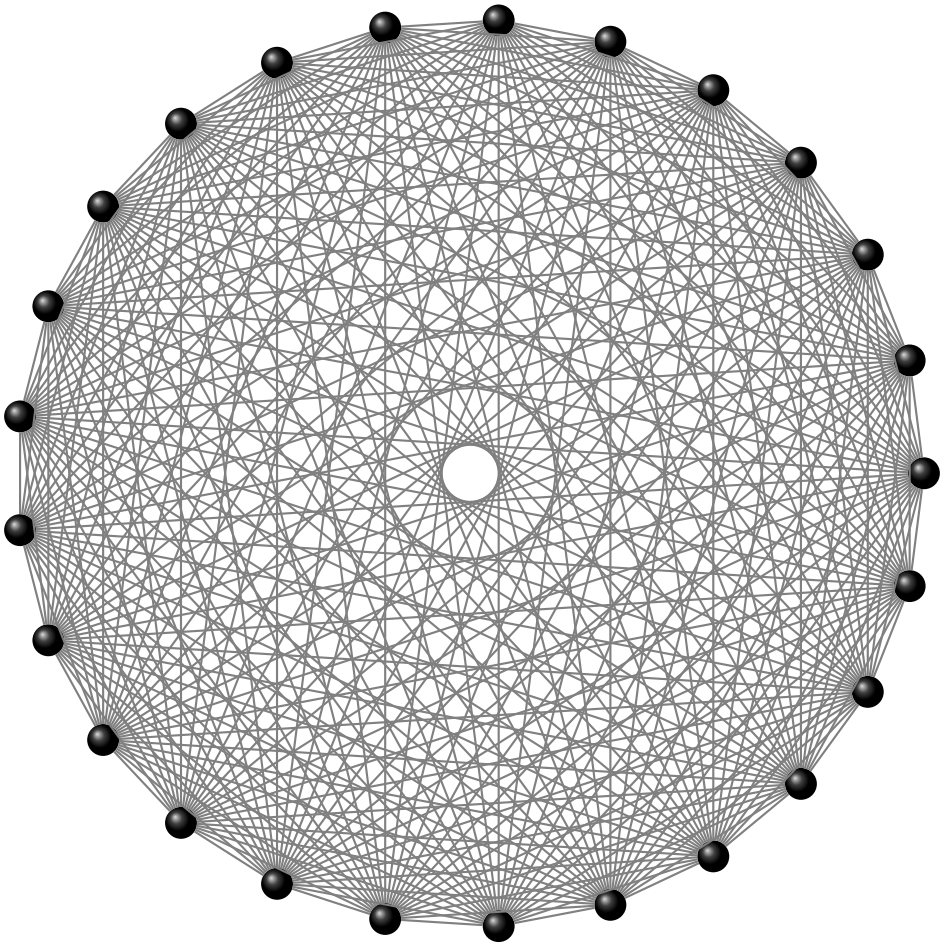
Els grafs: xarxes, camins i connexions

De la matemàtica discreta a la realitat

Aniol Garcia i Serrano
Tutor: Xavier Aguilera Colmenero
2n Batxillerat

Curs 2016-2017
21/11/2016



Qualssevol que siguin els girs i les voltes dels fils en l'espai, un sempre pot obtenir una expressió per al càlcul de les seves dimensions [dels fils], tot i que tal expressió serà d'escassa utilitat en la pràctica. Els artesans que construeixen una xarxa, una trena o alguns nusos estaran més preocupats no per assumptes de mesura, sinó de posició: el que li importarà serà la manera en què els fils s'entrellacen.

A. Vandermonde

Aquest treball vol ser un agraïment a totes aquelles persones que en un moment o un altre m'he anat trobant pel camí de les matemàtiques. Potser no ens hem aturat a parlar de la teoria de grafs, o sí; però, en qualsevol cas, totes les aportacions han estat un pilar indispensable. Cadascú, des del seu punt de vista, des de la seva experiència, des del seu camp de treball, ha anat enriquint el meu bagatge i aportant les dosis d'entusiasme necessàries per ajudar-me a avançar.

Gràcies doncs ...

a en Jordi Zanca, que em va començar a parlar de matrius;

a en Jordi Vergés, amb qui puc comptar quan tinc dubtes i entrebancs;

als professors i als companys del programa de Bojos per les matemàtiques, per les vivències, per les hores compartides... per l'experiència;

als companys del Math Summer Camp, en especial a en Marc Felipe, amb qui vam gaudir de llargues hores de converses, d'enigmes i de problemes; i a en Pere Pascual, per formar part de l'equip que ho va fer possible;

a l'equip de l'IRI (Institut de Robòtica i Informàtica Industrial), en especial a en Sergi Foix, en Gerard Canal i en Guillem Alenyà, per permetre'm de passar uns dies amb ells, encoratjar-me i donar-me savis consells. Ha estat un plaer.

M'agradaria agrair-ho també...

a la Iolanda Guevara, coordinadora del programa de Bojos per les matemàtiques, per fer-me d'enllaç i donar-me suport;

a l'Antoni Benseny i en Robert Salla, que m'han atès molt amablement i m'han estat acompanyant en tot moment;

a l'Anton Aubanell, per obrir-me la porta de casa seva, per escoltar-me, per interrogar-me, per rectificar-me, per explicar-me, per la seva paciència... i sempre amb un somriure i les mans esteses;

al tutor del treball, Xavier Aguilera, professor de l'escola Vedruna de Malgrat de Mar, pel seu acompanyament i per confiar en mi;

a la meva família, per la seva infinita paciència i la seva dedicació;

i a tots els que s'han interessat pel meu treball.

Moltes gràcies a tots!

Índex

Capítol 0

Presentació

El tema d'aquest treball, tal com diu el títol, tracta sobre els grafs: xarxes, camins i connexions. Pretén fer el pas de la matemàtica discreta a la realitat. La matemàtica discreta s'escarrega de l'estudi de conjunts, estructures i processos formats per elements que es poden comptar un a un i de manera separada. La teoria de grafs n'és una branca. Els grafs sovint formen part de l'entramat de mecanismes que fan funcionar moltes de les coses que ens envolten, que ens permeten relacionar-nos, o bé que ens faciliten el dia a dia, per exemple. Però, malgrat tot, són uns grans desconeguts. Al llarg del treball intento crear lligams entre la part més teòrica i abstracta i algunes de les aplicacions que se'n deriven. Intento endinsar-me en el coneixement per comprendre aquests mecanismes. Intento crear els mecanismes per que el coneixement esdevingui una eina pràctica i funcional.

Justificació

Des de feia un cert temps, ja tenia clar que el meu treball de recerca havia de ser d'àmbit científic o tècnic. Havia pensat en temes de robòtica i informàtica, camps en els quals ja havia treballat anteriorment, però els treballs que se m'acudien en aquestes disciplines eren massa concrets i es basaven en un sol producte final, cosa que no m'acabava de convèncer. D'altra banda, també hi havia els temes purament matemàtics. Aquesta opció suposava tractar unes matemàtiques a les quals no estava del tot avesat. Finalment, doncs, vaig decidir fer un treball que combinés la teoria i rigorositat de les matemàtiques amb una part més aplicada de programació. Aquesta unió en la vaig trobar en la teoria de grafs.

El meu primer contacte amb els grafs va ser el problema 9 de la Marató de Proble-

mes 2015 [1]. Sense saber-ho, vaig calcular els meus primers arbres d'Steiner i vaig utilitzar manualment algorismes de grafs com el de Kruskal. El procés de resolució d'aquest problema va ser curiós, diferent. El cert és que vaig percebre una porta oberta amb un llarg camí per endavant, un camí per descobrir, molt engrescador. I segurament aquest és el motiu per el qual vaig recuperar el tema i vaig decidir endisar-m'hi en el meu treball de recerca.

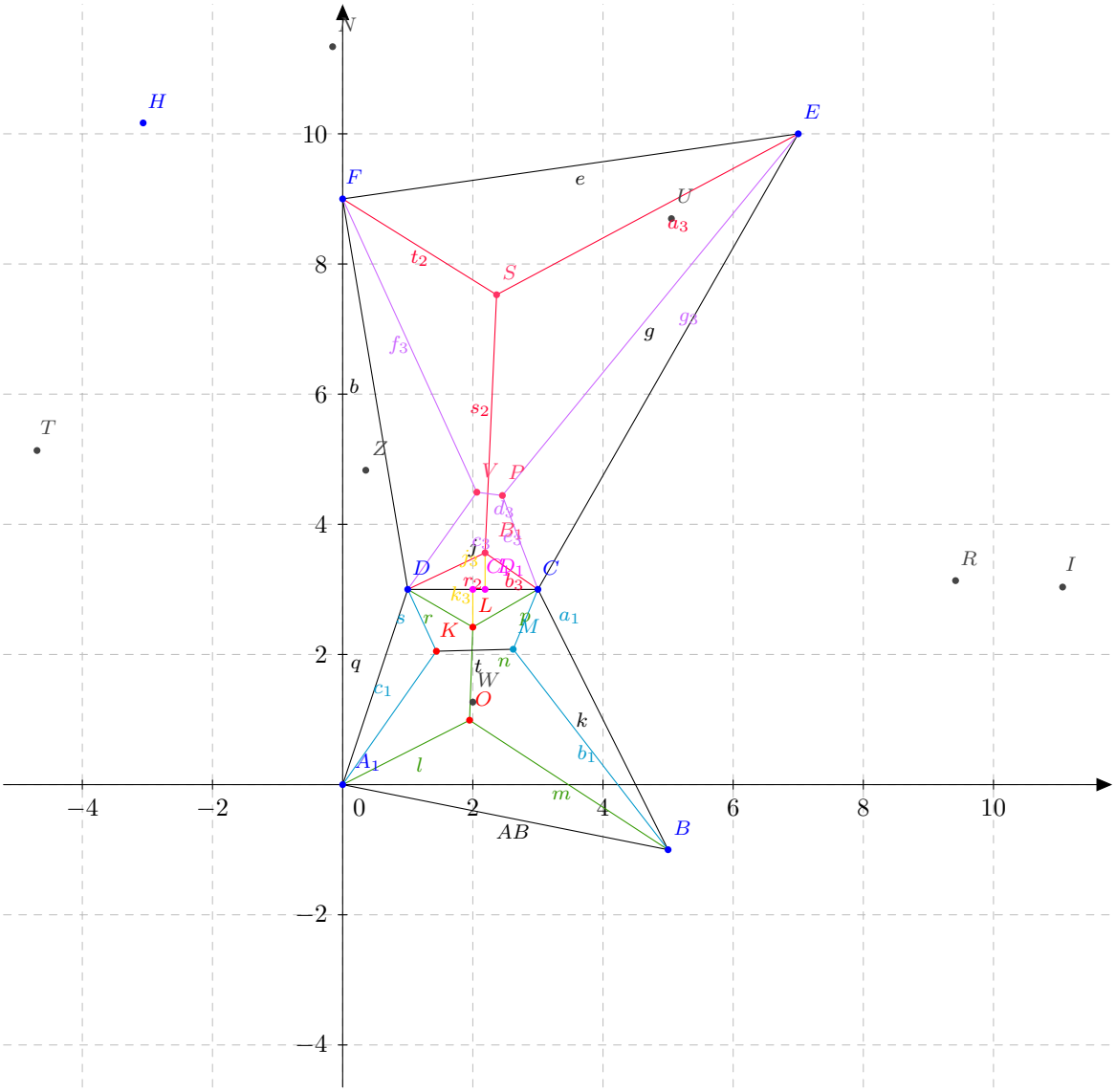


Figura 1: El meu primer graf

[1] Veure www.cangur.org/marato/2015/

Objectius

A partir d'aquest primer graf em vaig plantejar una sèrie de qüestions amb la intenció de trobar les respostes adients:

- Què són els grafs?
- Quins són els procediments, mecanismes i/o eines per aconseguir anar des del concepte abstracte de grafs fins a les dades concretes i reals que ofereixen una resposta a un planteig inicial?
- Quines són les seves aplicacions?

Els objectius del treball són, doncs:

- Conèixer la teoria de grafs.
- Estudiar i implementar algorismes que permetin resoldre problemes de grafs.
- Mostrar que la teoria de grafs té aplicacions en diferents àmbits de la realitat que ens envolta.

Formulació de la hipòtesi de partida

La teoria de grafs, com a branca de la matemàtica discreta, ens proporciona eines per modelitzar estructures i processos i ens permet crear aplicacions pràctiques mitjançant procediments algorísmics.

En aquest treball he intentat validar aquesta hipòtesi.

Estructura del treball i fonts d'informació

El cos del treball s'estructura en quatre capítols que permeten endinsar-se, de manera seqüenciada, en el coneixement del grafs i les seves aplicacions.

En el primer capítol es fa una introducció a la teoria de grafs. Es comença fent un recorregut per la història dels grafs, des dels seus orígens, amb el planteig dels primers problemes, fins la teoria de grafs moderna. Tot seguit s'expliquen els conceptes bàsics: és la part més teòrica en la qual es defineix el graf i es fa referència a la seva estructura i els seus components. S'explica també què són els isomorfismes i, finalment, es fa una descripció dels diferents tipus de grafs. Cal remarcar que, en aquest apartat, s'han exposat les propietats fonamentals i s'han inclòs demostracions sempre que ha estat possible. Majoritàriament, aquestes demostracions no

han estat fruit d'una recerca a la xarxa sinó d'una recerca personal. Les persones i entitats a les qui adreço part dels meus agraïments a l'inici d'aquest treball hi han tingut molt a veure.

En el segon capítol es fa un pas més enllà: es descriu què són els grafs ponderats i els dirigits, els tipus de camins i les maneres diferents de gestionar tota aquesta informació afegida. Finalment s'estudien diferents algorismes segons la seva finalitat. S'analitza el funcionament, les propietats fonamentals i les aplicacions més comuns de cadascun d'aquests procediments i s'especifica el pseudocodi. En l'Annex B adjunto els programes, amb llenguatge Python, que permeten resoldre els algorismes, juntament amb exemples d'entrada i de sortida. La implementació d'aquests algorismes és, també, una aportació personal.

El tercer capítol és un incís al disseny de grafs. Es fa referència al punt de Fermat i a l'arbre de Steiner per la seva rellevància. Se'n mostra un bonic exemple mitjançant les bombolles de sabó.

Finalment, en el darrer capítol, s'exposen algunes aplicacions pràctiques. Una d'elles és el recorregut per la xarxa de metro de Barcelona. Aporto al treball un algorisme que, a partir d'una estació de sortida i una altra d'arribada, ofereix, com a resposta, el trajecte que permet fer el recorregut amb el mínim temps possible. Aquesta resposta inclou les estacions per on es passarà, els transbordaments que caldrà fer i el temps total que es trigarà per anar d'un punt a l'altre. En l'Annex B adjunto la codificació d'aquest algorisme així com exemples d'entrades i sortides.

Les meves conclusions i valoracions esdevenen el punt i final d'aquest treball.

Les fonts d'informació utilitzades han estat molt diverses. En la bibliografia he detallat els documents i les pagines Webs consultades. Vull remarcar, però, que el gruix més important de la informació prové d'altres fonts que per a mi han estat més significatives pel fet que han estat més properes i, sobretot, vivencials: el guiatge de la UB per part de l'Antoni Benseny i, sobretot, el seguiment i l'acompanyament d'en Robert Salla han estat claus; la participació en el Math Summer Camp[2] i en el programa Bojos per les matemàtiques[3] també em va aportar moltíssima informació; les xerrades i consells per part dels investigadors de l'IRI m'han obert moltes portes; les converses amb l'Anton Aubanell, imprescindibles.

Per a l'elaboració d'aquest treball s'han utilitzat eines informàtiques provinents, en tots els casos, de programari lliure. Aquest programari m'ha permès escriure les fórmules i confegir els esquemes dels grafs que apareixen en el treball, els quals són tots d'elaboració pròpia. Esmentar l'ús del sistema operatiu GNU/Linux, el

[2]Activitat organitzada per Fundació Privada Cellex, UPC-FME i CFIS

[3]Programa organitzat per FEEMCAT, SCM i Fundació Catalunya La Pedrera

processador de text LaTeX, el programa GeoGebra, el llenguatge Python i el sistema de control de versions Git. L'ús d'aquestes eines ha estat una part important del meu aprenentatge i és per aquest motiu que les he detallades en l'Annex A.

La presentació d'aquest treball (el tipus de lletra, l'espaiat, les mides, el format...) ha quedat determinada pels valors propis del processador de text LaTeX.

Capítol 1

Introducció a la teoria de grafs

Aquest primer apartat consisteix en la part més teòrica del treball. Explicaré breument la història d'aquesta branca de la matemàtica i, posteriorment, ens endinsarem en la teoria de grafs com a tal. La teoria de grafs pot semblar bastant abstracta i, a vegades, complicada d'entendre, però estarà acompanyada de demostracions i exemples propis que pretenen facilitar el seguiment del treball.

1.1 Història del grafs

1.1.1 Els primers passos

Tot sovint, les noves branques de la matemàtica sorgeixen de cercar solucions a problemes. Problemes que no poden ser resolts ni demostrats amb el que coneixem i que forcen a desenvolupar nous mètodes i teories. La teoria de grafs no n'és una excepció i, tot seguit, presentaré els problemes que van forjar la creació d'aquesta branca.

Euler i els ponts de Königsberg

La teoria de grafs neix a partir de la solució d'un problema curiós: el problema dels ponts de Königberg (l'actual Kaliningrad, Rússia). El planteig del problema és el següent:

> *"El riu Pregel divideix Königsberg en quatre parts separades, i connectades per set ponts. És possible caminar per la ciutat passant per tots*

els ponts tan sols una vegada?"

Els ciutadans de Königsberg sabien que no era possible, però mai ningú ho havia demostrat fins que Leonhard Euler ho va fer. Aquesta demostració queda recollida en *"Solutio problematis ad geometriam situs pertinentis"* publicat el 1736, i l'article també va ser inclòs en el volum 8 de *"Commentarii Academiae Scientiarum Imperialis Petropolitanae"* publicat el 1741 (veure ref. [6]). En aquest text, Euler diu el següent:

> *"A més d'aquella part de la geometria que s'ocupa de les quantitats, la qual sempre ha generat un interés preferent, hi ha una altra part -encara pràcticament desconeguda- que ja fou esmentada per Leibniz amb el nom de geometria de posició. Aquesta part de la geometria estudia tot allò que pot ésser determinat únicament per la posició, així com les propietats de la posició; en aquest camp hom no hauria de preocupar-se de quantitats ni de com obtenir-les. No obstant això, els tipus de problemes que pertanyen a aquesta geometria de posició i els mètodes usats per resoldre'ls encara no són suficientment definits. Així doncs, quan darrerament se'm va plantejar un problema que semblava bàsicament geomètric, però que no demanava l'obtenció de quantitats ni admetia una sol·lució basada en el càlcul de quantitats, em vaig adonar que pertanyia a la geometria de posició, sobretot perquè només es podia usar la posició per resolde'l, mentre que el càlcul es mostrava inútil. Així, he decidit exposar ací, com a mostra de la geometria de posició, el mètode que he deduït per a resoldre problemes d'aquesta mena."*

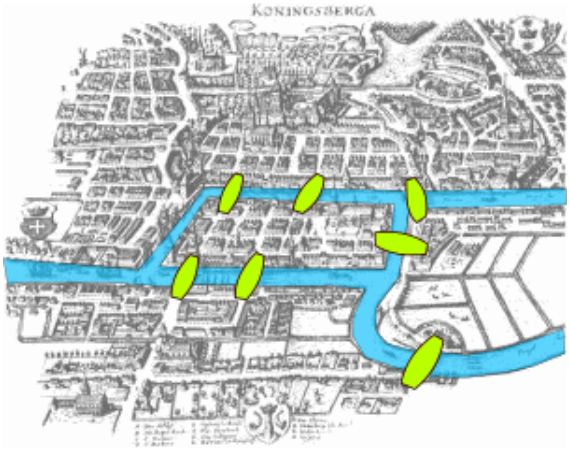


Figura 1.1: Representació dels ponts de Köingsberg [4]

[4] Autor: Bogdan Giuşcă. Imatge de domini públic sota la llicència CC BY-SA 3.0 i GNU Free Documentation License (versió 1.2)

Per aconseguir demostrar que el problema no tenia sol·lució, Euler va haver de representar el problema mitjançant un mapa topològic, posant les masses de terra com a punts i els ponts com a segments que unien aquests punts. Va crear d'aquesta manera el primer graf de la història. Euler va observar que, exceptuant el punt inicial i final del trajecte, quan s'arriba a un punt per un pont, s'ha de sortir per un altre. Això significa que el nombre de vegades que s'entra a un punt equival al nombre de vegades que cal sortir-ne (exceptuant, tal com s'ha dit abans, el punt inicial i final), cosa que no és possible si no hi ha un nombre parell de ponts que vagin a un punt. Cal observar que en el problema original hi ha 4 masses de terra, i totes elles tenen un nombre imparell de ponts o arestes, i per tant no és possible passar per tots els ponts tan sols una vegada. Euler va arribar a la conclusió de que es pot recòrrer un graf passant només una vegada per les seves arestes només si aquest tenia 0 o 2 nodes de grau senar, que serien els punts d'inici i final del recorregut. Aquest resultat es considera el primer en teoria de grafs ja que conté un important teorema d'aquesta branca. A més d'iniciar la teoria de grafs, amb aquest resultat també comença l'estudi dels grafs planars, introdueix el concepte de característica d'Euler de l'espai i el seu teorema de poliedres (teorema que després va utilitzar per demostrar que els sòlids platònics eren els únics sòlids regulars). Amb tot això, Euler posa les bases, no tan sols de l'estudi dels grafs, sinó també de la topologia.

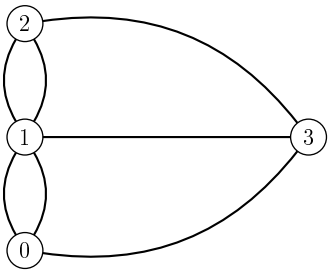


Figura 1.2: Representació topològica dels ponts de Königsberg

Vandermonde i el tour del cavall d'escacs

A partir de l'article d'Euler, diversos matemàtics van començar a interessar-se pel camp de la topologia (o geometria de la posició, com li deien en aquell moment). Un d'ells fou Alexandre-Théophile Vandermonde. Vandermonde va treballar i estudiar el problema dels cavalls, que planteja quins moviments cal fer per tal que un cavall passi per totes les caselles del tauler d'escacs, problema que també va tractar Euler. Els estudis que Vandermonde va fer sobre aquest problema van ser publicats el 1771 en el "*Remarques sur des problèmes de situation*", (veure ref.

[12]), on encara no es feia referència al concepte de graf, tot i que el problema es resol mitjançant aquests. Aquest treball inicia l'estudi de la teoria de nusos, una altra branca de la topologia.

1.1.2 Les primeres descobertes i aplicacions

Un cop establert l'inici, és difícil veure com va continuar desesenvolupant-se la teoria, ja que es van anar resolent problemes de manera aïllada, sense establir relacions entre ells. Tot i així, durant el segle XIX es van plantejar una gran quantitat de problemes i es van desenvolupar molts teoremes referents als grafs.

Francis Guthrie

El 1852, Francis Guthrie, matemàtic britànic, es planteja el següent problema mentre intenta pintar un mapa del Regne Unit:

> *"És possible pintar qualsevol mapa de països de tal manera que un país tingui un color diferent al de tots els seus veïns, utilitzant tan sols quatre colors?"*

D'aquest problema en surt el teorema a partir del qual s'estableix que qualsevol mapa pot ser pintat únicament amb quatre colors diferents, de tal manera que dues regions adjacents (entenem com a adjacents dues regions que comparteixiin frontera, no tan sols un punt) no tinguin colors iguals. Aquest problema, que pot semblar tan trivial, no va ser demostrat fins l'any 1976. Va passar per mans de pioners com De Morgan, Hamilton, Cayley, Kempe (que va fer una demostració publicada el 1879, veure ref. [8]), Heawood (que va concloure que la demostració de Kempe no era correcta)... Finalment el 1976 Appel i Hanken van demostrar, mitjançant un programa d'ordinador, que tot mapa es podia pintar només amb quatre colors. Aquesta demostració queda recollida a ref. [1, 2, 3]. Pel fet de basar-se en un programa d'ordinador, la demostració no va acabar de convèncer. Així doncs, aquest problema no va ser solucionat de manera formal fins l'any 1996 quan, recorrent a la teoria de grafs ja desenvolupada, Neil Robertson, Daniel P. Anders, Paul Seymour i Robin Thomas van publicar-ne una demostració. En els treballs d'Appel i Hanken es van definir alguns dels conceptes i fonaments de l'actual teoria de grafs.

