Problemes algorísimia

Miquel Pere Baztán Grau
Bernat Dosrius Lleonart
Xavier Momplet Gil
Emma Ventura Font

Universitat Politècnica de Catalunya

21 de desembre 2024

Setmana 2 (20 febrer)

Problema 1.3 (celebritat?)

En una festa, un convidat es diu que és una celebritat si tothom el coneix, però ell no coneix a ningú (tret d'ell mateix). Les relacions de coneixença donen lloc a un graf dirigit: cada convidat és un vèrtex, i hi ha un arc entre u i v si u coneix a v.

(a) **Doneu una formalització de la propietat de ser celebritat.**

Sigui $G = (V, E)$ un graf dirigit. Una celebritat és un vèrtex

$$v \in V \mid \forall v' \in V \; v' \neq v \implies (v', v) \in E \; i \; (v, v') \notin E$$

(b) **Doneu un algorisme que, donat un graf dirigit representat amb una matriu d'adjacència, indica si hi ha o no cap celebritat. En el cas que hi sigui, cal dir qui és. El vostre algorisme ha de funcionar en temps $\mathcal{O}(n)$, on n és el nombre de vèrtexs.**

Notem que, per la definició de celebritat, només pot existir una celebritat donat un graf dirigit simple. També cal notar que la matriu amb la que treballarem en aquest problema és una matriu que només té $1's$ i $0's$ a les seves entrades.
Sigui a_{ij} l'entrada de la i-èssima fila i la j-èssima columna de la matriu d'adjacència, aleshores:

$$a_{ij} = \begin{cases} 0 & \text{si } (i,j) \in E \\ 1 & \text{si } (i,j) \notin E \end{cases}$$

Per a determinar si el graf té una celebritat, començarem des de la posició $(0, 0)$ de la matriu, anem augmentant j fins a trobar un 1, això ens indica que el vèrtex

de la fila en la que ens trobem no és famós (ja que coneix a un altre vèrtex). Un cop trobat aquest 1 saltem a la fila j ($i = j$) i seguim agumentant j fins a trobar un 1 o fins a arribar a $j == n$. En el moment en que $j == n$, tenim un candidat a celebritat (i-èssim vèrtex). Per a comprovar si i és una celebritat cal recòrrer la i-èssima fila i la j-èssima columna, cal comprovar:

$$\forall\ k \in [n-1] \mid k \neq i\ a_{kj} == 1$$

$$\forall\ l \in [n-1]\ a_{il} == 0$$

Aquestes dues condicions ens diuen que el vèrtex i és conegut per tots els altres vèrtex i ell mateix no coneix a cap altre vèrtex i, per tant, és una celebritat. Si no es verifiquen les dues condicions anteriors quan $j == n$, aleshores no existeix cap celebritat en G.

Problema 1.9 (és fortament connex?)

Un graf dirigit és fortament connex quan, per cada parell de vèrtexs u, v, hi ha un camí de u a v. Doneu un algorisme per determinar si un graf dirigit és fortament connex.

La idea de l'algorisme és fer un DFS (arrelant l'arbre del recorregut en un node $v \in G$ qualsevol) i adonar-nos de que un graf dirigit serà fortament connex si i només sí, és connex respecte v (es poden visitar tots els nodes del graf des de v) i quan ordenem els vèrtexos (segons l'ordre de visita $ndfs[u]$, $u \in G$), passa el següent:

$\forall u \in G$ tal que $u \neq v$ (l'arrel) el número més baix que es pot arribar seguint un camí des de u fins a algun dels seus descendents en l'arbre de recorregut del DFS (pujant amb una aresta de retrocès), anomenem-lo $mesAlt[u]$, és més petit estricte que $ndfs[u]$. És a dir, que tot vèrtex (excepte l'arrel) es pot "escapar"(o tornar) del seu pare i anar més amunt de l'arbre. Matemàticament, $mesAlt$[u] $<$ $ndfs$[u].

