


## Problemes 1

- 1.1.  Suposem que tenim un vector  $A$  amb  $n$  nombres enters diferents, amb la propietat: existeix un únic índex  $p$  tal que els valors  $A[1 \dots p]$  estan en ordre creixent i els valors  $A[p \dots n]$  estan en ordre decreixent. Per exemple, en la següent vector tenim  $n = 10$  i  $p = 4$ :

$$A = (2, 5, 12, 17, 15, 10, 9, 4, 3, 1)$$

Dissenyeu un algorisme eficient per trobar  $p$  donada una matriu  $A$  amb la propietat anterior.

**Una solució:** L'algorisme recursiu es descriu en l'Algorisme FINDPEAK. Donada la matriu  $A$ , la resposta s'obté amb una crida amb  $i = 1$  i  $j = n$ . L'algorisme és una cerca binària, en cada pas, comparem els dos elements intermedis i veurem si estem en la part creixent o decreixent. El cas base ressol el problema d'obtenir la posició del màxim, però per a una entrada amb mida constant.

```
function FINDPEAK( $A, i, j$ )  
     $n = j - i + 1$   
    if  $n \leq 5$  then  
        return POSMAX( $A, i, j$ )  
     $k = (i + j) / 2$   
    if  $A[k] < A[k + 1]$  then  
        return FINDPEAK( $A, k + 1, j$ )  
    else  
        return FINDPEAK( $A, i, k$ )
```

**Correctesa:** Volem trobar l'índex  $p$ . Si  $A[k] < A[k + 1]$ , sabem que  $A[i] < \dots < A[k]$  per  $i < k$  i podem prescindir de forma segura els elements  $A[i \dots k]$ . De la mateixa manera, si  $A[k] > A[k + 1]$ , sabem que  $A[k + 1] > \dots > A[j]$  per  $j > k + 1$  i podem descartar amb seguretat els elements  $A[k + 1 \dots j]$ . La posició de  $p$  coincideix amb la del valor màxim al vector, per tant el cas base és correcte.

**Cost temporal:** El cas base té cost constant. A cada pas, es redueix la mida del problema a la meitat i, a més, el cost de les operacions és constant. Així, tenim la recurrència  $T(n) = T(n/2) + c$  per a alguna constant  $c$ . Sabem que, com a la cerca binària,  $T(n) = O(\log n)$ .

1.2. Un vector  $A[n]$  conté tots els enters entre 0 i  $n$  excepte un.

- (a) Dissenyeu un algorisme que, utilitzant un vector auxiliar  $B[n + 1]$ , detecti l'enter que no és a  $A$ , i ho faci en  $O(n)$  passos.
- (b) Supposem ara que  $n = 2^k - 1$  per a  $k \in \mathbb{N}$  i que els enters a  $A$  venen donats per la seva representació binària. En aquest cas, l'accés a cada enter no és constant, i llegir qualsevol enter té un cost  $\lg n$ . L'única operació que podem fer en temps constant es “recuperar” el  $j$ -èsim bit de l'enter a  $A[i]$ . Dissenyeu un algorisme que, utilitzant la representació binària per a cada enter, trobi l'enter que no és a  $A$  en  $O(n)$  passos.



1.4. Per a cadascú dels algorismes, digueu quin és el temps en cas pitjor, quan l'entrada és un enter positiu  $n > 0$ .

- En el pitjor cas fem  $2^k i$  iteracions  $\rightarrow 2^k i = n \Rightarrow 2^k = n/i \Rightarrow k = \log(n/i)$
- (a)  $\text{for } i = 1 \text{ to } n \text{ do}$   
 $\quad j = i$   
 $\quad \text{while } j < n \text{ do}$   
 $\quad \quad j = 2 * j$   
 $T(n) = n + \sum_{i=1}^n \log(n/i) = \log\left(\prod_{i=1}^n (n/i)\right) = \log(n^n/n!) = n \log n - \log n! \sim n \log n - \log(\sqrt{2\pi n} (n/e)^n) =$   
 $= n \log n - \log(\sqrt{2\pi n}) - \log((n/e)^n) = n \log n - \log(\sqrt{2\pi n}) - n \log n + n \log e = n \log e - \log(\sqrt{2\pi n}) =$   
 $= O(n)$
- (b)  $\text{for } i = 1 \text{ to } n \text{ do}$   
 $\quad j = n$   
 $\quad \text{while } i * i < j \text{ do}$   
 $\quad \quad j = j - 1$   
 $T(n) = n + \sum_{i=1}^{\sqrt{n}} (n - i^2) = n + \sum_{i=1}^{\sqrt{n}} n - \sum_{i=1}^{\sqrt{n}} i^2 = n + n\sqrt{n} - \frac{\sqrt{n}(\sqrt{n}-1)(2\sqrt{n}+1)}{6}$   
 $\text{El bucle intern s'executa quan } i < \sqrt{n} \text{ per cada } i \text{ concreta fa } (n - i^2) \text{ iteracions}$
- (c)  $\text{for } i = 1 \text{ to } n \text{ do}$   
 $\quad j = 2$   
 $\quad \text{while } j < i \text{ do}$   
 $\quad \quad j = j * j$   
 $j < i \rightarrow 2 < n, 4 < n, 16 < n, 256 < n$   
 $2^1 = 2^{2^0} \quad 2^2 = 2^{2^1} \quad 2^4 = 2^{2^2} \quad 2^8 = 2^{2^3} \rightarrow 2^{2^k} < i \rightarrow \log \log i = k$   
 $2^{2^k} = i$   
 $T(n) = n + \sum_{i=1}^n \log \log i \leq n + \sum_{i=1}^n \log \log n \rightarrow O(n \log \log n)$
- (d)  $i = 2$   
 $\text{while } (i * i < n) \text{ i } (n \bmod i \neq 0) \text{ do}$   
 $\quad i = i + 1$

