

MATEMÀTICA DISCRETA
Resums teòrics i Algorismes

Enginyeria Industrial
Curs 2003-2004

Departament d'Informàtica i Matemàtica Aplicada

September 1, 2003

Continguts

1	Introducció a la teoria de grafs	4
1.1	Generalitats	4
1.1.1	Grafs dirigits	5
1.1.2	Grafs no dirigits	6
1.1.3	Multigrafs	7
1.1.4	Algunes definicions	7
1.1.5	Teoremes sobre els graus	9
1.2	Emmagatzematge d'un graf en memòria	10
1.2.1	Matriu d'adjacència (o matriu d'incidència vèrtex-vèrtex)	10
1.2.2	Matriu d'incidència (vèrtex-arc o vèrtex-aresta)	11
1.2.3	Llistes d'adjacència	11
1.2.4	Llistes d'arestes	13
1.2.5	Operacions amb les matrius d'adjacència	13
1.3	Connexió i components	14
1.3.1	Connexió en el cas de graf no dirigit	14
1.3.2	Connexió en el cas de graf dirigit	15
1.4	Recorregut d'un graf	15
1.4.1	Depth First Search (Backtracking)	15
1.4.2	Breadth First Search	24
2	Camins mínims	26
2.1	Algorisme de Dijkstra	26
2.1.1	Complexitat de l'algorisme de Dijkstra	29
2.2	Algorisme de Ford	30
2.2.1	Complexitat de l'algorisme de Ford	33
2.3	Algorisme de Floyd	33
2.3.1	Complexitat de l'algorisme de Floyd	35

3	Arbres	36
3.1	Conceptes generals	36
3.2	Arbres generats d'una graf	37
3.2.1	Recerca d'arbres generadors	38
3.3	Arbres generats de cost mínim d'un graf	39
3.3.1	Algorisme de Kruskal	39
3.3.2	Algorisme de Prim	41
4	Xarxes de transport	44
4.1	Algorisme de Ford-Fulkerson	44
4.2	Algorisme de Busacker i Gowen	47
5	Grafs Eulerians i Hamiltonians	50
5.1	Algorisme de Fleury	50
5.2	Algorisme d'Edmonds	51
5.3	Algorisme del carter xinès per a grafs dirigits	52
5.4	Obtenció de circuits Hamiltonians pel mètode de les multiplicacions llatines . . .	53
5.5	Algorisme de Robert i Flores	54

Capítol 1

Introducció a la teoria de grafs

1.1 Generalitats

Els grafs fan possible representar l'estructura d'un gran nombre de situacions d'una manera senzilla, especialment aquelles en les quals es té una col·lecció d'elements amb relacions binàries (és a dir, algunes parelles d'elements tenen un lligam especial).

Un exemple clàssic és el de descriure una xarxa de comunicacions: ciutats unides per carreteres (naturalment, una carretera relaciona dues ciutats, i algunes parelles de ciutats potser no estan relacionades directament), línies de ferrocarril que enllacen estacions, enllaços telefònics, xarxes elèctriques, flux d'informació dins d'una organització, etc.

Els problemes que volem resoldre són de diversos tipus. Una primera categoria és els problemes de *connexió*, on ens preguntem per exemple quantes maneres diferents hi ha per anar d'una ciutat a una altra per una xarxa de carreteres, o coses una mica més difícils com ara quin és el nombre mínim de carreteres que cal tallar (escollint-les adequadament) per deixar desconnectades dues ciutats, o com s'ha de dissenyar una xarxa elèctrica redundat per tal que encara que fallin dos circuits, la connexió elèctrica no quedi interrompuda.

Una segona categoria és la de problemes d'*optimització*, on es tracta de fer mínim un cost o un consum o bé de fer màxim un guany. Exemples típics són el problema de l'itinerari de longitud total mínima que pot fer un viatjant que ha de visitar necessàriament unes ciutats donades, o bé la manera com s'ha de distribuir el tràfic ferroviari per tal que, amb la xarxa existent, es pugui transportar el màxim nombre de passatgers d'una ciutat a una altra.

També hi ha els problemes de *planaritat*: una xarxa de connexions, es pot representar en el pla sense que hi hagi cap creuament entre connexions? El problema té evidentment una aplicació al disseny de circuits integrats, i està resolt des de fa temps. De fet, generalment els circuits

són prou complicats com perquè no siguin mai planars, de manera que gairebé sempre cal estructurar-los per capes. Aleshores la pregunta és, naturalment, quin és el nombre mínim de capes amb les que podríem dissenyar el circuit. Aquesta qüestió també està resolta, però no és en absolut elemental, i no es veurà en aquest curs.

Des del punt de vista estrictament matemàtic, els grafs presenten problemes diferents dels d'altres branques de les matemàtiques. En general, no hi ha problemes d'existència, perquè en definitiva, només fem que treballar amb conjunts finits. En altres branques de les matemàtiques, els problemes d'existència solen ser difícils, mentre que en teoria de grafs sovint el problema de la simple existència de solució és trivial. Per exemple, en el problema esmentat més amunt del viatjant de comerç l'existència de solució és evident: es prenen tots els itineraris possibles, es calcula la seva longitud i, com que només n'hi ha un nombre finit, s'agafa el de longitud més petita. El que ja no és tan evident és com trobar-la *explícitament*, perquè el nombre de combinacions és tan gran que mai no podrem construir cap computador (amb la filosofia digital) capaç de representar-les totes, amens que es trobi una manera molt astuta de fer-ho: el que anomenem un *algorisme eficient*. El camp d'estudi de la teoria de grafs (o en general, de la matemàtica discreta) és la recerca d'algorismes eficients i també (problema generalment difícil) aclarir si hi ha una cota inferior per a la complexitat d'aquests algorismes, dit en altres paraules, si és que hi ha problemes *essencialment complexos*, en el sentit que no existeix una *drecera matemàtica* que ens estalviï una llarguíssima enumeració de casos.

La teoria de grafs, tot i ser una branca relativament nova de les matemàtiques, ha estat molt estudiada tant des del punt de vista de les qüestions teòriques com del de les aplicacions pràctiques i la quantitat de literatura existent és molt gran. En aquesta assignatura ens limitarem a fer una introducció i veure algunes de les aplicacions més habituals de la teoria de grafs.

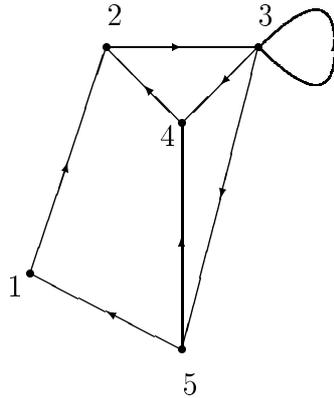
1.1.1 Grafs dirigits

Un graf dirigit $G(V, A)$ consta d'un conjunt V d'elements anomenats *vèrtexs* i un conjunt A de connexions entre aquests vèrtexs, anomenades *arcs*. Cada arc es pot pensar com una parella ordenada de vèrtexs: el vèrtex inicial i el final. Naturalment, entre dos vèrtexs a i b només pot haver-hi, com a molt, dos arcs: l'arc (a, b) i el (b, a) . També es permet l'arc (a, a) , anomenat *llaç*.

Gràficament, els vèrtexs es representen per punts del pla i els arcs per fletxes que surten del vèrtex inicial i van a parar al vèrtex final.

Exemple 1. La figura següent mostra el graf $G(V, A)$ definit per

$$\begin{aligned} V &= \{1, 2, 3, 4, 5\} \\ A &= \{(1, 2), (5, 1), (5, 4), (4, 2), (3, 4), (3, 5), (2, 3), (3, 3)\} \end{aligned}$$



Direm que el vèrtex b és *successor* de a si hi ha un arc amb origen a i final b . El conjunt de successors d'un vèrtex a es denota per Γ_a o $\Gamma(a)$. El conjunt dels successors de tots els successors de a es denota Γ_a^2 o $\Gamma^2(a)$, i en general es pot parlar del conjunt dels successors d'ordre n , denotat Γ_a^n .

Direm que el vèrtex c és *antecessor* o *predecessor* de a si hi ha un arc amb origen c i final a . El conjunt d'antecessors d'un vèrtex a es denota per Γ_a^{-1} o $\Gamma^{-1}(a)$. El conjunt dels antecessors de tots els antecessors de a es denota Γ_a^{-2} o $\Gamma^{-2}(a)$, i en general es pot parlar del conjunt dels antecessors d'ordre n , denotat Γ_a^{-n} o $\Gamma^{-n}(a)$.

Exemple 2. En el graf de l'exemple 1 tenim

$$\begin{aligned} \Gamma(5) &= \{1, 4\}, & \Gamma^2(5) &= \{2\}, \\ \Gamma^{-1}(4) &= \{3, 5\}, & \Gamma^{-2}(5) &= \{3, 2\}. \end{aligned}$$

1.1.2 Grafs no dirigits

En l'estudi de certes propietats dels grafs no és important el sentit en què es recorre cada arc. Això porta a la definició de graf *no dirigit*: un conjunt V de vèrtexs amb un conjunt A de parelles de vèrtexs, anomenades ara *arestes* que s'interpreten com a connexions no dirigides entre els vèrtexs dels seus extrems. En aquest sentit l'aresta (a, b) és la mateixa que la (b, a) i es representa gràficament com una línia que uneix els vèrtexs a i b , sense cap fletxa. També es permeten llaços (a, a) que uneixen un vèrtex amb ell mateix.

Els conceptes d'antecessor i de successor són reemplaçats en els grafs no dirigits pel concepte d'*adjacent*. Direm que el vèrtex b és adjacent al vèrtex a si existeix una aresta que els connecta. Denotarem per Γ_a o $\Gamma(a)$ el conjunt de tots els vèrtexs adjacents al vèrtex a . Anàlogament a com es feia en els grafs dirigits es defineixen els adjacents de segon ordre (conjunts dels adjacents a tots els adjacents), denotat Γ_a^2 o $\Gamma^2(a)$, i els adjacents d'ordre n .

Moltes vegades els grafs no dirigits s'anomenen també *simètrics*, tot i que alguns autors reserven la paraula simètric per a d'altres usos. En aquestes notes, *simètric* es considerarà sinònim de *no dirigit*.

1.1.3 Multigrafs

D'acord amb les definicions anteriors, en un graf no dirigit només pot haver-hi, com a molt, una aresta entre dos vèrtexs, mentre que si és dirigit pot haver-hi com a molt dos arcs, un en cada sentit.

