descendent de w, a l'igual que u ja que $DFI(w) \leq DFI(u) < DFI(v)$.

Sigui ara r_i l'arrel de l'arbre que conté tots els vèrtexs de V_i. Sigui $x \in V_i$ i y un vèrtex al camí orientat que connecta r amb x. Llavors, com que existeix un camí des de r a y, i un altre des de y a r passant per x, podem concloure que $y \in V_i$ cosa que acaba la demostració.

□

Notació 3.23. *Seguint la notació del Teorema 3.22 anterior, anomenem arrel del component fort G_i a l'arrel de l'arbre T_i, i el notem per r_i.*

De la mateixa manera que s'ha realitzat amb els índexs $P(v)$ i $DFI(v)$, ens cal definir-ne un de nou per facilitar computacionalment la cerca de components forts. Definim $Q(v)$ de la següent forma:

Definició 3.24. Notarem d'ara en endavant per $Q(v)$ el mínim entre $DFI(v)$ i $DFI(v')$ tal que $\overrightarrow{xv'} \in B_1$ o $\overrightarrow{xv'} \in C$ essent x un descendent de v i on l'arrel del component fort que conté v' és un ancestre de v. Per tant:

$$Q(v) = \min \begin{cases} DFI(v) \\ Q(v') \parallel v' \text{ és un fill de } v \\ DFI(v') \parallel \overrightarrow{vv'} \in B_1 \cup C \text{ on l'arrel del component fort que conté } v' \\ \qquad\qquad\qquad\qquad \text{és un ancestre de } v. \end{cases}$$

Observació 3.25. Evidentment, es compleix que $Q(v) \leq DFI(v)$.

Proposició 3.26. *Donat un dígraf $D = (V, A)$, v és l'arrel d'un component fort si i només si $Q(v) = DFI(v)$.*

Demostració.

⇒) Demostrem per contradicció. Suposem que $Q(v) < DFI(v)$. Llavors existeix un vèrtex v' tal que $DFI(v') < DFI(v)$. Com que $DFI(r) < DFI(v')$, tenim que $DFI(r) < DFI(v)$, però r i v pertanyen al mateix component fort perquè hi ha un camí de r a v, i un de v a r passant per v'. Llavors, v no pot ser l'arrel d'un component fort, contradicció, i al ser $Q(v) \leq DFI(v)$, es conclou que $Q(v) = DFI(v)$

⇐) Demostrem el contra-recíproc «si v no és l'arrel d'un component fort, llavors $Q(v) < DFI(v)$», i ho fem per contradicció. Suposem que $Q(v) = DFI(v)$, llavors no existeix v' definit com en $Q(v)$ tal que $DFI(v') < DFI(v)$. Com que v no és arrel, ho és algun vèrtex r. Com que existeix un camí orientat c de v a r tal que el primer vèrtex no és descendent de v. Anomenem v' a aquest vèrtex, clarament r i v' pertanyen a un mateix component fort. L'arc de c incident amb v' només pot ser de B_1 o de C, llavors $DFI(v') < DFI(v)$, cosa que és una contradicció.

□

La manera de reconèixer components forts d'un dígraf en el nostre algoritme consistirà doncs a trobar les arrels dels components forts. Del *Teorema 3.22* i del fet que una arrel no pot ser descendent d'una altra, deduïm que si $DFI(r_i) > DFI(r_j)$, aleshores el component fort G_i està format pels descendents de r_i que alhora no són descendents de $r_1, r_2, ..., r_{i-1}$.

Altra vegada, dotarem a l'algoritme d'una pila o *stack* on anirem apilant els vèrtexs fins que algun d'ells validi la condició d'arrel de component fort. Aleshores es buidaran de la pila aquells vèrtexs que són descendents de l'arrel trobada i els etiquetarem igual per indicar que pertanyen a un mateix component fort.

Exemple 3.27. Un cop vistos els detalls que s'afegeixen per tal d'estendre l'algoritme DFS perquè trobi components forts en dígrafs, a la *Figura 9* mostrem en pseudocodi el desenvolupament de la nova rutina. Usarem de base el pseudocodi de la *Figura 6* i hi afegirem les línies de codi corresponents.

```
procedure DFSSC(G)
1. set i:=0
2. empty the stack
3. for all v in V do DFI(v)=-1 and stacked(v)=false
4. while for some u, DFI(u)=-1 do dfssc(u)
5. procedure dfssc(v)
       begin
6.     DFI(v)=i
7.     Q(v)=DFI(v)
8.     i=i+1
9.     put v on stack and set stacked(v)=true
10.    for all v' in A(v) do
11.        if DFI(v')=-1 then
                   begin
12.                dfssc(v')
13.                Q(v)=min(Q(v),Q(v'))
                   end
14.        else if DFI(v')<DFI(v) and stacked (v') then
15.                Q(v)=min(Q(v),DFI(v'))
16.    if Q(v)=DFI(v) then
17.        pop and output the stack up to and including v
18.        for each popped vertex u rested stacked(u)=false
       end of dfssc
```

Figura 9: Escriptura en pseudocodi de l'algoritme DFS per trobar components forts en dígrafs.