Arthur Cayley

Arthur Cayley, matemàtic que treballava en la teoria de grups, topologia i combintòria, també va aportar una gran quantitat de coneixement a la teoria de grafs. Va treballar amb grafs de tipus arbre i va desenvolupar la fòrmula n^{n-2}, que determina el nombre d'arbres expansius que té un graf complet de n vèrtex (veure apartat 1.4.3). Una fórmula semblant apareixia en treballs de Carl Wilhelm Borchardt, en els quals Cayley es va basar i va extendre, tot i que, actualment, qui dóna nom a la fórmula és el mateix Cayley.

També va treballar en el desenvolupament d'una representació de l'estructura abstracta d'un grup, creant els grafs de Cayley i el teorema de Cayley. Finalment va contribuir també, l'any 1857, en la representació i ennumeració dels isòmers alcans (composts químics que comparteixen fórmula o composició però tenen diferent estructura molecular), representant cada compost mitjançant un graf de tipus arbre. Tot i això, Cayley no només va ser actiu en teoria de grafs, sinó que també va desenvolupar teoremes en àlgebra lineal, topologia i geometria.

William Hamilton i Thomas Kirkman

William Rowan Hamilton va plantejar, l'any 1859, un problema que consistia a trobar un camí que passés una sola vegada per cadascun dels 20 vèrtexs d'un dodecàedre tot recorrent les seves arestes. Hamilton el va comercialitzar com a joc sota el nom de "The Icosian game" (és important dir que el nom de icosian no feia referència a la utilització d'un icosaedre, sinó als 20 vèrtexs del dodecaedre per on s'havia de passar). Entorn aquest joc existeix un gran controvèrsia, ja que Euler anteriorment havia plantejat un problema semblant mentre estudiava el problema dels cavalls d'escacs, i Kirkman va plantejar, a la Royal Society, exactament el mateix problema que Hamilton un temps abans.

Gustav Kirchhoff

Gustav Kirchhoff, conegut majoritàriament en el camp de l'electrotècnia per les lleis de Kirchhoff, també va fer aportacions importants. Les seves lleis, publicades el 1874, es basen en la teoria de grafs, però a més, va ser el primer en utilitzar els grafs en aplicacions industrials. Va estudiar sobretot els de tipus arbre i, amb la investigació que va dur a terme, va formular el teorema de Kirchhoff, referent al nombre d'arbres d'expansió que es poden trobar en un graf. Aquest teorema es considera una generalització de la fórmula de Cayley.

1.1.3 Teoria de grafs moderna

Durant el segle XX, la teoria de grafs es va continuar desenvolupant. Amb les bases ja establertes durant el segle XIX, els matemàtics hi van començar a treballar i, el 1936, Dénes König va escriure el primer llibre de teoria de grafs: *"Theorie der endlichen und unendlichen Graphen"* (veure ref. [9]). Frank Harary va escriure el llibre *"Graph Theory"* l'any 1969 (veure ref. [7]), fent més accessible la teoria de grafs. El desenvolupament de la informàtica i les noves tècniques de computació van permetre treballar amb grafs a molt més gran escala, fent possible, per exemple, la ja citada primera demostració del teorema dels quatre colors per Appel i Hanken.

Actualment la teoria de grafs és una part molt important de la matemàtica discreta i està vinculada amb moltes branques diferents, com per exemple la topologia, la combinatòria, la teoria de grups, la geometria algebraica... Des del seu desenvolupament s'han utilitzat els grafs per resoldre problemes referents a aquests camps i representar-los visualment. Té aplicacions en molts altres àmbits com per exemple la computació, la informàtica, la física, la química, l'electrònica, les telecomunicacions, la biologia, la logística i fins i tot en l'àmbit econòmic.

1.2 Principis bàsics

Un *graf* $G = (V, E)$ es defineix com un conjunt de *vèrtexs* (o nodes) $V = \{v_1, v_2, ..., v_n\}$ i un conjunt d'*arestes* $E = \{e_1, e_2, ..., e_m\}$, cadascuna de les quals uneix dos vèrtex de V. Si v_i i v_j, amb $v_i, v_j \in V$, estan units per l'aresta e_k escriurem $e_k = \{v_i, v_j\}$. Quan volguem especificar el graf concret emprarem les notacions $V(G)$ i $E(G)$. Així doncs, un graf està format per un conjunt de punts i un conjunt d'arestes que uneixen alguns d'aquests punts. El nombre de vèrtexs d'un graf queda determinat pel nombre d'elements que hi ha en el conjunt V, per tant, ens referirem a ell com a $|V|$ (cardinal de V). Amb les arestes passa el mateix, i també utilitzarem $|E|$ per indicar el nombre d'arestes d'un graf. Definim també que dos vèrtexs són *adjacents* si estan units per una aresta i, com a conseqüència, són *incidents* a l'aresta.

La idea intuitiva de graf convida a utilitzar representacions gràfiques. Així, en la figura 1.3 es mostra un graf simple G format per:

- $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$
- $E = \{\{v_1, v_2\}, \{v_1, v_4\}, \{v_1, v_5\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_3, v_6\}, \{v_4, v_6\}\}$

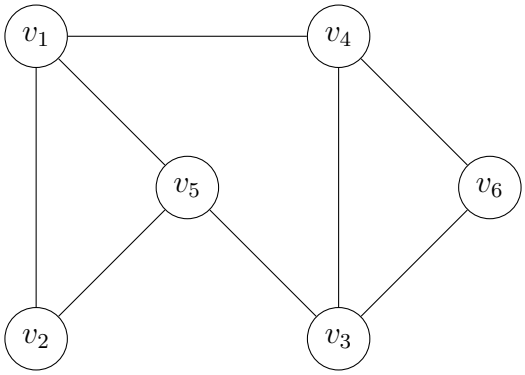


Figura 1.3: Representació gràfica del graf $G = (V, E)$ que s'ha presentat

Si una aresta comença i acaba en el mateix vèrtex (per exemple $e_m = \{v_i, v_i\}$ s'anomena *llaç* (figura 1.4, cas (a)). També pot ser que hi hagi dues arestes idèntiques, és a dir, dues arestes que uneixin v_i i v_j (figura 1.4, cas (b)). En qualsevol d'aquests dos casos anteriors, el graf s'anomena *multigraf* o *pseudograf*. En cas contrari, el graf serà anomenat *simple*. Amb el que hem vist fins ara, podem dir que $e_1 = \{v_1, v_2\}$ és equivalent a $e_2 = \{v_2, v_1\}$, ja que es tracta de parells no ordenats. Tanmateix, existeixen grafs en els quals les arestes han de ser recorregudes en un sentit determinat. En aquest cas, els grafs s'anomenen *dirigits*. Aleshores, si $e_1 = (v_1, v_2)$ i $e_2 = (v_2, v_1)$, $e_1 \neq e_2$, ja que es tracta de parells ordenats (figura 1.4, cas (c)). En la següent imatge es mostren els grafs esmentats anteriorment:

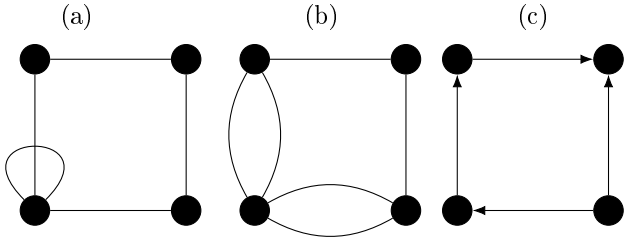


Figura 1.4: Graf amb un llaç (a), graf amb arestes múltiples (b) i graf dirigit (c)

Direm que un graf no dirigit $G = (V, E)$ és *connex* si per a qualsevol $v_i, v_j \in V$ $v_i \neq v_j$ existeix un *camí* (successió d'arestes) que els uneix.

El nombre d'arestes que són incidents a un vèrtex v (comptant els llaços com a dues arestes) determinen el que anomenarem *grau* de v, que es representa amb $g(v)$. La successió de graus d'un graf serà la successió que s'obté a l'ordenar els

graus dels seus vèrtex de manera creixent. Cal dir que no és un problema gens trivial saber si una successió de nombres naturals donada pot ser una successió de graus d'un graf. El grau mínim d'un graf $G = (V, E)$ queda determinat de la següent manera: $\delta(G) = min\{g(v) : v \in V(G)\}$. De manera similar, el grau màxim de G, $\Delta(G) = max\{g(v) : v \in V(G)\}$.

Si d'un graf connex en treiem una aresta o un node, en resulta un altre (o més d'un) graf connex. Si $v \in V(G)$, indicarem per $G - v$ el graf que s'obté al suprimir el vèrtex v i totes les seves arestes incidents. De la mateixa manera, si $e \in E(G)$, $G - e$ indicarà el graf que s'obté a l'eliminiar la aresta e.

Amb tots aquests conceptes ja podem veure el teorema d'Euler, un dels primers teoremes en teoria de grafs i un dels més importants.

Teorema 1 (Euler)

En tot graf $G = (V, E)$, la suma dels graus dels vèrtex és igual al doble del nombre d'arestes.

$$\sum_{v \in V(G)} g(v) = 2|E(G)|$$

Demostració: Només cal veure que cada aresta té dos extrems i que suma dos en el total dels graus. La demostració formal és més complicada, i comporta desenvolupar una inducció respecte del nombre d'arestes:

- Si $|E| = 0$, no cal considerar el cas. Si $|E| = 1$, o bé $|V| = 2$ i cada vèrtex té grau 1 o bé la aresta és un llaç i hi ha un sol vèrtex de grau 2. En qualsevol d'aquests dos casos, el teorema es verifica.
- Ara suposem que el teorema està demostrat per a $|E| \leq k$ i que G és un graf amb $|E| = k + 1$. Si e és una aresta de G prenem $H = G - e$.
- Llavors tots els vèrtexs de H tenen el mateix grau a H que a G excepte 2 que tenen un grau menys o un que té dos graus menys (només en el cas que e sigués un llaç). En tots dos casos obtenim que:

$$\sum_{v \in V(G)} g(v) = \sum_{v \in V(H)} g(v) + 2 = 2(|E| - 1) + 2 = 2|E|$$

D'aquesta demostració en treiem una altra afirmació:

En tot graf, el nombre de vèrtex amb grau imparell, és parell.

1.3 Isomorfismes

Es diu que dos grafs són isomorfs si existeix una funció bijectiva φ entre els seus vèrtexs que conservi les arestes. Formalment dos grafs G i H són isomorfs si

$\exists \varphi : V(G) \to V(H)$ tal que $\forall u, v \in V(G)$, $\overline{\varphi(v)\varphi(u)} \in E(H)$ si i només si $\overline{vu} \in E(G)$

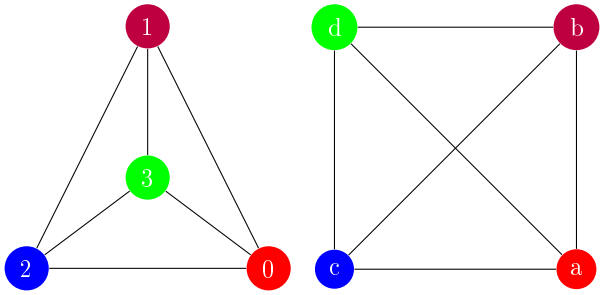


Figura 1.5: Grafs isomorfs

1.4 Tipus de grafs

Fins ara hem vist els conceptes bàsics en teoria de grafs i algunes de les propietats que compleixen tots els grafs. No obstant, existeixen diversos tipus de grafs que tenen propietats especials. A continuació se'n presenten alguns.

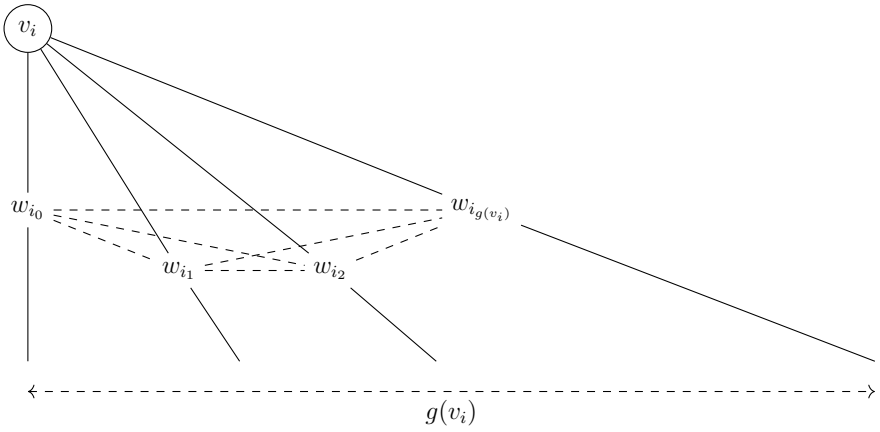
1.4.1 Graf lineal

Un graf lineal $L(G)$ d'un graf G és un graf que representa les adjacències entre les arestes de G. Formalment, donat un graf G, el graf lineal $L(G)$ és aquell en el qual cada vèrtex correspon a una aresta de G i dos vèrtexs són adjacents només si les arestes corresponents a G comparteixen un vèrtex.

Propietat 1 El graf lineal d'un graf amb n nodes, e arestes i amb vèrtexs de graus $g(v_i)$ té $n' = e$ nodes i e' arestes, on

$$e' = \frac{1}{2} \sum_{i=1}^{n} g(v_i)^2 - e$$

Demostració: Cada node v_i amb grau $g(v_i)$ del graf original generarà un graf complet de $g(v_i)$ nodes ($K_{g(v_i)}$) [5].



Un graf complet té $\binom{n}{k} = \frac{n(n-1)}{2}$ arestes, per tant, en aquest cas se'n generen $\frac{g(v_i)(g(v_i)-1)}{2}$. Però això es compleix per a cada vèrtex, i llavors podem escriure

$$\sum_{i=1}^{n} \frac{1}{2} g(v_i)(g(v_i)-1) = \frac{1}{2}\sum_{i=1}^{n}(g(v_i)^2 - g(v_i)) = \frac{1}{2}\sum_{i=1}^{n} g(v_i)^2 - \underbrace{\frac{1}{2}\underbrace{\sum_{i=0}^{n} g(v_i)}_{2|E|}}_{|E|} = \frac{1}{2}\sum_{i=1}^{n} g(v_i)^2 - e$$

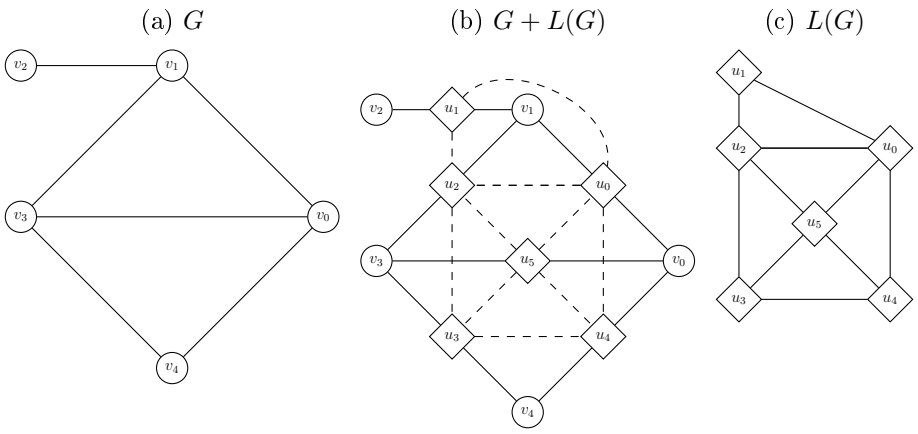


Figura 1.6: Procés de construcció d'un graf lineal

[5] Veure apartat 1.4.3

1.4.2 Cicles

Els *cicles* són grafs 2-regulars[6] amb n vèrtexs i n arestes, i s'anomenen C_n.

Propietat 1 El graf lineal d'un cicle és ell mateix.

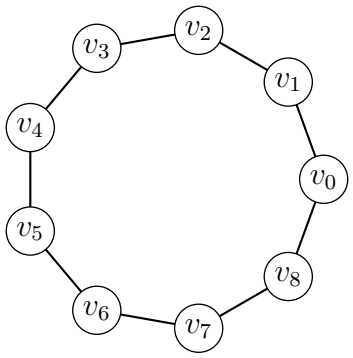


Figura 1.7: Cicle C_9

1.4.3 Grafs complets

Un *graf complet* és un graf on cada vèrtex està unit a tots els altres una sola vegada. Un graf complet amb n nodes és un graf simple, $(n-1)$-regular i la seva nomenclatura és K_n.

Propietat 1 Els grafs complets tenen $\binom{n}{2} = \frac{n(n-1)}{2}$ arestes.

Propietat 2 El nombre de cicles que conté un graf complet queda determinat per la següent igualtat:

$$C_n = \sum_{K=3}^{n} \frac{1}{2} \binom{n}{k} (k-1)!$$

On:

$\frac{1}{2}$ es multiplica perquè es conten dues vegades els cicles, ja que no són dirigits.

$\binom{n}{k}$ és el nombre de grups de k elements que es poden agafar.

$(k-1)!$ és el nombre de permutacions circulars de k elements.

[6] Veure apartat 1.4.12

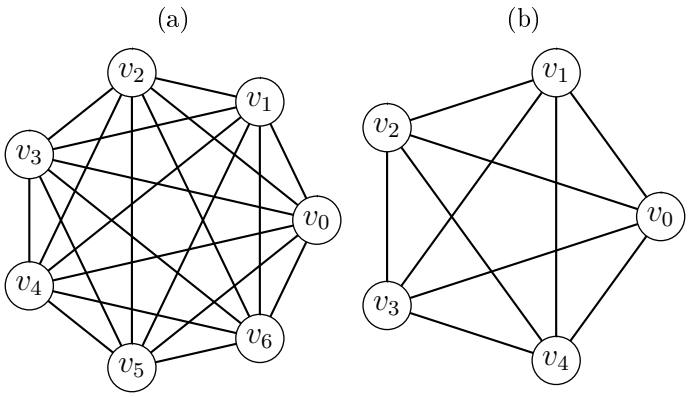


Figura 1.8: Grafs complets K_7(a) i K_5(b)

1.4.4 Grafs bipartits

Els *grafs bipartits* són aquells en els quals els vèrtexs es poden separar en dos conjunts disjunts U i W (és a dir, que els dos conjunts no tinguin elements comuns) de tal manera que un vèrtex d'un conjunt no sigui mai adjacent a un altre vèrtex del mateix conjunt. Es pot dir també que un graf és bipartit si i sols si no conté cicles de longitud senar.

Propietat 1 Tots els grafs C_n amb n parell, són també grafs bipartits.

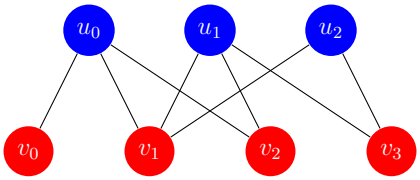


Figura 1.9: Graf bipartit

1.4.5 Grafs bipartits complets

Els *grafs bipartits complets* són grafs bipartits en els quals cada element del conjunt U està unit a tots els elements dels del conjunt W. S'anomenen $K_{m,n}$, on $m = |U|$ i $n = |W|$.

Propietat 1 En els grafs bipartits $K_{m,n} = K_{n,m}$, $|V| = n + m$ i $|E| = mn$.

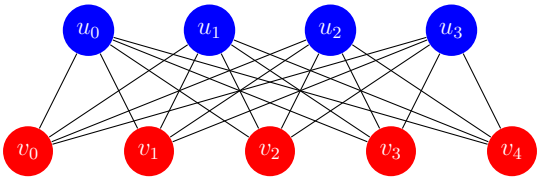


Figura 1.10: Graf bipartit complet $K_{4,5}$

1.4.6 Grafs xarxes

Els *grafs xarxes* bidimensionals $G_{m,n}$ són grafs bipartits que formen un entramat en forma de cuadrícula de $m \times n$ vèrtexs.

Propietat 1 Es pot greneralitzar per a xarxes de més dimensions com a $G_{m,n,o,\dots}$.

Propietat 2 Un graf xarxa té mn vèrtexs i $(m-1)n + (n-1)m$ arestes.

Demostració: Tal com es pot veure a la figura 1.11, un graf xarxa té

$$(n-2)(m-2)+2(m-2)+2(n-2)+4 = nm-\cancel{2m}-\cancel{2n}+\cancel{4}+\cancel{2m}-\cancel{4}+\cancel{2n}-\cancel{4}+\cancel{4} = mn$$

vèrtexs. D'altra banda, la suma dels graus de tots els vèrtexs és

$$\frac{4(m-2)(n-2) + 3 \times 2(m-2+n-2) + 4 \times 2}{2} = 2(m-2)(n-2)+3(m-2)+3(n-2)+4$$

Si es desenvolpua, s'obté

$$(n-2)(m-2) + \boxed{(n-2)(m-2) + 2(m-2)} + (m-2) + \boxed{2(n-2)} + (n-2) + \boxed{4}$$

on la suma dels elements marcats equival a mn. Si es continua desenvolupant tenint en compte aquest detall, s'obté

$$mn - 2m - 2n + 4 + m - 2 + n - 2 + mn = 2mn - m - n = \boxed{m(n-1) + n(m-1)}$$

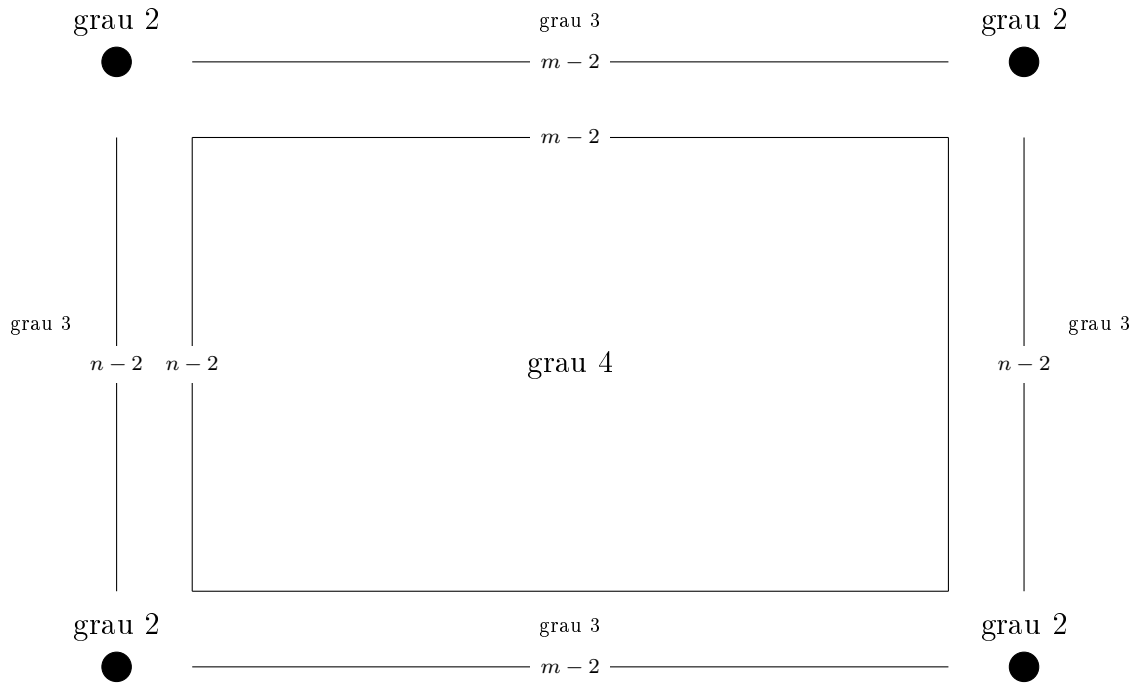


Figura 1.11: Representació dels graus dels diferents nodes d'un graf xarxa

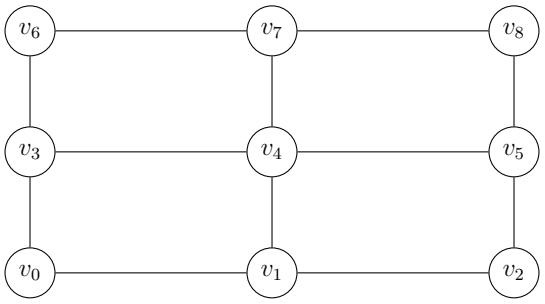


Figura 1.12: Graf xarxa $G_{3,3}$

1.4.7 Arbres

Els *arbres* són un tipus molt important de grafs: són grafs connexos sense cicles, de manera que existeix un únic camí entre dos vèrtexs.