Amb un dibuix,

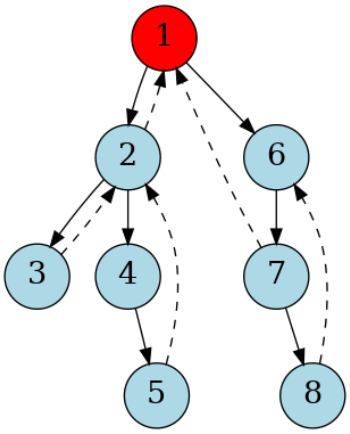

Figura 1: Exemple d'arbre fortament connex (les arestes discontínues són les arestes de retrocès)

L'algorisme és el següent:

Algorithm 1 DFS(G)

Require: G graf
Ensure: G és fortament connex?

```
1: for v ∈ V(G) do
2:     visitat[v] := false
3:     ndfs[v] := 0
4: end for
5:
6: numDfs := 0
7: esFortamentConnex := true
8:
9: v:= un vèrtex qualsevol de G
10: FortConnRec(G, v, v, esFortamentConnex, ndfs, visitat)
11:
12: if not esFortamentConnex then
13:     return false
14: end if
15:
16: for v ∈ V(G) do
17:     if not visitat[v] then
18:         return false
19:     end if
20: end for
21:
22: return esFortamentConnex
```

Algorithm 2 FortConnRec(G, v, pare, esFortamentConnex, ndfs, visistat)

Require: G graf

```
1: numDfs := numDfs + 1; ndfs[v] := numDfs
2: visistat[v] := true
3: mesAlt[v] := ndfs[v]
4:
5: for w ∈ G.ADJACENT(v) do
6:     if not visitat[w] then
7:         FortConnRec(G, w, v, esFortamentConnex, ndfs, visitat)
8:         mesAlt[v] := min(mesAlt[v], mesAlt[w])
9:     else
10:        mesAlt[v] := min(mesAlt[v], ndfs[w])
11:    end if
12: end for
13:
14: if pare != v then
15:     esFortamentConnex := esFortamentConnex or mesAlt[v] ≥ ndfs[v]
16: end if
```

Nota: Per calcular mesAlt[v] usem el mesAlt[w] dels fills directes que no hem visitat

(arestes contínues de l'arbre de la figura) i el ndfs[w] dels nodes que ja hem visitat (arestes discontínues) i prenem el mínim de tots ells.

Problema 1.10 (és semiconnex?)

Un graf dirigit $G = (V, E)$ és semiconnex si, per qualsevol parell de vèrtexs $u, v \in V$, tenim un camí dirigit de u a v o de v a u. Doneu un algorisme eficient per determinar si un graf dirigit G és semiconnex. Demostreu la correctesa del vostre algorisme i analitzeu-ne el cost. Dissenyeu el vostre algorisme fent us d'un algorisme que us proporcioni les components connexes fortes del graf en temps $\mathcal{O}(n + m)$.

Sigui $G = (V, E)$ un graf dirigit. Direm que G és semiconnex $\iff \forall u, v \in V,\ \exists C(u, v)$, on $C(u, v)$ és un camí entre u i v que definirem de la següent forma:

$$C(u, v) = \{u_0, u_1,\ \dots, u_n \in V \mid u_0 = u,\ u_n = v, (u_i, u_{i+1}) \in E\ \ \forall i = 0, \dots, n-1\}$$

Sigui $\sim\ \subseteq V \times V$, la relació d'equivalència tal que $\forall u, v \in V, u \sim v \iff \exists C(u, v)$ i $\exists C(v, u)$, és a dir, $u \sim v$ si i només si, pertanyen a la mateixa component fortament connexa.

Sigui $[v]$, la classe d'equivalència de v, és a dir, $[v] := \{u \in V \mid u \sim v\}$. D'aquesta manera podem definir el conjunt cocient $V_c := V/\sim = \{[u] \mid u \in V\}$ com el conjunt de totes les classes i $E_c = \{([u], [v]) : [u] \neq [v]$ i $\exists u' \in [u], v' \in [v]$ tal que $(u', v') \in E\}$ com les arestes entre vèrtexos de V_c.