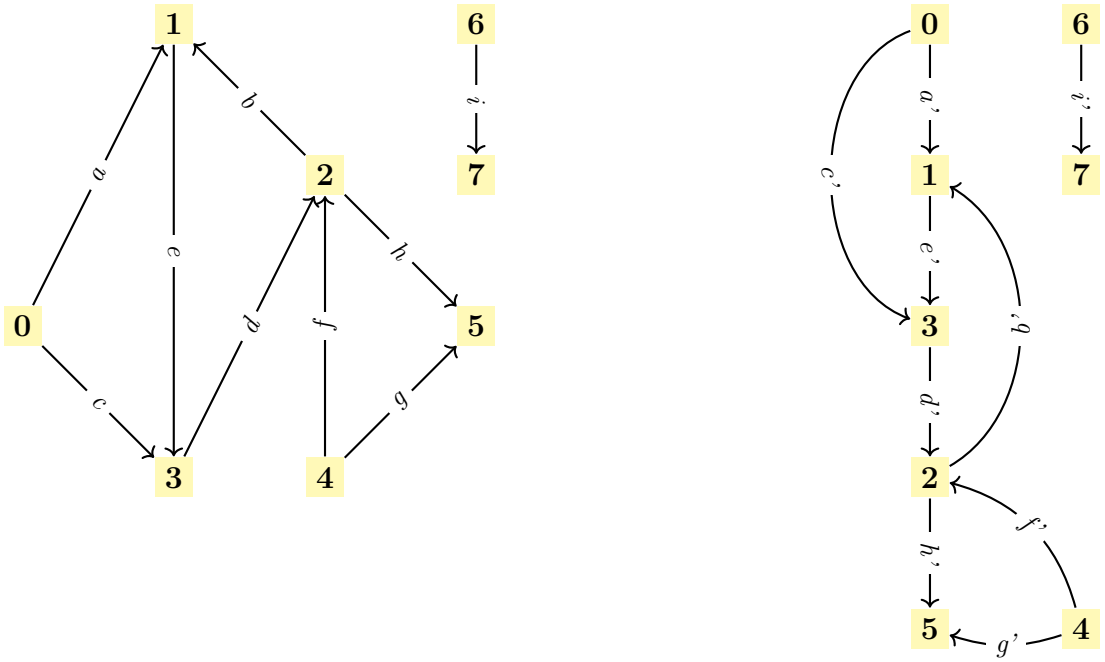
Es poden fer les consideracions següents del pseudocodi mostrat. Primerament, notem que afegim un caràcter booleà a la línia 3 que ens indicarà si un vèrtex v està o no a la pila. A la línia 7 inicialitzem la definició de $Q(v)$ en el seu valor màxim $DFI(v)$, i a les línies 13 i 15 l'actualitzem al seu valor mínim si és possible en cada cas. La segona actualització possible de $Q(v)$, només es dóna si compleix la condició de la línia 14 que invalida que (v, v') pugui ser un arc d'anada.

Cal observar que a la línia 14 $DFI(v') \neq -1$, per tant v' ja s'ha visitat, i està apilat. Llavors l'arrel del possible component fort que conté v' hauria d'estar ja dins la pila.

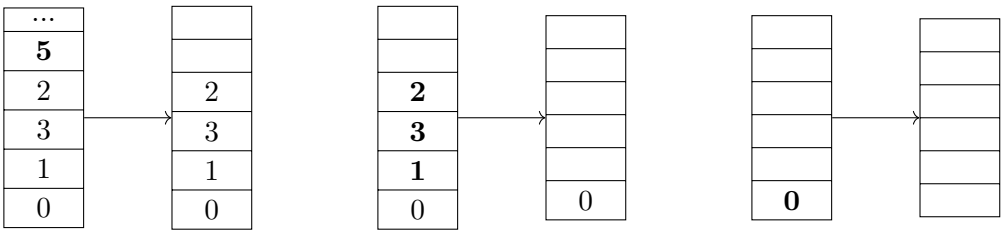
Les arrels dels components forts es detecten a la línia 16, seguint la *Proposició 3.26*. Aleshores, d'acord amb el *Teorema 3.22*, els vèrtexs apilats fins a l'arrel trobada formen un component fort (línia 17) i es canvia el valor booleà dels vèrtexs que conformen tal component fort (línia 18).

Per últim, destaquem que el nombre de vegades que s'inicialitza la rutina *dfssc* (en minúscula) a causa de la línia 4 (no a la línia 12) es correspon amb el nombre d'arbres del bosc expansiu de sortida generat pel DFS.

Exemple 3.28. Mostrem el funcionament recursiu del mètode aplicant-ho a un exemple. El dígraf D escollit al que aplicarem el mètode és el que està representat a dalt a l'esquerra de la figura de l'exemple



u	*DFI(u)*	*Q(u)*
0	0	0
1	1	1
2	3	1
3	2	1
4	5	5
5	4	4
6	6	6
7	7	7



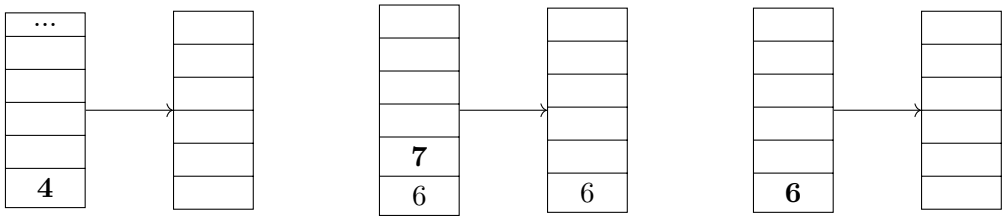


Figura 10: Exemple d'aplicació del mètode DFS per trobar components fortes.

Primerament hem representat el dígraf original (a l'esquerra) i el bosc expansiu generat pel mètode DFS (amb arcs rectes, a la dreta). Hem representat també els arcs que no són del bosc expansiu (és a a dir, els arcs d'anada, de tornada i creuats) amb arcs corbats. Posteriorment, mostrem una taula amb els vèrtexs i els índexs $DFI(u)$ i $Q(u)$ de cada vèrtex. Per últim, hem representat els sis cops que el codi detecta un component fort (es compleix la condició de la línia 16 del pseudocodi) i quins vèrtexs de la pila formen tal component fort (en negreta). Veiem, en cada cas, quins vèrtexs hi havia apilats i quins queden apilats a la pila després que es conformi cadascun dels components forts.

Observem que dels sis components forts trobats, només un component té més d'un vèrtex, i els altres cinc són un propi vèrtex. Per tant, només en un component caldrà identificar d'igual manera els arcs que en formen part. Així mateix, l'algoritme ha detectat tres arbres del bosc expansiu, amb arrels 0, 4 i 6 respectivament.

El primer cop que es detecta un component fort succeeix quan el codi està realitzant DFSSC(5) estant encara dins DFSSC(2), DFSSC(3), DFSSC(1) i DFSSC(0). La condició es compleix ja que $Q(5) = 4 = DFI(5)$.