Propietat 1 Si a un arbre se li afegeix una aresta, es genera un cicle, i si se'n treu una, el graf deixa de ser connex.

Propietat 2 Hi ha un tipus especial d'arbres anomenats *elementals o camins*, que són els arbres amb $|V| = 1$, $|V| = 2$ i en general tots aquells on $\delta = 1$ i $\Delta = 2$. S'anomenen P_n, on $n = |V|$. També es pot pensar en grafs elementals com a $G_{n,1}$.

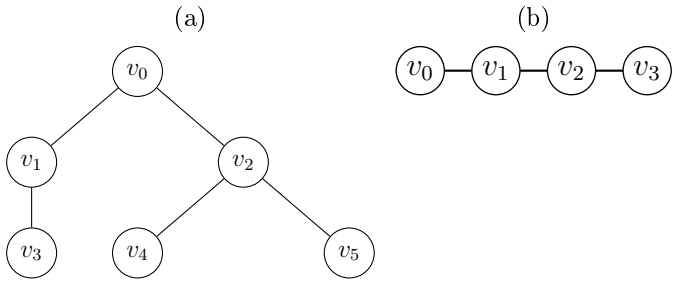


Figura 1.13: Arbre (a) i graf elemental P_4 (b)

Teorema 2

Un arbre amb n nodes té $n-1$ arestes.

Demostració: Si comprovem el cas on un arbre T té $n = 1$ vèrtexs, veiem que no té cap aresta, per tant el teorema es compleix. Si treiem una aresta de l'arbre, sorgiran 2 arbres més T_1 i T_2. Per hipòtesi, T_1 tindrà $v_1 = |V(T_1)|$ vèrtexs i $n_1 - 1 = |E(T_1)|$ arestes, i T_2 tindrà $n_2 = |V(T_2)|$ vèrtexs i $n_2 - 1 = |E(T_2)|$ arestes. El nombre d'arestes de T és $(n_1 - 1) + (n_2 - 1) + 1 = (n_1 + n_2) - 1$. Deduïm, llavors, que el nombre de nodes de T és $n_1 + n_2$. Dit d'una altra manera, $|E(T)| = |V(T)| - 1$.

Com a conseqüència d'aquest teorema, se'n pot deduir el següent:

Teorema 3

En un arbre T amb $|V| \geq 2$, hi ha com a mínim dos vèrtexs de grau 1 (anomenats fulles).

Demostració: Fem un raonament per reducció a l'absurd. Suposem que no es compleix el teorema que volem demostrar, és a dir, que $g(v_i) > 1$ per a tots els vèrtexs. Llavors es compleix que

$$\sum_{v \in V} g(v) > \sum_{v \in V} 2 = 2|V|$$

Però això no és correcte, ja que, com a conseqüència del Teorema 1 i el Teorema 2 podem dir que en un arbre

$$\sum_{v \in V} g(v) = 2|E| = 2|V| - 2$$

i estem dient que $2|V| - 2 \geq 2|V|$. Amb això ja podem veure que necessitem treure com a mínim dos graus, però demostrem també que amb un sol node de grau 1 tampoc és suficient. Suposem ara que l'arbre té un sol node tal que $g(v) = 1$ i els altres tenen com a mínim grau 2. Llavors

$$2|V| - 2 = \sum_{v \in V} g(v) = 1 + \sum_{\substack{v \in V \\ \text{si } g(v) \neq 1}} g(v) \geq 1 + \sum_{\substack{v \in V \\ \text{si } g(v) \neq 1}} 2 = 1 + 2(|V| - 1) = 2|V| - 1$$

Ara estem dient que $2|V| - 2 \geq 2|V| - 1$, que torna a ser una contradicció. Sabem que si treiem un grau més, el teorema es complirà, i hem demostrat que el mínim nombre de vèrtexs de grau 1 és 2.

Hi ha una altra manera de demostrar-ho, més directa: Suposem que a l'arbre hi ha k vèrtex de grau 1, i $|V| - k$ de grau $g(v_i) \geq 2$. D'aquesta manera

$$2|V| - 2 = \sum_{v \in V} g(v) \geq k + 2(|V| - k) = 2|V| - k$$

Llavors podem dir que $-2 \geq -k$ i per tant $2 \leq k$.

Un altre teorema relaciona el nombre de fulles amb el grau màxim d'un arbre:

Teorema 4

El nombre de fulles d'un arbre T és més gran o igual a $\Delta(T)$.

Demostració: Si eliminem un node de grau Δ de l'arbre, juntament amb totes les seves arestes incidents, obtenim un conjunt de Δ grafs disconnexos. Si alguns d'aquests grafs consisteixen en tan sols un node, vol dir que abans eren adjacents al node que hem eliminat, per tant, només tenien grau 1. Si, pel contrari, formen nous arbres, pel teorema 3 podem dir que hi haurà, com a mínim, dues fulles. Encara que existeix la possibilitat que un dels nodes amb grau 1 sigués l'adjacent al node que hem tret, sempre podem garantir que hi ha, com a mínim, una fulla. Per tant, també podem garantir que que hi haurà, com a mínim, Δ fulles.

1.4.8 Graf roda

Un *graf roda* amb n vèrtex ès un graf que conté un cicle de longitud $n-1$, on tots els vèrtex del cicle estan connectats a un vèrtex fora del cicle, anomenat *node central*. S'escriu W_n i, a vegades, simplement s'estudia com a $C_{n-1} + K_1$.

Propietat 1 El node central té grau $n-1$, i la resta de nodes tenen grau 3.

Propietat 2 El nombre de cicles que conté un graf roda amb n vètexs està determinat per $n^2 - 3n + 3$.
Demostració: El nombre de cicles que conté un graf roda és la suma del nombre de cicles de longitud des de 3 fins a n ($\sum_{i=3}^{n} C_i$). Amb l'exepció de C_{n-1}, el nombre de cicles per a cada i és $n-1$. En el cas de C_{n-1} n'hi ha n de possibles, però ho representem de la manera $n-1+1$, per tal que sigui més còmode operar. Com que els cicles són possibles a partir de longitud 3, en total hi ha $n-2$ longituds possibles. D'aquesta manera podem escriure $(n-2)(n-1)+1 = \boxed{n^2-3n+3}$.

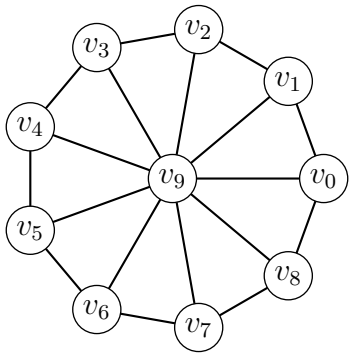


Figura 1.14: Roda W_{10}

1.4.9 Grafs estrella

Un *graf estrella* de grau n és aquell que conté un vèrtex amb grau $n-1$ i els $n-1$ vèrtexs restants de grau 1. La seva nomenclatura és S_n.

Propietat 1 Són estrelles tots els grafs bipartits de la forma $K_{1,n-1}$ o $K_{n-1,1}$.

Propietat 2 El graf lineal de S_n és K_{n-1}.

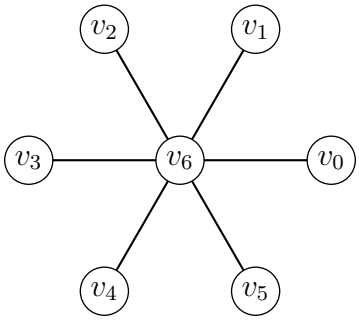


Figura 1.15: Graf estrella S_7

1.4.10 Grafs complementaris

El *graf complementari* del graf G és el graf H amb els mateixos vèrtexs que G, de manera que dos vèrtexs d'H seran adjacents si i només si a G no ho són. Formalment, si $G = (V, E)$ és un graf simple i $K = E(K_n)$ és el conjunt d'arestes derivades de totes les possibles combinacions de dos elements de V, essent $n = |V(G)|$, llavors $H = (V, K \setminus E)$ [7]. Per indicar el complementari de G s'escriu $\bar{G}$ o G'.

Propietat 1 Per obtenir el graf complementari de G tan sols cal afegir les arestes que falten per obtenir un graf complet i treure totes les que hi eren inicialment (ja que $E(G) + E(\bar{G}) = K_{|E(G)|}$). Veure figura 1.16.

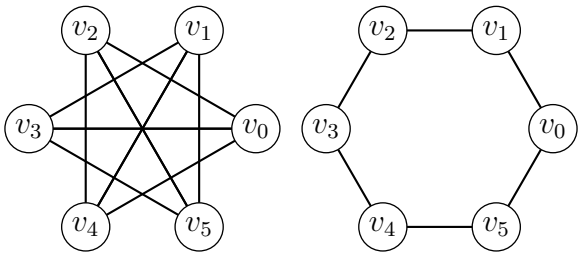


Figura 1.16: Un graf G i el seu complementari $\bar{G}$

1.4.11 Graf nul i grafs buits

Els *grafs buits* són grafs sense arestes, és a dir, conjunts d'n vértexs. Són els complementaris dels grafs complets K_n, i per tant la seva nomenclatura és $\bar{K}_n$, o

[7] La notació $K \setminus E$ indica el conjunt dels elements de K que no són a E

simplement, N_n. Estrictament s'anomena *graf nul* a N_0 i buits a la resta, però com que normalment no s'utilitza N_0, convencionalment es diuen nuls tots els elements del conjunt dels buits.

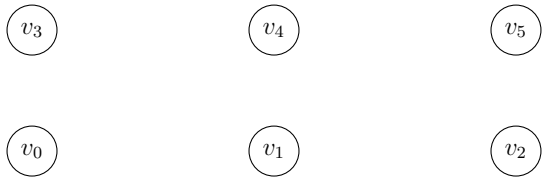


Figura 1.17: Graf buit N_6

1.4.12 Grafs regulars

Es diu que un graf $G = (V, E)$ és *regular de grau* r quan tots els seus vèrtexs tenen grau r. Formalment, un graf és r-regular quan $\Delta(G) = \delta(G) = r$. Un graf 0-regular és un graf nul [8], un graf 1-regular consisteix en arestes separades entre elles i un graf 2-regular consisteix en un o més cicles separats. A partir d'aquí, els grafs regulars més importants tenen noms propis, com per exemple els 3-regulars, que són els *cúbics*; els 4-regulars, *quàrtics*; els 7-regulars, *grafs de Witt truncats dobles*; els 8-regulars, *grafs de 24 cel·les*...

A la figura 1.18 es poden veure diversos exemples de grafs regulars.

Propietat 1 No necessàriament existeix un únic graf r-regular, sinó que sovint se'n poden fer amb diferent nombre de vèrtexs.

Propietat 2 Com a conseqüència del teorema 1, per a tots els grafs r-regulars amb n vèrtexs es compleix que

$$|E(G)| = \frac{1}{2}nr$$

on $|E(G)|$ és el nombre d'arestes.

[8] Veure apartat 1.4.11

(a) Graf cúbic (3-regular) (b) Graf 2-regular (c) Graf 4-regular

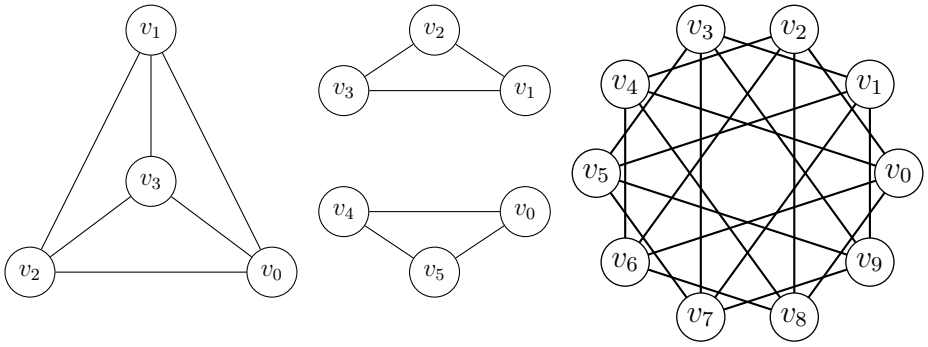


Figura 1.18: Exemples de grafs regulars

Capítol 2

Camins i algorismes

"Simplicity is a prerequisite for reliability"
Edsger W. Dijkstra

Sovint, quan utilitzem un graf per modelitzar quelcom, ens interessa poder-hi fer algunes operacions. Podem, per exemple, voler trobar un camí entre dos punts, recórrer el graf sencer o trobar el camí més curt per anar d'un vèrtex a un altre. Per aquest motiu utilitzem els camins, que trobarem o generarem mitjançant diversos algorismes. En aquesta secció se'n mostraran alguns.

2.1 Grafs ponderats i dirigits

2.1.1 Grafs ponderats

Els *grafs ponderats* són grafs on cada aresta e està associada a un nombre $w(e)$ anomenat *pes o cost*, tal que $w(e) \in \mathbb{R}$. El pes pot representar diverses quantitats, segons el que es vulgui modelitzar. Moltes vegades s'utilitza per representar distàncies, però si per exemple modelitzem una xarxa de distribució d'aigua, ens pot interessar representar el cabal de les canonades, o en una xarxa d'autobusos, la densitat de trànsit de cada tram.

Grafs dirigits

Els *grafs dirigits* són grafs les arestes dels quals només admeten un sentit. D'aquesta manera, una aresta $e_0 = (v_0, v_1) \neq e_1 = (v_1, v_0)$ [9]. De fet, no necessàriament ha d'existir una aresta contrària a una altra.

Aquest tipus de grafs poden ser útils per representar carreteres, o moviments vàlids en algun joc com les dames o els escacs.

2.2 Camins

Un *camí* p és una seqüència finita i ordenada d'arestes que connecta una seqüència ordenada de vèrtexs. Un camí p de longitud k (expressat com a $l(p) = k$) entre el vèrtex inicial v_0 i el vèrtex final v_k (sempre que $v_0 \neq v_k$) és una successió de k arestes i $k+1$ vèrtexs de la forma $\overline{v_0, v_1}, \overline{v_1, v_2}, \cdots, \overline{v_{k-1}, v_k}$. Per definició, també es pot representar un camí p entre v_0 i v_k com a successió de vèrtex $p = v_0 v_1 \cdots v_k$. En aquest cas, pot ser tractat com un graf elemental P_n.

Un cas especial és quan el camí comença i acaba al mateix vèrtex ($v_0 = v_k$). Llavors el camí és un cicle, i és l'equivalent a un graf cicle C_n.

Quan un camí té totes les arestes diferents, s'anomena *simple* i si, a més, té tots els vèrtexs diferents, s'anomena *elemental*.

En els grafs ponderats, la *longitud* d'un camí $p = v_0 v_1 \cdots v_n$ no es defineix pel nombre d'arestes per on passa el camí, sinó pel sumatori dels pesos de les arestes

$$\textbf{longitud}_w(c) = \sum_{i=0}^{n-1} w(\overline{v_i, v_{i+1}})$$

La *distància* entre dos vèrtexs v i u, $d_w(v, u)$, és la que s'obté a l'agafar la menor longitud d'entre tots els camins elementals entre v i u.

2.2.1 Camins Hamiltonians

Un camí Hamiltonià és un camí que passa una sola vegada per cadascún dels vèrtexs d'un graf. Un cicle Hamiltonià és un camí Hamiltonià en el qual el vèrtex inicial és igual al vèrtex final.

[9] Observi's la diferència de notació per a les arestes dels grafs dirigits i no dirigits, atenent al fet que l'ordre dels vèrtexs en un cas importa i en l'altre no.

A l'annex B.8 es pot trobar la implementació de l'algorisme que busca camins i/o cicles Hamiltionians.

2.2.2 Camins Eulerians

Un *camí Eulerià* d'un graf és un camí que recorre totes les arestes d'un graf una sola vegada. De la mateixa manera, un cicle Eulerià consisteix en un camí que utilitza totes les arestes d'un graf una sola vegada i el punt inicial i final és el mateix. Un graf connex té un camí Eulerià si i només si té 0 o 2 vèrtexs de grau imparell, i té un cicle Eulerià si i només sí tots els vèrtex del graf tenen grau parell.

A l'annex B.9 es pot trobar la implementació de l'algorisme que busca camins Eulerians.

2.3 Estructures de dades dels grafs

En els grafs, normalement s'ha de tractar una gran quantitat d'informació: nodes, arestes, pesos, sentits... Tota aquesta informació no és difícil de gestionar si el graf que s'estudia és petit, ja que fins i tot es pot dibuixar. El problema sorgeix quan el graf en qüestió és més gran, com per exemple podria ser una xarxa de clavagueram o de metro. Llavors la informació a tractar és molta, i és convenient organitzar-la en estructures de dades. Les estructures de dades solen ser utilitzades en la programació, però en aquest cas sol ser beneficiós usar-ne fins i tot quan no es treballa amb informàtica.

Les matrius i llistes d'adjacència són les estructures més utilitzades.

Matrius

Matriu d'adjacència

La *matriu d'adjacència* d'un graf és una matriu quadrada que conté informació respecte el nombre d'arestes que uneixen qualsevol parella de nodes, i s'organitza de la següent manera:

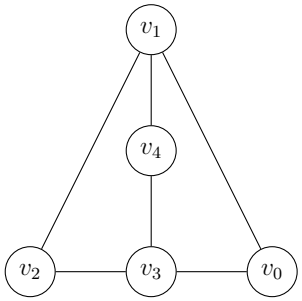
Si G és un graf amb $|V|$ nodes, $A(G) = (a_{ij})_{ij=1,\dots,|V|}$ és la seva matriu d'adjacència de $|V|\times|V|$, on a_{ij} correspon al nombre d'arestes que uneixen els nodes i i j, contant com a 2 els llaços, que sempre es trobaràn a la diagonal. La matriu serà simètrica si el graf ho ès, i podrem conèixer el grau d'un node i fent el sumatori de les

caselles de la i-èsima fila. A vegades, quan s'utilitzen grafs ponderats, les matrius d'adjacència s'omplen amb els pesos de les arestes, per no haver d'utilitzar altres estructures per emmagatzemar la resta d'informació. En aquest tipus de grafs, els llaços no necessàriament valdran 2, però es podran diferenciar perquè estaràn a la diagonal. En aquest cas, el problema serà que no es podran representar arestes múltiples amb pesos diferents. Tot i així, en termes de programació, utilitzar aquesta estructura de dades (sobretot en grafs grans) no és el més eficient, ja que la matriu creix molt ràpidament. La matriu d'un graf de $|V|$ nodes té $|V|^2$ espais, i en grafs poc densos, la majoria d'espais estan ocupats per 0, de tal manera que s'està utilitzant molta memòria que no conté informació útil. De la mateixa manera, si afegim un nou node al graf, que serà l'n-èssim, s'haurà d'afegir $2n-1$ espais a la matriu. Veure figura 2.1 cas (b).

Matriu d'incidència

La *matriu d'incidència* d'un graf G sense llaços, $I(G) = (b_{i,j})_{i=1,\ldots,|V(G)|,j=1,\ldots,|E(G)|}$, és la matriu binària de $|V(G)| \times |E(G)|$ on $b_{i,j}$ indica si l'aresta j és incident al node i. Veure figura 2.1 cas (c).

(a) Graf G



(b) A(G)

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

(c) I(G)

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Figura 2.1: Graf G amb les seves corresponents matrius d'adjacència i incidència

Llistes d'adjacència

Aquesta estructura de dades és molt utilitzada per tractar grafs, ja que ocupa menys memòria i només inclou la informació necessària. Les *llistes d'adjacència* d'un graf G consisteixen en un conjunt de $|V(G)|$ llistes. Cada llista correspon a un node del graf, i conté els nodes al quals és adjacent. En informàtica s'acostumen a fer mitjançant apuntadors, de tal manera que, a partir de cada element de la llista,

es pugui accedir a la seva pròpia llista, essent així molt més eficient fer iteracions [10]. Amb l'esquema següent es pot veure més clarament:

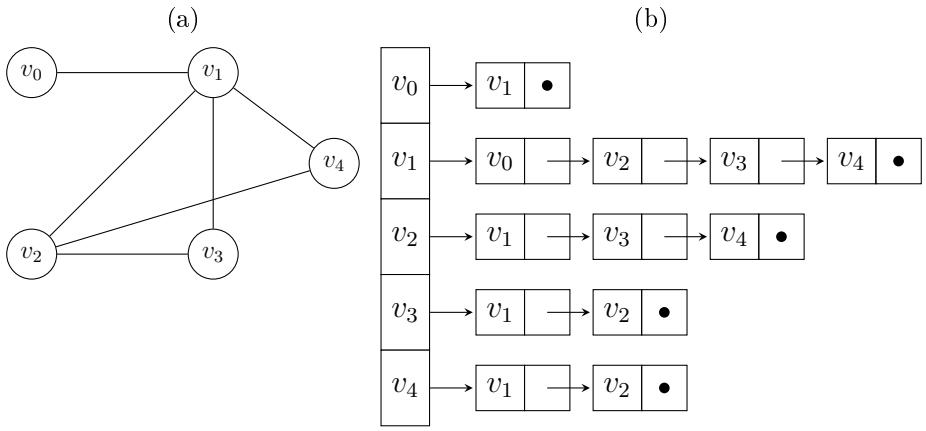


Figura 2.2: Un graf G (a) i la seva llista d'adjacència (b)

2.4 Algorismes

Un *algorisme* és un conjunt d'instruccions precises i ben definides que, donada una entrada, calculen la sortida corresponent segons les instruccions que té.

Per automatitzar diversos càlculs sobre grafs, molts científics han treballat desenvolupant processos universals que poden ser duts a terme per ordinadors. Aquests processos, anomenats *algorismes*, consisteixen en conjunts específics d'ordres que s'executen sobre unes estructures de dades que contenen el graf.

Per fer l'anàlisi dels algorismes, s'haurà de fer l'anàlisi de la cota superior asimptòtica, és a dir, analitzar el temps d'execució de l'algorisme quan l'entrada és la més gran i complexa possible. Per a aquest propòsit s'utiltizarà la notació de Landau, també anomenada notació de la gran "O". Els temps majoritàriament es calcularan en funció del nombre d'arestes o de nodes. Cal tenir present que tots els logaritmes que apareguin en funcions de Landau són logaritmes en base 2, sempre que no s'indiqui el contrari.

A continuació es mostren alguns algorismes que permeten fer diverses operacions sobre grafs.

[10] Una iteració és cadascuna de les repeticions d'un procés. Sovint els resultats d'una iteració confereixen l'entrada de la següent.

2.4.1 BFS

Aquest algorisme serveix per examinar l'estructura d'un graf o fer-ne un recorregut sistemàtic. La recerca per amplada prioritària (*breadht-first serch* en anglès, d'aquí **BFS**) fa l'exploració en paral·lel de totes les alternatives possibles per nivells des del vèrtex inicial. A la figura 2.3 es pot veure com funcionaria aquest algorisme en un graf. Per programar aquest algorisme s'acostuma a utilitzar un contenidor de tipus cua, que només permet afegir elements al final de la cua i treure'n de l'inici, sense poder accedir a elements del mig. El que farà el programa serà imprimir per pantalla la seqüència de vèrtexs ordenada segons l'ordre en què els ha visitat.