Si es permet que hi hagi més d'una aresta o més d'un arc en cada sentit s'obté el que s'anomena un *multigraf*. Això implica modificar la definició d'arestes o arcs perquè cal permetre repetir parelles de vèrtexs tipus (a, b) . Es pot arranjar associant a cada parella un nombre enter positiu que representi amb quina multiplicitat (i.e. quantes vegades repetida) considerem aquella aresta o arc.

Donat un graf dirigit, es pot considerar el graf que resulta en ignorar els sentits dels arcs i s'obté el que s'anomena *graf no dirigit subjacent* associat al graf original (que tècnicament és un multigraf, a menys que es vulgui definir com a graf no múltiple i ajuntar els dos arcs en una sola aresta quan sigui el cas). Naturalment prendre una definició o l'altra, com sempre en Matemàtiques, depèn de quines propietats es volen estudiar.

1.1.4 Algunes definicions

Graf simple: Un graf que no té llaços, tant si és dirigit com si no, s'anomena *simple* o *sense llaços*.

Camí (graf dirigit): successió d'arcs u_1, \dots, u_q tals que el vèrtex final de cada un d'ells coincideix amb el vèrtex inicial del següent.

Camí (graf no dirigit): successió d'arestes u_1, \dots, u_q tals que cada una d'elles (excepte la primera u_1 i la darrera u_q) té un vèrtex comú amb l'aresta anterior i la posterior. Alguns autors utilitzen *cadena* en el cas no dirigit.

Camí simple: camí que no passa dues vegades pel mateix vèrtex (intuïtivament, que no té bucles o que no recorre diverses vegades amunt i avall per la mateixa aresta abans de seguir). Concepte vàlid tant en el cas dirigit com en el no dirigit.

Circuit: camí en el qual els extrems coincideixen. Alguns autors utilitzen *cicle* en el cas no dirigit.

Circuit simple: camí simple en el qual els extrems coincideixen.

Grau d'un vèrtex: en un graf no dirigit, nombre d'arestes que concorren en aquell vèrtex.

Semigrau interior: en un graf dirigit, nombre d'arcs que entren a aquell vèrtex.

Semigrau exterior: en un graf dirigit, nombre d'arcs que surten d'aquell vèrtex.

Subgraf: direm que G' és un subgraf de G si està format per alguns vèrtexs de G (tot vèrtex de G' ho és de G) i totes les arestes o arcs entre aquests vèrtexs. Intuïtivament, si G és la xarxa de carreteres de Catalunya, G' podria ser la xarxa de carreteres de la província de Girona.

Graf parcial: direm que G' és un graf parcial de G si està format per tots els vèrtexs de G i algunes de les arestes o arcs de G . Intuïtivament, si G és la xarxa de carreteres de Catalunya, G' podria ser la xarxa d'autopistes de Catalunya.

Subgraf parcial: direm que G' és un subgraf parcial de G si està format per alguns vèrtexs de G (tot vèrtex de G' ho és de G) i algunes de les arestes o arcs entre aquests vèrtexs. Intuïtivament, si G és la xarxa de carreteres de Catalunya, G' podria ser la xarxa d'autopistes de la província de Girona.

Graf complet: el que té totes les arestes possibles (sense llaços). El graf complet amb n vèrtexs té $n(n - 1)/2$ arestes. Efectivament, de cada vèrtex surten $n - 1$ arestes dirigides als altres vèrtexs; multiplicat per n vèrtexs és $n(n - 1)$, però cal dividir per 2 perquè òbviament estem comptant les arestes dues vegades. En termes de combinatòria es tracta de les combinacions d'ordre 2 de n elements, $C(n, 2) = \binom{n}{2}$.

Graf complementari: direm que \overline{G} és el graf complementari de G si té els mateixos vèrtexs i exactament les arestes que li manquen a G per ser un graf complet.

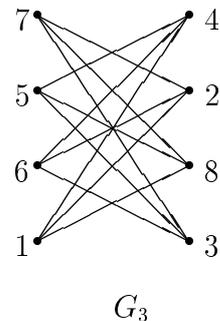
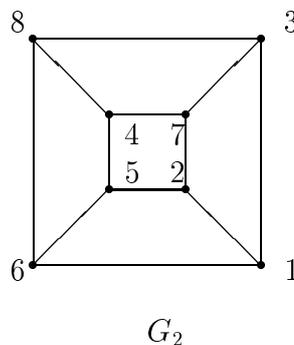
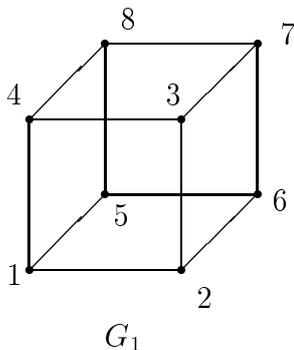
Graf k -regular: direm que un graf G és k -regular si tots els vèrtexs tenen el mateix grau k .

Graf bipartit: direm que un graf G és bipartit si els seus vèrtexs es poden separar en dos subconjunts disjunts V_1 i V_2 , de forma que cada aresta (v_i, v_j) verifica que $v_i \in V_1$ i $v_j \in V_2$.

Grafs isomorfs: direm que dos grafs són isomorfs si són el mateix graf excepte potser un canvi de nom dels vèrtexs.

El problema de l'isomorfisme de grafs

Els grafs següents són isomorfs (encara que potser no ho semblin)



perquè estan donats per

$$V_1 = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

$$A_1 = \{(1, 2), (1, 4), (1, 5), (2, 3), (3, 4), (2, 6), (4, 8), (3, 7), (5, 6), (5, 8), (6, 7), (7, 8)\}$$

$$V_2 = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

$$A_2 = \{(1, 2), (1, 3), (1, 6), (3, 8), (6, 8), (5, 6), (3, 7), (4, 8), (2, 7), (2, 5), (4, 5), (4, 7)\}$$

$$V_3 = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

$$A_3 = \{(1, 2), (1, 4), (1, 8), (2, 6), (4, 6), (3, 6), (4, 5), (3, 5), (5, 8), (2, 7), (3, 7), (7, 8)\}$$

i es pot veure que les de G_1 i G_2 són iguals amb el següent canvi de noms

$$1 \rightarrow 6, \quad 2 \rightarrow 1, \quad 4 \rightarrow 8, \quad 6 \rightarrow 2, \quad 8 \rightarrow 4.$$

De la mateixa manera, G_1 i G_3 són isomorfs si fem el canvi de noms

$$3 \rightarrow 7, \quad 4 \rightarrow 8, \quad 5 \rightarrow 4, \quad 7 \rightarrow 3, \quad 8 \rightarrow 5.$$

Si imaginem que en l'exemple anterior els grafs tenen alguns milers de vèrtexs, queda clar que la representació gràfica no ajuda en res i que un algorisme per a decidir si dos grafs són el mateix no és precisament trivial. Aquí es fa palesa una característica molt pròpia de la teoria de grafs: és evident que existeix una manera de decidir si dos grafs són iguals o no, perquè podrien assajar totes les possibles permutacions de vèrtexs del primer graf (que naturalment són en nombre finit) i provar per a cada una d'elles si s'obté el segon graf. Dissortadament l'esforç computacional que això requereix és enorme i no és factible atacar el problema simplement amb el mètode de *força bruta*. Existeixen algorismes sofisticats per al problema de l'isomorfisme de grafs, però tots ells són de cost computacional molt elevat.

1.1.5 Teoremes sobre els graus

Teorema 1

La suma de tots els graus dels vèrtexs d'un graf no dirigit és igual a dues vegades el nombre d'arestes.

Teorema 2

La suma de tots els semigrans (tant interiors com exteriors) dels vèrtexs d'un graf dirigit és igual a dues vegades el nombre d'arcs.

1.2 Emmagatzematge d'un graf en memòria

Aquesta és la primera qüestió no trivial de la teoria de grafs. És evident que per a grafs amb pocs vèrtexs i/o arestes (o arcs), no hi ha cap problema, però sí que cal pensar-ho bé en un graf amb diversos milers de vèrtexs.

Una primera opció és la simple definició de graf: llista de vèrtexs i llista de parelles de vèrtexs (arestes o arcs). Fins i tot es pot pensar que la llista de vèrtexs no cal, donat que els podem pensar numerats des de 1 fins a n (si és necessari podem mantenir una llista amb els noms dels vèrtexs, però aquesta llista no es farà servir en cap algorisme per a cap propòsit que no sigui purament identificatori); per tant és suficient amb donar una llista d'arestes (parelles de vèrtexs). En aquest cas, la quantitat de memòria necessària per a guardar un graf amb m arestes és $O(m)$ (exactament mc bytes, si c és la quantitat de bytes necessària per a guardar una parella d'enters).

El problema és que amb els grafs volem fer operacions del tipus trobar camins més curts, o més barats, o respondre preguntes similars, per a la qual cosa necessitarem constantment buscar els antecessors o els successors d'un vèrtex donat, cosa que suposa examinar tota la llista d'arestes, i això és un procés relativament costós. És millor guardar el graf d'una manera que el cost computacional sigui el menor possible.

Per tal de descriure un graf, es poden utilitzar un cert nombre de representacions. Des d'un punt de vista pràctic, però, no són equivalents pel que fa a l'eficiència dels diferents algorismes. Hi ha essencialment dues famílies de representacions: les que fan servir *matrius d'incidència* i les que fan servir *matrius d'adjacència*.

1.2.1 Matriu d'adjacència (o matriu d'incidència vèrtex–vèrtex)

Sigui G un graf dirigit o no, possiblement amb llaços, amb vèrtexs V i arestes A , però que no sigui multigraf. Sigui $n = \#V$ (nombre de vèrtexs) i $m = \#A$ (nombre d'arestes). Definim la matriu d'adjacència $\mathcal{A} = (a_{ij})$, amb $1 \leq i, j \leq n$ com la matriu $n \times n$ on cada element a_{ij} està donat per

$$a_{ij} = \begin{cases} 1 & \text{si } (i, j) \text{ és un arc o aresta} \\ 0 & \text{en cas contari} \end{cases}$$

En el cas no dirigit es pot considerar que a cada aresta (i, j) corresponen dos arcs (i, j) i (j, i) , de manera que la matriu és simètrica respecte la diagonal. Si el graf no té llaços tots els elements de la diagonal són zero.

Exemple 3. En el graf de l'exemple 1, la matriu d'adjacència és

$$\mathcal{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Les potències d'ordre k de la matriu d'adjacència tenen una interpretació interessant, perquè donen el nombre total de camins de longitud k entre cada parella de vèrtexs (veure Secció 1.2.5). No donen, però, els propis camins, només quants n'hi ha. Si hom vol els camins, cal buscar-los amb algun algorisme.