El segon cas succeeix quan el codi està en DFSSC(1) un cop acabats DFSSC(2) i DFSSC(3), i estant encara dins de DFSSC(0). En aquest cas es compleix la condició perquè $Q(1) = 1 = DFI(1)$.

El tercer cas succeeix quan el codi està en DFSSC(0) un cop acabat DFSSC(1). Es compleix la condició $Q(0) = 0 = DFI(0)$.

El quart cas succeeix quan el codi està en DFSSC(4), on hi entra a causa de la línia 4 del pseudocodi. Es compleix la condició $Q(4) = 5 = DFI(4)$.

El cinquè cas succeeix quan el codi està en DFSSC(7) estant encara dins DFSSC(6), després d'entrar-hi a causa de la línia 4 del pseudocodi. Es compleix la condició perquè $Q(7) = 7 = DFI(7)$.

Finalment, l'últim cas succeeix quan el codi està en DFSSC(6) un cop acabat DFSSC(7), i en aquest cas es compleix la condició perquè $Q(6) = 6 = DFI(6)$.

4 Implementació dels mètodes de cerca

En aquesta secció s'explica més detalladament com s'han implementat els diferents algoritmes descrits en la secció anterior. Com s'ha explicat a l'inici del treball, un dels objectius era implementar els algoritmes en un programa que detectés les característiques principals d'un graf o dígraf, així com els blocs i els components fortament connexos en cada cas. S'ha utilitzat el llenguatge de programació $C++$ i hem fet ús de l'entorn de programació *Microsoft Visual Studio* per posar en pràctica tal objectiu.

El programa consta de diferents arxius font principals, que podem considerar que defineixen una divisió teòrica del programa en parts diferenciades. En les subseccions següents es detallen les característiques més remarcables del programa atenent a aquesta divisió.

4.1 Entrada i sortida de dades

Abans que altra cosa, considerem convenient donar la informació de quina és la correspondència entre els tipus de variables que anirem trobant més endavant i la seva definició. Totes aquestes variables vindran definides en un *Header File*. A la *Figura 11* de continuació mostrem el glossari anunciat.

tipus	**definició**
vertex index arc edge length tree block type	`unsigned int`
coord	`double`
point	`pair<coord,coord>`
points	`vector<point>`
vip	`pair<vertex,index>`
graph digraph	`vector<vector<vertex>>`
edges	`map<vip,edge>`
arcs	`map<vip,arc>`

Figura 11: Les tipologies de les diferents variables del programa.

4.1.1 Entrada de dades

La informació principal que necessitem que llegeixi el programa són els grafs o els dígrafs. Tota la informació necessària d'entrada de les dades dels grafs o dígrafs es dona per mitjà de fitxers de text, d'extensió «.dat». En aquests fitxers s'inclou la informació següent ordenada per línies.

A la primera línia s'introdueix el nombre de vèrtexs junt amb el d'arestes o arcs, és a dir, ν i α. A les línies següents es llisten els vèrtexs seguits de dues coordenades (tipus coord) que els posicionaran a la interfície de visualització. Per últim, es llisten les arestes o arcs, seguits dels dos vèrtexs que uneix. En el cas que es vulgui tractar amb dígrafs l'entrada serà la mateixa, però el programa llegirà el primer vèrtex com la cua de cada arc i el segon com el cap. La numeració tant dels vèrtexs com dels arcs o arestes començarà per 0.

Fixem-nos ara en com tractem les dades obtingudes pels propòsits del programa. La primera acció que es realitza al llegir el graf és guardar ordenadament les coordenades de cada vèrtex en un vector `points` que, com hem vist a la *Figura 11*, és un `vector<pair<coord,coord>>`.

Seguidament i en el cas dels grafs, el que farem serà generar la seva llista d'adjacències. És per això que es tracten els grafs com a estructures de classe `vector<vector<vertex>>`, que resulten en què a cada vèrtex (del vector de vèrtexs del graf) se li associa un vector de vèrtexs adjacents a aquest. Així doncs, el que fa el procés és recórrer les següents línies del fitxer ≪.dat≫, on es troben les arestes etiquetades junt amb els dos vèrtexs que uneix, i es va afegint cada vèrtex adjacent a la llista d'adjacents de l'altre i viceversa, per mitjà d'instruccions `push_back`.

Una altra tasca que es realitza alhora que es van incloent els vèrtexs a la llista d'adjacències és associar als dos vèrtexs afegits una mateixa etiqueta d'aresta. Per fer-ho usarem una variable tipus ≪edges≫, que com hem vist, és un `map`. Un `map` és una variable que emmagatzema elements als quals se'ls associa un altre element, en el cas d'≪edges≫ són parelles vèrtex-índex als que se'ls associa una aresta. Les parelles vèrtex-índex estan formades de la manera següent: si s'està realitzant l'acció d'afegir v a la llista d'adjacències d'u, llavors la parella en qüestió serà el vèrtex u junt amb la posició que ocupa v en la llista dels adjacents d'u. Aleshores, a les dues parelles (u i índex, i v i índex) se les associa la mateixa etiqueta d'aresta per mitjà del `map`.

El cas dels dígrafs és lleugerament diferent. Tal i com s'ha vist a la subsecció 2.4, existeixen dues llistes d'adjacències diferents: la d'entrada i la de sortida. Nosaltres generarem les dues en una de sola, que podem anomenar *llista d'entrada i sortida*. L'estructura amb què tractarem els dígrafs serà doncs també `vector<vector<vertex>>`.

El que canviarà respecte el cas dels grafs és que, a l'hora d'etiquetar els arcs, si i és l'etiqueta d'un arc $\overrightarrow{uv}$, aleshores creem un arc fictici oposat a aquest que etiquetem amb el nombre $i + an$, essent an és el nombre total d'arcs ($an = \alpha(D)$). Així doncs, tots els arcs etiquetats amb nombres més grans o igual a an, seran arcs ficticis, i els vèrtexs cap a on apuntaran tals arcs ficticis seran els veïns d'entrada del vèrtex de la cua. El vèrtex cap a on apunti un arc real qualsevol, que haurà de ser un dels arcs etiquetats amb nombres més petits que an, serà òbviament un veí de sortida del vèrtex de la cua. Vegem-ho més fàcilment amb un exemple.