Funcionament

Donat un node incial v, es busquen tots els nodes no visitats que estiguin a distància i de v. Inicialment $i = 0$ i l'únic a distància 0 de v és ell mateix. Després, $i = 1$ i per tant, en aquesta iteració es visitaran tots els nodes adjacents a v. Quan $i = 2$ es visitaran els nodes adjacents als adjacents de v que no s'hagin visitat, i així consecutivament fins a visitar tots els nodes.

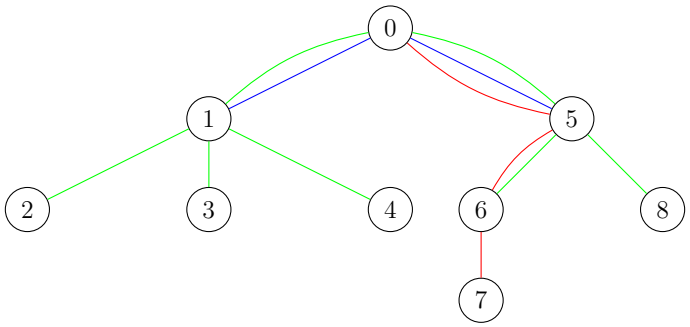


Figura 2.3: Exemple del recorregut que realitzaria l'algorisme

Pseudocodi

Algorisme 1: BFS

```
Data: Un graf G i un node inicial v
Result: Seqüència de de nodes visitats
nova cua Q
marca v com a visitat
afegeix v a la cua Q
nou node auxiliar
nou node següent
while la cua no estigui buida do
    auxiliar = primer element de Q
    imprimeix(auxiliar)
    elimina(primer element de Q)
    while hi hagi nodes adjacents a auxiliar i aquests no s'hagin visitat do
        marca adjacent(auxiliar) com a visitat
        afegeix adjacent(auxiliar) a la cua
    end
end
foreach node de G do
    marca'l com a no visitat
end
```

Aquesta és una manera bastant usual de programar l'algorisme BFS, i encara que és eficient, s'està desaprofitant propietats de l'algorisme. Amb l'algorisme BFS es pot saber a quina distància del punt inicial està cada node, el camí més curt per anar del node inicial a qualsevol altre i, fins i tot, es pot generar un arbre expansiu mínim, agafant les arestes per on 'passa' l'algorisme. El següent algorisme té en compte aquests detalls. Està pensat per ser implementat en el llengüatge Python, i per aquest motiu utilitza diccionaris (llistes on cada element té una clau i un valor), però en llenguatges basats en C, es poden utilitzar maps de la mateixa manera.

```
Data: Un graf G i un node inicial v
Result: Seqüència de nodes visitats, distància de cada node resperce v
nou diccionari dist
dist[v] = 0
nou diccionari anterior
anterior[v] = Nul
i = 0
nova llista frontera afegeix v a frontera
imprimeix(v)
while frontera no estigui buida do
    nova llista següent
    foreach node x de frontera do
        /* A cada iteració, x agafarà un valor diferent de frontera      */
        foreach node y adjacent a x do
            if y no existeix dins dist then
                dist[y] = i
                anterior[y] = x
                afegeix y a següent
                imprimeix(y)
            end
        end
    end
    frontera = següent
    i = i + 1
end
imprimeix(dist)
```

Veure annex B.1.

Aplicacions

Encara que aquest algorisme sembli molt senzill, ens pot aportar informació important, i fins i tot permet resoldre problemes senzills on haguem de trobar distàncies o el camí més curt entre dos nodes. Aquest algorisme s'utilitza també per operacions més complexes, com les següents:

- Buscadors com Google l'utilitzen per indexar pàgines web noves. Amb l'algorisme BFS pot recòrrer tota la xarxa d'internet sencera, i, si cada pàgina web és un node i cada enllaç és una aresta, si es posa un link d'una pàgina no indexada a una que sí ho està, l'algorisme trobarà el nou node.

- Les xarxes socials l'utilitzen per suggerir amistats. Amb aquest algorisme es poden trobar els amics d'una persona (els nodes que estan a distància 2 d'aquesta), que són susceptibles a ser amics seus. Com més amistats en comú amb la persona a distància 2, més probable és que es coneguin.
- Es pot sol·lucionar un cub de rubik. Si s'aconsegueix generar un graf on cada node sigui un estat diferent del cub i les arestes siguin un moviment d'una cara, donat un estat inicial, amb BFS es pot arribar a l'estat resolt amb els mínims moviments possibles.

2.4.2 DFS

La recerca per profunditat prioritària (*depth-first search* en anglès, d'aquí **DFS**) és un algorsime que utilitza uns principis semblants als de l'algorisme BFS, però en lloc de cobrir tota l'amplada d'un nivell abans de passar al següent, el que fa és cobrir tota la profunditat possible (arribar el més lluny possible) abans de tornar enrere.

Funcionament

L'algorisme DFS arriba el més lluny possible abans de retrocedir. Això significa que quan arriba a un node, se'n va a un d'adjacent no visitat i va emmagatzemant l'ordre amb el qual els visita. Quan un node no té nodes adjacents no visitats, torna al node anterior i continua a partir d'allà. A la figura 2.3 es mostra un graf amb els nodes numerats segons l'ordre en el qual es visiten i amb el recorregut que faria l'algorisme:

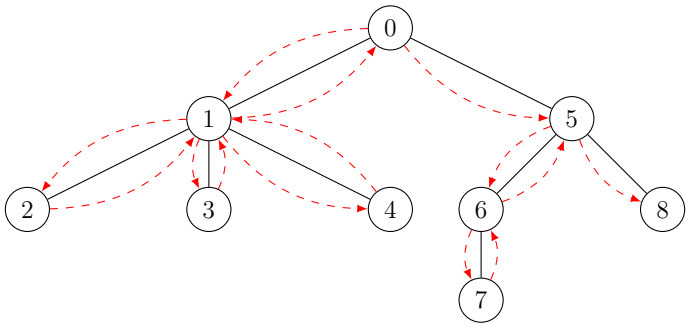


Figura 2.4: Exemple del recorregut que realitzaria l'algorisme

Pseudocodi

Tal com en el BFS, també hi ha diverses maneres d'implementar l'algorisme, i en presentaré dues. La primera utilitza un contenidor de tipus pila, on només es pot manipular, afegir o treure l'element de dalt de tot d'aquest.

Algorisme 2: DFS

```
Data: Un graf G i un node inicial v
Result: Seqüència de nodes visitats
nova pila S
nou node següent
marca v com a visitat
imprimeix(v)
afegeix v a la pila S
while la pila no estigui buida do
    següent = node adjacent no visitat de l'element superior de S
    /* En cas que no n'hi hagi cap, següent = Nul                    */
    if següent = Nul then
        elimina(element superior de S)
    else
        marca següent com a visitat
        imprimeix(següent)
        afegeix següent a S
    end
end
```

Aquest mètode, però, té un problema, i és que només funciona per a grafs no dirigits. Hi ha la possibilitat que, treballant amb un graf dirigit, un node del graf no sigui accessible des del node inicial que hem determinat. Aquest cas excepcional es pot arreglar fent que cada node no visitat per les iteracions anteriors sigui l'inicial. El segon algorisme, a part d'arreglar això, utilitza una funció recursiva [11].

[11] S'anomena funció recursiva a aquella que es crida a si mateixa

```
Data: Un graf G
Result: Seqüència de nodes visitats des de cada node
nou diccionari anterior
nova llista ordre
foreach node u del graf do
    if u no existeix dins anterior then
        imprimeix(u)
        anterior[u] = Nul
        DFSrecursiu(G, u)
    end
end
inverteix ordre
imprimeix(ordre)

/* La funció DFSrecursiu queda determinada pel següent algorisme:       */
Funció DFSrecursiu(G, v)
    foreach node x adjacent a v do
        if x no existeix a anterior then
            imprimeix(x)
            anterior[x] = Nul
            DFSrecursiu(G, x)
        end
    end
    /* Només si es vol obtenir la seqüència de recursió o ordenació
       topològica per a grafs dirigits acíclics,                        */
    afegeix v a ordre
```

Veure annex B.2.

Propietats

Propietat 1 En grafs dirigits es passa una vegada per cada aresta, mentre que en grafs no dirigits es passa dues vegades (una des de cada vèrtex).

Propietat 2 El segon algorisme, implementat en Python amb llistes d'adjacència, té un temps d'execució de $O(|V| + |E|)$, i per tant, un temps lineal.
Demostració: El bucle principal s'executa $|V|$ vegades (d'aquí $O(|V|)$). D'aquesta manera, s'executa la funció recursiva una vegada per vèrtex i, dins d'aquest, es

miren tots els nodes adjacents. El temps total que triguen totes les funcions recursives és de $O(\sum_{v \in V} |adjacents[v]|)$ que equival a $O(|E|)$.

Propietat 3 A través de l'algorisme es podrien classificar les arestes en 4 tipus diferents:

- Arestes de l'arbre: Són les arestes que pertanyen a l'arbre expansiu del graf.
- Arestes cap endavant: Són les que van d'un node cap a un dels seus descendents de l'arbre.
- Arestes cap enrere: Són les que van d'un node cap a un dels seus antecessors de l'arbre.
- Arestes creuades: Són les que no pertanyen a cap de les categories anteriors.

En el cas de grafs no dirigits, només hi ha arestes de l'arbre i arestes cap enrere.

Aplicacions

Aquest algorisme no té tantes utilitats pràctiques com l'algorisme BFS, però també té propietats útils, com per exemple, que passa per totes les arestes. A través d'ell podem obtenir informació importanat d'un graf:

- Es pot saber si un graf té cicles, comprovant si quan estem a la iteració d'un node (encara no n'hem explorat tots els nodes adjacents) el trobem a ell mateix. De la mateixa manera, es pot dir que un graf conté un cicle si i només si conté una aresta cap enrere.
- Es pot saber si un graf és bipartit assignant un color (entre un total de 2 colors possibles) a cada node mentre es recorre el graf, de manera que un node tingui un color diferent al dels nodes adjacents. Si això és possible voldrà dir que el graf és bipartit.
- Es pot dur a terme una ordenació topològica, si es tracta d'un graf dirigit sence cicles. Un exemple d'ordenció topològica és quan hi ha una llista de tasques a fer però per fer-ne una determinada, cal haver-ne fet primer una altra. Amb l'algorisme DFS, podem obtenir una de les seqüències vàlides per completar totes les tasques. A continuació es presenta un exemple d'ordenació topològica utilitzada durant la creació de la memòria d'aquest treball:

  Resulta que, a la secció 1.4, quan explicava un tipus concret de graf, sovint feia referència a altres tipus de grafs. Per determinar en quin ordre havien

d'estar els apartats, de tal manera que quan apareixès un tipus de graf (a una altra definició) aquest ja s'hagués explicat, es va utilitzar una ordenació topològica. Es va construïr un graf dirigit representant les dependències de cada apartat (veure figura 2.5), i es va executar l'algorisme d'ordenació topològica. El problema és que aquest graf té diversos cicles: no hi ha una seqüència lineal possible i les explicacions d'uns tipus de grafs queden supeditades a les explicacions d'altres. Això implica que en els diferents apartats no es pugui evitar fer referència a altres.

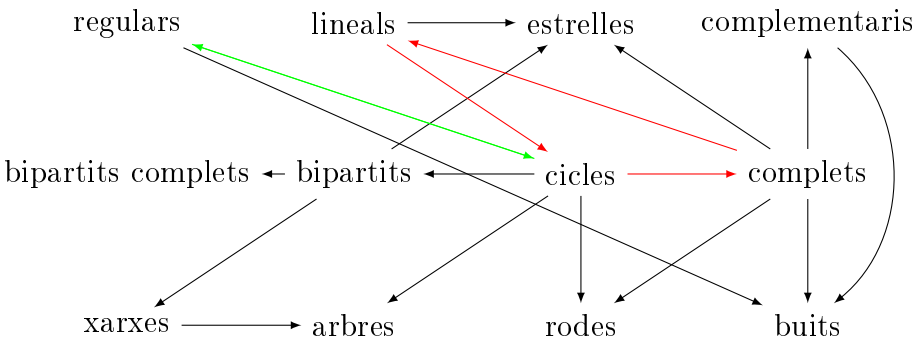


Figura 2.5: Graf de les relacions de dependència de les explicacions cada apartat

2.4.3 Dijkstra

Aquest algorisme, desdenvolupat per Edsger W. Dijkstra el 1956, serveix per trobar el camí més curt entre dos nodes d'un gaf ponderat. De fet, això és el que feia la variant original, però la variant presentada aquí troba el camí més curt entre un node inicial i tota la resta de nodes dels graf.

Funcionament

L'algorisme de Dijkstra es basa en dos principis:

- La *desigualtat triangular*: La desigualtat triangular com a tal diu que per a qualsevol triangle, la suma de dos dels seus costats serà igual o superior que al costat restant. Si a, b i c són les longituds dels costats d'unt triangle i c és la més gran, llavors $c \leq a + b$. Aplicant la desigualtat a un graf ponderat (sense pesos negatius) s'obté que per a tots els vèrtex $v, u, x \in V$, $d(v, u) \leq d(v, x) + d(x, u)$ (cal recordar que la distància entre dos vèrtexs ve donada pel camí més curt entre aquests vèrtexs).

- La *subestructura òptima*: En programació dinàmica, el concepte de subestructura òptima significa que es poden fer servir les solucions òptimes de problemes més petits per solucionar, de manera òptima, un problema més gran. En aquest cas en particular diem que un subcamí d'un camí mínim és també un camí mínim.

Donat un node inicial s, l'algorisme suposa que la distància mínima per arribar als seus nodes adjacents és simplement el pes de l'aresta que els uneix i per la resta de nodes és ∞. Aquesta serà una distància provisional, ja que posteriorment potser es trobarà un camí més curt. No obstant, hi ha una única distància que es pot determinar de manera definitiva: la del node v que està unit a s amb l'aresta de menys pes. Si existís un camí més curt entre s i v, aquest per força hauria de passar per algun node x adjacent a s, però per la desigualtat triangular $d(s, v) \leq d(s, x) + d(x, v)$.

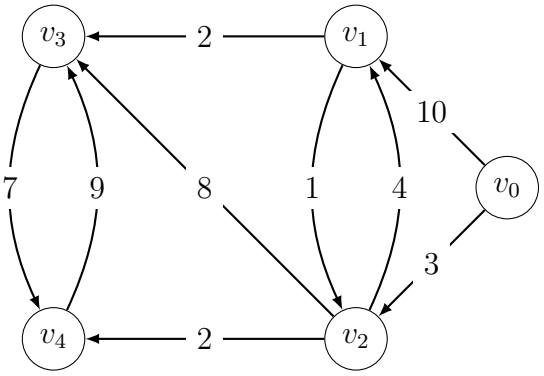
Això també es pot veure d'una manera més intuituva: si v està unit a s amb l'aresta de menys pes incident a s, per força l'aresta que uneix s i x ja té un pes superior a la primera i, per tant, el camí no podrà ser més curt.

Un cop fet aquest pas, s'obtenen les distàncies provisionals des del node s fins a tots els seus adjacents i el camí més curt entre dos nodes que, com a conseqüència de la subestructura òptima, forma part de l'arbre expansiu mínim.

A partir del node v, es busquen els seus adjacents i es calcula la distància a la qual estan respecte s (és a dir, la suma del camí acumulat i el pes de l'aresta). Si la nova distància calculada és menor a la provisional (cosa que passarà quan es trobi un camí més curt), la provisional passarà a ser la calculada. Un cop s'hagi fet tots els canvis necessaris, es buscarà el node no visitat que estigui a menor distància de s (i que per tant formarà part de l'arbre expansiu mínim), que passarà a ser v i es tornarà a fer el mateix procediment. Això s'anirà repetint fins que s'hagi visitat tots els vèrtexs.

En la figura 2.6 es mostra un graf i la taula amb les distàncies calculades per a cada iteració del programa. En aquest cas, el node inicial és v_0. A cada iteració s'agafa la distància més petita (marcada amb un quadre) i es calcularà la distància provisional dels seus nodes, sumant la distància acumulada fins al node més el pes de l'aresta. En cas de no conèixer una distància entre dos nodes, s'utilitzarà el valor ∞.

(a)



(b)

Distàncies provisionals

Iteració	v_0	v_1	v_2	v_3	v_4
1	0	10	[3]	∞	∞
2	|	7	|	11	[5]
3	|	[7]	|	11	|
4	|	|	|	[9]	|

Figura 2.6: Graf G (figura a) i taula amb les distàncies provisionals per als nodes de G en cada iteració (figura b)

Pseudocodi

Algorisme 3: Dijkstra

Data: Un graf ponderat G i un node inicial s
Result: Distància mínima entre s i la resta de nodes del graf, arbre expansiu mínim

```
nou diccionari dist
nou diccionari Q
foreach node v de G do
    Q[v] = ∞
    dist[v] = ∞
end
Q[s] = 0
while Q no estigui buit do
    u = minvalordeQ
    dist[u] = Q[u]
    foreach node v adjacent a u do
        if v existeix dins Q then
            if Q[v] > Q[u] + w(u,v) then
                /* w(u,v) és el pes de l'aresta (u,v)        */
                Q[v] = Q[u] + w(u,v)
            end
        end
    end
    elimina(Q[u])
end
imprimeix(dist)
```

Veure annex B.3.

Propietats

Propietat 1 L'algorisme de Dijkstra és el que s'anomena un *algorisme voraç*. Això significa que sempre agafa la millor opció en aquell moment, suposant que les opcions que es trobin posteriorment no la podràn millorar. Aquest és el motiu per el qual l'algorisme de Dijkstra no funciona amb pesos negatius. En un graf amb pesos negatius això no necessàriament es complirà.

Propietat 2 L'algorisme de Dijkstra no troba únicament el camí mínim entre un node i un altre perquè la millor manera de fer-ho que es coneix és precisament trobar els camins mínims des d'un node fins a tota la resta.

Propietat 3 L'algorisme té un temps d'execució de $O(|E| + |V|log(|V|))$.
Demostració: Les operacions que es fan són les següents:

- S'inicialitzen la cua i les distàncies per a cada node, operació que triga $O(|V|)$.
- Es busca el valor mínim de la cua que, en cas d'estar optimitzada, triga $O(log|V|)$. En cas de no estar optimitzada, s'hauria de recórrer tota la cua i anar comparant tots els elements, cosa que suposaria un temps de $O(|V|)$.
- Es calcula la distància provisional per a cada node que, en principi, tan sols ha de trigar $O(1)$.

La iteració principal acaba quan no queden nodes a la pila. Inicialment tots els nodes són a la pila i cada node es treu només una vegada. Per tant, per a cada node es buscarà una vegada el valor mínim de la pila. Això és un total de $O(|V|log(|V|))$ amb la cua optimitzada i $O(|V|^2)$ sense. El procediment per a calcular la nova distància s'executarà, com a màxim, una vegada per aresta, i com que el temps de cada operació és de $O(1)$, el total és de $O(|E|)$.

D'aquesta manera el temps d'execució total és de $O(|E| + |V|log(|V|))$ per cues optimitzades i de $O(|E| + |V|^2)$ per a cues normals.

Aplicacions

Aquest algorisme, encara que no pot treballar amb pesos negatius és molt útil i té una gran quantitat d'aplicacions pràctiques:

- Navegadors GPS, on les arestes són carrers i carreteres, els nodes cruïlles i els pesos distàncies. S'utilitza l'algorisme de Dijkstra per trobar els camins més curts entre dues destinacions.
- Problemes de canvis de divisa, on volem trobar la millor manera de canviar divises successives i guanyar més diners. Aquí els nodes són les diferents monedes o divises, les arestes les transaccions i els pesos les taxes de canvi. Amb aquest algorisme podem trobar la millor manera de fer els canvis de moneda. En aquest context prenen sentit les arestes amb pesos negatius, encara que l'algorisme de Dijkstra no els pot tractar.
- Els routers utilitzen l'algorisme per portar-te a través d'internet al servidor desitjat amb la menor quantitat de passos possibles.
- En robòtica s'utilitza per fer la planificació de moviment del robot. Cada node és una posició i representa una unitat d'espai i, omplint tot l'espai de nodes excepte els obstacles i executant l'algorisme en el graf resultant, s'obté el camí més òptim per arribar a la posició desitjada.
- En epidemologia aquest algorisme es pot utilitzar per modelitzar un grup de persones i els seus familiars per veure qui és més susceptible a emmalaltir. Això també pot funcionar entre ciutats o col·lectius més grans.

2.4.4 Bellman-Ford

Aquest algorisme té un funcionament i utilitats molt semblants a les de l'algorisme de Dijkstra, però té la particularitat de poder tractar sense problemes les arestes amb pesos negatius, mentre que l'algorisme de Dijkstra no ho permet. L'algorisme de Dijkstra es basa en la desigualtat triangular per tobar el camí més curt, però amb pesos negatius no es pot suposar que la desigualtat triangular es compleixi. A més, permet saber si un graf conté cicles negatius. Si un camí entre dos nodes conté un cicle de pes negatiu, no es pot trobar un camí mínim entre aquests dos nodes. Això es deu a que recorrent aquest cicle sempre es podria fer més curt el camí, i llavors el mínim possible seria de $-\infty$.

Funcionament

L'algorisme de Bellman-Ford utilitza una part de l'algorisme de Dijkstra, concretament la part on comprova si la distància suposada és major a la distància calculada en aquell moment. La diferència amb l'algorisme de Dijkstra és que, en lloc de seleccionar el node amb menor distància i fer l'operació sobre totes les seves

arestes incidents, simplement fa l'operació sobre totes les arestes del graf. Aquesta operació sobre les arestes es fa un total de $|V| - 1$ vegades i, a cada iteració, el nombre de nodes amb distàncies mínimes augmenta, fins que tots els nodes tenen la distància correcta.

Pseudocodi

Algorisme 4: Bellman-Ford

```
Data: Un graf ponderat G i un node inicial s
Result: Distància mínima entre s i la resta de nodes del graf, arbre
        expansiu mínim
nou diccionari dist;
nou diccionari anterior;
foreach node v de G do
    dist[v] = ∞;
    anterior[v] = Nul;
end
dist[s] = 0;
for i in range(0, len(Adj)-1) do
    foreach u dins Adj do
        foreach v dins Adj[u] do
            if dist[v] > dist[u] + w(u, v) then
                /* w(u, v) és el pes de l'aresta {u, v}          */
                dist[v] = dist[u] + w(u, v);
            end
        end
    end
end
foreach u dins Adj do
    foreach v dins Adj[u] do
        if dist[v] > dist[u] + w(u, v) then
            imprimeix("Hi ha cicles de pesos negatius");
        end
    end
end
imprimeix(dist);
imprimeix(anterior);
```

Veure annex B.4.

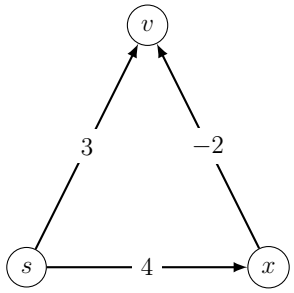


Figura 2.7: Graf amb arestes de pes negatiu on la desigualtat triangular no es compleix

Propietats

Propietat 1 L'algorisme de Bellman-Ford pot treballar amb pesos negatius ja que no es basa en la desigualtat triangular.

En el graf de la figura 2.7, l'algorisme de Dijkstra calcularia malament el camí mínim entre s i v: la distància provisional seria de 3 per arribar a v i de 4 per arribar a x. Llavors, com que 3 és la distància més petita que ha trobat (i suposa que és la correcta), visita v. Però v no té nodes adjacents, per tant acaba l'iteració del node v. El següent node no visitat amb mínim valor és x. Encara que v sigui adjacent a x, com que v ja ha estat visitat i per la desigualtat triangular s'assegurava que era el mínim, no el pot canviar.

En canvi, si s'executés l'algorisme de Bellman-Ford en aquest graf, el procediment que faria seria el següent: tal com l'algorisme de Dijkstra, posaria una distància provisional de 3 per arribar a v i de 4 per arribar a x. Ara visitaria v, però com que no té nodes adjacents, no hi faria res. A continuació visitaria x, que té v com a node adjacent. Com que la distància calculada des de x és de $4 - 2 = 2$ i $2 < 3$, simplement canviaria el valor de la distància entre s i v.