Hirem el pitjor cas:

$n$  es primer  $\Leftrightarrow n \bmod i \neq 0 \quad \forall i \in [2, n-1]$

$i * i < n \Rightarrow \sqrt{n}$  iteracions  $\rightarrow \text{cost } O(\sqrt{n})$

- 1.5. El problema 2SAT té com a entrada un conjunt de clàusules, on cada clàusula és la disjunció (OR) de dos literals (un literal és una variable booleana o la negació d'una variable booleana). Volem trobar una manera d'assignar un valor cert o fals a cadascuna de les variables perquè totes les clàusules es satisfuguin - és a dir, hi hagi al menys almenys un literal cert a cada clàusula. Per exemple, aquí teniu una instància de 2SAT:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_4).$$

Aquesta instància és satisfactible: fent  $x_1, x_2, x_3, x_4$  cert, fals, fals i cert, respectivament.

El propòsit d'aquest problema és conduir-vos a una manera de resoldre 2SAT de manera eficient reduint-ho al problema de trobar les components connexes fortes d'un graf dirigit. Donada una instància  $F$  de 2SAT amb  $n$  variables i  $m$  clàusules, construïm un graf dirigit  $G_F = (V, E)$  de la següent manera.

- $G_F$  té  $2n$  nodes, un per a cada variable i un per a la seva negació.
- $G_F$  té  $2m$  arcs: per a cada clàusula  $(\alpha \vee \beta)$  de  $F$  (on  $\alpha, \beta$  són literals),  $G_F$  té un arc des de la negació d' $\alpha$  a  $\beta$ , i un de la negació de  $\beta$  a  $\alpha$ .

Tingueu en compte que la clàusula  $(\alpha \vee \beta)$  és equivalent a qualsevol de les implicacions  $\neg\alpha \Rightarrow \beta$  o  $\neg\beta \Rightarrow \alpha$ . En aquest sentit,  $G_F$  representa totes les implicacions directes en  $F$ .

- Realitzeu aquesta construcció per a la instància de 2SAT indicada amunt.
- Demostreu que si  $G_F$  té una component connexa forta que conté  $x$  i  $\neg x$  per a alguna variable  $x$ , llavors no és satisfactible.
- Ara demostreu la inversa: és a dir, que si no hi ha cap component connexa forta que contingui tant un literal com la seva negació, llavors la instància ha de ser satisfactible.
- A la vista del resultat previ, hi ha un algorisme de temps lineal per resoldre 2SAT?

**Algorisme per a comprovar si la fórmula  $F$  és satisfactible.**

1. Construir el graf d'implicacions  $G_F$  de la instància d'entrada i trobar les seves components fortament connexes.

2. Mirar si alguna de les components fortament connexes conté una variable i la seva negació. Si és el cas, aleshores la fórmula és insatisfactible i acabem. Altrament serà satisfactible.

Aquesta darrera propietat és suficient per decidir si la fórmula  $F$  és satisfactible i la podem, doncs, utilitzar la seva comprovació per dissenyar l'algorisme que decideix la satisfactibilitat. Demostració:

$\Rightarrow$  Si a una component connexa forta hi són  $x$  i  $x$ , sabem que al graf d'implicacions  $G_F$  hi ha camins de  $x$  a  $x$ , i de  $x$  a  $x$ . Tenint en compte que els arcs de  $G_F$  es correspon amb implicacions lògiques correctes, deduïm que  $x \Rightarrow x$  i  $x \Rightarrow x$ ; deduïm, doncs, una contradicció de  $F$  i, per tant,  $F$  no és satisfactible.

$\Leftarrow$  Hem de veure que, si no hi ha cap component connexa forta de  $G_F$  que contingui tant un literal com la seva negació, aleshores la fórmula  $F$  és satisfactible. Demostrarem el recíproc: suposem que la fórmula  $F$  és no satisfactible; això implica que podem deduir una contradicció de la forma  $x \Leftarrow x$  per a alguna variable  $x$ . Per tant,  $x$  i  $x$  pertanyen a la mateixa component connexa forta.