Exemple 4.1. Sigui $D = (V, A)$ el dígraf tal que $V = (0, 1, 2)$ i $A = (\overrightarrow{01}, \overrightarrow{12})$, representat a dalt de la *Figura 12*.

El dígraf original D té un total de dos arcs. A baix es mostra D amb l'afegit dels arcs ficticis a' i b'. Si a l'hora de llistar els arcs $a = 0$ i $b = 1$, aleshores $a' = 0 + 2 = 2$ i $b' = 1 + 2 = 3$, i es compleix que $a', b' \geq an$. Aleshores, el vèrtex on apunta a és 1, que és el veí de sortida de 0, i el vèrtex on apunta a' és 0, que és el veí d'entrada de 1.

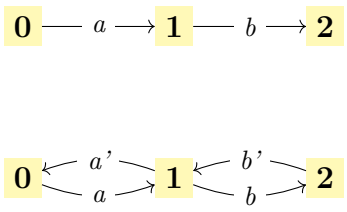


Figura 12: Exemplificació d'arcs ficticis per a la generació de la llista d'entrada i sortida.

Això ens permetrà discernir dins de la llista d'entrada i sortida quins vèrtexs seran veïns d'entrada i quins veïns de sortida, perquè als veïns d'entrada els associarem mitjançant el `map` ≪arcs≫, un arc amb etiqueta més gran o igual a *an*, i als de sortida, un arc amb etiqueta més petita que *an* (que correspondrà amb l'etiqueta real del seu arc). Per tant en el cas dels dígrafs no associarem un mateix arc a les parelles vèrtex-índex (l'índex es defineix igual que al cas dels grafs).

4.1.2 Sortida de dades

Hem implementat al programa dues formes diferents per donar sortida als resultats. La primera es realitza mitjançant l'ús de fitxers de text, que en aquest cas hem anomenat ≪.out≫. A partir de vàries funcions d'escriptura (una per a cada algoritme de cerca) que contenen successives crides `fstream`, es van escrivint al fitxer de text les informacions trobades en cada cas.

D'altra banda, també hem incorporat una funció que hem anomenat *digraph_tex* que escriu, fent servir també crides `fstream`, el codi adequat per representar gràficament els dígrafs a LaTeX utilitzant el paquet *TikZ*.

Totes les funcions explicades en aquesta subsecció 4.1 estan incloses dins l'arxiu font que hem anomenat *In-Out*.

4.2 Implementació dels algoritmes

Els algoritmes s'implementen com a funcions a l'arxiu principal, anomenat *Compute*, i es corresponen amb els que s'han explicat a la secció 3 anterior. Donarem algun detall més d'implementació dels algoritmes però sense entrar altre cop en els mètodes, que ja s'han explicat anteriorment a partir dels pseudocodis. Per començar, llistem les funcions que trobem a l'arxiu principal:

- *BFS*: executa l'algoritme BFS.
- *dfs*: la funció recursiva de l'algoritme DFS.
- *DFS*: executa l'algoritme DFS, conté una crida interna a la funció *dfs*.
- *dfs_block*: la funció recursiva de l'algoritme DFS per trobar blocs.
- *DFS_block*: executa l'algoritme DFS per trobar blocs, conté una crida interna a la funció *dfs_block*.
- *dfs_SC*: la funció recursiva de l'algoritme DFS per trobar components fortament connexos.

- *DFS_SC*: executa l'algoritme DFS per trobar components fortament connexos, conté una crida interna a la funció *dfs_SC*.

Una altra funció que trobem a l'arxiu i que es crida des de totes les funcions no recursives llistades anteriorment, és la funció *Search_Tree_Root*. En el cas dels dígrafs la tria del vèrtex que podem usar com a arrel r, ha de satisfer la condició que $g^+(r) > 0$. Tal funció ens assegura que escollim un vèrtex adequat per fer d'arrel. En el cas dels grafs ens retorna el següent vèrtex no visitat.

També trobem a l'arxiu una funció que hem anomenat *Digraph_Compute* que ens fa la classificació de les arestes del bosc expansiu generat pels mètodes de cerca després d'escollir quin mètode s'aplica. En el cas dels grafs és tan senzill com que fer que l'algoritme discerneixi les arestes que no són de l'arbre de les que sí que ho són. En el cas dels arcs d'un dígraf l'algoritme aplica les propietats que s'expliquen a la classificació d'arcs de 3.2.1.

4.3 Simulador

Un dels objectius era dotar el programa d'un simulador per tal que la presentació dels resultats de l'aplicació dels algoritmes fos més visual i interactiva. El presentem a continuació a partir d'una captura de pantalla d'aquest, en un moment en què encara no s'ha executat cap algoritme.