Propietat 2 Es pot aconseguir reduir el temps d'execució amb una implementació diferent de l'algorisme, que funciona per a grafs dirigits sense cicles. En aquesta nova implementació es fa una ordenació topològica del graf i es visiten, de manera ordenada, tots els nodes. A cada node es comprova si el pes suposat per arribar a cadascun dels seus adjacents es pot millorar. Aquesta implementació permet reduir el temps d'execució fins a $O(|V| + |E|)$.

Propietat 3 L'algorisme de Bellman-Ford té un temps d'execució de $O(|E| \times |V|)$. Es fan $|V| - 1$ iteracions i, a cada iteració, es comproven totes les arestes. A tot

això cal afegir-hi el temps de l'inicialització, que és de $O(|V|)$. Per simplificar-ho, es sol dir que té un temps d'execució de $O(|E| \times |V|)$, un temps que sempre serà lleugerament superior al temps real.

2.4.5 Kruskal

L'algorisme de Kruskal, descrit a [10], serveix per trobar un arbre expansiu mínim. Aquesta implementació de l'algorisme utilitza una estructura de dades especial, anomenada union-find. Aquesta estructura permet fer tres operacions diferents: crear conjunts (*make set*), determinar a quin conjunt està un element (*find*) i unir dos subconjunts en un de nou (*union*). L'ús d'aquesta estructura especialitzada fa que, en grafs petits o poc densos, l'algorisme sigui molt ràpid. En cavni, en grafs més grans, es relantitza el procés i, en aquest cas, és més eficient utilitzar-ne d'altres.

Funcionament

L'algorisme es basa en les següents propietats:

- Si un graf ponderat conté un cicle C i e és l'aresta de més pes de C, e no forma part de l'arbre expansiu mínim del graf.
  Demostració: Suposem que e pertany a l'arbre expansiu mínim. Llavors, si s'elimina e, l'arbre queda trencat en dos arbres, cadascun dels quals conté un extrem de e. Però la resta del cicle fa que els dos arbres siguin connexos i, en particular, hi haurà una aresta f que tindrà un extrem a cadascun dels arbres i que tindrà un pes inferior a e. Concretament f formarà part de l'arbre expansiu mínim.

- Si en un graf $G = (V, E)$ es fa un *tall*, es paretixen els vèrtex del graf en dos conjunts disjunts $(S, V(G) - S)$. D'entre les arestes que tenen un extrem a S i l'altre a $V(G) - S$, e és la de menor pes i forma part de l'arbre expansiu mínim.
  Demostració: Suposem que e no forma part de l'arbre expansiu mínim. Llavors, si s'afegeix e a aquest arbre expansiu, es genera un cicle C. Això comporta que hi ha una altra aresta f amb un extrem a S i l'altre a $V(G) - S$. D'aquesta manera, si es canvia f per e, es genera també un arbre expansiu que és mínim, ja que $w(e) < w(f)$.

El que fa l'algorisme és anar construint arbres expansius separats per acabar unint-los si el graf és connex. Per aconseguir-ho, l'algorisme crea un conjunt per a cada

node del graf (representant cadascun dels arbres), ordena les arestes amb ordre creixent segons el seu pes i les visita en aquest ordre. Quan es visita una aresta, s'uneixen els conjunts que les contenien (és a dir, es connecten els arbres). Si una aresta uneix nodes que formen part del mateix conjunt (és a dir, es crea un cicle), vol dir que l'aresta en concret no forma part de l'arbre expansiu del graf, tal com estableix la primera propietat.

En la figura 2.8 es mostra el procés que duu a terme l'algorisme de Kruskal. En aquesta figura cada color representa un conjunt diferent i, a cada pas, s'ajunten diversos grups dins d'un més gran, tenint en compte quines arestes els uneixen. Al final tan sols hi ha un únic grup i totes le arestes marcades formen part de l'arbre expansiu mínim.

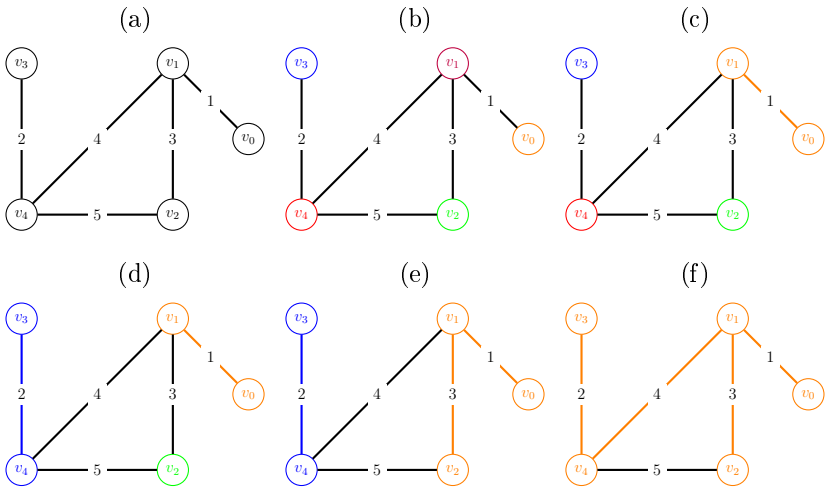


Figura 2.8: Passos corresponents a les diverses unions de grups que fa l'algorisme

Pseudocodi

```
Data: Un graf G i un node inicial s
Result: Arbre expansiu mínim de G
nova estructura union-find subgraf
nova llista arbre
ordena E(G) per ordre creixent de pesos
foreach node u de G do
    foreach node v adjacent a u do
        if subgraf[u]! = subgarf[v] then
            afegeix (u, v) a arbre
            union(subgraf[u], subgraf[v])
        end
    end
end
imprimeix(arbre)
```

Veure annex B.5.

Propietats

Propietat 1 Com que l'algorisme construeix l'arbre expansiu mínim a partir de diversos components inicialment disconnexos, encara que el graf no sigui connex funcionarà correctament. En particular, trobarà l'arbre expansiu mínim de cadascun dels conjunts de nodes connexos.

Propietat 2 El temps d'execució aproximat de l'algorisme de Kruskal és de $O(|E| \times log(|V|))$. L'operació d'ordenar les arestes triga $O(|E| \times log(|E|))$ i el total de les operacions Find i Union és de $O(|E| \times log(|V|))$. La suma total és de $O(|E| \times log(|V|) + |E| \times log(|E|))$. Tot i això, com que $|E|$ pot ser com a màxim $|V|^2$ podem escriure que $log(|E|) = log(|V|)$ ($log(|E|) = log(|V|^2) = 2 \times log(|V|)$, però en la notació de Landau $2 \times log(|V|)$ és essencialment $log(|V|)$). D'aquesta manera el temps total és de $O(2 \times (|E| \times log(|V|))) = O(|E| \times log(|V|))$.

2.4.6 Prim

L'algorisme de Prim, tal com el de Kruskal, serveix per trobar l'arbre expansiu mínim d'un graf ponderat no dirigit. Aquest algorisme funciona amb diccionaris,

estructures de dades més normalitzades que les Union-Find. Com a conseqüència, l'algorisme de Prim és més lent en grafs petits, però més ràpid en grafs molt densos.

Funcionament

L'algorisme de Prim es basa en la mateixa propietat dels talls que l'algorisme de Kruskal. L'algorisme de Prim divideix els nodes en dos grups S i $V(G) - S$, els que pot arribar amb les arestes de l'arbre que va construïnt i els que encara no. Sempre selecciona l'aresta de menys pes entre les que surten de nodes del primer grup i van a nodes del segon (ja que per la propietat de tall sabem que formarà part de l'arbre expansiu mínim), i afegeix el node final de l'aresta al primer grup (el que seria equivalent a la funció Union de l'algorisme de Kruskal). D'aquesta manera obté l'arbre expansiu mínim del graf. Aquest algorisme funciona per grafs connexos, però executant-lo per a cada node del graf, trobaria el *bosc* (conjunt d'arbres) mínim d'un graf no connex.

A la figura 2.9 es pot veure un exemple del funcionament de l'algorisme de Prim sobre un graf poderat. Els dos conjunts es representen mitjançant els colors taronja i negre, i les arestes que formin part de l'arbre expansiu mínim seràn les que estiguin pintades.

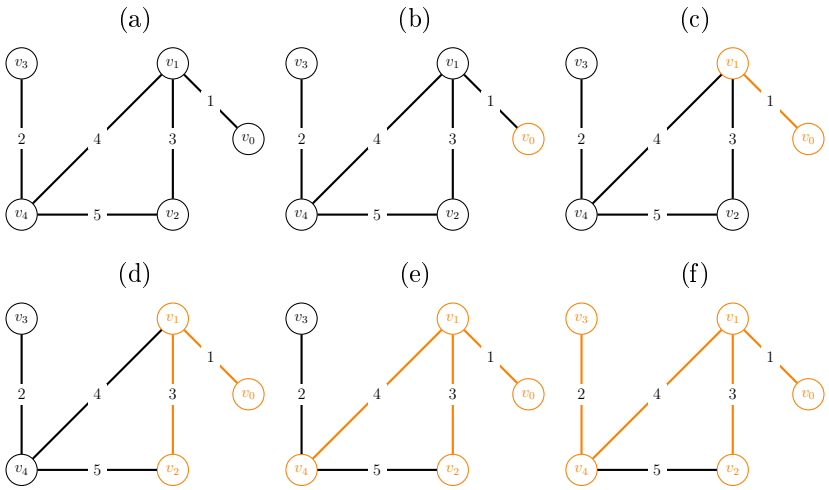


Figura 2.9: Passos corresponents a les fases d'expansió de l'arbre expansiu mínim

Pseudocodi

Algorisme 5: Prim

```
Data: Un graf G
Result: Arbre expansiu mínim de G
nou diccionari anterior
nou diccionari Q
foreach node v de G do
    Q[v] = ∞
end
Q[0] = 0
while Q no estigui buit do
    u = min{valor de Q}
    foreach node v adjacent a u do
        if v existeix dins Q then
            if Q[v] > w(u, v) then
                Q[v] = w(u, v)
                anterior[v] = 0
            end
        end
    end
    elimina(Q[u])
end
imprimeix(anterior)
```

Veure annex B.6.

Propietats

Propietat 1 En l'algorisme de Prim tan sols es necessiten dos conjunts, un que contindrà els nodes de l'arbre expansiu mínim i l'altre els que encara no en formen part. D'aquesta manera l'arbre creix en només un component, mentre que a l'algorisme de Kruskal l'arbre es forma en més components. Això fa que en grafs molt densos l'algorisme de Prim sigui més ràpid, però, com a contrapartida, no pot tractar amb grafs disconnexos.

Propietat 2 El temps d'execució aproximat d'aquesta implementació de l'algorisme de Prim és de $O(|E| + |V| \times log(|V|))$, però hi ha implementacions que

aconseguiexen temps de $O(|V|^2)$ o $O(|V| \times log(|V|))$. En aquest cas es busca el mínim valor de Q un total de $|V|$ vegades, i es fan $|E|$ comparacions de cost 1.

Aplicacions

Tant l'algorisme de Kruskal com el de Prim tenen aplicacions semblants, però segons la mida del graf és més convenient utilitzar-ne un o altre. Entre les aplicacions d'aquests dos algorismes hi ha:

- El disseny de xarxes de telèfon, aigua, gas, internet... En aquestes xarxes s'ha d'arribar a tots els punts on s'ha de fer la distribució, i amb l'arbre expansiu mínim es pot assegurar que la xarxa és la més curta possible.
- Els arbres expansius mínims es poden utilitzar per generar laberints.
- S'utilitzen com a subrutines (o funcions) d'algorismes més complexos.

2.4.7 Floyd-Warshall

L'algorisme de Floyd-Warshall és un algorisme que permet calcular les distàncies entre tots els nodes d'un graf ponderat. Per fer-ho, compara els pesos de tots els camins possibles. A cada iteració es defineix el conjunt de nodes que pot tenir cada camí i, si ja s'ha trobat un camí amb els mateixos extrems, es compara el pes total d'ambdós. El conjunt és de la forma $1, 2, \cdots, k$ i a cada iteració es va incrementant k (a l'inici $k = 0$). El resultat d'aquest algorisme és una matriu quadrada D de $|V| \times |V|$ on $D_{ij} = w(i, j)$.

Funcionament

Per funcionar, l'algorisme de Floyd-Warshall manté una matriu $D^{(x)}$ de mida $|V| \times |V|$, on $D^{(x)}_{vu}$ és la logitud del camí mínim entre v i u que passa com a màxim per x nodes. A cada iteració principal s'afegeix un node al conjunt de nodes que poden ser utilitzats, i es recalculen els valors de la matriu de la següent manera:

$$D^{(x)}_{vu} = min(D^{(x-1)}_{vu}, D^{(x-1)}_{vk} + D^{(x-1)}_{ku})$$

on $D^{(x)}$ és la matriu on hi ha les distàncies amb camins de, com a màxim, x nodes intermitjos.

Pseudocodi

Algorisme 6: Floyd-Warshall

```
Data: un graf G
Result: una matriu quadrada amb les distàncies entre tots els nodes
nova matriu dist de |V|x|V|
for i in range(0, |V|) do
    for j in range(0, |V|) do
        if i = j then
            dist[i][j] = 0
        else
            dist[i][j] = ∞
        end
    end
end
foreach node u de G do
    foreach node v adjacent a u do
        dist[u][v] = w(u, v)
    end
end
for x in range(0, |V|) do
    for u in range(0, |V|) do
        for v in range(0, |V|) do
            if dist[u][v] > dist[u][x] + dist[x][v] then
                dist[u][v] = dist[u][x] + dist[x][v]
            end
        end
    end
end
```

Veure annex B.7.

Propietats

Propietat 1 Assumint que l'entrada de l'algorisme és $D^{(0)}$, el temps d'execució de l'algorisme és de $O(|V|^3)$. Tan sols cal veure que hi ha tres bucles, un dins l'altre, i cadascun fa $|V|$ iteracions.

Aplicacions

Aquest algorisme es pot utilitzar per a qualsevol de les aplicacions en que s'utilitzaria Dijkstra amb més d'un node. S'utilitza sobretot quan es vol mantenir una base de dades de pesos precalculats, per no haver d'executar Dijkstra en cada cas concret. A part d'aquesta aplicació, s'utilitza també d'altres maneres:

- Per detectar cicles de pes negatiu, cosa que passarà quan $D_{ii} < 0$, quan a la diagonal de la matriu hi hagi un valor negatiu.
- Estudiar la *clausura transitiva* d'un graf, és a dir, veure quins nodes són accessibles des de cada node. Això es pot veure a la matriu resultant, on els valors ∞ indiquen que no es pot accedir a el node concret.

2.4.8 Coloració de grafs

La coloració de grafs és un problema que consisteix a pintar els nodes d'un graf de manera que dos nodes adjacents no tinguin el mateix color. El primer problema plantejat en aquesta branca va ser proposat per Francis Guthrie [12] i consistia a veure quans colors eren necessaris per pintar qualsevol mapa de manera que es complissin aquests mateixos requisits.

La majoria d'algorismes que realitzen aquest procediment són lents i relativament poc eficients. Per aquest motiu vaig decidir no utilitzar els que vaig trobar i dissenyar-ne un jo mateix.

Funcionament

El que fan la majoria d'algorismes és posar un color a un node i assignar-ne de diferents als seus adjacents, però aquest procediment no sempre troba una distribució eficient, ja que escull arbitràriament entre els diversos colors a l'assignar-los a un node. A més, el resultat que dóna l'algorisme depèn del node en el qual es comenci a executar. Amb l'algorisme que he dissenyat, s'utilitzen estrictament els colors necessaris, establint un ordre específic en els grafs de manera que hi ha una única distribució possible per a cada graf.

Per establir aquest ordre específic s'ordenen els nodes en funció del nombre de nodes adjacents. D'aquesta manera es comença assignant un color als nodes amb més grau i, per tant, més complicats de pintar en cas que ja hi hagi nodes adjacents

[12] Veure apartat 1.1.2

pintats. Un cop els nodes estan ordenats, s'agafa un color c (representat per un enter que inicialment és 0) i es pinta el node amb grau més alt. A continuació, i per nombre decreixent de grau, es comprova per a tots els nodes del graf si són adjacents a un altre pintat amb color c. En cas de no ser-ho, es pintaran també amb c i la resta es deixaràn igual. Un cop s'hagi recorregut tots els nodes, s'establirà un altre color i es repetirà tot el procediment fins que tots els nodes tinguin un color assignat.

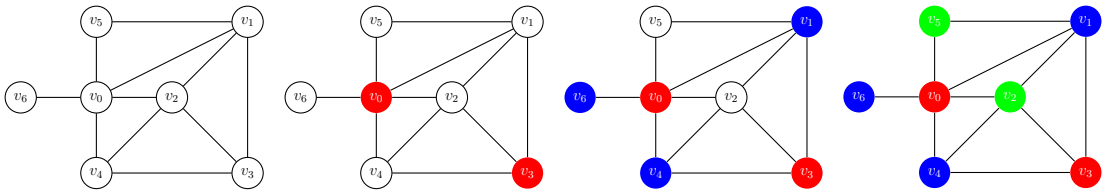


Figura 2.10: Procés de coloració que segueix l'algorisme

Pseudocodi

Algorisme 7: Coloració

```
Data: un graf G
Result: conjunt de nodes de G amb el color corresponent
nova llista nodes
nova llista colors
nou enter color_actual
nou booleà usat
usat = False
actual = 0
for i in range(0, |V|) do
    colors[i] = Nul
end
nodes =vèrtexs de G ordenats de manera decreixent segons el seu grau
color[nodes[0]] = 0
while no tots els nodes tinguin assignat un color do
    foreach node v de G do
        if colors[v] == Nul then
            foreach node u adjacent a v do
                if colors[u] == color_actual then
                    usat = True
                    break
                end
            end
            if usat == False then
                colors[v] = color_actual
            end
            usat = False
        end
    end
    color_actual = color_actual + 1
end
```

Veure annex B.10.

Propietats

Propietat 1 En el pitjor dels casos, l'algorisme triga $O(|V|^3)$.

Demostració: El pitjor dels casos pot ser que aquest algorisme rebi un graf complet K_n [13]. Aleshores, caldrà executar $n = |V|$ vegades el bucle principal, on cada vegada recorrerà tots els nodes del graf (un total de $|V|$) i els seus adjacents ($|V|$). D'aquesta manera es recorrerà 3 vegades el graf sencer, trigant un temps total de $O(|V|^3)$.

Propietat 2 Si el que es vol és pintar les arestes d'un graf en lloc dels vèrtexs, només cal executar l'algorisme sobre el graf lineal d'aquest i, un cop pintat, tornar a muntar el graf original.

Aplicacions

- Per la manera com està fet l'algorisme, es pot comprovar si un graf és bipartit. Tota execució de l'algorisme en un graf bipartit utilitzarà tan sols dos colors.
- S'utilitza la coloració de grafs per a problemes que requereixen separar els nodes en grups disjunts de manera que dos nodes del mateix grup no siguin adjacents.

[13] Cal observar que en un graf K_n tots els nodes estan connectats amb la resta, i que, per tant, es necessiten n colors

Capítol 3

Disseny de grafs

Fins ara, s'han mostrat i estudiat grafs que ja estan definits sobre els quals s'ha de fer alguna operació. Ara bé, en aplicacions reals de teoria de grafs, sovint no hi ha un graf determinat, sinó que s'ha de generar. Cal observar que, en aquesta part de disseny, els grafs són físics, és a dir, que existeixen com a tals situats en alguna superfície. En aquest cas no s'utilitzen com a modelitzacions abstractes i, per tant, les distàncies i situacions dels punts són importants.

3.1 Punt de Fermat i l'arbre de Steiner

Normalment, quan s'utilitzen els grafs com a models matemàtics de situacions reals, els nodes ja estaran determinats i el problema consistirà a trobar les arestes. Si això és així, segurament es podran afegir altres nodes. Un exemple d'aquest cas consistiria a haver d'unir tres ciutats amb carreteres de tal manera que des d'una es pugui arribar directament a les altres dues. La primera solució en què sol pensar una persona és fer un triangle en el qual les ciutats siguin els vèrtexs (l'equivalent a un graf K_3). Aquesta sol·lució és òptima si es vol anar d'una ciutat a l'altra amb la mínima distància possible, però si es vol construïr el mínim de carreteres possibles per a la xarxa conjunta, aquesta sol·lució no és la millor. En aquest cas, el millor és posar un altre node en el punt de Fermat del triangle que formen i obtenir un graf S_4. Donat un triangle acutangle, el punt de Fermat (també anomenat $X(13)$) és el punt tal que la suma de les distàncies des de cada vèrtex fins a aquest punt és la mínima. El punt de Fermat es determina gràficament amb el procediment següent:

- Donat un triangle T, es generen tres triangles equilàters a partir de cadascun

dels costats de T.

- S'uneixen els nous vèrtexs externs dels triangles equilàters amb els vèrtexs oposats de T.
- L'intersecció d'aquests tres segments dóna la posició del punt de Fermat.

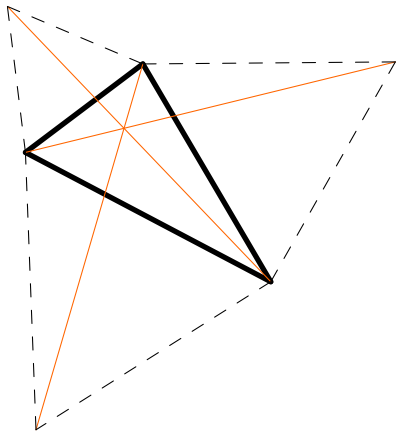


Figura 3.1: Procés de construcció del punt de Fermat per a un triangle

Aquest procediment és vàlid per a triangles amb angles menors a 120º, però en cas contrari, el punt de Fermat s'identificarà amb el vèrtex corresponent a l'angle que els superi.

El punt de Fermat es pot generalitzar per a altres polígons convexos a partir de triangulacions d'aquests. Un exemple senzill és el del quadrilàter, on en fer les diagonals ja es generen quatre triangles. En aquest cas, trobant els punts de Fermat en dos dels triangles oposats i unint-los ja es connecten tots els punts del quadrilàter amb la menor distància possible.

El problema de connectar amb la mínima distància un nombre determinat de punts essent possible afegir nodes és conegut com el problema de *l'arbre de Steiner*. Per aquest problema s'ha demostrat que l'arbre òptim té com a màxim $n-2$ nodes afegits (essent n el nombre inicial de nodes), que aquests tenen sempre grau 3, i que formen sempre angles de 120º.

3.1.1 Els arbres de Steiner amb bombolles de sabó

El problema dels arbres de Steiner és relativament complex per a conjunts de molts punts, i a més, actualment no hi ha bons algorismes per trobar-ne la solució. Un dels procediments que es pot utilitzar és omplir l'espai delimitat pel polígon que formen els punts inicials amb altres punts com si es tractés d'un graf xarxa i trobar el camí mínim que uneix els punts inicials. Tot i això, és tan sols una aproximació i, a més, no gaire precisa.

Les bombolles de sabó, sense utilitzar algorismes, són capaces de trobar arbres de Steiner amb precisió donats diversos punts. A les imatges de la figura 3.2 es poden veure diferents arbres de Steiner generats a partir de submergir les plaques amb els punts en sabó. Aquests muntatges consisteixen en parells de plaques transparents paral·leles unides per claus situats en els punts corresponents als nodes. Com es pot observar a la figura i tal com prediuen les lleis de Plateau, tots els angles que es formen són de 120º. De la mateixa manera, també es compleixen les altres propietats dels arbres de Steiner.