D'aquesta manera $G_c = (V_c, E_c)$ és el graf de components connexes i el problema es redueix a determinar si G_c, el qual és acíclic és semiconnex (si fos cíclic, existirien $u, v \in V_c$ tals que estarien en la mateixa classe d'equivalència !!!).

La idea de l'algorisme es basa en fer un DFS (arrelant l'arbre del recorregut en un node $v \in V_c$ qualsevol) i adonar-nos de que un graf dirigit acíclic serà semiconnex si i només si $\forall u \in V_c\ , \exists C(u, v)$ o bé $\exists C(v, u)$. Notem que, com és acíclic, si existeix $C(u, v)$ no pot existir $C(v, u)$.

Per tant, això motiva fer un DFS pel fet que d'aquesta manera determinarem tots els camins de l'arrel v a un altre node i posteriorment, amb el mateix DFS, determinarem els camins de la resta de nodes (no visitats) amb v.

Nota: Donat un arbre de recorregut, podem determinar si és semiconnex pels següents motius:

- Si tot node només té un fill, segur que és semiconnex
- Si hi ha algun node que té més d'un fill, aleshores serà semiconnex si i només si per cada ramificació $i + 1$ d'aquell node, la primera fulla apunta al primer fill del node de la ramificació i per tota i.

Justificació: La justificació es basa en el fet de que si no passés, aleshores no existiria cap camí entre un node de la ramificació $i + 1$ amb un de la ramificació i (el primer fill) (!!!)

Amb un exemple:

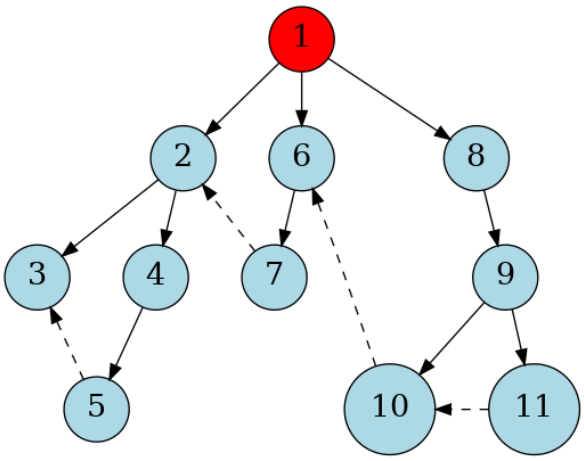

Figura 2: Exemple d'arbre semiconnex (les arestes discontínues són les arestes explicades al segon punt)

Per altra banda, si tenim dos arbres semiconnexos, la 'unió' ho serà si i només si, la primera fulla visitada pel segon arbre té una aresta cap el node pare del primer arbre.

Demostració: Suposarem que no i arribarem a contradicció. Suposem que no passa això, però que existeix un camí entre la primera fulla visitada pel segona arbre i l'arrel (al revés no pot passar perquè estaria en el primer DFS).

Això vol dir que $\exists v_1, ..., v_n$ tal que v_1= "node fulla del segon arbre", v_n = "arrel del primer arbre", i tal que $v_i \rightsquigarrow v_{i+1}$ $\forall i$. Per la definició de fulla, aquesta no té cap aresta dirigida cap a algun node no visitat. Per tant les úniques arestes que pot tenir són cap a nodes visitats. Com els únics nodes visitats són els del primer arbre del DFS i cap node de tal arbre té cap aresta cap a l'arrel del mateix (altrament es produiria un cicle), deduïm que l'únic camí entre la fulla i el pare ha de ser l'aresta directa (!!!).