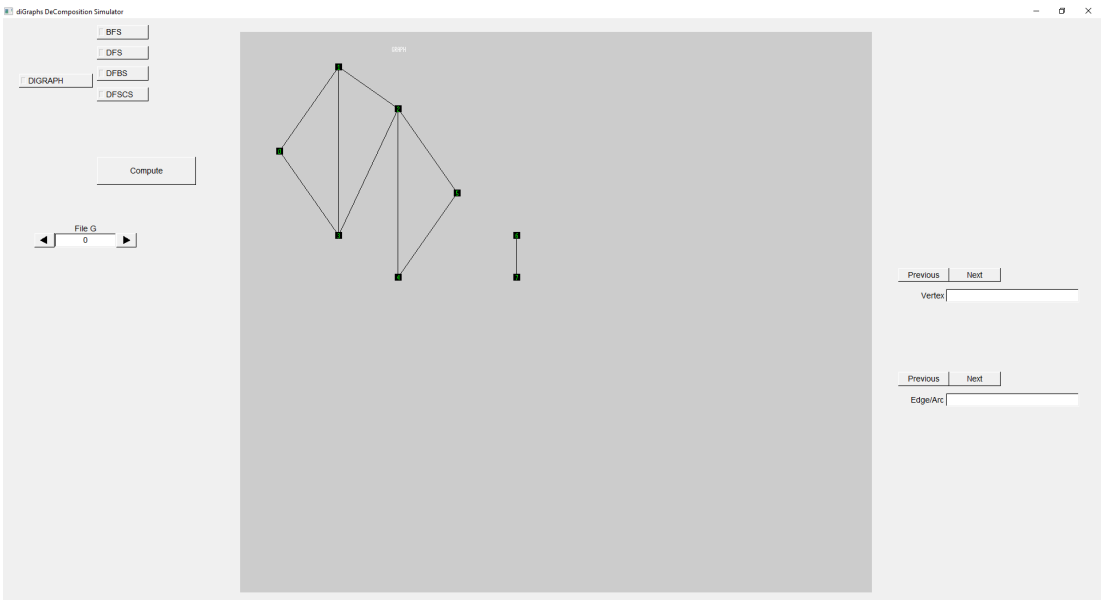


Figura 13: Captura del simulador

Podem fer una diferenciació de dues parts del simulador. Fixant-nos en la *Figura 13*, es pot observar d'una banda l'àrea de visualització (de color de fons gris fosc) i d'altra banda l'àrea de control (de color de fons gris clar). Tot seguit expliquem breument com s'ha implementat cadascuna de les parts.

Per a l'àrea de visualització s'han utilitzat funcions de la biblioteca OpenGL (Open Graphics Library). Les funcions d'aquesta àrea són les relacionades amb el dibuix dels resultats, i estan incloses dins l'arxiu font que hem anomenat *View*. Per exemple, la funció

més rellevant programada dins d'aquest arxiu és la referent a com generar en pantalla els arbres produïts per cada extensió de l'algoritme DFS.

Per a l'àrea de control s'han utilitzat funcions de la llibreria Flight Light Toolkit (FLTK). A través de les funcions de l'àrea de control podem escollir quin mètode s'executa i si volem que es tractin les dades com si fos un graf o com si fos un dígraf (a dalt a l'esquerra del simulador). També permeten la tria del graf (o dígraf) dels fitxers introduïts que es computa i s'implementa un botó per activar la computació de l'algoritme seleccionat (tot plegat a la part esquerra del simulador). Finalment, a la part dreta podem controlar quina aresta o arc així com quin vèrtex volem seleccionar (les seleccions es visualitzaran en color blanc). Totes aquestes funcions estan incloses a l'arxiu font que hem anomenat *Control*.

Per últim, també s'ha usat la llista d'eines GLUT, dins de la biblioteca ja esmentada OpenGL, per tal d'assegurar que el control del programa pugui adaptar-se a qualsevol sistema operatiu. Les funcions que hi tenen a veure s'han inclòs a l'arxiu font que hem anomenat *Glut*.

5 Resultats

En aquesta secció mostrem com es visualitzen els resultats a l'entorn del simulador després d'executar cadascun dels algoritmes. Utilitzarem exemples fets a mida, per tal que la cerca dels blocs i els components fortament connexos sigui exemplificadora de l'actuació de l'algoritme.

Hem preferit treballar amb dígrafs en tots els casos excepte en l'extensió de buscar blocs (ja que aquesta extensió només s'aplica a grafs no dirigits) perquè la classificació d'arcs del dígraf resulta més interessant.

Comencem doncs mostrant el resultat de l'aplicació del mètode BFS en un dígraf a la *Figura 14*.

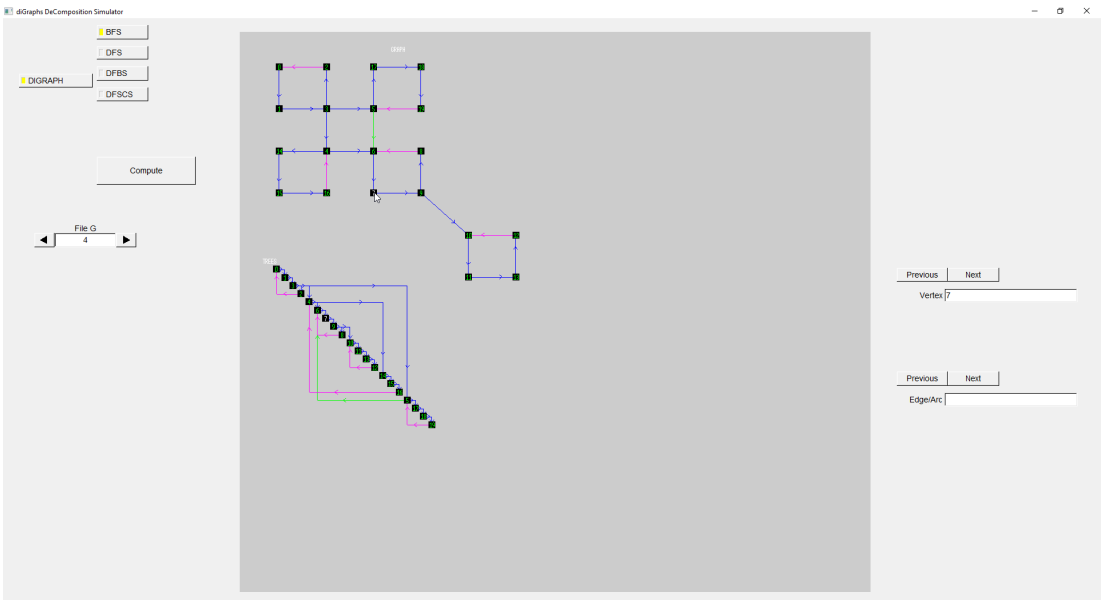


Figura 14: Visualització dels resultats aplicant el mètode BFS.

Podem observar la formació del bosc expansiu de sortida (en aquest cas arbre expansiu de sortida) a sota de la representació del dígraf original. Els arcs que en formen part són els que s'han pintat de color blau. A la mateixa representació hi hem afegit els arcs que no formen part de l'arbre expansiu: els arcs pintats de magenta són els arcs de tornada i els arcs pintats de verd són els arcs creuats. En aquest cas, degut a la idiosincràsia pròpia del mètode BFS, no trobem arcs d'anada.