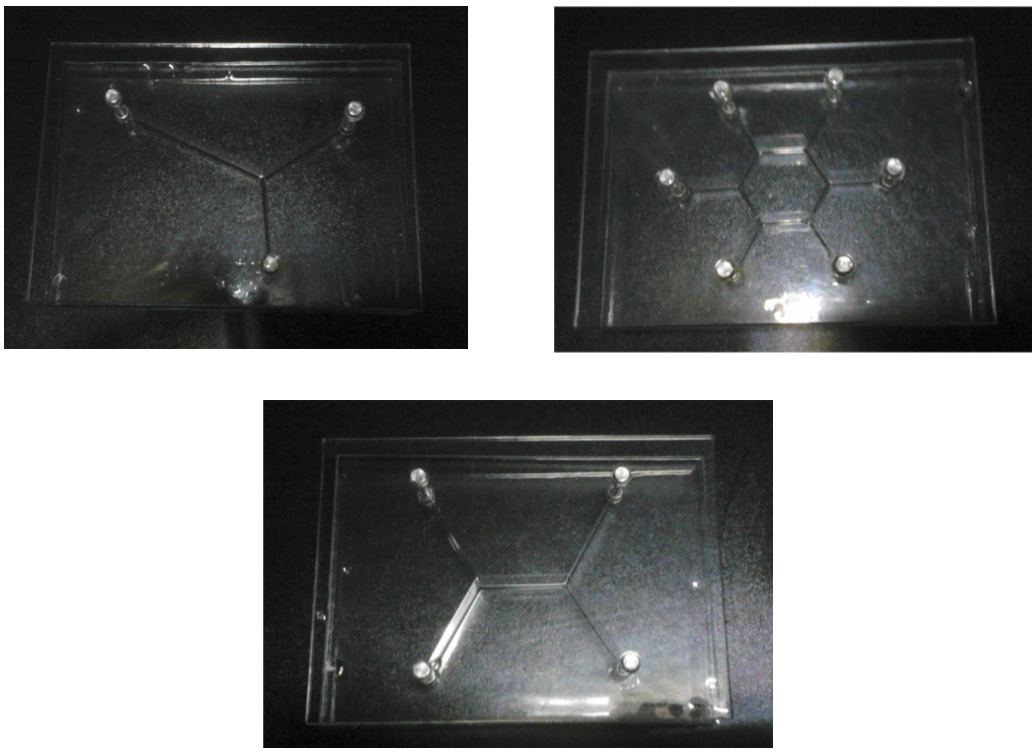


Figura 3.2: Arbres de Steiner amb bombolles de sabó [14]

[14] Autor: Anton Aubanell

3.2 Arbres expansius

Quan hi ha una situació on no es pot afegir cap node, el millor sol ser utilitzar els algorismes per trobar arbres expansius que ja s'han vist. Una opció és construir el graf complet amb totes les arestes i els seus corresponents pesos i executar algun algorisme que en trobi l'arbre expansiu mínim.

Capítol 4

Aplicacions pràctiques de la teoria de grafs

4.1 Algorisme PageRank

Una de les aplicacions de teoria de grafs que utilitzem cada dia és l'algorisme PageRank de Google. A través d'aquest algorisme, el cercador pot saber quines pàgines web són més valorades per la comunitat d'usuaris en general i, per tant, en quin ordre s'han de presentar els resultats d'una cerca.

Una versió simpificada d'aquest algorisme (l'utilitzat actualment és molt complex i s'escapa de la comprensió de l'autor) compta el nombre de links que s'adrecen a la pàgina web, així com la importància de les pàgines web on estan posats aquests enllaços, per saber quina importància té una web. La presuposició que fan és que quans més links cap a una web concreta i més important és la web d'origen, més important és la web de destí.

S'ha de pensar el conjunt de pàgines web com un graf dirigit i ponderat: cada vèrtex representa una pàgina i hi ha una aresta dirigida des del vèrtex v_i fins a v_j si hi ha un enllaç a v_i que porta a v_j. El pes de l'aresta $e = (v_i, v_j)$, $w(e)$, serà la probabilitat de passar del vèrtex v_i al vèrtex v_j (és a dir, $\frac{1}{n}$, on n és el nombre d'arestes que surten de v_i).

Un cop es té la matriu de probabilitats (la matriu d'adjacència del graf) es fa que un programa o bot navegui de manera aleatòria per aquest graf, d'acord amb les probabilitats de passar per cada aresta. Quan el programa ha estat un temps recorrent els nodes del graf, es fa el recompte de quantes vegades ha passat per cada pàgina i, a partir d'aquest valor, es determina la posició en els resultats.

4.2 Comerç online: productes relacionats

Quan es visita una botiga online i s'està mirant un producte, sovint apareix una secció de productes recomanats o relacionats. Hi ha diverses maneres de saber quins productes estan relacionats amb aquell que s'està mirant, i una de les que més s'utilitza es basa en teoria de grafs.

El procediment consisteix en anar construint un graf a mesura que els usuaris van visitant la web: cada producte és un vèrtex, i es genera una aresta dirigida cap a un altre producte quan aquest és visitat immediatament després del primer. Per exemple, si un usuari busca un ordinador portàtil, el més probable és que a l'acabar de mirar-ne un, en visiti un altre. En aquest cas es generaria una aresta des del primer ordinador fins el segon. A mesura que els usuaris van utilitzant la web, es van generant arestes i el graf va creixent. Tot i això, és possible que l'usuari, a l'acabar de mirar l'ordinador, es posi a buscar un altre article que no hi té res a veure, com per exemple un paraigües. Però això no és habitual i, per tant, hi haurà molt poques arestes que uneixin l'ordinador i el paraigües.

Sobre aquest graf s'executarà un algorisme de clusterització, que buscarà parts del graf que siguin més denses o altament connexes. Aquestes parts segurament seran d'una temàtica concreta, com per exemple el grup dels ordinadors i hi haurà molt poques arestes que surtin d'un grup cap a un altre. Així doncs, els productes relacionats seran aquells que es trobin dins del mateix clúster.

4.3 Instal·lació de càmeres de videovigilància

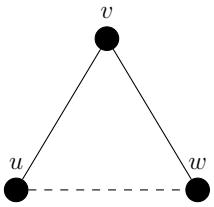
Un altre exemple és la optimització de sistemes de videovigilància. Amb la teoria de grafs es pot saber quin és el mínim nombre de càmeres necessàries per poder vigilar un local, tingui la forma que tingui. Aquest problema és conegut com el problema de la galeria d'art.

La intuició diu que, per una sala poligonal amb n vèrtexs, es pot posar una càmera a cada cantonada (que resulta un total de n càmeres) o bé, de manera més òptima, posant una càmera cada dos vèrtexs (que comporta tenir $\frac{n}{2}$ càmeres). Tot i això, Václav Chvátal afirma en el seu teorema que sempre se n'hauran d'utilitzar, com a màxim, la part entera de $\frac{n}{3}$.

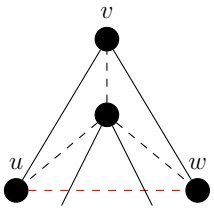
4.3.1 Demostració i procediment

La resolució d'aquest problema es basa en el fet que qualsevol polígon pot ser descompost en triangles i, més concretament, que sempre es pot traçar una corda en el polígon. Per demostrar-ho, s'agafa un vèrtex v qualsevol del polígon. Poden passar dues coses:

- La corda que formen els vèrtexs adjacents a v, que anomenarem u i w, està completament dins el polígon. En aquest cas es pot traçar la corda sense problemes.



- El propi polígon creua la corda entre u i w . Això voldrà dir que en l'espai delimitat pel triangle que formarien v,u i w hi haurà, com a mínim, un altre vèrtex del polígon. En aquest cas també es podran traçar diverses cordes amb aquest vèrtex i triangular d'una altra manera.



Si s'agafa el polígon triangulat com a graf (on els vèrtexs del polígon són els nodes i els costats i cordes són les arestes), els vèrtexs d'aquest poden ser pintats mitjançant tres colors, de tal manera que cada triangle tingui un vèrtex de cada color (d'aquí l'aproximació de $\frac{n}{3}$). Qualsevol conjunt de vèrtex d'un mateix color és un conjunt de posicions vàlides per a les càmeres de seguretat. Tan sols cal veure quin és el conjunt de menys elements per poder determinar les posicions vàlides de les càmeres.

Cal puntualitzar que hi ha casos on, amb aquest procediment, no es troba la solució òptima al problema; és tan sols una bona aproximació.

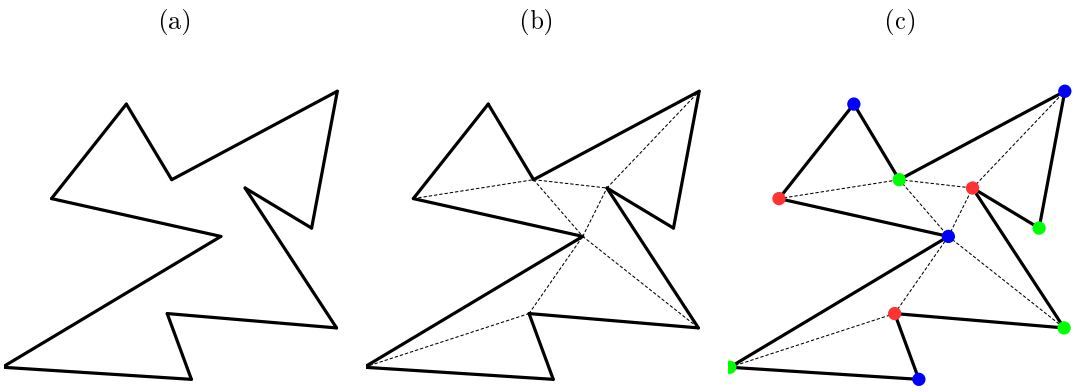


Figura 4.1: Exemple del procés de triangulació i coloració d'un polígon irregular

4.4 Xarxa de metro de Barcelona

Una xarxa de metro és un altre bon exemple de graf: es pot pensar com un conjunt de grafs elementals P_n (equivalents a les diverses línies) amb la propietat que, alguns d'aquests grafs, comparteixen nodes. Així doncs, com a una última aplicació pràctica, em vaig proposar fer un programa que sigués capaç de trobar un trajecte entre dues estacions qualssevol i calcular el temps de recorregut aproximat, inclosos els possibles transbords entre línies. En el cas que hi hagués més d'un trajecte possible, l'algorisme escollirà el més òptm que, en aquest cas, serà el que inverteixi menys temps.

Aquest algorisme que he generat és aplicable a qualsevol altra xarxa de transport públic; només caldrà disposar de la llista d'adjacència del graf en qüestió que inclogui els temps de trajecte i, a més, afegir-hi els temps estimats de parada.

4.4.1 Metodologia de treball

El primer gran repte consistia a representar l'àmplia xarxa de metro de Barcelona en forma de graf. Cada estació correspon a un node. Dos nodes seran adjacents si i només si les estacions estan connectades, ja sigui a través del propi metro o bé a través de passadissos (quan diverses línies comparteixen estació). Això comporta representar totes les adjacències de cadascuna de les estacions. Per fer-ho, vàrem dividir la xarxa total en les diferents línies. S'hi ha inclòs també les linies 6, 7 i 8 que corresponen als Ferrocarrils de la Generalitat de Catalunya i, per tant, hi ha

un total de dotze línies. Per poder tenir en compte els temps de transbord entre línies que comparteixen estació, s'han creat nodes, cadascun dels quals representa una estació diferent per a cada línia, unides per una aresta de pes proporcional al temps que es triga en fer el transbord a peu. Així doncs, per exemple, existeix un node corresponent a l'estació de Catalunya de la línia 1 (1_Catalunya) i un altre node que representa l'estació de Catalunya per a la línia 3 (3_Catalunya).

Un cop l'adjacència del graf va estar completa, calia posar pesos a cadascuna de les arestes. Aquests pesos corresponen al temps de viatge entre estacions mitjançant el metro o, en cas dels transbords, el temps caminant entre les andanes. Per aconseguir aquests temps, es va anar un dia a recórrer la xarxa sencera, tot cronometrant els intervals entre estacions, les aturades a les andanes i els transbordaments. Tots els temps van ser introduïts al graf en segons, per tal de facilitar el posterior càlcul del temps total.

4.4.2 Consideracions

Cal remarcar que els temps introduits al graf són els temps que va trigar el metro el diumenge 30 d'octubre de 2016 i que, aquests, no són constants, sinó que poden variar en funció del dia, l'hora i molts altres factors imprevisibles. El mateix passa amb el temps de les parades. Es va observar que el temps que el metro estava aturat en una estació s'incrementava en intervals de 5 segons a mesura que s'anava apropant al centre de la ciutat a partir de determinades estacions i a mesura que anava augmentant l'afluència de passatgers. Com que és un factor que també varia en funció de molts paràmetres i massa complex per tenir-lo en compte, s'ha establert que el temps de parada és d'uns 25 segons de mitjana. Aquest temps s'afegeix al total multiplicat pel nombre de parades menys dues (la inicial i la final). Així doncs, els valors de temps que calculi el programa seràn relatius i aproximats.

El format de les estacions en el graf és el següent:

(número de línia)_(nom d'estació)

Els noms de les estacions estan posats d'acord amb el plànol de la xarxa de metro que proporciona TMB (Veure figura 4.2).

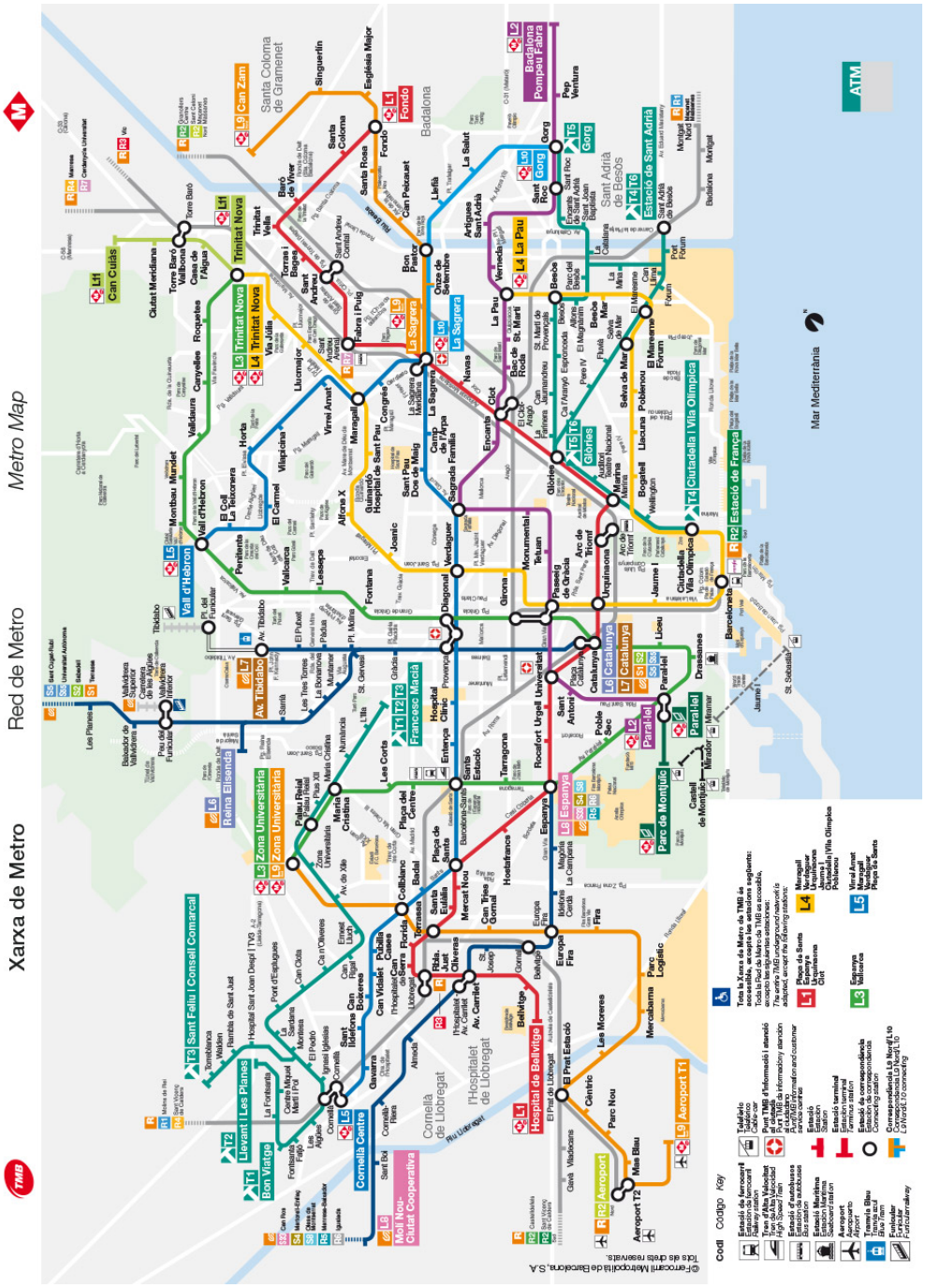


Figura 4.2: Plànol de la xarxa de metro de Barcelona [15]

[15]Autor: Ferrocarril Metropolità de Barcelona, S.A. Tots els drets reservats a l'autor

4.4.3 Algorisme

El programa que fa els càlculs funciona a partir d'una execució de l'algorisme de Dijkstra (en realitat en una versió modificada que pot tractar amb paraules com a noms de nodes).

```
Funció DijkstraOrdenat(G, s)
    nou diccionari dist amb les mateixes claus que G
    nou diccionari Q amb les mateixes claus que G
    nou diccionari arbre
    foreach node v de G do
        Q[v] = ∞
        dist[v] = ∞
    end
    Q[s] = 0
    while Q no estigui buit do
        u = min{valor de Q}
        dist[u] = Q[u]
        foreach node v adjacent a u do
            if v existeix dins Q then
                if Q[v] > Q[u] + w(u, v) then
                    Q[v] = Q[u] + w(u, v)
                    arbre[v] = u
                end
            end
        end
        elimina(Q[u])
    end
    return dist, arbre
```

Dijkstra retorna un primer diccionari amb els temps des del node inicial s fins a la resta de nodes i un segon diccionari amb l'arbre expansiu que té s com a arrel. Per saber el recorregut a fer, només cal seguir l'arbre de manera inversa, és a dir des del punt final fins a s. Al ser un arbre, només hi haurà un camí possible a seguir.

Per obtenir el temps caldrà buscar l'entrada corresponent al node final al diccionari dels temps. A aquest temps se li afegiran els temps per a cada parada i es farà la conversió de segons a minuts i segons.

El temps addicional corresponent a les parades, en un principi, s'afegiria al total,

multiplicat pel nombre de parades menys dues (la inicial i la final), però es va observar que aquest procediment no era del tot correcte. Per tal de veure el problema, considerem el següent graf:

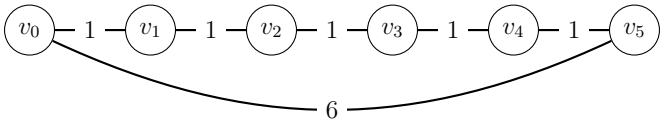


Figura 4.3: Exemple abstracte de xarxa de metro

En aquest graf, el camí mínim per anar de v_0 a v_5 segons Dijkstra és passant per la resta de nodes. Posteriorment l'algorisme del metro afegiria els 25 segons multiplicat pel nombre de parades menys una, afegint llavors 125 segons. El resultat total és de 130 segons. Però el camí mínim no és aquest: si no s'utilitza Dijkstra i s'agafa l'aresta directa entre v_0 i v_5 s'obté un temps de 6 segons al que només cal sumar 25 segons, trigant un total de 31 segons. Així doncs, l'algorisme era incorrecte, ja que també cal tenir en compte el nombre de nodes per on es passa. Aquest error està sol·lucionat en el següent algorisme:

Data: Graf G del metro, un node inicial *inici* i un node final $final$
Result: Temps total del trajecte, trajecte
nou vector *recorregut*
imprimeix("Punt inicial:", *inici*)
imprimeix("Punt final:", $final$)
dist, arbre = DijkstraOrdenat(G, *inici*)
$i = final$
while $arbre[i]$ *sigui diferent a inici* **do**
    afegeix $arbre[i]$ a *recorregut*
    $i = arbre[i]$
end
afegeix *inici* a *recorregut*
inverteix l'ordre dels elements de *recorregut*
$TempsTotal = dist[final] + (25 \times (len(recorregut) - 2)$
minuts = part entera de $TempsTotal/60$
segons = (residu de $TempsTotal/60) \times 0'60$
imprimeix("Temps total del recorregut:", *minuts*, "minuts i", *segons*, "segons")
imprimeix("Recorregut:")
for i *in* $range(0, len(recorregut))$ **do**
    imprimeix($recorregut[i]$)
end

El funcionament seria el següent:

Suposem que estem a Llucmajor i volem anar a la terminal T1 de l'aeroport. Li proposem a l'algorisme que volem anar des de l'estació de Llucmajor de la línia 4 (4_Llucmajor) fins l'estació Aeroport T1 de la línia 9 Sud (9S_Aeroport T1).

Exemple d'entrada 1

```
metro(metro_barcelona, "4_Llucmajor", "9S_Aeroport T1")
```

Exemple de sortida 1

```
Punt inicial: 4_Llucmajor
Punt final: 9S_Aeroport T1
Temps net del recorregut: 3565
```

```
Temps total del recorregut: 59 minuts i 25 segons
Recorregut: [ 4_Llucmajor, 4_Maragall, 4_Guinardó Hospital de Sant Pau,
 ↪  4_Alfons X, 4_Joanic, 4_Verdaguer, 5_Verdaguer, 5_Diagonal,
 ↪  5_Hospital Clínic, 5_Entença, 5_Sants Estació, 5_Plaça de Sants,
 ↪  5_Badal, 5_Collblanc, 9S_Collblanc, 9S_Torrassa, 9S_Can Tries Gornal,
 ↪  9S_Europa Fira, 9S_Fira, 9S_Parc Logístic, 9S_Mercabarna, 9S_Les
 ↪  Moreres, 9S_El Prat Estació, 9S_Cèntric, 9S_Parc Nou, 9S_Mas Blau,
 ↪  9S_Aeroport T2, 9S_Aeroport T1 ]
```

En aquest exemple de sortida s'indica la durada total del trajecte, inclosos els transbordament, que serà de 59 minuts i 25 segons aproximadament. S'indiquen totes les estacions per on passarà i, a més, podem veure que aquest passatger haurà de fer dos transbordaments: el primer serà a l'estació de Verdaguer, on haurà de canviar de la línia 4 a la línia 5; i el segon serà a l'estació de Collblanc, on haurà de canviar de la línia 5 a la línia 9 Sud.

Descrivim un altre trejecte: Imaginem que un estudiant es troba a la zona universitària i vol desplaçar-se fins la Sagrada Família. Li proposem a l'algorisme que volem anar des de l'estació de Zona Univeritària de la línia 3 (3_Zona Universitària) fins l'estació de Sagrada Família de la línia 5 (5_Sagrada Família).

Exemple d'entrada 2

```
metro(metro_barcelona, "3_Zona Universitària", "5_Sagrada Família")
```

Exemple de sortida 2

```
Punt inicial: 3_Zona Universitària
Punt final: 5_Sagrada Família
Temps net del recorregut: 1170
Temps total del recorregut: 19 minuts i 30 segons
Recorregut: [ 3_Zona Universitària, 3_Palau Reial, 3_Maria Cristina,
 ↪  3_Les Corts, 3_Plaça del Centre, 3_Sants Estació, 5_Sants Estació,
 ↪  5_Entença, 5_Hospital Clínic, 5_Diagonal, 5_Verdaguer, 5_Sagrada
 ↪  Família ]
```

En aquest exemple de sortida s'indica la durada total del trajecte que serà de 19 minuts i 30 segons aproximadament. S'indiquen totes les estacions per on passarà i, a més, podem veure que aquest passatger haurà de fer un transbordament: aquest serà a l'estació de Sants Estació, on haurà de canviar de la línia 3 a la línia 5.

En l'Annex B.11 s'adjunta la construcció de l'algoritme necessari per realitzar tots aquests càlculs.

Conclusions

Durant aquest treball hem fet un camí juntament amb la teoria de grafs: l'hem vist néixer del pensament de Leonard Euler; l'hem vist créixer acompanyada d'alguns dels més grans matemàtics de l'història; n'hem vist la seva manera d'entendre i modelitzar el món; l'hem acompanyada fins a la seva maduresa on, juntament amb més matemàtics i teòrics de les ciències de computació, ha col·laborat amb una infinitat d'àrees que en un principi semblaven distants a ella; i finalment ens ha ajudat a resoldre problemes quotidians.