En cas que la fórmula sigui satisfactible, volem també calcular una de les assignacions a les variables que produeixi aquesta satisfactibilitat. L'algorisme serà el següent:

1. Construir el graf condensat del graf d'implicacions (el graf que té un vèrtex per cada component fortament connexa, i una aresta de la component  $i$  a la component  $j$  sempre que el graf original contingui una arc  $(u, v)$  tal que  $u$  pertany a la component connexa  $i$  i  $v$  pertany a la component connexa  $j$ ).

2. Ordenar topològicament el graf condensat (noteu que el graf condensat és un graf dirigit acíclic i, per tant, es pot). Podem fer servir l'algorisme de Kosaraju o Tarjan, però ens va millor aquest últim perquè ja genera l'ordre topològic invers.

Una vegada tenim l'ordenació topològica inversa, podem començar a assignar valors als literals. L'assignació de valors la farem de la següent manera: Per a cada component en l'ordre topològic invers, si les seves variables no tenen ja una assignació a TRUE, posar tots els literals a la component a TRUE. Això fa que a tots els literals complementaris d'aquestes a altres components, se'ls hi assigni FALSE. Quan un literal es posa a TRUE, tots els literals que són accessibles a partir d'ell via una cadena d'implicacions també es posaran a TRUE. De manera simètrica, quan un literal és assignat a FALSE, tots els literals que porten a ell via una cadena d'implicacions s'assignaran també a FALSE. El cost d'aquest algorisme és  $O(n+m)$ , on  $n = |V|$  i  $m = |E|$  de  $G_F$ .

- 1.6. 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  $v$ . 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  $O(n)$ , on  $n$  és el nombre de vèrtexs.

Si  $\textcircled{u} \leftarrow \textcircled{v} \rightarrow v$  no pot ser celebritat

Si  $\textcircled{u} \rightarrow \textcircled{v} \rightarrow u$  no pot ser celebritat

Hi ha com a mínim  $n-1$  nodes que no són celebritats i com a màxim 1 node celebritat, per tant, si anem consultant parell de vèrtex sempre podem descartar un dels nodes com a candidat a celebritat. Llavors, amb  $n-1$  consultes hi ha prou per obtenir un candidat a celebritat i per comprovar si el candidat és, efectivament, una celebritat, s'ha de fer  $2(n-1)$  consultes.

Codi:

```
i=0; j=1; k=2;
while (k+1) ≤ n do
  if H[i][j] then i=k;
  else j=k;
  k++;
Celeb = TRUE;
k=0
while (k < n and celeb)
  for l=0 to n and celeb do
    celeb = H[l][k];
    if (k ≠ l and celeb) celeb = H[k][l];

if celeb then k + " is celebrity";
else "There is no celebrity";
```

1.7. Llisteu les següents funcions en ordre *creixent*, és a dir, si l'ordre és  $f_1; f_2; \dots$ , aleshores  $f_2 = \Omega(f_1); f_3 = \Omega(f_2)$ ; etc.

$$(\log n)^{100}, n \log n, 3^n, \frac{n^2}{\log n}, n2^n, 0.99^n, n^3, \sqrt{n}.$$

1.8. Digueu si cadascuna de les afirmacions següents són certes o falses (i per què).

- (a) Asimptòticament  $(1 + o(1))^{\omega(1)} = 1$
- (b) Si  $f(n) = (n + 2)n/2$  aleshores  $f(n) \in \Theta(n^2)$ .
- (c) Si  $f(n) = (n + 2)n/2$  aleshores  $f(n) \in \Theta(n^3)$ .
- (d)  $n^{1.1} \in O(n(\lg n)^2)$
- (e)  $n^{0.01} \in \omega((\lg n)^2)$




1.9. Digueu si la següent demostració de

$$\sum_{k=1}^n k = O(n)$$

és certa o no (i justifiqueu la vostra resposta).

**Demostració:** Per a  $k = 1$ , tenim  $\sum_{k=1}^1 k = 1 = O(1)$ . Per hipòtesi inductiva, assumim  $\sum_{k=1}^n k = O(n)$ , per a una certa  $n > 1$ . Llavors, per a  $n + 1$  tenim

$$\sum_{k=1}^{n+1} k = n + 1 + \sum_{k=1}^n k = n + 1 + O(n) = O(n).$$

- 1.10.  Tenim un conjunt de robots que es mouen en un edifici, cadascun d'ells és equipat amb un transmissor de ràdio. El robot pot utilitzar el transmissor per comunicar-se amb una estació base. No obstant això, si els robots són massa a prop un de l'altre hi ha problemes amb la interferència entre els transmissors. Volem trobar un pla de moviment dels robots, de manera que puguin procedir al seu destí final, sense perdre mai el contacte amb l'estació base.