Aprofitem per ensenyar també una de les particularitats del simulador: quan apropem el cursor a un vèrtex, el vèrtex 7 en el cas de la *Figura 14*, el vèrtex s'acoloreix de color blanc (en la representació del dígraf i de l'arbre dirigit) i ens apareix també a la secció de control dels vèrtexs.

A continuació observem el resultat de l'aplicació del mètode DFS en un dígraf mitjançant la *Figura 15*.

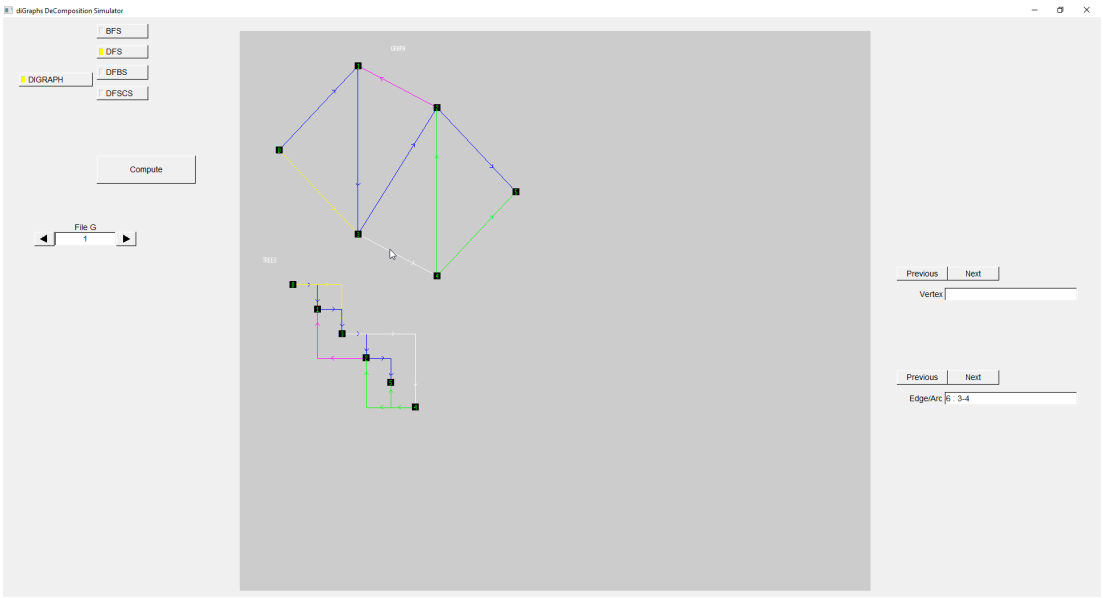


Figura 15: Visualització dels resultats aplicant el mètode DFS.

En aquest cas es mantenen els mateixos colors per indicar la classificació d'arcs, afegint el color groc que indica l'arc d'anada existent $\overrightarrow{03}$. Destaquem que a la representació de sota on s'observa l'arbre dirigit expansiu, els arcs de l'arbre i d'anada es dibuixen sempre per sobre de la diagonal de vèrtexs, mentre que els arcs de tornada i creuats per sota.

Aprofitem per observar també que, com en el cas anterior, quan s'apropa el cursor a un arc, aquest s'acoloreix de color blanc en totes les representacions. A més a més apareix a la secció de control dels arcs on també podem veure la seva etiqueta.

A continuació observem el resultat de l'aplicació del mètode DFS per trobar blocs en un graf mitjançant la *Figura 16*. El graf escollit és isomorf al de l'*Exemple 3.21*.

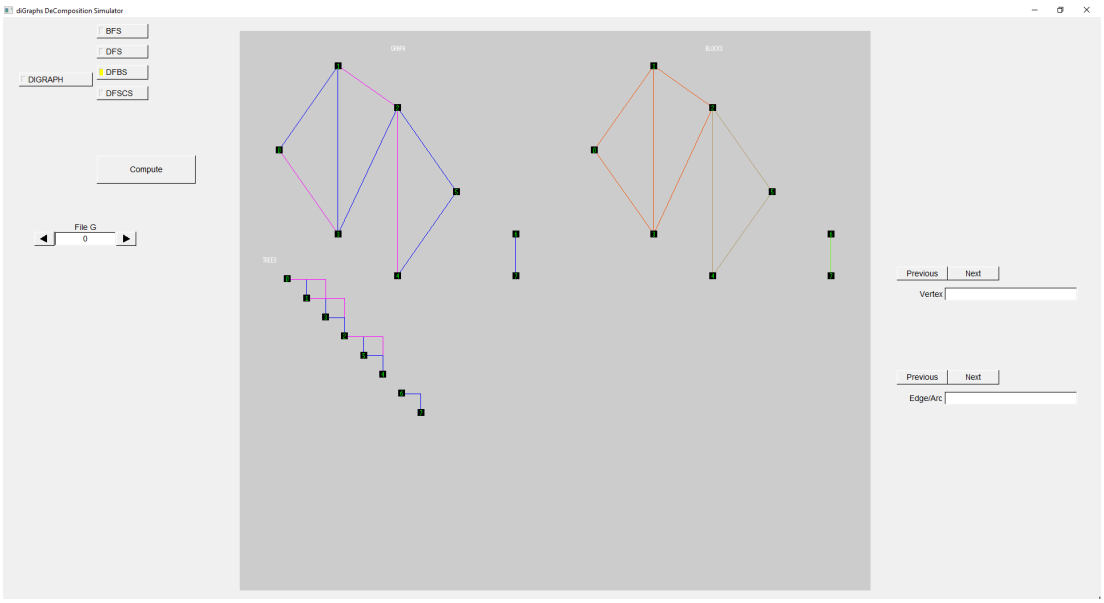


Figura 16: Visualització dels resultats aplicant el mètode DFS per trobar blocs.