Un cop acabat el treball, puc afirmar que he assolit els objectius proposats. D'una banda, m'he endinsat en el coneixement de la teoria de grafs començant per la seva història, que m'ha permès conèixer l'evolució. He definit diferents tipus de grafs i les seves propietats. Ha calgut incloure demostracions i me n'he adonat que són una part molt important de les matemàtiques. He demostrat, a tall d'exemple, la relació que s'estableix entre el nombre de nodes i d'arestes d'un graf lineal respecte el graf original, o bé l'expressió matemàtica que defineix el nombre d'arestes d'un graf xarxa. En aquest punt, com en tants d'altres, ha estat important compartir coneixements amb persones enteses en la matèria, la qual cosa m'ha permès d'anar avançant amb confiança.

D'altra banda, també he aconseguit estudiar amb certa profunditat i formalitat alguns dels algorismes més usats en aquest camp. N'he estudiat el temps d'execució, les propietats i les possibles aplicacions pràctiques i, de tots ells, n'he fet una implementació en Python, recollida a l'annex B. A més, també he implementat altres algorismes que no es tracten durant el treball, però que estàn estretament relacionats amb conceptes d'aquest, com els algorismes per trobar camins Eulerians i Hamiltonians en un graf. Podríem dir que he ampliat es objectius incials al generar nous algorismes i aplicar-los a problemes reals. El capítol 4 és la concreció de tots aquests objectius mitjançant una cas pràctic: a partir del coneixement del graf es planteja una situació que cal resoldre. Es crea un algorisme, s'aporten unes dades a partir d'un treball de camp i, finalment, s'obté un resultat que ens aporta la informació desitjada.

Amb els objectius acomplerts, la hipòtesi queda verificada. La teoria de grafs ens proporciona eines per modelitzar estructures i processos i ens permet crear aplicacions pràctiques mitjançant procediments algorísmics. Hem vist que podem modelitzar una gran quantitat d'estructures: des de xarxes de telecomunicacions, fins a representar com a graf les dependències de cadascun dels apartats d'un treball, passant per altres situacions com per exemple modelitzar la xarxa de metro. Un cop els models estan definits, ens trobem amb tot un assortiment d'algorismes informàtics que permeten extreure dades de models teòrics i transformar-los en informació útil, resultats. Mitjançant els algorismes hem aconseguit solucionar el problema de la coloració de grafs, ordenar aquest treball de manera òptima i poder conèixer el recorregut i temps de viatge entre dues estacions qualssevol del metro de Barcelona, entre d'altres.

Com a conclusions, a més d'assolir els objectius i verificar la hipòtesi inicial, m'agradaria afegir que:

- Els procediments algorísmics tenen un component matemàtic molt important. Alguns es basen en conceptes matemàtics més generals, com per exemple l'algorisme de Dijkstra amb la desigualtat triangular; en canvi d'altres, com el de Prim, es basen en propietats singulars dels grafs.
- Els grafs, no només formen part de la realitat que ens envolta si no que, a més, ens faciliten moltes gestions i ens proporcionen comoditats de les quals ens seria difícil prescindir: les xarxes socials, Internet, el subministrament elèctric, navegadors GPS...
- Implementar algorismes, a més de la part matemàtica, en ocasions requereix un treball de camp i/o una recollida de dades, com ha estat el cas de l'algorisme de la Xarxa de metro de Barcelona: per tal de treballar amb valors precisos ha estat necessari mesurar els temps reals de viatge entre les estacions de totes les línies.

Ara cal concloure el treball, però no el dono per acabat. Queda camí per fer: la teoria va més enllà d'on jo he arribat; hi ha algorismes que no he tractat com el de Johnson o el de A*, per exemple; i sobretot m'agradaria endinsar-me en la topologia, una de les branques més curioses de les matemàtiques.

Ara per ara, el pas més immediat seria acabar de programar la pàgina web `https://aniolgarcia.github.io/grafs/`, que permetrà consultar el treball, el fitxer font en LaTeX del treball i tot el codi dels programes realitzats.

Personalment, el treball m'ha ajudat molt. He après moltíssim de teoria de grafs, però el que és també molt important és el rerefons més abstracte: el tipus de raonament matemàtic, la formalització de conceptes i idees, les tècniques de demos-

tració, la metodologia matemàtica... Amb tot això he pogut descobrir la capacitat modelitzadora de les matemàtiques. Amb aquest treball també m'he iniciat a la programació en Phyton. En un principi tenia pensat escriure els algorismes en C++, llenguatge que he utilitzat molt més, però vaig decidir intentar fer-ho amb un llenguatge que aportés una visió diferent de la programació a la que tenia dels basats en C. Mitjançant la programació he viscut l'aventura de relacionar i aplicar idees matemàtiques a algorismes i posteriorment "dotar-los de vida" implementant-los en programes. També cal afegir-hi l'existència d'un component més emocional. És el camp d'un dels primers "grans" problemes a els quals em vaig enfrontar, i m'ho he passat molt bé aprenent més sobre aquesta branca. Tots aquests coneixements adquirits em fan veure el món d'una manera lleugerament diferent, i de ben segur que em seràn útils pels meus estudis futurs en el camp de les matemàtiques i la informàtica.

Encara que el treball final no contingui tots els aspectes que m'hagués agradat tractar, el camí que n'ha resultat és, fins i tot, més ampli del que en un principi podia imaginar.

Això sí: tot i que el camí sempre ha fet pujada, cal pensar que cada vegada som més aprop del cim.

Referències

[1] K. APPEL i W. HANKEN. "Every Planar Graph is Four-Colourable (I)". A: *Illinois Journ. Maths.* 21 (1977), pàg. 429 - 490.

[2] K. APPEL i W. HANKEN. "Every Planar Graph is Four-Colourable (II)". A: *Illinois Journ. Maths.* 21 (1977), pàg. 491 - 567.

[3] K. APPEL i W. HANKEN. "The Solution of the Four-Color Map Problem". A: *Scientific American* 27 (1977), pàg. 108 - 121.

[4] A. CAYLEY. "A theorem on trees". A: *Quart. Journ. Maths.* 23 (1889), pàg. 376 - 378.

[5] E. W. DIJKSTRA. "A note on two problems in connection with graphs". A: *Numerische Mathematik* 1 (1959), pàg. 269 - 271.

[6] L. EULER. "Solutio problematis ad geometriam situs pertinentis". A: *Commentarii academiae scientiarum Petropolitanae.* Vol. 8. 1741, pàg. 128 - 140.

[7] F. HARARY. *Graph Theory.* Addison-Wesley, 1969.

[8] A. B. KEMPE. "On the geographical problem of the four colours". A: *Amer. Journ. Maths.* 23 (1879), pàg. 193 - 200.

[9] D. KÖNIG. *Theorie der endlichen und unendlichen Graphen.* Akademische Verlagsgesellschaft, 1936.

[10] J. B. KRUSKAL. "On the shortest spanning subtree of a graph and the traveling salesman problem". A: *Proc. Amer. Math. Soc.* 7 (1956), pàg. 48 - 50.

[11] R. C. PRIM. "Shortest connection networks and some generalizations". A: *Bell Syst. Tech. Journ.* 36 (1957), pàg. 1389 - 1401.

[12] Alexandre-Théophile Vandermonde. "Remarques sur les problèmes de situation". A: *Mémoires de l'Académie Royale des Sciences* (1771), pàg. 566 - 574.

Bibliografia

J. BASART. *Grafs: fonaments i algorismes*. Universitat Autònoma de Barcelona, 1994.

E. DEMAINE i S. DEVADAS. *6.006 Introduction to Algorithms*. Massachusetts Institute of Technology: MIT OpenCourseWare. Fall 2011. URL: `https://ocw.mit.edu`.

E. DEMAINE, S. DEVADAS i N. LYNCH. *6.046J Design and Analysis of Algorithms*. Massachusetts Institute of Technology: MIT OpenCourseWare. Spring 2015. URL: `https://ocw.mit.edu`.

Geeks for Geeks. URL: `www.geeksforgeeks.org/fundamentals-of-algorithms`.

J. GIMBERT. "Les matemàtiques de GOOGLE: l'algorisme PageRank". A: *Butlletí de la Societat Catalana de Matemàtiques*. Vol. 26. 1. 2011, pàg. 29 - 55.

Math Overflow. URL: `mathoverflow.net`.

Math Stack Exchange. URL: `math.stackexchange.com`.

A. MATTHES. *Gallery of named graphs*. 2008. URL: `http://altermundus.fr/downloads/documents/NamedGraphs.pdf`.

A. MATTHES. *tkz-berge.sty*. 2008. URL: `http://www.altermundus.fr/pages/downloads/TKZdoc-berge.pdf`.

Pathfinding. URL: `http://www.redblobgames.com/pathfinding/`.

F.J. SORIA. *Apuntes de grafos*. Universitat de Barcelona, 2014.

Texample. URL: `texample.net`.

ERIC W. WEISSTEIN. *Wolfram MathWorld*. URL: `mathworld.wolfram.com`.

Wikipedia. URL: `wikipedia.org`.

Annex A

Programari

Per poder realitzar aquest treball ha estat necessari fer un ús extensiu d'eines informàtiques. Penso que és important posar en valor totes aquestes eines utilitzades que, directa o indirectament han afectat en el desenvolupament del treball. Tot el programari que s'ha utliitzat durant el transcurs del treball és programari lliure, de codi obert i gratuït.

A.1 Sistemes operatius

Encara que la majoria de programes utilitzats tenen versions per a altres sistemes operatius, jo els he utilitzat sobre sistemes GNU/Linux, concretament en distribucions Ubuntu i Debian.

A.2 Processador de text

Per tal de poder redactar el treball ha estat necessari un processador de textos especial. Ha calgut escriure un gran nombre de fórmules i expressions matemàtiques, pseudocodi, programes complets... i en aquest sentit LaTeX ha ajudat molt. LaTeX és un sistema de composició de text amb una "alta qualitat tipogràfica", on el text s'escriu en forma de codi i se li dona format amb ordres específiques incloses en el document. D'aquesta manera s'aconsegueix exactament el que es desitja, essent possible modificar molts dels paràmetres del document. Per exemple: la fórmula

$$e' = \frac{1}{2}\sum_{i=1}^{n} g(v_i)^2 - e$$

s'escriu a LaTeX com a

```
\[ e'=\frac{1}{2} \sum_{i=1}^{n}g(v_{i})^{2}-e \]
```

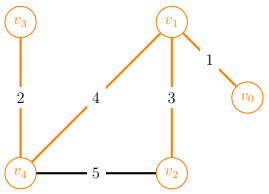
Posteriorment, tot el codi es compila i es genera un fitxer .pdf amb tot el codi interpretat.

Un altre avantatge de LaTeX és que també permet la utilització de diversos paquets que afegeixen funcionalitats addicionals. Durant la redaccó d'aquesta memòria se n'han utilitzat un total de 35, i els següents són els més importants:

A.2.1 Tikz i tkz-berge

Tikz i tkz-berge permeten dibuixar (mitjançant ordres escrites) directament al document, sense que sigui necessari importar imatges externes. La gran majoria de imatges fetes per mi mateix s'han fet a través d'aquests dos paquets. Aquests, al mateix temps, tenen un sistema propi de paquets que permet afegir funcions i ordres. Dins d'aquests paquets, n'hi ha que inclouen diverses funcions orientades a grafs i permeten dibuixar-los tal com es desitgin.

La següent imatge d'un graf



s'escriu a LaTeXcom a

```
\begin{figure}[H]
        \centering
        \begin{tikzpicture}[round/.style={circle, draw=black, thin,
 ↪  minimum size=2mm}, transform shape, scale=0.5]
                \node[round,orange] (3) at (0,4) {$v_{3}$};
                \node[round,orange] (4) at (0,0) {$v_{4}$};
                \node[round,orange] (2) at (4,0) {$v_{2}$};
                \node[round,orange] (1) at (4,4) {$v_{1}$};
                \node[round,orange] (0) at (6,2) {$v_{0}$};


                \Edge[label=$5$](2)(4)
                \tikzset{EdgeStyle/.style={orange}}
                \Edge[label=$1$](0)(1)
                \Edge[label=$3$](1)(2)
                \Edge[label=$4$](1)(4)
```

```
                \Edge[label=$2$](3)(4)

        \end{tikzpicture}
\end{figure}
```

A.2.2 Minted i Algorithm2e

Per poder incloure codi en el treball també ha sigut necessari utilitzar diversos paquets. Minted és el paquet que permet introduïr codi amb el seu format corresponent i ressaltant la sintaxi, i Algorithm2e és l'encarregat de donar format al pseudocodi. Aquests dos paquets permeten que el codi i pseudocodi quedin diferenciats de la resta del text i que, a més, es respecti la forma del programa.

A.3 GeoGebra

GeoGebra és un programa multiús, útil tant en el camp de la geometria, com en l'àlgebra o el càlcul. En aquest cas ha estat utilitzat per a generar les imatges dels punts de Fermat i alguna altra figura concreta com el polígon del problema de les càmeres de seguretat.

A.4 Python

Python és un lenguatge de propòsit general i d'alt nivell que busca senzillesa i eficiència, fent que la sintaxi sigui més planera. Aquest llenguatge permet escriure programes més entenedors i en menys línies que altres llenguatges com C/C++.

A.5 Git

Per tal de mantenir actualitzats els fitxer de redacció de la memòria i el fitxer amb els programes en Python he utilitzat Git. Git és un sistema de control de versions, és a dir, un conjunt d'eines que permeten tenir una còpia remota dels fitxers a la qual se li van afegint els canvis de cada sistema local. Per exemple: des d'un ordinador es realitza un canvi en el fitxer de la memòria del treball. Aquests canvis locals es sincronitzen amb la còpia remota i queda arxivat com a una nova versió. Quan es vol treballar des d'un altre ordinador es comprova si la versió local correspon a la versió remota, és a dir, es mira si hi ha hagut canvis que encara no

té. En cas afirmatiu, es descarreguen els canvis de la còpia remota i s'integren a la còpia local. D'aquesta manera sempre es té la darrera versió i sempre hi ha un registre de tots els canvis que, a més, permet tornar a una versió anterior si fos necessari.

Annex B

Implementació d'algorismes

Per tal de representar els grafs i introduir-los als algorismes s'han utilitat llistes d'adjacència. L'ús de llistes d'adjacència es basa en un dicionari de diccionaris en cas de grafs ponderats, i un dicionari de llistes en cas de grafs no ponderats.

Per exemple: el graf (a) de la figura B.1 té com a llista d'adjacència

```
a={0:[1], 1:[3], 2:[0,3], 3:[]}
```

i la del graf (b) és

```
b={0:{1:4, 2:3}, 1:{2:-2}, 2:{}}
```

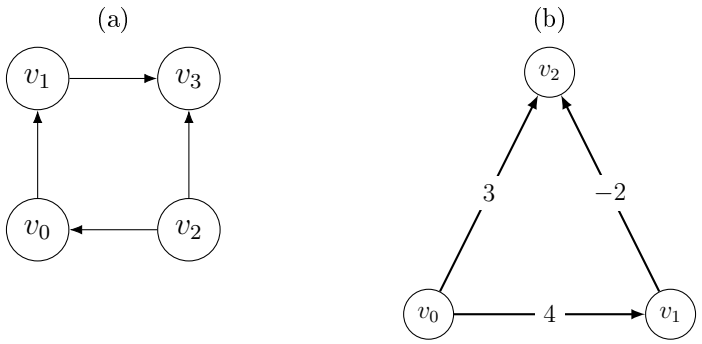


Figura B.1: Exemples de grafs

B.1 DFS

Codi

```
parent={}
topo=[]
def DFS(Adj):
    node=[]
    for i in range(0, len(Adj)):
        node.append(i)

    for s in node:
        if s not in parent:
            print "Des del node %d es pot arribar a:" %s
            print s
            parent[s]=None
            DFS_recursive(Adj, s)
    print "Ordre de recurssió (Ordenació topologica per a grafs dirigits
 ↪  acíclics):"
    topo.reverse()
    print topo


def DFS_recursive(Adj, s):
    for v in Adj[s]:
        if v not in parent:
            print v
            parent[v]=s
            DFS_recursive(Adj, v)
    topo.append(s)
```

Exemple d'entrada

```
G={0:[1,2,3], 1:[3,4], 2:[4], 3:[4], 4:[]}
print DFS(G)
```

Exemple de sortida

```
Des de 0 es pot arribar a:
0
1
3
4
2
```

```
Ordre de recurssió (Ordenació topologica per a grafs dirigits acíclics):
[0, 2, 1, 3, 4]
```

B.2 BFS

Codi

```
def BFS(Adj, s):
    level={s:0}
    parent={s:None}
    i=1
    frontier=[s]
    print s
    while frontier:
        next=[]
        for u in frontier:
            for v in Adj[u]:
                if v not in  level:
                    level[v]=i
                    parent[v]=u
                    next.append(v)
                    print v
        frontier=next
        i+=1
    print level
```

Exemple d'entrada

```
G={0:[1,2,3], 1:[3,4], 2:[4], 3:[4], 4:[]}
print BFS(G, 0)
```

Exemple de sortida

```
{0: 0, 1: 1, 2: 1, 3: 1, 4: 2}
```

B.3 Dijkstra

Codi

```
def Dijkstra(Adj, s):
    Q={}
    dist={}
    tree={}
    for i in range(0, len(Adj)):
        Q[i]=float("inf")
        dist[i]=float("inf")
    Q[s]=0
    while Q:
        u = min(Q, key=Q.get)
        dist[u] = Q[u]
        for v in Adj[u]:
            if v in Q:
                if Q[v] > Q[u] + Adj[u][v]:
                    Q[v] = Q[u] + Adj[u][v]
                    tree[v] = u
        Q.pop(u)

    return dist, tree

def OrderedDijkstra(Adj, s, k):
    Q = dict.fromkeys(Adj.keys(), float("inf"))
    dist = dict.fromkeys(Adj.keys(), float("inf"))
    tree = {}
    Q[s] = 0
    while Q:
        u = min(Q, key=Q.get)
        dist[u] = Q[u]
        for v in Adj[u]:
            if v in Q:
                if Q[v] > Q[u] + Adj[u][v]:
                    Q[v] = Q[u] + Adj[u][v] + k
                    tree[v] = u
        Q.pop(u)

    return dist, tree
```

Exemple d'entrada

```
G={0:{1:10,2:3},1:{2:1, 3:2},2:{1:4, 3:8, 4:2},3:{4:7},4:{3:9}}
distancies, arbre_anteriors = Dijkstra(G, 0)
print distancies
print arbre_apuntadors
```

Exemple de sortida

```
{0: 0, 1: 7, 2: 3, 3: 9, 4: 5}
{1: 2, 2: 0, 3: 1, 4: 2}
```

B.4 Bellman-Ford

Codi

```
def BellmanFord(Adj, s):
    dist={}
    tree={}
    for i in range(0, len(Adj)):
        dist[i]=float("inf")
        tree[i]=None
    dist[s]=0

    for i in range(0, len(Adj)-1):
        for u in range(0, len(Adj)):
            for v in Adj[u]:
                if dist[v] > dist[u] + Adj[u][v]:
                    dist[v] = dist[u] + Adj[u][v]
                    tree[v]=u
    for u in range(0, len(Adj)):
        for v in Adj[u]:
            if dist[v] > dist[u] + Adj[u][v]:
                print "Hi ha cicles de pesos negatius"
                break
    return dist, tree
```

Exemple d'entrada

```
G={0:{1:16,2:0}, 1:{2:-32}, 2:{3:8,4:0}, 3:{4:-16}, 4:{5:4,6:0},
 ↪  5:{6:-8}, 6:{7:2,8:0}, 7:{8:-4}, 8:{9:1,10:0}, 9:{10:-2},10:{}}
distancies, arbre_apuntadors = BellmanFord(G, 0)
print distancies
print arbre_apuntadors
```

Exemple de sortida

```
{0: 0, 1: 16, 2: -16, 3: -8, 4: -24, 5: -20, 6: -28, 7: -26, 8: -30, 9:
↪  -29, 10: -31}
{0: None, 1: 0, 2: 1, 3: 2, 4: 3, 5: 4, 6: 5, 7: 6, 8: 7, 9: 8, 10: 9}
```

B.5 Prim

Codi

```
def Prim(Adj):
    Q={}
    tree={}
    for i in range(0,len(Adj)):
        Q[i]=float("inf")
    Q[0]=0
    while Q:
        u = min(Q, key=Q.get)
        for v in Adj[u]:
            if v in Q and Adj[u][v] < Q[v]:
                Q[v] = Adj[u][v]
                tree[v] = u
        Q.pop(u)
    return tree
```

Exemple d'entrada

```
G={1:{2:1, 3:2},0:{1:10,2:3},2:{1:4, 3:8, 4:2},4:{3:9},3:{4:7}}
print Prim(G)
```

Exemple de sortida

```
{1: 2, 2: 0, 3: 1, 4: 2}
```

B.6 Kruskal

Codi

```
from UnionFind import UnionFind #una bona implementació de UnionFind es
 ↪  troba a https://www.ics.uci.edu/~eppstein/PADS/UnionFind.py
def Kruskal(Adj):
    subtree = UnionFind()
    tree = []
    for e, u, v in sorted((Adj[u][v],u,v) for u in Adj for v in Adj[u]):
        for u in Adj:
            for v in Adj[u]:
                if subtree[u] != subtree[v]:
                    tree.append((u,v))
                    subtree.union(u,v)
    return tree
```

Exemple d'entrada

```
G={0:{1:8,3:2},1:{0:8,2:5,3:3,4:1},2:{1:5,3:2,4:4},3:{0:2,1:3,2:2},4:{1:1,2:4}}
print Prim(G)
```

Exemple de sortida

```
[(0, 1), (0, 3), (1, 2), (1, 4)]
```

B.7 Floyd-Warshall

Codi

```
def FloydWarshall(Adj):
    dist=[[float("inf") for x in range(len(Adj))] for y in
 ↪  range(len(Adj))]
    for i in range(0,len(Adj)):
       dist[i][i] = 0
    for v in range(len(Adj)):
        for u in Adj[v]:
            dist[v][u] = Adj[v][u]
    for x in range(len(Adj)):
        for u in range(len(Adj)):
            for v in range(len(Adj)):
                if dist[u][v] > dist[u][x] + dist[x][v]:
                    dist[u][v] = dist[u][x] + dist[x][v]
    return dist
```

Exemple d'entrada

```
G={0:{1:10,2:3},1:{2:1, 3:2},2:{1:4, 3:8, 4:2},3:{4:7},4:{3:9}}
print FloydWarshall(G)
```

Exemple de sortida

```
[[0, 7, 3, 9, 5],
[inf, 0, 1, 2, 3],
[inf, 4, 0, 6, 2],
[inf, inf, inf, 0, 7],
[inf, inf, inf, 9, 0]]
```

B.8 Hamilton

Codi

```
def Hamilton_recursive(Adj, s, e, path):
    path = path + [s]
    if s == e:
        return path
    for n in Adj[s]:
        if n not in path:
            nou_path = Hamilton_recursive(Adj, n, e, path)
            if nou_path:
                return nou_path
    return None

def Hamilton(Adj, s, e):
    path=[]
    return Hamilton_recursive(Adj, s, e, path)
```

Exemple d'entrada

```
G={0:[2,3,5,1],1:[0,2,4,5],2:[0,1,3,4],3:[0,2,4,5],4:[1,2,3,5],5:[0,1,3,4]}
print Hamilton(G, 0, 5)
```

Exemple de sortida

```
[0, 2, 1, 4, 3, 5]
```

B.9 Euler

Codi

```
def Euler(Adj):
    graf = Adj
    senar = [v for v in graf.keys() if len(graf[v])%2 != 0]
    senar.append(graf.keys()[0])
    print senar

    if len(senar)>3:
        return None

    Q = [senar[0]]
    path = []
    while Q:
        v = Q[-1]
        if graf[v]:
            u = graf[v][0]
            Q.append(u)
            del graf[u][graf[u].index(v)]
            del graf[v][0]
        else:
            path.append(Q.pop())

    return path
```