Podem modelar aquest problema de la següent manera. Se'ns dona un graf  $G = (V, E)$  que representa el plànol d'un edifici, hi ha dos robots que inicialment es troben en els nodes  $a$  i  $b$ . El robot en el node  $a$  vol viatjar a la posició  $c$ , i el robot en el node  $b$  vol viatjar a la posició  $d$ . Això s'aconsegueix per mitjà d'una planificació: a cada pas de temps, el programa especifica que un dels robots es mou travessant una aresta. Al final de la planificació, els dos robots han d'estar en les seves destinacions finals.

Una planificació és *lliure* d'interferència si no hi ha un punt de temps en el qual els dos robots ocupen nodes que es troben a distància menor de  $r$ , per a un valor determinat del paràmetre  $r$ .

Doneu un algorisme de temps polinomial que decideixi si hi ha una planificació lliure donats, el graf, les posicions inicials i finals dels robots i el valor de  $r$ .

**Una solució.** Per resoldre el problema considerarem l'espai de configuracions on els robots es poden moure. És a dir, el conjunt de parells de posicions que estan a distància més gran o igual que  $r$ :

$$C = \{(u, v) \mid u, v \in V \text{ i } d(u, v) \geq r\}$$

Podem considerar la relació entre configuracions definida pels moviments permesos. Així tenim

$$M = \{((u, v), (u', v')) \mid (u, v), (u', v') \in C \text{ i } ((u = u' \text{ i } (v, v') \in E) \text{ o } (v = v' \text{ i } (u, u') \in E))\}.$$

A l'espai de configuracions podem considerar el graf  $\mathcal{G} = (C, M)$  on dos configuracions son veïnes si i només si un del robots pot canviar de posició sense interferir amb la posició de l'altre.

Els robots són inicialment a la configuració  $(a, b)$  i s'han de desplaçar amb moviments vàlids fins a la configuració  $(c, d)$ . Això serà possible únicament si hi ha un camí de  $(a, b)$  a  $(c, d)$ . D'acord amb el raonament anterior tenim que comprovar hi ha un camí entre dos vèrtexs a  $\mathcal{G}$ . Podem detectar-ho amb un BFS en tems  $O(|C| + |M|)$ .

Per calcular el cost hem de tenir en compte la mida de l'entrada. Si  $G = (V, E)$  i  $n = |V|$  i  $m = |E|$ , tenim  $|C| \leq n^2$  i  $|M| \leq 2m$ . Suposant que ens donen  $G$  mitjançant llistes d'adjacència la mida de l'entrada és  $n + m$ . Construir una descripció de  $\mathcal{G}$  mitjançant llistes d'adjacència té cost  $O(n^2 + m)$ . Fer un BFS sobre  $\mathcal{G}$  té cost  $O(n^2 + m)$ . El cost total es  $O(n^2 + m)$  però  $m \leq n^2$ . Llavors l'algorisme proposat té cost  $O(n^2)$ .

- 1.11. El quadrat d'un graf  $G = (V, E)$  és un altre graf  $G' = (V, E')$  on  $E' = \{(u, v) \mid \exists w \in V, (u, w) \in E \wedge (w, v) \in E\}$ . Dissenyeu i analitzeu un algorisme que, donat un graf representat amb una matriu d'adjacència, calculi el seu quadrat. Feu el mateix amb un graf representat amb llistes d'adjacència.

Answer: The edge  $(u, w)$  exists in the square of a graph  $G$  if there exists a vertex  $v$  such that the edges  $(u, v)$  and  $(v, w)$  exist in  $G$ . To calculate this efficiently from an adjacency matrix, we notice that this condition is exactly what we get when we square the matrix. The cell  $M^2(u, w) = \sum_v M(u, v) \cdot M(v, w)$  when we multiply two matrices. So, if we represent edges present in  $G$  with ones and all other entries as zeroes, we will get the square of the matrix with zeroes when edges aren't in the graph  $G$  and positive integers representing the number of paths of length exactly two for edges that are in  $G$ . Using Strassen's algorithm or other more sophisticated matrix multiplication algorithms, we can compute this in  $O(V^2.376)$ . Using adjacency lists, we need to loop over all edges in the graph  $G$ . For each edge  $(u, v)$ , we will look at the adjacency list of  $v$  for all edges  $(v, w)$  and add the edge  $(u, w)$  to the adjacency lists for  $G$ . The maximum number of edges in the adjacency list for  $v$  is  $V$ , so the total running time is  $O(VE)$ . This assumes that we can add and resolve conflicts when inserting into the adjacency lists for  $G$  in constant time. We can do this by having hash tables for each vertex instead of linked lists.

1.12. 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  $O(n + m)$ .

**Algorisme:**

1- Algorisme de Kosaraju-Sharir  $\rightarrow$  cost  $O(n + m)$

$\rightarrow$  Obtenim un nou graf  $G^{scc}$  que està ordenat topològicament  $O = \{v_1, v_2, \dots, v_k\} \in G^{scc}$

2-  $\forall v_i, v_{i+1} \in O : (v_i, v_{i+1}) \in (G^{scc}) \Leftrightarrow G_{\text{semiconnex}} \rightarrow \text{Cost } O(n + m)$

De manera que per tot parell de nodes consecutius retornats per l'algorisme de Kosaraju, on els nodes es retornen ordenats topològicament, existeix una aresta que els uneix, podem assegurar que el graf es semiconnex.