Com s'ha explicat, aquest és l'únic exemple de resultat on s'ha computat un algoritme a un graf. Fixem-nos que l'arbre expansiu DFS es segueix representant a sota del graf original. En aquest cas, les arestes pintades de blau són les arestes de l'arbre mentre que les pintades de magenta corresponen a les arestes de tornada.

A la dreta del graf original apareix el resultat de l'extensió de l'algoritme per trobar blocs. Les arestes que formen part d'un mateix bloc es pinten d'un mateix color escollit aleatòriament (si els colors no es diferenciessin suficient computant de nou el programa podrien canviar de color). En el nostre exemple, l'algoritme ha detectat 3 blocs. Un bloc està format per les arestes $\overline{01}$, $\overline{12}$, $\overline{23}$, $\overline{30}$ i $\overline{13}$, les arestes del qual queden pintades en aquest cas de color vermell. Un altre bloc està format per les arestes $\overline{24}$, $\overline{45}$ i $\overline{52}$, i en aquest cas les arestes queden pintades de color marró. L'últim bloc està format per una única aresta, l'aresta $\overline{67}$, i en aquest cas queda pintada de color verd.

Així doncs, podem concloure que els resultats coincideixen efectivament amb els que s'han obtingut a l'exemple citat de la part teòrica.

Per últim observem el resultat de l'aplicació del mètode DFS per trobar components fortament connexos en un dígraf exposat a la *Figura 17*. En aquest cas, el dígraf és isomorf al de l'*Exemple 3.28*.

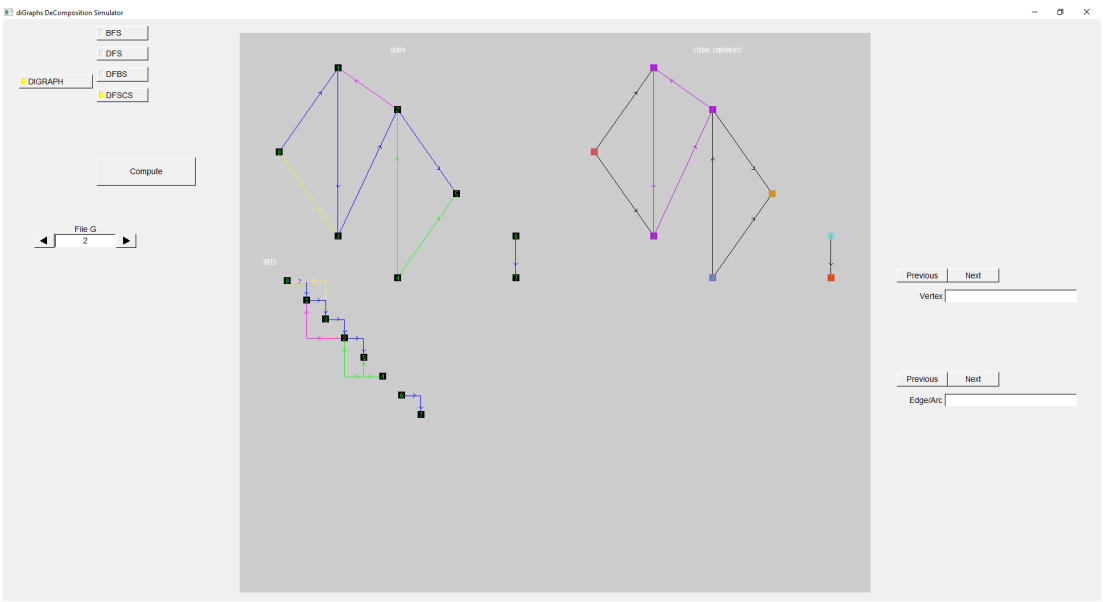


Figura 17: Visualització dels resultats aplicant el mètode DFS per trobar components fortament connexos.

Com en tots els mètodes aplicats, l'arbre expansiu DFS segueix representant-se sota del dígraf original. Els arcs se segueixen pintant en aquestes dues representacions seguint el codi explicat anteriorment que ens indica la tipologia de cadascun d'ells. En aquest cas, apareix també a la dreta del dígraf original, els resultats corresponents a la cerca de components fortament connexos.

Fixem-nos que és l'únic dels quatre algoritmes en què es pinten també els vèrtexs del dígraf. Tal i com s'explica a la subsecció 3.4, el que determina primerament un component fortament connex és quins vèrtexs hi pertanyen. Per això ens cal acolorir-los en aquest cas.

En l'exemple posat, distingim clarament un component connex format pels vèrtexs

$V' = 1, 2, 3$ pintats de magenta i, en conseqüència, pels arcs $\overrightarrow{13}$, $\overrightarrow{32}$ i $\overrightarrow{21}$ que els uneixen, que també estan pintats de magenta. La resta de vèrtexs formen cadascun d'ells un component fortament connex format per ell mateix. Es pot advertir donat que cada vèrtex està pintat d'un color diferent. Per últim, es pot observar que la resta d'arcs estan pintats de color negre i, per tant, no formen part de cap component fortament connex, cosa que concorda amb el que s'ha explicat.

Així doncs, també podem concloure que en aquest cas, els resultats també coincideixen amb els que s'han obtingut a l'exemple citat de la part teòrica.

6 Aplicacions

6.1 Aplicació del mètode DFS per a trobar blocs en el mapa del metro de Barcelona

El darrer dels objectius mencionats del treball és el d'aplicar algun dels mètodes exposats per trobar-ne les característiques a algun exemple més tangible. Ho hem realitzat provant d'aplicar el mètode DFS per trobar blocs en una modelització de les línies de la xarxa de metro de Barcelona a data actual (juny de 2019). No s'han inclòs al model les línies metropolitanes de FGC. Per deixar-ho clar, s'ha inclòs a la modelització les línies L1, L2, L3, L4, L5, L9 (nord i sud), L10 (nord i sud) i L11.

Per fer més clara la visualització, hem modificat els valors de certes funcions de l'arxiu de visualització per tal que es mostri únicament el resultat de l'algoritme DFS per trobar blocs, de forma més gran possible. També s'ha retocat la visualització de les arestes per tal de dibuixar-les més gruixudes.