1.2.2 Matriu d'incidència (vèrtex–arc o vèrtex–aresta)

Graf dirigit

En el mateix graf de l'apartat anterior, definim la matriu d'incidència $\mathcal{J} = (b_{iu})$, amb $1 \leq i \leq n$ i $1 \leq u \leq m$ com la matriu de m columnes i n files donada per (observeu que dels dos subíndexs el primer numera els vèrtexs i els segon numera els arcs)

$$b_{iu} = \begin{cases} 1 & \text{si l'arc } u \text{ surt del vèrtex } i \\ -1 & \text{si l'arc } u \text{ entra al vèrtex } i \\ 0 & \text{en cas contrari o bé si és llaç} \end{cases}$$

Graf no dirigit

En un graf no dirigit G amb n vèrtexs i m arestes, definim la matriu d'incidència $\mathcal{J} = (b_{iu})$, amb $1 \leq i \leq n$ i $1 \leq u \leq m$ com la matriu de m columnes i n files donada per (observeu que dels dos subíndexs el primer numera els vèrtexs i els segon numera les arestes)

$$b_{iu} = \begin{cases} 1 & \text{si l'aresta } u \text{ té el vèrtex } i \text{ com un dels extrems} \\ 0 & \text{en cas contrari o bé si és llaç} \end{cases}$$

1.2.3 Llistes d'adjacència

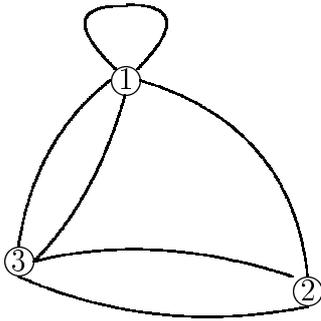
En el cas de grafs poc densos ($m \ll n^2$ en el cas dirigit i $m \ll n(n-1)/2$ en el cas no dirigit) això implica que una gran part de la matriu està ocupada per zeros, amb el consegüent malbaratament de memòria. En aquests casos convé definir només aquells elements de la matriu que són no nuls.

Graf dirigit

Utilitzem dues llistes o vectors $\alpha(\cdot)$ de longitud $n + 1$ i $\beta(\cdot)$ de longitud m . Per a cada vèrtex i la taula β llista els successors de i , començant a l'entrada indicada per $\alpha(i)$. Per tant, la llista dels vèrtexs successors de i ocupa les entrades compreses entre $\alpha(i)$ i $\alpha(i + 1) - 1$ de la taula $\beta(\cdot)$.

Si el graf té pesos als arcs, és a dir, si cada arc u té associat un pes $p(u)$, aleshores aquesta informació es pot guardar en una taula $p(\cdot)$ de longitud m .

Exemple 4.



$\alpha(\cdot)$	1	2	3	4
	1	3	5	7

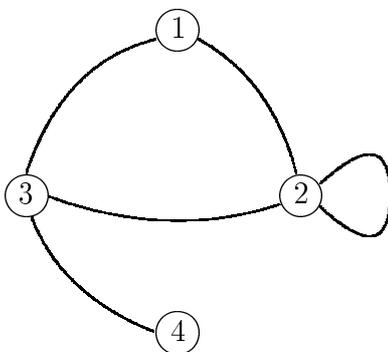
$\beta(\cdot)$	1	2	3	4	5	6	7
	1	3	1	3	2	1	

Observació: cal fer el vector $\alpha(\cdot)$ de longitud $n + 1$ (una unitat més que el nombre de vèrtexs) per tal que l'afirmació “la llista de successors de j ocupa les entrades entre $\alpha(j)$ i $\alpha(j + 1) - 1$ ” sigui certa també en el cas $j = n$.

Graf no dirigit

En el cas no dirigit utilitzem també dues llistes o vectors $\alpha(\cdot)$ de longitud $n + 1$ i $\beta(\cdot)$ de longitud $2m$. Per a cada vèrtex i la taula β llista els adjacents a i , començant a l'entrada indicada per $\alpha(i)$. Per tant, la llista dels vèrtexs adjacents a i ocupa les mateixes entrades que en el cas anterior. Si el graf té pesos a les arestes, es procedeix igual que en el cas dirigit.

Exemple 5.



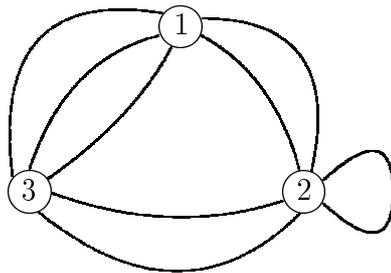
$\alpha(\cdot)$	1	2	3	4	5
	1	3	6	9	10

$\beta(\cdot)$	1	2	3	4	5	6	7	8	9	10
	2	3	1	2	3	1	2	4	3	

1.2.4 Llistes d'arestes

Aquesta manera de representar un graf G amb n vèrtexs i m arestes o arcs consisteix a donar dues llistes $\gamma_1(\cdot)$ i $\gamma_2(\cdot)$, de longitud m , on $\gamma_1(i)$ i $\gamma_2(i)$ són, respectivament, el vèrtex inicial i el final de l'aresta o arc i . Si el graf té pesos, es pot afegir una llista $p(\cdot)$ de longitud m que contingui els pesos de les arestes o arcs.

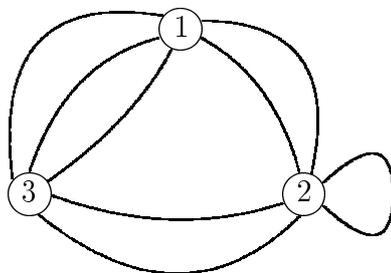
Exemple 6.



$\gamma_1(\cdot)$	1	2	3	4	5	6	7	8
	1	3	1	2	2	3	2	2
$\gamma_2(\cdot)$	3	1	3	1	1	2	3	2
	3	1	3	1	1	2	3	2

Una altra possibilitat és donar una taula $\alpha(\cdot)$ de longitud $n + 1$ i una taula $\beta(\cdot)$ de manera que a $\beta(\alpha(j))$ comença la llista de les arestes incidents al vèrtex j (cas no dirigit)

Exemple 7.



$\alpha(\cdot)$	1	2	3	4					
	1	3	7	9					
$\beta(\cdot)$	1	2	3	4	5	6	7	8	9
	1	3	4	5	7	8	2	6	
$\gamma(\cdot)$	3	3	1	1	3	2	1	2	
	3	3	1	1	3	2	1	2	

En el cas dirigit caldria donar, anàlogament, la llista dels arcs que tenen el vèrtex j com a vèrtex inicial, però cal acompanyar-la amb una llista $\gamma(\cdot)$ de la mateixa longitud que $\beta(\cdot)$, de manera que $\gamma(u)$ conté el vèrtex final de l'aresta u .

1.2.5 Operacions amb les matrius d'adjacència

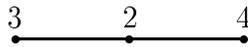
Suposem que tenim un graf $G(V, A)$ amb n vèrtexs, i que fem el producte de la matriu d'adjacència \mathcal{A} per ella mateixa

$$\begin{array}{c}
 \text{columna 2} \qquad \qquad \text{columna 4} \\
 \downarrow \qquad \qquad \qquad \downarrow \\
 \text{fila 3} \rightarrow \left(\begin{array}{c} \vdots \\ \boxed{\dots 1 \dots} \\ \vdots \end{array} \right) \left(\begin{array}{c} \vdots \\ \vdots \\ \vdots \\ \boxed{1} \\ \vdots \end{array} \right) \leftarrow \text{fila 2} \\
 \mathcal{A}^2 = \left(\begin{array}{c} \vdots \\ \vdots \\ \vdots \end{array} \right) \left(\begin{array}{c} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \end{array} \right)
 \end{array}$$

Anomenarem $a_{ij}^{(2)}$ l'element de \mathcal{A}^2 que està a la fila i , columna j . Per fixar idees pensem en un element concret, per exemple $a_{34}^{(2)}$. Tenim, d'acord amb la regla habitual del producte de matrius

$$a_{34}^{(2)} = a_{31}a_{14} + a_{32}a_{24} + a_{33}a_{34} + \dots + a_{3n}a_{n4}.$$

Suposem que el graf conté, entre d'altres, els arcs següents



En aquest cas es té

$$a_{32}a_{24} = 1 \cdot 1 = 1$$

i observem que quan hi ha un camí de longitud 2 entre els vèrtexs 3 i 4 (que passa pel vèrtex 2, en aquest exemple), apareix un terme 1 en l'expressió de $a_{34}^{(2)}$. Podem pensar que la suma que defineix $a_{34}^{(2)}$ explora successivament tots els vèrtexs j del graf per veure si hi ha un camí de longitud 2 del vèrtex 3 al 4, que passi pel vèrtex j . En cas afirmatiu, contribueix amb un 1 al valor numèric de $a_{34}^{(2)}$ i, per tant, el valor numèric d'aquest element de la matriu \mathcal{A}^2 és igual al nombre de camins de longitud 2 per anar de 3 a 4.

1.3 Connexió i components

1.3.1 Connexió en el cas de graf no dirigit

Direm que un graf no dirigit és *connex* si per a cada parella de vèrtexs (diferents) existeix un camí que els uneix. Direm que el graf és *disconnex* en cas contrari.

Intuïtivament, un graf disconnex és en realitat una col·lecció de grafs sense cap relació entre ells. Aquests diferents grafs s'anomenen les *components connexes* del graf original. Naturalment, és important de trobar un algorisme per a determinar si un graf donat és connex i, en cas contrari, determinar les seves components connexes.

1.3.2 Connexió en el cas de graf dirigit

En el cas dirigit el concepte de connectivitat segueix tenint sentit, però cal distingir si dos vèrtexs qualssevol estan connectats en els dos sentits possibles o bé només en un sentit. Direm que un graf dirigit G és *fortament connex* si per a cada parella de vèrtexs (diferents) a i b existeixen dos camins, un que va de a cap a b i l'altre de b cap a a . Direm que un graf dirigit G és *simplement connex* si per a cada parella de vèrtexs (diferents) a i b existeix un camí que va de l'un a l'altre, no importa en quin sentit.