Amb això podem definir l'algorisme formalment:

Algorithm 3 DFS(G)

Require: G graf acíclic
Ensure: G és semiconnex?

```
1: for v ∈ V(G) do
2:     visitat[v] := false
3:     ndfs[v] := 0
4: end for
5:
6: numDfs := 0
7: esSemiConnex := true
8: superPare := -1
9:
10: for v ∈ G and esSemiConnex do
11:     if not visitat[v] then
12:         SemiConnRec(G, v, superPare, esSemiConnex, ndfs, visitat)
13:         superPare := v
14:     end if
15: end for
16:
17: return esSemiConnex
```

Algorithm 4 SemiConnRec(G, v, superPare, esSemiConnex, ndfs, visistat)

Require: G graf acíclic

```
1: numDfs := numDfs + 1; ndfs[v] := numDfs
2: visistat[v] := true
3: esFulla := true
4: teAresta := false
5:
6: for w ∈ G.ADJACENT(v) do
7:     if not visitat[w] then
8:         esFulla := false
9:         SemiConneRec(G, w, superPare, esSemiConnex, ndfs, visitat)
10:        superPare := w
11:    else
12:        teAresta = teAresta or (w == superPare)
13:    end if
14: end for
15:
16: if esFulla and superPare != -1 then
17:     esSemiConnex := esSemiConnex and teAresta
18: end if
```

Problema 1.12 (clique max?)

Donat un graf no dirigit $G = (V, E)$ i un subconjunt de vèrtex V1, el subgraf induït per $V1$, $G[V1]$ té com a vèrtex $V1$ i con a arestes totes les arestes a E que connecten vèrtexs en $V1$. Un clique és un subgraf indiut per un conjunt C on tots els vèrtexs estan connectats entre ells. Considereu el següent algorisme de dividir-i-vèncer per al problema de trobar un clique en un graf no dirigit $G = (V, A)$.

CliqueDV(G)

1: Enumereu els vèrtexs V com $1, 2, \ldots, n$, on $n = |V|$
2: Si $n = 1$ tornar V
3: Dividir V en $V_1 = \{1, 2, \ldots, \lfloor n/2 \rfloor\}$ i $V_2 = \{\lfloor n/2 \rfloor + 1, \ldots, n\}$
4: Sigui $G_1 = G[V_1]$ i $G_2 = G[V_2]$
5: C_1= CliqueDV(G_1) i C_2= CliqueDV(G_2)
6: $C_1^+ = C_1$ i $C_2^+ = C_2$
7: **for** $u \in C_1$ **do**
8: **if** u està connectat a tots els vèrtexs a C_2^+ **then**
9: $C_2^+ = C_2^+ \cup \{u\}$
10: **for** $u \in C_2$ **do**
11: **if** u està connectat a tots els vèrtexs a C_1^+ **then**
12: $C_1^+ = C_1^+ \cup \{u\}$
13: Retorneu el més gran d'entre C_1^+ i C_2^+

Figura 3: Enter Caption

Contesteu les següents preguntes:

(a) **Demostreu que l'algorisme CliqueDV sempre retorna un subgraf de G que és un clique.**
Cas base ($n = 1$):
Clarament, si $n = 1$, *CliqueDV(G)* retorna V que és una clique.
Pas inductiu:
Suposem que *CliqueDV(G)* retorna una clique amb $k < n$ vèrtexos. Vegem que retorna una clique amb n vèrtexos. Notem que C_1 i C_2 són cliques per hipòtesi, ja que el nombre de vèrtexos de G_1 i G_2 és $\frac{n}{2}$. Per altra banda, si fem $C_1^+ = C_1$ i $C_2^+ = C_2$, en els bucles, mirem si hi ha algun vèrtex de C_1 que estigui connectat a tots els vèrtexos de C_2^+ i en tal cas l'afegim, cosa que conserva el fet de que C_2^+ sigui una clique (idem per C_1^+). Finalment, es retorna el Ci^+ que tingui grau més gran i per tant, que és una clique, ja que totes les operacions que hem fet conserven el fet de que ho siguin.

Per tant hem demostrat que *CliqueDV(G)* retorna una clique de G.