Un exemple de graf connex ordenat topològicament:



Deixem un exemple en cas de no complir el que hem comentat anteriorment:




Com podem veure no hi ha un camí que vagi des del node  $v_1$  al  $v_3, v_4$  o  $v_5$  al revés un camí que vagi des de  $v_3, v_4$  o  $v_5$  al  $v_2$ .

1.13. Definim els  $k$ -mers com les subcadenaes de DNA, amb grandària  $k$ . Per tant, per a un valor donat  $k$  podem assumir que tenim una base de dades amb tots els  $4^k$   $k$ -mers. Una manera utilitzada en l'experimentació clínica per a identificar noves seqüències de DNA, consisteix a agafar mostres aleatòries de una cadena i determinar quins  $k$ -mers conté, on els  $k$ -mers es poden solapar. A partir d'aquest procés, podem reconstruir tota la seqüència de DNA.

Formalment, donada una cadena  $w \in \{A, C, T, G\}^*$ , i un enter  $k$ , sigui  $S(w)$  el multi-conjunt de tots els  $k$ -mers que apareixen a  $w$ . Notem que  $|S(w)| = |w| - k + 1$ . El problema consisteix en, donat un multi-conjunt  $C$  de  $k$ -mers, trobar la cadena de DNA  $w$  tal que  $S(w) = C$ .

- a Demostreu que hi ha una reducció polinòmica d'aquest problema al problema del camí Hamiltonià, que és NPC (utilitzeu els  $k$ -mers com a vèrtexs i el solapament entre  $k$ -mers com condició d'existència d'arestes).
- b Demostreu que hi ha una reducció d'aquest problema al problema del camí Eulerià, que és a P (aquest cop, utilitzeu  $k$ -mers com a arestes dirigides).
- c Vol dir això que aquest problema és a P i a NPC, i per tant  $P=NP$ ?

- 1.14.  El Professor JD ha corregit els exàmens finals del curs, de cara a tenir una distribució maca de les notes finals decideix formar  $k$  grups, cada grup amb el mateix nombre d'alumnes, i donar la mateixa nota a tots els alumnes que són al mateix grup. La condició més important és que qualsevol dels alumnes al grup  $i$  han de tenir nota d'examen superior a qualsevol alumne d'un grup inferior (grups de 1 fins a  $i - 1$ ). L'ordre dintre de cadascun dels grups es irrelevant. Dissenyeu un algorisme que donada una taula  $A$  no ordenada, que a cada registre conté la identificació d'un estudiant amb la seva notes d'examen, divideix  $A$  en els  $k$  grups, amb les propietat descrita a dalt. El vostre algorisme ha de funcionar en temps  $O(n \lg k)$ . Al vostre anàlisis podeu suposar que  $n$  és múltiple de  $k$  i  $k$  és una potencia de 2.

**Una solució:** Sigui AGRUPAR l'algorisme recursiu que, té com a entrada una taula de alumnes-notes  $N$  i dos enters  $\ell$  i  $t$ , i fa el següent:

- Mentre  $\ell \neq 1$  troba la mediana de  $A$  i fa una partició al seu voltant en temps  $O(|N|)$ .
- Considerem la sub-aula  $N_e$  esquerra i la sub-aula dreta  $N_d$ .
- Cridem recursivament  
AGRUPAR( $N_e, \ell/2, 2t$ ) i AGRUPAR( $N_d, \ell/2, 2t + 1$ ).
- Quan  $\ell = 1$ , la taula constitueix la partició  $t$ .

La crida inicial la farem amb  $N$ ,  $\ell = k$  i  $t = 0$ . La correctesa ve de com particionem els elements. Sempre tenim dos meitats i els elements a  $N_e$  són més petits que la mediana i els elements a  $N_d$  són més grans o iguals que la mediana. Aconseguirem  $\ell = 1$  després de  $\lg k$  iteracions, en aquell moment la taula té  $n/k$  elements. La variable  $t$  comptabilitza l'ordre de las crides. Al primer nivell tenim només una taula i  $t = 0$ . Al segon tindrem dos taules, la de l'esquerra etiquetada amb 0 i la de la dreta amb 1. Al següent nivell, tindrem 0,1,2,3 (e-e,d,d-e,d-d). Llavors  $t$  comptabilitza l'ordre correcte de les particions per garantir la propietat requerida.

El cost de l'algorisme és  $T(n, k) = 2T(n/2, k/2) + \Theta(n)$  amb  $T(n, 1) = \Theta(1)$ , per a tot  $n$ . Desplegant la recursió tenim

$$\begin{aligned} T(n, k) &= 2T(n/2, k/2) + cn = 4T(n/4, k/4) + 2c(n/2) + cn \\ &= 4T(n/4, k/4) + 2cn = k + cn \lg k. \end{aligned}$$

llavors,  $T(n) = \Theta(n \lg k)$ .