1.4 Recorregut d'un graf

Cal tenir sempre present que un graf associat a un problema real vindrà donat d'alguna de les maneres de la secció 1.2 i no tindrem, ni potser serà factible tenir-lo, un dibuix del graf on totes les seves propietats siguin evidents. Dit d'una altra manera, cal acostumar-se a pensar que un graf és una llista molt i molt llarga de nombres (potser milers), o de parelles de nombres, que representen vèrtexs o arestes, i que en aquestes llarguíssimes llistes de nombres res, absolutament res, no és evident. Això és cert, en particular, per a una pregunta tan simple com la següent: això és un graf o són uns quants grafs sense connexió?

Veurem a continuació un parell d'algorismes que permeten respondre aquesta pregunta.

1.4.1 Depth First Search (Backtracking)

La idea intuïtiva de l'algorisme DFS (recorregut per fondària prioritària) és la següent. Comencem a qualsevol vèrtex del graf i caminem primer tan lluny com sigui possible dintre del graf sense formar un circuit. Quan no podem avançar més, tornem a la darrera bifurcació on havíem deixat un o més camins i prenem una de les arestes que no s'havien fet servir (aquesta tècnica també rep el nom de *backtracking*). Seguim aquest procés fins que tornem al vèrtex del qual havíem sortit. El conjunt de vèrtexs per on hem passat és una component connexa.

Si amb això s'han exhaurit els vèrtexs del graf, aleshores el graf és connex. En cas contrari, hem trobat una component connexa i seguim el procés amb un vèrtex nou.

Aquesta idea, tan senzilla en aparença, presenta alguna dificultat quan s'ha d'implementar sense ajuda visual, només disposant, per exemple, de llistes de vèrtexs adjacents. Cal guardar en memòria diverses coses: en primer lloc el vèrtex des del qual s'ha accedit a un vèrtex determinat, per tal de poder tirar enrera i fer el backtracking; en segon lloc, quines arestes s'han explorat des de cada vèrtex per tal de no tornar a explorar-les. A més, també cal guardar en quin ordre s'exploren els vèrtexs del graf.

Farem un exemple sobre un graf senzill per tal de fixar idees. Considerem el graf G amb vèrtexs i arestes

$$V = \{1, 2, 3, 4, 5, 6\},$$

$$A = \{(1, 3), (1, 2), (1, 4), (2, 3), (2, 4), (2, 5), (4, 5), (3, 5), (4, 6)\}.$$

Fabriquem una taula de la forma següent

						ordre d'exploració
						predecessors
1	2	3	4	5	6	vèrtexs
4	1	1	2	2	4	} adjacents
2	3	2	1	4		
3	4	5	5	3		
5		6				

Anirem modificant la plantilla a mesura que executem l'algorisme, i al costat de cada pas executat posarem la plantilla tal com queda després de l'execució. Encara que no és estrictament necessari, posarem un asterisc per indicar a quin vèrtex estem al final de cada pas.

Suposem que comencem pel vèrtex 3. Primer inicialitzem i a continuació entrem en el bucle principal .

Inicialització

- caselles ordre d'exploració buides
- caselles predecessor buides
- anem a la columna 3 (primer vèrtex)
- posem 1 a ordre d'exploració (3 és el primer vèrtex explorat)
- posem 3 (o qualsevol cosa excepte deixar-la buida) a la casella predecessor

		1			
		3			
1	2	3*	4	5	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	5		6		

Bucle principal

- Estem en el vèrtex 3.

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 3 és 1. Anem a la columna 1.

El vèrtex 1 té predecessor? NO, per tant fem:

- * posar 2 a ordre d'exploració (1 és el 2n vèrtex de l'exploració)
- * posar 3 a predecessor (venim de 3)
- * eliminar 1 de la columna d'adjacents de 3 (ja ha estat explorat)

2		1			
3		3			
1*	2	3	4	5	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	5		6		

- Estem en el vèrtex 1.

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 1 és 4. Anem a la columna 4.

El vèrtex 4 té predecessor? NO, per tant fem:

- * posar 3 a ordre d'exploració (4 és el 3r vèrtex de l'exploració)
- * posar 1 a predecessor (venim de 1)
- * eliminar 4 de la columna d'adjacents de 1 (ja ha estat explorat)

2		1	3		
3		3	1		
1	2	3	4*	5	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	5		6		

- Estem en el vèrtex 4.

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 4 és 2. Anem a la columna 2.

El vèrtex 2 té predecessor? NO, per tant fem:

- * posar 4 a ordre d'exploració (2 és el 4t vèrtex de l'exploració)
- * posar 4 a predecessor (venim de 4)
- * eliminar 2 de la columna d'adjacents de 4 (ja ha estat explorat)

2	4	1	3		
3	4	3	1		
1	2*	3	4	5	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	5		6		

- Estem en el vèrtex 2.

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 2 és 1. Anem a la columna 1.

El vèrtex 1 té predecessor? SI, per tant fem:

- * tornar al vèrtex 2.
- * eliminar 1 de la columna d'adjacents de 2 (ja ha estat explorat)

2	4	1	3		
3	4	3	1		
1	2*	3	4	5	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	5		6		

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 2 és 3. Anem a la columna 3.

El vèrtex 3 té predecessor? SI, per tant fem:

- * tornar al vèrtex 2.
- * eliminar 3 de la columna d'adjacents de 2 (ja ha estat explorat)

2	4	1	3		
3	4	3	1		
1	2*	3	4	5	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	5		6		

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 2 és 4. Anem a la columna 4.

El vèrtex 4 té predecessor? SI, per tant fem:

- * tornar al vèrtex 2.
- * eliminar 4 de la columna d'adjacents de 2 (ja ha estat explorat)

2	4	1	3		
3	4	3	1		
1	2*	3	4	5	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	5		6		

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 2 és 5. Anem a la columna 5.

El vèrtex 5 té predecessor? NO, per tant fem:

- * posar 5 a ordre d'exploració (5 és el 5è vèrtex de l'exploració)
- * posar 2 a predecessor (venim de 2)
- * eliminar 5 de la columna d'adjacents de 2 (ja ha estat explorat)

2	4	1	3	5	
3	4	3	1	2	
1	2	3	4	5*	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	5		6		

- Estem en el vèrtex 5.

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 5 és 2. Anem a la columna 2.

El vèrtex 2 té predecessor? SI, per tant fem:

- * tornar al vèrtex 5.
- * eliminar 2 de la columna d'adjacents de 5 (ja ha estat explorat)

2	4	1	3	5	
3	4	3	1	2	
1	2	3	4	5*	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	5		6		

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 5 és 4. Anem a la columna 4.

El vèrtex 4 té predecessor? SI, per tant fem:

- * tornar al vèrtex 5.
- * eliminar 4 de la columna d'adjacents de 5 (ja ha estat explorat)

2	4	1	3	5	
3	4	3	1	2	
1	2	3	4	5*	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	3		6		

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 5 és 3. Anem a la columna 3.

El vèrtex 3 té predecessor? SI, per tant fem:

- * tornar al vèrtex 5.
- * eliminar 3 de la columna d'adjacents de 5 (ja ha estat explorat)

2	4	1	3	5	
3	4	3	1	2	
1	2	3	4	5*	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	3		6		

Queden adjacents per explorar? NO, per tant fem BACKTRACKING i anem a la columna del predecessor: columna 2.

2	4	1	3	5	
3	4	3	1	2	
1	2*	3	4	5	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	3		6		

- Estem en el vèrtex 2.

Queden adjacents per explorar? NO, per tant fem BACKTRACKING i anem a la columna del predecessor: columna 4.

2	4	1	3	5	
3	4	3	1	2	
1	2	3	4*	5	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	3		6		

- Estem en el vèrtex 4.

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 4 és 1. Anem a la columna 1.

El vèrtex 1 té predecessor? SI, per tant fem:

- * tornar al vèrtex 4.
- * eliminar 1 de la columna d'adjacents de 4 (ja ha estat explorat)

2	4	1	3	5	
3	4	3	1	2	
1	2	3	4*	5	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	5	3	
	3		6		

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 4 és 5. Anem a la columna 5.

El vèrtex 5 té predecessor? SI, per tant fem:

- * tornar al vèrtex 4.
- * eliminar 5 de la columna d'adjacents de 4 (ja ha estat explorat)

2	4	1	3	5	
3	4	3	1	2	
1	2	3	4*	5	6
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	3	3	
	3		6		

Queden adjacents per explorar? SI, per tant fem:

- El primer adjacent no explorat de 4 és 6. Anem a la columna 6.

El vèrtex 6 té predecessor? NO, per tant fem:

- * posar 6 a ordre d'exploració (6 és el 6è vèrtex de l'exploració)
- * posar 4 a predecessor (venim de 4)
- * eliminar 6 de la columna d'adjacents de 4 (ja ha estat explorat)

2	4	1	3	5	6
3	4	3	1	2	4
1	2	3	4	5	6*
4	1	1	2	2	4
2	3	2	1	4	
3	4	5	3	3	
	3		6		

Final (Perquè ja hem explorat tots els vèrtexs del graf).

S'observarà que la filosofia general de l'algorisme, tal com s'havia dit al principi, és anar tan lluny com sigui possible i tornar enrera quan no es pot progressar més. Vist sobre la plantilla això vol dir que quan estem en un vèrtex (o sigui, en una columna de la plantilla) procedim segons les prioritats següents

- Primera prioritat: canviar de columna i anar a un vèrtex no explorat.
- Segona prioritat: avançar dins d'aquella columna.
- Tercera prioritat: fer backtracking (tirar enrera).

Presentem a continuació una versió més formal de l'algorisme (algunes vegades anomenat algorisme de Trémaux)

Sigui N el nombre de vèrtexs del graf. Farem servir els següents vectors i variables

- $\text{pred}(i)$, $i = 1, \dots, n$. Conté el predecessor del vèrtex i en l'exploració.
- $\text{odexp}(i)$, $i = 1, \dots, n$. Conté el número d'ordre amb el qual s'ha incorporat el vèrtex i a l'exploració.
- $\text{gr}(i)$, $i = 1, \dots, n$. Conté el grau del vèrtex i . Aquest vector és constant i no s'actualitza.
- $\text{adjexpl}(i)$, $i = 1 \dots n$. Conté un valor entre 0 i $\text{gr}(i)$ que apunta al primer vèrtex no eliminat en la llista d'adjacents de i .

- $l_{listadj}(i, j)$, $i = 1, \dots, n$; $j = 1, \dots, gr(i)$. Conté la llista d'adjacents al vèrtex i .
- k . El seu valor és el nombre de vèrtexs incorporats a l'exploració.
- $vertact$. El seu valor és el vèrtex actual.