(b) **Doneu una expressió asimptòtica del nombre de passos de l'algorisme CliqueDV.**
Notem que l'algorisme compleix la recurrència $T(n) = 2T(n/2) + \Theta(n^2)$, ja que es fan dues crides recursives al mateix algorisme amb mides $n/2$ i els bucles tenen

ordre n^2, ja que es fa un bucle d'ordre n i el *if* també és d'ordre n . Resolent la recurrència amb el teorema master veiem que $T(n) = \Theta(n^2)$.

(c) **Doneu un exemple d'un graf G on l'algorisme CliqueDV retorna un clique que no és de grandària màxima.**
Un exemple de graf on l'algorisme no retorna una clique de grandària màxima és el següent:

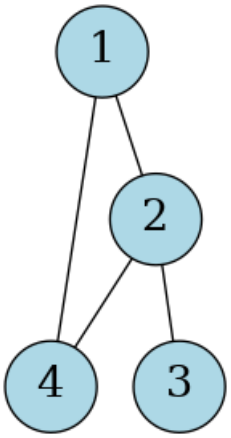

Figura 4: Exemple de graf on l'algorisme no retorna una clique màxima

(d) **Creieu que és fàcil modificar CliqueDV de manera que sempre done el clique màxim, sense incrementar el temps pitjor de l'algorisme? Expliqueu la vostra resposta**
Notem que el problema de clique màxima és un problema NP-complet. Per tant, si es pogués P = NP i hauríem resolt el problema del mileni.

Per altra banda, si suposem que sí que es pot modificar, fent hipòtesi d'inducció *CliqueDV* retornaria la clique màxima de la primera meitat dels nodes i de la segona (tal com es veu en la línia 5 del codi). No obstant, la clique màxima no depèn de les cliques màximes de la meitat de nodes, ja que el fet de ser clique màxima és global.

Extra 1

Donat un graf $G = (V, E)$ direm que aquest graf és connex:

$$G \text{ és connex} \iff \forall u, v \in V \mid v \neq u \; \exists \; C(u, v)$$

On $C(u, v)$ és un camí entre u i v que definirem de la següent forma:

$$C(u, v) = \{u_0, u_1, \; \ldots \; , u_n \in V \mid u_0 = u, \; u_n = v, (u_i, u_{i+1}) \in E \;\; \forall i \in [n-1]\}$$

Extra 2

Donat un graf $G = (V, E)$ direm que aquest graf és complet:

$$G \text{ és complet} \iff \forall v, u \in V \mid u \neq v \; (v, u) \in E$$

Extra 3

Volem veure que tot arbre binari complet amb n nodes té un total de $\frac{n+1}{2}$ fulles, entenent com a fulles aquells nodes que no són pares de cap altre node. Notem que només existeixen arbres binaris complets amb n senar ($n = 1 + 2 * pares$). Ho veurem per inducció:

Cas base ($n = 1$):

En aquest cas només tenim $n = 1$ nodes, i per tant, només tenim $\frac{n+1}{2} = 1$ fulles, per tant, en el cas base es verifica la propietat que volem demostrar.

Pas inductiu:

Suposem que tenim un arbre binari complet amb $n-1$ nodes i suposem que la propietat que volem demostrar és certa, aleshores tenim un total de $\frac{n}{2}$ fulles. Ara, per a construir un arbre binari complet de $n + 1$ nodes hem d'afegir dos fills a qualsevol d'aquestes fulles (el cas d'un arbre amb n nodes no té sentit ja que només existeixen els arbres binaris complets amb n nodes si n és imparell, per tant si $n - 1$ és imparell, el següent arbre binari complet tindrà $n + 1$ nodes). Aleshores, la fulla que escollim deixa de ser-ho i passem a tenir dues noves fulles (els fills d'aquesta fulla). D'aquesta forma el còmput global de fulles queda:

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

Per tant hem vist que:

$$\forall B_n \text{ arbre binari complet amb } n \text{ nodes } (n \equiv 1 \; (mod 2)) \implies \#Fulles(B_n) = \frac{n+1}{2}$$

Miquel Pere Baztán Grau
Bernat Dosrius Lleonart
Xavier Momplet Gil
Emma Ventura Font

Universitat Politècnica de Catalunya