1.15. Resoleu les següents recurrències

(a)  $T(n) = 16T(n/2) + \binom{n}{3} \lg^4 n$

(b)  $T(n) = 5T(n/2) + \sqrt{n}$

(c)  $T(n) = 2T(n/4) + 6.046\sqrt{n}$

(d)  $T(n) = 2T(n/2) + \frac{n}{\lg n}$

(e)  $T(n) = T(n - 10) + n$

- 1.16. Donat un graf no dirigit  $G = (V, E)$  i un subconjunt de vèrtex  $V_1$ , el subgraf induït per  $V_1$ ,  $G[V_1]$  té com a vèrtex  $V_1$  i con a arestes totes les arestes a  $E$  que connecten vèrtexs en  $V_1$ . Un clique és un subgraf induït 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  $G = (V, A)$ .

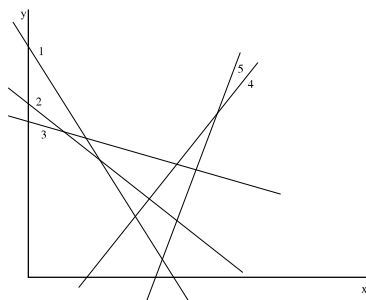
```
CliqueDV( $G$ )
1: Enumereu els vèrtexs  $V$  com  $1, 2, \dots, n$ , on  $n = |V|$ 
2: Si  $n = 1$  tornar  $V$ 
3: Dividir  $V$  en  $V_1 = \{1, 2, \dots, \lfloor n/2 \rfloor\}$  i  $V_2 = \{\lfloor n/2 \rfloor + 1, \dots, n\}$ 
4: Sigui  $G_1 = G[V_1]$  i  $G_2 = G[V_2]$ 
5:  $C_1 = \text{CliqueDV}(G_1)$  i  $C_2 = \text{CliqueDV}(G_2)$ 
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  $C_2^+$ 
```

Contesteu les següents preguntes:

- (a) Demostreu que l'algorisme **CliqueDV** sempre retorna un subgraf de  $G$  que és un clique.
- (b) Doneu una expressió asimptòtica del nombre de passos de l'algorisme **CliqueDV**.
- (c) Doneu un exemple d'un graf  $G$  on l'algorisme **CliqueDV** retorna un clique que no és de grandària 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.



- 1.17. El problema de l'eliminació de superfícies ocultes és un problema important en informàtica gràfica. Si des de la teva perspectiva, en Pepet està davant d'en Ramonet, podràs veure en Pepet però no en Ramonet. Considereu el següent problema, restringit al pla. Us donen  $n$  rectes no verticals al pla,  $L_1, \dots, L_n$ , on la recta  $L_i$  ve especificada per l'equació  $y = a_i x + b_i$ . Assumim, que no hi han tres rectes que es creuen exactament al mateix punt. Direm que  $L_i$  és *maximal* en  $x_0$  de la coordenada  $x$ , si per qualsevol  $1 \leq j \leq n$  amb  $j \neq i$  tenim que  $a_i x_0 + b_i > a_j x_0 + b_j$ . Direm que  $L_i$  és *visible* si té algun punt maximal.



Donat com a entrada un conjunt de  $n$  rectes  $\mathcal{L} = \{L_1, \dots, L_n\}$ , doneu un algorisme que, en temps  $O(n \lg n)$ , torne las rectes no visibles. A la figura de sobre teniu un exemple amb  $\mathcal{L} = \{1, 2, 3, 4, 5\}$ . Totes les rectes excepte la 2 són visibles (considerem rectes infinites).

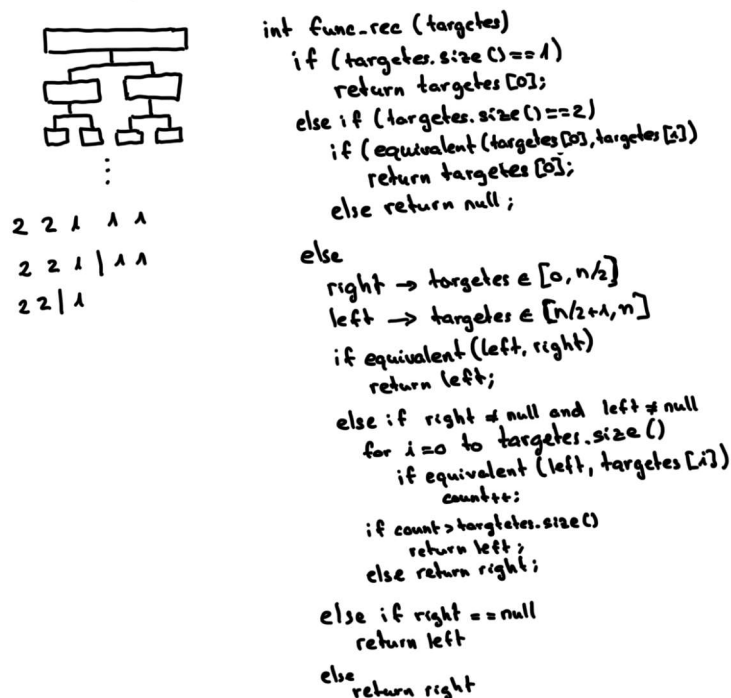
- 1.18. Supposeu que sou consultors per a un banc que està molt amoïnat amb el tema de la detecció de fraus. El banc ha confiscat  $n$  targetes de crèdit que se sospita han estat utilitzades en negocis fraudulents. Cada targeta conté una banda magnètica amb dades encriptades, entre elles el número del compte bancari on es carrega la targeta. Cada targeta es carrega a un únic compte bancari, però un mateix compte pot tenir moltes targetes. Direm que dues targetes són *equivalents* si corresponen al mateix compte.