Inicialització començant en el vèrtex ℓ

$pred(i) = 0$, $i = 1, \dots, n$
 $gr(i) = \text{grau del vèrtex } i$, $i = 1, \dots, n$
 $adjexpl(i) = 0$, $i = 1, \dots, n$
 $k = 0$
 $odexpl(\ell) = 1$
 $pred(\ell) = \ell$, o qualsevol valor $\neq 0$
 $vertact = \ell$.

Bucle principal

- Si ($vertact = \ell$ i $adjexpl(vertact) = gr(vertact)$), hem acabat.
És a dir, hem tornat al vèrtex de sortida ℓ i no queden adjacents de ℓ per explorar.
- Mentre no hagim acabat, repetir
 - Si $adjexpl(vertact) = gr(vertact)$ fem $vertact = pred(vertact)$.
És a dir, fem backtracking.
 - En cas contrari fem
 - * $adjexpl(vertact) = adjexpl(vertact) + 1$.
És a dir, si queden adjacents del vèrtex actual per explorar passem al primer no eliminat de la llista d'adjacents.
Això equival a eliminar $adjexpl(vertact)$ d'aquesta llista.
 - * $j = l_{listadj}(vertact, adjexpl(vertact))$
Ara j és el primer vèrtex no eliminat de la llista d'adjacents del vèrtex actual.
 - * Si $pred(j) = 0$ fem
 - $pred(j) = vertact$
 - $vertact = j$
 - $k = k + 1$
 - $odexpl(vertact) = k$

És a dir, si encara no havíem incorporat j a l'exploració, anotem a partir de quin vèrtex hi hem arribat, saltem al vèrtex j i anotem amb quin número d'ordre l'incorporem a l'exploració.

- Si $k = n$ el graf és connex. En cas contrari n'hem trobat una component connexa.

1.4.2 Breadth First Search

La idea intuïtiva de l'algorisme BFS (recorregut per amplada prioritària) és la següent. Comencem a qualsevol vèrtex ℓ del graf i recorrem primer tots els adjacents v_1, v_2, \dots, v_k a aquest vèrtex. Quan no queda cap adjacent per explorar, anem a v_1 i explorem tots els seus adjacents, després fem el mateix amb $v_2, v_3 \dots$ fins arribar a v_k . Quan hem incorporat l'últim adjacent a v_k , anem al primer adjacent a v_1 i anem repetint el procés.

Capítol 2

Camins mínims

Els grafs s'utilitzen sovint per representar xarxes de comunicació o transport. Per aquestes xarxes, ens interessarà conèixer quin és el camí més curt entre vèrtexs.

A partir d'ara considerem el graf $G(V, A)$ que tenen un nombre anomenat pes associat a les seves arestes o arcs. Aquest pesos poden representar un cost, un temps de recorregut, una capacitat, etc. Per a representar els pesos en les arestes o arcs, considerem cada aresta o arc com una terna (a, b, p) on (a, b) indica el vèrtex inicial i final de l'aresta o arc i p el pes associat a (a, b) .

Tractarem el problema de determinar el camí de cost total mínim que uneix dos vèrtexs fixats s (vèrtex inicial) i t (vèrtex final) en un graf G amb costos en les arestes. Els diferents algorismes que es presenten corresponen a diverses possibilitats depenent de si els costos són positius o negatius i del nombre de camins que volem trobar, és a dir

- si volem trobar el camí de cost mínim entre dos vèrtexs fixats, o bé
- si volem el camí de cost mínim des d'un vèrtex inicial fixat a tots els altres vèrtexs del graf, o bé
- si volem el camí de cost mínim entre totes les possibles parelles de vèrtexs.

2.1 Algorisme de Dijkstra

Aquest algorisme troba un camí de cost mínim entre un vèrtex inicial fixat s i un vèrtex final fixat t , o bé, entre un vèrtex inicial fixat s i tots els vèrtexs del graf G .

Les característiques que ha de tenir el graf per poder aplicar aquest algorisme són que no hi pot haver costos negatius en les arcs i que el graf ha de ser dirigit (si el graf és simètric podem desdoblar cada aresta en dos arcs, de sentits oposats, i l'algorisme és igualment aplicable).

La idea intuïtiva de l'algorisme és anar etiquetant els vèrtexs de forma que en cada pas donem per definitiva una etiqueta que serà la mínima que pot rebre el vèrtex. L'etiquetatge dels vèrtexs es fa analitzant els successors de cada un dels vèrtexs. En línies generals els passos a realitzar són:

Inicialització

- $eti(s) = 0$, els altres vèrtexs amb etiqueta $+\infty$.
- El vèrtex s ha rebut l'etiqueta definitiva i el marquem amb $*$ o bé $def(s) = 1$.
- Ens situem en el vèrtex s , és a dir el vèrtex actual serà s .

Bucle principal

- Totes les etiquetes són definitives? o (el vèrtex t té etiqueta definitiva?)
 - SI; hem acabat
 - NO
 1. Explorar els vèrtexs adjacents al vèrtex actual (fer canvis si cal i actualitzar el vector d'antecedents)
 2. Buscar l'etiqueta mínima de les no definitives, fer-la definitiva i fer que el vèrtex actual sigui el vèrtex on hi ha l'etiqueta mínima.

Per anar seguint els passos que realitzaria l'algorisme, farem un exemple sobre un graf senzill. Sigui G un graf amb vèrtexs i arestes definides per

$$V = \{1, 2, 3, 4, 5, 6, 7\}$$

$$A = \{(1, 2, 3), (1, 3, 5), (3, 2, 8), (2, 6, 2), (2, 5, 5), (3, 5, 4), (3, 4, 9), (5, 4, 1), (5, 6, 2), (6, 7, 5), (5, 7, 7), (4, 7, 3)\}$$

Suposem que el vèrtex inicial fixat s és el vèrtex 1. i fixem un vèrtex final $t = 7$.

L'execució de l'algorisme l'estructurem amb dues taules. La primera hi guardem les etiquetes que anem associant a cada vèrtex, amb la següent estructura

vèrtexs	etiqueta						
1	0*						
2	∞	3*					
3	∞	5	5*				
4	∞	∞	∞	14	14	9*	
5	∞	∞	8	8	8*		
6	∞	∞	5	5*			
7	∞	∞	∞	∞	10	10	10*

La segona taula conté els antecedents a cada vèrtex per a poder reconstruir el camí de cost total mínim. Els antecedents es van modificant quan s'efectua algun canvi d'etiqueta.

1	2	3	4	5	6	7
0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	1	1	0	2	2	0
0	1	1	3	2	2	0
0	1	1	3	2	2	6
0	1	1	5	2	2	6
0	1	1	5	2	2	6

Si hi ha costos negatius pot ser que s'arribi a t sense haver detectat un camí de cost més petit al trobat. Això és així, perquè l'algorisme suposa que tots els costos són no negatius quan assumeix que tornar a un vèrtex ja estudiat anteriorment incrementarà sempre el cost aconseguit.

Per exemple, si apliquem l'algorisme al següent graf:

$$V = \{1, 2, 3, 4\}$$

$$A = \{(1, 2, 5), (1, 3, 9), (3, 2, -6), (2, 4, 3)\}$$

prenent com a vèrtex inicial $s = 1$ i com a vèrtex final $t = 4$, obtenim que el cost mínim és 8 quan en realitat es pot trobar un camí de longitud 6.

A continuació, presentem una versió més formal de l'algorisme.

Sigui n el nombre de vèrtexs del graf. Farem servir les següents variables:

- $eti(i)$, $i = 1, \dots, n$. Conté les etiquetes de cada un dels vèrtexs.
- $def(i)$, $i = 1, \dots, n$. Aquesta variable prendrà el valor 0 si el vèrtex no té l'etiqueta definitiva i el valor 1 si el vèrtex ha rebut l'etiqueta definitiva.
- $ant(i)$, $i = 1, \dots, n$. Aquesta variable conté el vèrtex antecessor al vèrtex i .
- $vact$ que conté el vèrtex que estem explorant en aquell moment.
- $Cost(i, j)$ representa el pes associat a l'aresta (i, j) .

Algorisme de Dijkstra

```
etiqa(s) = 0; //Origen
def(s) = 1; //Marquem que hem passat per l'origen
per tot v ≠ s fer
    etiqa(v) := ∞; //Posem tots els destins a la distància màxima possible
fper tot
u:=s; //Node on estem a partir del qual mirarem quin és el més proper
//Mentre no arribem al node que volem anar fem
mentre def(t) = 0 fer
    v := min{etiqa(u)}; //agafem el node que tenim més a prop de u i que no hi haguem passat
    //Per tots els nodes w successors de u mirem quin camí és més curt
    //si el que tenim o el que passa per u i arriba a ells
    per tot w amb def(w) ≠ 1 successor de u fer
        //En cas que sigui millor el nou camí el guardem
        si etiqa(w) > etiqa(u) + Cost(u, w) llavors
            etiqa(w) := etiqa(u) + Cost(u, w);
        fsi
    fper tot
    def(v) = 1; // Marquem que hem passat per v
    u:=v; // Ara aquest és el node a partir del qual mirarem la resta de nodes del graf
fmentre
si etiqa(t) = ∞ llavors
    escriure ("No hi ha cap camí d's a t")
sino
    escriure ("El cost del camí de cost mínim d's a t és", etiqa(t))
fsi
```

2.1.1 Complexitat de l'algorisme de Dijkstra

Per avaluar la complexitat de l'algorisme, notem que el pas que porta més feina és el que determina l'etiqueta mínima entre els vèrtexs que no tenen etiqueta definitiva. Inicialment, tenim n vèrtexs, per tant hem de fer $n - 1$ comparacions, i en cada pas donem una etiqueta definitiva. Així doncs el nombre total de comparacions és

$$(1 + 2 + \dots + (n - 1)) = \frac{(1 + n - 1)(n - 1)}{2} = \frac{n(n - 1)}{2} = O(n^2)$$

2.2 Algorisme de Ford

Aquest algorisme, determina els costos de tots els camins de cost mínim per anar des d'un vèrtex inicial fixat s fins a tot altre vèrtex $v_i \in V$ del graf.

En aquest cas, per poder aplicar l'algorisme, els costos no estan restringits, però no pot tenir circuits de cost negatiu. El graf ha de ser dirigit, però al igual que en cas anterior, si el graf és simètric podem desdoblar cada aresta en dos arcs, de sentits oposats, i l'algorisme és igualment aplicable, sempre i quant no hi hagi arestes de cost negatiu, ja que llavors tindríem un circuit de cost negatiu, i en aquest cas, l'algorisme podria quedar bloquejat actualitzant indefinidament les etiquetes dels vèrtexs del circuit.