Exemple d'entrada

```
G={0:[1,2,3],1:[0,2,3],2:[0,1,3,4],3:[0,1,2,4],4:[3,2]}
print Euler(G)
```

Exemple de sortida

```
[1, 3, 4, 2, 3, 0, 2, 1, 0]
```

B.10 Coloració

Codi

```
def coloring(Adj):
    graph = sorted(Adj, key=lambda k:len(Adj[k]), reverse=True)
    colors = {}
    usat = False
    actual = 0

    for i in range(0, len(Adj)):
        colors[i]=None
    colors[graph[0]]=0

    while None in colors.values():
        for v in graph:
            if colors[v] == None:
                for k in Adj[v]:
                    if colors[k] == actual:
                        usat = True
                        break

                if usat == False:
                    colors[v] = actual
                usat = False
        actual = actual + 1
    return colors
```

Exemple d'entrada

```
bipartite={0:[4,5,6,7], 1:[4,5,6,7], 2:[4,5,6,7], 3:[4,5,6,7],
 ↪  4:[0,1,2,3], 5:[0,1,2,3], 6:[0,1,2,3], 7:[0,1,2,3]}
print coloring(bipartite)
```

Exemple de sortida

```
{0: 0, 1: 0, 2: 0, 3: 0, 4: 1, 5: 1, 6: 1, 7: 1}
```

B.11 Metro

Codi

```
def metro(Adj, inici, final, k): #on k és el temps de parada acada estació
    recorregut=[]

    print "Punt inicial:", inici.decode("ISO-8859-15")

    print "Punt final:", final.decode("ISO-8859-15")

    dist, tree = OrderedDijkstra(Adj, inici, k)

    i = final
    while tree[i] != inici:
        recorregut.append(tree[i])
        i = tree[i]

    recorregut.append(inici)
    recorregut.reverse()

    total= dist[final]-k

    print "Temps amb estacions del recorregut:", dist[final], "Temps real:", total

    if total < 60:
        print "Temps total del recorregut:",int(total), "segons"
    else:
        minuts = total/60
        segons = (total%60)
        print "Temps total del recorregut:", int(minuts),"minuts i", int(segons), "segons"

    print "Recorregut:",
    print "[",
    for i in range(0,len(recorregut)):
        print recorregut[i].decode("ISO-8859-15")+",",

    print final.decode("ISO-8859-15"),"]"
```

Exemple d'entrada

```
metro_barcelona={"1_Hospital de Bellvitge":{"1_Bellvitge":90},
 ↪  "1_Bellvitge":{"1_Hospital de Bellvitge":90, "1_Av.
 ↪  Carrilet":100},"1_Av. Carrilet":{"1_Bellvitge":100, "1_Rbla. Just
 ↪  Oliveras":65, "8_L'Hospitalet - Av. Carrilet":180}, "1_Rbla. Just
 ↪  Oliveras":{"1_Av. Carrilet":65, "1_Can Serra":60}, "1_Can
 ↪  Serra":{"1_Rbla. Just Oliveras":60, "1_Florida":60},
 ↪  "1_Florida":{"1_Can Serra":60, "1_Torrassa":60},
 ↪  "1_Torrassa":{"1_Florida":60, "1_Santa Eulàlia":85,
 ↪  "9S_Torrassa":240}, "1_Santa Eulàlia":{"1_Torrassa":85,"1_Mercat
 ↪  Nou":75}, "1_Mercat Nou":{"1_Santa Eulàlia":75, "1_Plaça de
 ↪  Sants":60}, "1_Plaça de Sants":{"1_Mercat Nou":60,
 ↪  "1_Hostafrancs":50, "5_Plaça de Sants":282},
 ↪  "1_Hostafrancs":{"1_Plaça de Sants:":50, "1_Espanya":55},
 ↪  "1_Espanya":{"1_Hostafrancs":55, "1_Rocafort":60, "3_Espanya":209,
 ↪  "8_Espanya":120}, "1_Rocafort":{"1_Espanya":60, "1_Urgell":55},
 ↪  "1_Urgell":{"1_Rocafort":55, "1_Universitat":58},
 ↪  "1_Universitat":{"1_Urgell":58, "1_Catalunya":50,
 ↪  "2_Universitat":144}, "1_Catalunya":{"1_Universitat":50,
 ↪  "1_Urquinaona":58, "3_Catalunya":180, "6_Catalunya":360,
 ↪  "7_Catalunya":360}, "1_Urquinaona":{"1_Catalunya":58, "1_Arc de
 ↪  Triomf":85, "4_Urquinaona":256}, "1_Arc de
 ↪  Triomf":{"1_Urquinaona":85, "1_Marina":54}, "1_Marina":{"1_Arc de
 ↪  Triomf":54, "1_Glòries":80}, "1_Glòries":{"1_Marina":80,
 ↪  "1_Clot":78}, "1_Clot":{"1_Glòries":78,"1_Navas":65, "2_Clot":120},
 ↪  "1_Navas":{"1_Clot":65, "1_La Sagrera":80}, "1_La
 ↪  Sagrera":{"1_Navas":80, "1_Fabra i Puig":89, "5_La Sagrera":100,
 ↪  "9N_La Sagrera":178, "10_La Sagrera":178}, "1_Fabra i Puig":{"1_La
 ↪  Sagrera":89, "1_Sant Andreu":101}, "1_Sant Andreu":{"1_Fabra i
 ↪  Puig":101, "1_Torras i Bages":88}, "1_Torras i Bages":{"1_Sant
 ↪  Andreu":88, "1_Trinitat Vella":77}, "1_Trinitat Vella":{"1_Torras i
 ↪  Bages":77, "1_Baró de Viver":60}, "1_Baró de Viver":{"1_Trinitat
 ↪  Vella":60, "1_Santa Coloma":83}, "1_Santa Coloma":{"1_Baró de
 ↪  Viver":83, "1_Fondo":84}, "1_Fondo":{"1_Santa Coloma":84,
 ↪  "9N_Fondo":140}, "2_Paral·lel":{"2_Sant Antoni":80, "3_Paral·lel":83},
 ↪  "2_Sant Antoni":{"2_Paral·lel":80, "2_Universitat":66},
 ↪  "2_Universitat":{"2_Sant Antoni":66, "2_Passeig de Gràcia":80,
 ↪  "1_Universitat":144}, "2_Passeig de Gràcia":{"2_Universitat":80,
 ↪  "2_Tetuan":95, "3_Passeig de Gràcia":360, "4_Passeig de Gràcia":120},
 ↪  "2_Tetuan":{"2_Passeig de Gràcia":95, "2_Monumental":93},
 ↪  "2_Monumental":{"2_Tetuan":93, "2_Sagrada Família":63}, "2_Sagrada
 ↪  Família":{"2_Monumental":63, "2_Encants":114, "5_Sagrada
 ↪  Família":178}, "2_Encants":{"2_Sagrada Família":114, "2_Clot":53},
 ↪  "2_Clot":{"2_Encants":53, "2_Bac de Roda":89, "1_Clot":120}, "2_Bac
 ↪  de Roda":{"2_Clot":89, "2_Sant Martí":61}, "2_Sant Martí":{"2_Bac de
 ↪  Roda":61, "2_La Pau":74}, "2_La Pau":{"2_Sant Martí":74,
 ↪  "2_Verneda":76, "4_La Pau":60}, "2_Verneda":{"2_La Pau":76,
 ↪  "2_Artigues Sant Adrià":81}, "2_Artigues Sant Adrià":{"2_Verneda":81,
 ↪  "2_Sant Roc":91}, "2_Sant Roc":{"2_Artigues Sant Adrià":91,
 ↪  "2_Gorg":60},
```

```
"2_Gorg":{"2_Sant Roc":60, "2_Pep Ventura":58, "10_Gorg":78}, "2_Pep
↪  Ventura":{"2_Gorg":58, "2_Badalona Pompeu Fabra":74}, "2_Badalona
↪  Pompeu Fabra":{"2_Pep Ventura":74}, "3_Zona Universitària":{"3_Palau
↪  Reial":67, "9S_Zona Universitària":265}, "3_Palau Reial":{"3_Zona
↪  Universitària":67, "3_Maria Cristina":65}, "3_Maria
↪  Cristina":{"3_Palau Reial":65, "3_Les Corts":68}, "3_Les
↪  Corts":{"3_Maria Cristina":68, "3_Plaça del Centre":63}, "3_Plaça del
↪  Centre":{"3_Les Corts":63, "3_Sants Estació":57},"3_Sants
↪  Estació":{"3_Plaça del Centre":57, "3_Tarragona":55, "5_Sants
↪  Estació":232},"3_Tarragona":{"3_Sants Estació":55, "3_Espanya":65},
↪  "3_Espanya":{"3_Tarragona":65, "3_Poble Sec":67, "1_Espanya":209,
↪  "8_Espanya":360}, "3_Poble Sec":{"3_Espanya":67, "3_Paral·lel":70},
↪  "3_Paral·lel":{"3_Poble Sec":70, "3_Drassanes":72, "2_Paral·lel":83},
↪  "3_Drassanes":{"3_Paral·lel":72, "3_Liceu":75},
↪  "3_Liceu":{"3_Drassanes":75, "3_Catalunya":60},
↪  "3_Catalunya":{"3_Liceu":60, "3_Passeig de Gràcia":70,
↪  "1_Catalunya":180, "6_Catalunya":180, "7_Catalunya":180}, "3_Passeig
↪  de Gràcia":{"3_Catalunya":70, "3_Diagonal":77, "2_Passig de
↪  Gràcia":360, "4_Passeig de Gràcia":238}, "3_Diagonal":{"3_Passeig de
↪  Gràcia":77, "3_Fontana":75, "5_Diagonal":217, "6_Provença":360,
↪  "7_Provença":360}, "3_Fontana":{"3_Diagonal":75, "3_Lesseps":55},
↪  "3_Lesseps":{"3_Fontana":55, "3_Vallcarca":85},
↪  "3_Vallcarca":{"3_Lesseps":85, "3_Penitents":90},
↪  "3_Penitents":{"3_Vallcarca":90, "3_Vall d'Hebron":80}, "3_Vall
↪  d'Hebron":{"3_Penitents":80, "3_Montbau":62, "5_Vall d'Hebron":204},
↪  "3_Montbau":{"3_Vall d'Hebron":62, "3_Mundet":64},
↪  "3_Mundet":{"3_Montbau":64, "3_Valldaura":78},
↪  "3_Valldaura":{"3_Mundet":78, "3_Canyelles":79},
↪  "3_Canyelles":{"3_Valldaura":79, "3_Roquetes":94},
↪  "3_Roquetes":{"3_Canyelles":94, "3_Trinitat Nova":84}, "3_Trinitat
↪  Nova":{"3_Roquetes":84, "4_Trinitat Nova":210, "11_Trinitat
↪  Nova":210}, "4_Trinitat Nova":{"4_Via Júlia":99, "3_Trinitat
↪  Nova":210, "11_Trinitat Nova":0}, "4_Via Júlia":{"4_Trinitat
↪  Nova":99, "4_Llucmajor":87}, "4_Llucmajor":{"4_Via Júlia":87,
↪  "4_Maragall":161}, "4_Maragall":{"4_Llucmajor":161, "4_Guinardó
↪  Hospital de Sant Pau":88, "5_Maragall":191}, "4_Guinardó Hospital de
↪  Sant Pau":{"4_Maragall":88, "4_Alfons X":86}, "4_Alfons
↪  X":{"4_Guinardó Hospital de Sant Pau":86, "4_Joanic":77},
↪  "4_Joanic":{"4_Alfons X":77, "4_Verdaguer":89},
↪  "4_Verdaguer":{"4_Joanic":89, "4_Girona":86, "5_Verdaguer":218},
↪  "4_Girona":{"4_Verdaguer":86, "4_Passeig de Gràcia":83}, "4_Passeig
↪  de Gràcia":{"4_Girona":83, "4_Urquinaona":92, "2_Passeig de
↪  Gràcia":120, "3_Passeig de Gràcia":238}, "4_Urquinaona":{"4_Passeig
↪  de Gràcia":92, "4_Jaume I":60, "1_Urquinaona":256}, "4_Jaume
↪  I":{"4_Urquinaona":60, "4_Barceloneta":78}, "4_Barceloneta":{"4_Jaume
↪  I":78, "4_Ciutadella Vila Olímpica":82}, "4_Ciutadella Vila
↪  Olímpica":{"4_Barceloneta":82, "4_Bogatell":113},
↪  "4_Bogatell":{"4_Ciutadella Vila Olímpica":113, "4_Llacuna":62},
↪  "4_Llacuna":{"4_Bogatell":62, "4_Poblenou":64},
↪  "4_Poblenou":{"4_Llacuna":64, "4_Selva del Mar":66}, "4_Selva del
↪  Mar":{"4_Poblenou":66, "4_El Maresme Fòrum":86}, "4_El Maresme
↪  Fòrum":{"4_Selva del Mar":86, "4_Besòs Mar":65},
```

```
"4_Besòs Mar":{"4_El Maresme Fòrum":65, "4_Besòs":67},
↪  "4_Besòs":{"4_Besòs Mar":67, "4_La Pau":76}, "4_La
↪  Pau":{"4_Besòs":76, "2_La Pau":60}, "5_Cornellà
↪  Centre":{"5_Gavarra":90}, "5_Gavarra":{"5_Cornellà Centre":90,
↪  "5_Sant Ildefons":85}, "5_Sant Ildefons":{"5_Gavarra":85, "5_Can
↪  Boixeres":74}, "5_Can Boixeres":{"5_Sant Ildefons":74, "5_Can
↪  Vidalet":90}, "5_Can Vidalet":{"5_Can Boixeres":90, "5_Pubilla
↪  Cases":83}, "5_Pubilla Cases":{"5_Can Vidalet":83,
↪  "5_Collblanc":105}, "5_Collblanc":{"5_Pubilla Cases":105,
↪  "5_Badal":70, "9S_Collblanc":240}, "5_Badal":{"5_Collblanc":70,
↪  "5_Plaça de Sants":74}, "5_Plaça de Sants":{"5_Badal":74, "5_Sants
↪  Estació":79, "1_Plaça de Sants":282}, "5_Sants Estació":{"5_Plaça de
↪  Sants":79, "5_Entença":73, "3_Sants Estació":232},
↪  "5_Entença":{"5_Sants Estació":73, "5_Hospital Clínic":64},
↪  "5_Hospital Clínic":{"5_Entença":64, "5_Diagonal":78},
↪  "5_Diagonal":{"5_Hospital Clínic":78, "5_Verdaguer":78,
↪  "3_Diagonal":217, "6_Provença":180, "7_Provença":180},
↪  "5_Verdaguer":{"5_Diagonal":78, "5_Sagrada Família":75,
↪  "4_Verdaguer":218}, "5_Sagrada Família":{"5_Verdaguer":75, "5_Sant
↪  Pau Dos de Maig":85, "2_Sagrada Família":178}, "5_Sant Pau Dos de
↪  Maig":{"5_Sagrada Família":85, "5_Camp de l'Arpa":63}, "5_Camp de
↪  l'Arpa":{"5_Sant Pau Dos de Maig":63, "5_La Sagrera":93}, "5_La
↪  Sagrera":{"5_Camp de l'Arpa":93, "5_Congrés":85, "1_La Sagrera":100,
↪  "9N_La Sagrera":233, "10_La Sagrera":233}, "5_Congrés":{"5_La
↪  Sagrera":85, "5_Maragall":60}, "5_Maragall":{"5_Congrés":60,
↪  "5_Virrei Amat":60, "4_Maragall":191}, "5_Virrei
↪  Amat":{"5_Maragall":60, "5_Vilapicina":74}, "5_Vilapicina":{"5_Virrei
↪  Amat":74, "5_Horta":75}, "5_Horta":{"5_Vilapicina":75, "5_El
↪  Carmel":78}, "5_El Carmel":{"5_Horta":78, "5_El Coll La
↪  Teixonera":80}, "5_El Coll La Teixonera":{"5_El Carmel":80, "5_Vall
↪  d'Hebron":81}, "5_Vall d'Hebron":{"5_El Coll La Teixonera":81,
↪  "3_Vall d'Hebron":204}, "6_Catalunya":{"6_Provença":120,
↪  "1_Catalunya":360, "3_Catalunya":180, "7_Catalunya":5},
↪  "6_Provença":{"6_Catalunya":120, "6_Gràcia":120, "3_Diagonal":360,
↪  "5_Diagonal":180, "7_Provença":5}, "6_Gràcia":{"6_Provença":120,
↪  "6_Sant Gervasi":60, "7_Gràcia":5}, "6_Sant Gervasi":{"6_Gràcia":60,
↪  "6_Muntaner":120, "7_Plaça Molina":5}, "6_Muntaner":{"6_Sant
↪  Gervasi":120, "6_La Bonanova":60}, "6_La Bonanova":{"6_Muntaner":60,
↪  "6_Les Tres Torres":120}, "6_Les Tres Torres":{"6_La Bonanova":120,
↪  "6_Sarrià":60}, "6_Sarrià":{"6_Les Tres Torres":60, "6_Reina
↪  Elisenda":120}, "6_Reina
↪  Elisenda":{"6_Sarrià":120},"7_Catalunya":{"7_Provença":120,
↪  "1_Catalunya":360,"3_Catalunya":180,
↪  "6_Catalunya":5},"7_Provença":{"7_Catalunya":120, "7_Gràcia":120,
↪  "3_Diagonal":360, "5_Diagonal":180,
↪  "6_Provença":5},"7_Gràcia":{"7_Provença":120,"7_Plaça
↪  Molina":60,"6_Gràcia":5},"7_Plaça
↪  Molina":{"7_Gràcia":60,"7_Pàdua":120, "6_Sant
↪  Gervasi":5},"7_Pàdua":{"7_Plaça Molina":120,"7_El Putxet":60},"7_El
↪  Putxet":{"7_Pàdua":60, "7_Av. Tibidabo":60}, "7_Av. Tibidabo":{"7_El
↪  Putxet":60}, "8_Espanya":{"8_Magòria La Campana":120,
↪  "1_Espanya":120, "3_Espanya":360}, "8_Magòria La
↪  Campana":{"8_Espanya":120, "8_Ildefons Cerdà":120},
```

```
"8_Ildefons Cerdà":{"8_Magòria La Campana":120, "8_Europa Fira":120},
 ↪  "8_Europa Fira":{"8_Ildefons Cerdà":120, "8_Gornal":120, "9S_Europa
 ↪  Fira":180}, "8_Gornal":{"8_Europa Fira":120, "8_Sant Josep":120},
 ↪  "8_Sant Josep":{"8_Gornal":120, "8_L'Hospitalet - Av. Carrilet":60},
 ↪  "8_L'Hospitalet - Av. Carrilet":{"8_Sant Josep":60, "8_Almeda":180,
 ↪  "1_Av. Carrilet":180}, "8_Almeda":{"8_L'Hospitalet - Av.
 ↪  Carrilet":180, "8_Cornellà - Riera":120}, "8_Cornellà -
 ↪  Riera":{"8_Almeda":120, "8_Sant Boi":180}, "8_Sant Boi":{"8_Cornellà
 ↪  - Riera":180, "8_Molí Nou - Ciutat Cooperativa":120}, "8_Molí Nou -
 ↪  Ciutat Cooperativa":{"8_Sant Boi":120}, "9N_La Sagrera":{"9N_Onze de
 ↪  Setembre":116, "1_La Sagrera":178, "5_La Sagrera":233, "10_La
 ↪  Sagrera":5}, "9N_Onze de Setembre":{"9N_La Sagrera":116, "9N_Bon
 ↪  Pastor":115, "10_Onze de Setembre":5}, "9N_Bon Pastor":{"9N_Onze de
 ↪  Setembre":114, "9N_Can Peixauet":125, "10_Bon Pastor":5}, "9N_Can
 ↪  Peixauet":{"9N_Bon Pastor":125, "9N_Santa Rosa":65}, "9N_Santa
 ↪  Rosa":{"9N_Can Peixauet":65, "9N_Fondo":87}, "9N_Fondo":{"9N_Santa
 ↪  Rosa":87, "9N_Església Major":72, "1_Fondo":140}, "9N_Església
 ↪  Major":{"9N_Fondo":72, "9N_Singuerlín":76},
 ↪  "9N_Singuerlín":{"9N_Església Major":76, "9N_Can Zam":103}, "9N_Can
 ↪  Zam":{"9N_Singuerlín":103}, "9S_Aeroport T1":{"9S_Aeroport T2":240},
 ↪  "9S_Aeroport T2":{"9S_Aeroport T1":240, "9S_Mas Blau":80}, "9S_Mas
 ↪  Blau":{"9S_Aeroport T2":80, "9S_Parc Nou":120}, "9S_Parc
 ↪  Nou":{"9S_Mas Blau":120, "9S_Cèntric":85}, "9S_Cèntric":{"9S_Parc
 ↪  Nou":85, "9S_El Prat Estació":95}, "9S_El Prat
 ↪  Estació":{"9S_Cèntric":95, "9S_Les Moreres":150}, "9S_Les
 ↪  Moreres":{"9S_El Prat Estació":150, "9S_Mercabarna":100},
 ↪  "9S_Mercabarna":{"9S_Les Moreres":100, "9S_Parc Logístic":120},
 ↪  "9S_Parc Logístic":{"9S_Mercabarna":120, "9S_Fira":125},
 ↪  "9S_Fira":{"9S_Parc Logístic":125, "9S_Europa Fira":70}, "9S_Europa
 ↪  Fira":{"9S_Fira":70, "9S_Can Tries Gornal":70, "8_Europa Fira":180},
 ↪  "9S_Can Tries Gornal":{"9S_Europa Fira":70, "9S_Torrassa":75},
 ↪  "9S_Torrassa":{"9S_Can Tries Gornal":75, "9S_Collblanc":110,
 ↪  "1_Torrassa":240}, "9S_Collblanc":{"9S_Torrassa":110, "9S_Zona
 ↪  Universitària":105, "5_Collblanc":240}, "9S_Zona
 ↪  Universitària":{"9S_Collblanc":105, "3_Zona Universitària":265},
 ↪  "10_La Sagrera":{"10_Onze de Setembre":116, "1_La Sagrera":178, "5_La
 ↪  Sagrera":233, "9N_La Sagrera":5}, "10_Onze de Setembre":{"10_La
 ↪  Sagrera":116, "10_Bon Pastor":115, "9N_Onze de Setembre":5}, "10_Bon
 ↪  Pastor":{"10_Onze de Setembre":115, "10_Llefià":133, "9N_Bon
 ↪  Pastor":5}, "10_Llefià":{"10_Bon Pastor":133, "10_La Salut":59},
 ↪  "10_La Salut":{"10_Llefià":59, "10_Gorg":111}, "10_Gorg":{"10_La
 ↪  Salut":111, "2_Gorg":78}, "11_Trinitat Nova":{"11_Casa de
 ↪  l'Aigua":43, "3_Trinitat Nova":210, "4_Trinitat Nova":0}, "11_Casa de
 ↪  l'Aigua":{"11_Trinitat Nova":43, "11_Torre Baró Vallbona":111},
 ↪  "11_Torre Baró Vallbona":{"11_Casa de l'Aigua":111, "11_Ciutat
 ↪  Meridiana":60}, "11_Ciutat Meridiana":{"11_Torre Baró Vallbona":60,
 ↪  "11_Can Cuiàs":45}, "11_Can Cuiàs":{"11_Ciutat Meridiana":45}}

metro(metro_barcelona, "2_Paral·lel", "11_Casa de l'Aigua", 25)
```

Exemple de sortida

```
Punt inicial: 2_Paral·lel
Punt final: 11_Casa de l'Aigua
Temps net del recorregut: 1595
Temps total del recorregut: 26 minuts i 35 segons
Recorregut: [ 2_Paral·lel, 2_Sant Antoni, 2_Universitat, 2_Passeig de
 ↪  Gràcia, 4_Passeig de Gràcia, 4_Girona, 4_Verdaguer, 4_Joanic,
 ↪  4_Alfons X, 4_Guinardó Hospital de Sant Pau, 4_Maragall, 4_Llucmajor,
 ↪  4_Via Júlia, 4_Trinitat Nova, 11_Trinitat Nova, 11_Casa de l'Aigua ]
```