És molt difícil de llegir directament el número de compte d'una targeta intel·ligent, però el banc té una tecnologia que donades dues targetes permet determinar si són equivalents.

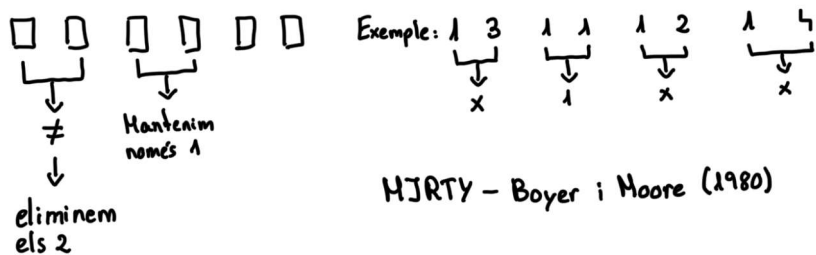
La qüestió que el banc vol resoldre és la següent: donades les  $n$  targetes, volen conèixer si hi ha un conjunt on més de  $\lceil n/2 \rceil$  targetes són totes equivalents entre si. Suposem que les úniques operacions possibles que pot fer amb les targetes és connectar-les de dues en dues, al sistema que comprova si són equivalents.

Doneu un algorisme que resolgui el problema utilitzant només  $O(n \lg n)$  comprovacions d'equivalència entre targetes. Sabríeu com fer-ho en temps lineal?

$O(n \log n) \rightarrow$  forma: algorisme recursiu amb 2 crides recursives



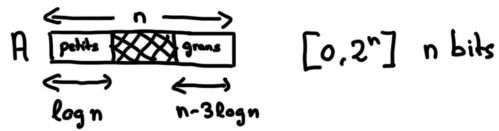
$O(n)$



MAJORITY - Boyer i Moore (1980)

→ Seguirà sent majoritari perquè eliminem 1 majoritari i 1 no majoritari

- 1.19. Donada una taula  $A$  amb  $n$  registres, on cada registre conté un enter de valor entre 0 i  $2^n$ , i els continguts de la taula estan desordenats, dissenyeu un algorisme lineal per a obtenir una llista ordenada dels elements a  $A$  que tenen valor més gran que els  $\log n$  elements més petits a  $A$ , i al mateix temps, tenen valor més petit que els  $n - 3 \log n$  elements més grans a  $A$ .



1- Quickselect del  $3 \log n$ -èsim  $\rightarrow$  cost  $O(n)$

$\rightarrow$  permet seleccionar l'element  $i$ -èsim més petit d'un conjunt  $n$   
 A més ordena els  $i$ -èsims elements més petits

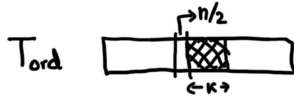
2- Quickselect del  $\log n$ -èsim  $\rightarrow$  cost  $O(n)$

3- Apliquem un algorisme d'ordenació dels elements entre  $\log n$  i  $n - 3 \log n \rightarrow$  cost  $O(n \log n)$   
 $\rightarrow O(\log n \times \log \log n) \leq O(n) \Rightarrow$

Cost :  $O(n)$

- 1.20. Tenim un taula  $T$  amb  $n$  claus (no necessàriament numèriques) que pertanyen a un conjunt totalment ordenat. Doneu un algorisme  $O(n + k \log k)$  per a ordenar els  $k$  elements a  $T$  que són els més petits d'entre els més grans que la mediana de les claus a  $T$ .

Conjunt totalment ordenat  $\rightarrow$  Conjunt que es pot ordenar (comparant elements)  
 $\rightarrow$  No vol dir que el vector estigui ordenat




- 1- Quickselect ( $T_1, n/2$ )  $\rightarrow O(n)$
- 2- Quickselect ( $T_n, n/2+k$ )  $\rightarrow O(n)$
- 3- Algorisme d'ordenació  $O(n \log n) \rightarrow O(k \log k)$

1.21. Com ordenar eficientment elements de longitud variable:

- (a) Donada una taula d'enters, on els enters emmagatzemats poden tenir diferent nombre de dígit. Però sabem que el nombre total de dígit sobre tots els enters de la matriu és  $n$ . Mostreu com ordenar la matriu en  $O(n)$  passos.
- (b) Se us proporciona una sèrie de cadenes de caràcters, on les diferents cadenes poden tenir diferent nombre de caràcters. Com en al cas previ, el nombre total de caràcters sobre totes les cadenes és  $n$ . Mostreu com ordenar les cadenes en ordre alfabètic fent servir  $O(n)$  passos. (Tingueu en compte que l'ordre desitjat és l'ordre alfabètic estàndard, per exemple,  $a < ab < b$ .)