La idea intuïtiva de l'algorisme és anar etiquetant els vèrtexs de forma que en cada pas donem per definitiva una etiqueta que serà la mínima que pot rebre el vèrtex. L'etiquetatge es fa en un ordre que vindrà donat pels arcs. Així doncs, abans d'aplicar l'algorisme, numerem els arcs com a_1, a_2, \dots, a_m i executem l'algorisme analitzant cada un dels arcs per ordre. Després de la primera passada, en fem les posteriors fins que no es produeix cap millora en les etiquetes.

En línies generals, els passos a realitzar són els següents:

Inicialització

- $\text{eti}(s) = 0$, els altres vèrtexs amb etiqueta $+\infty$.
- Enumerem les arestes, que ens establiran l'ordre del recorregut.

Bucle principal

- Explorar els vèrtexs en l'ordre que ens indica la numeració de les arestes (fer canvis si cal)
- L'algorisme s'acaba quan després d'una passada per totes les arestes, no s'ha efectuat cap canvi.

A continuació, presentem una versió més formal de l'algorisme de Ford.

Algorisme de Ford

```
etiq( $s$ ) = 0;
per tot  $v \neq s$  fer
    etiq( $v$ ) :=  $\infty$ ;
fper tot
passes:=0;
canvis:=cert;
mentre canvis i passes  $\leq n$  fer
    canvis:=fals;
    passes:=passes+1;
    per tot  $(v, w) \in A$  fer
        si etiq( $w$ ) > etiq( $v$ ) + Cost( $v, w$ ) llavors
            etiq( $w$ ) := etiq( $v$ ) + Cost( $v, w$ );
            canvis:=cert;
        fsi
    fper tot
fmentre
si passes= $n + 1$  llavors
    escriure (“Grafs amb circuits de cost negatiu”)
sino
    per tot  $v \neq s$  fer
        escriure (“El cost del camí de cost mínim d’ $s$  a  $v$  és”, etiq( $v$ ))
    fper tot
fsi
```

Per anar seguint els passos que realitzaria l'algorisme, farem un exemple sobre un graf senzill. Sigui G un graf amb vèrtexs i arestes

$$V = \{1, 2, 3, 4\}$$

$$A = \{(4, 2, -5, a_1), (3, 1, 3, a_2), (1, 2, -1, a_3), (3, 4, -1, a_4), (2, 3, 7, a_5), (1, 4, 3, a_6)\}$$

Suposem que el vèrtex inicial fixat s és el vèrtex 1.

Considerem una taula que ens dóna les etiquetes de cada un dels vèrtexs, amb la següent estructura

aresta	1	2	3	4	canvis
etiqueta inicial	0	∞	∞	∞	0
a_1	0	∞	∞	∞	0
a_2	0	∞	∞	∞	0
a_3	0	-1	∞	∞	1
a_4	0	-1	∞	∞	0
a_5	0	-1	6	∞	1
a_6	0	-1	6	3	1
a_1	0	-2	6	3	1
a_2	0	-2	6	3	0
a_3	0	-2	6	3	0
a_4	0	-2	6	3	0
a_5	0	-2	5	3	1
a_6	0	-2	5	3	0
a_1	0	-2	5	3	0
a_2	0	-2	5	3	0
a_3	0	-2	5	3	0
a_4	0	-2	5	3	0
a_5	0	-2	5	3	0
a_6	0	-2	5	3	0

Per a reconstruir els diferents camins que van de s a cada un dels vèrtexs del graf, utilitzarem el següent algorisme

- **Definicions:** $\text{Cost}(v, w) = \text{cost de l'arc } (v, w)$
Camí = conjunt de vèrtexs que formen el camí de cost mínim d' s a t .
- **Condicions inicials:** $\text{eti}_q(v_i)$ conté els valors obtinguts amb l'algorisme original.
- **Resultat:** La llista "Camí" conté el camí de cost mínim trobat.

Algorisme de reconstruir el camí

```
Camí:={t};
w := t;
inici:=fals;
mentre no inici fer
    trobat:=fals;
    per tot v antecessor de w i no trobat fer
        si etiq(v) = etiq(w) - Cost(v, w) llavors
            Camí:=Camí+{v};
            trobat:=cert;
            w := v;
            si v = s llavors
                inici:=cert;
            fsi
        fsi
    fper tot
fmentre
```

2.2.1 Complexitat de l'algorisme de Ford

Si el graf G no conté circuits de cost negatiu, el procés serà finit. A més, si un camí òptim de s a t està format per k arcs, aleshores, en la k -èsima passada, l'etiqueta de t segur que n'és ja l'etiqueta final. Això és així, perquè, en cada pas, almenys un vèrtex rep l'etiqueta òptima, que ja li queda com a definitiva, i la rep d'un vèrtex que ja té l'etiqueta òptima. D'aquesta manera el nombre total de passos serà $O(|V||A|)$.

2.3 Algorisme de Floyd

L'algorisme de Floyd determina el cost per a cada parella de vèrtexs $v_i, v_j \in V$, $i, j = 1, \dots, n$. Per poder aplicar aquest algorisme, el graf ha de ser dirigit. Si tenim un graf simètric podem desdoblar cada aresta en dos arcs, de sentits oposats, i l'algorisme és igualment aplicable. En aquest cas, a diferència de Ford, no cal exigir que no hi hagi circuits de cost negatiu, ja que el mètode de Floyd els detecta fàcilment.

La idea intuïtiva de l'algorisme és anar etiquetant els vèrtexs comparant els costos dels camins quan en cada pas s'hi afegeix un nou vèrtex.

- **Definicions:** C_{ij}^k = cost d'un camí de cost mínim des de v_i fins a v_j que no conté cap vèrtex etiquetat amb un subíndex major que k

$\text{cost}(v, w)$ = pes de l'arc (v, w)

$n = |V|$

$$C_{ij}^0 = \begin{cases} \text{cost}(v_i, v_j) & \text{si } (v_i, v_j) \in A \\ \infty & \text{en cas contrari} \end{cases}$$

- **Resultat:** El cost del camí de cost mínim de v_i a v_j es troba a C_{ij}^n .
-

Algorisme de Floyd

```

per k := 1 fins n fer
  per i := 1 fins n fer
    per j := 1 fins n fer
       $C_{ij}^k := \min \{C_{ij}^{k-1}, C_{ik}^{k-1} + C_{kj}^{k-1}\}$ 
    fper
  fper
fper
negatiu:=fals;
per i := 1 fins n fer
  si  $C_{ii}^n < 0$  llavors
    negatiu:=cert;
  fsi
fper
si negatiu llavors
  escriure ("Grafs amb circuits de cost negatiu")
sino
  per i := 1 fins n fer
    per j := 1 fins n fer
      escriure ("El cost del camí de cost mínim de  $v_i$  a  $v_j$  és",  $C_{ij}^n$ )
    fper
  fper
fper

```

Sigui G un graf dirigit

$$V = \{1, 2, 3, 4\}$$

$$A = \{(1, 2, 4), (2, 1, -1), (2, 4, -3), (2, 3, 3), (1, 3, 3), (3, 1, 4), (1, 4, 2), (4, 1, 1), (3, 4, 5), (4, 3, -1)\}$$

En la taula següent indiquem, en les columnes, les diferents etapes del procés i, en les files, els cost del camí actual més curt entre cada parella de vèrtexs.

	C^0	C^1	C^2	C^3	C^4
11	0	0	0	0	0
12	4	4	4	4	4
13	3	3	3	3	0
14	2	2	1	1	1
21	-1	-1	-1	-1	-2
22	0	0	0	0	0
23	3	2	2	2	-4
24	-3	-3	-3	-3	-3
31	4	4	4	4	4
32	∞	8	8	8	8
33	0	0	0	0	0
34	5	5	5	5	5
41	1	1	1	1	1
42	∞	5	5	5	5
43	-1	-1	-1	-1	-1
44	0	0	0	0	0

2.3.1 Complexitat de l'algorisme de Floyd

La complexitat de l'algorisme és $O(n^3)$, ja que analitzem els n vèrtexs, per tant es fan n passos i en cada pas es fan n^2 comparacions.

Quan es vol determinar un camí òptim entre cada parella de vèrtexs del graf, resulta més eficient usar l'algorisme de Floyd que no pas aplicar l'algorisme de Dijkstra n^2 vegades, una per cada parella, o l'algorisme de Ford n vegades, una a partir de cada vèrtex.

Capítol 3

Arbres

3.1 Conceptes generals

Definició 3.1 (Arbre)

Sigui $G = (V, A)$ un graf simètric i sense llaços. El graf G és un arbre si G és connex i no conté cicles.

La definició d'arbre admet una sèrie de definicions equivalents. Sigui T un graf d'ordre n , aleshores, són equivalents:

1. T és un arbre.
2. T és un graf connex de n vèrtexs i $n - 1$ arestes.
3. T és un graf connex on cada aresta és una aresta pont.
4. T és un graf connex on cada parella de vèrtex es pot connectar per un únic camí simple.

Definició 3.2 (Arbre dirigit)

Sigui G un graf dirigit, llavors G és un arbre dirigit si el graf no dirigit associat amb G és un arbre. Si G és un arbre dirigit, G és un arbre amb arrel si existeix un únic vèrtex r , anomenat arrel amb grau interior igual a 0 i tots els altres vèrtexs amb grau interior igual a 1.

Per una arbre dirigit $T = (V, A)$ amb arrel r , podem donar les definicions següents:

1. Si uv és un arc, u és l'únic vèrtex adjacent cap a v . Aleshores es diu que u és el *pare* de v i que v és el *fill* de u .
2. Una *fulla* o *vèrtex terminal*, és un vèrtex u amb grau exterior igual a 0, és a dir, un vèrtex que no té fills.

3. Els vèrtexs que no són fulles es diuen *interns* o no terminals.
4. L'arbre amb arrel és *m-ari* si el grau exterior per a tots els vèrtexs interns és m . Quan $m = 2$ direm que és un arbre binari.
5. El *nivell* de $u \in V$ és la longitud de l'únic $r - u$ camí.
6. L'*altura* o *profunditat* de T és $h(T) = \max\{\text{nivell}(u) : u \in V\}$

Proposició 3.1

Si $T = (V, A)$ un arbre m -ari amb $|V| = n$. Si T té h fulles i l vèrtexs interns, llavors

1. $n = ml + 1$
2. $h = (m - 1)l + 1$

Demostració.