A la *Figura 18* ensenyem una captura de pantalla del resultat del simulador un cop s'ha computat el mètode a la modelització establerta i amb els canvis de visualització mencionats.

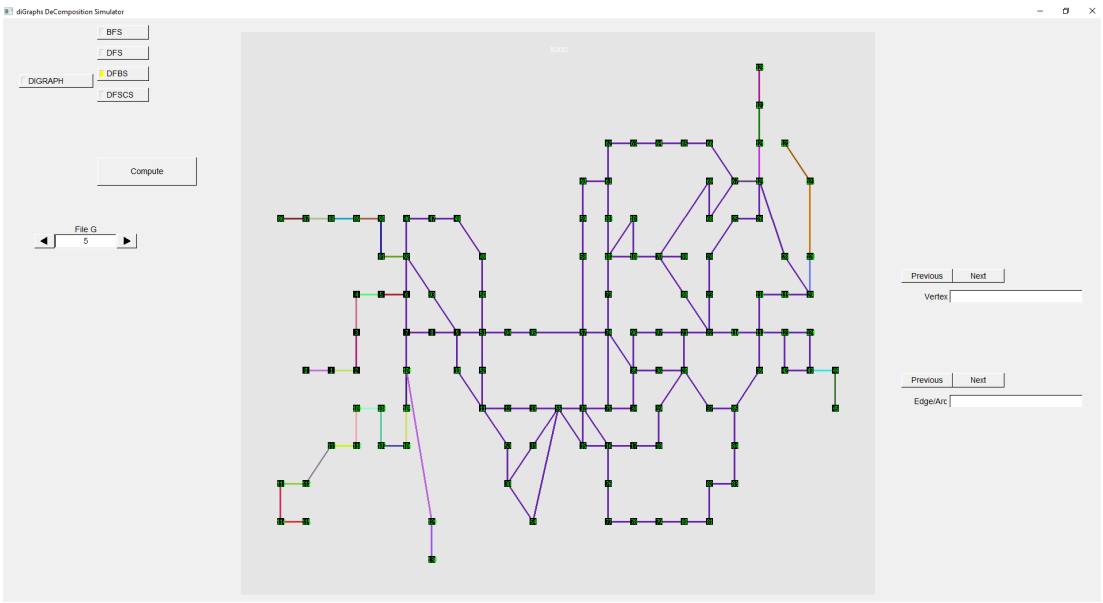


Figura 18: Visualització al simulador de l'aplicació del mètode DFS per trobar blocs a la xarxa de metro de Barcelona.

S'observa clarament l'existència d'un bloc relativament gran, format per molts dels vèrtexs i les arestes del model, pintades de color indi. Veiem que, de fet, és l'únic bloc format per més d'una aresta. La resta de blocs es formen a les «fulles» que pengen d'aquest bloc principal.

Les conclusions que en podem extreure, seguint els que s'explica a la introducció del treball, és que per estudiar certes propietats del graf que formen les línies de metro de Barcelona (com ara la planaritat del graf) podem restringir-nos a estudiar el subgraf format pel bloc principal que s'ha trobat, i obviar-ne les fulles. Efectuar aquest procés s'anomena sovint «podar les fulles».

També podem dir que el fet d'haver trobat un únic bloc de més d'una aresta tan gran és tan esperable com desitjable: això significa que la xarxa de metro no té punts d'articulació importants. Si en tingués, una afectació en aquella estació desconnectaria la xarxa en dos components connexos diferenciats.

Per acabar, a partir de l'estudi realitzat es pot fer el raonament que tot i no tenir punts d'articulació importants, la xarxa té massa punts d'articulació. Una solució que podria alleugerir el problema seria connectar mitjançant noves línies de metro les estacions més perifèriques, que és on es troben la majoria de les estacions que són punts d'articulació en el nostre model. Aquest és, en essència, l'objectiu principal (o un d'ells) de la implementació completa de les línies 9 i 10 (amb el tram central, s'entén). Ambdues línies esdevindrien les primeres línies perifèriques de la xarxa de metro de Barcelona.

7 Conclusions

La realització d'aquest treball ha permès una recopilació del marc teòric on s'engloba el projecte. També ha permès la construcció d'una eina informàtica per trobar blocs i components fortament connexos que mostra els resultats d'una manera senzilla i entenedora.

Durant la consecució del projecte, hem anat recollint idees per a una possible futura ampliació del programa. Una d'elles seria incloure a la interfície de visualització les dades per escrit del número de blocs o components forts així com la seva composició. Pel que fa al codi, es podria dotar d'alguna o algunes funcions que permetessin decidir quin o quins vèrtexs s'escullen com a arrels des dels quals aplicar posteriorment els mètodes de cerca. Finalment, arran de l'estudi fet a la xarxa del metro de Barcelona, hem convingut que podríem fer una ampliació del programa per tal que trobés també cicles en grafs no dirigits.

També i en relació amb l'anterior paràgraf, una altra conclusió a què s'ha arribat és que l'àrea que hem decidit estudiar té un camp molt ampli de treball, tant teòric com d'aplicació pràctica.

Com a conclusió final del treball, podem afirmar que s'han acomplert els objectius que ens havíem proposat a l'inici d'aquest treball.

Referències

[1] Bondy, J. A.; Murty U.S.R.: *Graph Theory*, Springer, New York, 2008.

[2] Gibbons, A.: *Algorithmic graph theory*, Cambridge University Press, Cambridge, 1985.

[3] Gross, J. L., Yellen, J.:*Handbook of Graph Theory*, CRC Press, Boca Raton (Florida), 2004.

[4] Harary, F.: *Graph Theory*, Addison-Wesley, Reading (Massachusetts), 1994.

[5] McHugh, J. A., *Alogirthmic Graph Theory*, Prentice Hall, Englewood Cliffs (Nova Jersey), 1990.

[6] Jungnickel, D.: *Graphs, Networks and Algorithms*, Springer, Heidelberg, 2005.

[7] Soria, J.: *Graphs*, Apunts del curs de Grafs de la Universitat de Barcelona.