- 1.22. Hi ha un concurs de TV amb  $n$  participants on cada participant escull un enter entre 0 i 1000000. El premi és per als dos concursants que escullen els enters més propers. Dissenyeu un algorisme que, en temps lineal, li digui al presentador quins són els dos concursants guanyadors o l'indiqui que hi ha més d'un parell de concursants candidats a rebre el premi.


- 1.23.  Donat un vector  $A$  amb  $n$  elements, és possible posar en ordre creixent els  $\sqrt{n}$  elements més petits i fer-ho en  $O(n)$  passos?

**Una solució:** Seleccionar l'element  $\sqrt{n}$ -èsim i particionar al voltant, d'aquest element (cost  $O(n)$ ). Ordenar la part esquerra en  $O(\sqrt{n}^2)$ .

Alternativament, construir un min-heap en  $O(n)$  i extreure el mínim element  $\sqrt{n}$  cops, el nombre de passos és  $O(n + \sqrt{n} \lg n) = O(n)$ .

- 1.24. Tenim un conjunt de  $2n$  valors tots diferents. Una meitat dels valors estan emmagatzemats a una taula  $A$  i l'altra meitat a una taula  $B$ . Cadascuna de les dues taules està ordenada en ordre creixent i es troba a un ordinador diferent. No hi ha cap relació d'ordre entre els valors a  $A$  i els valors a  $B$ . Volem trobar la mediana del total dels  $2n$  valors. Doneu un algorisme amb cost  $O(\lg n)$  que permeti obtenir la mediana sota la hipòtesis que només podeu fer crides de la forma **Element**( $i, A$ ) o **Element**( $i, B$ ), per  $1 \leq i \leq n$ , que retornen l'element  $i$ -ésim a  $A$  o a  $B$ , respectivament (amb cost  $O(1)$ ).



- 1.25.  Tenim un vector  $A[1, \dots, n]$  no ordenat amb claus no necessàriament numèriques, però que pertanyen a un conjunt totalment ordenat de claus. Sigui  $x_i$  el  $2^i$ -èsim element més petit en  $A$ . Doneu un algorisme per calcular la suma dels valors  $x_i$ , per  $1 \leq 2^i \leq n$ , en  $\Theta(n)$  passos.

**Una solució**

Sea  $k$  el valor tal que  $2^k \leq n$  y  $2^{k+1} > n$ .  $k = O(\lg n)$  y se puede calcular junto con el valor  $2^k$  en tiempo  $O(\lg n)$ .

Nos piden calcular la suma de los elementos en posiciones,  $1, 2, 4, \dots, 2^k$  de un vector con  $n$  elementos.

Si utilizamos el algoritmo de selección para cada uno de los valores  $1, 2, \dots, 2^k$ , el coste del algoritmo será  $O(n \lg n)$ . Para reducir el coste necesitamos reducir el tamaño del vector en cada iteración.

El elemento  $x_i$ ,  $i < k$ , ocupa la posición  $2^i$  en  $A$  pero también ocupa la posición  $2^i$  entre los elementos que son menores que  $x_{i+1}$ . Además el número de elementos  $\leq x_{i+1}$  es  $2^{i+1}$ .

- Obtener  $k$ , coste  $O(\lg n)$ .
- Usando el algoritmo de selección encontrar  $x_k$ , el elemento  $2^k$ -ésimo de  $A$  con coste  $O(n)$ .
- Sea  $B_k$  el conjunto de elementos menores que  $x_k$ ,  $B_k$  tiene  $\leq 2^k$  elementos. Coste  $\Theta(n)$ .
- For  $i = k - 1$  to  $0$ 
  - Usando el algoritmo de selección encontrar  $x_i$ , el elemento  $2^i$ -ésimo de  $B$  con coste  $O(2^{i+1})$ .
  - Sea  $B_i$  el conjunto de elementos menores que  $x_i$ ,  $B_i$  tiene  $\leq 2^i$  elementos. Coste  $\Theta(2^{i+1})$ .
- Calcular la suma de los elementos en el vector  $x$ , coste  $O(\lg n)$

De acuerdo con la propiedad anterior el algoritmo calcula correctamente los valores  $x_i$  pedidos. El coste del bucle es  $\leq \sum_{i=0}^k 2^i = O(2^{k+1}) = O(n)$ . El coste del algoritmo es por lo tanto  $\Theta(n)$ .