1. De la fórmula de graus tenim

$$|A| = \sum \text{grau exterior}(i) = m * l + h * 0 = ml$$

Com T és arbre amb $|V| = n$, llavors $|A| = n - 1$. Igualant tenim $n - 1 = ml \Rightarrow n = ml + 1$.

2. Si hi ha $n = ml + 1$ vèrtexs,

$$h = n - l = ml + 1 - l = (m - 1)l + 1$$

3.2 Arbres generats d'una graf

Definició 3.3 (Arbre generat)

Donat un graf $G = (V, A)$ connex, un arbre generat és un graf parcial de G que és arbre.

Donat un graf dirigit G , i fixat un vèrtex arrel r , volem determinar quants arbres generats dirigits d'arrel r conté G . Per resoldre aquest problema s'utilitza un mètode anomenat mètode de Tutte.

Definició 3.4 (Matriu de graus entrant)

Donat un multigraf dirigit $G = (V, A)$, la matriu de graus entrant és una matriu quadrada d'ordre n on

$$d_{ij} = \begin{cases} \text{grau interior}(i) & \text{si } i = j \\ -k & \text{en cas contrari} \end{cases}$$

essent k el nombre d'arcs (i, j) que hi ha a G . (Si G no és un multigraf, llavors $k = 0$ o $k = 1$)

Proposició 3.2 (Métode de Tutte)

El nombre d'arbres dirigits generats amb arrel v_r , d'un graf dirigit G sense llaços, ve donat pel menor de la seva matriu $D(G)$, obtingut un cop eliminades la r -èsima fila i columna.

Exemple 3.1

Considerem el multigraf dirigit definit per

$$V = \{1, 2, 3\}$$

$$A = \{(1, 2), (2, 1), (1, 3), (3, 1), (2, 3), (2, 3)\}$$

La matriu de graus entrant és

$$D = \begin{pmatrix} 2 & -1 & -1 \\ -1 & 1 & -2 \\ -1 & 0 & 3 \end{pmatrix}$$

Si prenem com arrel el vèrtex 2, tenim

$$D = \begin{vmatrix} 2 & -1 \\ -1 & 3 \end{vmatrix} = 5$$

i per tant tindrem 5 arbres diferents.

Es poden obtenir resultats semblants en el cas de que G sigui un multigraf simètric. En aquest cas, el mètode de Tutte s'aplica sobre la matriu de graus,

Definició 3.5 (Matriu de graus)

Donat un multigraf simètric $G = (V, A)$, la matriu de graus és una matriu quadrada d'ordre n on

$$d_{ij} = \begin{cases} \text{grau}(i) & \text{si } i = j \\ -k & \text{en cas contrari} \end{cases}$$

essent k el nombre d'arestes (i, j) que hi ha a G .

Observem que com que la matriu de graus és simètrica, el mètode de Tutte serà independent del vèrtex escollit com arrel.

En el cas particular de que $G = K_n$ s'obtenen n^{n-2} arbres diferents.

3.2.1 Recerca d'arbres generadors

Una forma de trobar arbres generadors és la que consisteix en suprimir del graf G donat, tots els seus cicles, de manera que ens quedi un graf connex però sense cicles, és a dir, un arbre. El problema és que aquest mètode no és eficient, ja que necessitem que els cicles estiguin identificats. Podem utilitzar els algorismes DFS (recerca en profunditat) i BFS (recerca en amplada) per obtenir arbres generadors.

3.3 Arbres generats de cost mínim d'un graf

Considerem un graf $G = (V, A)$ simètric amb un cost $\text{cost}(a_k)$ associat a cada aresta $a_k \in A$ (graf ponderat), que pot denotar una distancia entre nodes, un temps de recorregut, etc. Estudiarem el problema de determinar un arbre generat a G de cost total mínim. Els algorismes de Kruskal i Prim, que tractarem tot seguit, permeten obtenir arbres generadors de cost mínim d'un graf connex G . També podem obtenir arbres generadors de cost total màxim realitzant petites modificacions sobre els algorismes.

3.3.1 Algorisme de Kruskal

L'algorisme de Kruskal, té com a entrada un graf ponderat d'ordre n i com a sortida el conjunt d'arestes d'un arbre generador minimal de G . A cada etapa s'escull una aresta de pes mínim que no formi circuit amb les ja elegides. Quan se'n tenen $n - 1$ s'atura el procés. El conjunt d'aquestes arestes genera un arbre generador minimal de G . Les variables de l'algorisme són el conjunt F d'arestes que ja es troben en el graf que anem formant, el conjunt L d'arestes que no es troben en el graf que anem formant; el resultat és l'arbre $G(V, F)$.

Algorisme de Kruskal

```
 $F := \emptyset;$   
 $L := A;$   
 $G := (V, F);$   
Ordenar  $L$  en funció del cost;  
 $k := 0;$   
repetir  
     $a :=$ primera aresta de  $L$ ;  
     $L := L - \{a\};$   
    si  $a$  no forma circuit a  $G$  llavors  
         $k := k + 1;$   
         $F := F + \{a\};$   
    fsi  
fins que  $k = n - 1$ 
```

El control perquè no es formi circuit al anar afegint les arestes, es pot fer etiquetant inicialment tots els vèrtexs amb una etiqueta diferent. Al unir dos vèrtexs per una aresta, els hi assignem

la mateixa etiqueta. D'aquesta manera, si en anar afegint una aresta, dos vèrtexs tenen la mateixa etiqueta, no prendrem aquesta aresta ja que es crearia un circuit.

Per anar seguint els passos que realitzaria l'algorisme, farem un exemple sobre un graf senzill. Sigui G un graf amb vèrtexs i arestes

$$V = \{1, 2, 3, 4, 5, 6\}$$

$$A = \{(1, 2, 6), (1, 3, 4), (1, 5, 3), (2, 3, 2), (2, 4, 8), (2, 5, 4), (2, 6, 6), (3, 4, 9), (4, 5, 7), (5, 6, 2)\}$$

Considerem una taula amb la següent estructura

arestes ordenades	1	2	3	4	5	6	F
	A	B	C	D	E	F	
(3, 2)	A	B	B	D	E	F	SI
(5, 6)	A	B	B	D	E	E	SI
(5, 1)	A	B	B	D	A	A	SI
(2, 5)	B	B	B	D	B	B	SI
(1, 3)	B	B	B	D	B	B	NO
(1, 2)	B	B	B	D	B	B	NO
(2, 6)	B	B	B	D	B	B	NO
(4, 5)	B	B	B	B	B	B	SI
(4, 2)							
(3, 4)							

L'arbre generador de cost mínim que s'obté aplicant l'algorisme de Kruskal és $T = (V, F)$ on

$$V = \{1, 2, 3, 4, 5, 6\}$$

$$F = \{(3, 2, 2), (5, 6, 2), (1, 5, 3), (2, 5, 4), (4, 5, 7)\}$$

El cost total de l'arbre és 18.

Complexitat de l'algorisme de Kruskal

Pel que fa a la complexitat de l'algorisme, notem que el pas que porta més feina és el de l'ordenació de les arestes, la qual es pot realitzar amb un nombre d'operacions elementals de l'ordre $O(m \log m)$ on m denota el nombre d'arestes del graf.

3.3.2 Algorisme de Prim

L'estratègia de l'algorisme de Prim és escollir en cada etapa l'aresta de pes mínim incident amb alguna de les ja escollides i que no formi cicle. Si n és l'ordre del graf donat, quan es tenen $n - 1$ arestes s'atura el procés i el graf generat per aquestes arestes és un arbre generador minimal de G . Les variables de l'algorisme són el conjunt Q de vèrtexs que ja es troben en l'arbre que anem formant, el conjunt E d'arestes que ja es troben en l'arbre que anem formant, el conjunt C d'arestes candidates a entrar a l'arbre en el proper pas, $\text{Cost}(v, w)$ el cost de l'aresta (v, w) . El resultat és l'arbre $G(Q, E)$.

Algorisme de Prim

```

 $\forall (v, w) \notin G(V, A) \text{ cost}(v, w) = +\infty$ 
 $Q := \{v_1\};$ 
 $E := \emptyset;$ 
 $C := \{(v_1, v_i) \mid v_i \neq v_1\}$ 
mentre  $C \neq \emptyset$  fer
     $(v_i^*, v_j^*) = \min_{v_i \in Q, (v_i, v_j) \in C} \{C(v_i, v_j)\};$ 
     $Q := Q + \{v_j^*\};$ 
     $E := E + (v_i^*, v_j^*);$ 
     $C := C - (v_i^*, v_j^*);$ 
    per tot  $v_s \notin Q$  fer
        per tot  $(v_r, v_s) \in C$  fer
            si  $\text{Cost}(v_j^*, v_s) < \text{Cost}(v_r, v_s)$  llavors
                 $C := C - (v_r, v_s);$ 
                 $C := C + (v_j^*, v_s);$ 
            fsi
        fper tot
    fper tot
fmentre

```

Sigui G el graf anterior on

$$V = \{1, 2, 3, 4, 5, 6\}$$

$$A = \{(1, 2, 6), (1, 3, 4), (1, 5, 3), (2, 3, 2), (2, 4, 8), (2, 5, 4), (2, 6, 6), (3, 4, 9), (4, 5, 7), (5, 6, 2)\}$$

Q	1		5		6		3		2		4
E			(1,5)		(5,6)		(1,3)		(3,2)		(5,4)
C	(1,2)	6	(5,2)	4	(5,2)	4	(3,2)	2*			
	(1,3)	4	(1,3)	4	(1,3)	4*					
	(1,4)	∞	(5,4)	7	(5,4)	7	(5,4)	7	(5,4)	7*	
	(1,5)	3*									
	(1,6)	∞	(5,6)	2*							

L'arbre generador de cost mínim que s'obté aplicant l'algorisme de Prim és $T = (V, E)$ on

$$V = \{1, 2, 3, 4, 5, 6\}$$

$$E = \{(1, 5, 3), (5, 6, 2), (1, 3, 4), (3, 2, 2), (5, 4, 7)\}$$

El cost total de l'arbre és 18.

Complexitat de l'algorisme de Prim

Pel que fa a la complexitat de l'algorisme, els passos de buscar el mínim cost i el de comparar costos, realitzen, cadascun, $n - k$ comparacions ($k = 1, \dots, n - 1$). Així, el total és $O(n^2)$ comparacions.

Com a conclusió, podem establir que és millor usar l'algorisme de Prim si el graf és força dens, donat que la seva complexitat no depèn del nombre d'arestes.

Capítol 4

Xarxes de transport

4.1 Algorisme de Ford-Fulkerson

- **Descripció:** Troba el flux màxim que pot passar per una xarxa $G(V, A)$
- **Definicions:**
 - $F = \{f[v, w]\}$ flux que passa pel graf de tal manera que $f[v, w]$ correspon al flux que passa per l'arc (v, w)
 - $\text{Etiqueta}[v] = [E1[v], E2[v]]$
 - $\text{Capacitat}[v, w] = \text{capacitat de l'arc } (v, w)$
- **Resultat:** El flux màxim es troba a F_{\max} i correspon al flux $F = \{f[v, w]\}$.

Algorisme

$F_{\max} := 0;$

per tot $(v,w) \in A$ **fer**

$f[v,w] := 0;$

fper tot

Fer 0 totes les etiquetes;

$llista := \{s\};$

$etiqueta[s] := [s, \infty];$

mentre $llista \neq \emptyset$ **fer**

$v :=$ un vèrtex qualsevol de la llista;

$llista := llista - \{v\};$

Explorar el vèrtex v ;

si t està etiquetat **llavors**

augmentar el flux F ;

fer 0 totes les etiquetes;

$llista := \{s\};$

$etiqueta[s] := [s, \infty]$

fsi

fmentre

Escriure (“El flux màxim que pot passar per la xarxa és”, F_{\max})

acció Explorar el vèrtex v ;

per tot $w \in \Gamma(v)$ no etiquetat **fer**

si $f[v,w] < \text{capacitat}[v,w]$ **llavors**

$E1(w) := v$;

$E2(w) := \min\{E2(v), \text{capacitat}[v,w] - f[v,w]\}$;

$\text{llista} := \text{llista} + \{w\}$;

fsi

fper tot

per tot $w \in \Gamma^{-1}(v)$ no etiquetat **fer**

si $f[v,w] > 0$ **llavors**

$E1(w) := -v$;

$E2(w) := \min\{E2(v), f[v,w]\}$;

$\text{llista} := \text{llista} + \{w\}$;

fsi

fper tot

facció

acció Augmentar el flux F ;

$\text{flux} := E2[t]$;

$w := t$;

$v := E1[t]$;

mentre $v \neq s$ **fer**

si $v > 0$ **llavors**

$f[v,w] := f[v,w] + \text{flux}$;

sino

$v := -v$;

$f[v,w] := f[v,w] - \text{flux}$;

fsi

$w := v$;

$v := E1[w]$;

fmentre

$F_{\max} := F_{\max} + \text{flux}$;

facció

4.2 Algorisme de Busacker i Gowen

- **Descripció:** Troba el camí de cost mínim per a on ha de passar el flux fixat ϵ

- **Definicions:**

- ϵ = flux demanat
- $F = \{f_{ij}\}$ Flux que passa per la xarxa, pels arcs (v_i, v_j)
- $C(P)$ = cost del camí P
- $K \circ (P)$ associa a G un flux $F = \{f_{ij}\}$ de valor

$$f_{ij} = \begin{cases} k & \text{si } (v_i, v_j) \in P \\ 0 & \text{si } (v_i, v_j) \notin P \end{cases}$$

- $G^\mu(F) = G(V^\mu, A^\mu)$ on

$$V^\mu = V$$

$$A^\mu = A_1^\mu \cup A_2^\mu \text{ on}$$

$$A_1^\mu = \{(v_i^\mu, v_j^\mu) \mid F_{ij} < q_{ij}\} \text{ amb } q_{ij}^\mu = q_{ij} - F_{ij} \text{ i } c_{ij}^\mu = c_{ij}$$

$$A_2^\mu = \{(v_j^\mu, v_i^\mu) \mid F_{ij} > 0\} \text{ amb } q_{ij}^\mu = F_{ij} \text{ i } c_{ij}^\mu = -c_{ij}$$

- **Resultat:** El cost es troba a la variable “cost”, i el flux passa segons $F = \{f_{ij}\}$

Algorisme

```
F:=0;
cost:=0;
mentre F <  $\epsilon$  fer
  construir  $G^\mu(F)$ ;
  P:=camí de cost mínim a  $G^\mu(F)$ ;
  k:=  $\min_{(v_i^\mu, v_j^\mu) \in P} \{q_{ij}^\mu\}$ ;
  si F + (k  $\circ$  (P))  $\leq \epsilon$  llavors
    F := F + (k  $\circ$  (P));
    cost := cost+k*C(P);
  sino
    h := F + (k  $\circ$  (P));
    F := F + ((k - h +  $\epsilon$ )  $\circ$  (P));
    cost := cost + (k - h +  $\epsilon$ ) * C(P);
  fsi
fmentre
Escriure (“el cost és”, cost)
```


Capítol 5

Grafs Eulerians i Hamiltonians

5.1 Algorisme de Fleury

- **Descripció:** Troba un circuit eulerià d'un graf $G(V,A)$.
- **Definicions:** v = vèrtex actual
 $A' = \{ \text{arestes ja recorregudes} \}$
 $L = \{ \text{vèrtexs ordenats en la seqüència que han estat recorreguts} \}$
 $A(v) = \{ \text{vèrtexs adjacents al vèrtex } v \text{ dins el graf } G(V, A-A') \}$
 w = primer vèrtex del recorregut
- **Resultat:** El circuit Eulerià es troba a la llista L

Algorisme

```
L := w;  
v := w;  
A' := ∅;  
mentre  $|A(w)| > 0$  fer  
    si  $|A(v)| > 1$  llavors  
        buscar un  $u \in A(v)$  tal que  $(v,u)$  no sigui un pont de  $G(V, A-A')$   
        sino sigui  $u$  el vèrtex de  $A(v)$   
    fsi  
     $A(v) := A(v) - u$ ;  
     $A(u) := A(u) - v$ ;  
     $A' := A' + (v,u)$ ;  
     $v := u$ ;  
     $L := L + v$ ;  
fmentre
```

5.2 Algorisme d'Edmonds

- **Descripció:** Troba una solució al problema del carter xinès per a grafs simètrics.
- **Definicions:** $\text{Cost}(C_{ij}) = \text{cost del camí mínim entre } v_i \text{ i } v_j$

Algorisme

- PAS 1: Donat $G(V,A)$ formar una partició de V en dos conjunts
- $$V_p = \{v_i \in V \mid \text{grau}(v_i) \text{ és parell}\}$$
- $$V_s = \{v_i \in V \mid \text{grau}(v_i) \text{ és senar}\}$$
- PAS 2: Obtenir $\text{Cost}(C_{ij})$ per a qualssevol $v_i, v_j \in V_s, v_i \neq v_j$
- PAS 3: Formar el graf complet $G'(V_s, A')$ que té en cada arc (v_i, v_j) un cost associat $c_{ij} = \text{Cost}(C_{ij})$
- PAS 4: Determinar l'aparellament perfecte de cost menor M^* a $G'(V_s, A')$
- PAS 5: Duplicar a G les arestes dels camins de cost c_{ij} corresponents als costos seleccionats a M^*
- PAS 6: El graf resultant té un circuit eulerià que és la solució

5.3 Algorisme del carter xinès per a grafs dirigits

- **Descripció:** Troba una solució al problema del carter xinès per a grafs dirigits.

Algorisme

PAS 1: Per a qualssevol $v_i \in V$ $g(v_i) = \text{grau interior}(v_i) - \text{grau exterior}(v_i)$

PAS 2: Construir una xarxa de transport $G'(V', A')$ amb

$$V' = V \cup \{v_s, v_t\}$$

A' verificant:

$$\forall v_i \in V \quad g(v_i) > 0 \text{ fer l'arc } (v_s, v_i) \text{ amb } q_{si} = g(v_i) \text{ i } c_{si} = 0$$

$$\forall v_j \in V \quad g(v_j) < 0 \text{ fer l'arc } (v_j, v_t) \text{ amb } q_{jt} = -g(v_j) \text{ i } c_{jt} = 0$$

$$\forall a_{ij} \in A \quad \text{mantenir l'arc amb } q_{ij} = \infty \text{ i } c_{ij} = c_{ij}$$

PAS 3:
$$\epsilon = \sum_{a_{si} \in A'} q_{si} = \sum_{a_{jt} \in A'} q_{jt}$$

Minimitzar els costos per a un flux fixat ϵ

PAS 4: Construir un graf $G''(V'', A'')$ amb

$$V'' = V$$

$$A'' = A \cup \{ \text{afegir els arcs de } G' \text{ per on passa flux, tants cops com unitats de flux passin} \}$$

5.4 Obtenció de circuits Hamiltonians pel mètode de les multiplicacions llatines

- **Restriccions:** Troba circuits hamiltonians s i només si el graf en té.
- **Definicions:** $n = \text{card}(V)$
- **Resultats:** La solució es troba a M^{n-1}

Algorisme

Formar la matriu quadrada $M^1 = (m_{ij}^1)$ d'ordre n on:

$$m_{ij}^1 = \begin{cases} i, j & \text{si } (v_i, v_j) \in A \\ 0 & \text{en cas contrari} \end{cases}$$

Formar la matriu quadrada $N = (n_{ij})$ d'ordre n on:

$$n_{ij} = \begin{cases} j & \text{si } (v_i, v_j) \in A \\ 0 & \text{en cas contrari} \end{cases}$$

per $k:=1$ **fins a** $n-2$ **fer**

$$M^{k+1} := M^k \otimes N$$

fper

5.5 Algorisme de Robert i Flores

1. INICIALITZACIÓ

$v := v_1$; $S := \{v\}$; cap vèrtex de P marcat

2. ESCOLLIR UN VÈRTEX CANDIDAT

Si existeix un vèrtex w candidat¹ a la columna de P corresponent a v

Lavors anar al pas 3

Sinó anar al pas 5

3. AVANÇAR

Marcar w a la columna de P

$S := S \cup \{w\}$

Si $|S| = n$ Lavors anar al pas 4

Sinó $v := w$; anar al pas 2

4. CONDICIÓ D'ACABAMENT

Si busquem un camí Lavors STOP

Si busquem un circuit Lavors

Si existeix l'arc de retorn (w, v_1) Lavors $S := S \cup \{v_1\}$; STOP

Sinó anar al pas 5

5. DESFER EL CAMÍ

treure el darrer element de S

desmarcar-lo

$v :=$ darrer element de S

anar al pas 2

¹candidat= $\{w \notin S, \text{ no marcat i no desmarcat en el darrer pas } \}$